



**SAP - ABAP Development User Guide Version | CUSTOMER**  
%NW-ASABAP-LONG-7.55% FPS00 and SAP Cloud Platform ABAP Environment  
Document Version: 3.14 – 2020-11-13

# **SAP - ABAP Development User Guide**

## **Client Version 3.14**

# Content

<b>1</b>	<b>About the ABAP Development User Guide. . . . .</b>	<b>6</b>
<b>2</b>	<b>Cloud Enabling SAP Cloud Platform ABAP Environment for ADT. . . . .</b>	<b>8</b>
<b>3</b>	<b>Getting Started. . . . .</b>	<b>10</b>
3.1	Eclipse Basics. . . . .	10
3.2	Quick Launch. . . . .	15
3.3	Basic Tutorials. . . . .	18
<b>4</b>	<b>Concepts. . . . .</b>	<b>19</b>
4.1	ABAP Development Objects. . . . .	19
	Tools for ABAP Development. . . . .	19
	Status of a Development Object. . . . .	20
	Source Code Unit. . . . .	21
	Modularization with Function Modules. . . . .	22
4.2	Documentation of Development Objects. . . . .	25
	Short Texts. . . . .	26
	Long Texts. . . . .	28
4.3	Element Info. . . . .	33
4.4	ABAP Repository. . . . .	36
	Relation Explorer. . . . .	36
	ABAP Packages. . . . .	38
	ABAP Repository Trees. . . . .	40
	ABAP Search. . . . .	42
	ABAP Type Hierarchy View. . . . .	46
4.5	ABAP Runtime Errors and Short Dumps. . . . .	46
4.6	ABAP Perspectives. . . . .	47
4.7	ABAP Profiling. . . . .	48
4.8	ABAP Projects. . . . .	56
4.9	ABAP Editors. . . . .	57
	ABAP Source Code Editor. . . . .	57
	ABAP Package Editor. . . . .	61
	ABAP Dictionary Editors. . . . .	63
	Messages and Message Classes. . . . .	66
	Transformation Editor. . . . .	69
	RND Parser. . . . .	69
4.10	Activation. . . . .	70

4.11	Unit Testing in ABAP. . . . .	71
4.12	ATC Quality Checking. . . . .	72
4.13	ABAP Debugger. . . . .	73
	Breakpoints in the Debugger - Characteristics. . . . .	75
	Watchpoints. . . . .	76
	Debugging Modes. . . . .	76
4.14	Data Preview. . . . .	79
4.15	Feed Reader View. . . . .	80
4.16	Outline View. . . . .	80
	Advantages of Using the Outline View. . . . .	82
	Icons in the Outline View. . . . .	83
4.17	Quick Assists. . . . .	85
	ABAP Refactorings. . . . .	86
	ABAP Quick Fixes. . . . .	90
	Quick Assist View. . . . .	91
	Quick Assist Overview. . . . .	92
4.18	Software Component. . . . .	99
4.19	Released APIs. . . . .	101
	Add Custom Fields via Key User App (C0). . . . .	101
	Use System-Internally (C1). . . . .	102
	Use as Remote API (C2). . . . .	102
	Deprecation. . . . .	103
4.20	Transport Organizer. . . . .	104
	Transport Layer. . . . .	107
	Transport Request. . . . .	108
	Icons and Decorators in the Transport Organizer. . . . .	110
	Request Editor. . . . .	111
	Task Editor. . . . .	113
4.21	Accessibility Features in ADT. . . . .	113
4.22	Authorization Fields and Objects. . . . .	114
<b>5</b>	<b>Tasks. . . . .</b>	<b>116</b>
5.1	Fundamental Tasks and Tools. . . . .	116
	Working with ABAP Projects. . . . .	117
	Working with ABAP Packages. . . . .	132
	Working with ABAP Repository Trees. . . . .	140
	Working with Development Objects. . . . .	155
	Working with ABAP Source Code Objects. . . . .	184
	Working with Classic Objects in ABAP Dictionary. . . . .	213
	Working with Number Range Objects. . . . .	245
	Working with Business Services. . . . .	248

Documenting Development Objects . . . . .	258
Working with Enhancements . . . . .	262
Working with Transformations for XML . . . . .	270
Working with Messages and Message Classes . . . . .	275
Cloud Working with the HTTP Service Editor . . . . .	286
Working with the Relation Explorer . . . . .	287
Searching in ABAP Projects . . . . .	291
Editing ABAP Source Code . . . . .	294
Applying Quick Assists . . . . .	342
Working with Transport Organizer . . . . .	437
Accessing ABAP Keyword Documentation . . . . .	461
Working with Released APIs . . . . .	462
Working with Data Preview . . . . .	468
Previewing Analytical Queries . . . . .	486
Getting Feeds . . . . .	487
Working with Bookmarks . . . . .	491
Cloud Working with Authorization Objects and Fields . . . . .	493
5.2 Ensuring Quality of ABAP Code . . . . .	498
Unit Testing with ABAP Unit . . . . .	499
Checking Quality of ABAP Code with ATC . . . . .	579
5.3 Using Troubleshooting Tools . . . . .	612
Debugging ABAP Code . . . . .	613
Profiling ABAP Code . . . . .	640
Analyzing SQL Statements . . . . .	692
Cloud Using Dynamic Logpoints . . . . .	694
Analyzing ABAP Runtime Errors . . . . .	710
5.4 Enabling Accessibility Features in ADT . . . . .	716
Setting Configurations and Preferences . . . . .	717
Using Screen Readers . . . . .	720
<b>6 Reference . . . . .</b>	<b>738</b>
6.1 Cloud Released Object Types . . . . .	738
6.2 Cloud Comparison of the ADT Features in the Context of ABAP Environment vs. ABAP Platform . . . . .	743
6.3 Keyboard Shortcuts for ABAP Development . . . . .	747
Edit Actions . . . . .	747
Displaying Actions . . . . .	748
Navigation Actions . . . . .	749
Moving Actions . . . . .	749
Commenting Actions . . . . .	750
Search and Help . . . . .	750
Windows™-based Shortcuts . . . . .	750

6.4	Syntax of ABAP Dictionary Objects.....	751
	Structures.....	751
	Database Tables.....	772
6.5	Syntax for Breakpoint Conditions.....	800
	Variables in Breakpoint Conditions.....	801
	Literals in Breakpoint Conditions.....	802
	Built-in Functions in Breakpoint Conditions.....	803
	Operators in Breakpoint Conditions.....	804
<b>7</b>	<b>Security Guide.....</b>	<b>805</b>
7.1	Resource Protection on Front-End Client.....	806
7.2	Installing Plug-ins from Third Parties.....	807
<b>8</b>	<b>What's New in ABAP Core Development.....</b>	<b>808</b>
8.1	Cloud Version 3.14.....	808
8.2	Cloud Version 3.12.....	814
8.3	Cloud Version 3.10.....	819
8.4	Cloud Version 3.8.....	824
8.5	Cloud Version 3.6.....	824
8.6	Version 3.4.....	829
8.7	Version 3.2.....	837
8.8	Version 3.0.....	842
8.9	Version 2.102.....	848

# 1 About the ABAP Development User Guide

## Scope of Documentation

This documentation describes the functionality and usage of the ABAP Development Tools (ADT) within the ABAP core development scenario. In particular, it focuses on use cases for creating, editing, testing, debugging, and profiling development objects in an 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 the ABAP core development scenario and want to work with an Eclipse-based IDE.

## Validity of Documentation

This documentation belongs to the ABAP Development Tools **client version 3.8** 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 Core Scenario

### Note

Visit also the [ABAP Development](#)  SAP community channels for more information and to interact with other ABAP developers.

## 2 Enabling SAP Cloud Platform ABAP Environment for ADT

SAP Cloud Platform ABAP Environment is part of the SAP Cloud Platform and offered as Platform as a Service (PaaS). ABAP Environment is a special variant of the ABAP platform and supports a subset of the ABAP language version.

### Overview

SAP Cloud Platform ABAP Environment enables customers to develop and expose business logic to build ABAP applications and extensions mainly for SAP S/4HANA Cloud.

ABAP developers can only use ABAP Development Tools (ADT) to develop these kinds of ABAP applications.

### Logon Credential Prerequisites

To provide access for ABAP developers to the relevant space on SAP Cloud Platform Cloud Foundry using ADT, your company has to meet the following general prerequisites:

- **Get a global account** to develop and deploy applications on SAP Cloud Platform.
- **Set up your account model** to get your SAP Cloud Platform global accounts and subaccounts that enable you to access the relevant SAP Cloud products within the production landscape.
- **Configure your environment** to adapt the service instance that represents the ABAP system to your company's needs. For example, you might want to provide user access for the application.
- **Develop and deploy applications** to define and expose business logic using the technical setup, such as the client installations of ADT on the relevant computers or an infrastructure to distribute your applications within the system landscape or between development teams.

#### Note

In case of any questions, contact the SAP Cloud Platform expert or the relevant administrator of your company.

### ADT-Specific Prerequisites

Before you can finally start to create, process, and test development objects using ADT, you have to create an ABAP cloud project. This project plays a mediating role between the ABAP system and the Eclipse-based IDE.

You have the following two possibilities to create an ABAP cloud project:

- [Using a Service Key Provided by the Service Instance \[page 119\]](#) if you want to check the access credentials using a service instance. The service key is then provided by the SAP Cloud Platform Cloud Foundry Environment instance.
- [Using an Existing Service Key \[page 121\]](#) if you already have a service key in JSON format and want to copy and paste or import it directly.

## Related Information

[Getting Started with a Customer Account: Workflow in the ABAP Environment](#)

[SAP Cloud Platform ABAP Environment expert page](#) 

[ABAP Projects \[page 56\]](#)

[Working with ABAP Cloud Projects \[page 117\]](#)

[Eclipse Basics \[page 10\]](#)

[Quick Launch \[page 15\]](#)

# 3 Getting Started

## 3.1 Eclipse Basics

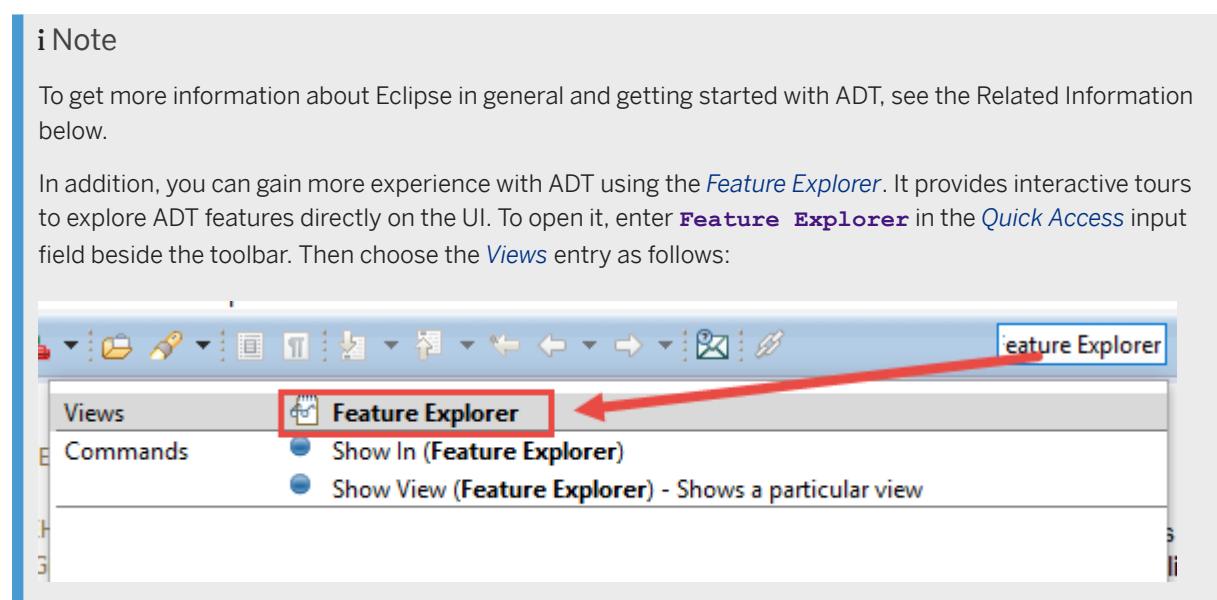
The following information provides you with a selected overview of Eclipse features in order to make your transition working with ABAP Development Tools (ADT) easier.

This section describes general information about Eclipse that is relevant for ABAP developers working with ADT.

### i Note

To get more information about Eclipse in general and getting started with ADT, see the Related Information below.

In addition, you can gain more experience with ADT using the *Feature Explorer*. It provides interactive tours to explore ADT features directly on the UI. To open it, enter **Feature Explorer** in the *Quick Access* input field beside the toolbar. Then choose the *Views* entry as follows:

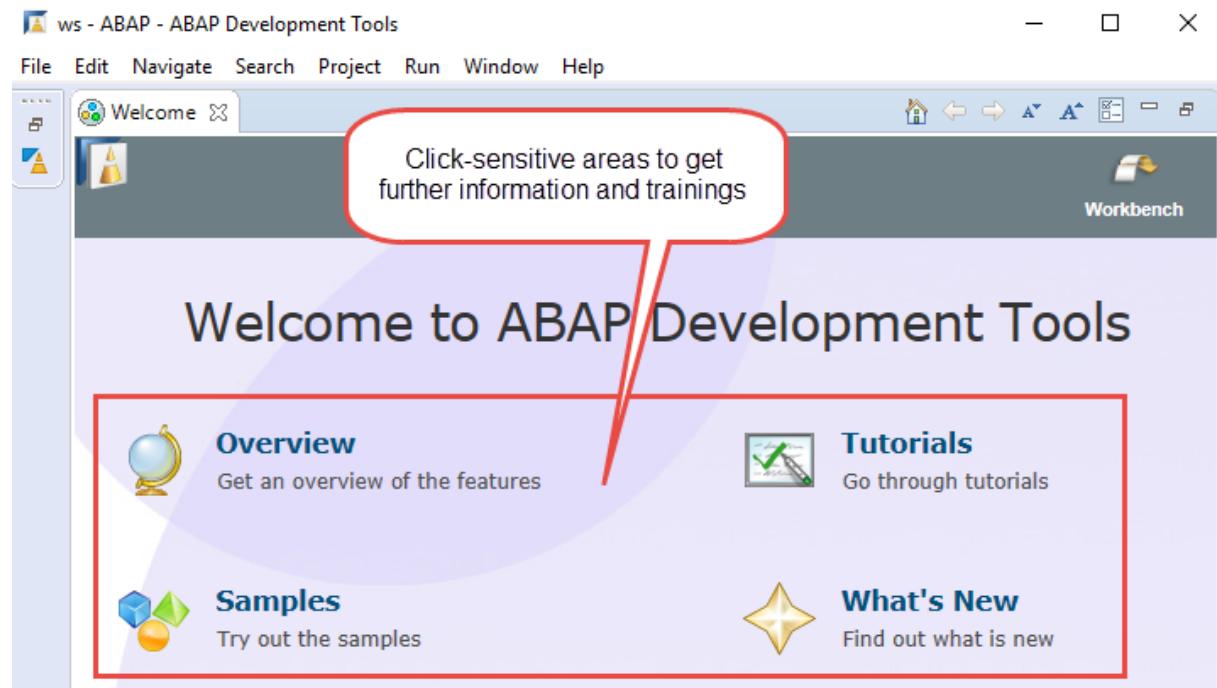


You will find here the following frequently asked questions:

- [What is displayed when I open ADT for the first time? \[page 11\]](#)
- [What does the Eclipse UI display and where can I found what? \[page 11\]](#)
- [How can I browse through the repository content? \[page 12\]](#)
- [Can I open the SAP GUI anyway? \[page 12\]](#)
- [How can I arrange and modify the perspective? \[page 13\]](#)
- [How can I reset the perspective? \[page 13\]](#)
- [Where can I set general properties/preferences? \[page 13\]](#)
- [How can I open a view quickly? \[page 13\]](#)
- [How can I display input help? \[page 14\]](#)

## What is displayed when I open ADT for the first time?

After installing and opening the Eclipse-based IDE for the first time, the *Welcome* page is displayed. From here, you can open a feature overview of the elementary activities in the area of ABAP projects, perform tutorials like cheat sheets, use cases, and code examples, as well as get information about new Eclipse features.

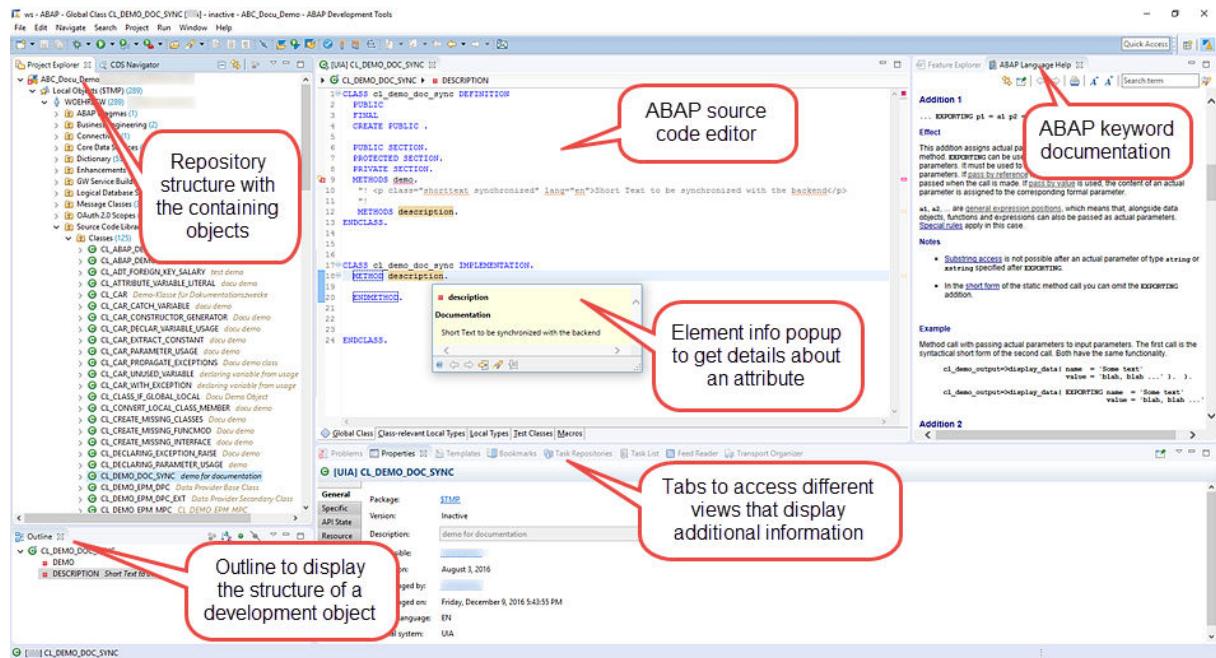


Welcome page that provides further details about working with ADT

## What does the Eclipse UI display and where can I found what?

After creating an ABAP project, you are connected with the back end. You can then open and edit a development object.

Your screen might look like as follows:



Example of your screen when editing an ABAP class with ADT

Each tab can represent

- A view such as the *Project Explorer* to reflect the repository as well as, for example, the *Outline* or *Properties* to display object-specific information
- A source- or form-based editor to display or edit objects

From here you start working.

For more information, see [ABAP Source Code Editor \[page 57\]](#)

## How can I browse through the repository content?

In the *Project Explorer*, the whole repository is represented as ABAP repository trees and folders. From here you can, for example, open or navigate to other development objects. In addition, you can perform functions from the context menu of a development object.

For more information, see

- [ABAP Repository Trees \[page 40\]](#)
- [Working with ABAP Repository Trees \[page 140\]](#)

## Can I open the SAP GUI anyway?

### i Note

SAP GUI is not relevant in the context of SAP Cloud Platform ABAP Environment.

## How can I arrange and modify the perspective?

A perspective is the initial arrangement of views and editors. You can adapt the perspective in order to arrange it to your current needs. To do this, position the mouse cursor on the tab of a view or editor. Drag it to the relevant new position. A green frame will highlight the position where you can drop them.

### → Tip

You can also move a view or editor outside of the Eclipse application, for example, on a second screen.

For more information, see [Perspectives](#)

## How can I reset the perspective?

To get the default perspective back, choose     from the menu bar.

For more information, see [Resetting perspectives](#)

## Where can I set general properties/preferences?

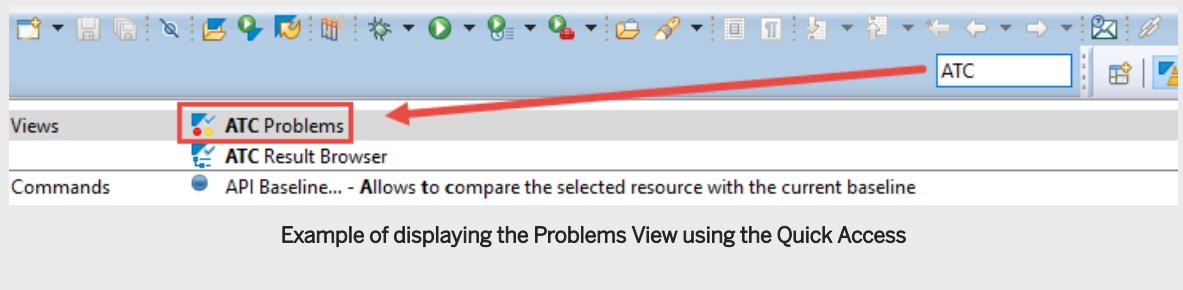
You define the behavior of ADT, for example the ABAP source editors, Project Explorer, ABAP Debugger, and so on, using the   pages.

For more information, see [Setting Configurations and Preferences \[page 717\]](#)

## How can I open a view quickly?

The fastest way to open a view is using the [Quick Access](#) next to the toolbar. Just type in your search string.

### • Example

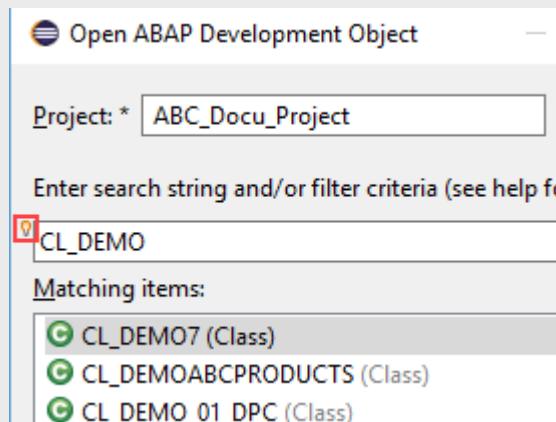


## How can I display input help?

When you have focused an input field that provides input help, the  lightbulb or  "i" decorator is shown. To display the relevant information for the current position in the tooltip, mouse over the relevant decorator.

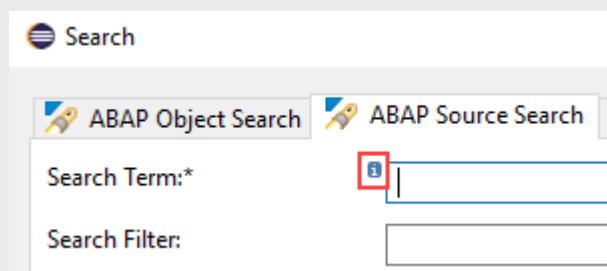
### Example

The lightbulb decorator displays information about input values, such as using asterisks and so on:



Example for the lightbulb decorator when using Ctrl Shift A

The "i" decorator displays information about possible input features, such as using code completion and so on:



Example for the "i" decorator when using the ABAP Source Search

## Related Information

[Quick Launch \[page 15\]](#)

[Frequently Asked Questions \(FAQs\) in the context of SAP Cloud Platform ABAP Environment](#)

[Eclipse documentation - Current Release](#)

[Get Started with the ABAP Development Tools for SAP NetWeaver](#)

## 3.2 Quick Launch

This *Quick Launch* aims to provide you with the compact knowledge you may need when working with ABAP Development Tools (ADT) during the introductory or training period.

### Getting Started...

- >To start, create an ABAP cloud project from the  **New**  menu path. An ABAP project always represents one system connection. It acts as an intermediary between an ABAP system and the front-end IDE client.

#### → Tip

To be able to work in multiple ABAP systems in parallel, you need to create one additional ABAP project.

More on this:

- [Working with ABAP Cloud Projects \[page 117\]](#)
- [Creating an ABAP Cloud Project \[page 118\]](#)

- ▶  **Open ABAP Development Objects**  in your project using **Ctrl** + **Shift** + **A**.

More on this: [Searching Development Objects \[page 291\]](#)

- Browse the contents of a project in the *Project Explorer*. The root of an ABAP project contains a list of ABAP packages that are grouped either under the favorites list (*Favorite Packages*) or the *System Library* node. You can find the development objects simply by expanding the package node.

More on this: [Browsing Development Objects in the Project Explorer \[page 128\]](#)

- Add ABAP packages that are relevant for your work to the list of favorites.

More on this: [Adding a Favorite Package \[page 137\]](#)

- To open a development object, starting from the project tree, double-click the corresponding node in the project.

More on this: [Opening Development Objects \[page 159\]](#)

- Use **Link with Editor**  to synchronize the project structure in the *Project Explorer* with the editor.

Whenever you change an object in the editor, this object will be selected in the expanded *Project Explorer* tree.

- Use the  **Outline**  view to display the structure of a development object.

- Add a new ABAP development object, for example a new class to a package by selecting the package and using the context menu  **New** .

More on this: [Creating Development Objects \[page 156\]](#)

### Working with the Editor

- You can **open multiple development objects** in multiple editor tabs.

- You can search for:
    - ABAP development objects
- More on this:
- [Searching Development Objects \[page 291\]](#)
- **Search for references** to your development object using the **Where-Used** function  (`Ctrl` + `Shift` + `G`).
- More on this: [Searching Usages \(Where-Used\) \[page 293\]](#)
- Open the **Quick Outline** in the source editor currently opened using `Ctrl` + `O`.
- More on this: [Using Quick Views \[page 337\]](#)
- **Navigate** the source code using **F3** or `Ctrl` + `Click`.
  - To navigate between editors, click the back  or forward  arrow key on the toolbar.
  - To switch between the class, its local types, or its ABAP Unit test classes, use the bottom tabs of the class editor.
  - Press **F2** to **display the signature** of a class or method.

## Writing ABAP Source Code

- To **change the source** simply start typing directly in the editor.
- Use **code completion** (`Ctrl` + `Space`) in the editor.
- Insert **source code templates** using code completion and then `Shift` + `Enter`.  
More on this: [Working with Source Code Templates \[page 305\]](#)
- Start the **ABAP Keyword Documentation** with `F1` in the source code editor.  
More on this: [Accessing ABAP Keyword Documentation \[page 461\]](#)
- Format the complete source code (`Shift` + `F1`) or a source code block (`Ctrl` + `Shift` + `F1`) within the editor.  
More on this: [Formatting ABAP Source Code \[page 307\]](#)
- Activate one  or multiple  development objects using the corresponding icons in the toolbar.  
More on this: [Activation \[page 70\]](#)
- Compare source code objects, even across different ABAP systems, using the **Compare with** function in the context menu of the editor or *Project Explorer*.  
More on this: [Comparing Source Code \[page 207\]](#)

## Running ABAP Unit Tests

- Run ABAP Unit tests for one or multiple development objects using the context menu  **Run As**  **ABAP Unit Test** in the editor or *Project Explorer*.  
More on this: [Launching ABAP Unit Tests \[page 551\]](#)
- ABAP Unit **test results** are displayed automatically for you in the **ABAP Unit Runner** view.  
More on this: [Evaluating ABAP Unit Test Results \[page 569\]](#)

## Running ABAP Programs and Test Environments

Run ABAP programs or test environment for classes or function modules using the context menu  [Run As](#)  [ABAP Application](#) in the *Project Explorer* or by pressing **F8**.

## Working with ABAP Runtime Errors (ABAP Short Dumps)

If a runtime error occurs in a program that you have started, the IDE will alert you to an ABAP feed with a small window. However, you can also subscribe to an ABAP feed from the ABAP repository. To subscribe to an additional ABAP feed, press  +  and select [Feed Reader](#).

More on this: [Subscribing to Feeds from ABAP Repository \[page 488\]](#)

## Getting User Assistance Support

- The [Welcome](#) page   provides a standardized set of pages that introduce the ABAP Development Tools to new users.
- To access the complete reference user guide in the context of standard ABAP development, choose   .
- Press **F1** to request context-sensitive help for a given tool or UI component.
-  The [tutorial navigator](#)  provides you access to self-training material in order to learn essential tasks interactively.  
More on this: [Developer Center](#) 

## Enabling Accessibility Features in ADT

To configure Eclipse for using a screen reader, bigger fonts, higher contrasts

More on this: [Accessibility Features in ADT \[page 113\]](#)

## Most Common Keyboard Shortcuts

Development Objects

Shortcut	Description
 +  + 	Open
 +  + 	New

Shortcut	Description
<code>Ctrl + S</code>	Save
<code>Ctrl + F2</code>	Check
<code>Ctrl + F3</code>	Activate

#### Source Code Editing

Shortcut	Description
<code>Shift + F1</code>	Format source code (a.k.a Pretty Printer)
<code>Ctrl + Shift + F1</code>	Format source block
<code>Ctrl + &lt;</code>	Add comment
<code>Ctrl + Shift + &lt;</code>	Remove comment
<code>Ctrl + Space</code>	Code completion

#### Navigation

Shortcut	Description
<code>F3</code>	Open source code
<code>Alt + Left</code>	Backward navigation
<code>Alt + Right</code>	Forward navigation

#### Search and Help

Shortcut	Description
<code>Ctrl + Shift + G</code>	Where-used list
<code>F1</code>	Show context-sensitive help
<code>F2</code>	Show tooltip description

More on this: [Keyboard Shortcuts for ABAP Development \[page 747\]](#)

## 3.3 Basic Tutorials

This topic aims to provide you with the introductory knowledge material that you may need when working with ABAP Development Tools (ADT) the first time.

### Benefit from the tutorials available on the SAP Developer Center

Learn how to use ADT by stepping through the [Tutorial Navigator](#).

# 4 Concepts

## 4.1 ABAP Development Objects

Development objects are the individual parts that are used to build an ABAP application.

The AS ABAP stores the development objects in a repository. For this reason, they are also called repository objects. When the user activates the development objects, the system generates a corresponding runtime version of these objects. The runtime objects are also stored in the repository.

Examples of development objects are:

- ABAP program types such as global classes, interfaces, or function groups
- ABAP Dictionary objects like data elements, domains.
- Objects in the context of Core Data Services like CDS views, behavior definitions.

ABAP packages are used to group development objects into units that belong together semantically. In addition to organizing the development objects, packages also take care of the link-up of the development objects to the software logistics.

In the ABAP Repository, some of the objects have subobjects. Classes are main objects and have methods as subobjects. Function groups are divided in the function modules.

A main development object is uniquely determined by the specification of its name and object type whereas for the specification of a subobject the name and type of the main object is required in addition.

### Related Information

[ABAP Type Hierarchy View \[page 46\]](#)

[Source Code Unit \[page 21\]](#)

[Activation \[page 70\]](#)

[Status of a Development Object \[page 20\]](#)

[ABAP Projects \[page 56\]](#)

[Creating Development Objects \[page 156\]](#)

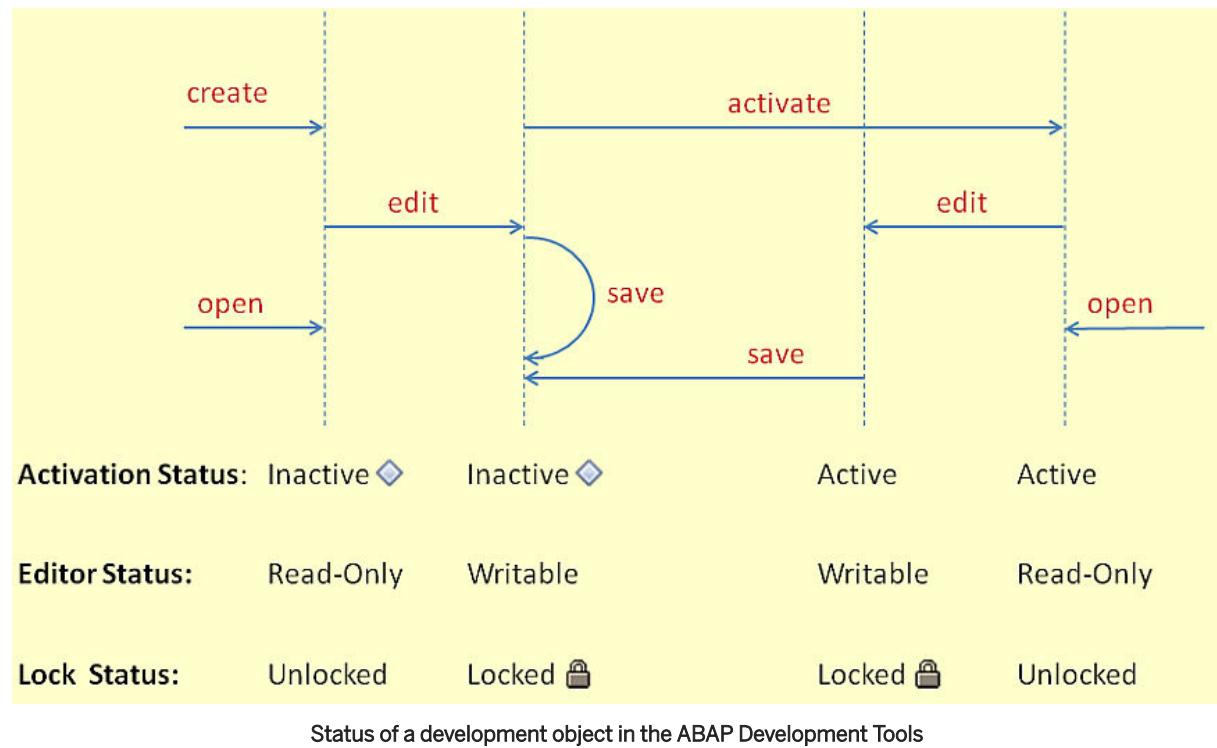
### 4.1.1 Tools for ABAP Development

ABAP Development Tools is the Eclipse-based development environment for ABAP. They are characterized by the same look and feel and the same navigation behavior that is typical for the Eclipse Workbench. Some examples of such native Eclipse tools are different variants of ABAP Source Code editors, editors for ABAP Dictionary objects, CDS editors, the integration in **Outline**, and **Tasks**, **Problems**, **Project Explorer**, **ABAP Unit** views.

## 4.1.2 Status of a Development Object

During the course of being processed within the ABAP Development Tools, a development object takes on various statuses.

The following figure shows the different statuses, depending on the user action involved:



The status of a development object is indicated by a decorator:

Icon/Decorator	Meaning
	Inactive development object in editor or editor tab
	Inactive development object (ABAP class)
	Locked development object (ABAP class)
	Locked and inactive development object (ABAP class)

### Related Information

[ABAP Development Objects \[page 19\]](#)

[Activation \[page 70\]](#)

[Displaying Properties of Development Objects \[page 174\]](#)

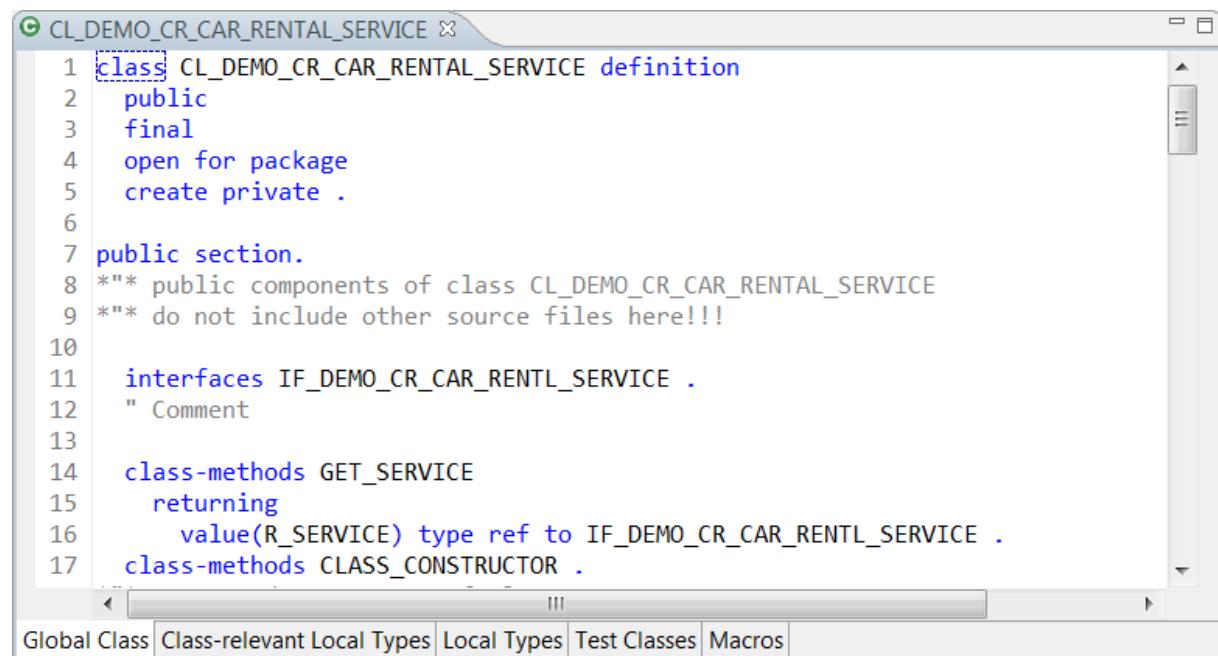
[Switching between Inactive and Active Versions of a Source-based Object \[page 341\]](#)

### 4.1.3 Source Code Unit

A source code unit comprises a coherent source-code part of an ABAP development object that is output to an editor window. Certain functions within the ABAP Development Tools can only be used within one and the same source code unit. For example, changes in variable names only take effect for occurrences within the same source code unit.

The entire source code of a development object can even be divided across several source code units. This is the case, for example, with global ABAP classes where the different source code units are spread across the respective editor tabs. In this way, the definition and implementation part of a class forms a single source code unit and is output under the *Global Class* tab. Local definitions and test classes that belong to the class define their own source codes, respectively.

Another example can be seen in function groups. Each individual include of a function group as well as the function group itself form precisely one source code unit in each case.



```
1 class CL_DEMO_CR_CAR_RENTAL_SERVICE definition
2   public
3   final
4   open for package
5   create private .
6
7   public section.
8   *** public components of class CL_DEMO_CR_CAR_RENTAL_SERVICE
9   *** do not include other source files here!!!
10
11  interfaces IF_DEMO_CR_CAR_RENTL_SERVICE .
12  " Comment
13
14  class-methods GET_SERVICE
15    returning
16      value(R_SERVICE) type ref to IF_DEMO_CR_CAR_RENTL_SERVICE .
17  class-methods CLASS_CONSTRUCTOR .
```

Each individual tab in the class editor represents exactly one source code unit of an ABAP class

### Related Information

[ABAP Development Objects \[page 19\]](#)

[Creating ABAP Classes \[page 185\]](#)

[Comparing Source Code \[page 207\]](#)

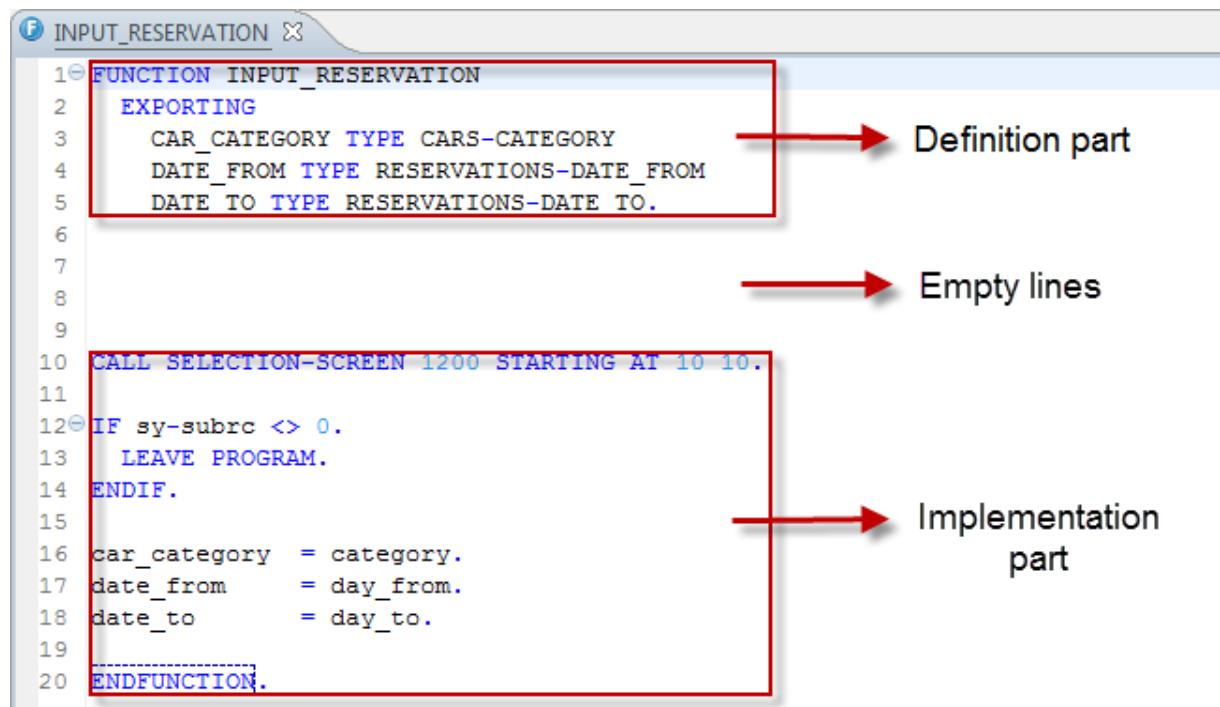
## 4.1.4 Modularization with Function Modules

Function modules allow you to encapsulate and reuse global functions in an ABAP system.

You can create, display, test, and administer the following development objects in the ABAP Development Tool (ADT):

- **ABAP function group**: Container for grouping and defining the interface of function modules with similar or complementary functions
- **ABAP function module**: Procedures with interfaces that allow the reuse of functions within other programs
- **ABAP function group include**: Container for grouping other units – for example, subroutines, PAI, PBO modules, local class declarations, event blocks, etc.

The editor generates a **synthetic view** of a function module. The function module will be displayed as one source code unit. The content set is displayed as the [Definition Part \[page 23\]](#) and [Implementation Part \[page 24\]](#) in the editor, retrieved from the back end.



```
1① FUNCTION INPUT_RESERVATION
2   EXPORTING
3     CAR_CATEGORY TYPE CARS-CATEGORY
4     DATE_FROM TYPE RESERVATIONS-DATE_FROM
5     DATE_TO TYPE RESERVATIONS-DATE_TO.
6
7
8
9
10  CALL SELECTION-SCREEN 1200 STARTING AT 10 10.
11
12② IF sy-subrc <> 0.
13   LEAVE PROGRAM.
14 ENDIF.
15
16 car_category  = category.
17 date_from      = day_from.
18 date_to        = day_to.
19
20 ENDFUNCTION.
```

Definition part

Empty lines

Implementation part

Editor that displays the source code of a function module, divided into definition part and implementation part

### Related Information

[Definition Part \[page 23\]](#)

[Implementation Part \[page 24\]](#)

[Creating an ABAP Function Group \[page 198\]](#)

[Creating an ABAP Function Module \[page 202\]](#)

[Creating an ABAP Function Group Include \[page 199\]](#)

[Adding Include Suffixes \[page 201\]](#)

## 4.1.4.1 Definition Part

The definition part declares the parameters of a function module. It starts with `FUNCTION < func_name >` and ends with a period after the last declaration.

### i Note

You must define a type for each parameter. Otherwise the meaning is not clear.

When you open an existing function module, the system checks if the type of an `IMPORTING`, `EXPORTING`, or `CHANGING` parameter has been declared in the back end. If this is not the case, the editor automatically assigns the addition `TYPE ANY` to the parameter. For table parameters without a type, the editor adds the addition `TYPE STANDARD TABLE`.

For both untyped parameters, the `##ADT_PARAMETER_UNTYPED` pragma is added to the definition.

When you save, the parameter is kept untyped and unchanged in the back end.

## Example

The following example shows a definition part that starts with `FUNCTION` and ends after the last parameter declaration with a period. You can also see how `TYPE ANY` has been added to the exporting parameter `EX_PARAM_3`. The table parameter `TAB_PARAM_2` was not assigned a type and will be changed to `TYPE STANDARD TABLE`.

```
FUNCTION MY_FUNCTION_MODULE
  IMPORTING
    VALUE(IM_PARAM_1) TYPE STRING OPTIONAL
    VALUE(IM_PARAM_2) TYPE I DEFAULT 42
    IM_PARAM_3 TYPE STRING
  EXPORTING
    EX_PARAM_1 TYPE REF TO STRING
    EX_PARAM_2 TYPE ANY
    VALUE(EX_PARAM_3) TYPE ANY
    VALUE(EX_PARAM_4) TYPE ANY
  CHANGING
    CH_PARAM_1 TYPE STRING
    VALUE(CH_PARAM_2) LIKE TADIR
  TABLES
    TAB_PARAM_1 LIKE TAB LINE
    TAB_PARAM_2 TYPE STANDARD TABLE
  EXCEPTIONS
    EXCEPTIONS_1.
```

## Limitations

The ADT source code-based view of a function module is not persisted in the database in the back end. This leads to the following limitations:

- You cannot add or save any comments in the definition part. They will be lost after activation.

- After activation, since the source code is rebuilt from the persisted components in the back end, the pretty printer will convert the definition part into upper case letters.

## Related Information

[Modularization with Function Modules \[page 22\]](#)

[Implementation Part \[page 24\]](#)

[Creating an ABAP Function Group \[page 198\]](#)

[Creating an ABAP Function Module \[page 202\]](#)

[Changing a Definition Part and Implementation Part of a Function Module \[page 204\]](#)

[Creating an ABAP Function Group Include \[page 199\]](#)

[Adding Include Suffixes \[page 201\]](#)

### 4.1.4.2 Implementation Part

The implementation part of a function module contains the functionality of the function module and ends with the statement `ENDFUNCTION..`

When you open a function module in the ABAP source code editor, there can be empty lines between the definition part and the implementation part. The reason for this is that the editor automatically adapts the first implementation line. The position is important for the navigation services, the syntax check, and the position of error markers.

• Example

WW\_INPUT\_RESERVATION

```
1 FUNCTION WW_INPUT_RESERVATION
2   EXPORTING
3     CAR_CATEGORY TYPE WWCARS-CATEGORY
4     WWDATE_FROM TYPE WWRESERVATIONS-DATE_FROM
5     WWDATE_TO TYPE WWRESERVATIONS-DATE_TO.
6   }
7   Empty lines between the definition part and
8   the implementation part
9
10  CALL SELECTION-SCREEN 1200 STARTING AT 10 10.
11
12  IF sy-subrc <> 0.
13    LEAVE PROGRAM.
14  ENDIF.
15
16  car_category = category.
17  wdate_from = day_from.
18  wdate_to = day_to.
19
20 ENDFUNCTION.
```

First line of the implementation part

Display of the first line of the implementation part and the added empty lines in the editor of ADT

## Related Information

[Modularization with Function Modules \[page 22\]](#)

[Definition Part \[page 23\]](#)

[Creating an ABAP Function Group \[page 198\]](#)

[Creating an ABAP Function Module \[page 202\]](#)

[Changing a Definition Part and Implementation Part of a Function Module \[page 204\]](#)

[Creating an ABAP Function Group Include \[page 199\]](#)

[Adding Include Suffixes \[page 201\]](#)

## 4.2 Documentation of Development Objects

Development objects and their elements can be documented with short and long texts.

## Related Information

[Short Texts \[page 26\]](#)

[Long Texts \[page 28\]](#)

### 4.2.1 Short Texts

Short texts are available for the following elements among others:

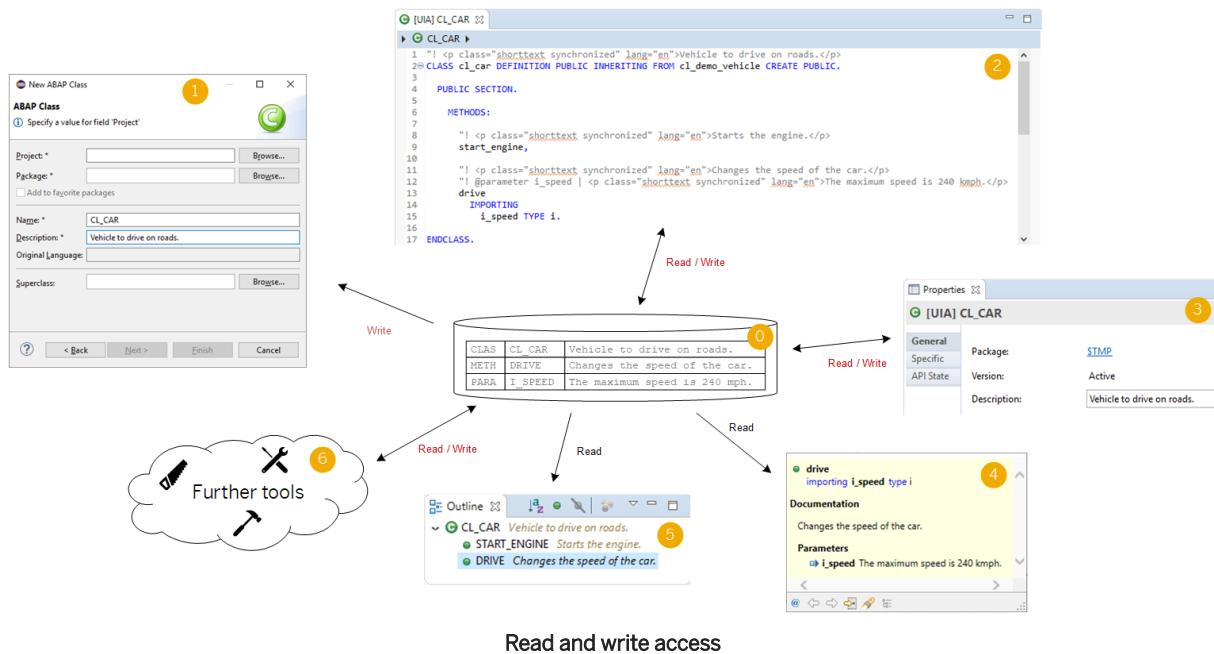
- Function modules
  - Parameters
- Global classes and interfaces
  - Methods
  - Parameters
  - Attributes
  - Types
  - Constants
  - Events

Short texts are stored as metadata in the corresponding function module, class, or interface pool.

These texts are visible in several locations and are sometimes also editable.

- **Read and write**
  - Source code (ABAPdoc)
  - Properties view (only short texts of the main object)
  - Further tools (for example, abapGit)
- **Read-only**
  - Outline view
  - Project Explorer
  - Element information and code completion
- **Write-only**
  - Creation wizards

Whenever a short text is changed, the change is reflected in all locations where the short text is visible.



## 0 Persistence of Short Texts

If you create or change short texts, the changes are saved and synchronized on the back end.

## 1 Creation Wizard

In the creation wizard, you can fill in a *Description* for the object you want to create.

## 2 Source Code

### CDS and DDIC Source-based Objects

If you edit the short texts in the *Editor* of the CDS objects or source-based DDIC objects, the changes are updated automatically in the *Properties* view.

### ABAP Objects

The short texts in ABAP Docs are not visible by default, unless they were created there. Otherwise they have to be generated using quick fixes or explicit synchronization features.

ABAP Doc can also be seen as the source code-based option for writing short texts.

## 3 Properties View

The short texts for main objects can be edited in the *Properties* view if you are logged on in the master language of the object.

## 4 Element Information

*Element Information* displays short texts, but only if no ABAPdoc comment exists. If an ABAPdoc comment exists, then this replaces an existing short text, even if the 'short text synchronized' tag is not used.

However, if the short text is added to ABAPdoc by using the 'short text' tag or the 'short text synchronized' tag, the short text is also shown in the *Element Information*.

## 5 Outline View

The short texts for objects and their parts are displayed in the *Outline* view and in *Project Explorer*.

## 6 Further Tools

Further tools like abapGit that allow you to read or write ABAP short texts.

### Related Information

[Displaying Properties of Development Objects \[page 174\]](#)

[Element Info \[page 33\]](#)

[ABAP Doc Comments \[page 29\]](#)

[Synchronizing ABAP Doc Comments \[page 321\]](#)

[Outline View \[page 80\]](#)

### 4.2.2 Long Texts

Long texts can be created in the following formats:

- [ABAP Doc Comments \[page 29\]](#) to document:
  - Classes
  - Interfaces

- [Knowledge Transfer Document \[page 30\]](#) to document:
  - Behavior Definitions
  - Service Bindings
  - Data Definitions
  - Service Definitions
- SAP Script to document:
  - Classes
  - Interfaces
  - Dictionary Objects (Structures, Domains)

Long Texts are displayed in the [\*Element Info\*](#).

## Related Information

[ABAP Doc Comments \[page 29\]](#)

[Knowledge Transfer Document \[page 30\]](#)

### 4.2.2.1 ABAP Doc Comments

ABAP Doc enables you to document classes, interfaces, and function modules as ABAP Doc comments. These comments consist of one or more comment lines, prefixed by "!".

In the source code editor, they can be placed in an empty line directly in front of a declarative statement. You can document both globally available artifacts (such as classes, interfaces, their methods, attributes, or method parameter) and local artifacts (such as variables and field-symbols).

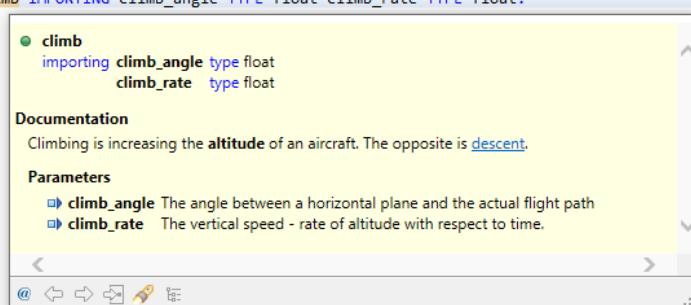
ABAP Doc comments are displayed in

- [\*Element Info\* view](#)
- [\*Element Info\* popup](#)
- code completion list

If for an object, both ABAP Doc Comment and ABAP Short Text exist and they are not synchronized, ABAP Doc Comment is displayed in the [\*Element Info\*](#).

## Example

```
    " ! Climbing is increasing the <strong>altitude</strong> of an aircraft. The opposite is {@link cl_demo_plane.meth:descent }.
    " ! @parameter climb_angle | The angle between a horizontal plane and the actual flight path
    " ! @parameter climb_rate | The vertical speed - rate of altitude with respect to time.
METHODS climb IMPORTING climb_angle TYPE float climb_rate TYPE float.
```



ABAP Doc Comment in the ABAP Editor and in the Element Info

The figure shows ABAP Doc Comments for the method `climb`. If you press **[F2]** on the method name, the information of the method and its documentation is displayed in the *Element Info* pop-up.

The source code editor verifies the position and the content structure of ABAP Doc comments when you execute the **ABAP syntax check**. So, if comments are added at the wrong position or contain incorrect syntax, a warning is displayed in the *Problems* view.

You can apply quick assists to generate ABAP Doc Comments

- 'Add ABAP Doc' for a method without ABAP Doc comments.
- 'Add missing parameters to documentation', if a parameter has been added to the existing method.

## Related Information

[Element Info \[page 33\]](#)

[Editing ABAP Doc Comments \[page 314\]](#)

[ABAP Doc \(ABAP Keyword Documentation\)](#)

[ABAP Doc \(ABAP Keyword Documentation\)](#)

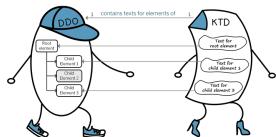
[Short Texts \[page 26\]](#)

[Synchronizing ABAP Doc Comments \[page 321\]](#)

## 4.2.2.2 Knowledge Transfer Document

Knowledge transfer documents contain documentation of development objects and their elements. It is based on markdown language with plain text formatting syntax.

In the knowledge transfer document, you can create documentation for every element of the object individually.



Knowledge Transfer Document

The figure shows an abstract example of a hierarchically structured **documented development object** (DDO) with an already existing **knowledge transfer document** (KTD). Some of the elements are documented, while one element has no documentation (gray).

## Lifecycle of Knowledge Transfer Document and Documented Development Object

### Create

The KTD must be created in the same package as the referenced DDO. The name of the KTD is freely selectable.

If the [package assignment \[page 182\]](#) of the DDO is changed, the package assignment of the KTD is changed as well.



Creating a KTD for a Development Object

### Copy

When copying a DDO, the associated KTD is not copied automatically. It is not possible to copy a KTD.



Copying of the DDO

### Transport

When transporting a DDO, the associated KTD is not transported automatically. You can transport DDO and KTD both together and separately.



Transport of the KTD and DDO

### Translate

KTD texts can be translated. The texts in all languages are stored inside one KTD.



### Translating Texts of the KTD

## Delete

When deleting a DDO, the associated KTD is deleted automatically.



### Deleting of the DDO and KTD

## Related Information

[Creating Knowledge Transfer Documents \[page 258\]](#)

[Editing Knowledge Transfer Documents \[page 259\]](#)

[Writing Texts Using Markdown \[page 32\]](#)

[Linking to Development Objects \[page 261\]](#)

## 4.2.2.2.1 Writing Texts Using Markdown

The following tables give you an overview of the general formatting options and how to format links.

### General Formatting Options

Formatting Elements	Rendered Preview
This text contains <b>bold elements</b>	This text contains <b>bold elements</b>
This text contains <i>italic elements</i>	This text contains <i>italic elements</i>
This text contains `inline code`	This text contains inline code
```	
SELECT * FROM tadir INTO TABLE @DATA(entries).	SELECT * FROM tadir INTO TABLE @DATA(entries).
```	
This is an unordered list of items	This is an unordered list of items
* first item	● first item
* second item	● second item
This is an ordered list of items	This is an ordered list of items
1. first item	1. first item
2. second item	2. second item

## Links

Formatting Elements	Rendered Preview
<www.help.sap.com>	www.help.sap.com
[SAP Help Portal](www.help.sap.com)	SAP Help Portal
<cl_plane>	cl_plane
[start engine](BDEF:DD_PLANE.ENTB:DD_EN-GINE.ACTN:start)	start engine

### **i** Note

To display a character that would otherwise be used to format the text, add a backslash \ in front of the character.

You can link to the following development objects and their elements:

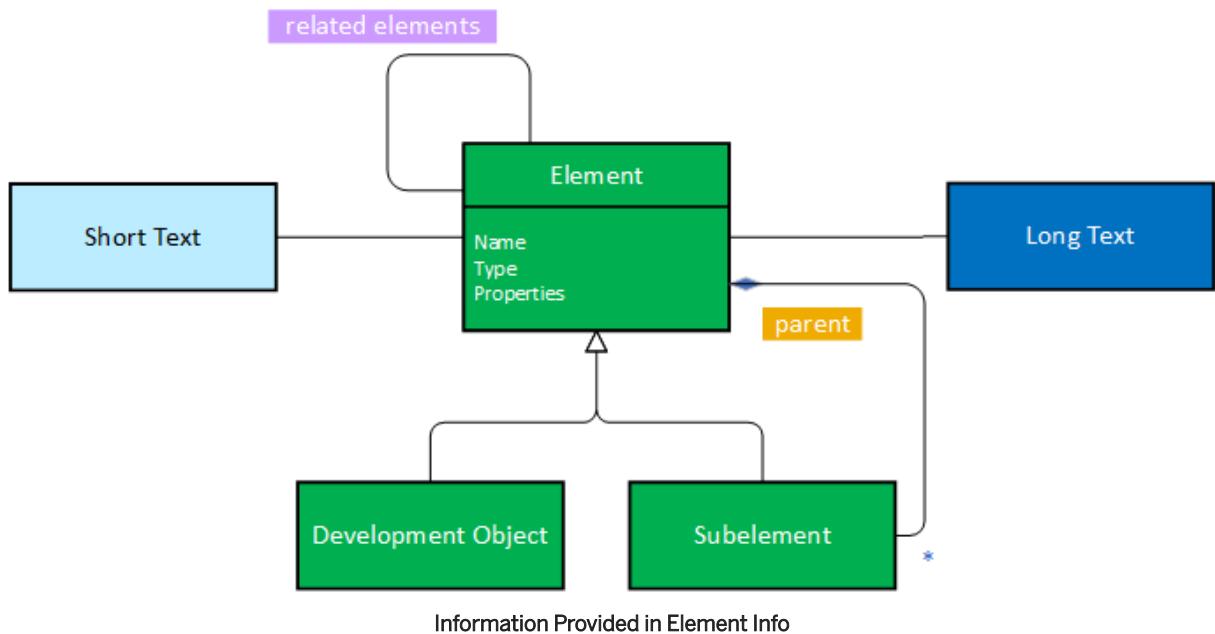
- Classes and interfaces
- Function groups and modules
- Dictionary objects (data elements, domains, database tables, table types)
- Behavior definitions
- CDS views

## Related Information

- [Knowledge Transfer Document \[page 30\]](#)  
[Editing Knowledge Transfer Documents \[page 259\]](#)  
[Linking to Development Objects \[page 261\]](#)

## 4.3 Element Info

An *element* is any development object or any of its subobjects.



A development object is an *element* in itself. Development objects may have subobjects, which are elements as well. These *subelements* may have further subelements. Each element has a name and a type and may have further properties. Each subelement has a *parent element*. Elements may be related to other elements. An element may have a short text and/or a long text.

*Element Info* provides all this information for a selected element. It is available for the following object types and their subobjects.

- ABAP Dictionary (domains, data elements, structures, database tables, table types)
- Core Data Services (CDS views, behavior definitions)
- Source Library (classes, interfaces, function modules)

## Examples

The following examples show how the *Element Info* is displayed.

The screenshot shows the SAP ABAP Element Info for the method `drive`. The method is annotated with the following elements:

- `drive` (highlighted in green) is an element (subobject).
- `importing i_speed type int8` and `acceleration type acceleration_rate` are subelements.
- `acceleration_rate` is a related element.

**Documentation**

Starts the engine. (short text)

A car accelerates at a certain rate and drives with a certain speed. (long text)

At the bottom, there are navigation icons and a toolbar.

Element Info for the Method

The screenshot shows the SAP ABAP Element Info for the database table `planes`. The table is annotated with the following elements:

- `planes` (highlighted in green) is an element (development object).
- `Database table for planes` and `[read more...]` are short text and a link to long text.
- `type_id`, `max_total_range`, `max_takeoff_weight`, `max_seating_capacity`, and `max_wing_span` are related elements.
- `total_range`, `seating_capacity`, and `wing_span` are subelements.

**Column**

Column	Component Type	Data Type	Description
<code>type_id</code>		char(32)	
<code>max_total_range</code>	<code>total_range</code>	int8(19)	Maximum total range of an aircraft
<code>max_takeoff_weight</code>		dec(4,4)	
<code>max_seating_capacity</code>	<code>seating_capacity</code>	int4(10)	Maximum number of seats in a airplane
<code>max_wing_span</code>		int8(19)	Wing span of an aircraft

At the bottom, there are navigation icons and a toolbar.

Element Info for the Database Table

## Related Information

[Displaying Element Info in Source Code Editors \[page 328\]](#)

[ABAP Doc Comments \[page 29\]](#)

[Short Texts \[page 26\]](#)

[Long Texts \[page 28\]](#)

## 4.4 ABAP Repository

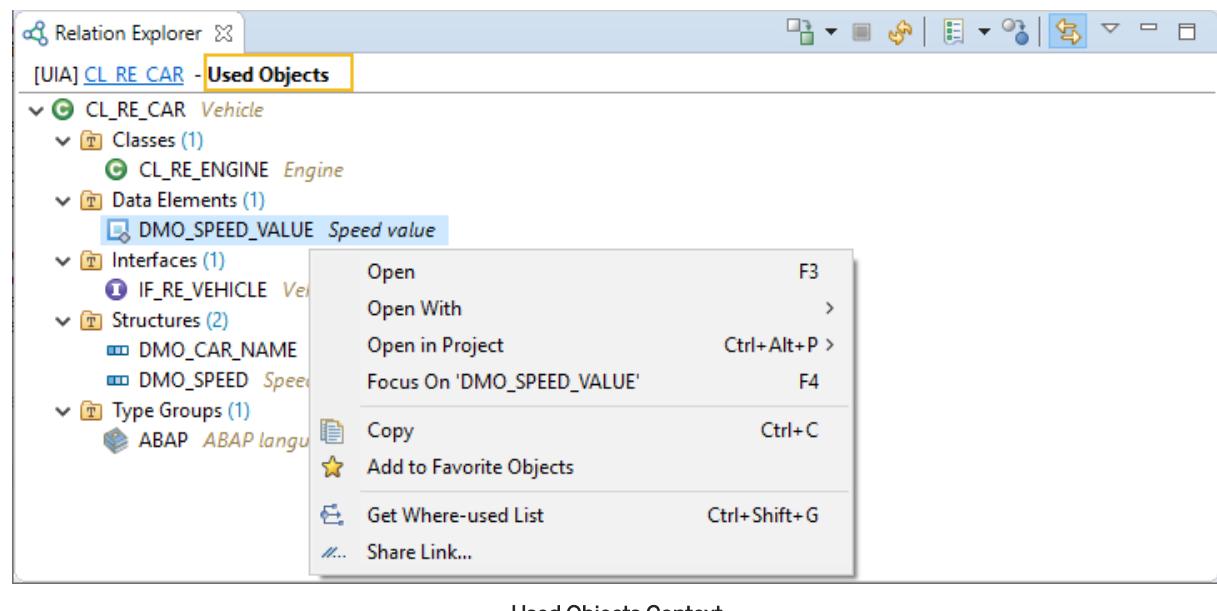
### 4.4.1 Relation Explorer

The *Relation Explorer* helps you to understand how the objects are related to each other.

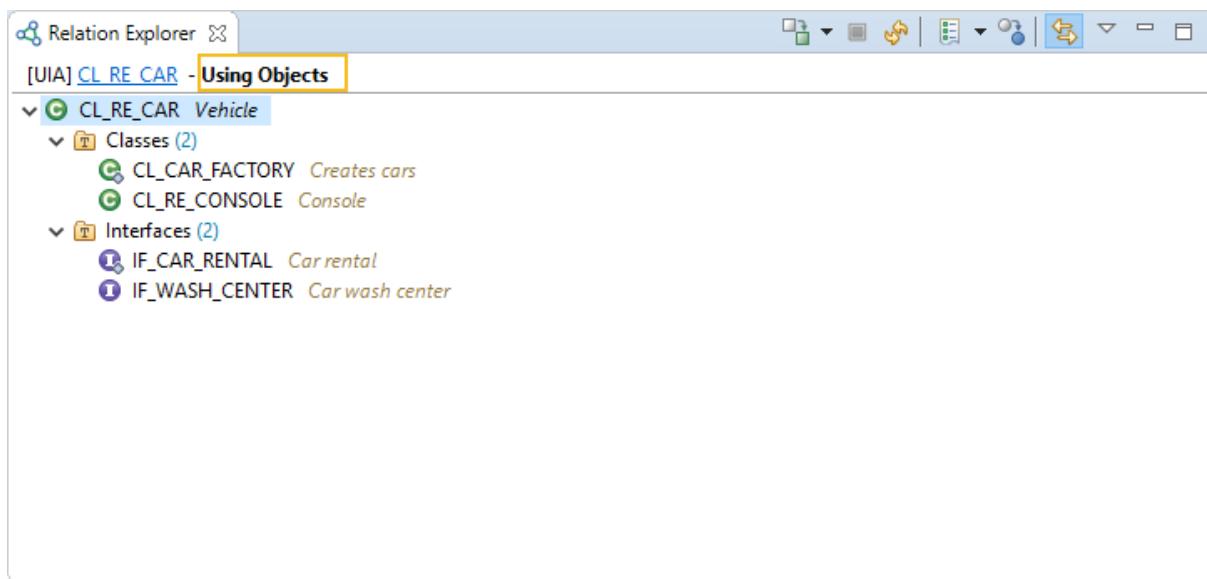
In general, ABAP repository objects are related to each other in many ways. The objects, together with their relations, build networks. For any object, the Relation Explorer determines related objects and displays them as a tree. The object you start with is called **entry object**.

Depending on the kind of relations you are interested in, this might only be a subset of the related objects. By selecting a certain **context**, you can concentrate on one of those subsets.

In the following example, class *CL\_RE\_CAR* is used as an entry object, which is displayed in the context *Used Objects*. As a result, you can see related objects that are used by the entry object.



Additionally, you might be interested in objects that use the entry object. Therefore, switch to the *Using Objects* context.



Using Objects Context

## Contexts

The following contexts are available in the *Relation Explorer*:

Available Contexts for the Objects

Context	Entry Object
Business Object	Data Definition Language Sources
	Behavior Definitions
	Behavior Implementation Classes
	CDS Views
Core Data Services	CDS Views
	Access Controls
Used Objects	Almost all objects with some exceptions
Using Objects	Almost all objects with some exceptions
Extended Objects	Enhancement implementations
Extending Objects	Objects that support Enhancement Framework

## Related Information

[Working with the Relation Explorer \[page 287\]](#)

[Creating a New Instance of the Relation Explorer \[page 290\]](#)

## 4.4.2 ABAP Packages

An ABAP package is a transportable ABAP repository object that groups development objects of an ABAP system.

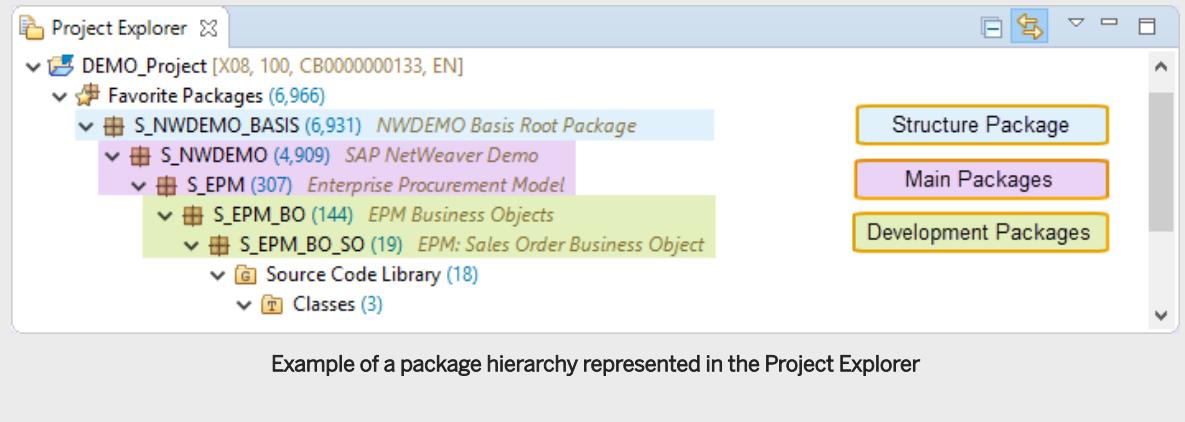
### Overview

In ABAP Development Tools (ADT), packages form the basic modules of ABAP projects. They are displayed as ABAP repository tree in the *Project Explorer*. Each package node can contain subpackages or development objects as subnodes.

The package hierarchy is represented as a node tree. It represents the technical view of the architecture from an application or an SAP system.

#### • Example

The following example displays how the hierarchy of an ABAP package might be represented in the *Project Explorer*:



Use the functionalities of the ABAP repository trees to adopt the representation of a package to your current needs.

Packages which contain no development objects within their hierarchy are highlighted with the empty package  icon in the *Project Explorer*.

### Types

You can work with the following package types:

Type	Description
Structure package	<p>A structure package is the root container of a package hierarchy that defines the basic architecture of an application.</p> <p>It is used to provide the technical prerequisites for decoupling software components.</p> <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> <b>i Note</b>            Structure packages cannot contain any development objects except their own package interfaces and subpackages.         </div>
Main package	<p>A main package defines the root container for one or more development packages.</p> <p>It is used to group functions semantically that share the same system, transport layer, and customer delivery status.</p> <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> <b>i Note</b>            Main packages cannot contain any further repository objects except its own package interfaces and subpackages.         </div>
Development package	<p>A development package groups development objects of the same transport layer in a self-contained unit.</p> <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> <b>i Note</b>            Development packages can contain any number of repository objects, but each one can only be assigned to one development package.         </div>

## Features

Packages provide the following features:

- Creating packages and their subpackages to build up a package hierarchy
- Defining the transport behavior of all development objects from each package
- Adding package nodes to your *Favorite Packages* list within the ABAP repository tree

## Components

An SAP product consists of a delivery unit and product unit which consist of a set of single packages. Such a set of functions forms a software component that represents a view on software logistics. A package or development object is assigned to a software component through its superpackage or when you assign it manually.

SAP's development objects belong to packages that are assigned to an SAP-internal application component.

### i Note

Customers cannot adopt SAP-internal packages.

## Related Information

[ABAP Package Editor \[page 61\]](#)

[ABAP Projects \[page 56\]](#)

[ABAP Repository Trees \[page 40\]](#)

[Adding a Favorite Package \[page 137\]](#)

[Changing the Package Assignment of Development Objects \[page 182\]](#)

[Working with ABAP Packages \[page 132\]](#)

## 4.4.3 ABAP Repository Trees

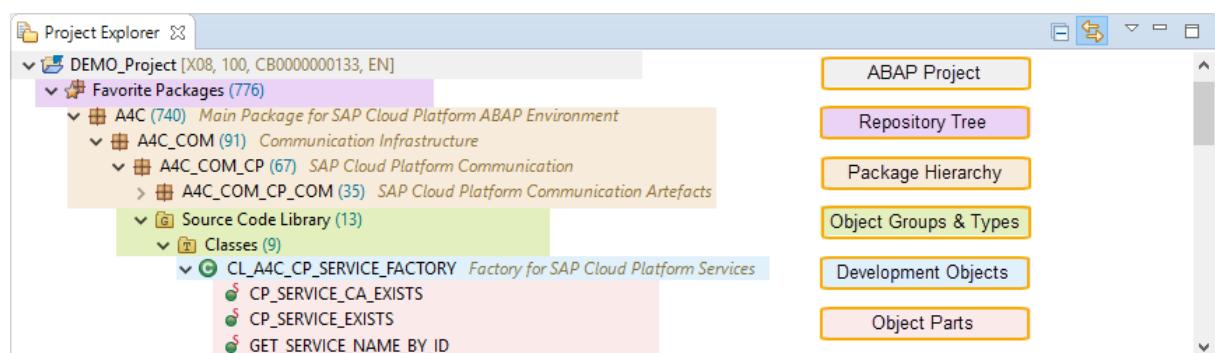
An ABAP repository tree is a container that represents a predefined structure of development objects in the *Project Browser*. It consists of ABAP repository folders. Each folder contains one or more development objects.

### i Note

Your ABAP system needs to be connected with a SAP HANA database.

For simplification, ABAP repository trees and folders are called trees and folders in this documentation.

## Overview



Example of an ABAP repository tree and its ABAP repository folders

By default, ABAP Development Tools (ADT) displays the following trees:

- *Favorite Packages*: Contains a predefined list of ABAP packages – sorted by ABAP package and object type.
- *Favorite Objects*: Contains [favorite development objects \[page 154\]](#) added by user.

-  [Released Objects](#): Contains all released development objects - sorted by ABAP package and object type.

The following information is displayed for each tree and folder:

- Object counters that inform you how many objects are contained in a specific folder
- Short texts for each development object (including ABAP packages)

 **Note**

To hide this information, deselect the [Display object counters](#) and [Display short description of development objects](#) checkboxes on the  [Preferences](#)  [ABAP Development](#)  [Project Explorer](#)  preferences page.

## General Functionalities

You can use existing trees to:

- Navigate to a development object that you want to open or edit.
- Perform mass operations on each folder, for example, ATC checks, ABAP Unit Tests, and so on.
- Link the content of trees and folders with the source code editor. Therefore, [Link with Editor](#)  has to be enabled. Then, the selection automatically navigates to the position where the currently opened development object is located in the tree. In this special case, you can also load the full content if only a limited selection is displayed.
- Visualize the distribution of the containing development objects from a tree in a chart. This enables you to get an impression, for example, in which year most of the development objects have been created, and so on.

You can create, adjust, and delete these trees or even create new ones from scratch to display favorite content of the ABAP repository in the [Project Explorer](#). For this reason, you can define your own tree structure by configuring the so-called tree levels such as the creation year, object types, user who created an development object, and so on.

## Creating ABAP Repository Trees

You have the following possibilities to create trees:

1. For each ABAP project through the configuration wizard from the context menu of the [Project Explorer](#).
2. On base of templates that contain a predefined filter set.
3. By deriving from an existing one, for example, through duplicating the relevant tree.

## Adjusting ABAP Repository Trees

You can adjust the total set of objects to be displayed in the tree through refining the filters to your new needs. You can do this, for example, by adding a package to the [Favorite Packages](#) tree.

You can also change its structuring. You can do this, for example, by adding the *Creation Year* tree level to the *Local Objects (\$TMP)* tree.

You can use the following properties to change the structuring/filtering of a tree and/or folder:

Name	Description
Owner	User who has created or is responsible for a development object
Object Type	Object type of the development objects
Package	ABAP package where the development objects are assigned to
API State	Release state that describes the availability of a development object
Creation Year	Year in which a development object was created
Original Language	Original language in which a development object has been created

## Related Information

[Working with ABAP Repository Trees \[page 140\]](#)

[Released APIs \[page 101\]](#)

## 4.4.4 ABAP Search

ABAP search describes all functions that are provided for performing a search in an ABAP system in all of the back end you work with.

The following functions are provided:

- **ABAP Source Search:** Full text search inside the ABAP source code of the entire system.
- **ABAP Development Object Search:** Search for development objects on the basis of the corresponding name.

### i Note

Alternatively, you can search for a development object using the shortcut `Ctrl Shift A`.

- **Where-Used Function [page 43]:** Find the usage of identifiers like:
  - Development objects that are displayed on the main level of the *Project Browser* like classes, interfaces, database tables, message class, and so on.
  - Sub-objects like members of a class, fields of a table, messages, and so on.

- Other identifiers like local variables, parameters, macros, and so on.

**i Note**

In addition, Eclipse provides search functions that are not ABAP-specific, such as the Task Search, and so on. Use the ABAP search functions to detect ABAP content. The Eclipse file search results may be incomplete.

## Related Information

[Searching Development Objects \[page 291\]](#)

[Searching Usages \(Where-Used\) \[page 293\]](#)

[Opening Development Objects \[page 159\]](#)

### 4.4.4.1 Where-Used Function

The where-used function enables you to determine the development objects that use a specific object, thereby making the dependencies between the development objects clear.

## Related Information

[ABAP Search \[page 42\]](#)

[Searching Usages \(Where-Used\) \[page 293\]](#)

### 4.4.4.1.1 Availability

You can start the where-used function from the following locations:

- Project Explorer
- Outline
- Source code editor
- Another where-used result

You can use the where-used function for:

- Main ABAP development objects, such as function groups or global ABAP classes
- Subobjects, such as attributes, messages, function modules as part of a function group, or class methods as part of a class, and so on

#### 4.4.4.1.2 Execution

You can trigger the where-used function in any of the following ways:

- Choose the icon  *Get Where-Used List...* in the toolbar.
- Use the shortcut `Ctrl+Shift+G`
- In the context menu, choose the entry *Get Where-Used List*.

#### 4.4.4.1.3 Display of Search Results

The findings of a where-used search are displayed in a hierarchical tree in the *Search* view. They represent the relationships between development objects and their elements.

The *Search* view is divided into the following levels:

- Summary of the related development object or element and number of matches
- Display of the related development objects
- Display of the occurrences in a subtree

In addition, on the first level you will also see the number of matches and the objects where a match was found.

##### i Note

The number of related objects is automatically evaluated and displayed. ABAP Development Tools (ADT) displays a question mark as placeholder for the number of matches for those levels you have not selected thus far. The reason for this is that evaluating the number of matches is time-consuming. To evaluate and display this number, select **Expand Tree** from the context menu of the relevant subtree.

#### 4.4.4.1.4 General Functionalities (Toolbar)

From the integrated toolbar in the *Search* view, you can do the following:

- Navigate to the next or previous match
- Expand or collapse a main tree or a subsequent subtree
- Show or hide the ABAP package hierarchy
- Rerun the search execution
- Stop the current search
- Display or clear as well as navigate in the search result history
- Pin the search view
- Display the search result view as list or tree

If you hover a finding, a tooltip is displayed that provides you with further information.

In addition, variables with write or read-only access are highlighted with a colored background.

#### i Note

The color settings can be defined in the **Window > Preferences > General > Editors > Text Editors > Annotations** preferences.

### 4.4.4.1.5 Filter Functionalities (Toolbar)

You can open the filter dialog from the integrated toolbar in the *Search* view to limit the search.

You can limit the search with the following criteria:

#### i Note

Use **Ctrl + Space** to open the Content Assist in order to consider existing entries.

- **Package(s)**: Select the main ABAP package or specific subpackages in which you want to search.
- **Object Type(s)**: Select one or more development object(s), like ABAP classes, ABAP interfaces, and so on, that are relevant for you.
- **Responsible User(s)**: Select one or more users that are involved in your project and might have created or edited the relevant development object(s).
- **Code Category**: Select one of the following categories to define the code area you want to investigate:
  - **Production and Test Code**
  - **Production Code Only**
  - **Test Code Only**
- **Exclude Indirect References**: By default, the where-used result also contains indirect usages. Perform a selection in order to minimize the number of results that relate to components like structures, and so on, or ABAP classes.

#### i Note

When you search for a structure, for example, you will get:

- Direct usages: DATA: speed TYPE zmg\_speed.
- Component usages: speed-value = 50. "value is a field of the searched structure
- Other indirect usages: drive( speed ). "speed is typed with the searched structure

You can exclude the indirect matches by selecting the **Exclude indirect usages** checkbox.

### 4.4.4.1.6 Other Functionalities (Context Menu)

You can run an ABAP unit test or search for another usage from the context menu of a development object or its elements that are displayed in the search results.

## 4.4.5 ABAP Type Hierarchy View

The ABAP Type Hierarchy view displays the inheritance tree of a selected ABAP class or interface.

In the case of classes, it lists the superclass(es) and subclasses of a given class that are located in the relevant ABAP backend system (which is associated with the chosen ABAP project).

### Related Information

[Viewing the ABAP Type Hierarchy \[page 334\]](#)

## 4.5 ABAP Runtime Errors and Short Dumps

If an ABAP program encounters a runtime error (because of an uncaught exception, an exit message, or a failed assertion), the program writes a short dump to document the runtime error and help in its analysis.

### Definition

A short dump provides ...

- well-structured information for localizing and understanding a runtime error, including a textual description of the error and its likely cause
- an excerpt showing the location of the error in the source code
- tables of variables and their values,
- and more.

### Use

If a runtime error occurs, then the ADT alerts you of the runtime error in a small dialog window. From this dialog, you can navigate to the *ABAP Runtime Error* viewer or display the last state of the terminated process in the post-mortem ABAP Debugger.

You can subscribe to an ABAP feed of the runtime errors. In the *Feed Reader* view, two entries for runtime errors are added by default. These are runtime errors caused by you and runtime errors for objects you are responsible for.

You can [add new feed queries and edit feeds \[page 714\]](#) by defining filter conditions for runtime errors.

### i Note

Cloud icon In the context of ABAP Environment, the [Feed Reader](#) is the central view for error analysis of runtime errors.

## Related Information

[Analyzing ABAP Runtime Errors \[page 710\]](#)

[Displaying ABAP Runtime Errors \[page 711\]](#)

[Analyzing ABAP Runtime Errors with Debugger \[page 714\]](#)

[Selecting ABAP Runtime Errors for a Feed \[page 714\]](#)

[Cloud icon Short Dump \(ABAP Keyword Documentation\)](#)

[Cloud icon Runtime Error \(ABAP Keyword Documentation\)](#)

## 4.6 ABAP Perspectives

Just like any other perspectives in Eclipse, the ABAP perspectives define the initial set and layout of tools (views and editors) in the Eclipse window and in this way provide a set of functions aimed at accomplishing a specific type of task. In particular, they work with ABAP development objects that are managed by an ABAP backend system. When using ABAP perspectives, you always have to establish a system connection that is managed by a corresponding ABAP project.

The ABAP development tools contribute the following perspectives to the Eclipse workbench:

### ABAP

This (default) perspective is designed for working with arbitrary ABAP development objects that the user can access by means of ABAP projects.

It consists of an editor area, where the various (ABAP) source code editors are placed, and the following views:

- Project Explorer
- [Outline \[page 80\]](#)
- ABAP Unit
- [Search \[page 293\]](#)
- Problems
- [Relation Explorer \[page 36\]](#)
- Templates

## Debug

This perspective allows you to manage the debugging of ABAP development objects. It consists of the following views:

- Debug
- Breakpoints
- Variables
- ABAP Internal Table

## ABAP Profiling

This is a perspective designed for analyzing ABAP statements and performance with the ABAP trace functionality. It consists of the following views:

- Project Explorer
- ABAP Traces
- ABAP Trace Requests

## Related Information

[ABAP Projects \[page 56\]](#)

## 4.7 ABAP Profiling

ABAP profiling lets you analyze the runtime behavior of an ABAP program.

## ABAP Trace

The ABAP Trace has been integrated into ABAP Developer Tools (ADT).

The ABAP Trace shows you where runtime is being consumed, and where effort for refactoring and optimization can best be applied. The ABAP Trace also lets you analyze and understand program flow, which can be useful when you are trying to understand a problem or learn about code you must analyze or maintain.

## ABAP Cross Trace

### Use

The ABAP Cross Trace ...

- is designed for ABAP developers who build OData Services or other ABAP functionality related to the ABAP RESTful Application Programming Model (RAP).
- provides insights into the RAP runtime framework. This includes the processing of OData requests, for example, in SAP Fiori applications.
- includes functionality which is similar to the payload trace of SAP Gateway trace in SAP GUI. However, it does **not** include a replay functionality for OData requests.

The [ABAP Cross Trace](#) has been integrated into ABAP Developer Tools (ADT).

### Tracing Sensitive Data

To trace potentially sensitive data, you need the authorization object `S_XTRACE` with the appropriate configuration. However, tracing of potentially sensitive data can also be restricted through an authorization configuration or deactivated by the user.

The ABAP Cross Trace differentiates between sensitive data (for example financial data, personal data, or business data) and non-sensitive data. The transferred data might contain sensitive data. Consequently, a user needs specific authorizations to trace or view sensitive data.

### Tracing Scope

Components are used to configure the framework layers to be traced, for example, SAP Gateway. Components can have sub-components for more detailed classification.

A trace record contains information about the component from which it results. A trace record is written for a certain trace level.

#### Note

By default, ABAP Development Tools only displays records with trace level [Essential Information](#). Note that only this level is relevant for SAP Fiori application developers.

## Comparison of ABAP Trace vs. ABAP Cross Trace

The following table compares the characteristics of ABAP Traces and ABAP Cross Traces:

A  
B  
A  
P  
C  
rA  
GB  
sA  
sp  
T  
rr  
aa  
cc  
ee  
-  
C  
a  
m  
tn  
ro  
at  
d  
e  
ch  
ac  
te  
ad  
a  
t  
a  
-

A  
B  
A  
P  
C  
rA  
dB  
sA  
sP  
T  
r  
a  
c  
e  
-  
C  
a  
m  
t  
r  
a  
c  
e  
a  
p  
p  
y  
s  
b  
u  
ic  
lu  
tt  
v  
ib  
t  
he  
tA  
tB  
sA  
RP  
As  
Ro  
pu  
rr  
cc  
g  
rc  
ao  
rd  
re  
-

A  
B  
A  
P  
C  
rA  
dB  
sA  
sP  
T  
rr  
a  
c  
e  
-  
i  
n  
g  
m  
o  
d  
e  
l  
-

A  
B  
A  
P  
C  
rA  
dB  
sA  
sP  
T  
r  
a  
c  
e  
-  
R  
@  
C  
o  
m  
m  
@  
m  
d  
@  
d  
ff  
o  
r  
re  
x  
a  
tc  
d  
ap  
ue  
s  
f  
a  
n  
am  
la  
yn  
s  
ie  
s  
n  
fa  
il  
y

A  
B  
A  
P  
C  
rA  
dB  
sA  
sP  
T  
rr  
a  
c  
e  
-  
s  
ú  
s  
.

i  
N  
o  
t  
e  
N  
o  
t  
r  
e  
c  
o  
m  
m  
e  
n  
d  
e  
d  
f  
o  
r  
e  
x  
a  
c  
t  
p  
e  
r  
f

A  
B  
A  
P  
C  
rA  
dB  
sA  
sP  
T  
rr  
a  
c  
e  
-

o  
r  
m  
a  
n  
c  
e  
a  
n  
a  
l  
i  
y  
s  
i  
s  
.

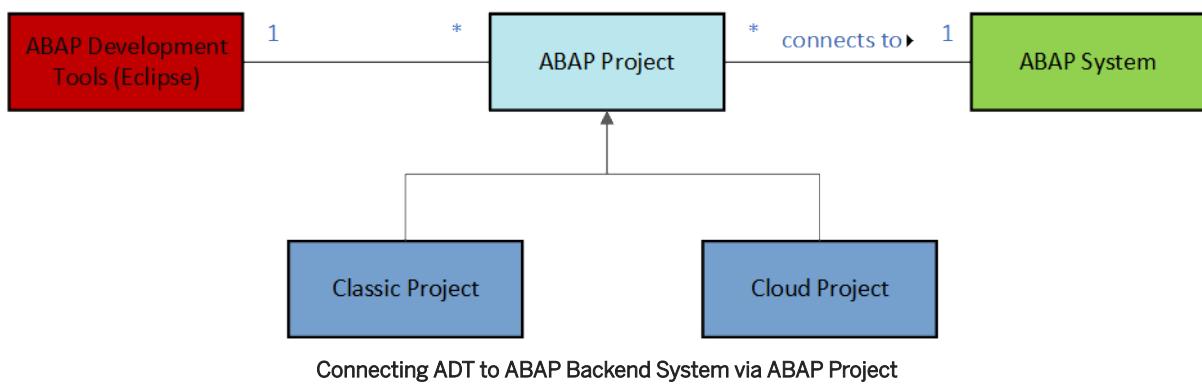
## Related Information

- [Profiling ABAP Code \[page 640\]](#)
- [Starting the ABAP Profiler \[page 640\]](#)
- [Displaying the Trace Overview \[page 649\]](#)
- [Using Analysis Tools \[page 650\]](#)
- [Setting Trace Preferences \[page 678\]](#)
- [Glossary \[page 690\]](#)
- [Working with the ABAP Cross Trace \[page 683\]](#)

## 4.8 ABAP Projects

In ABAP Development Tools (ADT) you use an ABAP Project to connect to a certain backend system. You can work with different backend systems in parallel. This enables you, among others things, to compare development objects from different systems.

You can even have several projects for the same system, for example if you want to work in different clients or log on with different languages.



Connecting ADT to ABAP Backend System via ABAP Project

Basically, there are two kinds of ABAP Projects, *classic projects* and *cloud projects*.

ADT provides different creation wizards to create these kinds of projects. As soon as the connection to a certain backend system is established, ADT provides various editors and many tools to work with the development objects of ABAP repository.

Despite many similarities, the feature sets differ in some aspects depending on the backend system. In cloud projects, for example, the object types are strictly limited, while in typical classic projects much more object types are supported.

### ABAP Projects Comparison

	Classic Project	Cloud Project
Available Object Types	All	Restricted (for example reports cannot be created)
Available ABAP Statements	All	Restricted
Usable SAP Objects	All objects can be used	Only released objects can be used
SAP GUI	Available	Not Available
Project Explorer/Repository Trees	Local Objects System Library Favorite Packages Favorite Objects	Favorite Packages Favorite Objects Released Objects

## Related Information

[ABAP Repository Trees \[page 40\]](#)

[Creating an ABAP Cloud Project \[page 118\]](#)

## 4.9 ABAP Editors

An ABAP editor enables you to create and edit ABAP repository objects using ABAP Development Tools (ADT).

In ADT, the following ABAP editor types are provided:

- [ABAP Source Code Editor \[page 57\]](#) to work with ABAP source code
- [ABAP Package Editor \[page 61\]](#) to work with non-ABAP source-based development objects
- [ABAP Dictionary Editors \[page 63\]](#) to develop or edit source-based or form-based ABAP Dictionary objects
- Other editors, such as:
  - [Messages and Message Classes \[page 66\]](#)
  - [Transformation Editor \[page 69\]](#)

### 4.9.1 ABAP Source Code Editor

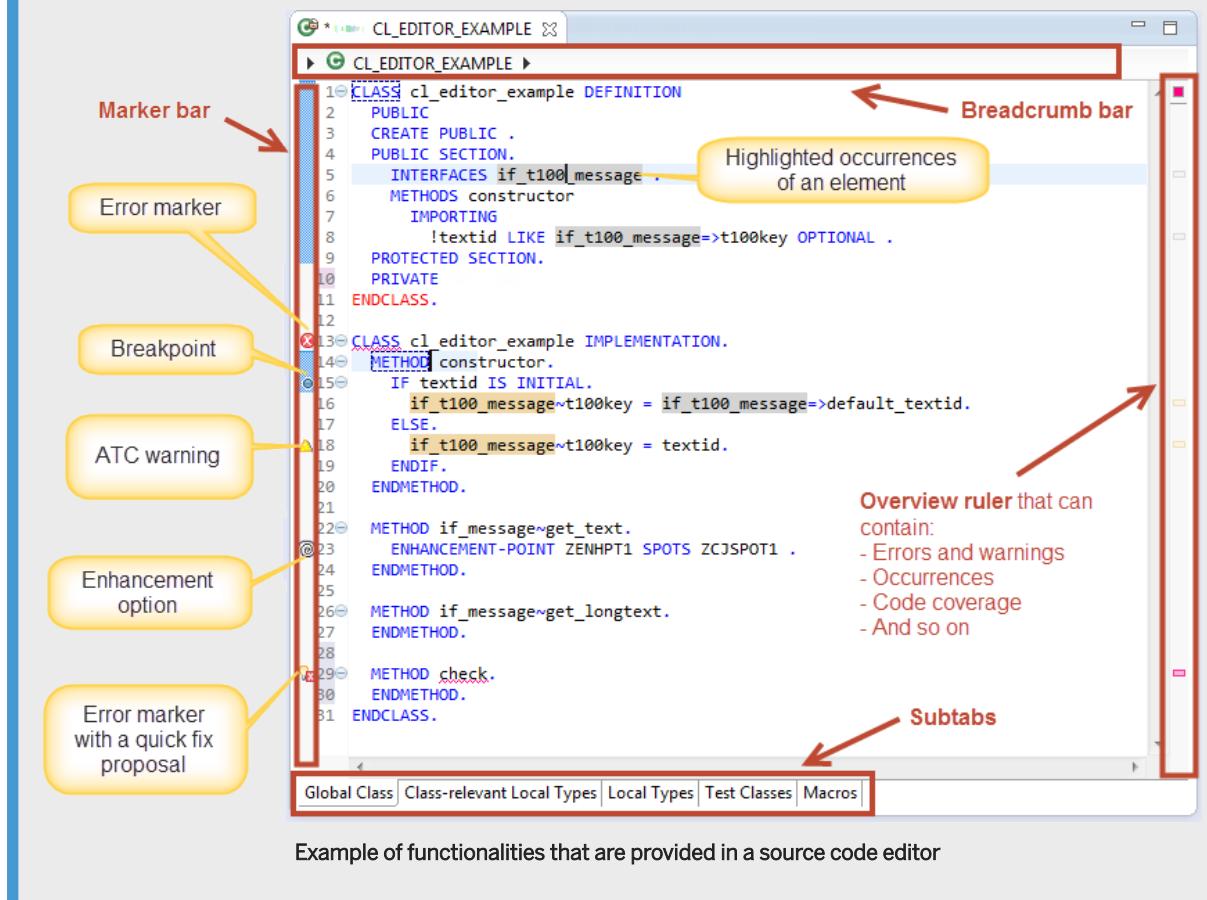
The ABAP source code editor is provided for developing or editing ABAP source code using ABAP Development Tools.

The ABAP source code editor is based on the Eclipse standard text editor and its functionalities have been broadened for programming ABAP source code.

You can open an editor, for example, by double-clicking a development object from the tree in the *Project Explorer* or from a search result list.

## • Example

The following screenshot provides an overview of the UI elements that might be displayed in an ABAP source code editor:



## Related Information

[ABAP Code Templates \[page 59\]](#)

[ABAP Development Objects \[page 19\]](#)

[Editing ABAP Source Code \[page 294\]](#)

[Accessing ABAP Keyword Documentation \[page 461\]](#)

[Comparing Source Code \[page 207\]](#)

[Keyboard Shortcuts for ABAP Development \[page 747\]](#)

## 4.9.1.1 ABAP Code Templates

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

Code templates help the developer to reduce the time spent on routine editing activities and improve consistency when writing code. They allow for quick generation of commonly used ABAP code, as well as easy customization. ABAP code templates go beyond the functionality of simple code snippets. They include direct integration with code completion, the ability to use predefined and custom variables, and the ability to improve style uniformity and validity of ABAP code. When using code snippets, you basically get a shell that you can insert and edit manually.

ABAP Development Tools provides a number of predefined code templates for ABAP. In addition, you can create your own templates or edit existing ones.

You can open and display ABAP code templates in the *Templates* view. To open it, choose  *Window*  *Show View*  *Templates* from the menu.

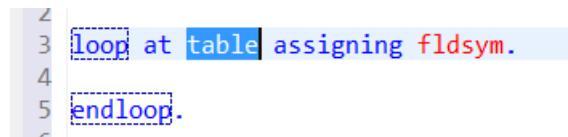
You can view, edit, or create new ABAP code templates using the  *ABAP Development*  *Source Code Editor*  *Templates* preferences page.

Templates are inserted into the ABAP source editor

- Using code assist ( + 

### Example: LOOP AT statement

A common coding pattern in ABAP is to iterate over the rows of an internal table using a loop that reads the internal table in a loop, where each row is assigned to a field symbol. By using a template for this pattern, you can avoid typing in the complete code for the loop. Invoking code assist after typing the word `loop` will present you with a list of possible loop templates. Selecting this template will insert the code into the editor and position your cursor so that you can edit the details.



```
4
3 loop at table assigning fldsym.
4
5 endloop.
6
```

Inserted code in the ABAP Source Editor when using the `loopAtAssign` template

### Related Information

[Working with Source Code Templates \[page 305\]](#)

[Templates and Variables \[page 60\]](#)

## 4.9.1.1.1 Templates and Variables

ABAP code templates can be as simple as a string, but, in general, they contain multiple combinations of string elements and variables. The variables can insert elements in your code such as the current date and time up to the name of the system or the logged on user. However, the most useful feature of variables in templates is the ability to create custom variables. These custom variables act as placeholders in your template for which you provide the values when you insert the template.

```
4
3 loop at table assigning fldsym.
4
5 endloop.
6
```

Definition of the loopAtAssign template

### General Template Variables

ABAP Development Tools come with the following set of variables:

Variable	Description
cursor	Specifies the cursor position after leaving template edit mode.
date	Equates to the current date.
dollar	Equates to the dollar symbol '\$'.
line_selection	Equates to content of all currently selected lines.
time	Equates to the current time.
word_selection	Equates to the content of the current text selection.
year	Equates to the current year.

### ABAP-Specific Template Variables

Variable	Description
enclosing_object	Equates to the development object's name of the current editor, such as name of a class or include.
enclosing_package	Equates to the package, which the development object belongs to.
system_id	Equates to the ABAP System ID.
user	Equates to the name of the user that is currently logged in the ABAP system.

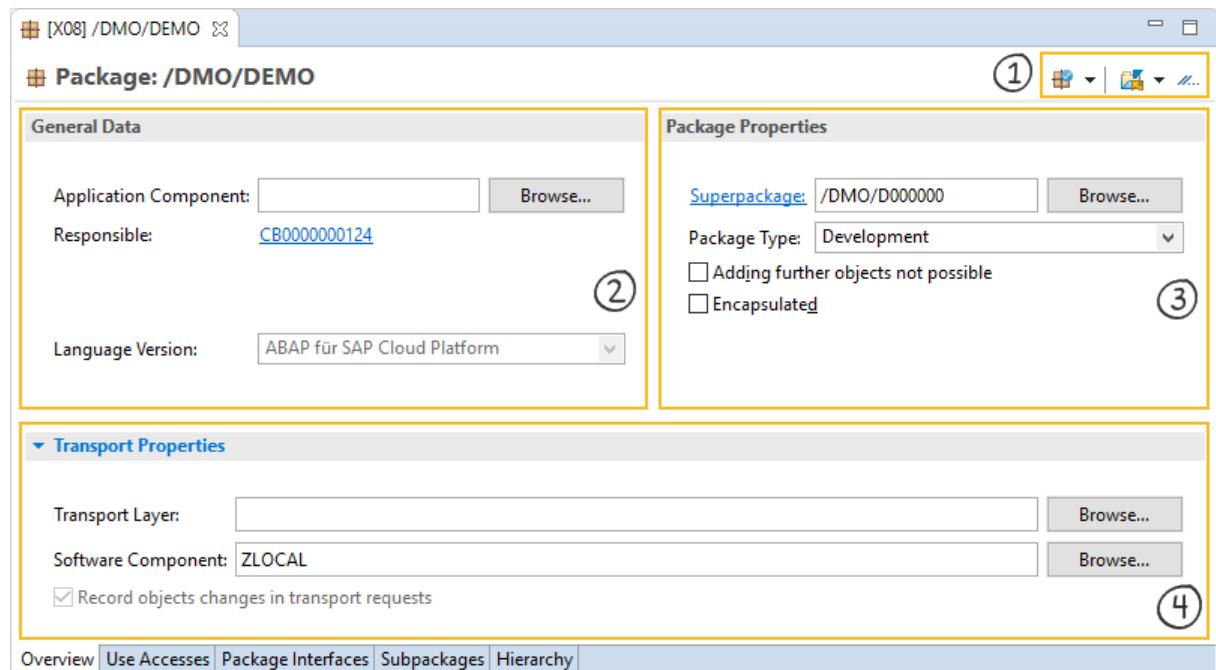
## Related Information

[ABAP Code Templates \[page 59\]](#)

### 4.9.2 ABAP Package Editor

#### Overview Subtab

The editor is a multi-page editor. It displays the following information in the *Overview* editor tab by default:



1. The *Toolbar* contains actions, such as checking the package for all objects, opening packages in another ABAP project, and sharing ADT links.
2. *General Data* displays the application component, the responsible system user, and the language version. As a default, new objects get a language version of the package where they have been created.

#### i Note

To edit the description or to display further general data, open the *Properties* view.

The package breadcrumb and application component are only provided if your ABAP system is connected with an SAP HANA database.

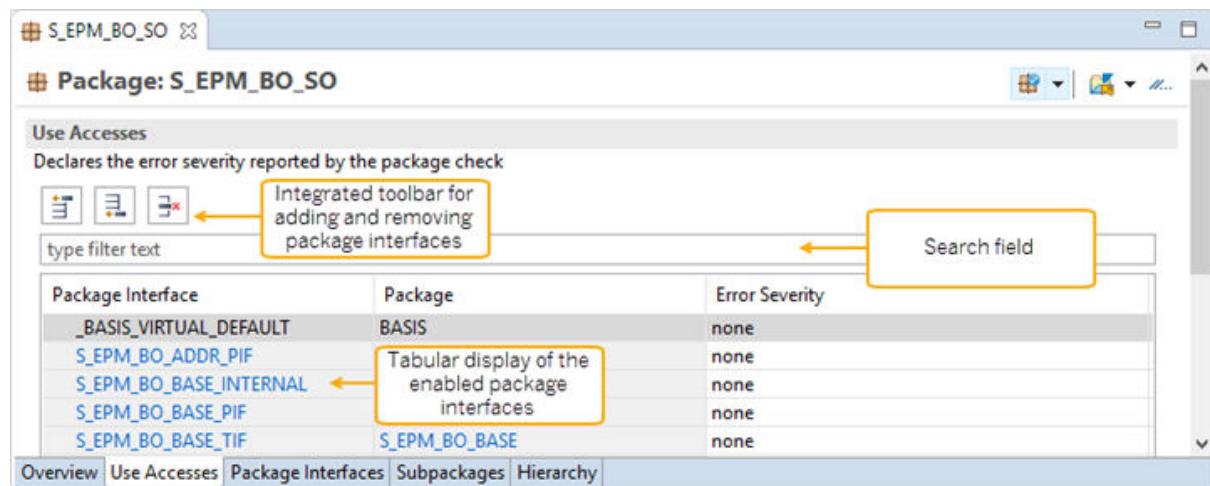
3. *Package Properties* display package-relating information, such as a possible superpackage or package type.
4. *Transport Properties* contains transport-related information, such as transport layer, software component.

## Other Editor Tabs

In addition, you can open the following editor tabs:

### Use Accesses

This editor tab displays the package interface(s) that are used for the selected package, nested within the package hierarchy, or visible in the selected package. Here you will also find details about the severity for each package interface.



The screenshot shows the 'Use Accesses' editor tab for package S\_EPM\_BO\_SO. The tab title is 'Package: S\_EPM\_BO\_SO'. The main area displays a table of package interfaces:

Package Interface	Package	Error Severity
_BASIS_VIRTUAL_DEFAULT	BASIS	none
S_EPM_BO_ADDR_PIF		none
S_EPM_BO_BASE_INTERNAL		none
S_EPM_BO_BASE_PIF		none
S_EPM_BO_BASE_TIF	S_EPM_BO_BASE	none

Annotations highlight the following features:

- An integrated toolbar for adding and removing package interfaces, located above the table.
- A search field on the right side of the table.
- A tabular display of the enabled package interfaces, indicated by the highlighted table row.

Example of an Use Accesses editor tab where the used package interfaces are listed

From the toolbar in the editor tab, you can add package interfaces at the beginning or end of the list. In addition, you can delete entries from the list.

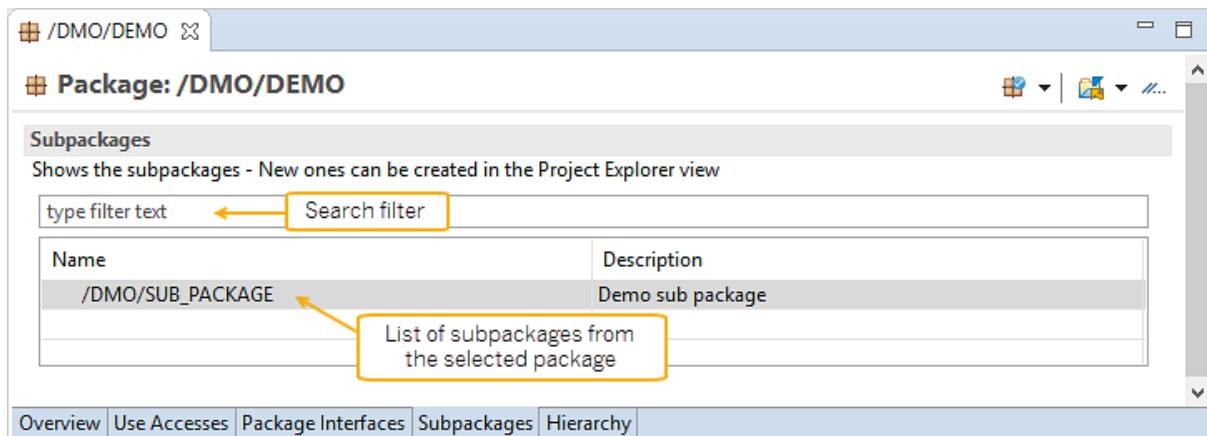
For each package interface, you can define the following severity values for the package check:

- No Response
- Error
- Warning
- Information
- Obsolete

If you use the same package interfaces several times within the package hierarchy, the one with the supremely severity will then be considered.

### Subpackages

This editor tab lists all subpackages that are added beneath the main packages.



Example of an Subpackages editor tab where all subpackages are listed

### i Note

To create a new subpackage, choose [New](#) from the context menu of the relevant node in the *Project Explorer*.

To display the *Package Hierarchy*, check the node structure in the *Project Explorer*.

## Related Information

[ABAP Packages \[page 38\]](#)

[Transport Layer \[page 107\]](#)

## 4.9.3 ABAP Dictionary Editors

The ABAP Dictionary editors are provided for developing or editing classic objects of ABAP Dictionary in ABAP Development Tools (ADT).

The following table provides you an overview of the objects, which can be edited in ADT:

ABAP Dictionary Objects	Availability
Data Elements	✓
Domains	✓
Structures	✓

ABAP Dictionary Objects	Availability
Table Types	✓
Database Tables	✓
Views	✗
Lock Objects	✓
Search Helps	✗
Type Groups	✓

## Source-based editors

The following source-based editors are provided in ADT:

- [Editor for structures \[page 228\]](#)
- [Editor for database tables \[page 232\]](#)

The programming language of the source code is similar to the DDL of ABAP CDS, where metadata is defined using annotations and technical properties (for example, fields or components) are defined in SQL. The source code is generated when it is read from the database. When you save or activate a structure or a database table, the source code is converted to the existing persistence. The source code represents the metadata of the database table, but it is not a transport object itself.

Underneath the editor saves the definition in ABAP Dictionary. This means that any individual formatting of the source code is overwritten after saving.

The  enhancement marker in the ruler before the `define` statement informs you that customizing includes or append structures are included. Hover the marker, to open a popup that displays the enhancements. To navigate to an enhancement, select the linked object name in the popup.

Database tables are persisted as metadata and represented as source code in the source-based editor. A database table consists of its name and fields. One or more fields build the table key, which is mandatory. [Database Table Annotations \[page 774\]](#) are used to provide additional information for the entire table and individual fields.

### i Note

To edit technical settings (data class, size category, buffering, and storage type) or to create/edit indexes, open the integrated SAP GUI.

## Form-based editors

The following form-based editors are provided in ADT:

- [Editor for data elements \[page 220\]](#)
- [Editor for domains \[page 216\]](#)
- [Editor for lock objects \[page 242\]](#)

## Other Classic ABAP Dictionary Objects

Currently, the remaining classic objects in ABAP Dictionary, such as database views, and so on, can be opened and edited in the integrated SAP GUI.

### Related Information

[ABAP Source Code Editor \[page 57\]](#)

[Syntax of ABAP Dictionary Objects \[page 751\]](#)

[Working with Classic Objects in ABAP Dictionary \[page 213\]](#)

[Working with Lock Objects \[page 240\]](#)

### 4.9.3.1 Code Templates for ABAP Dictionary Objects

Code templates help you to reduce the time spent on routine editing activities and improve consistency when writing code.

### Structures and Tables

As in the ABAP source code editor, you can add code snippets with specific syntax like a development object, annotation, or component at the current cursor position in the ABAP Dictionary editor. After adding, you need to adapt the placeholders with variables, elements, lengths, decimals, and so on.

You can add code templates, for example, through code completion (`Ctrl` + `Space`).

For structures, the following template types are provided:

- `currencyKey`: Currency and its Currency Key
- `quantityUnit`: Quantity and its unit
- `searchHelp`: Search help attachment
- `foreignKey`: Foreign key
- `appendStructure`: Append structure

- And so on

#### **i Note**

For further details about the variables used in the corresponding code templates, see Related Information.

## **Related Information**

[Templates and Variables \[page 60\]](#)

[ABAP Code Templates \[page 59\]](#)

## **4.9.4 Messages and Message Classes**

You can create and group messages in message classes in order to inform the user about an error or a status, or to issue a warning.

### **Messages**

Messages notify a user about unexpected behavior, an error, a status, or a result of an action. An application will display the message at runtime – for example, in a dialog box or in the status bar.

A message is specified by a unique 3-digit message number, a single-digit language key, text information, and a message class.

Messages are stored in the database table **T100**.

### **Message Classes**

Message classes are created within an ABAP project at the ABAP package level. They are used to group messages within a development object. So, you can combine related messages and reuse them in other programs. After creating a message class, you add the individual message(s).

#### **Example**

The message with the number 045 contains the placeholder for a parameter. The placeholder "&" stands for the value of the parameter "carrid". In ABAP programs, messages are called using the MESSAGE statement.

Locked	Number	Short Text	Self Explanatory	Last Changed By	Changed On
	003	Choose an example program	<input type="checkbox"/>		2019-07-17
	045	No authorization for airline carrier &	<input type="checkbox"/>		2019-07-17
	047	No flight found for this date	<input checked="" type="checkbox"/>		2019-07-17
<Enter new value>					

Example of a message class where you can add messages

This example of MESSAGE statement MESSAGE i045 (SABAP\_DOCU) WITH `carrid` contains the following elements:

- Message type: "i"
- Message number: "045"
- Message class: "SABAP\_DOCU"
- Short text: "No authorization for airline carrier &"
- Placeholder "&" which is defined for the parameter "carrid"

## Related Information

[Message Types \[page 67\]](#)

[Creating a Message Class \[page 275\]](#)

[Adding a Message to a Message Class \[page 277\]](#)

[Filtering Messages in Message Classes \[page 282\]](#)

[Opening a Message from the Source Code in the Message Class Editor \[page 283\]](#)

[Opening in Another System \[page 163\]](#)

[Sharing Links of Selected Messages \[page 284\]](#)

[Searching the Usage \(Where-used\) of a Message \[page 286\]](#)

### 4.9.4.1 Message Types

A message type determines how program execution is handled after the message has been raised.

The following message types are available:

Message Type	Message	Behavior	Message will be displayed in / on:
I	Information	Once the user has confirmed the message, the program continues immediately after the MESSAGE statement.	Dialog box
S	Status	The program continues normally after the MESSAGE statement.	Status line of the next screen
W	Warning	Depending on the program context, an error dialog box appears and the program stops or terminates. Here the user has to take action.	Status line
E	Error	Depending on the program context, an error dialog box appears and the program stops or terminates. Here the user has to take action.	Status line or dialog box
A	Termination	The program terminates when the user has confirmed the message. Control returns to the next highest area menu.	Dialog box
X	Exit	The program will terminate with a short dump if a runtime error occurs. In this case no message will be displayed. NOTE: Message type X allows you to force program termination. The short dump contains the message ID.	-

## Related Information

[Creating a Message Class \[page 275\]](#)

[Adding a Message to a Message Class \[page 277\]](#)

[Adding a Long Text to a Message \[page 278\]](#)

[Filtering Messages in Message Classes \[page 282\]](#)

[Previewing a Message Long Text \[page 280\]](#)

[Opening a Message from the Source Code in the Message Class Editor \[page 283\]](#)

[Opening in Another System \[page 163\]](#)

[Sharing Links of Selected Messages \[page 284\]](#)

[Searching the Usage \(Where-used\) of a Message \[page 286\]](#)

## 4.9.5 Transformation Editor

The ABAP kernel contains processors for transformations for XML.

The development objects created there are called simple transformations (ST) and XSL transformations. They are stored in the Repository, and linked to the Transport Organizer. ST is an SAP programming language used for describing transformations between ABAP data and XML formats. XSL transformations transform XML to XML.

The *Transformation Editor* is a tool for defining transformations for XML that are later executed on the application server. They are displayed and edited in XML format.

You can use the *Transformation Editor* to:

- Create transformation programs for defining transformation rules
- Edit source code, for example, to map structures or data elements onto XML documents
- Check and activate transformation programs
- Test transformations
- Create breakpoints for debugging simple transformations

### Related Information

[Working with Transformations for XML \[page 270\]](#)

[Transformation for XML \(ABAP Keyword Documentation\)](#)

## 4.9.6 RND Parser

Recursive Non-Descent (RND) parsing is used for obtaining a descriptively specified parser for the programming language ABAP.

In the context of ABAP source code development, the RND parser is used for syntax coloring and code completion. To this end, the definition of the ABAP language is loaded from the ABAP back end.

To define the RND parser, open the context menu from an ABAP project and choose *Properties*. From the *Properties* page, navigate to the  *ABAP Development*  *Editors*  *Source Code Editors*  *RND Parser*  properties page.

From here, you can set the following options:

Option	Description
<a href="#">Open folder</a>	This button is used by kernel developers who want to replace the existing .pad file directly. After you have restarted the client, the syntax coloring of all opened editors is performed.  <b>i Note</b> A restart itself is not required.
<a href="#">Reload from backend</a>	This button is used whenever new ABAP language features are available, such as ABAP keywords and so on, or if the syntax coloring has changed. In this case, you need to reload the configuration from the back end.
<a href="#">Reload from client</a>	This button is only used in the context of kernel development.

## 4.10 Activation

A development object (repository object) can exist as an **inactive** and an **active** version. When you create a new development object, it is always available first as an inactive version.

In its inactive form, the development object is saved as a database object and is thus part of the ABAP Repository of a system. With the activation process you create an active version of a development object, thus also a corresponding runtime object, from an existing inactive version. Successful activation requires that the development object in question is free of syntax errors. Hence, when you trigger an activation process, a syntax check is performed for the entire development object before a runtime object is generated. An active version of development object is therefore always used for **generating a runtime object**.

### Advantages of the Activation Concept

The inactive versions of development objects provide the developers with a separate local view of the repository and they form the basis for a "**local runtime system**". Changes to development objects can be tested within this local system without disturbing the wider development environment.

The main advantage of this is that the **development process becomes seamless**. For example, it makes it possible for you to change the interface of a function module without the changes immediately becoming visible in programs that call it. The changes are only visible system-wide once the development object has been activated.

### Activation Process in ABAP Development Tools

When working with ABAP projects, you can apply the activation to:

- An individual development object of a project
- A multiple number of development objects of a project.

All inactive objects that belong to an ABAP project are listed in the *Worklist of Inactive Objects*. This worklist allows you to create a selection of objects you wish to activate in one single step.

The activation status of development objects affects the following operations and activities:

Operation	Description
Create	Creates an inactive version that is stored in the database of the ABAP system (ABAP Repository). Adds the development object to the worklist of inactive objects of your ABAP project.
Save	Saves the development object as an inactive version without a syntax check.
Activate	Creates an active version from the existing inactive version. The active version is used to generate the runtime version of the development object.
Delete	Deletes active and inactive versions.
Syntax check	Checks the current editor content.
Run / Execute	Executes the runtime object (active version). A runtime object can only be generated from a syntactically correct active version.
Transport	Releasing of transport requests is only used for active objects. You cannot release a transport request until all objects included have been activated.

## Related Information

[Activating Development Objects \[page 178\]](#)

## 4.11 Unit Testing in ABAP

ABAP Unit is a unit-testing framework that is integrated into the ABAP language.

In unit testing, a developer ensures that the correct behavior of the smallest testable units – such as the methods of classes – of his or her code is verifiable. Unit testing makes it easier to verify quality, to refactor code, to perform regression testing, and to write tests according to the test-driven development model.

The most important features for writing ABAP unit tests are the following:

- ABAP Unit tests are written in ABAP. You do not have to learn an additional scripting language for testing.
- You write tests with the standard ABAP Development Tools (ADT). You do not have to use additional tools for developing tests.

- ABAP Unit tests are transported with the ABAP repository objects that they test. They are therefore available in all of the systems of your development and testing landscape.

The most important features of ABAP Unit for running and evaluating unit tests are these:

- You can run ABAP Unit tests as you develop code. You can launch tests, for example, directly from the ABAP editor in the ADT.
- Code coverage measurement is integrated into ABAP Unit testing, so that you can verify the thoroughness of your unit testing and easily find untested code.
- ABAP Unit testing can be automated and is part of mass quality testing.
- Test results are displayed in the *ABAP Unit* view for easy evaluation and analysis.
- ABAP Unit test methods have no parameters. No special knowledge or test framework is required to run ABAP tests; they can be run by anyone, and not just by the developer.
- ABAP Unit tests cannot be executed in productively used ABAP systems.

## Related Information

[Writing ABAP Unit Tests \[page 501\]](#)

[Launching ABAP Unit Tests \[page 551\]](#)

[Evaluating ABAP Unit Test Results \[page 569\]](#)

[Checking Quality of ABAP Code with ATC \[page 579\]](#)

## 4.12 ATC Quality Checking

The ABAP Test Cockpit (ATC) is the main ABAP tool for quality assurance.

Using the ATC, you can check your ABAP development objects for many types of problems, including syntax errors, potential problems in performance, sub-optimal or potentially faulty programming, adherence to standards, errors in ABAP Unit testing, and many others.

Here are the main features of the ATC for developers working in the ABAP Development Tools (ADT):

- Local quality check of your ABAP development objects directly in your development environment, starting from *Project Explorer* or the ABAP source editor
- Display of ATC findings, finding-specific details, and help directly in the *ATC Problems* view
- Notification of high-priority ATC findings from central ATC quality checks through an ABAP feed
- Display of complete central ATC results originating from mass regression check runs in your quality system in a specific *ATC Result Browser* view
- Tool integration for handling exemptions for ATC findings.

## More Details on the ATC

The ATC reports problems as *findings*, messages that describe the problem. Findings have a priority, whereby errors and warnings usually indicate a serious problem that needs to be corrected quickly. All findings offer

context-sensitive help, which includes details of each finding that may not appear in the finding messages themselves.

In ADT, you can use the ATC to check your development objects as you work, directly from the *Project Explorer* or the ABAP source editor. In this case, the findings are for your own use.

Your quality manager can use the ATC to run central "official" quality checks, usually in your integration or consolidation system. You can see your errors and warnings from the active central ATC runs with an ABAP feed.

You usually need to clear central ATC findings by correcting your objects with ADT in your development system, and then transporting the changes to the integration and consolidation systems, or wherever central ATC testing takes place. Should you not be able to clear an "official" ATC finding, you can request an ATC *exemption* that has to be approved by the quality manager. An exemption suppresses an ATC finding either temporarily or permanently.

The *ATC Result Browser* view is provided for requesting exemptions and for working with the complete set of ATC findings in central ATC runs. Quality managers can set up the ATC so that findings from central runs are replicated to development systems. The ATC can also be set up to let you request central exemptions from the *ATC Result Browser* in your development system.

## Related Information

[Checking Quality of ABAP Code with ATC \[page 579\]](#)

[ABAP Testing and Analysis Community](#)

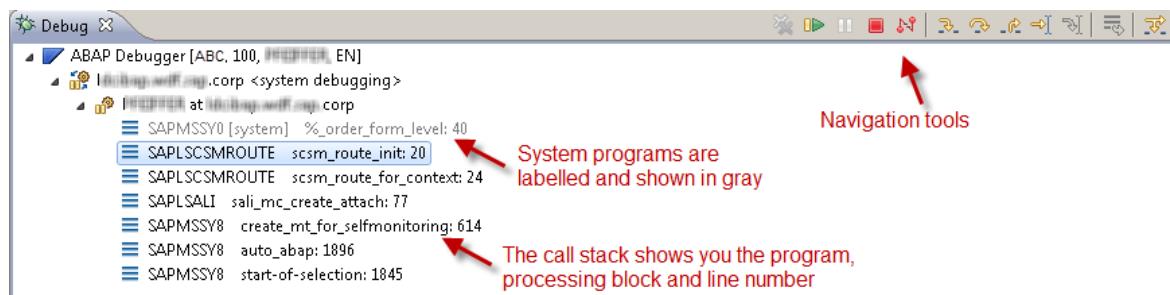
## 4.13 ABAP Debugger

The *Debug* perspective allows you to manage the debugging of ABAP development objects.

When ABAP encounters an active breakpoint, ABAP Development Tools (ADT) opens the *Debug* perspective in the IDE. Depending on how you have adjusted the IDE, the *Debug* perspective may automatically start, or you may be asked to confirm the start.

This section gives you a quick orientation to the *Debug* perspective, so that you know what you are seeing.

- The navigation toolbar on top of the *Debug* perspective lets you control execution in the debugger, stepping through your code, resuming or canceling execution.



The *Debug* view also shows you the active call stack. You can click on entries in the call stack to open the code at that level of the stack. You can then inspect active variables, set breakpoints, or edit your code.

- *Variables* and *Breakpoints* on different tabs.

Name	Value	Actual Type	Technical Type	Length
↳ <Enter variable>				0
↳ SERVER->REQUEST	{0:4*\CLASS=CL_HTTP_REQUEST}	CL_HTTP_REQUEST	Ref to IF_HTTP_REQUEST	0
↳ SY-SUBRC	0	SYSUBRC	I	4
ME	{0:1*\CLASS=CL_ABAP_TOOL...	CL_ABAP_TOOLS_DOCU_HAN...	Ref to CL_ABAP_TOOLS_DOCU...	0
Locals				0
↳ SERVER	{0:2*\CLASS=CL_HTTP_SERVE...	CL_HTTP_SERVER_NET	Ref to IF_HTTP_SERVER	0
↳ : HOST		STRING	CString	0
↳ . : PORT		STRING	CString	0
{0:4*\CLASS=CL_HTTP_REQUEST}				

Type in the name of a variable, or double-click on the variable in the editor to display the variable

Type over the value to set a variable in the debugger

Expand an object to display its attributes

Double-clicking a variable in the source code editor shows you the value and attributes of the variable in the *Variables* view.

In the *Breakpoints* view, you can see the list of active and inactive breakpoints.

Double-click on a breakpoint to jump to its locations in the source code. Mark or unmark a breakpoint to activate or deactivate it. Any such changes are effective immediately in the current debugging session.

- The editor displays the code that you are debugging. You can follow debugger execution in the code, show the values of variables by hovering with the cursor over them, or open them in the *Variables* view by clicking on them.
- You can correct mistakes in your coding directly in the editor; there's no need to switch to the *ABAP* perspective to edit your code.

#### → Tip

SAP recommends to restart the debug session after activating changes in the currently executed code. To do so, terminate or disconnect the debugger and then restart your program.

- The *ABAP Internal Table* view opens at the bottom of the *Debug* perspective. Double-clicking on an internal table opens the table not only in the *Variables* view but also in the *ABAP Internal Table* view. You can type an internal table name directly into the view or find a table in the code that you are debugging.

## Related Information

[Breakpoints in the Debugger - Characteristics \[page 75\]](#)

[Watchpoints \[page 76\]](#)

[Debugging ABAP Code \[page 613\]](#)

## 4.13.1 Breakpoints in the Debugger - Characteristics

Breakpoints in the ABAP Development Tools (ADT) are by default external user breakpoints. They are valid:

- In your current ABAP Project
- For programs running under your ABAP user
- On all of the application servers of the back end system.

These breakpoint attributes mean that you can use breakpoints to debug your ABAP code when you start a program and run it interactively.

You can also use breakpoints to capture in the debugger a program that has been started asynchronously and without your direct action. You do not necessarily have to run your code interactively from the ADT to debug it. A program that you want to capture in the debugger does not even have to be started from your current computer.

For example, you can set a breakpoint in the IDE in ABAP code that handles an HTTP or RFC request. If you send an HTTP request from an Internet browser to the back end system, then the IDE will open the code in the debugger when the request processing reaches your breakpoint. It is best to have the code with the breakpoint open in the IDE editor. But the breakpoint is respected even if you do not have the program with the breakpoint open in the editor.

You can of course also start an ABAP program to debug interactively from the IDE. Even if you run an ABAP `PROGRAM` that executes in the embedded SAP GUI, the IDE *Debug* perspective opens when your breakpoint is reached.

## Dynamic and Static Breakpoints

You can set a static breakpoint at a particular line of code. The breakpoint sticks to the code line; if you add or delete code above the line, then the breakpoint relocates along with the code line.

You can also set dynamic breakpoints. These are breakpoints that are triggered when the program that you are running reaches a particular ABAP statement or exception class.

Dynamic breakpoints take effect for any program that runs under your user. But you can limit their scope to the scope of the debugger debugger in order to avoid stopping at all different kind of programs that are executed along the way.

## Related Information

[Debugging ABAP Code \[page 613\]](#)

[Setting and Managing Breakpoints \[page 614\]](#)

## 4.13.2 Watchpoints

A watchpoint is a conditional breakpoint that is only defined in the **ABAP Debugger** during a running debug session. It is one of the runtime utilities provided to stop and debug ABAP programs. When debugging ABAP code, you can use watchpoints to track the value of individual ABAP variables. The ABAP debugger stops as soon as the value of a watched variable has changed. Watchpoints help ABAP developers monitor the contents of specified variables and the change of their values during runtime processing.

Watchpoint features are as follows:

- Watchpoints can be created while ABAP Debugger is on and are removed automatically when that debug session is terminated.
- Logical conditions may be specified for watchpoints. A watchpoint is provided with a relational operator and comparison field to specify the conditions for interrupting.
- Like Breakpoints, watchpoints may be deleted as needed.
- Once a watchpoint is reached, it is indicated on the program statement and a corresponding alert is given.

### i Note

Because a watchpoint creates a clone of the specified object, it can negatively impact performance and memory, especially with large size data objects like internal tables. Watchpoints are ideal for variables of less size that are active during runtime, but only for a short duration.

## Related Information

[Using Watchpoints \[page 618\]](#)

[Conditions for Watchpoints \[page 620\]](#)

## 4.13.3 Debugging Modes

You as an ABAP developer might experience a “restricted” debug mode during ABAP debugging. This restricted mode is called non-exclusive debugging. By default, every new debug session runs in exclusive mode.

### Exclusive Mode

Exclusive means that the application that is being analyzed exclusively allocates **one work process** on the AS ABAP **for a debug session**. The allocated work process cannot then be used for other sessions. To ensure that the server is not completely blocked, there is a limit to how many exclusive debug sessions are possible at the same time. By default, the limit is half the number of available dialog work processes. On a server with 50 dialog work processes, for example, up to 25 of these can be used for exclusive debugging at the same time.

### → Remember

Once the limit is reached, older versions of ADT are not able to start new debug sessions at all. The current version of the ADT can provide non-exclusive debug sessions in this kind of scenario however. Since non-

exclusive debug sessions do not require an exclusive server work process, their number is theoretically unlimited.

## Non-Exclusive Mode

In non-exclusive debug mode, the system requests a roll-in/roll-out in the application after each debugger interaction. Therefore, every debug step performs an implicit database commit.

Due to the implicit database commit, you must consider the following effects when debugging in non-exclusive mode:

- Since implicit commits are changing operations caused by the debugger, only users with debug-change-authorization can use the non-exclusive debug mode.
- Datasets that are usually committed together might be committed in separate steps. This can cause data inconsistencies at database level. Rollbacks have no effect because intermediate results have already been committed by previous stepping.

### **i** Note

For this reason, the non-exclusive mode is **not enabled in productive systems**. If you experience non-exclusive debug mode in such systems, ABAP server you are using might be configured incorrectly.

- It is not possible to step through `SELECT` and `ENDSELECT` loops because the database cursor needs to be closed when using an implicit database `COMMIT` statement. In cases like this, program execution is terminated by a `DBIF_RSQL_INVALID_CURSOR` short dump.
- It is not possible to debug `Open Cursor` or `Fetch` commands, since the cursor is closed after implicit database commit.

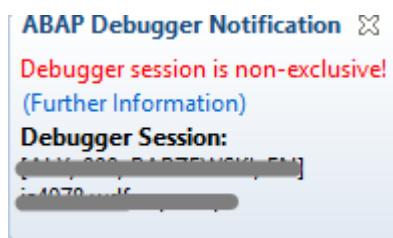
## Debugging in Non-Exclusive Mode in ADT

### **i** Note

 In general, you do not have change authorization while debugging. Thus, you cannot debug in the non-exclusive mode. You can debug, once an exclusive workprocess will be available again. Therefore, wait and retry.

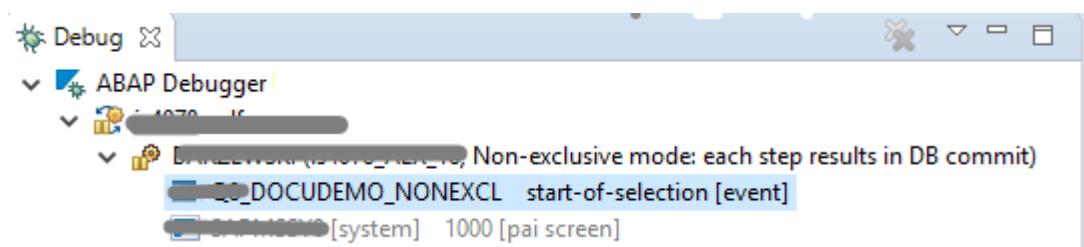
The ABAP Debugger always tries to open an exclusive debug session. The number of exclusive debugging work processes is limited however, as otherwise the application server would not respond at all (in case all work processes were allocated for exclusive debugging).

When you start to debug in ADT, and your session is non-exclusive, you will receive a notification in the form of a flyer (see figure below). You can ignore this notification however, and just continue working. The flyer will disappear automatically after a few seconds.



Notification in ADT appears if debugger session is non-exclusive

In addition, the description text <Non-exclusive mode: each step results in DB commit> is displayed for the corresponding debug session in the *Debug* view.

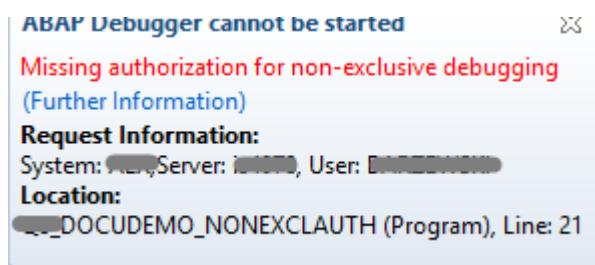


Text that informs you about the non-exclusive debug mode in the Debug view

The ABAP Debugger automatically tries to re-allocate an exclusive debugging session after every single debugging action (such as after a single step). If this attempt is successful, the information text above is no longer displayed.

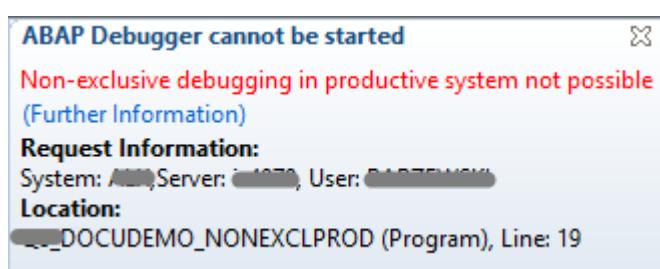
You will be informed through a notification flyer in some other situations in the context of non-exclusive debug mode (and in the case of missing debug authorization):

#### Non-Exclusive Debug Mode Without Debugging Change Privilege for the Current User



The message "Missing authorization for non-exclusive debugging" is issued

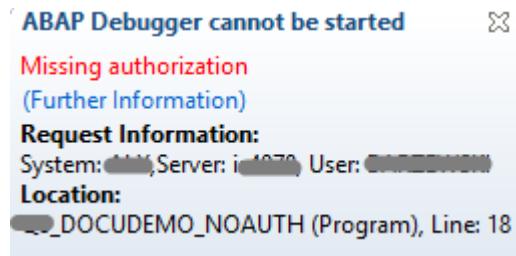
#### In a Productive System, All Exclusive Debugging Work Processes are Occupied



The message "Non-exclusive debugging in productive systems not possible" is issued

## User has no sufficient debugging privileges

This message comes up if you use Terminal ID activation for debugging and have not sufficient privilege to debug the requests of the user, who starts the request with the relevant Terminal ID.



The message "Missing authorization" is issued

## 4.14 Data Preview

Data Preview retrieves, sorts, and filters records available in the ABAP Data Dictionary tables, Views, external Views, and ABAP DDL sources. Data Preview constructs and executes ABAP SQL queries to perform the operations listed below:

- Provide a result-set size in the Data Preview tool, which represents the records that the tool can retrieve from ABAP Dictionary. The default value is 100 and the maximum value is 5000.
- Configure columns to view.
- Sort records in ascending or descending order.
- Set filter criteria. You can set filter criteria for multiple columns.
- Remove filters applied to a column or delete a filter.
- Set local filters to view specific data. Local filters filter the data from the result-set fetched from the ABAP Dictionary.
- View Data preview logs. These logs display SQL queries generated while working with several options in data preview tool.
- Set Distinct Value option for a column. This option allows you to view unique values in tabular format.
- Save records on local system.

### ! Restriction

Data preview does not render properly if you use the dark theme.

## Related Information

[Saving Result Set \[page 482\]](#)

[Viewing Distinct Column Values \[page 481\]](#)

[Viewing Logs \[page 480\]](#)

[Adding or Removing Filters \[page 473\]](#)

[Configuring Columns \[page 471\]](#)

[Setting Result-Set Size \[page 470\]](#)

[Accessing Data Preview \[page 468\]](#)

## 4.15 Feed Reader View

A feed reader enables you to automatically subscribe and display feeds.

### Use

In ABAP Development Tools (ADT), use the *Feeds* view to display and read feeds from ABAP Repository or other native feeds. You can also add or delete a feed.

By default, runtime errors and system messages are displayed in the *Feeds* view. In addition, feeds from other components, such as SAP Gateway can be displayed here.

### Overview

You can also subscribe to any native feed that is published in **Atom** or **RSS** format. In this case however, you only have to specify the URL that you get from the feed provider.

Once you have subscribed to a feed, ADT refreshes it periodically. You can change the default refresh interval when you subscribe to the feed.

Feeds are updated automatically only if you have opened an ABAP project. Otherwise, the feeds are dormant. Dormant feeds are shown in gray type in the *Feed Reader* view.

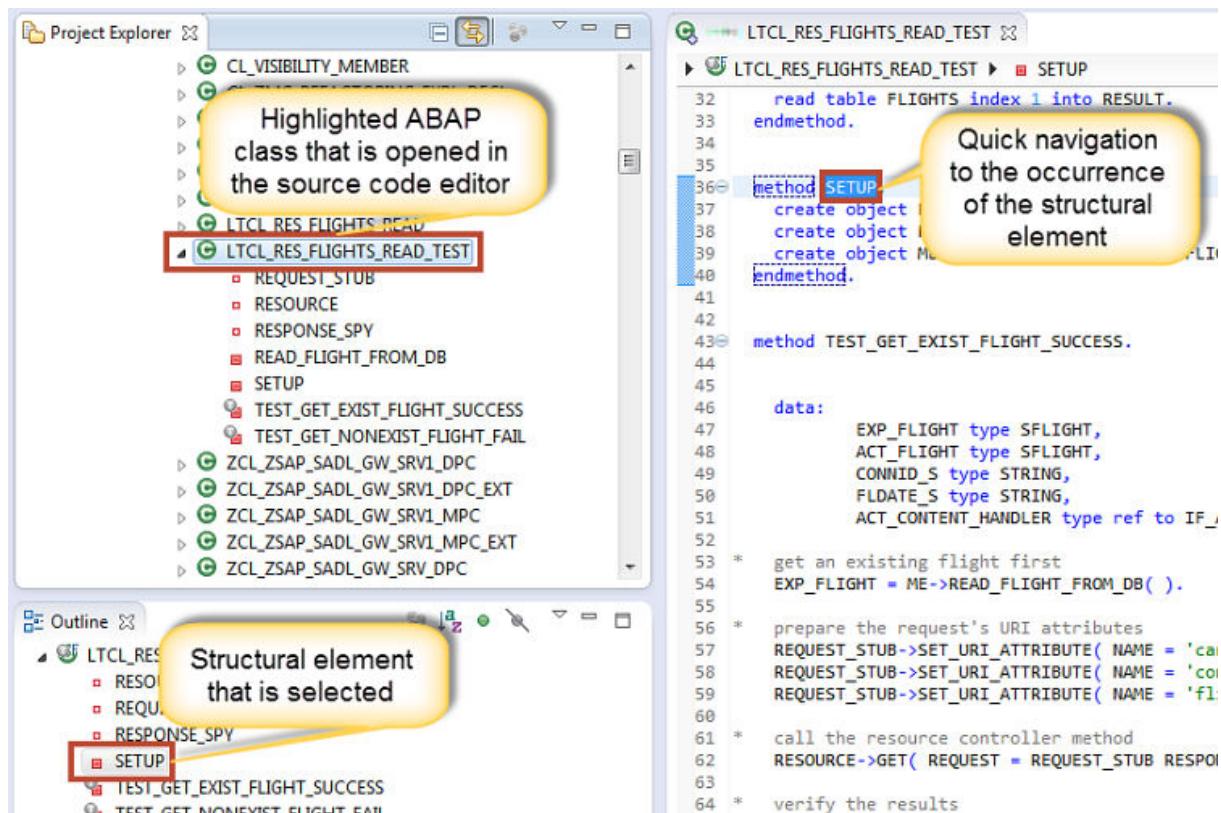
### Related Information

[Getting Feeds \[page 487\]](#)

## 4.16 Outline View

The Outline view displays the internal structure of an ABAP class, interface, or program that is currently opened in the ABAP source code editor.

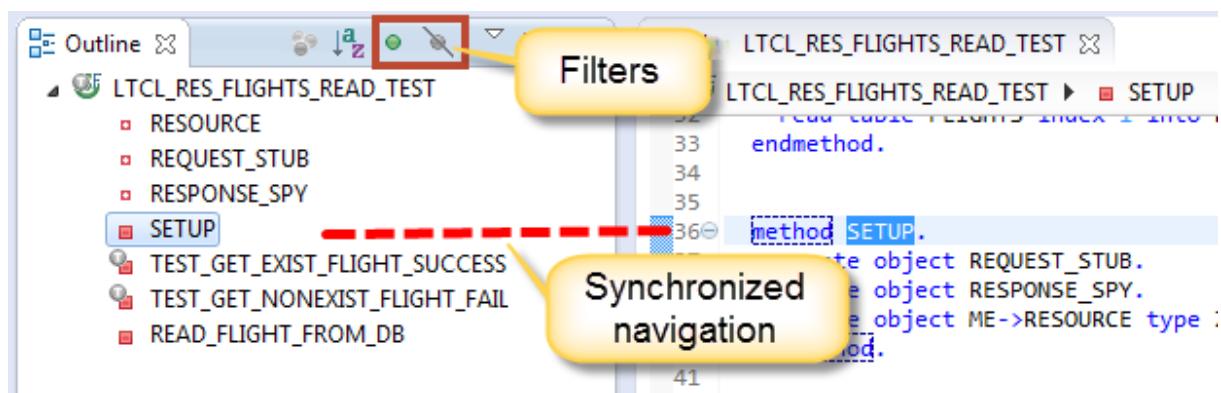
In a tree, structuring elements like attributes, data types, methods, and so on are displayed in the order of their occurrence in the open source code-based development object.



Display of an internal structure of an ABAP class in the Outline view

When you select one of the structural elements, the cursor navigates to its relevant source code position. In addition, you can also filter the display in order to hide:

- All non-public members of ABAP classes
- All members that are not methods, such as attributes, data types, and events



Synchronized navigation from the Outline view to an element

The *Outline* view is opened in the ABAP perspective by default.

## Related Information

[Tools for ABAP Development \[page 19\]](#)  
[Advantages of Using the Outline View \[page 82\]](#)  
[Icons in the Outline View \[page 83\]](#)  
[Viewing the Outline \[page 332\]](#)  
[Using Quick Views \[page 337\]](#)  
[Viewing the ABAP Type Hierarchy \[page 334\]](#)  
[Getting Orientation in the Source Code Using Breadcrumbs \[page 335\]](#)  
[Short Texts \[page 26\]](#)

### 4.16.1 Advantages of Using the Outline View

The *Outline* view supports you by providing the following advantages:

- Particularly for development objects with a more complex structure, the outline provides a good overview of the internal object structure. Here, you can also avail of the filter functions with which you can display or hide certain elements in order to achieve a better overview.
- The outline is synchronized with the contents of the source editor. Therefore, when you select an element in the *Outline* view, you can navigate quickly to the corresponding position in the ABAP source code.

#### Note

The outline is synchronized with the contents of the editor, even if the latter are not yet saved. If you add a new attribute in the class editor, for example, it is immediately shown in the *Outline* view (without the editor contents having been saved beforehand).

## Related Information

[Tools for ABAP Development \[page 19\]](#)  
[Icons in the Outline View \[page 83\]](#)  
[Viewing the Outline \[page 332\]](#)  
[Using Quick Views \[page 337\]](#)  
[Viewing the ABAP Type Hierarchy \[page 334\]](#)  
[Getting Orientation in the Source Code Using Breadcrumbs \[page 335\]](#)

## 4.16.2 Icons in the Outline View

In the *Outline* view, the following icons are provided:

### Main Icons

Icon	Description
	ABAP class
	ABAP interface
	Method (public)
	Method (protected)
	Method (private)
	Method (visibility unknown)
	Attribute (public)
	Attribute (protected)
	Attribute (private)
	Type (public)
	Type (protected)
	Type (private)
	Event (public)
	Event (protected)
	Event (private)
	Event handler method (public)
	Event handler method (protected)
	Event handler method (private)
	Friend
	ABAP program
	ABAP include

Icon	Description
	Subroutine
	Global variable
	Type
	Macro
	Function group
	Function module
	Type group
	Test seam for ABAP programs

## Decorators

Decorator	Description
	Static member
	Final member
	Abstract member
	Redefinition
	Test class or test method
	Read-only
	Constant
	Constructor
	Receiving event
	Sending event

## Related Information

- [Outline View \[page 80\]](#)
- [Advantages of Using the Outline View \[page 82\]](#)
- [Tools for ABAP Development \[page 19\]](#)
- [Viewing the Outline \[page 332\]](#)
- [Using Quick Views \[page 337\]](#)
- [Viewing the ABAP Type Hierarchy \[page 334\]](#)

## 4.17 Quick Assists

Quick assists help you to change ABAP source code in a semi-automated way.

ABAP Development Tools (ADT) provides a set of quick assists that can be divided into the following categories:

- [ABAP Refactorings \[page 86\]](#) for changing the structure of source code without changing its behavior
- [ABAP Quick Fixes \[page 90\]](#) for resolving errors and warnings

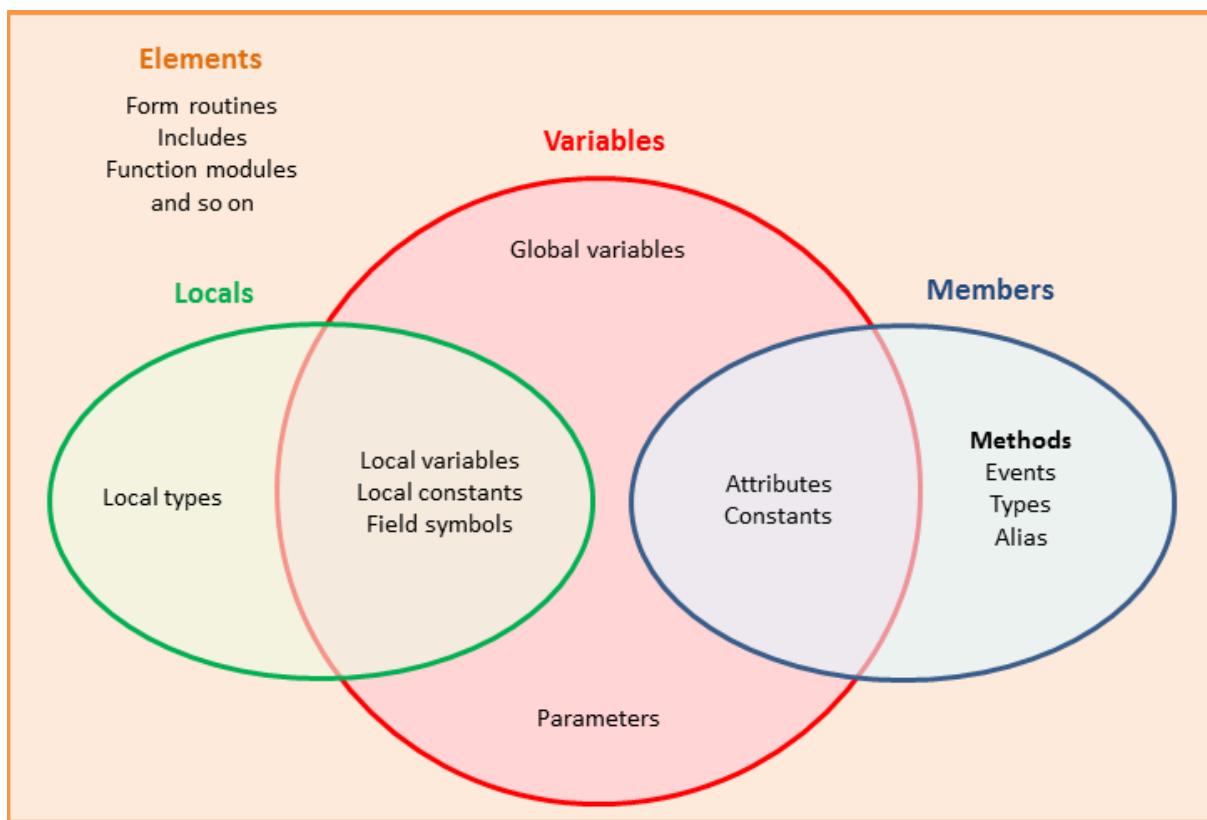
Various quick assists are neither refactorings nor quick fixes. These are, for example, generating constructors or creation of text symbols.

All supported transformations are available through the quick fix menu that can be triggered from the Source menu or through the `Ctrl 1` shortcut. As an alternative the *Quick Assist* view provides a permanent view on the available quick assists.

The proposals are context-sensitive. This means that, depending on your current source code position, appropriate proposals are provided.

Many of the quick assist options can be triggered on identifiers from the following types:

- **Elements** describe all named parts of the ABAP source code.
- **Variables** are named values that stand for a storage location.
- **Members** represent the data and behavior of an ABAP class.
- **Locals** are internal data fields within a class that can have any data type.
- And so on



Positions in the ABAP source code where a refactoring option can be executed

## Related Information

- [Quick Assist View \[page 91\]](#)
- [Quick Assist Overview \[page 92\]](#)
- [Refactoring ABAP Source Code \[page 342\]](#)
- [Applying ABAP Quick Fixes \[page 402\]](#)
- [Applying Other Quick Assists \[page 422\]](#)

## 4.17.1 ABAP Refactorings

Refactoring is the structural improvement of source code. The functional behavior of source code is retained.

Your wish to minimize the effort required for error analysis and the need to improve the following items may prompt you to refactor source code:

- readability
- understandability
- maintenance
- enhancement possibilities

Good support for automated refactoring of source code is a key part of test-driven development (TDD) and other advanced programming techniques. To this end, ABAP Development Tools (ADT) provides various refactoring functionalities.

## Related Information

[Quick Assists \[page 85\]](#)

[Quick Assist Overview \[page 92\]](#)

[Refactoring ABAP Source Code \[page 342\]](#)

### 4.17.1.1 Extracting Constants from Literals

In the source code editor of a development object, you can substitute literals with constants. This enables you to replace a specific, unchanging value that is used several times and to improve the readability of ABAP source code. However, literals should only be extracted if the abstraction results in a compliant solution.

#### i Note

Literals are separated into:

- Textual literals: character, string, string template
- Numeric literals: integer (> i), float (> decfloat 16)

#### Example

In the following examples, literals can be replaced with a constant:

- The number **18** with the constant "voting\_age"
- The message **'Hello World'** with a generated constant "hello\_world" wherever it is used in the source code of a certain development object

The following examples display the extraction of a certain literal in a constant:

Literal Type	Example	Constant Name	Constant Type	Constant Value
Character	'Hello World !'	hello_world	String	'Hello World !'
String	'Hello World !'	hello_world	String	'Hello World !'
String Template	Hello { name }!	hello	String	'Hello ' or '!'
Integer	255	_255	i	255
Float	'3.141529'	_3141529	decfloat16	'3.141529'

## Functionalities

You can extract constants from a literal using the following refactoring functions:

- Extract a local constant from a literal that is used only in the current method. The new constant definition is created in the current method. Here all occurrences of the literal are replaced. Extract a member constant from a literal to define it in the private section of the current ABAP class. Here all occurrences of the literal are replaced.
- Use an existing constant with the same value as the literal.

## Results

When you extract a constant in the source code of a development object, the definition `CONSTANTS 'your name' TYPE <type> VALUE <value>` is added. Also, in the development object the corresponding literal is replaced with the constant.

The name of the generated constant is derived from the literal and automatically generated. These are the rules for the creation of the constant name:

- Spaces are replaced by underscores.
- Invalid characters (for example &, %, blanks, and so on) are eliminated.
- The maximum length is 30 characters.
- If the literal starts with a number, an underscore prefix is set.

Finally a pragma (`##NO_TEXT`) is added in the declaration. Therefore, no translation warnings will be issued by the ABAP Compiler syntax check.)

## Limitations

The following limitations exist for extracting constants from literals:

- Member constants are always created as private members.
- If the same literal is already used in other classes or interfaces, these instances are not replaced.
- Expressions cannot be extracted to a constant, because in ABAP they are allowed in constant definitions.
- This option is provided only within ABAP classes and interfaces.

## Related Information

[Extracting Variables from Literals \[page 89\]](#)

[Quick Assists \[page 85\]](#)

[Extracting Constants \[page 353\]](#)

[Extracting and Converting Variables \[page 357\]](#)

[Changing Visibility of Members \[page 378\]](#)

## 4.17.1.2 Extracting Variables from Literals

In the source code editor of a development object, you can substitute literals with variables. This enables you to replace a specific, unchanging value that is used several times or to improve the readability of ABAP source code. However, literals should only be extracted if the abstraction results in a compliant solution.

### i Note

Literals are separated into:

- Textual literals: character, string, string template
- Numeric literals: integer (> i), float (> decfloat 16)

### Example

In the following examples, literals can be replaced with a variable:

- The number **50** with the constant "max\_speed"
- The message **'Hello Planet'** with a generated constant "hello\_planet" wherever it is used in the source code of a certain development object

The following examples display the extraction of a certain literal in a variable:

Literal Type	Example	Constant Name	Constant Type	Constant Value
Character	'Hello Planet !'	hello_planet	String	Float
'Hello Planet !'	'Hello Planet !'	hello_planet	String	'Hello Planet !'
String Template	Hello { name }!	hello	String	'Hello ' or '!'
Integer	255	_255	i	255
Float	'3.141529'	_3141529	decfloat16	'3.141529'

### Limitations

The following limitations apply when extracting variables from literals:

- Member variables are always created as private members.
- If the same literal is already used in other classes or interfaces, these instances are not replaced.
- Currently, expressions cannot be extracted to a variable.
- This option is provided only within ABAP classes and interfaces.

### Functionalities

You can extract variables from a literal using the following refactoring functions:

- Extract a local variable from a literal that is used only in the current method. The new variable definition is created in the current method. Here all occurrences of the literal are replaced.

- Extract an attribute from a literal to define it in the private section of the current ABAP class. Here all occurrences of the literal are replaced. Attributes are always created as private members.

## Results

When you extract a variable in the source code of a development object, the definition DATA 'your name' TYPE <type> VALUE <value> is added. Also, in the development object the corresponding literal is replaced with the variable.

The name of the generated variable is derived from the literal and automatically generated. The following limitations exist for the creation of the variable name:

- Spaces are replaced by underscores.
- Invalid characters (for example &, %, blanks, etc.) are eliminated.
- The maximum length is 30 characters.
- If the literal starts with a number, an underscore prefix is set.

Finally, a pragma (#NO\_TEXT) is added in the declaration. Therefore, no translation warnings will be issued by the ABAP Compiler syntax check.

## Related Information

[Extracting Constants from Literals \[page 87\]](#)

[Quick Assists \[page 85\]](#)

[Extracting Constants \[page 353\]](#)

[Extracting and Converting Variables \[page 357\]](#)

[Changing Visibility of Members \[page 378\]](#)

## 4.17.2 ABAP Quick Fixes

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

Quick Fixes can be used to make typing much faster.

## Related Information

[Applying ABAP Quick Fixes \[page 402\]](#)

[Quick Assists \[page 85\]](#)

[ABAP Refactorings \[page 86\]](#)

[Quick Assist Overview \[page 92\]](#)

### 4.17.3 Quick Assist View

The *Quick Assist* view supports you when executing refactorings and ABAP quick fixes, or when generating ABAP source code.

It interacts directly with the ABAP source code editor. At every position where a quick assist can be applied, the content of the *Quick Assist* view is therefore automatically updated according to the selection made.

As an alternative to the Quick Fix (Ctrl 1) menu, the *Quick Assist* view (Ctrl Shift 1) provides a permanent view of the available quick assists.

The layout of the *Quick Assist* view is divided into the following areas:

- **Proposal** overview. This provides a list of possible quick assists at the current cursor position in the source code editor.
- **Description per proposal** This provides details about the operating steps of the quick assist that you can do next.

#### i Note

The display depends on the entry that you have selected in the *Proposal* overview.

- **Previous Result** Displaying the changes performed by the recently applied quick assist. If the quick assist cannot be applied, an error message is displayed.

From the integrated toolbar, you can do the following:

- Apply the selected proposal.
- Disable link with editor to turn off automatic proposal calculation.
- Rearrange the *Quick Assist* view layout setup to horizontal, vertical, or automatic to your selected perspective.
- Delete the displayed Previous Result content.
- Display or clear the history of the executed quick assists.
- Enable a preview of changes to be performed.

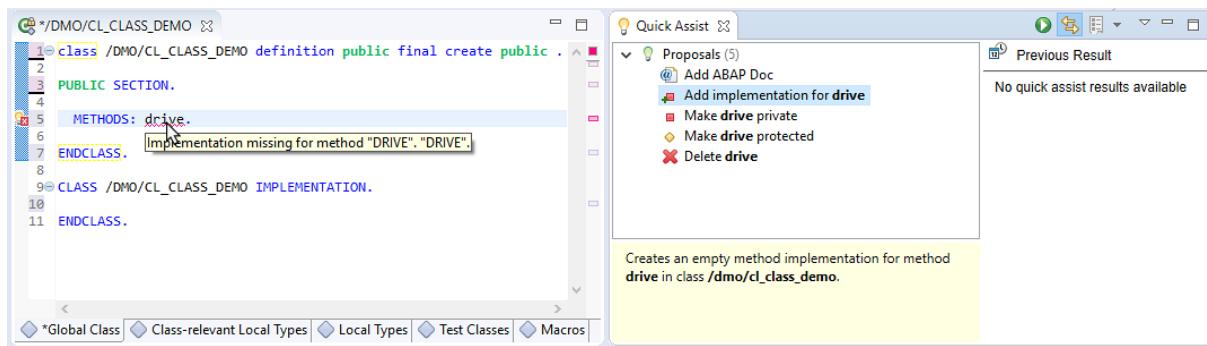
#### i Note

To open the *Quick Assist* view, choose  **Window**  **Show view**  **Other...**  from the menu. In the *Show view* dialog, select **Quick Assist** from the ABAP tree.

## Example

### Before Execution

You get an error message as the method `drive` is not implemented yet. When you select the method name, the possible quick assists are displayed in the *Proposals* area. If you select one of these, additional information is displayed in the area below.



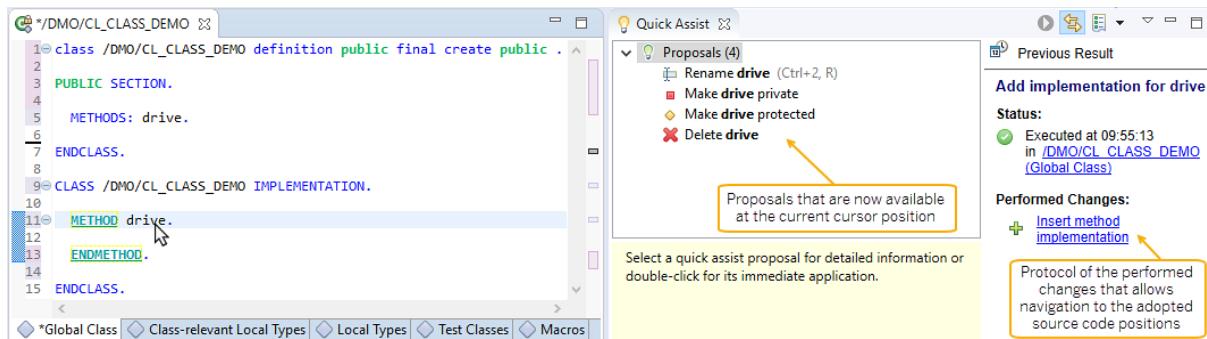
Quick assist view setup before executing a quick assist

### After Execution

After double-clicking the proposal or selecting **Apply Proposal** from the context menu, the method definition is moved to the private section of the ABAP class.

The tree in the *Proposals* area is updated to the quick assists that are now available at the selected cursor position.

If the quick assist is applied without any errors, the *Previous Result* area is highlighted in green for two seconds. Here, also the status with the last execution date and the insertion result of the performed changes are now displayed. You can navigate from here directly to the positions where changes have been performed in the source code editor.



Quick assist view setup after executing a quick assist

## Related Information

[Quick Assists \[page 85\]](#)

## 4.17.4 Quick Assist Overview

In ABAP Development Tools, you can apply the following refactoring options or ABAP quick fix at the following positions:

- [Methods \[page 93\]](#)
- [Constants \[page 94\]](#)
- [Variables \[page 94\]](#)

- [Members \[page 95\]](#)
- [Text Literals \[page 95\]](#)
- [Exceptions \[page 96\]](#)
- [ABAP Unit Test Classes \[page 97\]](#)
- [Others \[page 98\]](#)

**i Note**

The Rename functionality can be applied to all identifiers.

## Related Information

- [ABAP Refactorings \[page 86\]](#)
- [ABAP Quick Fixes \[page 90\]](#)
- [Quick Assist View \[page 91\]](#)
- [Refactoring ABAP Source Code \[page 342\]](#)
- [Applying ABAP Quick Fixes \[page 402\]](#)
- [Applying Other Quick Assists \[page 422\]](#)

### 4.17.4.1 Methods

Function	Description
<a href="#">Extracting Methods from Statements [page 348]</a>	Creating a new method in the current class. The selected code is moved into the body of the new method and replaced with a call to the new method. Further occurrences of similar code are not replaced.
<a href="#">Extracting Methods from Expressions [page 351]</a>	Creating a new method in the implementation of an ABAP class that returns the result of a selected expression. The selected expression is replaced with the call to the new method. Further occurrences of similar expressions are not replaced.
<a href="#">Creating Method Implementations from the Method Definition [page 404]</a>	Creating an empty method implementation from an existing declaration in an ABAP class.
<a href="#">Creating Method Definitions from Implementation Parts [page 406]</a>	Creating empty method implementations of the methods defined in an implemented ABAP interface and of other unimplemented methods within an ABAP class.
<a href="#">Creating Implementation Parts for Unimplemented Methods [page 409]</a>	Promoting the implementation part of methods that are defined in an ABAP interface as well as method stubs of other unimplemented methods.
<a href="#">Creating Methods from Method Calls [page 414]</a>	Creating a method from the method call. The signature is derived from the existing method call.
<a href="#">Generating Class Constructor Methods [page 423]</a>	Creating an empty <code>class constructor</code> method in the public section of the current ABAP class.

Function	Description
<a href="#">Generating Constructor Methods [page 424]</a>	<p>Creating a constructor in the public section of the current ABAP class.</p> <div style="border-left: 3px solid #0070C0; padding-left: 10px; margin-left: 20px;"> <b>i Note</b> <p>If the class has attributes, a dialog is opened where you can select the attributes that should be instantiated by the constructor.</p> </div>
<a href="#">Generating Factory Methods [page 425]</a>	<p>Creating a static <code>create</code> method in the public section of the current ABAP class.</p> <div style="border-left: 3px solid #0070C0; padding-left: 10px; margin-left: 20px;"> <b>i Note</b> <p>If the class has attributes, a dialog is opened where you can select the attributes that should be instantiated by the factory method.</p> </div>

## 4.17.4.2 Constants

Function	Description
<a href="#">Extracting Constants from Literals [page 353]</a>	Creating a constant with the value of the literal and replacing all occurrences.
<a href="#">Reusing Existing Constants [page 355]</a>	Replacing a literal with an existing constant. All other occurrences of the same literals are not changed.
<a href="#">Converting Locals to Class Members [page 360]</a>	Converting a local constant, local variable, or local type to a class member such as a member constant, attribute, or member type.

## 4.17.4.3 Variables

Function	Description
<a href="#">Extracting Local Variables from Expressions [page 358]</a>	Assigning the selected expression to a new local variable. The selected expression is replaced with the new local variable.
<a href="#">Assigning a Statement to a New Variable [page 363]</a>	Assigning the value of the selected statement to a new local variable or attribute.
<a href="#">Converting Locals to Class Members [page 360]</a>	Converting a local constant, local variable, or local type to a class member such as a member constant, attribute, or member type of the current class.
<a href="#">Converting Local Variables to Parameters [page 362]</a>	In a certain method, converting an existing local variable to a new parameter.
<a href="#">Declaring Variables from Usage [page 368]</a>	Creating a declaration for an attribute within a method.

Function	Description
<a href="#">Declaring Inline Variables Explicitly [page 365]</a>	In the method signature, converting an existing inline declaration of a local variable to an explicit declaration.
<a href="#">Using Similar Variables [page 369]</a>	You have following possibilities for using similar variables: <ul style="list-style-type: none"> <li>• <a href="#">Correcting Misspelled Variables [page 370]</a></li> <li>• <a href="#">Correcting an Interface Attribute Access [page 371]</a></li> </ul>
<a href="#">Deleting Unused Variables [page 375]</a>	Supported deletion of unused data declarations and variables.

## 4.17.4.4 Members

Function	Description
<a href="#">Changing Visibility of Members [page 378]</a>	Changing the visibility of a member by moving it into the public, protected, or private section.
<a href="#">Converting Locals to Class Members [page 360]</a>	Converting a local constant, local variable, or local type to a class member such as a member constant, attribute, or member type of the current class.
<a href="#">Pull-up Members to Superclass [page 380]</a>	Removing member definitions from a subclass and adding it to the superclass.
<a href="#">Pull-up Members to Interface [page 382]</a>	Removing member definitions and adding them to the implemented interface. Additionally, aliases are declared to avoid invalidation of existing usages.

## 4.17.4.5 Text Literals

Function	Description
<a href="#">Creating a Text Symbol in the Text Pool [page 431]</a>	Creating a text symbol in the text pool of an ABAP program.
<a href="#">Editing Text Symbols [page 434]</a>	Changing a text symbol in the text pool of an ABAP program.
<a href="#">Correcting Inconsistencies Within Text Symbols [page 435]</a>	Balancing mismatches between source code and text pool.
<a href="#">Switching Notations [page 436]</a>	Adopting text changes of an existing text symbol in the text pool or the source code and vice versa.

## 4.17.4.6 Exceptions

### Source Code Selection That Includes Raised Exceptions

The exception(s) are added to the signature of the calling method or surrounded with a TRY CATCH block.

Function	Description
<a href="#">Propagating All Exceptions [page 386]</a>	Adding the unhandled exceptions of the selected block to the signature of the calling method in order to propagate them.
<a href="#">Surrounding with TRY CATCH [page 389]</a>	<b>Single catches:</b> Surrounding the selected block of statements with a try catch statement to handle raised exception(s). Each exception is handled in a separate catch block
	<b>Multi catch:</b> Surrounding the selected block of statements with a try catch statement to handle raised exception(s). One catch block handles all exceptions.

### Existing TRY CATCH Block

The statement(s) are surrounded by a TRY CATCH block in order to handle the exceptions.

Function	Description
<a href="#">Extracting a Catch Variable [page 393]</a>	Adding a local variable and adding the INTO clause to an existing catch block.
<a href="#">Extending a TRY CATCH Statement [page 397]</a>	Adding a new catch block to an existing try catch statement.
<a href="#">Removing a TRY CATCH Statement [page 401]</a>	Removing the entire try catch statement.
<a href="#">Splitting a MULTI CATCH Block [page 399]</a>	Replacing the existing multi catch block by individual catch blocks per exception.

### RAISE EXCEPTION Statement

Function	Description
<a href="#">Propagating an Exception [page 388]</a>	Adding an exception class to the signature of a method signature that is based on an existing RAISE EXCEPTION statement.
<a href="#">Extracting an Exception Variable from a RAISE Statement [page 395]</a>	Adding a variable declaration for the exception and source code to instantiate the exception before raising it.

## 4.17.4.7 ABAP Unit Test Classes

Lists all the supported quick assist proposals for context specific source code generation in ABAP unit test classes.

## CDS Test Double Framework

Lists all the supported quick assist proposals for context specific source code generation while writing ABAP unit tests for a CDS entity.

Functions	Description
Create all the CDS test fixture methods	Create all the CDS test fixture methods for the given CDS entity under test along with the corresponding method definition. It generates three fixture methods - <code>class_setup</code> , <code>setup</code> , and <code>class_teardown</code> .  If some of these three fixture methods are available, then those are not generated again. If all the three fixture methods are available, then the system does not show any proposals.
Create CDS test <code>class_setup</code> fixture method	Creating a new fixture method <code>class_setup</code> to set up the test environment including creation of required Doubles and Clones (s). If all the three methods are available, then this proposal is not be visible at all. Also, if this fixture method definition already exists, then the system does not display any proposals.
Create CDS test <code>setup</code> fixture method	Creating a new fixture method <code>setup</code> to clear the test data for all the Doubles used in the test method before executing each test method. If this fixture method definition already exists, the system does not display any proposals.
Create CDS test <code>class_teardown</code> fixture method	Creating a new fixture method <code>class_teardown</code> to delete the generated database entities (Doubles and Clones) at the end of test class execution. If this fixture method definition already exists, the system does not display any proposals.

Functions	Description
Prepare test data	Generating the boiler plate code for getting the test double from the CDS test environment for a given dependency of the CDS Entity under test. It also sets the test data into the test double framework. This proposal must be used for each row of test data and each dependency separately. This proposal is displayed only with in the methods inside an ABAP test class.
Create a new Test Method and prepare test data	Creates a new test method inside the Test Class. It creates doubles from the CDS test environment for the given dependency of the CDS Entity under test using wizard. Test data can also be set to the selected dependencies as a second step of the code generation.  Test method name must be provided for this proposal.
Prepare test data (Using wizard)	This proposal is displayed inside the existing test method only. Allows to set data to the selected dependencies using wizard and generates the code accordingly. Test method name is prefilled for this proposal.

## Related Information

[Writing a Test for a CDS Entity Using Quick Assist Proposals \[page 523\]](#)

## 4.17.4.8 Others

Identifier	Function	Description
Classes and Interfaces	<a href="#">Creating ABAP Classes or ABAP Interfaces from Usage [page 417]</a>	Starting the creation wizard of a global ABAP class or interface directly from the name of the missing repository object.
Classes	<a href="#">Generating Class Constructor Methods [page 423]</a>	Creating an empty <code>class constructor</code> method in the public section of the current ABAP class.
	<a href="#">Generating Constructor Methods [page 424]</a>	Creating a constructor in the public section of the current ABAP class.

**i Note**

If the class has attributes, a dialog is opened where you can select the attributes that should be instantiated by the constructor.

Identifier	Function	Description
	<a href="#">Generating Factory Methods [page 425]</a>	Creating a static <code>create</code> method in the public section of the current ABAP class.
	<a href="#">Generating Getters and Setters [page 427]</a>	<p><b>i Note</b></p> <p>If the class has attributes, a dialog is opened where you can select the attributes that should be instantiated by the factory method.</p>
	<a href="#">Regenerating a Constructor for Exception Classes [page 428]</a>	
Function Modules	<a href="#">Creating ABAP Function Modules from Usage [page 420]</a>	Starting the creation wizard of an ABAP function module from the name of the missing repository object.
Includes	<a href="#">Creating ABAP Includes from Usage [page 421]</a>	Starting the creation wizard of an ABAP include from the name of the missing repository object.
Text Symbols	<a href="#">Creating and Editing Text Symbols (Quick Assists) [page 430]</a> <a href="#">Editing Text Symbols [page 434]</a> <a href="#">Correcting Inconsistencies Within Text Symbols [page 435]</a>	Creating a text symbol in the text pool of an ABAP program. Changing a text symbol in the text pool of an ABAP program. Balancing mismatches between source code and text pool.
Notations	<a href="#">Switching Notations [page 436]</a>	Adopting text changes of an existing text symbol in the text pool or the source code and vice versa.

## 4.18 Software Component

A software component defines a delivery and product unit of a SAP software product. It comprises a set of packages that are delivered in a single unit.

### SAP Cloud Platform ABAP Environment

#### Use

A software component ...

- is the **most** granular unit which can be imported or exported.
- is used to group a multiple ABAP packages in a single unit. This minimizes the complexity of your development activities.
- contains ABAP Repository objects which **cannot** be used in other software components.

## Local Objects

By default, SAP provides the `ZLOCAL` ABAP package for local development. Development objects assigned to this package cannot be transported within the system landscape of your company.

The `ZLOCAL` package is available in the *Favorite Packages* tree in the *Project Explorer*.

If you want to create a new ABAP package for local development, you use the `ZLOCAL` superpackage.

If you want to create new local development objects, you assign them to the `ZLOCAL` software component.

### i Note

The local `$TMP` ABAP package is no longer available.

## Namespaces

The prefix `/DMO/` is SAP's registered namespace for samples and other demo content.

### → Recommendation

SAP recommends that you to

- create your own namespace(s) for production development activities **or**
- use the `Z*` and `Y*` namespaces for your customer-specific development activities.

### i Note

The `Z*` and `Y*` namespaces are provided by default.

## Administration

An administrator creates software components using the *Software Component Lifecycle Management* app inside the *Administration SAP Fiori Launchpad* of your service instance.

### i Note

To create a software component, the `SAP_A4C_BC_MSCL_PC` business catalog needs assigned to the administrator. Please contact your administrator for further information.

## Related Information

[ABAP Packages \[page 38\]](#)

[ABAP Development Objects \[page 19\]](#)

## 4.19 Released APIs

The API state of a development object defines whether a development object can be used for custom development.

### API States

Objects can be released as APIs for different purposes. Depending on the underlying release contract, released objects need to adhere to different stability criteria. The API State tab is visible only for types of objects which can be released as APIs. You have the following release contracts available:

- [Add Custom Fields via Key User App \(CO\) \[page 101\]](#)
- [Use System-Internally \(C1\) \[page 102\]](#)
- [Use as Remote API \(C2\) \[page 102\]](#)
- [Deprecation \[page 103\]](#)

### Related Information

[Setting the API Release State \[page 465\]](#)

### 4.19.1 Add Custom Fields via Key User App (CO)

This API state ensures stability at dedicated extension points in APIs. These extension points must only be used for adding custom fields using app *Custom Fields and Logic* or when adding custom fields to web dynpro applications with the help of web dynpro configuration.

In addition, SAP development can define extension points in the types of objects listed below. These extension points are available exclusively in the context of the app *Custom Fields and Logic*:

- Business Object Types
- CDS Entities
- OData Services
- Function Modules
- SOAP Services
- IDoc Types
-

## 4.19.2 Use System-Internally (C1)

This API state ensures a technically stable public interface for use in extension items created using key user apps like Custom CDS Views or Custom Analytical Queries, or with means of the ABAP Development Tools. Compatibility of the public interface includes all its existing parameters, elements, associations to released targets, and their technical data types. Further optional parameters, elements, or associations might be added later, but existing parameters, elements, or associations must not be changed or removed. Alphanumeric elements can be prolonged, but must not be shortened.

Affected Object Types:

- Authorization Fields
- Authorization Objects
- BAdI Definitions
- Behavior Definitions
- CDS Entities
- Classes
- Data Elements
- Domains
- Function Modules
- Interfaces
- Message Classes
- Search Helps
- Structures
- Table Types
- Type Groups
- Transformations

## 4.19.3 Use as Remote API (C2)

### i Note

This API state is set by SAP for delivered remote APIs and not intended for custom development objects .

This contract ensures a technically stable public interface for use as remote API. Like Use System-Internally, compatibility of the public interface includes all its existing parameters, elements, associations to released targets, and their technical data types. Further optional parameters, elements, or associations might be added later, but existing parameters, elements, or associations must not be changed or removed. Furthermore, to ensure that external consumers of the API do not need to be adjusted after an upgrade, the prolongation of its elements and parameters is not allowed.

Affected Object Types:

- CDS Entities
- Service Bindings

## 4.19.4 Deprecation

Delivered development objects can be deprecated by SAP. These development objects are then no more displayed for example in code completion. Where a deprecated development object is used, a syntax warning is displayed indicating a successor. It is recommended to switch to the successor as soon as possible.

As a customer, you can mark a released custom development object as deprecated, if you have defined an improved successor that is to be used instead.

Additionally, you can mark single elements or public associations within a CDS entity as deprecated and specify their successors. This takes effect only once the CDS view itself was released as stable API.

### **i** Note

You must define a released successor for every deprecated development object and every deprecated element or public association of a released CDS entity.

### Affected Object Types:

- CDS entities
- ABAP classes
- Data elements
- Domains
- Function modules
- ABAP interfaces
- Structures
- Table types

### Related Information

[Deprecating Development Objects \[page 466\]](#)

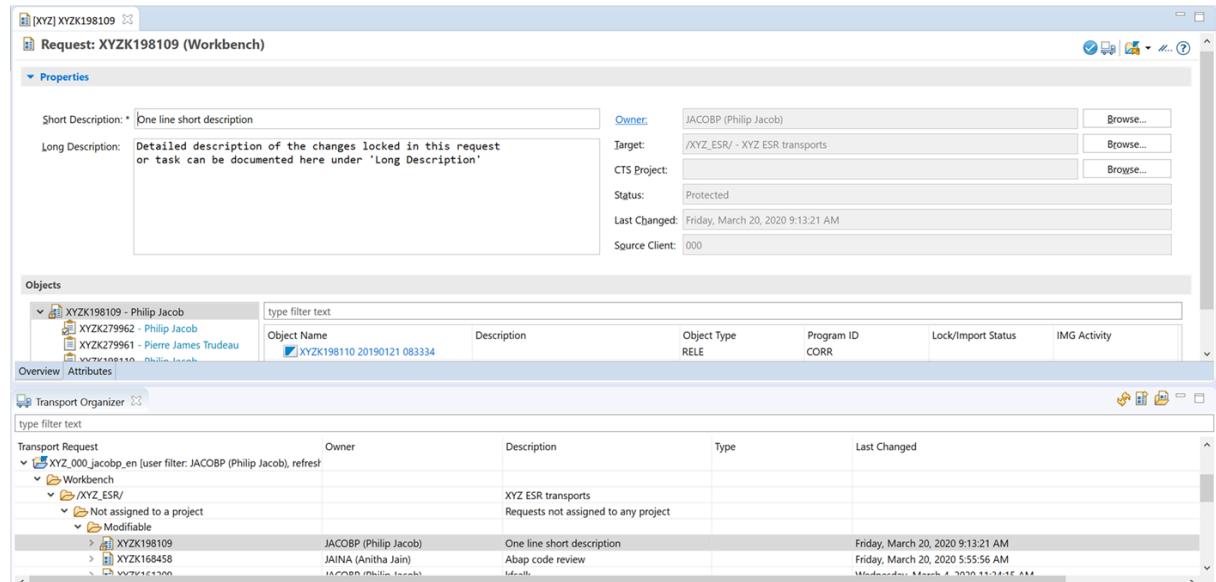
## 4.20 Transport Organizer

The *Transport Organizer* in ABAP Development Tools (ADT) contains the basic transport functions of workbench. The requests record changes made to the ABAP workbench objects.

### Overview

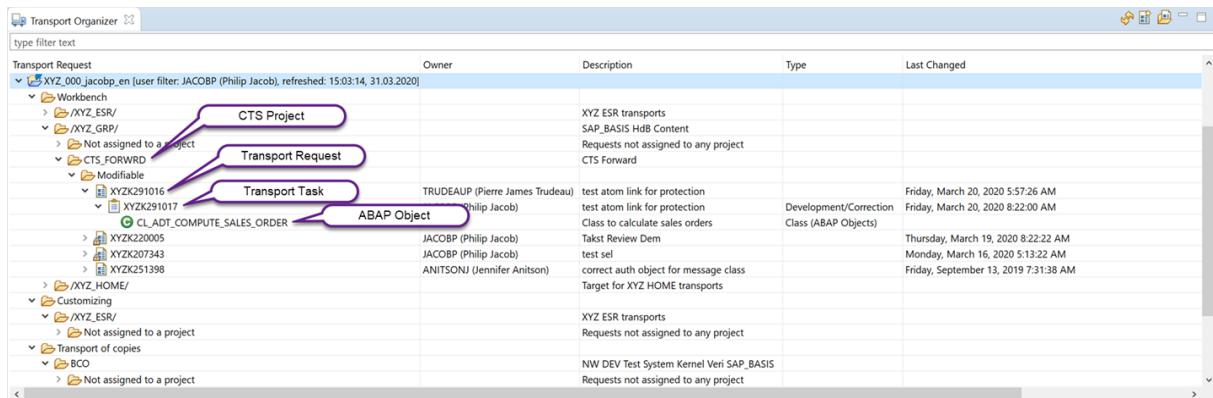
The key feature of the *Transport Organizer* view is the option to display the assignment of objects to the tasks and transport requests in a folder structure. Also, to perform some basic actions.

The display depends on the logged-on user. For example, you can display the most important information relating to the transport of change requests of a user without leaving the development environment. A request or task has an editor to view and edit the properties and objects.



Display of a request that is selected in the Transport Organizer view and the Request Editor is opened in ADT

## Transport Organizer View



Transport Organizer View

The *Transport Organizer* view displays for each ABAP project the following columns:

- *Transport Request*:

Folder Structure of the Transport Request Tree

Folder Level	Label	Description
1	Name of the ABAP project, client, user name, and logon language	Details of connection of an ABAP project to an ABAP system
2	Workbench	Transport requests of the SAP GUI-based workbench that have recorded changes made to ABAP objects.
	<b>i Note</b>	Here you can only open and edit modifiable transport requests, create or delete tasks, or edit ABAP objects.
2	Customizing	Modifiable and released transport requests of the customer-specific development activities
3	Transport Groups	A transport domain can consist of one or more transport groups that share the same transport directory of one or more ABAP systems.
3	Local Change Requests	Development activities that are not exported
	<b>i Note</b>	These changes are transported only within SAP's system or customer systems.
4	CTS Project	Development work and customizing activities can be grouped into project structure.  And the changes that are done independently in different projects can also be imported independently into the target systems

Folder Level	Label	Description
5	Modifiable	<p>Unreleased transport requests and tasks that can be edited in addition to the objects that have been released</p> <p><b>i Note</b></p> <p>The <i>Modifiable</i> subfolder is displayed for all transport groups and in the Local Change Request folder.</p>
5	Released	<p>All transport requests, tasks, and objects that have been released within the last two weeks</p> <p><b>i Note</b></p> <p>The <i>Released</i> subfolder is displayed for all transport groups and in the Local Change Request folder.</p> <p>Released content cannot be edited.</p>

- *Owner*: User who has created or been assigned to the corresponding transport requests and tasks
- *Type*: Type of ABAP objects that are assigned to the transport request or task
- *Description*: Short descriptions that have been provided
- *Last Changed*: Last changed date and time of the request and the task

Transport Organizer view allows you to perform few of the common actions irrespective of the level using the context menu option:

- Copy
- Copy Request Number
- Refresh
- Run as..

**i Note**

You will find further background information in the help documentation of the SAP GUI- based Transport Organizer.

## Related Information

[Transport Layer \[page 107\]](#)

[Transport Request \[page 108\]](#)

[Request Editor \[page 111\]](#)

[Task Editor \[page 113\]](#)

[Icons and Decorators in the Transport Organizer \[page 110\]](#)

[Filtering Transport Requests and Tasks \[page 439\]](#)

[Configuring Tree \[page 440\]](#)

[Adding Users to a Transport Request \[page 442\]](#)

[Changing the Owner of Transport Requests and Tasks \[page 443\]](#)

[Checking Consistency of Objects \[page 445\]](#)

[Releasing in the Transport Organizer \[page 446\]](#)

[Deleting in the Transport Organizer \[page 449\]](#)

## 4.20.1 Transport Layer

The transport layer defines the transport behavior of ABAP packages. Since packages are used as transport units, each individual package determines the transport behavior of all development objects contained therein – that is, whether and how a package and all its development objects are transported.

The assignment of a package to a transport layer ensures that all its development objects are connected to the Change and Transport System (CTS), thereby defining the integration and consolidation system for its development objects.

### The transport layer defines:

- Whether the development objects of a package are assigned to a local (objects are kept only locally in a development system) or transportable change request
- The SAP system in which development objects are developed and changed
- Whether these objects are transported into a discrete system sequence or – through a well-defined path – into a consolidation system.

### Related Information

[ABAP Packages \[page 38\]](#)

[ABAP Development Objects \[page 19\]](#)

[Transport Organizer \[page 104\]](#)

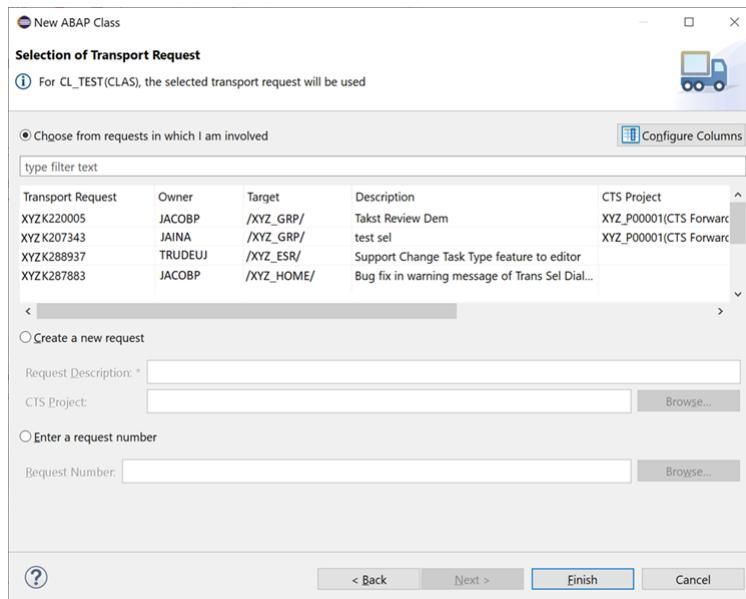
[Transport Request \[page 108\]](#)

## 4.20.2 Transport Request

A transport request collects development objects and categories for export to your local computer or to an SAP transport system.

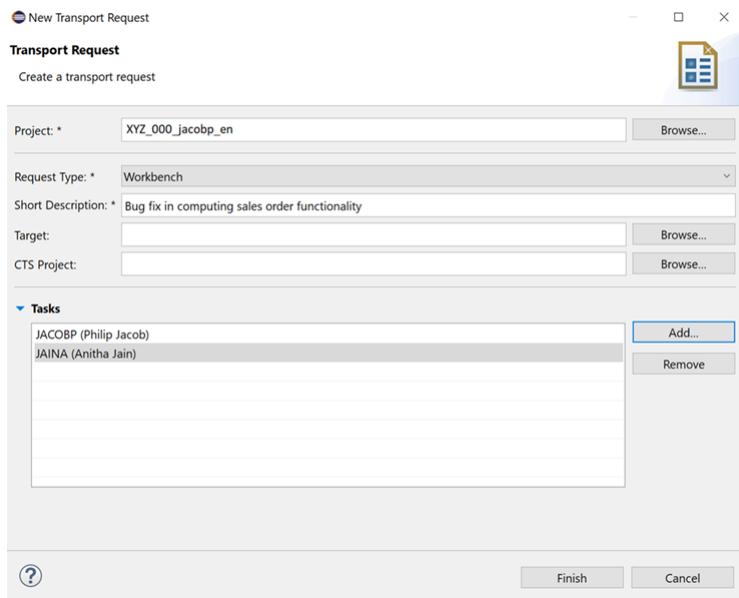
When you create a development object (ABAP repository object) in a non-temporary ABAP package, you must always assign the object to a transport request.

Dialog for Selection of Transport Request



You can create transport request from the transport organizer view. For more information, see [Creating a Transport Request \[page 438\]](#).

You can configure the table columns, add or remove the desired column(s) in the transport selection dialog using [Configure Columns](#).



Dialog for creating a transport request

Changes to development objects from non-temporary packages are organized in tasks and take place in transport requests. A transport request records all changes to development objects. The assignment of a package to a transport layer ensures that the package is connected to the Change and Transport System, thereby defining the integration and consolidation system for its development objects.

With regard to the transport behavior of development objects, we have to distinguish between these three cases:

- **Transportable objects** – development objects from packages that can be transported within a system landscape.  
Transportable packages are assigned to both a software component and a transport layer. So-called "HOME objects" represent a special case. They can be transported within a system landscape, but are not designated for external delivery (customer delivery). Packages of this type are assigned to a transport layer and the specific software component `HOME`. All changes to transportable development objects **are recorded in transport requests**.
- **Local objects** – development objects from local packages (`TEST_...`, `$TMP...`) Local packages are assigned to software component `LOCAL`, but are not assigned to a transport layer. Thus, they cannot be transported in other systems. Objects from local packages are not generally designed for productive purposes but solely for local developments, especially as part of prototyping or testing. All changes to development objects of this type **are only recorded in local transport requests**, if the recording option for the relevant package is activated.
- **Temporary objects** – special case of local objects. Temporary objects belong to temporary packages, which are, in turn, a subset of local packages. Names of temporary packages always start with the `$` sign (example: `$TMP`). Changes to temporary packages **are NOT recorded in transport requests**.

## Related Information

[Transport Layer \[page 107\]](#)

[Transport Organizer \[page 104\]](#)

[ABAP Packages \[page 38\]](#)  
[Software Component \[page 99\]](#)  
[Creating Development Objects \[page 156\]](#)  
[Request Editor \[page 111\]](#)  
[Task Editor \[page 113\]](#)

### 4.20.3 Icons and Decorators in the Transport Organizer

The following decorators are used to display transport requests, tasks, and objects in the *Transport Organizer*:

Deco-rator	Description
	Transport requests that can be edited.
	Unreleased tasks
	Released tasks
	Protected transport request
	<b>i Note</b> Only the transport request owner can add users to this type of request.
	Objects that cannot be edited in ABAP Development Tools.
	<b>i Note</b> Objects that can be edited are displayed with the same icon as they are represented in the <i>Source Library</i> folder.
	Reassign Task
	Change Owner
	Open Request or Task
	New Transport Request
	Run Consistency Checks

### Related Information

[Filtering Transport Requests and Tasks \[page 439\]](#)  
[Configuring Tree \[page 440\]](#)  
[Adding Users to a Transport Request \[page 442\]](#)

[Changing the Owner of Transport Requests and Tasks \[page 443\]](#)

[Checking Consistency of Objects \[page 445\]](#)

[Releasing in the Transport Organizer \[page 446\]](#)

[Deleting in the Transport Organizer \[page 449\]](#)

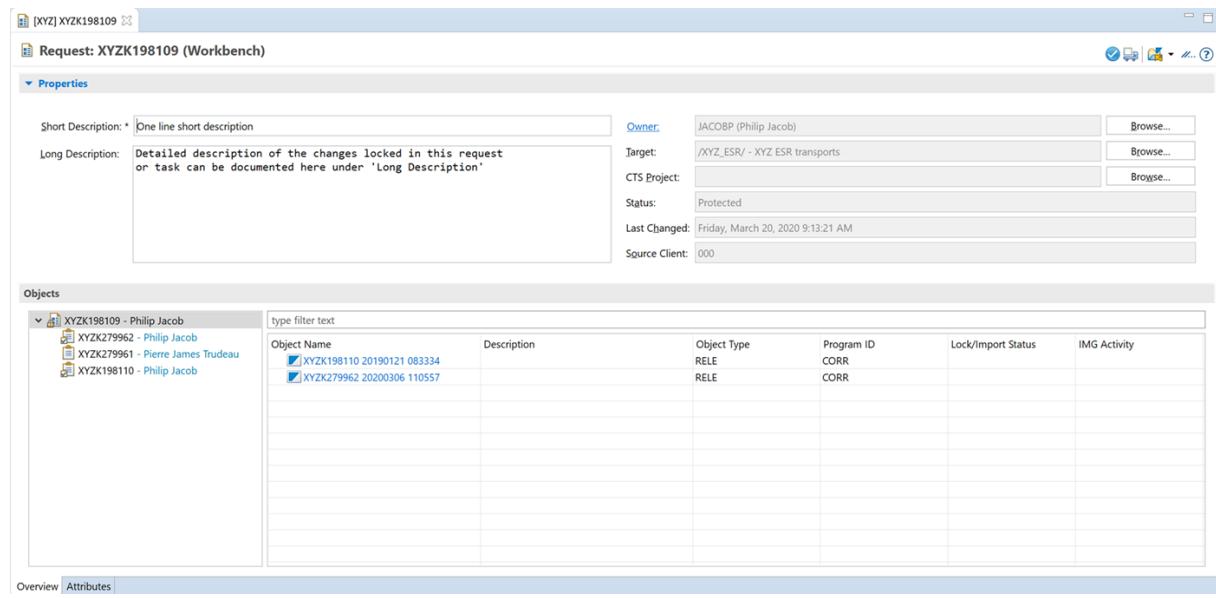
[Transport Request \[page 108\]](#)

[Transport Organizer \[page 104\]](#)

## 4.20.4 Request Editor

The Request Editor allows you to manage the content of an individual change request. Access the Request Editor by double-clicking a request in the tree structure of the change requests from the Transport Organizer View.

This feature is supported as of Application Server ABAP 7.53 SP00.



Object Name	Description	Object Type	Program ID	Lock/import Status	IMG Activity
XYZK198110 20190121 083334		RELE	CORR		
XYZK279962 20200306 110557		RELE	CORR		

The Transport Request Editor has the following information:

## Overview

- *Properties:*
  - Short Description: This field contains a one-line, short description of the request.
  - Long Description: Detailed description of the request can be documented here.
  - Owner: You can change the owner of a request or task only if it has not been released. A owner must be a user recognized by the system. Only users with the authorization object S\_CTS\_PROJECT can change the owners of projects.

- Target: The transport target specifies where a transport request is imported.
- CTS Project: Name of a project in the Change and Transport System. You can organize requests and tasks in the Change and Transport System by assigning them to a project. This project assignment can apply across systems, helping you to group requests when you import them.
- Status: Status of a request
  - Modifiable: The request or task can be edited by the owner.
  - Protected: The request can be edited by the owner, however only the owner of the request can create new tasks.
  - Release started: The export is running or was terminated. If exports terminate, release the request or task again.
  - Released: The request or task is released and can no longer be changed.
  - Last Changed: Gives the last changed time stamp.
- Source Client: The source client of a request. All client-specific objects in this request originate from this client.
- **Objects** : The left pane contains the list of tasks under a request and the right pane shows the objects locked under each task or the request. *All objects* displays released and modifiable ABAP objects under the request.

## Attributes

This tab allows you to edit and modify the attributes which define the change request. They are mainly used for analysis of a request.

## Transport Logs

This tab allows you to view the logs of a released Transport Request.

This tab is available only for the Transport Requests that are released.

## Related Information

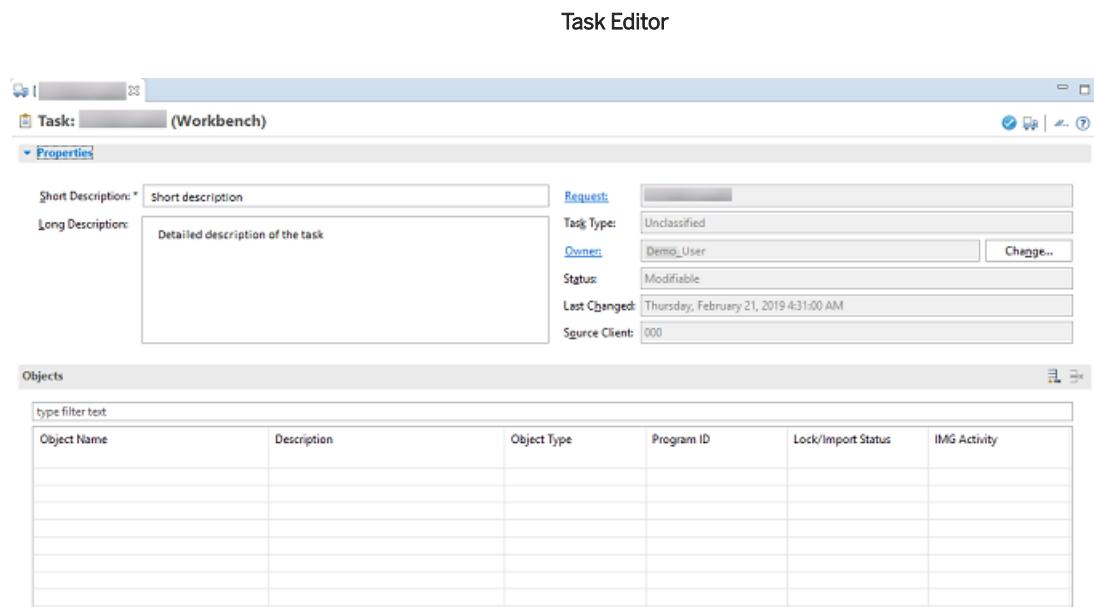
[Transport Organizer \[page 104\]](#)

[Task Editor \[page 113\]](#)

[Opening a Request or Task Editor \[page 453\]](#)

## 4.20.5 Task Editor

The Task Editor allows you to manage the content of an individual task under a modifiable request. Access the Task Editor by double-clicking a task in the tree structure of the change requests from the Transport Organizer View.



The Task Editor has sections similar to the Request Editor, you can modify the properties and documentation of a task here.

Unlike Request Editor, the Task Editor does not include the attributes section as attributes are specific to a change request.

## Related Information

[Transport Organizer \[page 104\]](#)

[Request Editor \[page 111\]](#)

[Opening a Request or Task Editor \[page 453\]](#)

## 4.21 Accessibility Features in ADT

Accessibility at SAP refers to the possibility for everyone, including and especially people with disabilities, to access and use technology and information products.

ABAP Development Tools (ADT) supports following possibilities to utilize accessibility features:

- By using **Tools** like screen readers that read text from the user interface and transfer it into audible voice. So, you can hear the context of the current position.

- By setting specific **window configurations** and **Eclipse preferences** to adjust font size or contrast of text to the background. So, you can improve readability.

#### Note

ADT is integrated into the Eclipse application framework. General information about the accessibility functionalities in Eclipse can be found here:

- [Accessibility features in Eclipse](#)
- [Accessibility](#)

SAP ensures the accessibility of the products through the [Product Standard Accessibility](#).

For more information how to use and set accessibility features in ADT see the Related Information below.

## Related Information

[Enabling Accessibility Features in ADT \[page 716\]](#)

[Setting Configurations and Preferences \[page 717\]](#)

[Using Screen Readers \[page 720\]](#)

[Keyboard Shortcuts for ABAP Development \[page 747\]](#)

## 4.22 Authorization Fields and Objects

You can use authorization fields and objects to enable authorization checks.

To enable authorization checks in your services, access control lists and Core Data Services, you need to define authorization fields and bundle them in authorization objects. The values in the authorization fields are then tested in authorization checks.

After you have completed the settings in your authorization objects, you can use them in services and define default authorization values. When you create an external app, these default values are added automatically from the service the app is based on.

## Example

The authorization field *Product Type* belongs to the authorization object *Product*. You can define allowed activities, such as *Display* or *Release* for this authorization object and classify them as *Read*, *Write* or *Value Help*.

## Related Information

[Working with Authorization Objects and Fields \[page 493\]](#)

[Defining Authorization Fields \[page 493\]](#)

[Defining Authorization Objects \[page 494\]](#)

[Maintaining Authorization Default Values \[page 495\]](#)

[Creating an App \[page 124\]](#)

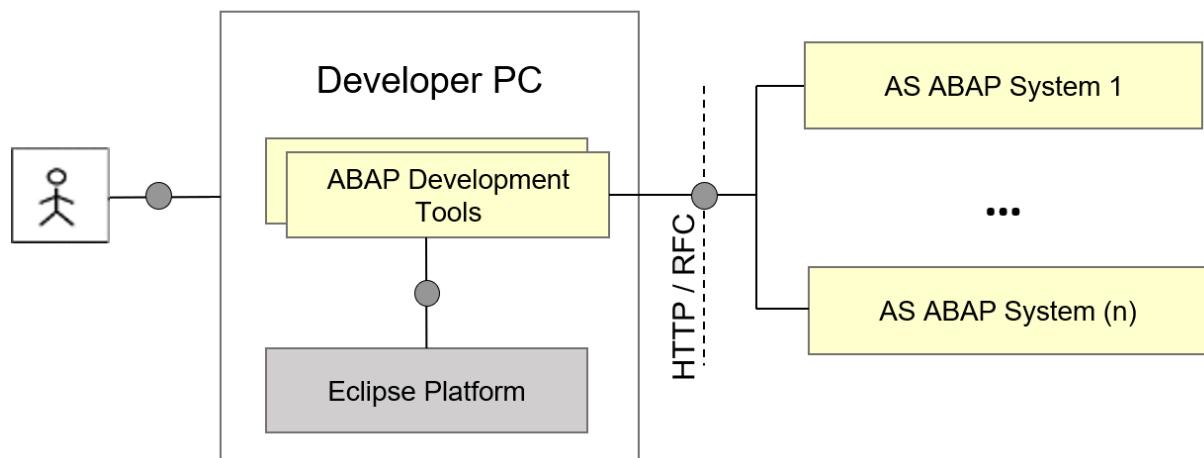
# 5 Tasks

## 5.1 Fundamental Tasks and Tools

### Context

The general idea of ABAP Development Tools (ADT) is to provide one Eclipse client that is installed on the developer's PC and can connect to several ABAP back-end systems of different releases. The connection between the client and the back end is achieved through the HTTP/RFC protocol.

The development paradigm is still server-based. This means that the development objects are stored solely in the ABAP repository of the back end (no local copies via check-in and check-out), and services such as syntax check, search, and where-used run on the back end.



Architecture view of the new integrated development environment for ABAP

The client part of ABAP Development Tools (ADT) provides a set of standard development tools such as the [Project Explorer](#) for system browsing and a variety of creation wizards, search, and other views. In addition, there are specific editors available for ABAP development object types, such as ABAP classes.

The ADT provides the well-known ABAP capabilities (transport, activation, version management, where-used list, and others) combined with an Eclipse-based state-of-the-art UI and user interaction capabilities in a new integrated development environment.

ADT integrates Eclipse-based front-end tools using native Eclipse technology.

### Related Information

[Working with ABAP Projects \[page 117\]](#)

[Working with Development Objects \[page 155\]](#)  
[Searching in ABAP Projects \[page 291\]](#)  
[Editing ABAP Source Code \[page 294\]](#)  
[Getting Feeds \[page 487\]](#)  
[Accessing ABAP Keyword Documentation \[page 461\]](#)  
[Working with Bookmarks \[page 491\]](#)

## 5.1.1 Working with ABAP Projects

### Context

ABAP projects form the **starting point** for the ABAP developer's work in the Eclipse-based IDE. An ABAP project mediates between an ABAP backend system and the Eclipse-based IDE, and it provides a framework for the creation, processing, and testing of development objects.

In this part of the documentation you will find detailed descriptions of the elementary activities in the area of ABAP projects. These will be important to you, particularly during the introductory phase.

### Related Information

[ABAP Projects \[page 56\]](#)  
[Adding a Favorite Package \[page 137\]](#)  
[Browsing Development Objects in the Project Explorer \[page 128\]](#)  
[Closing ABAP Projects \[page 131\]](#)  
[Deleting ABAP Projects \[page 131\]](#)

## 5.1.1.1 Working with ABAP Cloud Projects

ABAP cloud projects form the starting point for the ABAP developer's work in the Eclipse-based IDE in the context of SAP Cloud Platform ABAP Environment.

In this part of the documentation, you will find detailed descriptions of the elementary activities in the area of ABAP cloud projects. These will be important to you, particularly during the introductory phase.

### Related Information

[ABAP Projects \[page 56\]](#)

## 5.1.1.1.1 Creating an ABAP Cloud Project

An ABAP cloud project mediates between the ABAP back end for developing ABAP applications in the context of ABAP Environment and ADT. Projects like these provide a framework that enables you to create, process, and test development objects for SAP Cloud Platform products.

Before you can start using ABAP Development Tools (ADT), you first need to create an ABAP cloud project which represents the system connection to your ABAP back-end.

### Prerequisites

Before you can create an ABAP cloud project, ensure that you are

- member of the relevant space in the Cloud Foundry environment.
- assigned the developer role **or** you have a service key in JSON format.

#### Note

For general information about the enablement of ABAP environment, see [Getting Started with a Customer Account: Workflow in the ABAP Environment](#).

For basic information about the enablement of ABAP Development Tools in the context of ABAP Environment, see [Enabling SAP Cloud Platform ABAP Environment for ADT \[page 8\]](#).

For more information about ABAP development in the context of ABAP environment, see [ABAP Development](#).

For any questions regarding your company-specific access and administration details to the SAP Cloud Platform, contact your SAP administrator.

### Overview

You can create ABAP cloud projects in the following ways:

- [Using a Service Key Provided by the Service Instance \[page 119\]](#) if you want to check the access credentials using a service instance. The service key is then provided by the SAP Cloud Platform Cloud Foundry Environment instance.
- [Using an Existing Service Key \[page 121\]](#) if you already have a service key in JSON format and want to copy and paste or import it directly.

## 5.1.1.1.1.1 Using a Service Key Provided by the Service Instance

A service instance describes a single instantiation of a service running on SAP Cloud Platform Cloud Foundry. The service key enables you to generate the access credentials using ADT.

### Prerequisites

- You have access to and a user account for the relevant space in SAP Cloud Platform Cloud Foundry.

### Context

You want to connect your Eclipse-based IDE to the ABAP system using the service key provided by the service instance.

### Procedure

1. From the menu, choose   

The first page of the *New ABAP Cloud Project* creation wizard is opened.

2. Choose the *SAP Cloud Platform Cloud Foundry Environment* radio button.

The *Connection Settings* wizard page is opened. Here you define the connection to SAP Cloud Platform Cloud Foundry Environment.

3. To select the server from which you want to generate the service key, choose the *Region* from the drop-down list box.

#### Note

The region defines the server location where the service instance of the SAP Cloud Platform Cloud Foundry Environment is provided. If you use a region, the URL in the *API endpoint* input field is set accordingly.

If want to add your individual region, choose *Other* from the drop-down list box. You can then add the URL of your own API endpoint.

4. To access the global account in SAP Cloud Platform Cloud Foundry Environment, enter your *User Name* or *Email* as well as your *Password*.
5. Choose *Next*.

The *Connection Settings* wizard page is opened. Here you define the connection details to the service instance.

6. To select the service instance, choose the relevant entries from the following drop-down list boxes:

- *Organization* to select the development account of your company
  - *Space* to select the logical development unit of your company
  - *Service Instance* to select the relevant service instance that represents an ABAP system
7. Choose *Next*.

The next *System Connection* wizard page is opened.

In the *Service Instance Logon* section, an integrated browser is opened.

The content displayed in the browser is configured by the relevant administrator of your company. It displays the logon screen by default.

8. To log on the service instance, you have the following options:

- Using the integrated browser: Enter the configured *Email* address and *Password* to identify yourself in the relevant input fields.

**i Note**

Authentication is performed in the integrated browser, which means Single Sign On is **not** supported. Also, tools such as password managers are **not** supported for this logon option.

- Using the default browser: Choose the *Log on with Browser* button.

**i Note**

When you select this button, the same logon screen as in the integrated browser is opened in your default browser. Proceed as described above.

- Using another browser: Choose the *Copy Logon URL* button.

When you select this button, the URL is then copied to the clipboard of your computer.

9. To get access, choose the *Log On* button.

The *Service Instance Connection* wizard page is opened.

Here, the following connection settings are displayed:

- **Service Instance URL**: URL of the server instance where the ABAP system runs
- **Email**: ID of the user who is authorized in the configured identity provider to access the ABAP system
- **User ID**: ID of the user who is assigned to the e-mail
- **System ID**: Name of the ABAP system
- **Language**: Abbreviation of the logon language

**i Note**

To adapt the logon language, enter the relevant abbreviation.

- **Client**: ID of the logon client

10. Choose *Next*.

The *Project Name and Favorite Packages* wizard page is opened.

11. [Optional:] If you want to change the project name, enter a new *Project Name*.

**i Note**

When you create the project, it is added to the `ZLOCAL` ABAP package by default. This package contains all local objects from all users of the ABAP system. The containing objects cannot be transported into other systems.

If you want to add further ABAP packages, choose [Add...](#) and enter the name of the package in the corresponding input field. Keep in mind that this package must exist in the system.

If you want to group your projects, you can add working sets. To do this, choose [New](#) in the [Working Sets](#) section and select the relevant projects.

12. To create the ABAP cloud project, choose **Finish**.

## Results

The ABAP cloud project is created and added in the [Project Explorer](#).

To check the results of the procedure, expand the first level of the project structure.

Verify that the project structure already includes the following nodes:

- [Favorite Packages](#) contains the local packages and the packages which you have added to your favorites list.
- [Released Objects](#) contains all development objects from SAP that are available to be used.

### 5.1.1.1.2 Using an Existing Service Key

A service key is used to connect to a service instance using ABAP Development Tools (ADT).

## Prerequisites

- You have access to and a user account for the relevant space in SAP Cloud Platform Cloud Foundry.
- You have stored the service key in JSON format in a text file or copied it to the clipboard.

## Context

You want to connect your Eclipse-based IDE to the ABAP system using a service key in JSON format.

### Note

In case of any questions or problems, contact the administrator of your company.

## Procedure

1. From the menu, choose    The first page of the *New ABAP Cloud Project* creation wizard is opened.
2. Choose the *Service Key* radio button and choose *Next*.  
The *System Connection* wizard page is opened.
3. Paste the service key from the clipboard into the *Service Key* text box **or** choose the *Import...* button below the text box to import a text file that contains the service key.  
The service key is added into the text box.
4. Choose *Next*.  
The *Logon to the SAP Cloud System* wizard page is opened.
5. Choose one of the following buttons to log on to the ABAP back-end using a browser:
  - **Open Logon Page in Browser** to open the default browser that is defined for your operating system.  
As a result, the default browser is opened and displays the logon page.
  - **Copy Logon URL to Clipboard** to copy the URL and store it in the clipboard. Open the desired browser and paste the URL into the address line. After you have confirmed, the logon page will be opened.
6. To log on the service instance and to identify yourself, enter the configured *Email* address and *Password* in the relevant input fields. Choose then the *Log On* button.

### Note

Authentication is performed in the browser.

7. Close the browser.

The *Service Instance Connection* wizard page is opened.

Here, the following connection settings are displayed:

- **Service Instance URL:** URL of the service instance
- **Email:** ID of the user who is authorized in the configured identity provider for accessing the service instance
- **User ID:** ID of the user who is assigned to the E-mail
- **System ID:** Name of the ABAP system
- **Language:** Abbreviation of the logon language

### Note

To adapt the logon language, enter the relevant abbreviation.

- **Client:** ID of the logon client

8. Choose *Next*.

The *Project Name and Favorite Packages* wizard page is opened.

9. [Optional:] If you want to change the project name, enter a new *Project Name*.

### **i** Note

After project creation, the ABAP package `ZLOCAL` is added to the new project by default. This package contains all local objects from all users of the ABAP system. These objects cannot be transported into other systems.

If you want to add further ABAP packages, choose *Add...* and enter the name of the package in the corresponding input field. Keep in mind that this package must be available in the system.

If you want to group your projects, you can add working sets. To do this, choose *New* in the *Working Sets* section and select the relevant projects.

10. To create the ABAP cloud project, choose **Finish**.

## Results

The ABAP cloud project is created and added in the *Project Explorer*.

To check the results of the procedure, expand the first level of the project structure.

Verify that the project structure already includes the following nodes:

- *Favorite Packages* contains the local packages and the packages which you have added to your favorites list.
- *Released Objects* contains all development objects from SAP that are available to be used.

## Related Information

[Creating Service Keys](#)

### 5.1.1.2 Consuming Services in a UI

## Context

To consume services in an SAP Fiori app, you need to define an external app and assign the services to the app. This external app represents the back end of an application. It ensures that you can define the necessary authorizations and enables your business user to consume these services in the SAP Fiori app.

You first create the app itself and as a next step you can create business catalogs and business role templates you can use as a basis for creating business roles.

You need to complete the following steps:

## Procedure

1. [Creating an App \[page 124\]](#)
2. [Creating a Business Catalog \[page 126\]](#)
3. [Creating a Business Role Template \[page 126\]](#)

### 5.1.1.1.2.1 Creating an App

## Context

To create an app, proceed as follows:

## Procedure

1. In the *Project Explorer*, select the relevant package node.
2. Open the context menu and choose                        *Cloud Identity and Access Management*  *IAM App*   to launch the creation wizard.
3. Enter a name and a description and choose *Next*.

The following application types are available:

- **EXT - External App** (selected by default)  
An external app represents an app extension. It defines what is included in a tile (this is determined by the services). You can use the external app to bundle services, define authorizations and assign business catalogs.
- **MBC - Business Configuration App**  
A business configuration app serves as an entry point to the configuration objects provided by the different application areas or partners. The configuration objects can be adjusted to change the system behaviour.
- **IBS - Custom Inbound Service App**  
This application type is only created in the development system and is only required for authorization purposes to allow the developers to test the inbound services locally.

Please note that the application type you selected here can't be changed later on in the process.

4. Select the *Services* tab.
5. Add the required services to the app by choosing *Add*, selecting a service type, entering a service name and choosing *OK*.
6. Open the *Authorizations* tab. The authorization default data of the services are added automatically. You can change these values and define new entries if required.

The authorization objects used in the the app are displayed at the top of the *Authorizations* tab. The actual instances of the authorization objects including the corresponding fields and values are shown below.

### Activities

You need to define all possible activities you want to allow for this app, such as *Change*, *Delete*, or *Display*. If you use the app in a business role, the business role controls the allowed activities depending on the allowed restrictions and the maintained authorizations are included in the generated authorization profiles.

If you, for example, only define the *Read* activities for a certain authorization object, even if the business role grants *Write* access, the authorization object still only has *Read* access.

The *Maintenance Status* indicates if default values have been changed. The following status options are available:

- **Standard**  
The values are taken over from the services that have been added.
- **Maintained**  
One or more authorization fields have been changed after being taken over from the services (*Standard* maintenance status).
- **Manual**  
The authorization objects that the authorization fields are included in have been added to the app manually.

If you change any values that have been defined in the *Authorization Default Values* area in your service, a new instance is created and the standard instance is set to inactive. This means it is not used anymore.

You need to define all possible activities you want to allow for this app, such as *Change*, *Delete*, or *Display*. If you use the app in a business role, the business role controls the allowed activities depending on the allowed restrictions and the maintained authorizations are included in the generated authorization profiles.

The following further functions are available:

- **Synchronize**  
Allows to synchronize the standard values with authorization default values.
- **Copy**  
Copies an authorization instance you can modify with new values.
- **Delete**  
Deletes an authorization instance.
- **Deactivate/Activate**  
Activates or deactivates an authorization instance. Inactive instances are ignored.

7. Save your entries.

## 5.1.1.1.2.2 Creating a Business Catalog

### Context

To create a business catalog, proceed as follows:

### Procedure

1. Select the relevant package node in the *Project Explorer*.
2. Open the context menu and choose   *Other ABAP Repository Object*  to launch the creation wizard.
3. Select  *Cloud Identity and Access Management*  *Business Catalog*  and choose *Next*.
4. Enter a name and a description and choose *Next*.
5. Select the *Apps* tab.
6. Add the required apps to the business catalog.
7. If you want to test your app directly in the development system, you need to publish it locally to make sure that the required authorizations for consuming the service have been generated.

### Results

Now you can use this business catalog in your business roles.

## 5.1.1.1.2.3 Creating a Business Role Template

Create a business role template that you can use as a basis for creating new business roles.

### Context

A business role template is a pre-defined set of authorizations and assigned business catalogs that you can use as a basis for creating new business roles.

To create a business role template, proceed as follows:

## Procedure

1. Select the relevant package node in the *Project Explorer*.
2. Open the context menu and choose   *Other ABAP Repository Object*  to launch the creation wizard.
3. Select   *Cloud Identity and Access Management*  *Business Role Template*  and choose *Next*.
4. Enter a name and a description. Choose *Next*.
5. Assign your template to an existing transport request or create a new one. Choose *Finish* to complete the creation process.
6. In the newly created business role template, choose *Add* in the *Business Catalogs* section.
7. Add the required business catalogs to the business role templates. You can either use business catalogs whitelisted by SAP or ones that you have defined yourself.
8. Choose *Publish Locally* to make the business role template usable. The system displays a *Progress Information* message to show you the publication status.
9. Choose *Finish*.

## Results

You can now use this business role template to create business roles. It is available in the *Business Role Templates* app.

### 5.1.1.1.3 Consuming Services in the Context of API with Technical Users

## Context

To access services from external APIs with a technical user, you need to define a communication scenario and add the required services to the scenario. You can then use this scenario and create a communication arrangement for it.

#### Note

A communication scenario bundles inbound communication design-time artifacts, provides the basis for communication between systems. Every communication arrangement must be based on a communication scenario. A communication arrangement provides the necessary metadata for configuring the services.

## Procedure

1. Select the relevant package node in the *Project Explorer*.
2. Open the context menu and choose    *Other ABAP Repository Object*  to launch the creation wizard.
3. Select  *Communication Management*  *Communication Scenario*  and choose *Next*.
4. Enter a name and a description and choose *Next*.
5. Choose *Finish* to create the transport request.
6. Select the *Inbound* tab.
7. Add the required services to the scenario and choose *Finish*.
8. If you want to test your communication scenario directly in the development system, you need to publish it locally to make sure that the required services have been generated.

## Results

Now you can create a communication arrangement based on this communication scenario.

### 5.1.1.2 Browsing Development Objects in the Project Explorer

The *Project Explorer* displays the contents of an ABAP project.

## Use

You can use the *Project Explorer* to browse through a project in order to locate, edit, or display development objects of the ABAP Repository.

## Overview

In the *Project Explorer*, the objects are structured hierarchically, basically grouped by packages and object types.

The structured listing of ABAP packages enables you to access development objects by continually expanding the tree structure.

But there are more comfortable ways to find a specific development object. You can, for example,

- Use  +  +  if you know the object name

- Use the ABAP Source Search if you know the specific coding that is used in the object to be found
- Use the Where-used functionality if you know how the objects are referenced between each other
- Navigate to a development object through pressing **F3** from its usage in the source code

When the relevant development object is found and then opened, the corresponding position within the expanded tree is highlighted and the object is opened in the active editor.

## Linking Content with the Editor

The *Link with Editor* functionality enables you to synchronize the cursor position within the *Project Explorer* in accordance to the selected object within the ABAP source code editor. This means, when opening another object from the editor, the cursor position within the *Project Explorer* is adopted in accordance. The relevant tree is then expanded.

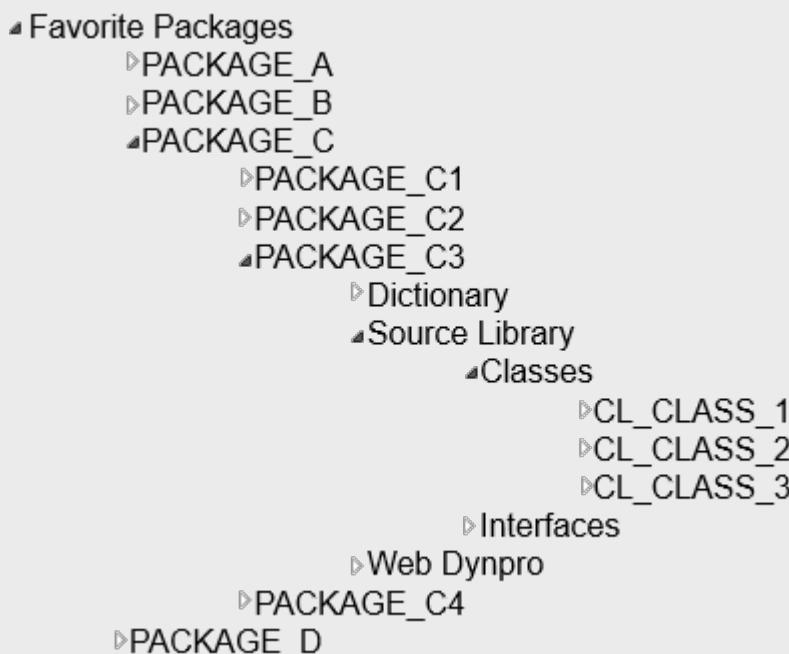
*Link with Editor* is activated by default and effects all projects. You can turn it off at every time and reactivate it again choosing the *Link with Editor* icon from the toolbar. This enables you, for example, to freeze the *Project Explorer* while navigating within source code.



Toolbar icon to activate/deactivate the Link with Editor functionality

### Example

The following example represents the content of an arbitrary ABAP project:



Containing packages and objects of an ABAP project

When using the Link with Editor functionality, ABAP Development Tools (ADT) only opens the relevant package and lists (possible) subpackages. Only the package of the relevant object is expanded with all of its objects. The relevant object is selected in the *Project Explorer*.

### • Example

You, as a developer, opened the `CL_CLASS_2` object.

As a result, ADT opens the corresponding ABAP project in the *Project Explorer* as follows.

- Favorite Packages
  - PACKAGE\_C
    - PACKAGE\_C3
      - ▷ Dictionary
      - Source Library
      - Classes
        - ▷ CL\_CLASS\_1
        - ▷ **CL\_CLASS\_2**
        - ▷ CL\_CLASS\_3
      - ▷ Interfaces
    - ▷ Web Dynpro

Display of the expanded folder with its trees and containing objects

The entire content of `PACKAGE_C3` package is listed. But only the one – where the relevant object is stored – is expanded completely. In the tree, where the relevant object is stored, all containing objects are listed. Note that only here in the example the relevant object is highlighted in bold.

### i Note

To display the hidden objects of a partly loaded folder, select the relevant one and choose *Load Full Content* from the context menu or press `F5`.

## Related Information

[ABAP Projects \[page 56\]](#)

[ABAP Repository Trees \[page 40\]](#)

[Configuring ABAP Repository Trees \[page 153\]](#)

[Searching in ABAP Projects \[page 291\]](#)

[Opening Development Objects \[page 159\]](#)

## 5.1.1.3 Closing ABAP Projects

### Context

It can happen when you have multiple projects that you do not wish to process a certain ABAP project. In this case, you could (temporarily) close this project. The consequence of this is that the corresponding project tree is not longer visible. In addition, a closed ABAP project is no longer taken into consideration for the object search.

### Procedure

1. In the *Project Explorer*, select the root node of the relevant ABAP project.
2. Open the context menu and choose **Close Project**.

### Results

The project node is still visible in the *Project Explorer* so that you are able to re-open the project whenever you need it.

You can also re-open a closed project.

### Related Information

[ABAP Projects \[page 56\]](#)

[Deleting ABAP Projects \[page 131\]](#)

## 5.1.1.4 Deleting ABAP Projects

### Context

You can delete an ABAP project if you are entirely sure that you will no longer need it.

### Procedure

1. In the *Project Explorer*, select the root node of the ABAP project in question.

2. Open the context menu and choose **Delete**.
3. In the dialog window that now appears, check the option *Delete project content on disk....*
4. Confirm with **OK**.

## Related Information

[ABAP Projects \[page 56\]](#)

[Closing ABAP Projects \[page 131\]](#)

## 5.1.2 Working with ABAP Packages

You use ABAP packages to group development objects in a self-contained unit in the ABAP repository. In addition, you define transport properties and administrative information, such as application component and the responsible user.

In this part of the documentation, you will find detailed descriptions of the elementary activities in the ABAP package area.

## Related Information

[ABAP Packages \[page 38\]](#)

[ABAP Package Editor \[page 61\]](#)

[Creating ABAP Packages \[page 132\]](#)

[Editing ABAP Packages \[page 136\]](#)

### 5.1.2.1 Creating ABAP Packages

You create an ABAP package to encapsulate ABAP repository objects in a self-contained unit. This enables you to combine several packages in a software component.

## Prerequisites

- If you create a main or a development package as a subpackage, its superpackage must already exist in the relevant system. The superpackage serves as the anchor for adding new packages to the package hierarchy.
-  Your user must be assigned to the `SAP_A4C_BC_DEV_PC` business catalog. In case of any questions, contact your administrator.

## Context

You want to create, for example, a

- *Structure package* as the root container of a new package hierarchy.
- *Main package* as a root container of one or more development packages.
- *Development package* to group development objects using the same transport layer in a subpackage of a main package.

## Procedure

To create a package, proceed as follows:

### → Tip

When editing an input field, you can use the content assist functionality by choosing **Ctrl** + **Space**.

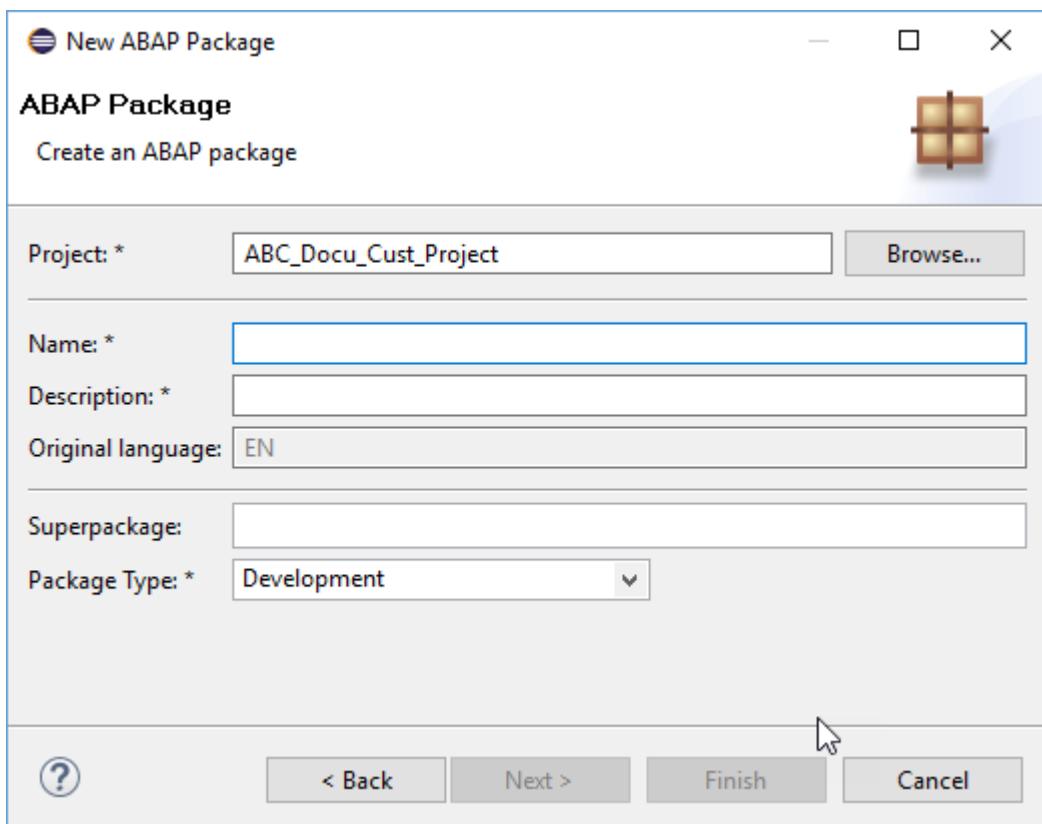
1. In the *Project Explorer*, select the relevant package node.

### i Note

To create a subpackage, navigate to the relevant superpackage node and select it.

2. Open the context menu and choose **► New ► ABAP Package**.

The creation wizard is opened.



Example of the creation wizard

3. Enter the *Name*.

#### i Note

Consider the following naming conventions for packages:

- Customer-specific objects starting with `Y` or `Z` **or** your own defined namespace
  - Customer specific namespaces are enclosed with slashes (`/<namespace>/`)
- Private test packages starting with `T` or `TEST`
  - These packages can only be assigned to the `LOCAL` software component.
- For your local demo objects starting with `/DMO/`

For detailed information about the naming conventions, see the *Related Information*.

4. Enter a meaningful *Description* short text.
5. [Optional:] To create a subpackage, enter or edit the name of the *Superpackage*.

#### i Note

When you create a subpackage, ADT automatically proposes the transport properties of the superpackage on the next wizard page.

6. To classify your package, choose one of the following entries from the *Package Type* drop-down menu:
  - `Structure` to define the highest level of package hierarchy in order to organize applications and software components.
  - `Main` to define the highest level of package hierarchy that contains a structure package.

- *Development* to define subpackages

### i Note

Your selection has influence on the:

- Packages that should not be shipped to customers must be assigned to the `HOME` software component.

7. Choose *Next*.

The *Basic Package Properties* page is opened.

8. Enter the *Software Component* that defines the set of package(s) that can only be delivered together to customers.
9. Enter the *Application Component* to define the development area to which the containing packages or development objects belong.
10. Enter the *Transport Layer* to define the transport behavior of the package.

### → Recommendation

To keep your ABAP cloud project manageable, SAP recommends you to use the transport layer which is defined for your software component. You should **not** add another transport layer.

11. [Optional:] If you work on an SAP-internal ABAP system, you can define translation of the containing text elements, such as UI texts, messages, or help texts into the required language(s). To do this, you can select one of the following entries from the *Translation Relevance* drop-down menu:
  - *Administration Tools* to provide text for system administrators in four standard language versions: English, German, French, and Japanese
  - *End User - Translation into all Supported Languages* to provide text for business end users in all supported languages
  - *End User - Reduced Translation Scope* to provide text for business end users in a set of predefined languages only. This set is defined by the translation coordinator (SLS Account Manager).
  - *Development Tools* to provide text for ABAP developers
  - *No Translation* for packages with text that does not need to be translated

### i Note

You cannot change translation relevance anymore. Consider your selection briefly.

The input field *Translation Relevance* is **SAP-internal classifications** only. This field is only displayed in systems of system type 'SAP'.

12. Assign a transport request.
13. Start the creation with *Finish*.

## Results

The back-end creates the package that is stored in the repository and opens the package in form-based editor.

In the *Project Explorer*, the new package is added to the corresponding package node. In the *Properties View*, additional package details are displayed.

If you have created, for example, an empty

- Structure package, you can now continue building your package hierarchy.
- Main package, you can now, for example, create another main package or add further main packages and development packages.
- Development package, you can now start developing and creating ABAP repository objects.

## Related Information

[ABAP Packages \[page 38\]](#)

[Adding Users to a Transport Request \[page 442\]](#)

### 5.1.2.2 Editing ABAP Packages

You can edit package properties using the package editor.

#### Prerequisites

Cloud You need the same business catalog as for creating packages.

#### Context

After creating a package for your company, you as a customer want to adopt the package-specific properties.

##### i Note

You as a customer cannot edit SAP-internal packages.

In the [Overview](#) editor tab, you can adopt the following checkboxes:

- [Package Properties](#) section
  - **Adding further objects to package not possible.** Adding further development objects to a package is blocked.

##### i Note

Existing development objects can still be moved to other packages.

- **Encapsulated** to enclose the development objects of a package. When selected, only the development elements exposed in package interfaces of the package are visible to the outside. Possible client packages need use access to those package interfaces that contain the development elements used. Otherwise errors might occur.

### i Note

If a package is encapsulated, you must define at least one package interface to provide development objects for other packages. If the package is not encapsulated, all development objects will be available for other packages.

- *Transport Properties* section
  - **Record Changes to All Objects of the Package in Transport Requests** to record changes made to the containing development objects in the *Transport Organizer*.

In the *Use Accesses* editor tab, you can add or adopt the package interfaces. To do this, add a line at the corresponding position or select the relevant entry. In the *Package Interface* column, enter the name. In the *Error Severity* column, select the relevant entry from the dropdown menu. Note that you cannot edit the package. It will be derived from the package interface.

In the *Package Interfaces* editor tab, you can only display the relevant names and descriptions.

In the *Subpackages* editor tab, you can only display the relevant names and descriptions.

To adopt the package description, open the *Properties* view and enter your new *Description*.

## Related Information

- [ABAP Packages \[page 38\]](#)
- [Transport Organizer \[page 104\]](#)
- [Creating ABAP Packages \[page 132\]](#)

### 5.1.2.3 Adding a Favorite Package

#### Prerequisites

In ABAP projects, you can only add packages (not other types of development objects) to your favorites list.

#### Context

You, as an ABAP developer, will generally work with a limited selection of ABAP packages. Therefore, it is appropriate to include those ABAP packages that are relevant for your work in your favorites list. The packages remain assigned to the ABAP project on a permanent basis.

To add a package to your favorite list, select a package in the Project Explorer. From the context menu, select *Add to Favorite Packages*.

You can also remove packages from the favorite list, using context menu option *Remove from Favorite Packages*.

## Related Information

[ABAP Projects \[page 56\]](#)

[ABAP Packages \[page 38\]](#)

### 5.1.2.3.1 Adding by Using the Favorite Packages Node

#### Context

To add a package to your favorites list using the *Favorite Packages* node, proceed as follows:

#### Procedure

1. In the *Project Explorer*, expand the first level of your ABAP project.
2. Select the *Favorite Packages* node, open the context menu and choose *Add a Package...*.
3. In the dialog box that appears, enter a valid name of a package and confirm with *OK*.

#### → Tip

When entering the name of the package, you can benefit from the content assist functionality by editing the first letters and then pressing `Ctrl` + `Space`.

## Related Information

[Adding a Favorite Package \[page 137\]](#)

[Adding from the System Library Node \[page 139\]](#)

[Deleting an ABAP Package from the Favorites List \[page 139\]](#)

## 5.1.2.3.2 Adding from the System Library Node

### Context

To add a package to your favorites list starting from the *System Library* node, proceed as follows:

### Procedure

1. In the *Project Explorer*, expand the navigation tree of the *System Library* node until you find the package you wish to work with.

→ Tip

For the package search, you can also use **CTRL** + **Shift** + **A** in connection with the activated *Link with Editor*. The node of the package found is then displayed at the corresponding position within the expanded project tree.

2. Select the package node, open the context menu, and choose **Add to Favorites**.

### Related Information

[Adding a Favorite Package \[page 137\]](#)

[Adding by Using the Favorite Packages Node \[page 138\]](#)

[Deleting an ABAP Package from the Favorites List \[page 139\]](#)

## 5.1.2.3.3 Deleting an ABAP Package from the Favorites List

### Context

To remove an individual package from your favorites list, proceed as follows:

### Procedure

1. Expand the *Favorite Packages* node.
2. Open the context menu for the package in question.

3. Choose *Remove from Favorite Packages*.

## Results

The package is removed from the favorites list but not deleted in the ABAP Repository.

## Related Information

[Adding a Favorite Package \[page 137\]](#)

[Adding from the System Library Node \[page 139\]](#)

[Deleting an ABAP Package from the Favorites List \[page 139\]](#)

## 5.1.3 Working with ABAP Repository Trees

ABAP repository trees enable you to display your favorite development objects of an ABAP repository in the *Project Explorer*.

## Related Information

[ABAP Repository Trees \[page 40\]](#)

[Creating an ABAP Repository Tree \[page 141\]](#)

[Deriving an ABAP Repository Tree from an ABAP Repository Folder \[page 144\]](#)

[Adapting an ABAP Repository Tree \[page 148\]](#)

[Duplicating ABAP Repository Trees \[page 150\]](#)

[Displaying the Distribution of an ABAP Repository Tree in a Chart \[page 152\]](#)

## 5.1.3.1 Creating an ABAP Repository Tree

You create an ABAP repository tree to display a defined selection of development objects in the *Project Explorer*.

### Context

You have the following possibilities to create a tree by:

- Defining your own object selection
- Using a predefined template from SAP
- Deriving a new tree from an existing tree by duplicating it

The following procedure describes the possibility to create a tree on base of a template.

For further information about the other possibilities, see the Related Information section from below.

### Procedure

1. In the *Project Explorer*, choose  **ABAP Repository Tree...** from the context menu of the relevant ABAP project where you want to create a tree.

The *ABAP Repository Tree Configuration* wizard is opened.

2. Choose a template

ABAP Development Tools (ADT) provides templates that contain a predefined filter set and structure.

The following templates are provided:

Name	Predefined Object Selection (Filters)	Predefined Tree Structure	Description
Blanc	-	-	This template is empty. It has no restrictions and predefined structure.
System Library	-	ABAP packages and object type	You need to rename it and define the object selection and tree structure.
			This template contains all packages of the entire ABAP system. It does not contain any filters.
			It is structured by ABAP packages and object type.

Name	Predefined Object Selection (Filters)	Predefined Tree Structure	Description
Application Component Hierarchy	-	Application components, packages, and object type	<p>This template contains all objects of the entire system.</p> <p>It is structured by application component, package and object type.</p>
Package Collection	Package	Packages and object type	<p>This template contains a collection of packages.</p> <p>It is structured by object type.</p> <p>This can be used to collect some packages in a tree, which you need to work on a specific task.</p>
My Objects	User	ABAP packages and object type	<p>This template filters for all development objects of the currently logged on user where he/she is assigned to and/or responsible for.</p> <p>It is structured for all ABAP packages of the entire system, sorted by object type.</p>
Released Objects	API	ABAP packages and object type	<p>This template filters for all APIs those release state is set on Released.</p> <p>It is structured for all ABAP packages of the entire system, sorted by object type.</p>
Deprecated Objects	API	API state and object type	<p>This template filters for all APIs those release state is set on Deprecated.</p>

Name	Predefined Object Selection (Filters)	Predefined Tree Structure	Description
Core Data Services	Object type	Object type and ABAP packages	<p>This template filters all development objects in the context of Core Data Services (CDS) such as data definitions, access controls, metadata extensions, and CDS entities.</p> <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> <b>i Note</b>            This template is just an example. You can define such a setup also for other object types.         </div>
Exception Classes	Object type and object name pattern	ABAP packages	<p>It is structured for development objects of the CDS context, sorted by ABAP packages of the entire system.</p> <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> <b>i Note</b>            This template filters all exception classes (starting with the CX_ prefix).         </div>

3. To continue, choose *Next*.

The second wizard page is opened and displays the filter possibilities in accordance to your selection.

4. [Optional:] If you want to adopt a template or if you have selected the *Blank* template, you have the following possibilities:

For further information, see [Adjusting the Name, Property Filter, and Structure of an ABAP Repository Tree \[page 145\]](#).

- In the *Tree Presentation* area, enter the name of the tree to be created.
  - In the *Object Selection* area, restrict the number of displayed development objects manually.
  - In the *Tree Structure (based on Properties)* area, you can edit the structure of a tree.
5. To trigger creation, choose *Finish*.

## Results

The tree is added to the selected ABAP project in the *Project Browser*. From here, you can, for example, collapse the tree and navigate to the relevant development object.

The *Link with Editor* functionality is enabled by default.

## Related Information

[Deriving an ABAP Repository Tree from an ABAP Repository Folder \[page 144\]](#)

[Adjusting the Name, Property Filter, and Structure of an ABAP Repository Tree \[page 145\]](#)

[Adapting an ABAP Repository Tree \[page 148\]](#)

[Duplicating ABAP Repository Trees \[page 150\]](#)

[Getting Support from the Content Assist \[page 295\]](#)

[Setting the API Release State \[page 465\]](#)

### 5.1.3.2 Deriving an ABAP Repository Tree from an ABAP Repository Folder

In addition to duplicating ABAP repository trees, you can also create an ABAP repository tree based on an existing configuration from an ABAP repository folder. This enables you to create a new tree based on a subtree.

## Procedure

1. In the *Project Explorer*, select the ABAP repository folder from which you want to derive the configuration and choose *Derive New Tree...* from the context menu.  
The *Derive Tree Configuration* wizard is opened and displays the existing object selection and tree structure.
2. Adapt the tree configuration according your development needs.  
For more information how to adopt the selection, see the Related Information below.
3. To trigger deriving, choose *Finish*.

## Results

The new tree is created and added to the *Project Explorer*.

If you do not change the name in the configuration wizard, the same name as from the derived folder is used.

The *Link with Editor* functionality is enabled by default.

You can now, for example, collapse the tree to navigate to the relevant development object.

## Related Information

[Adjusting the Name, Property Filter, and Structure of an ABAP Repository Tree \[page 145\]](#)

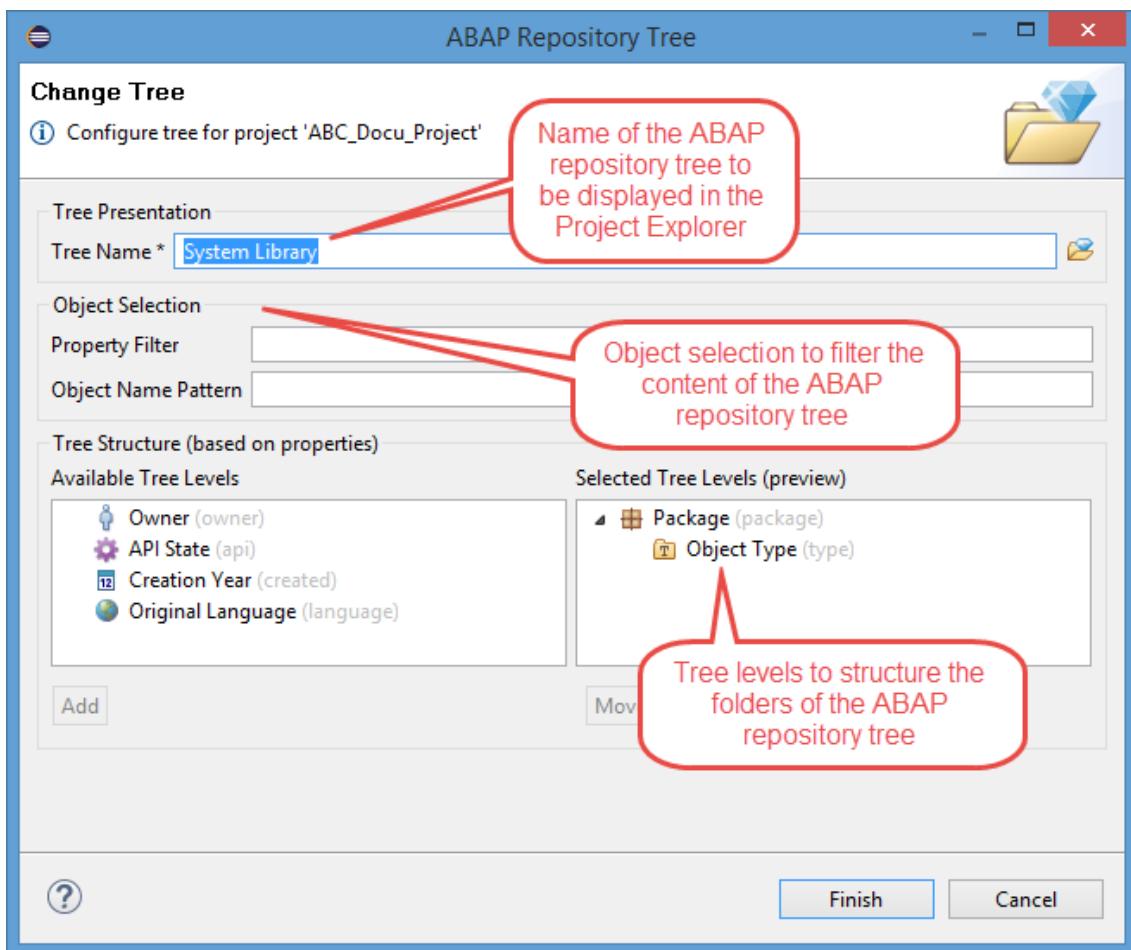
### 5.1.3.3 Adjusting the Name, Property Filter, and Structure of an ABAP Repository Tree

You can adjust the name, property filters, and structure of an ABAP Repository Tree when you, for example, create or adapt ABAP repository trees.

#### Procedure

1. Open the relevant tree configuration wizard.

The *ABAP Repository Tree Configuration* wizard is opened and displays the existing object selection.



Example of the wizard to configure a ABAP repository tree

2. In the *Tree Presentation* area, you can overwrite the existing name. To do this, select the existing name and enter the new one.
3. In the *Object Selection* area, you can restrict the number of displayed development objects manually by entering the
  - o *Property Filter* on base of the following tree levels from the *Tree Structure* area to filter for:

Property	Technical Key	Filter Example	Description
Owner	owner	owner:testuser	User who has created or is responsible for a development object

Property	Technical Key	Filter Example	Description
Object Type	type	type:clas,inf,stru,dtab	Object type of the development objects  stru and dtab are examples for alias names to differentiate between database tables and structures.  Alias types are used for non-unique transport types.
Package	package	package:basis	ABAP package where the development objects are assigned to
API State	api	api:released	Release state that describes the availability of a development objects
Creation Year	created	created:2016	Year in which a development object was created
Original Language	language	language:en	Original language in which a development object has been created

### Example

You can limit your filter, for example, as follows:

- type:clas: for ABAP classes
- package:basis: for ABAP objects that are saved in the ABAP package "basis"
- type:clas,tabl package:basis: for ABAP classes and database tables that are saved in the ABAP package "basis"

You can also combine filters, such as `api:released type:clas,owner:test package:basis`.

### Note

Use the content assist functionality (`Ctrl + Space`) to add predefined filter criteria. Then, the completion list is opened from where you can select the relevant filter criteria.

- *Object Name Pattern* to enter and limit for object names or name patterns.

### Note

You can use the

- Complete name of a development object
- Prefix of its name

- Search pattern using the following wildcards:
  - "\*" for any string
  - "?" for any character
  - Ending with "<" to suppress automatic prefix matching

#### • Example

Example for possible name patterns: \*abc?test<, c1\*

4. In the *Tree Structure (based on Properties)* area, you can adopt the structure of a tree. To do this, drag and drop the relevant tree level from the *Available Tree Levels* section to the *Selected Tree Levels (preview)* section.

#### i Note

The hierarchy of a tree level is created in accordance to the way you drag and drop them.

In addition, you have the following possibilities:

- To modify the hierarchy of the tree levels, select the relevant tree level in the *Selected Tree Levels (preview)* section and choose the *Move Up* or *Move Down* button.
- To eliminate a criteria from the selection, select the relevant tree level in the *Selected Tree Levels (preview)* section and choose the *Remove* button.
- To supplement your selection without using drag and drop, select the relevant tree level in the *Available Tree Levels* section and choose the *Add* button.
- To replace an existing tree level, drag and drop the new tree level on the one to be replaced.
- You can use multiselection. The hierarchy of the tree levels is considered in accordance to the order you have selected them.

In the *Selected Tree Levels (preview)* section, the hierarchy of the tree levels is automatically adopted.

## Related Information

[Adapting an ABAP Repository Tree \[page 148\]](#)

### 5.1.3.4 Adapting an ABAP Repository Tree

You can adapt an existing ABAP repository tree to adjust it to your current development needs.

## Procedure

1. In the *Project Explorer*, choose *Configure ABAP Repository Tree...* from the context menu of the relevant tree you want to adapt.

The *ABAP Repository Tree Configuration* wizard is opened and displays the existing object selection.

2. In the *Object Selection* area, restrict the number of displayed development objects manually.

For more information how to adapt the selection, see the Related Information section from below.

3. To add your adaptions, choose *Finish*.

## Results

The tree is updated and the relevant content is displayed in the *Project Browser*.

## Related Information

[Adjusting the Name, Property Filter, and Structure of an ABAP Repository Tree \[page 145\]](#)

### 5.1.3.5 Expanding an ABAP Repository Tree

You configure how to display the structure of the development objects beneath the ABAP project in the *Project Explorer*.

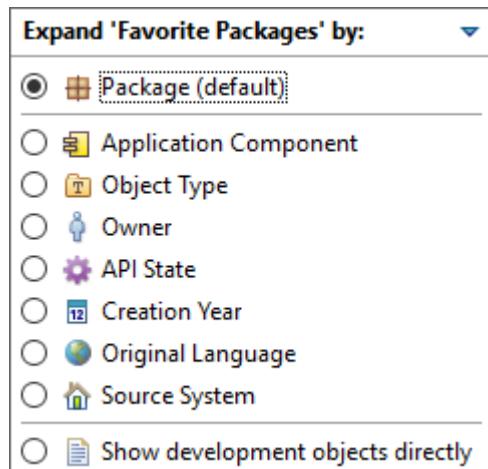
## Context

You want to sort the development objects of the repository tree.

## Procedure

1. Select one of the first levels directly under the ABAP project and choose *Expand Tree by...* from the context menu.

The selection dialog is opened.



2. Choose the relevant radio button.

You can choose one of the following radio buttons to sort the development objects in the relevant folders as follows:

- **Owner (default):** Name of the users who have created the development objects
- **Application Component:** Application development of the ABAP package where the development objects belong
- **Package:** ABAP package where the objects are grouped in a self-contained development unit
- **Object Type:** Types of development object, such as `clas` for ABAP classes
- **API State:** State whether a development object is intended for public usage or for internal SAP usage
- **Creation Year:** Year when the development objects were created
- **Original Language:** Language in which the development objects were originally created
- **Source System:** Name of the ABAP system to which the development objects belong
- **Show development objects directly:** To show an alphabetical sorted plain list with all development objects

## Results

In the *Project Explorer*, the relevant sublevel is reloaded and adopted in accordance to your selection. Depending on the number of development objects, creating and sorting might take a few seconds.

You can now navigate to the relevant development object manually or using `Ctrl` + `Shift` + `A` to open it.

## Related Information

[Displaying the Distribution of an ABAP Repository Tree in a Chart \[page 152\]](#)

### 5.1.3.6 Duplicating ABAP Repository Trees

You can create an ABAP repository tree by duplicating an existing one and copying its configuration.

## Procedure

1. In the *Project Explorer*, select the tree that you want to duplicate and choose *Duplicate ABAP Repository Tree...* from the context menu.

The *ABAP Repository Tree Configuration* wizard is opened and displays the existing object selection and tree structure.

2. Adopt the tree configuration according your development needs.

For more information how to adopt the selection, see the Related Information below.

3. To trigger duplication, choose *Finish*.

## Results

The tree is added to the selected ABAP project in the *Project Browser*. The number 2 is added to the existing name. Alternatively, you can rename the duplicated tree.

The *Link with Editor* functionality is enabled by default.

You can now, for example, collapse the tree to navigate to the relevant development object.

## Related Information

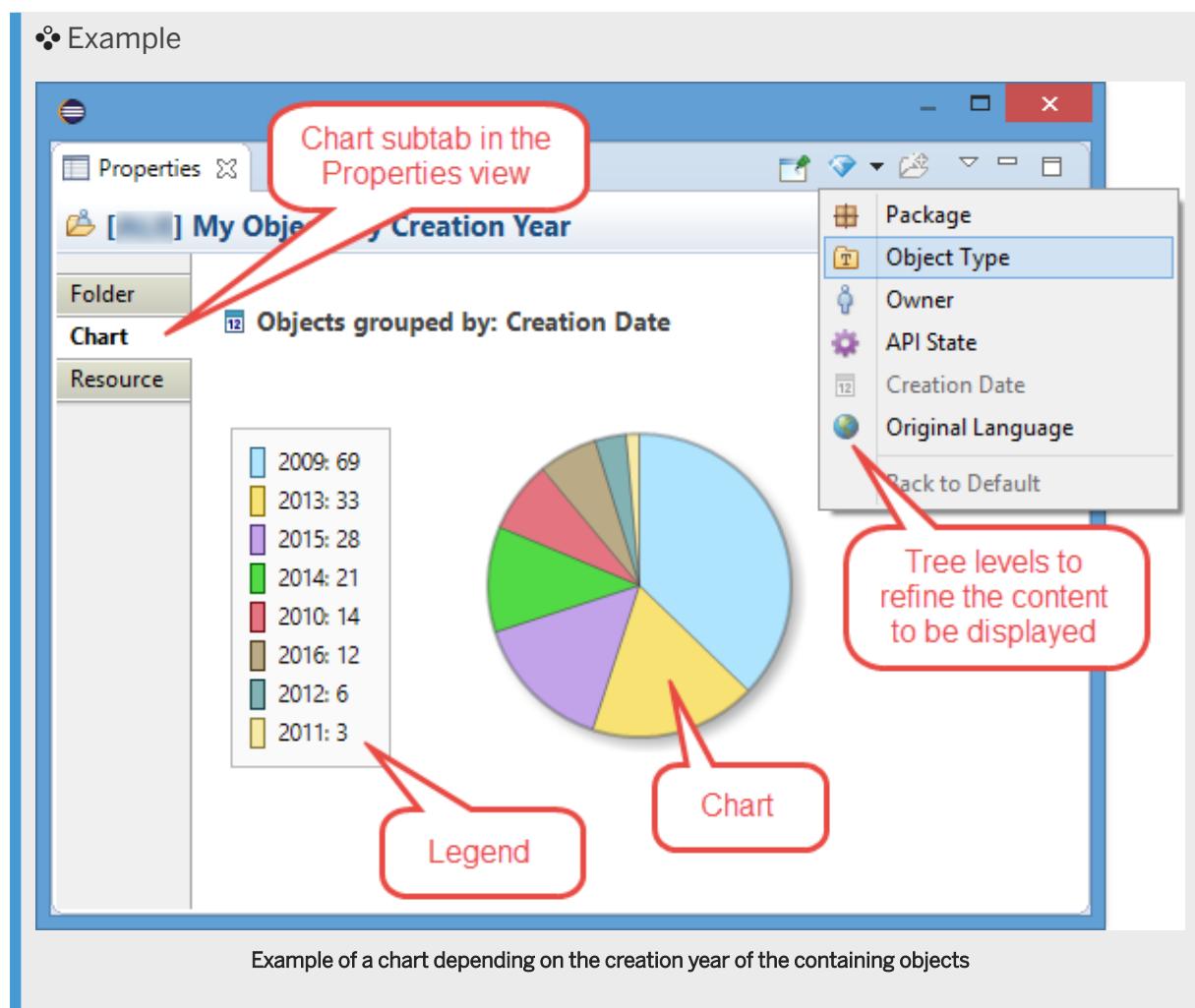
[Adjusting the Name, Property Filter, and Structure of an ABAP Repository Tree \[page 145\]](#)

### 5.1.3.7 Displaying the Distribution of an ABAP Repository Tree in a Chart

You can visualize the apportionment of the content from an ABAP repository tree in a diagram.

#### Context

To get a visual overview of the content from a tree, proceed as follows:



#### Procedure

1. In the *Project Explorer*, select the tree for which you want to generate the chart.
2. In the *Properties* view, select the *Chart* subtab.

## Results

The chart is displayed in the *Chart* subtab.



Group object by button from the toolbar of the Properties view

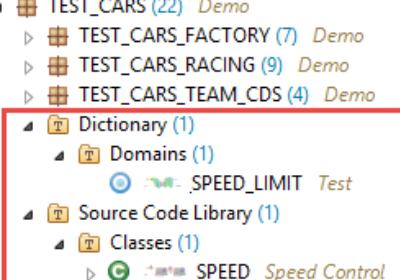
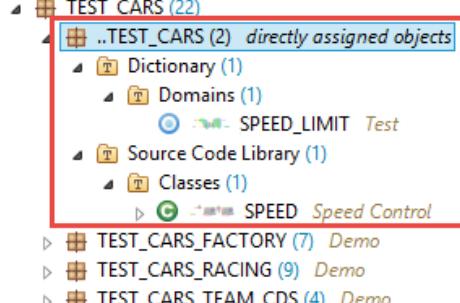
To redefine your selection for another tree level, choose the relevant entry through the *Group objects by* button from the toolbar.

### 5.1.3.8 Configuring ABAP Repository Trees

You can customize the preferences of ABAP repository trees in order to define their behavior for all ABAP projects of your Eclipse-based IDE.

To define the   preferences, the following options are provided:

Checkbox / Input field	Description
Display short descriptions	<p>Checkbox to hide/display the relevant short description after each development object (as well as its subobjects) and ABAP package in italic.</p> <p>To predefined its color, choose the <i>Change color...</i> link and define the <i>Decoration color</i>.</p> <p>For more information, see the Related Information below.</p>
Display object counters	<p>Checkbox to hide/display the number of the containing objects in brackets after each tree and folder label</p> <p>To predefined its color, choose the <i>Change color...</i> link and define the <i>Counter color</i>.</p> <p>For more information, see the Related Information below.</p>
Collect directly assigned objects in extra .. package	<p>Checkbox to display development objects – that are not assigned to a subpackage – in an artificial subpackage node.</p> <p>When selected, ADT will create an artificial subpackage node under the selected package and collect all directly assigned development objects in this subpackage node. Its folder name is derived from the selected package plus two dots as prefix.</p> <p>When deselecting the checkbox, the subpackage is dissolved and all objects are no more collected in it.</p>

Checkbox / Input field	Description
	<p>Example without selection:</p>  <p>The development objects are directly displayed in the selected parent package (TEST_CARS).</p>
	<p>Example with selection:</p>  <p>The previously unassigned development objects are now assigned to an artificial subpackage node (..TEST_CARS).</p>

## Related Information

[Changing the Font Color of Texts \[page 310\]](#)

[Browsing Development Objects in the Project Explorer \[page 128\]](#)

### 5.1.3.9 Working with Favorite Objects

#### Adding Objects to Favorite Objects Tree

You can add objects that you work with frequently to the *Favorite Objects* tree for quick access.

There are different options to add an object:

- **From a development object**

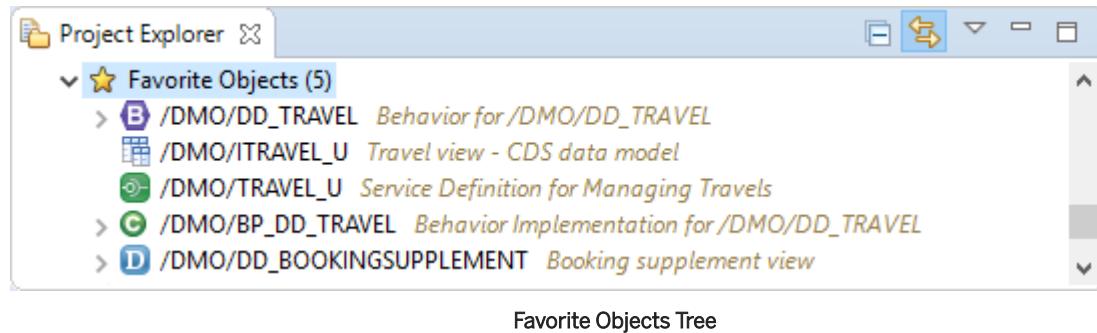
Select one or more objects in a view, for example, in the Project Explorer, in the [Search View \[page 293\]](#), or in the [Relation Explorer \[page 36\]](#). Open the context menu and select [Add to Favorite Objects](#).

- **From the Favorite Objects tree**

1. In the Project Explorer, select the *Favorite Objects* tree.
2. Open the context menu and select [Add object...](#).
3. In the dialog, select one or more objects you want to add.
4. Select **OK**.

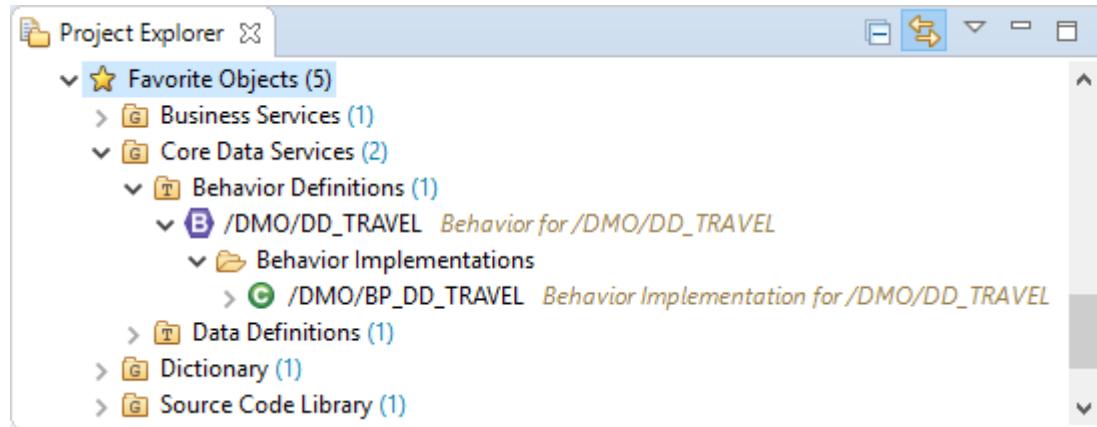
## Configuring Favorite Objects Tree

After adding objects to your favorites, you can see them in the *Favorite Objects* tree.



Favorite Objects Tree

You can [configure the tree \[page 153\]](#) to group the objects, for example, by type.



Favorite Objects Grouped by Object Type

## Removing Objects from Favorite Objects Tree

You can remove objects from the *Favorite Objects* tree.

To do so, select one or more objects in a view, for example, in the *Project Explorer*, *Relation Explorer*, or *Search* view. From the context menu, select [Remove from Favorite Objects](#).

### 5.1.4 Working with Development Objects

#### Context

In the new Eclipse-based IDE, packages form the basic organizational units for ABAP projects. Each of the package nodes can, in turn, contain subpackages or ABAP development objects as subnodes. The development objects are the individual parts that are used to build an ABAP application.

In this part of the documentation you will find detailed descriptions of the elementary activities concerning the ABAP development objects as such.

## Related Information

[ABAP Development Objects \[page 19\]](#)  
[Status of a Development Object \[page 20\]](#)  
[Opening Development Objects \[page 159\]](#)  
[Opening in Another System \[page 163\]](#)  
[Checking ABAP Syntax \[page 176\]](#)  
[Adding Code Templates to an ABAP Exception Class \[page 190\]](#)  
[Adding Local Objects of Other Users to Favorites \[page 165\]](#)  
[Copying and Duplicating Development Objects \[page 171\]](#)  
[Linking to Open Development Objects in Another ADT Installation \[page 167\]](#)  
[Displaying Properties of Development Objects \[page 174\]](#)  
[Checking ABAP Syntax \[page 176\]](#)  
[Activating Development Objects \[page 178\]](#)  
[Deleting Development Objects \[page 183\]](#)

### 5.1.4.1 Creating Development Objects

#### Context

Using the ABAP project as a starting point, you can create any arbitrary development object for the ABAP Repository. The procedures to create development objects depend on the object type to be created.

To create a development object, choose one of the following options:

- **Create an Object from the Main Menu**
  1. From the main menu, select  **File**  **New**. Alternatively, press **Alt** + **Shift** + **N**.
  2. In the menu list, select the object you want to create. If the object is not in the list, select *Other...*
- **Create an Object from the Tool Bar**
  1. From the tool bar, select .
  2. Select the type of object you want to create or type it in the input field.
- **Create an Object Using a Shortcut**
  1. Press **Ctrl** + **N**.
  2. Select the type of object you want to create or type it in the input field.
- **Create an Object from the Project Explorer**
  1. In the Project Explorer, select a node.

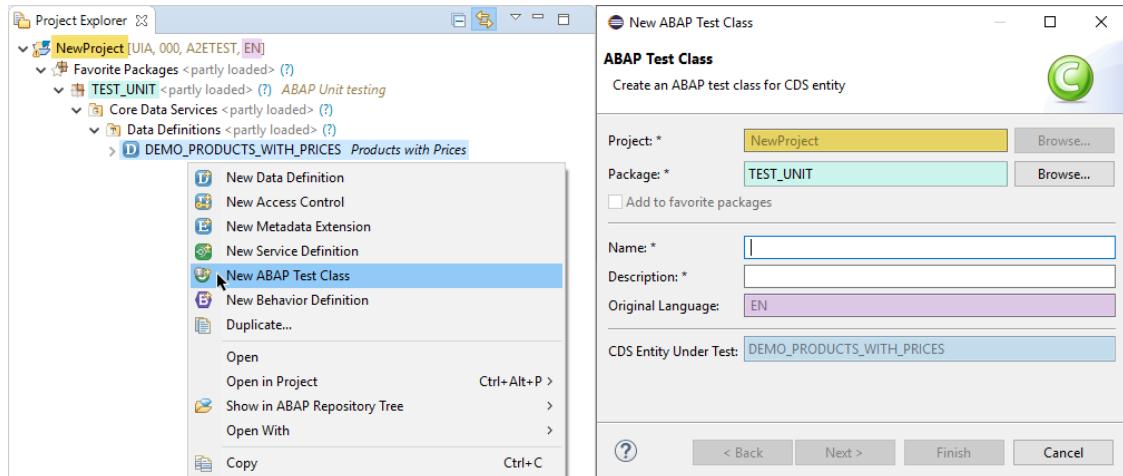
#### Note

The deeper the selected node, the less information you have to fill in later.

2. From the context menu, select an entry, for example, [New ABAP Test Class](#) or  **New**  **Other ABAP Repository Object**, if the object is not in the list.

The context menu and the information you enter in the input fields of the creation wizard depend on the location of the selection.

In the following example, the selected node is a DDL source. In the wizard, the name of the project, the package, the original language, and the CDS entity to test are already filled in.

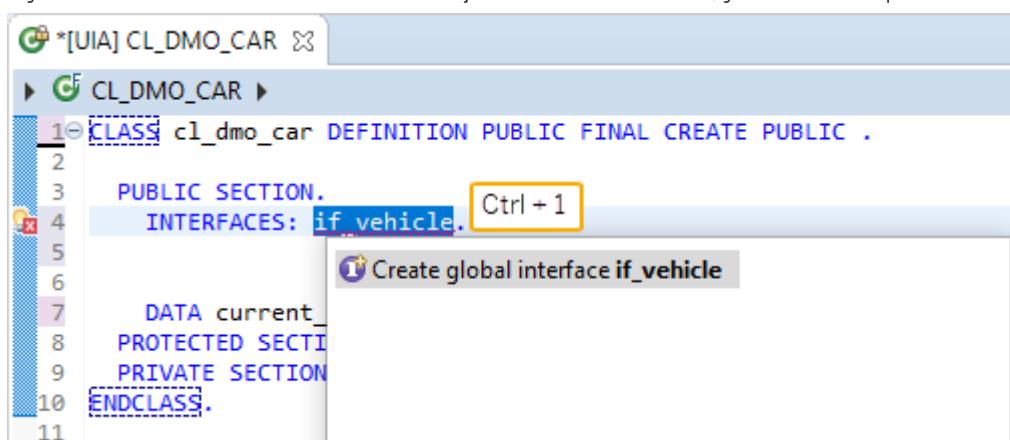


Creating an Object from the Project Explorer

- **Create an Object from Usage**

Source Code Editors

If you write source code and refer to an object that does not exist, you can use a quick fix to create it.



Creating Objects from Usage in the Source Code Editor

Form Editor

If you edit an object in the form editor and refer to an object that does not exist, you can use a quick fix to create it by clicking the label left from the input field.

The screenshot shows the SAP ABAP Development Tools (ADT) interface for creating a data element. The title bar says '\*[UIA] DMO\_COLOR'. The main area is titled 'Data Element: DMO\_COLOR' with an error message 'No domain or data type was defined' in a red box. The 'Data Type Information' section on the left has a 'Category' dropdown set to 'Domain' and a 'Type Name:' input field. The 'Field Labels' section on the right has four input fields for 'Short', 'Medium', 'Long', and 'Heading' with length settings of 10, 20, 40, and 55 respectively. A yellow box and cursor are on the 'Type Name:' field.

#### Creating Objects from Usage in the Form Editor

- **Create an Object by Duplicating or Copying the Object.**

You can create new objects by copying existing objects. Select *Duplicate* or *Copy* from the context menu on the object in the Project Explorer.

## Related Information

[ABAP Development Objects \[page 19\]](#)

[Creating ABAP Classes \[page 185\]](#)

[Creating an ABAP Function Group \[page 198\]](#)

[Creating an ABAP Function Module \[page 202\]](#)

[Creating an ABAP Function Group Include \[page 199\]](#)

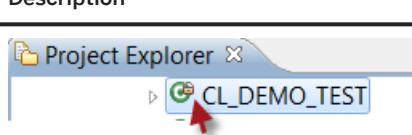
[Creating a Message Class \[page 275\]](#)

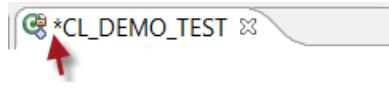
## 5.1.4.2 Unlocking Development Objects

### Context

As soon as you start to edit a development object with ABAP Development Tools (ADT), the corresponding editor is locked so that contents cannot simultaneously be changed by other users.

The status is displayed as "locked", with a decorator, both in the *Project Explorer* and on the editor tab:

UI	Description
Indication in the <i>Project Explorer</i>	

UI	Description
Indication in the ABAP source code editor	

This kind of editor lock is lifted implicitly for the respective object as soon as you have closed the editor.

In certain use cases, however, it can happen that you would like to explicitly remove the existing lock in the editor for the object concerned in order to allow changes by another user for a short time, for example.

## Procedure

1. Navigate to the relevant editor.
2. Open the context menu of the editor and choose *Unlock*.

Alternatively, you can use the toolbar   from the menu or choose the *Unlock* function in the *Project Explorer*.

## Results

The locked decorator is removed from the editor tab and the corresponding item in the *Project Explorer*.

## Related Information

[Status of a Development Object \[page 20\]](#)

[Opening Development Objects \[page 159\]](#)

### 5.1.4.3 Opening Development Objects

You have the following possibilities to open a development object:

#### Starting from the Project Explorer

1. Open the context menu on the corresponding node.
2. Choose *Open*.

## Using the Open ABAP Development Object Dialog

1. Choose  *Navigate*  *Open ABAP Development Object ...*  from the menu bar or **Ctrl** + **Shift** + **A**.

For more information, look here ['Open ABAP Development Object' Dialog \[page 160\]](#)

## Starting from any Editor

You have the following standard navigation capabilities to open a development object from an editor:

- From a usage, press **F3** or **Ctrl** + **click** on the left mouse button on the name of the corresponding development object.
- Choose the  icon from the toolbar.
- In a form-based editor, select the link on a label.

## Starting from any View

You can open a development object from a view, that is displayed, for example, from a test or search result list.

1. Double-click the relevant item from the result list or a view.

## Further Possibilities

If you have already opened a specific development object, you can also open it:

- [In another ABAP project \(system\) \[page 163\]](#)

## Related Information

[Linking to Open Development Objects in Another ADT Installation \[page 167\]](#)

[Searching Development Objects \[page 291\]](#)

### 5.1.4.3.1 'Open ABAP Development Object' Dialog

To open the ABAP development object dialog, select:

- **Ctrl** + **Shift** + **A**

-  [Navigate](#)  [Open ABAP Development Object...](#)  from the menu bar

In the dialog, enter the name of a development object or its prefix in the search field.

## Advanced Search Options

- Using Wildcards
  - "\*" for any string
  - "?" for any character
  - Ending with "<" to suppress automatic prefix matching
- Using Filter Criteria
  - Filter criteria allow you to restrict the result to a certain object type, ABAP package, user name, and so on.
  - You can use the content assist functionality ( + ) to limit your search for:
    - Types, packages, and users
    - API states
    -  Additional filter criteria
      - You can use additional filter criteria such as application component, language, system, and favorites.

### Example

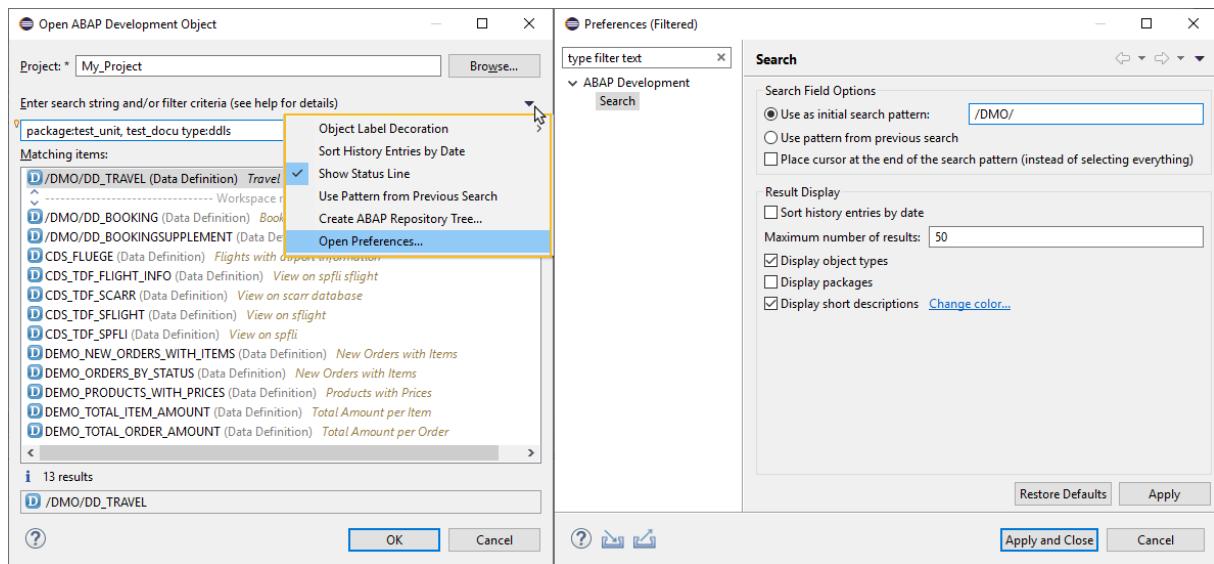
You can limit your search as follows:

- `type:clas`: for ABAP classes
- `package:basis`: for ABAP objects that are saved in the ABAP package "basis"
- `type:clas,tabl package:basis`: for ABAP classes and database tables that are saved in the ABAP package "basis"

You can combine search strings and filters, such as `*abc?test< type:clas,dtel package:basis`.

## Configuring Preferences

In the dropdown list, you can configure search preferences and how the search results are displayed. For more configuration options, select [Open Preferences...](#) from the list.



### Configuring Preferences

#### Search Field Preferences

You can predefine a default search string that is automatically entered in the search input field.

- *Use as initial search pattern* to enter an initial search pattern in the input field, for example `type: class`.
- *Use pattern from previous search* to display and reuse the last entered search string.

Additionally, you can configure whether the search pattern is selected and you can overwrite it or the cursor is placed at the end of the search pattern. Therefore, select or deselect *Place cursor at the end of the search pattern*.

#### Preferences for Displaying Results

- Sort history entries by date  
Recently opened objects are displayed in the *History* section at the top of the list. You can remove items by choosing *Remove from History* from the context menu.
- Maximum number of results
- Object label decoration  
You can define if type, package, or description of the objects are displayed.
- Show status line

#### Creating ABAP Repository Tree

Sometimes it is not sufficient to get the search results as a simple list (even if the entries are decorated with description and package).

Therefore, you can store any search permanently as *ABAP Repository Tree* that gives you the following advantages:

- Structuring of results (by object type, package, or other properties)
- Additional operations on found objects (without opening the objects in an editor), for example, execution of unit tests, duplicate, share link, etc.

To create a tree, select the [Create ABAP Repository Tree...](#) entry from the dropdown list.

## Related Information

[Opening Development Objects \[page 159\]](#)

[ABAP Repository Trees \[page 40\]](#)

[Creating an ABAP Repository Tree \[page 141\]](#)

### 5.1.4.3.2 Opening in Another System

This functionality enables you to navigate quickly between two versions of the same development object in two different projects.

#### Prerequisites

The same development object has to exist in the system in which you want to open it.

#### Procedure

1. You can initiate the functionalities from the following positions:
  - Select the relevant node in the *Project Explorer*.
  - Open the development object in the *ABAP source code editor*.
2. Open the context menu and choose [Open in Project](#).

#### → Tip

Alternatively, you can use the `Ctrl` + `Alt` + `P` shortcut to open the submenu with the project list directly.

3. Select the project in which you want to open the development object.

#### i Note

If the project does not exist, you can create a new one by selecting [New Project...](#)

Alternatively, you can also use the `Ctrl` + `Alt` + `P` shortcut. ABAP Development Tools (ADT) then automatically opens the development object in the project you opened previously. Thus, you can easily toggle between both projects.

## Results

In the editor, the development object of the other system is opened.

## Related Information

[ABAP Development Objects \[page 19\]](#)

[Tools for ABAP Development \[page 19\]](#)

[Comparing Source Code \[page 207\]](#)

[Opening Development Objects \[page 159\]](#)

### 5.1.4.4 Adding a Development Object to a Transport Request

You add a development object to a transport request to transfer it to another ABAP system. Then, in the target system the same object is used.

## Context

You want to create a new or to edit an existing development object. For this, you create a new or assign an existing transport request for/to it.

### i Note

If you edit a development object that is already edited by another developer, a transport request might already be assigned to this object.

## Procedure

In the creation wizard, choose one of following radio button on the *Selection of Transport Request* page:

- Select the *Choose from requests in which I am involved* to pick an already existing transport request.
- Select the *Create a new request* to generate a new transport request. In the **Request description** field, type further information.
- Select the *Enter a request number* to add your class to an existing request. Choose the *Browse...* button if you want to select a request that has already been created by a certain user.

## Related Information

[Working with Transport Organizer \[page 437\]](#)

[Working with ABAP Source Code Objects \[page 184\]](#)

[Working with ABAP Classes Assigned to Multiple Transport Requests \[page 192\]](#)

### 5.1.4.5 Adding Local Objects of Other Users to Favorites

#### Context

You can display and edit the local development objects from other users. This enables you to reuse a section of code for your own development activities.

#### Procedure

1. Select the relevant ABAP project in the *Project Explorer*.
2. Click **Add Local Objects of User...** in the context menu of the *Favorite Package*.
3. In the *Select* a user field, enter the user name for selection. Then choose **OK**.

#### → Tip

When editing the **User Name**, you can benefit from the content assist functionality by pressing **Ctrl** + **Space**.

#### Results

The last activated local objects from the chosen user are added to the *Favorite Packages*.

## Related Information

[Status of a Development Object \[page 20\]](#)

[Searching Development Objects \[page 291\]](#)

## 5.1.4.6 Sharing and Opening a Link to a Development Object

You can share a specific development object or a highlighted part of the ABAP source code with other ABAP developers if you wish to communicate a programming improvement or similar.

### Prerequisites

You need access permissions for the corresponding ABAP system and at least the read permissions for the linked development object.

### Context

ABAP Development Tools (ADT) provides the option of creating hyperlinks to development objects. The receiver can then open the corresponding source code in another Eclipse-based IDE as well as in a Web browser.

The following functions for sharing ABAP source code are available:

- [Linking to Open Development Objects in Another ADT Installation \[page 167\]](#): You create a hyperlink to open a development object directly in the referenced ABAP project within another ADT installation.
- [Linking for Displaying ABAP Source Code in a Web Browser \[page 168\]](#): You create a hyperlink to render a development object in a Web browser.
- [Linking to Selected Parts of ABAP Source Code \[page 169\]](#): You create a hyperlink to highlight a specific section of ABAP source code. The link can be opened in a Web browser or in the same ABAP system of another ADT installation where the section is marked.
- [Opening ADT Links \[page 170\]](#): You can open an ADT link, for example, from an email or from an MS Word or Excel file, by clicking it. In addition, you can also enter the URL into a dialog.

### Related Information

[Sharing Links of Selected Messages \[page 284\]](#)

[Copying and Duplicating Development Objects \[page 171\]](#)

[ABAP Development Objects \[page 19\]](#)

[Messages and Message Classes \[page 66\]](#)

## 5.1.4.6.1 Linking to Open Development Objects in Another ADT Installation

### Context

You can initiate the creation of an ADT link that, as a hyperlink, references a specific development object. The receiver can then open the referenced development object directly in ABAP Development Tools (ADT) in the corresponding ABAP project.

#### • Example

The following example displays an ADT link of a specific ABAP class:

```
adt://[ABAP system name]/sap/bc/adt/oo/classes/cl_demo_test_save
```

#### • Note

ADT links are available for all development objects except trace files, profiler settings, debugger variables, and so on.

### Procedure

1. In the *Project Explorer*, select the development object.
2. In the context menu, select the *Share Link...* entry.
3. In the *Share Link* dialog, select *ADT Link*.

You then have the following two options on how to proceed:

- a. Choose the *Email link* button to generate an email from your default email client.  
An email is then opened that contains a generated header and the hyperlink.
- b. Choose the *Copy link to clipboard* button to paste the ADT link from the clipboard into any document or editor.

The ADT link path is then saved in temporary storage. If you paste the path into your document or editor, it will be added as text.

### Related Information

[Opening ADT Links \[page 170\]](#)

[Linking for Displaying ABAP Source Code in a Web Browser \[page 168\]](#)

[Linking to Selected Parts of ABAP Source Code \[page 169\]](#)

## 5.1.4.6.2 Linking for Displaying ABAP Source Code in a Web Browser

### Context

You can initiate the creation of a hyperlink that refers to a specific development object. The receiver can then open the ABAP source code of the development object in a Web browser.

For this purpose, you can generate an email where the hyperlink path is added or paste it from the clipboard manually into any document.

#### i Note

HTTP links are available for the following development objects:

- ABAP includes
- ABAP function groups
- ABAP function group includes
- ABAP function modules
- ABAP classes
- ABAP interfaces

### Procedure

1. In the *ABAP source code editor*, open the development object and select the element or statement that you want to refer to.
2. In the context menu, select *Share Link for Selection....*  
The *Share Link* dialog is opened.
3. Select *HTTP Link* from the list and choose one of the following buttons:
  - To generate an email from your default email client, Choose *Email link*.  
An email client is opened that contains a generated header and the hyperlink.
  - To paste the link into any document, choose *Copy link to clipboard*.  
The hyperlink path is then saved in temporary storage.  
If you paste it, the path is added as text in your document or editor.

### Results

#### Example

The following example displays a hyperlink of a specific ABAP class that is opened in a specific ABAP system:

```
https://app.server:12345/sap/bc/abc/oo/classes/cl_test_class/source/main
```

### i Note

The following prerequisites apply in order to be able to open the hyperlink:

- In order to display the ABAP source code of the linked development object in HTML, you need to install a Web browser or a text editor.
- A hyperlink handler defines the application that is launched when a hyperlink is opened. This means you have to define a Web browser application where you want to open the development object.
- It might be necessary to make the pasted hyperlink click-sensitive. To do this, edit the hyperlink reference.

## Related Information

[Linking to Open Development Objects in Another ADT Installation \[page 167\]](#)

[Linking for Displaying ABAP Source Code in a Web Browser \[page 168\]](#)

[Sharing Links of Selected Messages \[page 284\]](#)

[Copying and Duplicating Development Objects \[page 171\]](#)

[ABAP Development Objects \[page 19\]](#)

[Messages and Message Classes \[page 66\]](#)

## 5.1.4.6.3 Linking to Selected Parts of ABAP Source Code

### Context

In a source code-based development object, you can initiate the creation of a hyperlink that refers to a specific element or statement. The receiver can then open the referenced message directly in ABAP Development Tools (ADT) in the corresponding ABAP project or in a Web browser.

For this purpose, you can generate an email where the hyperlink path is added or paste it from the clipboard manually into any document.

### Procedure

1. In the [ABAP source code editor](#), open the development object and select the element or statement that you want to refer to.
2. In the context menu, select the [Share Link for Selection...](#) entry.  
The [Share Link for Selection](#) dialog is opened.
3. Select [ADT Link](#) or [HTTP Link](#) from the list and choose one of the following buttons:
  - To generate an email from your default email client, choose [Email link](#).  
An email client is opened that contains a generated header and the hyperlink.

- To paste the link into any document, choose *Copy link to clipboard*.  
The hyperlink path is then saved in temporary storage.  
If you paste it, the path is added as text in your document or editor.

## Results

The hyperlink is saved in the clipboard.

### • Example

The following example displays an ADT hyperlink that includes the position details from the selection within a specific ABAP class:

```
adt://[ABAP system name]/sap/bc/abc/classes/cl_demo_test_save/source/  
main#start=3,4;end=7,53
```

### i Note

To make a hyperlink click-sensitive, the receiver needs to define the application that is launched when opening a hyperlink.

## Related Information

[Linking to Open Development Objects in Another ADT Installation \[page 167\]](#)

[Linking to Selected Parts of ABAP Source Code \[page 169\]](#)

[Sharing Links of Selected Messages \[page 284\]](#)

[Copying and Duplicating Development Objects \[page 171\]](#)

[ABAP Development Objects \[page 19\]](#)

[Messages and Message Classes \[page 66\]](#)

## 5.1.4.6.4 Opening ADT Links

You have two options for opening an ADT link: You can either click an ADT link that has been sent to you, for example, in an Email, or one that is provided, for example, in an MS Word file. You can also enter its URL into a dialog.

## Prerequisites

ADT link handling needs to be enabled in advance. To do this, open the  *Window*  *Preferences*  *General*  preferences page and select the checkbox for the *adt* scheme.

## Context

To open an ADT link through a dialog, proceed as follows:

## Procedure

1. Choose  *Navigate*  from the menu bar.  
The *Open ADT Link* dialog is opened.
2. Enter the copied URL or paste it from the clipboard.
3. Choose *OK*.

## Results

The relevant development object is opened in the corresponding editor.

## Related Information

[Linking to Open Development Objects in Another ADT Installation \[page 167\]](#)

## 5.1.4.7 Copying and Duplicating Development Objects

### Context

You can create copies of the following development objects:

- ABAP class
- ABAP interface
-  Data definitions, access controls, and so on

For this, following functionalities are provided:

- [Copying Source Development Objects \[page 172\]](#) to copy and paste the exact same entity into another ABAP package of your choice.
- [Duplicating Development Objects \[page 173\]](#) to create exact same entity in a selected ABAP package.

### Note

You can only copy and paste as well as duplicate within the same ABAP system.

## Related Information

[ABAP Development Objects \[page 19\]](#)

[Tools for ABAP Development \[page 19\]](#)

[Creating ABAP Classes \[page 185\]](#)

[Adding Local Objects of Other Users to Favorites \[page 165\]](#)

### 5.1.4.7.1 Copying Source Development Objects

#### Context

**i** Note

You cannot copy includes that are contained in ABAP programs.

#### Procedure

1. Select the relevant node for the development object in the *Project Explorer*.
2. Open the context menu and choose *Copy*.
3. Open the context menu in the package to which you want to add the new development object.
4. Choose *Paste* to create the selected development object into an package of your choice.
5. In the *Creation* wizard, enter the required data corresponding to the development object.
6. Choose *Next*.
7. Assign a transport request and start the creation process with *Finish*.

#### Results

In the *Project Explorer*, a new and inactive development object is created and opened in the corresponding editor.

**i** Note

If you copy ABAP classes or interfaces, existing references are automatically changed in the source code to the name of the new development object.

## Related Information

[Copying and Duplicating Development Objects \[page 171\]](#)

[Duplicating Development Objects \[page 173\]](#)

### 5.1.4.7.2 Duplicating Development Objects

#### Procedure

1. Select the relevant node for the development object in the *Project Explorer*.
2. Open the context menu and choose *Duplicate*.
3. In the *Creation* wizard, enter the required data corresponding to the development object.
4. Select the package where you want to add the to be created development object by choosing the *Browse...* button.
5. Choose *Next*.
6. Assign a transport request and start the creation process with *Finish*.

#### Results

In the *Source Library* folder, a new and inactive development object is created and opened in the corresponding editor.

##### i Note

If you duplicate ABAP classes or interfaces, existing references are automatically changed in the source code to the name of the new development object.

## Related Information

[Copying and Duplicating Development Objects \[page 171\]](#)

[Copying Source Development Objects \[page 172\]](#)

## 5.1.4.8 Pinning Content

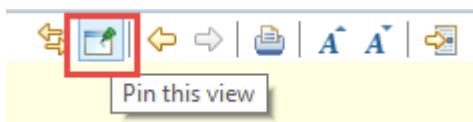
You can pin the content in the *Project Explorer*, *Properties*, *Outline*, and *ABAP Element Info* view. The content can then no longer be adapted in accordance with the current selection in the ABAP source code editor.

### Context

You pin content, for example, to arrange several views beside in order to compare their content.

### Procedure

1. Choose the *Pin this view* icon from the toolbar.



Toolbar icon to pin the content of a view

If you select another element within the source code, the content or selection of the active view remains.

2. To unlock pinning, choose the same icon again.

## 5.1.4.9 Displaying Properties of Development Objects

In the *Properties* view, you can display details, such as the owner, creation date, original language, and so on, of a source code-based development object.

### Context

#### i Note

You can change the object description of your own development objects in the *Properties* view. For this purpose, the *Description* field is provided. It is an editable field if the logon language (project language) does not differ from the original language of the object. Otherwise, this field is not editable and is labeled with a warning decorator.

## Procedure

1. Open the relevant development object in the ABAP source code editor.
2. Choose the toolbar menu  *Window*  *Show View*  *Properties* .

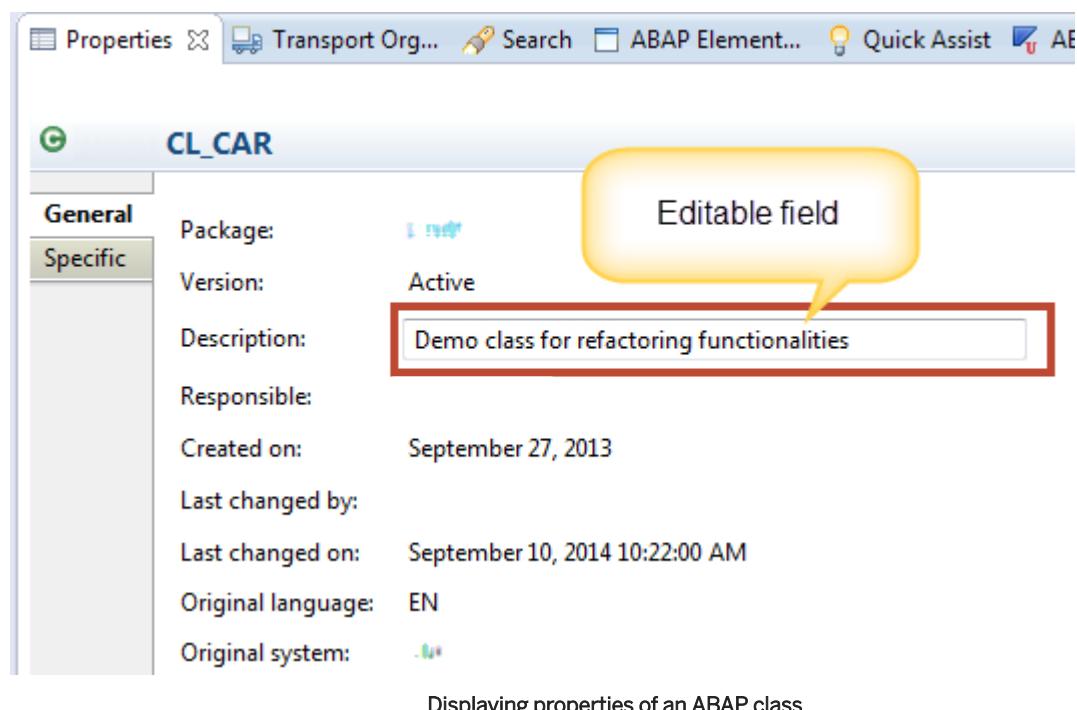
Alternatively, you can open the context menu of the editor and choose  *Show in*  *Properties* .

## Results

The properties are displayed in the *Properties* view.

Here the following subtabs are provided:

- **General:** Details that are the same for all development objects of this type
- **Specific:** Details that are individual for this type of development object



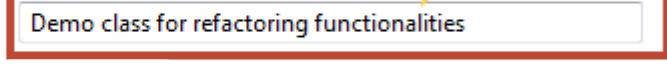
CL\_CAR

General

Specific

Package:  

Version: Active

Description:   
Demo class for refactoring functionalities

Responsible:

Created on: September 27, 2013

Last changed by:

Last changed on: September 10, 2014 10:22:00 AM

Original language: EN

Original system: 

Displaying properties of an ABAP class

### i Note

To edit the description field, put the cursor in the ABAP Editor area.

## Related Information

[ABAP Development Objects \[page 19\]](#)

[Adding a Favorite Package \[page 137\]](#)

[Browsing Development Objects in the Project Explorer \[page 128\]](#)

[Closing ABAP Projects \[page 131\]](#)

[Deleting ABAP Projects \[page 131\]](#)

[Short Texts \[page 26\]](#)

## 5.1.4.10 Checking ABAP Syntax

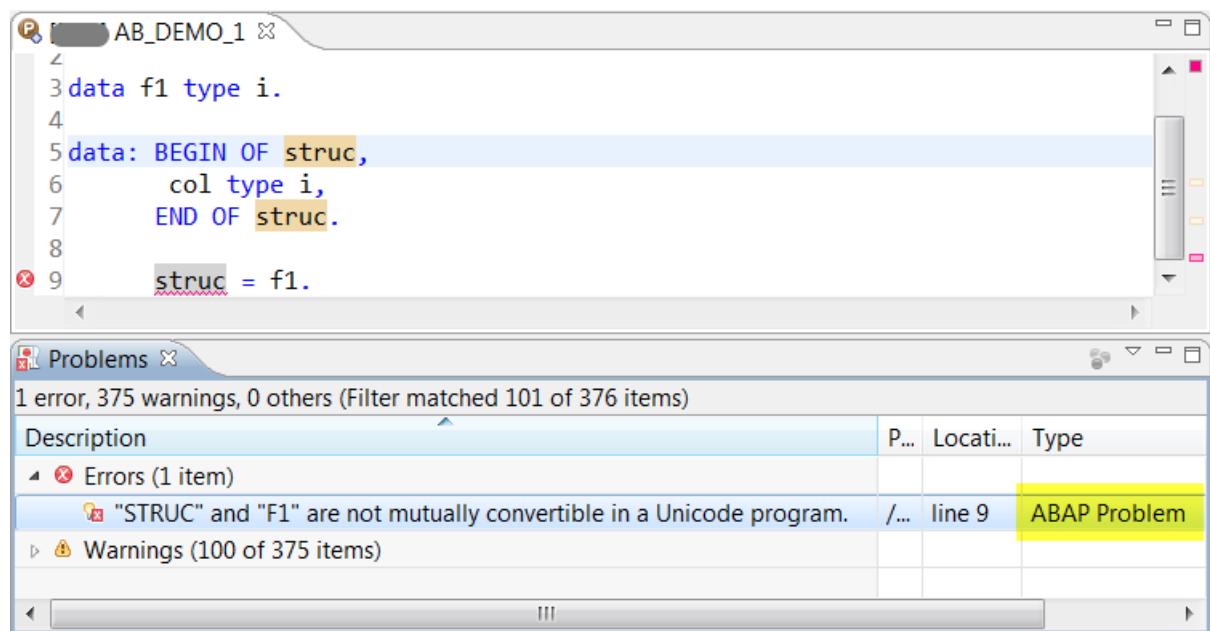
### Context

The ABAP syntax check enables you to verify code-based development objects before activating them. You can run the syntax check by:

- [Working with Manual ABAP Syntax Check \[page 177\]](#)
- [Working with Automatic ABAP Syntax Check While Programming \[page 178\]](#)

If syntax errors occur, these are issued to the *Problems* view and displayed as an ABAP Problem. In addition, the code line with the error is labeled with a decorator in the ABAP source editor.

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



Syntax errors are displayed in the Problems view

### i Note

The package check during the syntax check allows you to check whether you access objects that are visible in your object. In order to activate this functionality for an ABAP system, select the following option from the menu bar: **Project > Properties > ABAP Development > Editors > Do package check during syntax check**

## Related Information

[ABAP Development Objects \[page 19\]](#)

[Adding a Favorite Package \[page 137\]](#)

[Browsing Development Objects in the Project Explorer \[page 128\]](#)

[Closing ABAP Projects \[page 131\]](#)

[Deleting ABAP Projects \[page 131\]](#)

### 5.1.4.10.1 Working with Manual ABAP Syntax Check

#### Prerequisites

The syntax check always is executed solely for the ABAP source code in the editor currently opened.

#### Context

You can run the ABAP syntax check directly through a toolbar icon in the main window of the ABAP perspective.

#### Procedure

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

## Related Information

[Working with Automatic ABAP Syntax Check While Programming \[page 178\]](#)

[Working with Manual ABAP Syntax Check \[page 177\]](#)

## 5.1.4.10.2 Working with Automatic ABAP Syntax Check While Programming

### Prerequisites

The syntax check always is executed solely for the ABAP source code in the editor currently opened.

### Context

By default, the ABAP source code editor automatically checks the syntax of a development object you are currently working on.

#### i Note

- The automatic syntax check is executed 500 ms after you have stopped typing.
- The automatic syntax check is activated by default. Use the  [Window](#)  [Preferences](#)  [ABAP Development](#)  [Editors](#)  [Source Code Editor](#)  pages to deactivate the **Automatic syntax check on all open ABAP source editors** checkbox.
- The syntax check is always only executed for the ABAP source code of the currently opened editor. When you save a development object, all other opened source code editors are also checked.

### Related Information

[Checking ABAP Syntax \[page 176\]](#)

[Working with Manual ABAP Syntax Check \[page 177\]](#)

## 5.1.4.11 Activating Development Objects

### Context

You use this activation function to activate one or all of your inactive development object(s) in your ABAP project, any selected objects, or just subobjects of a main object (methods in classes).

#### i Note

Make sure that the development object you wish to activate is syntactically correct. Before an object is activated, the system performs a syntax check of the entire object (main program, function group, class, and so on). However, you can also activate development objects even if they contain syntax errors.

### i Note

You can run the activation of one or more development objects in the background. This enables you to continue working on other development objects while the activation process is being continued. For this purpose, select the *Run in Background* button while the activation is running.

## Procedure

1. To activate the development object that is currently opened in the ABAP source editor, click the icon  (Activate the ABAP Development Object) in the toolbar.
2. To activate a selection of development objects:
  - a. Multi-select (single selection is also possible) the relevant object's nodes in the *Project Explorer*.
  - b. Open the context menu and choose *Activate*.
3. To activate development objects using the inactive objects worklist for the ABAP project:
  - a. Open the ABAP source editor for a single development object of the project.
  - b. Open the list to the right of the activation icon  in the toolbar and choose *Activate Inactive ABAP Development Objects....*

The worklist with entries for all inactive objects appears in the *Activate inactive ABAP development objects* dialog. Here, the inactive objects are automatically sorted upwards according to the names of the transport tasks.

To sort for the name of the inactive objects, deselect the *Group object list by transport request* checkbox.
- c. In the *Worklist* window, select the relevant objects and choose *Activate*.

## Results

If no error exists, activation will be completed. Otherwise, an error message is displayed. In this case, SAP recommends that you select the **No** button to solve the error in the *Problems* view. If you select the **Yes** button, the development objects will be activated with errors.

## Related Information

[Troubleshooting for Activation Errors in the Context of ABAP Dictionary Objects \[page 181\]](#)

[Troubleshooting for ABAP Syntax Errors and Warnings \[page 180\]](#)

[ABAP Development Objects \[page 19\]](#)

[Adding a Favorite Package \[page 137\]](#)

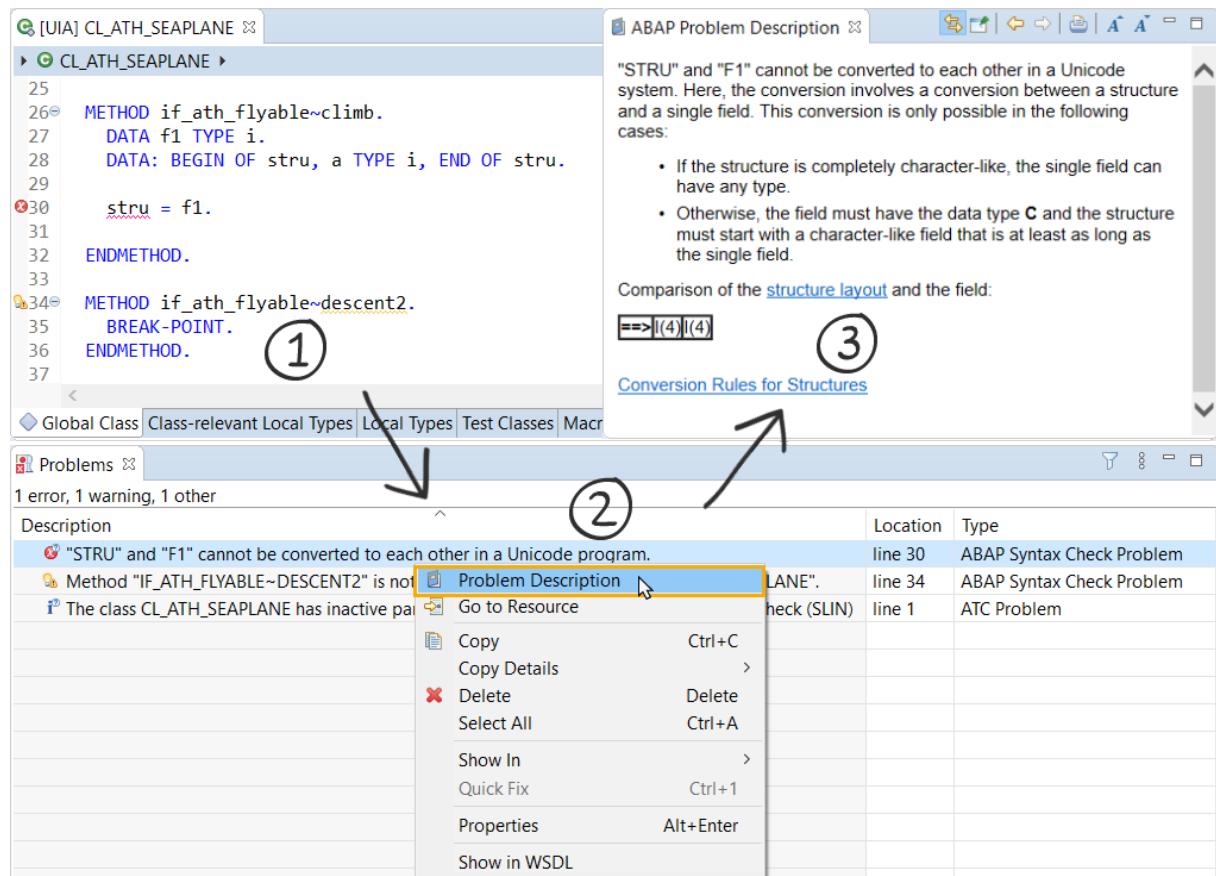
[Browsing Development Objects in the Project Explorer \[page 128\]](#)

[Closing ABAP Projects \[page 131\]](#)

## 5.1.4.11.1 Troubleshooting for ABAP Syntax Errors and Warnings

In case of activation errors, it can be of advantage for you, to get more information about the diagnosis and troubleshooting.

Whenever a long-text help for a ABAP syntax error is available in the corresponding ABAP back-end system, you can access it in the *Problems* view. To do so, open the context menu for the respective error item in the *Problems* view and choose the option **Show ABAP Problem Help**. The long-text help is then displayed in the *ABAP Problems Documentation* view.



A long-text help for a syntax error is available in the system, you can access it using the context menu in the Problems view.

### Related Information

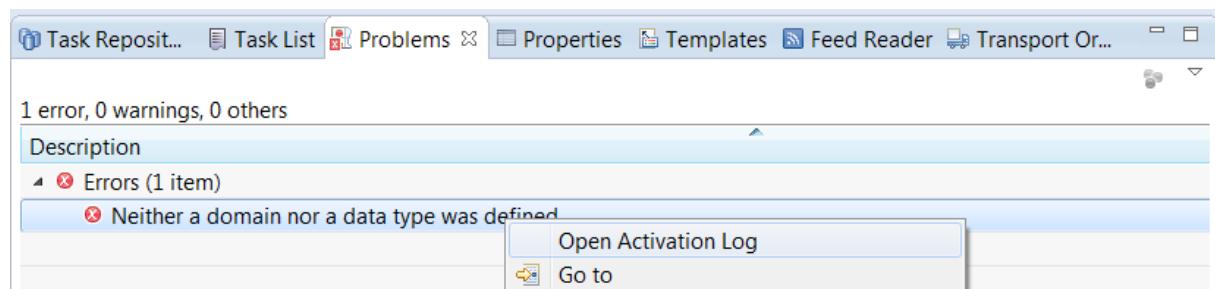
[Activating Development Objects \[page 178\]](#)

[Troubleshooting for Activation Errors in the Context of ABAP Dictionary Objects \[page 181\]](#)

## 5.1.4.11.2 Troubleshooting for Activation Errors in the Context of ABAP Dictionary Objects

The activation of ABAP Dictionary objects or other ABAP objects, which in turn reference ABAP Dictionary objects (such as data types), can cause errors. This can, for example, occur if for a data element neither a domain nor a data type is defined in the ABAP Dictionary.

In such a case, you can examine the ABAP Dictionary log for further troubleshooting. To view it for an error item in the *Problems* view, select the corresponding function in the context menu.



Message	Severity	Message Key
Activation of worklist	Information	D0(302)
Technical log for mass activation	Information	D0(302)
See log 20130325133210:ACT	Information	PU(323)
✖ DTEL TEST3 was not activated	Error	D0(408)
Check data element TEST3	Information	D0(302)
✖ Neither a domain nor a data type was defined	Error	D0(371)
Data element ZMB_TEST3 is inconsistent	Information	D0(304)
End of activation of worklist	Information	D0(401)
	Information	D0(302)

Activation error message that is displayed in the ABAP Log view

### Related Information

[Activating Development Objects \[page 178\]](#)

[Troubleshooting for ABAP Syntax Errors and Warnings \[page 180\]](#)

## 5.1.4.12 Changing the Package Assignment of Development Objects

You can move a development object into another ABAP package.

### Context

You have, for example, created an ABAP class in an ABAP package and you want to ship it with the productive ABAP source code to other systems.

#### i Note

This functionality is provided for all development objects that are directly assigned to an ABAP package.

Currently, it is not possible to select multiple objects for a package reassignment.

### Procedure

1. Open the context menu from the corresponding development object in the *Project Explorer* or *ABAP source code editor*.
2. Choose *Change Package Assignment*.  
The *Change Package Assignment* wizard is opened.
3. In the *New package* entry field of the *Choose Package* page, enter the name of the ABAP package you want to move the object to.

#### i Note

To display the list of the available ABAP packages, choose `Ctrl` + `Space`. A list is then opened where you can select the relevant entry.

You can also use wildcards like the asterisk (\*) to limit the display of relevant entries.

4. Choose *Next*.  
The *Choose Transport* page is opened. In the *Affected objects* list, the objects that are assigned to the new package are displayed.
5. To create a new transport request or to assign your change to an existing transport request, choose the *Change transport* button.
6. Choose *Next*.  
The *Change Package Assignment of [object name]* page is opened. All the objects that are to be changed are listed here.
7. [Optional: ] Deselect those objects where you do not wish the change to be performed.
8. To initiate the object move, choose *Finish*.

## Results

The development object is moved to the selected ABAP package and the content of the *Project Explorer* is refreshed.

## Related Information

[ABAP Packages \[page 38\]](#)

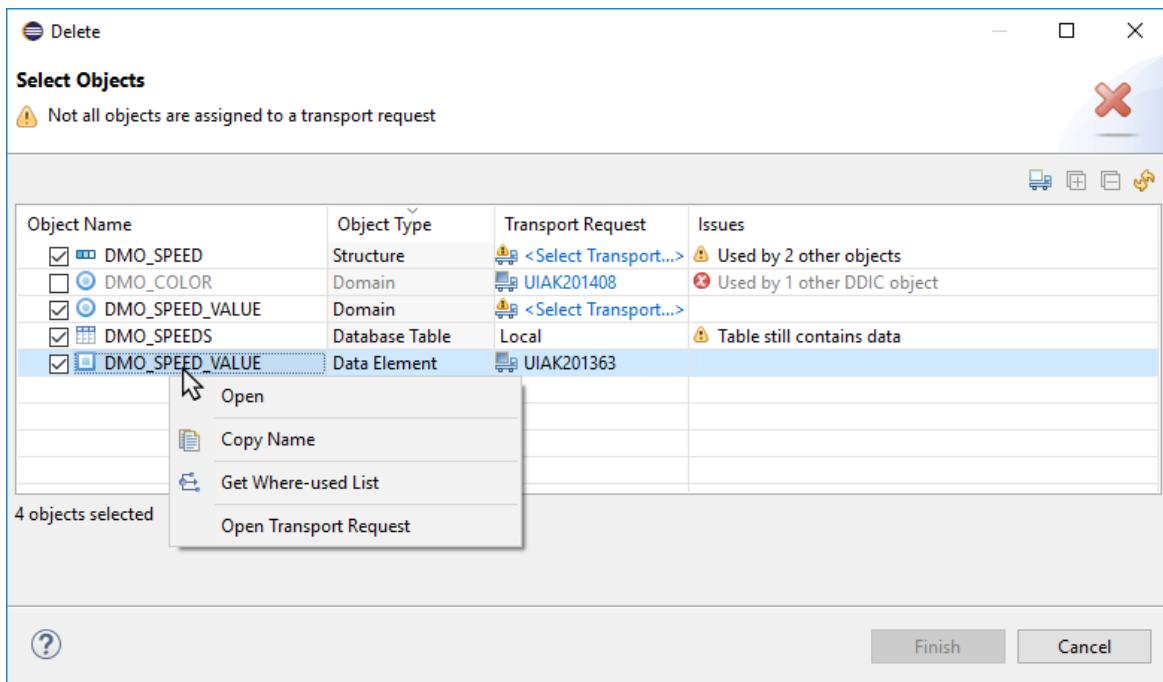
[Adding a Favorite Package \[page 137\]](#)

### 5.1.4.13 Deleting Development Objects

To delete development objects, perform the following steps:

1. In the *Project Explorer*, select one or more development objects.
2. Open the context menu and select *Delete*.

The deletion dialog opens. You can see a list with all your selected objects along with additional information about individual objects.



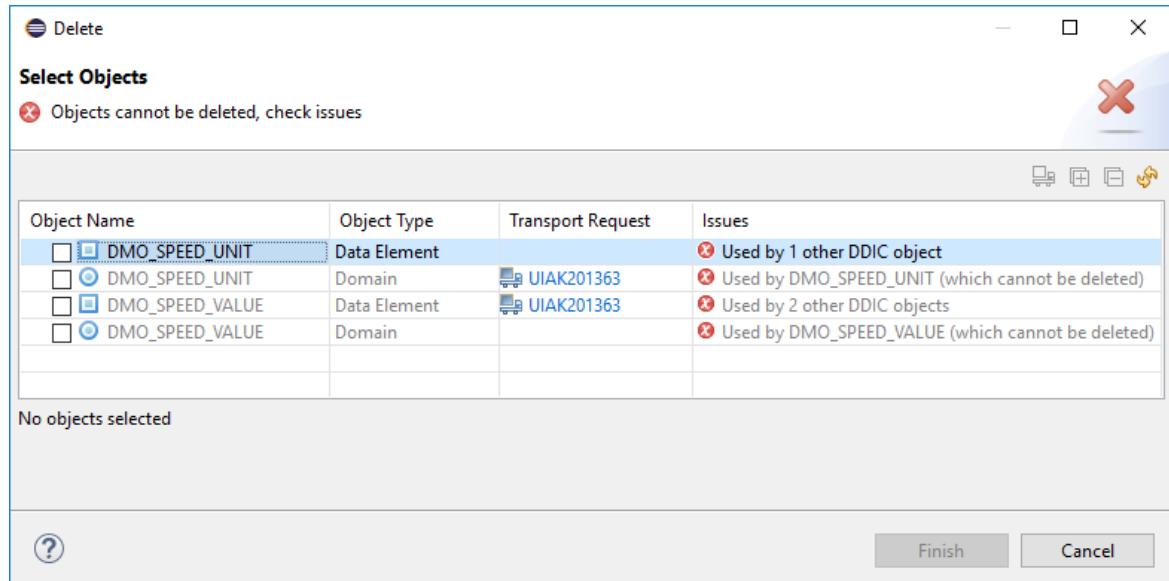
Deletion dialog

The preview shows whether an individual object can be deleted. If the object can be deleted, the checkbox next to the object is selected. If not, the checkbox is deselected and you can see the error description in the *Issues* column. Objects that cannot be deleted are grayed out.

On each object, you can trigger the context menu.

In some cases, you get a warning message in the *Issues* column. For example, if the object is used by another object, or you are currently editing the object. However, you can delete the objects.

If you get an error message in the *Issues* column, you cannot delete the object. For example, if you try to delete an ABAP Dictionary object that is used by another ABAP Dictionary object.



#### Deletion of ABAP Dictionary objects being used by other ABAP Dictionary objects

3. In the *Transport Request* column, you can see whether the object is assigned to a transport request. If it is assigned, you see the transport request. Click it to open the transport request. If it is not assigned, click the entry [<Select Transport...>](#) to select an existing request, or to create a new one.

You can assign all objects that are not yet assigned to a transport request using  in the tool bar.

4. Select *Finish*.

#### i Note

ABAP Packages containing development objects are a special case. To delete the entire package with its content, do the following:

1. Delete the individual objects of the package.
2. Release the transport request with the deletion of the individual objects.
3. Now, you can delete the package as it no longer contains any objects.

## 5.1.5 Working with ABAP Source Code Objects

## 5.1.5.1 Working with ABAP Classes and Interfaces

An ABAP class is a repository object that contains ABAP source code and describes a real project.

### Related Information

[Creating ABAP Classes \[page 185\]](#)

[Creating ABAP Exception Classes \[page 187\]](#)

[Adding Code Templates to an ABAP Exception Class \[page 190\]](#)

[Working with ABAP Classes Assigned to Multiple Transport Requests \[page 192\]](#)

[Launching an ABAP Application \(Console\) \[page 194\]](#)

### 5.1.5.1.1 Creating ABAP Classes

#### Context

You can create local or global ABAP classes to build a template for objects.

##### i Note

The following procedure is for all development objects that you can create completely using the Eclipse-based wizards. ABAP interfaces and executable programs are created in a similar manner.

#### Procedure

1. In the *Project Explorer*, select the relevant package node.
2. Open the context menu and choose to launch the creation wizard.
3. Specify the *Project* and *Package* properties of the ABAP class to be created.
4. Enter the *Name* and *Description* for the class to be created and choose *Next*.

##### i Note

Use **[Your Prefix]CL\_[Your Class Name]** as the name.

5. [OPTIONAL:] Enter the name of the *Superclass* in case the new class has to be inherited from a specific superclass.

##### → Tip

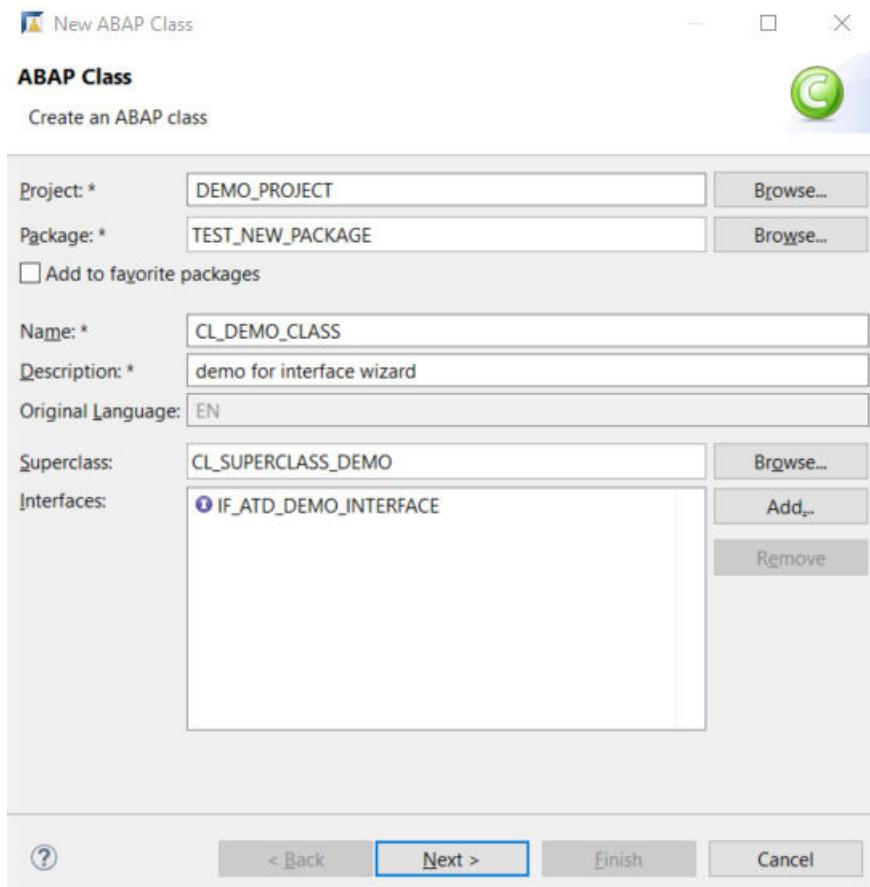
When editing the superclass, you can avail of the content assist functionality by pressing **Ctrl** + **Space**.

6. [OPTIONAL:] Choose *Add...* to search and select an interface.

**i Note**

You can add one or more interfaces.

7. Choose *Next*.



Class Creation Wizard

8. Assign a transport request.

9. Choose *Finish*.

## Results

The back-end system creates an inactive version of a class pool in the selected package that is stored in the class library of the repository. In the *Project Explorer*, the new class is added to the *Source Library* of the corresponding package node. In the source code editor, the initially generated source code is displayed and ready for editing.

After you have created an ABAP class, the following tabs are available in the class editor:

Tab	Description
Global Class	Class that is stored in the class library of the central ABAP Repository
Class-Relevant Local Types	Contains local definitions that are referenced in the private section of the global class
	<b>i Note</b> When these local definitions are changed, all subclasses and friends of the global class must be recompiled because they might depend on these definitions.
Local Types	Contains local definitions that are NOT referenced by the global class
Test Classes	Contains ABAP Unit test classes that enable automated tests during the development phase
Macros	Allows you to add macro definitions for the given global class
	<b>⚠ Caution</b> Macros are regarded as obsolete technology and should no longer be used in ABAP classes.

## Related Information

- [ABAP Development Objects \[page 19\]](#)
- [Transport Request \[page 108\]](#)
- [Tools for ABAP Development \[page 19\]](#)
- [Creating ABAP Exception Classes \[page 187\]](#)
- [Adding a Favorite Package \[page 137\]](#)
- [Adding a Development Object to a Transport Request \[page 164\]](#)

## 5.1.5.1.2 Creating ABAP Exception Classes

### Context

You create an exception class to build a template for objects that is the basis for catchable exceptions.

**i Note**

An ABAP exception class is automatically created whenever one of the following superclasses is added to an ABAP class:

- CX\_STATIC\_CHECK
- CX\_DYNAMIC\_CHECK
- CX\_NO\_CHECK

#### i Note

Cloud icon The exception class implements the system interface [IF\\_T100\\_DYN\\_MSG](#).

The system interface [IF\\_T100\\_DYN\\_MSG](#) allows you to assign arbitrary messages of message classes to the exception by using the statement `RAISE EXCEPTION ... MESSAGE ...` at runtime.

## Procedure

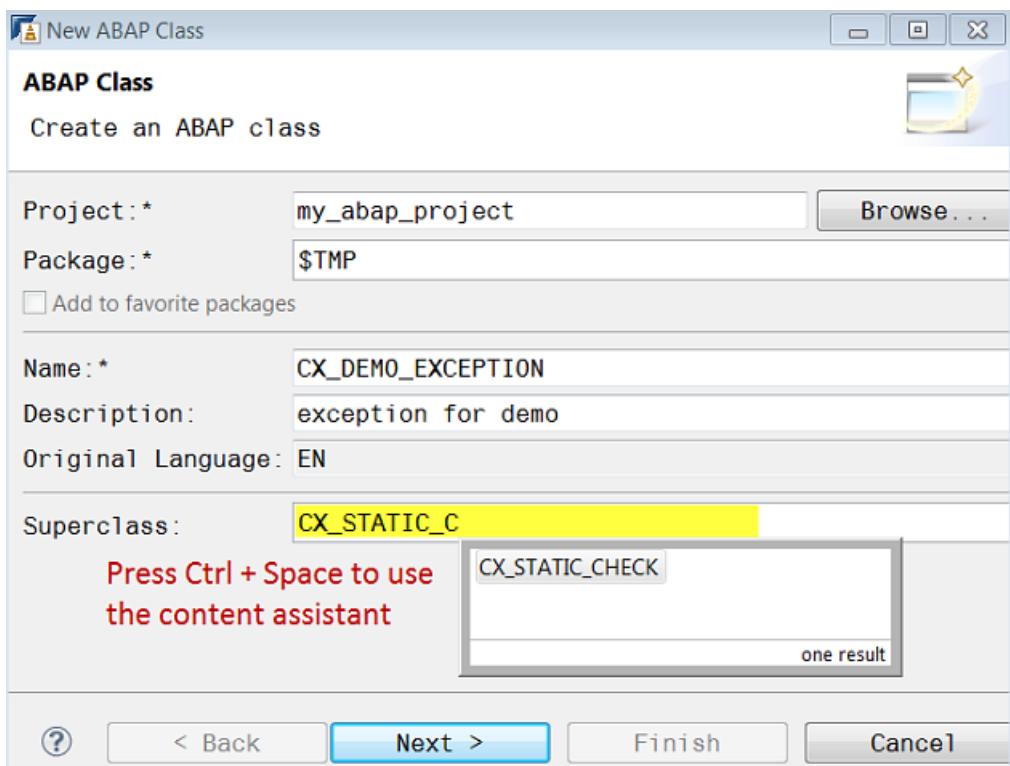
1. In the [Project Explorer](#), select the relevant package node.
2. Open the context menu and choose  [New > ABAP Class](#) to launch the creation wizard.
3. Specify the [Project](#) and [Package](#) properties of the class to be created.
4. Enter the [Name](#) and [Description](#) for the class to be created and choose **Next**.

#### i Note

Use **[Your Prefix]CX\_[Your Class Name]** as the name.

5. In the [Superclass](#) field, enter one name of the following superclasses:
  - CX\_STATIC\_CHECK
  - CX\_DYNAMIC\_CHECK
  - CX\_NO\_CHECK

When editing the superclass, you can use the content assist functionality by pressing **Ctrl** + **Space**.



Creation of an exception class with CX\_STATIC\_CHECK as superclass

6. Choose *Next*.
7. In the following window, select a transport request in order to hand over your new class to another system.
8. Start the creation with *Finish*.

## Results

The exception class is created just like a regular ABAP class. Therefore, the back-end system creates an inactive version of a class pool in the selected package that is stored in the class library of the repository. In the *Project Explorer*, the new class is added to the *Source Library* of the corresponding package node.

In the source code editor, the initially generated source code is displayed and ready for editing.

Here, the constructor method of the superclass that is declared in the public section is added. It contains generated code that reflects the current state of the class hierarchy. Any further changes to the superclass constructor or the class itself will not trigger automatic adaptation of the constructor code in the exception class. This behavior is ensured by the `##ADT_SUPPRESS_GENERATION` pragma that is added where the implementation of the constructor method begins.

The constructor code can be only regenerated by executing the cleanup utility in the back end. As a result, your own code in the constructor method will be lost in ADT after the cleanup.

### i Note

In the exception class, you can add a code template to the definition part. This enables you to reuse source code in order to display a specific T100 message.

## Related Information

[Creating ABAP Classes \[page 185\]](#)

[Adding Code Templates to an ABAP Exception Class \[page 190\]](#)

[Adding a Development Object to a Transport Request \[page 164\]](#)

[Adding Documentation of Parameters and Exceptions in ABAP Doc Comments \[page 317\]](#)

### 5.1.5.1.3 Adding Code Templates to an ABAP Exception Class

#### Context

In the source code editor of an ABAP exception class, you can add the code template `textIdExceptionClass`. This code template contains the constant definitions for assigning the message number and attributes of a message class that are to be displayed.

#### Procedure

1. In the private section of the ABAP exception class, type `textID` and open the code completion window.

2.  **Sample Code**

This code template is added:

```
<example>The code template is added:<codeblock><codeph>CONSTANTS:  
BEGIN OF [Name of the exception class],  
  msgid TYPE symsgid VALUE 'msgid',  
  msgno TYPE symsgno VALUE 'msgno',  
  attr1 TYPE scx_attrname VALUE 'attr1',  
  attr2 TYPE scx_attrname VALUE 'attr2',  
  attr3 TYPE scx_attrname VALUE 'attr3',  
  attr4 TYPE scx_attrname VALUE 'attr4',  
END OF [Name of the exception class].  
</codeph>  
</codeblock></example>
```

Add the `textIdExceptionClass` code template from the popup.

3. In the framed entry fields of the code template, replace the following variables:
  - a. Add the name of the message class in the `msgid` field.
  - b. Add the message number in the `msgno` field.
  - c. [Optional:] If the message contains attributes, add the value for the corresponding `<emphasis>attr</emphasis>` attribute that should be filled in the message.

## Results

### EXAMPLE

In the following example, you see the code snippet of an ABAP exception class referring to a message in the message class BC\_DATAMODEL\_SERVICE.

There is the exception class CX\_MY\_EXCEPTION\_CLASS:

1. The relevant message class has the **ID** BC\_DATAMODEL\_SERVICE.
2. The message that should be displayed has the number **118** and contains the short text 'Number of booked places &1 exceeds flight capacity &2'.
3. The containing variables &1 and &2 are replaced by the values of the attributes BOOKED\_SEATS with the value '242' and SEATS with the value '200'.

As a result, the exception message is displayed as 'Number of booked places 242 exceeds flight capacity 200'.

### Sample Code

```
CONSTANTS:  
BEGIN OF CX_MY_EXCEPTION_CLASS,  
  msgid TYPE symsgid VALUE 'BC_DATAMODEL_SERVICE',  
  msgno TYPE symsgno VALUE '118',  
  attr1 TYPE scx_attrname VALUE 'BOOKED_SEATS',  
  attr2 TYPE scx_attrname VALUE 'SEATS',  
  attr3 TYPE scx_attrname VALUE '',  
  attr4 TYPE scx_attrname VALUE '',  
END OF CX_MY_EXCEPTION_CLASS.  
  
DATA seats TYPE i READ-ONLY.  
DATA booked_seats TYPE i READ-ONLY.
```

## Related Information

[ABAP Development Objects \[page 19\]](#)

[Tools for ABAP Development \[page 19\]](#)

[Creating ABAP Classes \[page 185\]](#)

[Creating ABAP Exception Classes \[page 187\]](#)

## 5.1.5.1.4 Working with ABAP Classes Assigned to Multiple Transport Requests

A class consists of several subobjects, such as a public section, method implementation, or local class includes. Each subobject can be assigned separately to a transport request.

### Context

If you want to edit an ABAP class where the changes have already been assigned to a transport request by another user, you can assign your changes to a different transport request.

### Related Information

[Activating Development Objects \[page 178\]](#)

[Working with Transport Organizer \[page 437\]](#)

[Creating ABAP Classes \[page 185\]](#)

## 5.1.5.1.4.1 Assigning Changes in an ABAP Class to Several Transport Requests

### Context

If you want to edit an ABAP class where the changes have already been assigned to a transport request by another user, you can decide whether you want to assign your changes

- to a transport request to which parts of the current class are already assigned
- to a different transport request.

The source code editor will then use and assign all of your changes to the selected transport request.

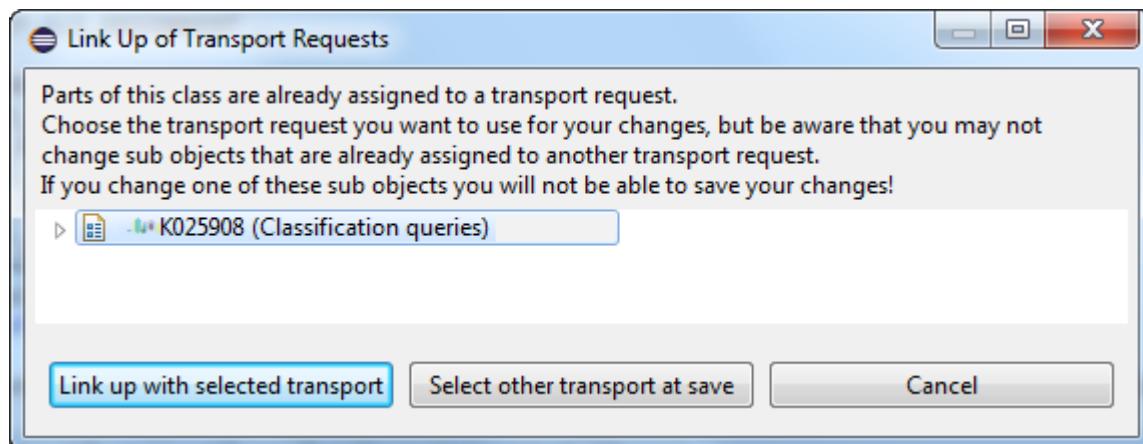
#### Note

Only one developer at a time can edit a class. The class is then locked for other users.

### Procedure

1. Open the relevant class.
2. In the [ABAP source code editor](#), start editing at the position you want to modify.

The *Link Up of Transport Request* dialog is opened.



Example of a dialog where you select the assignment of the transport request

3. Here you have following options:

- Select the **Link up with selected transport** button to combine your changes with the most relevant transport request that is already being used for this class.

**i Note**

The list of displayed transport requests is automatically preselected by ABAP Development Tools.

- Select the **Select other transport at save** button to assign your changes to another existing transport request from your ABAP system or create a new transport request.

The dialog is closed and you can start editing.

4. Add your changes to the source code.

**i Note**

If you are changing the same subobject that has already been assigned to another transport request, you will not be able to save your changes. In this case, you will need to select the transport request to which the subobject is assigned. The reason for this is because changes that depend on each other must be assigned to the same transport request. Otherwise the functionality may not work.

In order to determine the relevant subobjects, set the **Show Transport Request Information for ABAP Class** preference. An icon is displayed afterwards in the marker bar where the changes that are not yet released are added. For further information, see [Displaying the Assignment of Class Artifacts to Transport Requests \[page 194\]](#)

5. Save the changes.

**i Note**

If you have selected the **Select other transport at save** button in step 3, a dialog is opened where you can select an existing transport request or create a new one.

6. [Optional:] Activate the changes.

**i Note**

If you have split the changes amongst several transport requests, an activation or confirmation dialog is opened. Here you select the transport requests or objects you want to activate.

## Results

Your changes are stored, assigned to the selected transport request, and can now be transported to subsequent systems through your releasing the transport request.

### 5.1.5.1.4.2 Displaying the Assignment of Class Artifacts to Transport Requests

#### Context

In the marker bar of the *ABAP source code editor*, you can display the following icons that highlight the subobjects of the ABAP class to which the changes are assigned

- : a different transport request.
- : your selected transport request.

When you hoverover the icon, a popup is opened that displays the:

- Transport number of the assigned transport request and the task number in brackets
- Name of the relevant class
- Name of the user who is owner of the transport request
- Changed subobject(s)

If you wish that ABAP Development Tools (ADT) displays the icon, you will need to activate the following setting in the *Preferences* page:

► *Window* ► *Preferences* ► *ABAP Development* ► *Editors* ► *Source Code Editors* ► *Show Transport Request Information for ABAP Class* ▶

### 5.1.5.1.5 Launching an ABAP Application (Console)

You can execute an ABAP class directly in ABAP Development Tools (ADT).

#### Context

You want to display any kind of text and/or content of internal tables into the *Console* view in ADT.

## Procedure

To launch an ABAP class, proceed as follows:

1. Open the relevant ABAP class.
2. In the PUBLIC section, add the `if_oo_adt_classrun` ABAP interface.
3. Implement the `main` method from this interface using the [Add implementation for main](#) quick fix.  
The `if_oo_adt_classrun~main` method is implemented.
4. In the method stub, implement your logic to be executed.
5. [Optional:] To write any kind of text or content of internal tables, call one of the methods from the given import parameter.

### Example

Example of an implementation of the `main` method:

```
METHOD if_oo_adt_classrun~main.
  SELECT * FROM snwd_bpa INTO TABLE @DATA(lt_data).
  out->write(
    EXPORTING
      data = lt_data
      name = 'Output Title'
  ).
ENDMETHOD.
```

To define the output, the name of the output title and an internal table are added as exporting parameters.

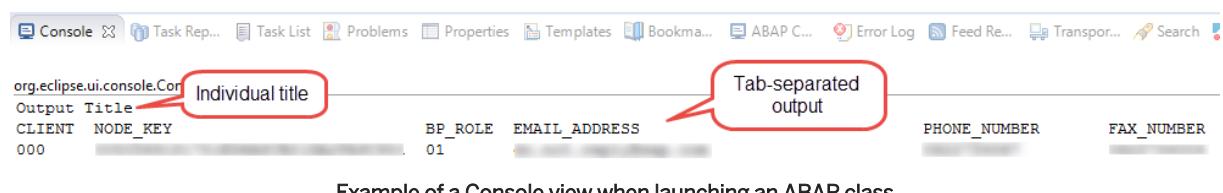
Here, to add a title to the console view, an export parameter is used in the `out->write` statement.

6. Check and activate the class.
7. Assign a transport request.
8. In the source code editor, choose `Run As` + `ABAP Application (Console)` from the context menu or `F9` directly.

Alternatively, you can launch the ABAP class from the corresponding context menu in the [Project Explorer](#).

## Results

The main method of the class is executed and the text output is written to the [Console](#) view that is opened.



Example of a Console view when launching an ABAP class

You can now check the text output, or copy and paste it, for example, to a text file.

## 5.1.5.1.6 Creating ABAP Interfaces

### Context

#### i Note

The following procedure is for all development objects that you can create completely using the Eclipse-based wizards. ABAP classes and executable programs are created in a similar manner.

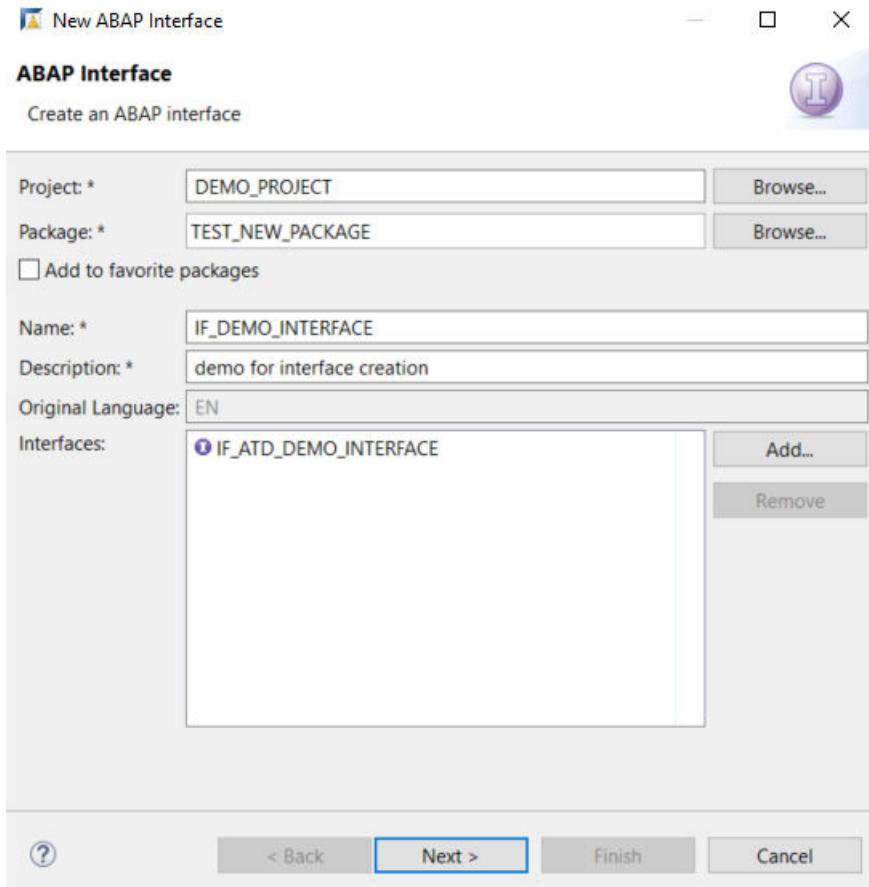
### Procedure

1. In the *Project Explorer*, select the relevant package node.
2. Open the context menu and choose   to launch the creation wizard.
3. Specify the *Project* and *Package* properties of the ABAP interface to be created.
4. Enter the *Name* and *Description* for the interface to be created and choose *Next*.

#### i Note

Use **[Your Prefix]IF\_[Your Interface Name]** as the name.

5. [OPTIONAL:] Choose *Add...* to search and select one or more interfaces to be implemented in the interface you want to create.
6. Choose *Next*.



Interface Creation Wizard

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

## Results

You have created an ABAP interface in the specified package.

### 5.1.5.2 Working with ABAP Function Groups and Modules

Function modules allow you to encapsulate and reuse global functions in an ABAP system.

#### Related Information

[Creating an ABAP Function Group \[page 198\]](#)

[Creating an ABAP Function Module \[page 202\]](#)

[Creating an ABAP Function Group Include \[page 199\]](#)

[Modularization with Function Modules \[page 22\]](#)

## 5.1.5.2.1 Creating an ABAP Function Group

### Context

You create an ABAP Function Group in order to group function modules and includes with similar or complementary functions.

### Procedure

1. In your ABAP project, select the relevant *Package* node in the *Project Explorer*.
2. Open the context menu and select ► *New* ► *Other Repository Objects* ►.
3. Expand the *Source Library* folder and select *ABAP Function Group*.
4. Choose *Next*.
5. In the following dialog, enter a unique name for the new function group in the *Name* field.

#### i Note

You can enter maximum 26 digits.

6. Enter a *Description* providing additional details.

#### i Note

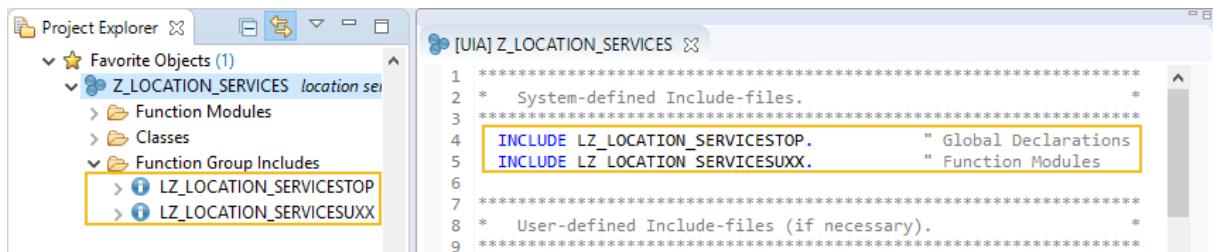
The *Original Language* is already predefined by the default language of the ABAP project.

7. Choose *Next*.
8. Assign a *Transport Request*.
9. Choose *Finish*.
10. Activate the new ABAP Function Group.

### Results

The ABAP repository creates a new function group in the selected package and adds it in the *Project Explorer* to the *Source Library* folder.

The function group automatically contains a TOP include and a UXX include.



Display of created includes as new folder level in the Project Browser and source code in the definition part

## Related Information

[Creating an ABAP Function Module \[page 202\]](#)

[Creating an ABAP Function Group Include \[page 199\]](#)

[Modularization with Function Modules \[page 22\]](#)

[Adding a Development Object to a Transport Request \[page 164\]](#)

## 5.1.5.2.2 Creating an ABAP Function Group Include

### Prerequisites

You create an ABAP Function Group Include to separate and modularize functionality.

### Context

You can create an ABAP Function Module to which you provide reusable functions.

### Procedure

1. In your ABAP project, select the relevant *Package* node in the *Project Explorer*.
2. Expand the *Source Library* folder and select the relevant *Function Group*.
3. Open the context menu and select *New* *ABAP Function Group Include*.
4. In the following dialog the *Name* will automatically be entered.

#### Note

The name will be constructed using the following convention: [/name space/] [L<Name of the function group>ID of the include suffix]. L stands for the standard prefix for ABAP function group includes.

5. Enter a *Description* that provides additional details.

### i Note

The *Original Language* will automatically be assigned with the same language as defined in the function group.

6. [Optional] Click the button *Browser...* if you want to add the function module to another **Function Group**.
7. Enter the *Suffix* for the include.

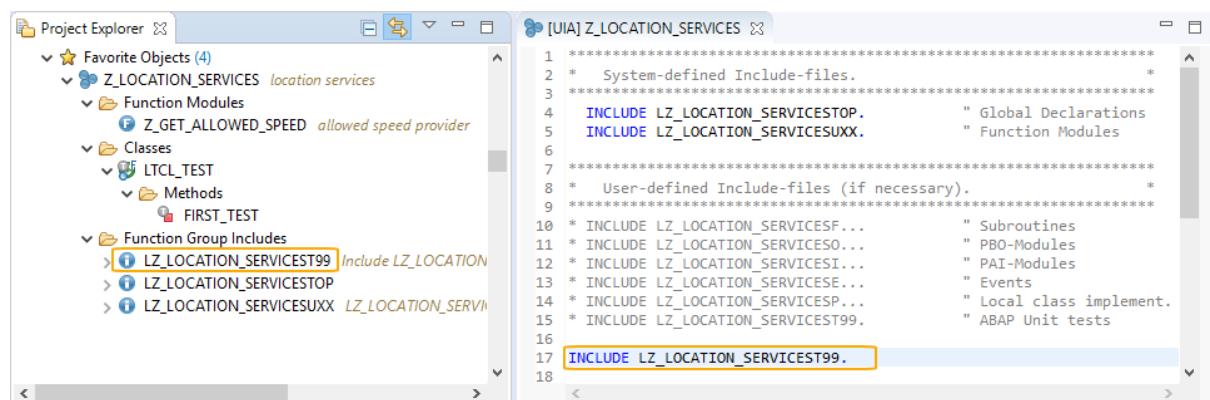
### i Note

See [Adding Include Suffixes \[page 201\]](#) for more details.

8. Choose *Next*.
9. Assign a *Transport Request*.
10. Choose *Finish*.
11. Activate the new ABAP Function Group Include.

## Results

The include will be created on the relevant function group in the *Project Browser*. It will also be added with the statement `INCLUDE <Include Name>` into the main program.



Display of the created group include in the Project Browser and in the editor of the main program

## Related Information

[Adding a Development Object to a Transport Request \[page 164\]](#)

[Modularization with Function Modules \[page 22\]](#)

[Creating an ABAP Function Group \[page 198\]](#)

[Adding Include Suffixes \[page 201\]](#)

## 5.1.5.2.2.1 Adding Include Suffixes

### Context

You can add following suffixes to the ABAP function group includes:

Include Type	Prefix	Suffix	Usage
Top include	L	TOP	Declaration of global data of the function group
		DNN	Declaration of local classes
Optional include		D..*	Declaration of local classes definitions that are to be included in the top include of the function groups
Include		UXX	Implementation section of the function group that includes all existing U.. includes NOTE: These includes are not editable.
		U01-99	For each function module of the function group
Optional include		T99	For ABAP unit test classes that are to be included in the implementation section of the function group
		P..*	Methods of local class implementations
		PNN	Implementation of local classes that are to be included in global data of the function groups
		O..* and ONN	PBO modules of dynpros that are to be included in the implementation part of the function group
		I..* and INN	PAI modules of dynpros that are to be included in the implementation plant of the function group
		E..*	Event blocks in the implementation part of the function group
		F..*	Subroutines in the implementation part of the function group

\*The periods '..' represents a double-digit number.

### i Note

\$nn, Unn, Vnn, Cnn, T01 - T98 are reserved suffixes. They cannot be used.

## Related Information

[Creating an ABAP Function Group Include \[page 199\]](#)

[Modularization with Function Modules \[page 22\]](#)

### 5.1.5.2.3 Creating an ABAP Function Module

#### Prerequisites

You have created a function group in which you want to add the function module.

#### Context

You can create an ABAP Function Module for which you provide reusable functions.

#### Procedure

1. In your ABAP project, select the relevant *Package* node in the *Project Explorer*.
2. Expand the *Source Library* folder and select the relevant *Function Group*.
3. Open the context menu and select  *New*  *ABAP Function Module*.
4. In the following dialog, enter a unique name for the new function group in the *Name* field.

### i Note

You can enter maximum 30 digits.

5. Enter a *Description* providing additional details.

### i Note

The Original Language will automatically be assigned with the same language as defined in the function group.

6. Enter a *Description* providing additional details.

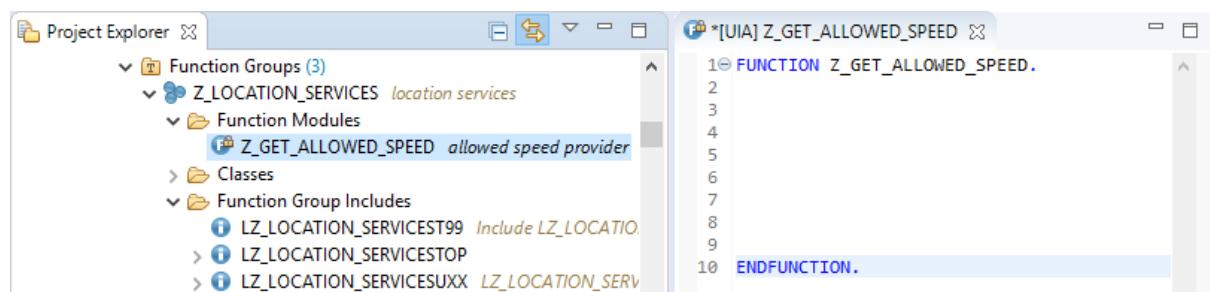
### i Note

The *Original Language* is already predefined by the default language of the ABAP project.

7. [Optional] Choose the *Browser...* button if you want to add the function module to another *Function Group*.
8. Choose *Next*.
9. Assign a **Transport Request**.
10. Choose *Finish*.

## Results

The function module is assigned to the function group selected in the function module folder of the *Source Library*.



```
1 FUNCTION Z_GET_ALLOWED_SPEED.
2
3
4
5
6
7
8
9
10 ENDFUNCTION.
```

Editor that is displayed after creating the function module

In the next step, you can enter the definition part and implementation part of the function module.

## Related Information

[Adding a Development Object to a Transport Request \[page 164\]](#)

[Creating an ABAP Function Group \[page 198\]](#)

[Creating an ABAP Function Group Include \[page 199\]](#)

[Changing a Definition Part and Implementation Part of a Function Module \[page 204\]](#)

[Modularization with Function Modules \[page 22\]](#)

## 5.1.5.2.4 Changing Function Modules

## Related Information

[Changing a Definition Part and Implementation Part of a Function Module \[page 204\]](#)

[Specific Properties of Function Modules \[page 205\]](#)

### 5.1.5.2.4.1 Changing a Definition Part and Implementation Part of a Function Module

#### Context

#### Procedure

1. Open the function module in the editor.
2. Edit the definition part in the editor.

##### **i** Note

Finish the definition part with a period.

##### **⚠** Caution

Do not add any comments to parameters in the definition part. Otherwise they will be deleted after saving and activating the function module.

##### **→** Tip

You can use the following template in the definition part of your function module:

```
IMPORTING
  IM_PARAM1 TYPE ANY
  IM_PARAM2 TYPE REF TO <ref_type>
EXPORTING
  EX_PARAM_1 TYPE ANY
  EX_PARAM_2 TYPE REF TO <ref_type>
CHANGING
  CH_PARAM TYPE ANY
TABLES
  TAB_PARAM TYPE STANDARD TABLE
EXCEPTIONS
  EXCEPTIONS_VALUE.
```

3. Edit the implementation part in the editor.

##### **⚠** Caution

Do not start your own comments with \*" or \*"-- as these are reserved for the parameter block in the function editor of the back end. Start your comments for each line with \* or ".

Consider the implementation part as finished with `ENDFUNCTION..`

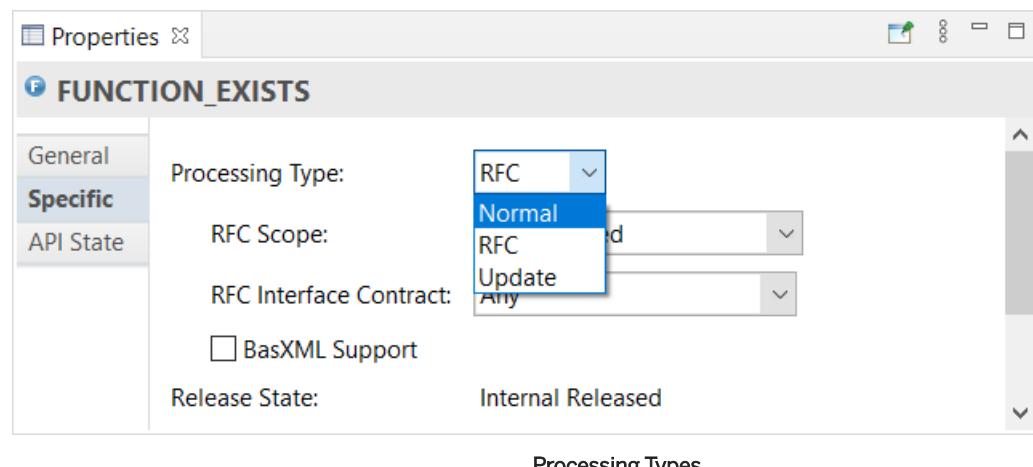
4. Save the editor contents.
5. Activate the ABAP Function Module.

## Related Information

[Creating an ABAP Function Group \[page 198\]](#)  
[Creating an ABAP Function Module \[page 202\]](#)  
[Modularization with Function Modules \[page 22\]](#)

### 5.1.5.2.4.2 Specific Properties of Function Modules

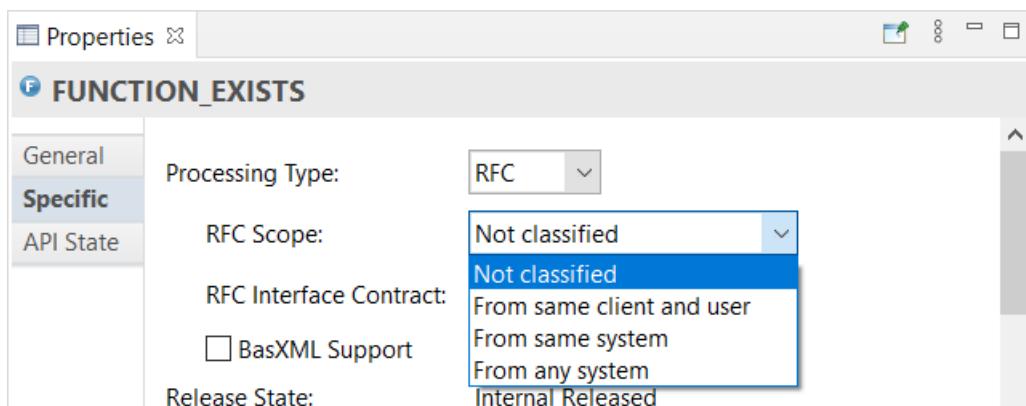
In the *Properties* view, under the *Specific* tab, you can change the processing type of a function module to an RFC-enabled or an update module.



## RFC-Enabled Modules

If you select the processing type *RFC*, you can define *RFC Scope* and *RFC Interface Contract*.

### Scope of RFC Function Module



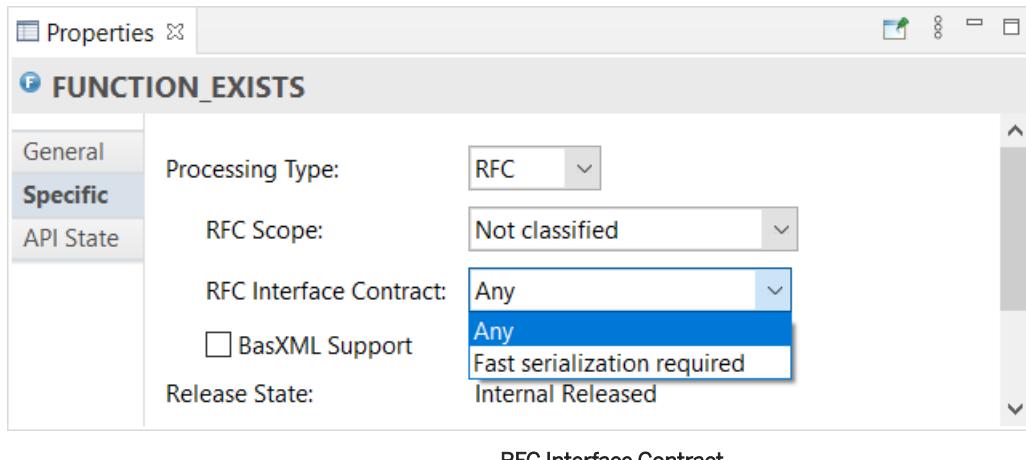
This is where you define the scope of function module calls. Modules whose call scope is not defined explicitly can be called from anywhere.

Function modules used for parallelization or for decoupling should only be called within the same system without switching the user. In this case, choose *From same client and user*. RFC runtime checks the call scope, which is why you do not need any additional authorization checks for authorization calls.

Modules calling from one client to another or switch users (such as administration modules) should be declared as *From same system*.

The *From any system* setting enables calls to be made from any system with any user. Verify that the required authorization checks are in place.

## RFC Interface Contract



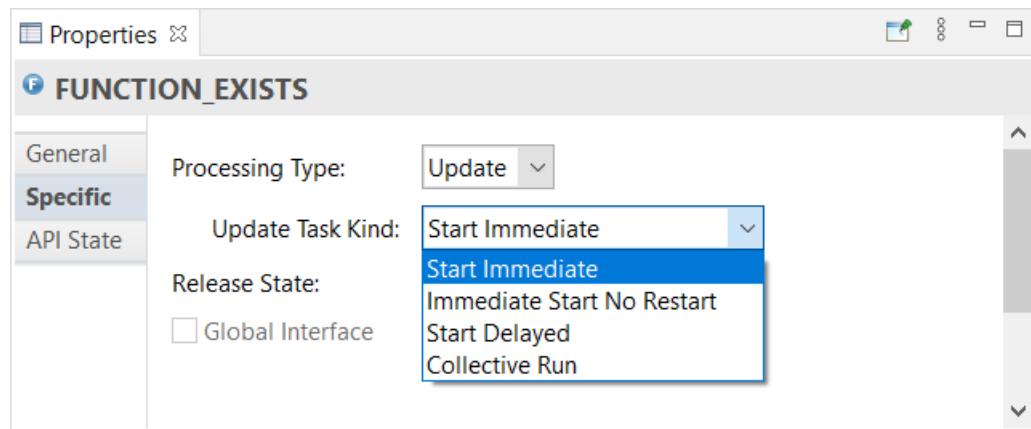
You select *Interface Contract* to specify which serializations are permitted for the function module. Among other things, the selected serialization or serializations determine where the function module can be enhanced (for example, if you need additional fields in a structure in a parameter).

## BasXML Support

If you selected *Any* for *RFc Interface Contract*, you can enable the checkbox *BasXML Support*. It allows you to switch from the classic RFC protocol to the basXML protocol for RFC-enabled function modules for RFC communication.

When this checkbox is enabled, the developer of the function module shows that the function module for the classic RFC protocol and the basXML protocol behaves in the same way, from a semantic point of view. Semantic differences result from the conceptional differences between the two protocols. The classic RFC protocol uses a position logic, while the basXML protocol uses a name identification.

## Update Modules



Update Task Kinds

Selecting the processing type *Update*, you can define the update task kind.

- Start Immediate  
Set this option for high priority ("V1") functions that run in a shared (SAP LUW). These functions can be restarted by the update task in case of errors.
- Immediate Start No Restart  
Set this option for high priority ("V1") functions that run in a shared (SAP LUW). These functions may not be restarted by the update task.
- Start Delayed  
Set this option for low priority ("V2") functions that run in their own update transactions. These functions can be restarted by the update task in case of errors.

### 5.1.5.3 Comparing Source Code

#### Context

ABAP Development Tools (ADT) provides you with tools for comparing source code. You can see how code has changed over time. You can also compare code versions across ABAP projects (ABAP systems).

Differences are displayed in the compare editor. Here, you can edit and merge changes. This enables you to take over single or all adoptions from one system to another or from previous to the inactive version of the object. For more information, look here [Merging changes in the compare editor](#). Additionally, you can unlock objects or open them in their default editor through the context menu.

There are following code comparison tools available:

- Comparing Local Changes [page 208]
- Comparing Transported Code Versions [page 209]
- Comparing Across ABAP Projects (ABAP Systems) [page 210]

## Related Information

[Adding a Favorite Package \[page 137\]](#)

[Switching between Inactive and Active Versions of a Source-based Object \[page 341\]](#)

[ABAP Projects \[page 56\]](#)

### 5.1.5.3.1 Comparing Local Code Versions

#### Context

For finely detailed analysis of recent changes to source code, use the local version history. The local version history lets you examine changes between saves in the ADT, showing changes in the code as stored in the local workspace.

#### Procedure

1. Open the development object in the editor.
2. From the context menu, choose  .
3. In the *History* view, double-click a local version for comparison with the version of the code that is open in the editor.

#### Results

The *Compare Editor* is opened. It shows the currently open code in the *Local:* view. The saved code version is shown with its time stamp in an adjacent view.

Changes are highlighted in the two windows. Locations or changes are shown as icons at the right edge of the saved code version.

## Related Information

[Comparing Source Code \[page 207\]](#)

[Comparing Non-Code Repository Objects \[page 211\]](#)

[Comparing Across ABAP Projects \(ABAP Systems\) \[page 210\]](#)

[Comparing Transported Code Versions \[page 209\]](#)

[Comparing Local Code Versions \[page 208\]](#)

### 5.1.5.3.2 Comparing Transported Code Versions

At the next level of granularity, you can compare changes from one transport of source code to another.

#### Context

You usually transport ABAP code much less frequently than you save it in the ADT. Therefore, comparing transported code versions shows changes over a much longer period of time than the local version history does.

#### Procedure

1. From the context menu of an object in the *Project Explorer* or in the source code editor, choose  [Compare With Revision History](#)

In the ABAP editor, the comparison automatically appears with the open code. If you are comparing an ABAP class and start from the *Project Explorer*, you can choose the section of code you wish to compare. For example, you can compare 'global class' code or 'ABAP Unit test code' versions.

2. Double-click the active version or one of the transported versions of the code.

#### Results

The [Compare Editor](#) is opened. It shows the currently open code in the *Local:* view. The saved code version is shown with its time stamp in an adjacent view.

Changes are highlighted in the two windows. Locations or changes are shown as icons at the right edge of the saved code version.

## Related Information

[Comparing Source Code \[page 207\]](#)  
[Comparing Non-Code Repository Objects \[page 211\]](#)  
[Comparing Across ABAP Projects \(ABAP Systems\) \[page 210\]](#)  
[Comparing Transported Code Versions \[page 209\]](#)  
[Comparing Local Code Versions \[page 208\]](#)

### 5.1.5.3.3 Comparing Across ABAP Projects (ABAP Systems)

#### Context

You can also compare the newest version of a program in each of two ABAP projects. You can use this comparison to check code levels in separate ABAP back end systems. (Comparing projects within a single system always shows you the same code version, since ABAP code is independent of clients within a system.)

For example, you can check whether a correction in the development system has already been transported into the consolidation system.

#### Procedure

From the context menu of an object in the *Project Explorer* or in the source code editor, choose  [Compare With > ABAP project](#) , where ABAP project is one of the ABAP projects defined in your ADT.

#### Results

A new editor window opens, showing the currently open code in the *Local:* view. The saved code version is shown with its time stamp in an adjacent view.

Changes are highlighted in the two windows. Locations or changes are shown as icons at the right edge of the saved code version.

##### Note

You can use the `Alt Shift C` shortcut. Then, ADT identifies the changes between the same development object between the current and the last used ABAP system. The differences are displayed in a popup.

## Related Information

[Comparing Source Code \[page 207\]](#)  
[Comparing Non-Code Repository Objects \[page 211\]](#)  
[Comparing Transported Code Versions \[page 209\]](#)  
[Comparing Local Code Versions \[page 208\]](#)

### 5.1.5.3.4 Comparing Non-Code Repository Objects

#### Context

**i** Note

You can compare non-code objects, such as Data Dictionary structures or tables, using the SAP GUI tools in the back end system.

## Related Information

[Comparing Source Code \[page 207\]](#)  
[Comparing Across ABAP Projects \(ABAP Systems\) \[page 210\]](#)  
[Comparing Transported Code Versions \[page 209\]](#)  
[Comparing Local Code Versions \[page 208\]](#)

### 5.1.5.4 Using Code Folding

You can use code folding to hide or display a block of code in order to get a better overview and to improve readability.

#### Context

You can use code folding to hide or display coherent parts of source code blocks such as:

- A class itself,
- A class definition and/or implementation part,

- Methods,
- If clauses,
- Loops,
- Test seams,
- A list of several coherent CDS annotations or comments,
- And so on

To use code folding, you have the following possibilities:

## Procedure

1. To define the general behavior for a development object, choose one of the following entries from the context menu of the ruler in the source editor:
  - *Folding > Enable Folding (Ctrl+Numpad\_Divide)*: To enable/disable code folding in general
  - *Folding > Expand All (Ctrl+Numpad\_Multiply)*: To display all coherent parts
  - *Folding > Collapse All (Ctrl+Shift+Numpad\_Multiply)*: To collapse all coherent parts (Only the first row of the fold is then displayed.)
2. To collapse or expand only selected parts, choose the or the sign from the ruler.

## Results

When the block is expanded, the sign is displayed and a vertical line from the first to the last row of the fold indicates its range.

When the block is collapsed, the sign and only the first row of the fold is displayed. The remaining code of the block disappears from the source editor. The latter is indicated by the indicator. The row numbering remains as before execution. Note that the hidden source code is not deleted and is still available for your development object.

### Example

By default, the complete source code is displayed without any code folding.

```

1 @AbapCatalog.sqlViewName: 'sflight_sql_view'
2 @AbapCatalog.compiler.compareFilter: true
3 @AccessControl.authorizationCheck: #CHECK
4 // @EndUserText.label: 'Availability Query'
5 @Metadata.allowExtensions: true
6 // @Search.searchable: true
7
8 define view Sflight_Demo_Ddlx_Test
  with parameters

```

Example of a fully expanded and displayed code fold

When it is collapsed, only the first row of the fold is displayed:

```
1④ @AbapCatalog.sqlViewName: 'sflight_sql_view'④
④ 8 define view Sflight_Demo_Ddlx_Test
④ 9 with parameters
```

Example of a collapsed fold to hide

When hovering over the  sign, a popup will display the hidden code.

```
D SFLIGHT_DEMO_DDLX_TEST X
i 1④ @AbapCatalog.sqlViewName: 'sflight_sql_view'④
④ 9 @AbapCatalog.compiler.compareFilter: true
10 // @AccessControl.authorizationCheck: #CHECK
11 @EndUserText.label: 'demo neu'
12 @Metadata.allowExtensions: true
13 // @Search.searchable: true
14 @OData.publish: true
15
16
17④
18
```

Example for a popup to display hidden code

## 5.1.6 Working with Classic Objects in ABAP Dictionary

In this part of the documentation you will find detailed descriptions of the elementary activities concerning classic objects in ABAP Dictionary as such.

The ABAP IDE provides a number of functions and utilities for efficiently editing the following ABAP Dictionary objects:

- [Working with Domains \[page 214\]](#)[Working with Data Elements \[page 219\]](#)
- [Working with Structures \[page 225\]](#)
- [Working with Database Tables \[page 230\]](#)
- [Working with Table Types \[page 236\]](#)
- [Working with Lock Objects \[page 240\]](#)

### Related Information

[ABAP Dictionary Editors \[page 63\]](#)

[Syntax of ABAP Dictionary Objects \[page 751\]](#)

[Keyboard Shortcuts for ABAP Development \[page 747\]](#)

## 5.1.6.1 Working with Domains

You can create and work with domains in ABAP Development Tools (ADT) using the [Domain Editor](#).

### Related Information

[Domains \(ABAP Keyword Documentation\)](#)

[Creating Domains \[page 214\]](#)

[Creating Append Domains \[page 215\]](#)

[Editing Domains \[page 216\]](#)

### 5.1.6.1.1 Creating Domains

A domain specifies the technical characteristics and the allowed values of a field.

#### Prerequisites

- You have identified the ABAP package in which the development object is going to be created.

##### → Tip

If you have not yet already done so, add this package to the [Favorites](#) of your ABAP project. See also: [Adding a Favorite Package \[page 137\]](#)

#### Context

You create a domain to define technical and semantic attributes of a field.

#### Procedure

1. In the [Project Explorer](#), select the relevant [Package](#) node.
2. Open the context menu and choose    .
3. Select [Domain](#) to launch the creation wizard.
4. Enter the [Name](#) and [Description](#) for the domain to be created.
5. Choose [Next](#).

6. Assign a transport request.
7. Start the creation with *Finish*.

## Results

The inactive version of a domain is created and stored in ABAP Dictionary. In the *Project Explorer*, the new domain is added to the *Dictionary* folder of the corresponding package.

The Domain Editor is opened.

You now need to define the data type format of this domain.

## Related Information

[Creating Append Domains \[page 215\]](#)

[Adding a Favorite Package \[page 137\]](#)

[Adding a Development Object to a Transport Request \[page 164\]](#)

## 5.1.6.1.1.1 Creating Append Domains

### Prerequisites

- You can create an append domain only on the basis of an existing domain.

### Context

You create an append to add further fixed values to an existing domain without modification.

### Procedure

1. In the *Project Explorer*, choose *New Append Domain* from the context menu of the base domain.  
The *Creation Wizard* is opened.
2. Enter the *Name* and *Description* for the append domain to be created.

### i Note

The *Base Domain* is automatically selected and cannot be changed.

3. Choose *Next*.
4. Assign a transport request.
5. Start the creation with *Finish*.

## Results

The inactive version of an append domain is created and stored in ABAP Dictionary. In the *Project Explorer*, the new append domain is added to the *Dictionary* folder of the corresponding package.

The ABAP Dictionary editor is opened. Here you define the fixed values that you want to add to the base domain.

## Related Information

[Editing Domains \[page 216\]](#)

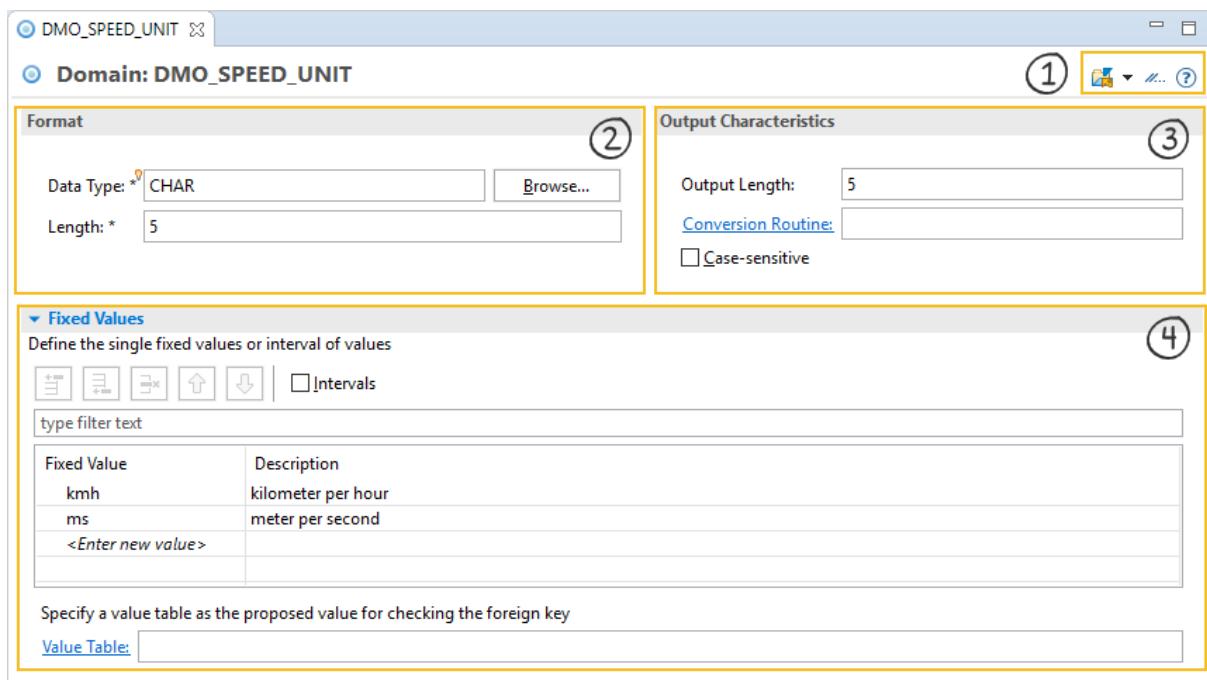
[Creating Domains \[page 214\]](#)

[Adding a Development Object to a Transport Request \[page 164\]](#)

### 5.1.6.1.2 Editing Domains

## Context

You can edit domains in the *Domain* editor.



Example of a domain opened in the Domain editor

The editor provides the following information about an opened domain:

1. *Toolbar* containing actions, such as opening the object in another project, sharing link, opening the context help
2. *Format* for defining the technical attributes, such as the predefined type, its length, and decimal places
3. *Output Characteristics* for output-relevant semantic attributes, such as output length, conversion routine, sign, and case-sensitivity
4. *Fixed Values* with buttons for defining, editing, sorting, and deleting values or intervals of values as well as for adding a value table

## Procedure

1. Open an existing domain or create a new domain.

The domain is opened in the form-based editor.

2. In the *Format* section, you can adapt the technical attributes such as the data type, length, or the number of decimal places.

- a. Enter the predefined type in the *Data Type\** input field.

### → Tip

To get support from the content assist, choose **Ctrl** + **Space**. Then, the relevant proposals are displayed for selection.

- b. In accordance to the selected data type, enter the *Length\** and *Decimals\** to define the number of decimal places.

3. In the *Output Characteristics* section, you can maintain the semantic output attributes.
    - a. Enter the *Output Length*.
    - b. [Optional:] You can select a function module as a conversion rule. This enables you to define automatic conversion of input in dynpro fields from display format to SAP-internal format and vice versa. To assign the function modules, follow the *Conversion Routine* link.
  - The *Open ABAP Development Object* dialog is opened and lists the conversion function modules.
  - c. Depending on the selected data type, you can maintain additional information such as *Numeric Sign*, *Case-sensitive*, *AM/PM time format supported*, and so on.
4. In the *Fixed Values* section, you can enter the possible values for this domain and assign a value table.

- a. Define a value in the *Fixed Value* column and provide a *Description* for it.

To simplify editing, you can use the following possibilities:

Icon	Option	Description
	<i>Insert before/ Insert after</i>	To insert a value/value range before/after the current cursor position.
	<i>Move up / Move down</i>	To move a value/value range one row up/down from the current cursor position.
	<i>Delete</i>	To delete the selected value/value range.

- b. [Optional] To define a value range with lower and upper limits, select the *Intervals* checkbox.
- c. [Optional] To define a database table as a check table, add the relevant one in the *Value Table* input field.

→ Tip

To get support from the content assist, choose **Ctrl** + **Space**. Then, the relevant proposals are displayed for selection.

5. Choose
6. Choose

## Related Information

[Technical Attributes of Domains \(ABAP Keyword Documentation\)](#)

[Semantic Attributes of Domains \(ABAP Keyword Documentation\)](#)

[Creating Domains \[page 214\]](#)

[Creating Append Domains \[page 215\]](#)

## 5.1.6.2 Working with Data Elements

You can create and work with data elements in ABAP Development Tools (ADT) using the Data Element Editor. You create and edit data elements to define an elementary data type or a reference type.

### Related Information

[ABAP Dictionary Editors \[page 63\]](#)

[Creating Data Elements \[page 219\]](#)

[Editing Data Elements \[page 220\]](#)

[Data Elements \(ABAP Keyword Documentation\)](#)

### 5.1.6.2.1 Creating Data Elements

#### Prerequisites

- You have identified the ABAP package in which the development object is going to be created.

##### → Tip

If you have not yet already done so, add this package to the Favorites of your ABAP project. See also:  
[Adding a Favorite Package \[page 137\]](#)

#### Procedure

1. In the *Project Explorer*, select the relevant *Package* node.
2. Open the context menu and choose   .
3. Select *Data Element* to launch the creation wizard.
4. Enter the *Name* and *Description* for the data element to be created.
5. Choose *Next*.
6. Assign a transport request.
7. Start the creation with *Finish*.

## Results

The inactive version of a data element is created and stored in ABAP Dictionary. In the *Project Explorer*, the new data element is added to the *Dictionary* folder of the corresponding package.

The Data Element Editor is opened. Here you can edit the data element.

## Related Information

[Editing Data Elements \[page 220\]](#)

### 5.1.6.2.2 Editing Data Elements

#### Prerequisites

## Context

Example of a data element opened in the form-based editor

This editor contains the following information:

1. *Toolbar* containing actions, such as [opening the object in another project \[page 163\]](#), [sharing link \[page 166\]](#), and opening the context help
2. *Data Type Information* for defining the type of the data element based on a predefined type, a domain, and so on.

Select the relevant *Category* from the dropdown listbox.

The subsequent options are available to you.

### → Tip

When entering, a name in an entry field, for example, you can benefit from the content assist functionality. Edit its first letters. Then, choose `Ctrl` + `Space` to display and select relevant proposals.

- If the data element needs to have the type attributes of a domain, select *Domain*.  
In this case, enter the name of a domain in the *Type Name* field.  
You can also define a new domain. To create it, enter the name of the new domain in the *Type Name* field and then double-click it.
- If you choose *Predefined Type*, you can define the data type, length, and the number of decimal places of the data element using the direct type in the *Type Name* field.

- If the data element needs to describe a reference type, choose one of the following *Reference To* types to define the data type, length, and number of decimal places of the data element:
  - *Reference to Predefined Type* to map to a predefined type in ABAP Dictionary and its fundamental technical attributes
  - *Reference to Dictionary Type* to map to a data type of the ABAP Dictionary
  - *Reference to Class/Interface* to map to an ABAP class or interface of the repository
- 3. *Field Labels* for defining the relevant UI text and its maximum length as well as to add supplementary documentation.

You can use short, medium, and long field labels, and the title. Each label is an input template. It represents entry fields that refer to this data element.

- 4. *Additional Properties* for adding search helps or defining the reading direction of UI text and so on.

In this section, you have the following options:

- In the *Search Help* area, choose the link on the *Name* label or enter the name of a new search help object in order to reference or create a new search help object. In addition, you can define a *Parameter*. You can assign a search help to the data element that is displayed in all the screen fields referring to this data element when the input help (`F4` help) is called. Assign the search help by specifying the name of the search help in the *Name* field and an export parameter in the *Parameter* field. The data element of the parameter is automatically copied from the selection method. The data element defines the output attributes as well as the F1 help of the parameter in the hit list and in the dialog box for value selection.
- Enter a *Parameter ID* to associate a data element with an SPA/GPA parameter. A field can be filled with default values from the SAP memory using a parameter ID. A screen field is automatically filled with the value stored under the parameter ID of the data element only if this was explicitly permitted in the Screen Painter.

### ❖ Example

If a user only has authorization for company code 001, this company code can be stored in the memory under the corresponding parameter ID at the beginning of a transaction. Fields that refer to the data element for the company code are then automatically filled with the value **001** in all subsequent screen templates. In this case, the corresponding parameter ID only needs to be entered in the data element for the company code.

- Enter a *Component Name* to structure the reference of components or table fields to the entered data element. You can store a proposal for the name of the table fields or structure components that refer to a data element. This results in a more unified assignment of field and component names. Assign the default component in the *Component Name* field.

### → Tip

Use an English-language default name.

Always use this default name for components in BAPI structures (structures with a fixed interface).

- Select the checkbox *Change Document Logging* to log changes to fields in database tables defined with reference to this data element. The data of a business-oriented object can be distributed to several tables. To trace changes to a business object, these tables can be combined in a change document object. Function modules that can be integrated in the corresponding application programs and that log these changes are generated from such an object.

You can find information about the activation flow in the activation log. To display it, open the [ABAP Log](#) view. If errors occurred, the activation log is automatically displayed.

## Related Information

[Creating Data Elements \[page 219\]](#)

 [Data Elements \(ABAP Keyword Documentation\)](#)

[Activating Development Objects \[page 178\]](#)

[Opening Development Objects \[page 159\]](#)

### 5.1.6.2.3 Changing the Documentation Status

The documentation status is a subobject of a data element. It specifies whether documentation has already been written or whether it is required.

#### Context

To specify the documentation status of a data element, proceed as follows:

#### Procedure

1. In the toolbar of the *Field Labels* section, choose the  [Documentation Status](#) button.  
The *Change Documentation Status* dialog is opened.
2. Choose one of the following status options to indicate the usage and availability of the documentation:
  - *Object requires documentation*: [Default] Documentation either already exists or should be written.
  - *Object is explained sufficiently by short text*: Documentation is not required but provided by the short text.
  - *Object is not used in any screens*: Documentation is not required because the data element is not used in any screen fields.
  - *Documentation is postponed temporarily*: Documentation does not (yet) exist. The reason might be, for example, because the use of the data element has not yet been fully clarified.

#### Note

If you choose a status that does not explicitly require documentation for a data element, the existing documentation will be deleted.

3. To assign the changes you made to the subobject to an existing transport request or a new transport request, choose [Next](#).

4. Confirm with *Finish*.

## Results

The documentation relevance of the data element is adopted according to your selection.

## Related Information

[Providing Supplementary Documentation \[page 224\]](#)

[Semantic Attributes of Data Elements \(ABAP Keyword Documentation\)](#)

[Adding a Development Object to a Transport Request \[page 164\]](#)

### 5.1.6.2.4 Providing Supplementary Documentation

You can provide additional supplement documentation for each data element.

## Context

You create supplementary documentation for a data element in order to provide individual program-specific and dynpro-specific field help.

## Procedure

1. In the toolbar of the *Field Labels* section, choose the  *Supplement Documentation* button.
2. In the dropdown listbox, choose *New...* to create new supplementary documentation.

The *New Supplementary Documentation* dialog is opened.

3. Enter a four-digit *Supplement ID*.
4. Confirm with *Finish*.

The documentation editor is opened in the integrated *SAP GUI*. Here you can add your individual documentation.

## Results

The supplementary documentation has been created.

### i Note

To display your supplementary documentation on a dynpro field, you must assign the supplement ID to the corresponding dynpro field in the `THLPF` database table.

## Related Information

[Creating Data Elements \[page 219\]](#)

[Editing Data Elements \[page 220\]](#)

## 5.1.6.3 Working with Structures

You can create and work with structures in ABAP Development Tools (ADT) using the *Structure Editor*.

## Related Information

[ABAP Dictionary Editors \[page 63\]](#)

[Syntax of ABAP Dictionary Objects \[page 751\]](#)

[Creating Structures \[page 225\]](#)

[Creating Append Structures \[page 226\]](#)

[Structures \(ABAP Keyword Documentation\)](#)

## 5.1.6.3.1 Creating Structures

### Prerequisites

- You have identified the ABAP package in which the development object is going to be created.

### → Tip

If you have not yet already done so, add this package to the Favorites of your ABAP project. See also:

[Adding a Favorite Package \[page 137\]](#)

## Context

You create structures to define a structured type that contains other data types as components.

## Procedure

1. In the *Project Explorer*, select the relevant *Package* node.
2. Open the context menu and choose    *Dictionary*.
3. Select *Structure* to launch the creation wizard.
4. Enter the *Name* and *Description* for the structure to be created.
5. Choose *Next*.
6. Assign a transport request.
7. Start the creation with *Finish*.

## Results

The inactive version of a structure is created and stored in ABAP Dictionary. In the *Project Explorer*, the new structure is added to the *Dictionary* folder of the corresponding package.

The *Structure Editor* is opened. Here, the initially generated source code is displayed and ready for editing.

## Related Information

- [Structures \[page 751\]](#)
- [Editing Structures \[page 228\]](#)
- [Transport Request \[page 108\]](#)
- [Tools for ABAP Development \[page 19\]](#)
- [Adding a Favorite Package \[page 137\]](#)
- [Creating Append Structures \[page 226\]](#)
- [Adding a Development Object to a Transport Request \[page 164\]](#)

## 5.1.6.3.1.1 Creating Append Structures

### Prerequisites

- You can create an append structure only on the basis of the existing structure.

## Context

You create append structures to add components or component extensions to an existing structure without modifying the latter:

## Procedure

1. Trigger the creation of an append structure.

### **i** Note

For this, you have, for example, the following possibilities:

- By opening the context menu on the base structure in the *Project Explorer*: Then, select *New Append Structure*.
- By opening the base structure in its *ABAP Dictionary editor*: Then, in its `define type` statement, get the quick assist proposals by choosing **Quick Fix** from the context menu (shortcut `Ctrl 1`) or by using the *Quick Assist* view. Apply **Create append structure for**.
- By adding a code template: To do this, select the complete source code of the base structure and add the `appendStructure` code template. Note that you rename the append structure after adding the code template. Otherwise, a syntax error is displayed when you save.

The *Creation Wizard* is opened.

2. Enter the *Name* and *Description* for the append structure to be created.

### **i** Note

The base structure is automatically selected and cannot be changed.

3. Choose *Next*.
4. Assign a transport request.
5. Start the creation with *Finish*.

## Results

The inactive version of an append structure is created and stored in ABAP Dictionary. In the *Project Explorer*, the new append structure is added to the *Dictionary* folder of the corresponding package.

The source-based ABAP Dictionary editor is opened. Here, the initially generated source code is displayed and ready for editing.

The append structure is introduced with the structure annotations of the base structure, followed by the `extend type ... with` keywords. Within the curly brackets, you can add the components that you want to add on to the base structure.

## Related Information

[Creating Structures \[page 225\]](#)

[Append Structures \[page 771\]](#)

[Code Templates for ABAP Dictionary Objects \[page 65\]](#)

### 5.1.6.3.2 Editing Structures

In the ABAP Dictionary editor for structures, you can use most of the functions you are already used to for development objects.

## Context



Example of functionalities that are provided in a source code editor for structures

The following table gives you an overview of the supported features:

Feature types	Features	Key Shortcuts	Availability
Standard	Editing [page 294]		✓
	Deleting [page 183]		✓
	Activation [page 70]	<code>Ctrl</code> + <code>F3</code>	✓
	Duplicating [page 173]		✓
Search	Searching Usages (Where-Used) [page 293]	<code>Ctrl</code> + <code>Shift</code> + <code>G</code>	✓
		<code>Ctrl</code> + <code>H</code>	✓
Convenience	Navigation	<code>F3</code>	✓
	Formatting [page 307]	<code>Shift</code> + <code>F1</code>	✗
	Outline [page 80]		✓
	Quick Outline [page 338]	<code>CTRL</code> + <code>O</code>	✓
	Code Completion [page 298]	<code>Ctrl</code> + <code>Space</code>	✓
	Syntax Highlighting [page 176]		✓
	Automatic Syntax Check		✓
	Element Info [page 33]	<code>F2</code>	✓
	Quick Assists [page 342]	<code>Ctrl</code> + <code>I</code>	✓
Others	Version History [page 208]		✓

Feature types	Features	Key Shortcuts	Availability
	Comparing Source Code [page 207]		✓
	Share Link (ADT link only)		✓
	Context-sensitive syntax documentation	F1	✓

## Related Information

[Structures \[page 751\]](#)

[Creating Structures \[page 225\]](#)

## 5.1.6.4 Working with Database Tables

You can create and work with database tables in ABAP Development Tools (ADT) using the [Database Table Editor](#).

## Related Information

[ABAP Dictionary Editors \[page 63\]](#)

[Creating Database Tables \[page 231\]](#)

[Editing Database Tables \[page 232\]](#)

[Database Tables \[page 772\]](#)

[Creating Append Structures \[page 226\]](#)

## 5.1.6.4.1 Creating Database Tables

A database table represents the display format for data in a relational database.

### Prerequisites

- You have identified the ABAP package in which the development object is going to be created.

#### → Tip

If you have not yet already done so, add this package to the Favorites of your ABAP project. See also: [Adding a Favorite Package \[page 137\]](#)

### Context

You want to create the database-independent definition of a database table.

### Procedure

1. In the *Project Explorer*, select the relevant *Package* node.
  2. Open the context menu and choose    *Dictionary*.
  3. Select *Database Table* to launch the creation wizard.
- The *creation wizard* is opened.
4. Enter the *Name* and *Description* for the database table to be created.
  5. Choose *Next*.
  6. Assign a transport request.
  7. Start the creation with *Finish*.

### Results

The inactive version of a database table is created and stored in ABAP Dictionary. In the *Project Explorer*, the new database table is added to the *Dictionary* folder of the corresponding package.

When the table is created in the database, the technical settings of the table are created automatically with the default values. In the Project Explorer, you can find technical setting in the *Technical Table Settings* folder below the database table.

The *Database Table Editor* is opened. Here, the source code initially generated is displayed and ready for editing.

You now need to continue, for example, defining the annotations, adding fields. To do this, you can benefit from the content assist (`Ctrl` + `Space`).

To edit the technical settings of the table open the *Technical Table Settings Editor*.

## Related Information

[ABAP Dictionary Editors \[page 63\]](#)

[Creating Append Structures \[page 226\]](#)

[Editing Database Tables \[page 232\]](#)

[Database Tables \[page 772\]](#)

## 5.1.6.4.2 Editing Database Tables

### Context

In the following figure, you see an editor for database tables in ADT.

```

1  @EndUserText.label : 'Demo database table for employees'
2  @AbapCatalog.enhancementCategory : #NOT_EXTENSIBLE
3  @AbapCatalog.tableCategory : #TRANSPARENT
4  @AbapCatalog.deliveryClass : #A
5  @AbapCatalog.dataMaintenance : #LIMITED
6  define table employees_demo {
7      @EndUserText.label : 'Client'
8      key mandt : abap.clnt not null;
9      @EndUserText.label : 'User Name'
10     key userid : abap.char(12) not null;
11     @EndUserText.label : 'Last Name'
12     lastname : abap.char(20) not null;
13     @EndUserText.label : 'Surname'
14     firstname : abap.char(20) not null;
15     @EndUserText.label : 'Street'
16     street : abap.char(40) not null;
17     @EndUserText.label : 'ZIP Code'
18     zipcode : abap.char(5) not null;
19     @EndUserText.label : 'Residence'
20     city : abap.char(40) not null;
21     @EndUserText.label : 'Country'
22     country : taxl
23     @EndUserText.label :
24     salary : abap.de
25
26
27 }

```

Annotations for database properties

Table fields

Overview ruler that highlights errors, warnings, and so on

Error marker

Marker bar

Proposal popup for code completion

Example of features that are provided in a source code editor for database tables

The following features are supported:

Feature types	Features	Key Shortcuts	Availability
Standard	<a href="#">Editing [page 294]</a>		✓
	<a href="#">Deleting [page 183]</a>		✓

Feature types	Features	Key Shortcuts	Availability
	Activation [page 70]	<code>Ctrl</code> + <code>F3</code>	✓
	Duplicating [page 173]		✓
Search	Searching Usages (Where-Used) [page 293]	<code>Ctrl</code> + <code>Shift</code> + <code>G</code>	✓
		<code>Ctrl</code> + <code>H</code>	✓
Convenience	Navigation	<code>F3</code>	✓
	Formatting [page 307]	<code>Shift</code> + <code>F1</code>	✗
	Outline [page 80]		✓
	Quick Outline [page 338]	<code>CTRL</code> + <code>O</code>	✓
	Code Completion [page 298]	<code>Ctrl</code> + <code>Space</code>	✓
	Syntax Highlighting [page 176]		✓
	Automatic Syntax Check		✓
	Element Info [page 33]	<code>F2</code>	✓
	Quick Assists [page 342]	<code>Ctrl</code> + <code>1</code>	✓
Others	Version History [page 208]		✓
	Comparing Source Code [page 207]		✓
	Share Link [page 166]		✓
	Context-sensitive syntax documentation	<code>F1</code>	✓

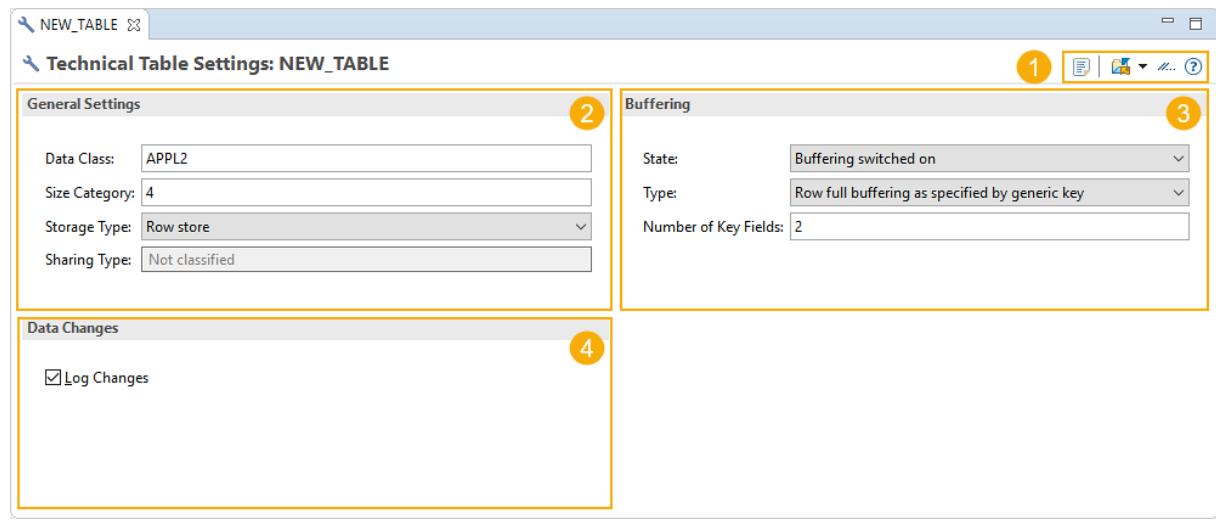
## Related Information

[ABAP Dictionary Editors \[page 63\]](#)

[Creating Database Tables \[page 231\]](#)

### 5.1.6.4.3 Editing Technical Table Settings

You can edit technical settings of the database table using the *Technical Settings Editor*:



#### 1. Toolbar

The toolbar contains actions such as opening the dictionary log, opening the object in another project, sharing ADT link, and opening documentation.

#### 2. General Settings

In this section you can define the *Data Class*, *Size Category*, *Storage Type*, and *Sharing Type* of the table.

#### 3. Buffering

In the buffering section you can specify whether the table is buffered and how. From the dropdown menu of the *State* field, select one of the following buffering states.

- Buffering not allowed

- Buffering switched on
- Buffering allowed but switched off

If the buffering is allowed or switched on, specify the buffering type. From the dropdown menu of the *Type* field select one of the following types.

- Single entries of table were buffered
- Row full buffering as specified by generic key (for this entry specify the number of key fields.)
- Full table is passed to buffer
- No buffering

## 4. Data Changes

Enable or disable the *Log Changes* checkbox to specify whether changes to the table's records are logged.

### Related Information

[Creating Database Tables \[page 231\]](#)

[Editing Database Tables \[page 232\]](#)

## 5.1.6.5 Working with Table Types

### Context

You can create and edit dictionary table types in ADT.

### Related Information

[Creating Table Types \[page 237\]](#)

[Editing Table Types \[page 238\]](#)

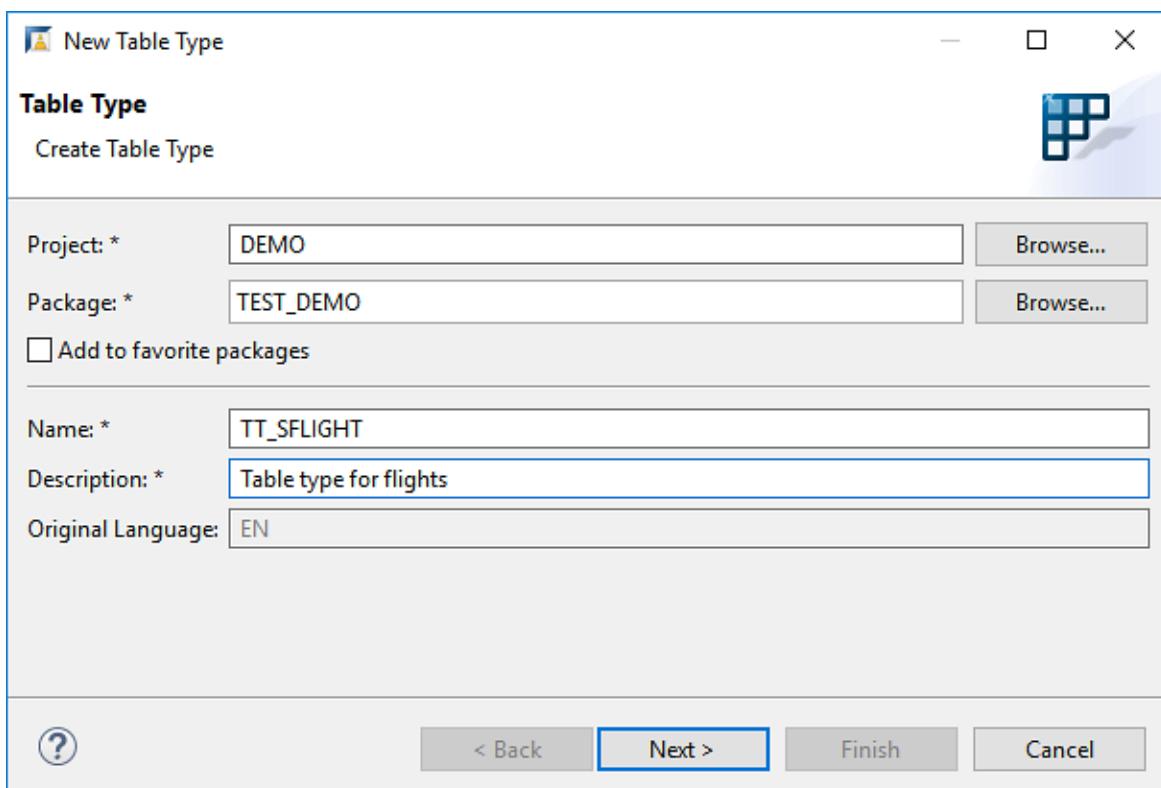
## 5.1.6.5.1 Creating Table Types

### Context

To create a dictionary table type, proceed as follows:

### Procedure

1. In Project Explorer, select the relevant package node.
2. Open the context menu and select **New > Other ABAP Repository Object > Dictionary > Table Type**.
3. Specify the *Project* and *Package*.
4. Enter the *Name* and *Description* for the table type you want to create.



Creation wizard for a table type

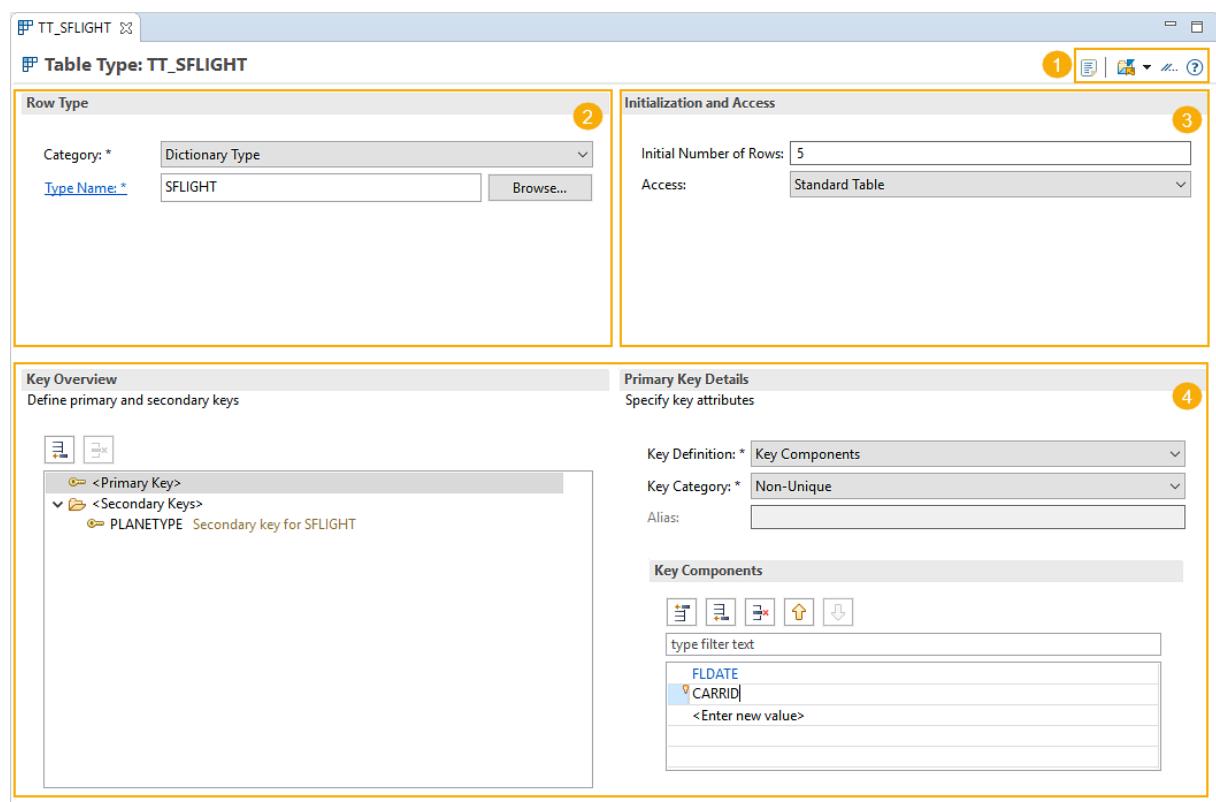
5. Select *Next*.
6. Assign a transport request.
7. Select *Finish*.

## Related Information

[Editing Table Types \[page 238\]](#)

### 5.1.6.5.2 Editing Table Types

You can edit the existing table type in the *Table Type Editor*.



Editor for table types

#### 1. Toolbar

The toolbar contains actions such as opening the dictionary log, opening the object in another project, sharing ADT link, and opening documentation.

#### 2. Row Type

In this section, you can define the data type of a row in the internal table.

In the dropdown list of the *Category*, select one of the following categories:

- Dictionary type
- Predefined type
- Reference to predefined type
- Reference to dictionary type
- Reference to class/interface
- Range table on predefined type
- Range table on data element

In the field *Type Name*, select the type. For example, if you chose dictionary type as a category, enter a dictionary object, such as a database table, view, or data element in the field. You can use functionality or use the *Browse* button to search for the type.

### Range Tables as Special Row Types

For *Range Table on Predefined Type*, select the data type in the *Data Type* field. Make sure you select the same data type as you use in the range structure. You can create a range structure by clicking the link *Range Structure* or select the existing one using the *Browse* button. After selecting the range structure, you can navigate to it by clicking the *Range Structure* link.

For *Range Table on Data Element*, you can create a data element by clicking the link *Type Name* and a new range structure by clicking the link *Range Structure* or select the existing objects using the *Browse* button.

#### → Tip

You can navigate to the type or to the range structure by clicking the *Type Name* or *Range Structure* link.

## 3. Initialization and Access

In this section, you can define the initial number of rows and options for accessing the data in the internal table. Select the access mode in the dropdown list of the *Access* field.

## 4. Key Overview and Key Details

In this section, you can define primary and secondary keys and specify key attributes.

Select the key on the left side in the *Key Overview* section to specify the attributes on the right side in the *Key Details* section.

You can add secondary keys as follows:

- Using the key shortcut `Shift` + `Enter`
- Using the button 
- From the context menu, select *Create secondary key* on the *<Secondary Keys>* entry.

In the section *Key Details* on the right side, you can specify a key definition and key category for the primary key. For the secondary key, you can specify a name, description, key definition, and key access.

If you selected *Key Components* in the field *Key Definition*, you can select the components for the key in the *Key Components* section using content assist functionality.

## Related Information

[Creating Table Types \[page 237\]](#)

### 5.1.6.6 Working with Lock Objects

You can create and work with lock objects in ABAP Development Tools (ADT) using the *Lock Object Editor*.

## Related Information

[Creating Lock Objects \[page 240\]](#)

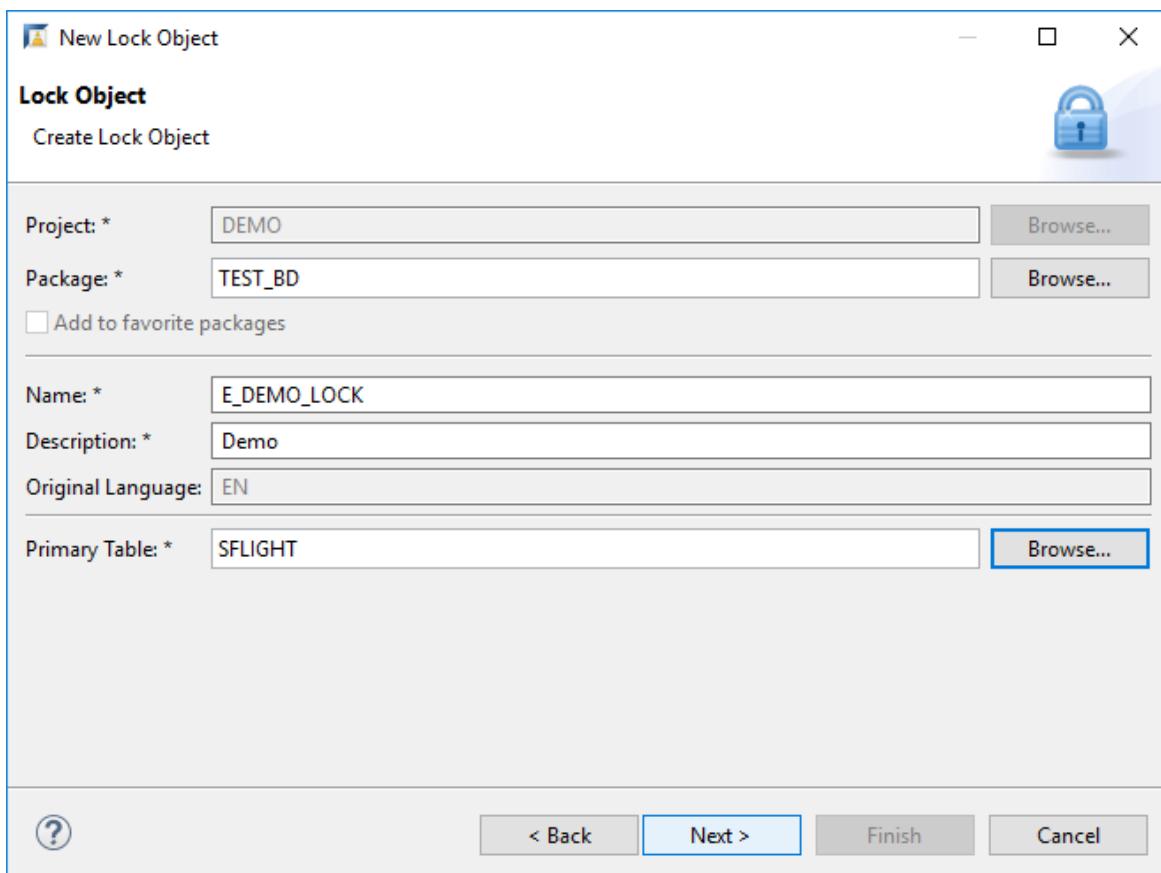
[Editing Lock Objects \[page 242\]](#)

#### 5.1.6.6.1 Creating Lock Objects

You can create lock objects in ABAP Development Tools (ADT) using the creation wizard.

To create a lock object, proceed as follows:

1. In the *Project Explorer*, select the relevant *Package* node.
2. Open the context menu and select ► *New* ► *Other ABAP Repository Object* ► *Dictionary* ► *Lock Object* ▾.
3. In the wizard that opens, the *Project* and *Package* are automatically inserted. You can change the *Package* if needed.
4. Enter the *Name* and *Description* for the lock object you want to create.
5. Select a primary table. You can change it later if needed.



Creation wizard for lock objects

6. Select *Next*.
7. Assign a transport request.
8. Select *Finish*.

## Result

The inactive version of the lock object is created. In the *Project Explorer*, the new lock object is added to the *Dictionary* folder.

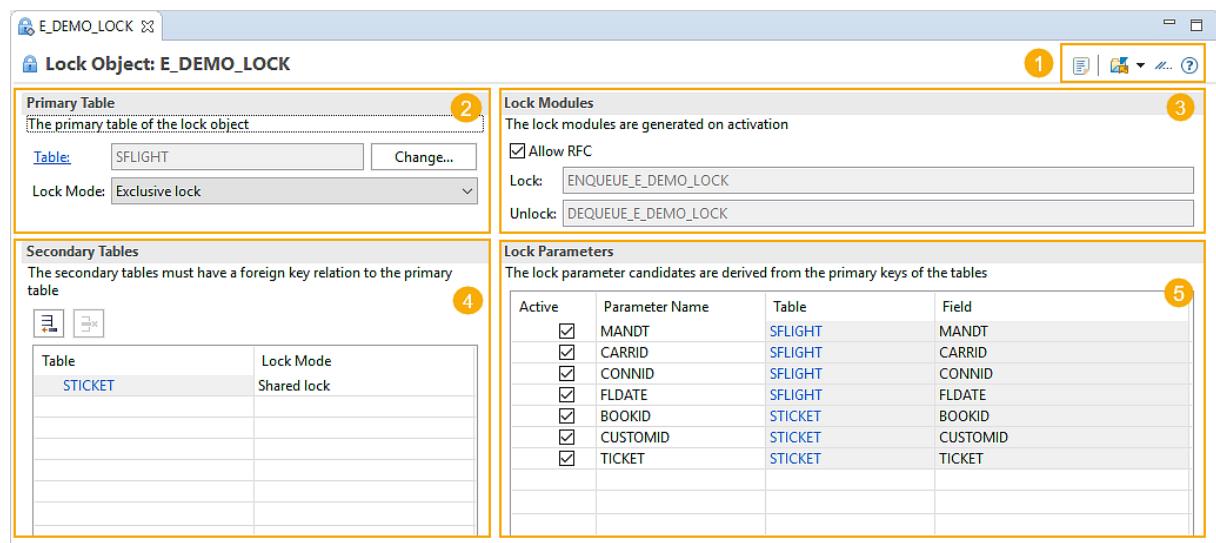
The lock object editor is open and ready for editing now.

## Related Information

[Editing Lock Objects \[page 242\]](#)

## 5.1.6.6.2 Editing Lock Objects

The following figure shows the editor for lock objects.



Editor for lock objects

The editor provides the following information:

### 1. Toolbar

The toolbar contains actions such as opening the dictionary log, opening the object in another project, sharing ADT link, opening documentation.

### 2. Primary Table

The primary table, which you selected in the creation wizard is automatically inserted. If needed, you can change the table. Navigate to the selected table by clicking the link *Table*:

In the dropdown menu, select the lock mode.

### 3. Lock Modules

You can enable or disable accessing the lock modules from other systems with *Allow RFC* checkbox. The lock modules are automatically generated and inserted in the *Lock*, *Unlock* fields after activating the lock object.

## 4. Secondary Tables

You can maintain the secondary tables, which have a foreign key relation to the primary table. Use  and  buttons for inserting and deleting the tables.

## 5. Lock Parameters

Primary keys of the selected tables are displayed here. You can switch off the lock parameters by clicking the checkboxes in the *Active* column. If the parameter names of the primary and secondary tables coincide, rename the corresponding entries.

Select a lock mode for the secondary table.

## Related Information

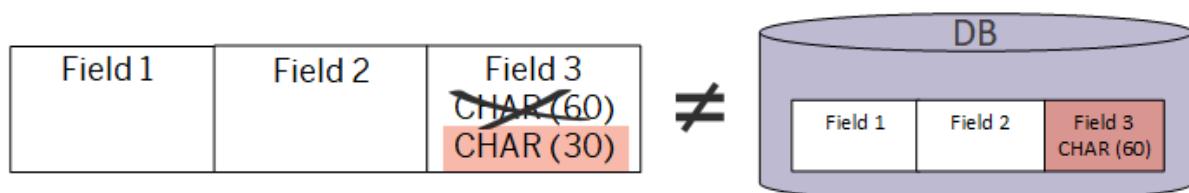
[Creating Lock Objects \[page 240\]](#)

### 5.1.6.7 Adjusting Database Tables

When you change a database table in the ABAP Dictionary, the database table needs to be adjusted accordingly.

This can be caused by direct changes in the database table or indirect changes in domains, data elements, or structures that are used by the database table.

#### Direct Changes



If you change the type of the table field directly in the database editor, the modified field type in the editor does not match the field type in the database. Apply a quickfix to adjust and activate the database table with data conversion or data deletion.

```

1 @EndUserText.Label : 'Table with cars2'
2 @AbapCatalog.enhancementCategory : #NOT_EXTENSIBLE
3 @AbapCatalog.tableCategory : #TRANSPARENT
4 @AbapCatalog.deliveryClass : #A
5 @AbapCatalog.dataMaintenance : #LIMITED
6 define table cars {
7   key brand : abap.char(10)
8   color   : dmo_color;
9   model   : abap.char(30);
10  }
11

```

## Indirect Changes

If you change the type of the table field indirectly, for example in the data element or domain editor, you cannot activate the object due to a mismatch of the field type in the dependent table. In the Problems view, select the error with a quick fix and type **Ctrl** + **1**. In the opened dialog, select a quickfix to adjust the table and to activate the objects.

The type of the field has been changed

Ctrl + 1

## 5.1.7 Working with Number Range Objects

You can create and edit *Number Range Objects* in *Number Range Object Editor*.

### Related Information

[Creating Number Range Objects \[page 245\]](#)

[Editing Number Range Objects \[page 247\]](#)

### 5.1.7.1 Creating Number Range Objects

#### Context

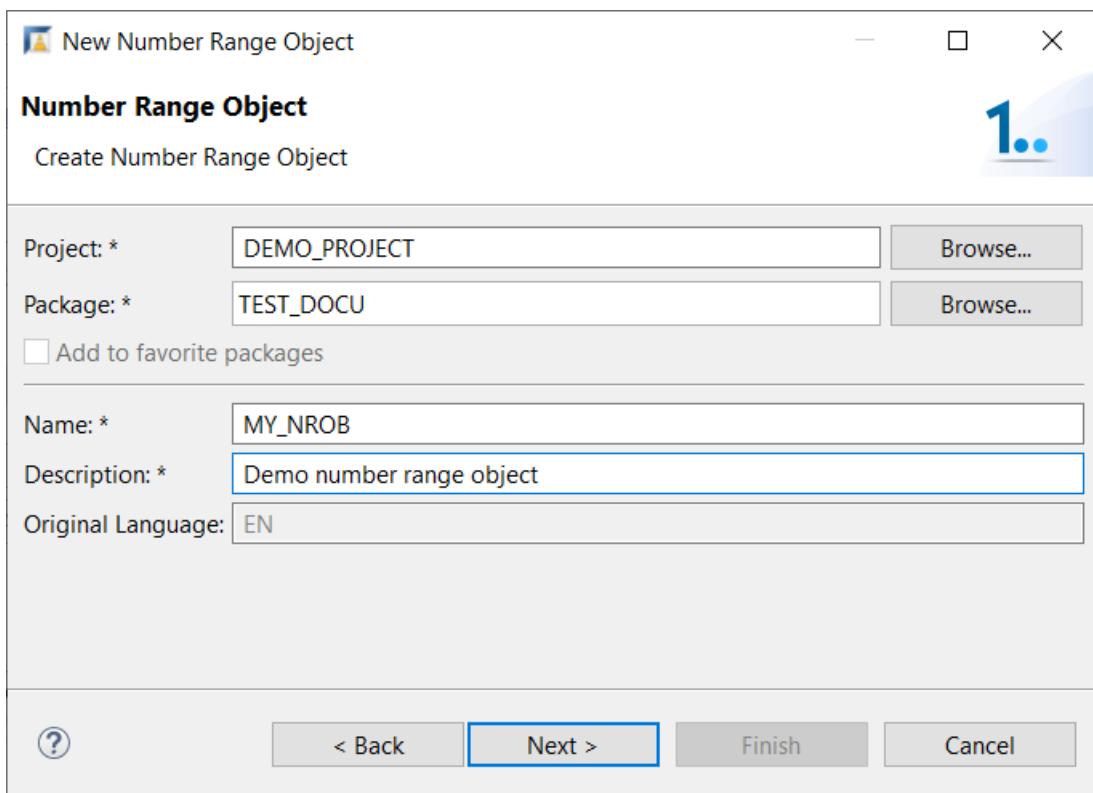
You can create number range objects in ABAP Development Tools (ADT) using the creation wizard.

#### Procedure

1. You can start the creation wizard from the main menu, toolbar, Project Explorer, or using a shortcut. For more information, see [Creating Development Objects \[page 156\]](#).
2. In the creation wizard, fill in the project and package, if not already filled.
3. Enter the name and description of a range number object.

#### i Note

The name cannot be longer than 10 characters.



Creation Wizard for Number Range Objects

4. Select *Next*.
5. Assign a transport request.
6. Select *Finish*.

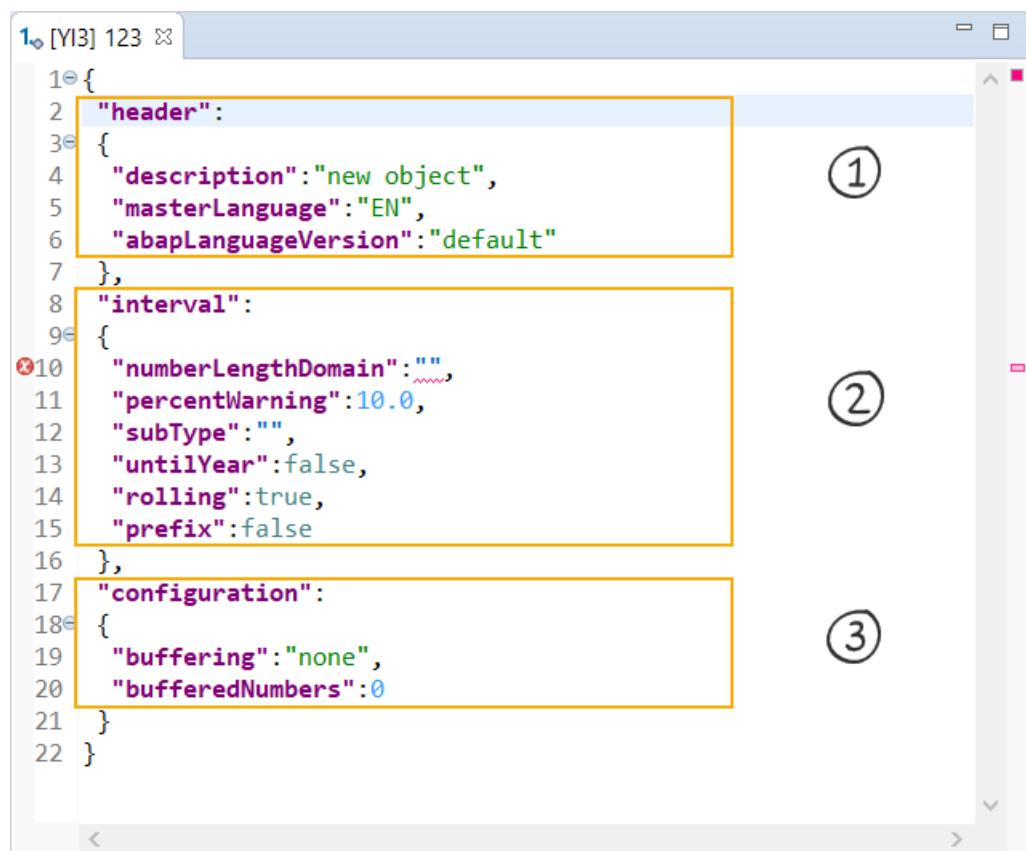
A new number range object is created and opened in the *Number Range Object Editor*.

## Related Information

[Editing Number Range Objects \[page 247\]](#)

## 5.1.7.2 Editing Number Range Objects

### Context



```
1 [YI3] 123
1① {
2   "header": {
3     "description": "new object",
4     "masterLanguage": "EN",
5     "abapLanguageVersion": "default"
6   },
7   "interval": {
8     "numberLengthDomain": "...",
9     "percentWarning": 10.0,
10    "subType": "",
11    "untilYear": false,
12    "rolling": true,
13    "prefix": false
14  },
15  "configuration": {
16    "buffering": "none",
17    "bufferedNumbers": 0
18  }
19 }
20 }
```

Number Range Object Editor

1. Section `header` includes Header Data Information for defining the Number Range Object. `description` can be changed in the *Properties* view. `masterLanguage` and `abapLanguageVersion` are information only and can't be changed.
2. In section `interval`, define the properties of intervals of the number range object. For `numberLengthDomain` add an own data element of your component. If needed change value `percentWarning`. It's a percentage of numbers remaining in a number range, upon reaching which in number assignment a warning is given. Assuming an interval from 1 to 1000 and a percentage of 10 (%) is used a notification will be triggered if number 900 has been reached.  
Add for a data element if the number range intervals should distinguish between further subobjects.  
Set `untilYear` to true if the number range intervals are distinguished according to To-fiscals.  
Set `rolling` to false to prevent the number range object intervals from automatically starting from the beginning at the upper limit.  
If `prefix` is set to true determined numbers consist of the prefix (name of subobject) and the numbers.
3. Section `configuration` defines properties of the number range object.

For `transactionId` add a transaction name. It creates a parameter transaction to start the maintenance of number range intervals. It has to be ignored in an ABAP CP system.

Choose for `buffering` a buffer type `none` for no buffering, `mainBuffer` for buffering via main memory or `parallel` for parallel buffering.

In case of parallel and main memory buffering add a number for `bufferedNumbers`. It determines how many numbers are reserved in buffer for the intervals.

## Related Information

[Creating Number Range Objects \[page 245\]](#)

## 5.1.8 Working with Business Services

In the context of ABAP Environment, Business Services consist of Service Consumption Model.

The Service Consumption Model helps to create proxies for remote service. These proxies can be consumed to trigger various operations on remote service.

### 5.1.8.1 Creating Service Consumption Model

Create a service consumption model using service metadata file.

#### Prerequisites

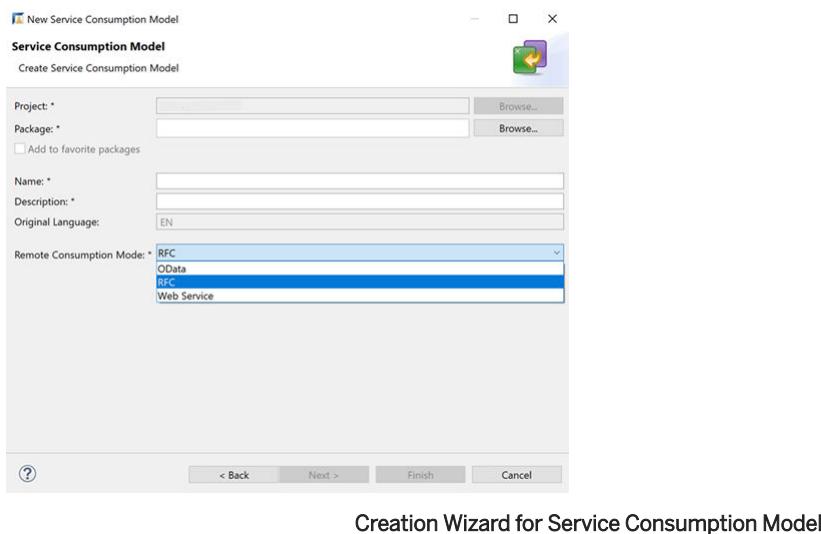
- You need the standard developer authorization profile to create ABAP development objects.
- You have the metadata for the remote OData, RFC, or Web service stored either in an EDMX, XML, or WSDL file on your local system.

#### Context

You can create a service consumption model with an EDMX for OData, XML file for RFC, or WSDL file for Web service that contains the service metadata.

## Procedure

1. In your ABAP project, open the context menu and choose **New > Other ABAP Repository Object > Business Services > Service Consumption Model** to launch the creation wizard.
2. Enter a name and description. In addition to the *Project* and *Package*.
3. From the drop-down list of the *Remote Consumption Mode*, select OData, RFC, or Web Service.
4. Choose *Next*.



Creation Wizard for Service Consumption Model

## Related Information

[Generating Proxies for Remote OData Service \[page 249\]](#)

[Generating Proxies for Remote Web Service \[page 253\]](#)

### 5.1.8.2 Generating Proxies for Remote OData Service

Create a service consumption model to generate proxies for remote OData service.

## Prerequisites

- You need the standard developer authorization profile to create ABAP development objects.
- You have the metadata for the remote OData service stored in an EDMX or XML file on your local system.

## Context

You can create a service consumption model using service metadata file.

## Procedure

1. In your ABAP project, open the context menu and choose **► New ► Other ABAP Repository Object ► Business Services ► Service Consumption Model** to launch the creation wizard..
2. Enter a name and description. In addition to the *Project* and *Package*.
3. From the drop down list of the *Remote Consumption Mode*, select OData.
4. Choose *Next*.
5. Search the *Service Metadata File* to generate OData Consumption proxy.



6. Enter the *Prefix* value.
7. The list of ABAP names with prefix values are displayed in the next page. You can remove the prefix provided or continue with the generation of the proxy objects.
8. From the list of entity sets from the edmx file, select the entity sets you want to generate and choose *Next*.

### i Note

- For update scenario, Etag support should be available. If Etag support is marked in your edmx file, the Etag support checkbox is selected by default. You cannot change this selection in the wizard if it selected through the edmx file.
- You can edit the ABAP artifact name for the entity set that you've selected for generation.
- Entity sets that have issues cannot be selected for generation. You need to fix the issue in the edmx file.

 New Service Consumption Model

**Define Entity Set**

Select entity sets for generation, edit the ABAP artifact name and select ETag support. Entity sets with issues are not generated.

	Service Entity Set	ABAP Artifact Name	ETag...	Issue
<input checked="" type="checkbox"/>	A_SalesOrderWith...	ZA_SALESORDERWITHO...	<input checked="" type="checkbox"/>	
<input checked="" type="checkbox"/>	A_SalesOrderWith...	ZA_SALESORDERWITHO...	<input type="checkbox"/>	
<input checked="" type="checkbox"/>	A_SlsOrdWthoutC...	ZA_SLSORDWTHOUTCH...	<input type="checkbox"/>	
<input checked="" type="checkbox"/>	A_SlsOrdWthoutC...	ZA_SLSORDWTHOUTCH...	<input type="checkbox"/>	
<input checked="" type="checkbox"/>	A_SlsOrdWthoutC...	ZA_SLSORDWTHOUTCH...	<input type="checkbox"/>	

**Select All** **Deselect All**

**?** **< Back** **Next >** **Finish** **Cancel**

9. The *ABAP Artifact Generation List* segregated by service definition, data definition and behaviour definition is displayed. Choose *Next*.

The screenshot shows the 'ABAP Artifact Generation List' dialog box. At the top, there is a title bar with the text 'New Service Consumption Model' and a close button. Below the title bar, the text 'ABAP Artifact Generation List' is displayed, followed by a sub-instruction 'List of ABAP artifacts that are going to be generated'. On the right side of the dialog, there is a small icon of a green and purple square with a yellow arrow. The main area of the dialog is a table with the following columns: 'ABAP Artifact Name', 'Type', and 'ETag S...'. The table contains the following data:

ABAP Artifact Name	Type	ETag S...
ZRN_SRVC_TEST_3	Service Definition	
ZA_SALESORDERWITHOUT	Data Definition	<input checked="" type="checkbox"/>
ZA_SALESORDERWITHOUT	Data Definition	<input type="checkbox"/>
ZA_SLSORDWTHOUTCHRG	Data Definition	<input type="checkbox"/>
ZA_SLSORDWTHOUTCHRG	Data Definition	<input type="checkbox"/>
ZA_SLSORDWTHOUTCHRG	Data Definition	<input type="checkbox"/>

Below the table is a search bar labeled 'type filter text'. At the bottom of the dialog, there are several buttons: a question mark icon, '< Back', 'Next >', 'Finish' (which is highlighted with a blue border), and 'Cancel'.

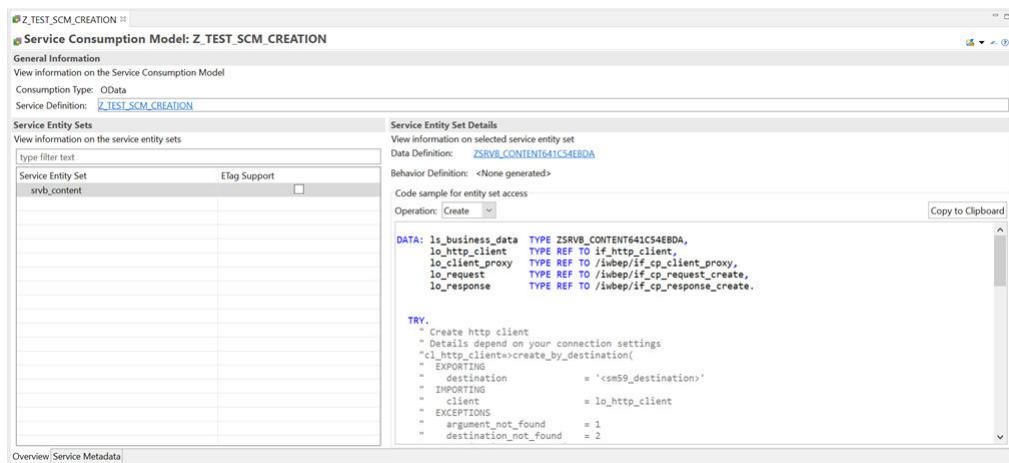
10. Assign a transport request.

11. Choose *Finish*.

## Results

In the selected package, the ABAP back end system creates a service consumption model.

In the *Project Explorer*, the new service consumption model is added to the *Business Services* folder of the corresponding package node. As a result of this creation procedure, the form editor will be opened. Here, you can view the list of generated artifacts.



Service Consumption Model Editor

### i Note

Service Consumption Model for OData remote consumption is an Active only object. It cannot be activated or deactivated explicitly. You can only edit the *Description* of the Service Consumption Model in the *Properties* page.

## 5.1.8.3 Generating Proxies for Remote Web Service

Create a service consumption model to generate proxies for the remote Web service.

### Prerequisites

- You need the standard developer authorization profile to create ABAP development objects.
- You have the metadata for the remote Web service stored in a WSDL file on your local system. For more information, see [Consuming a Web Service](#).

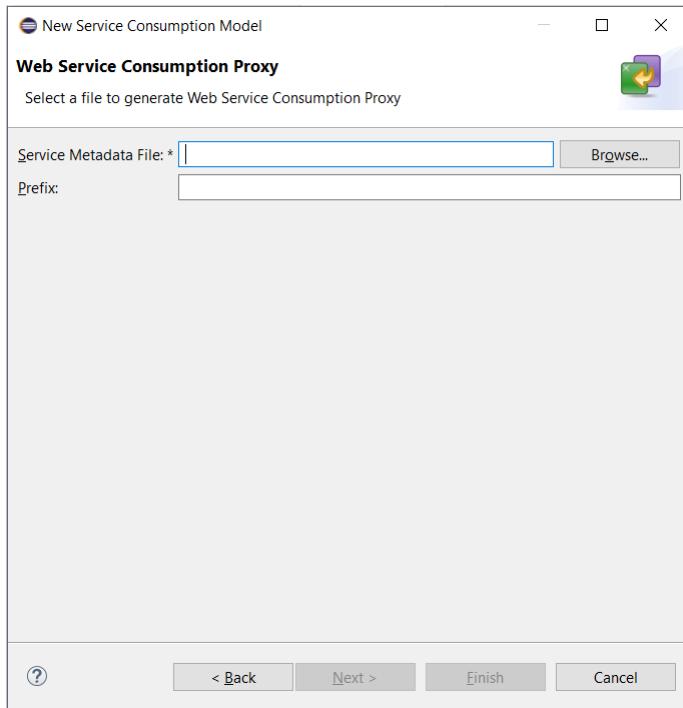
### Context

You can create a service consumption model using WSDL file.

### Procedure

1. In your ABAP project, open the context menu and choose     to launch the creation wizard..

2. Enter a name and description. In addition to the *Project* and *Package*.
3. From the drop down list of the *Remote Consumption Mode*, select Web Service.
4. Choose *Next*.
5. Search the *Service Metadata File* to generate Web service consumption proxy and enter the *Prefix*.

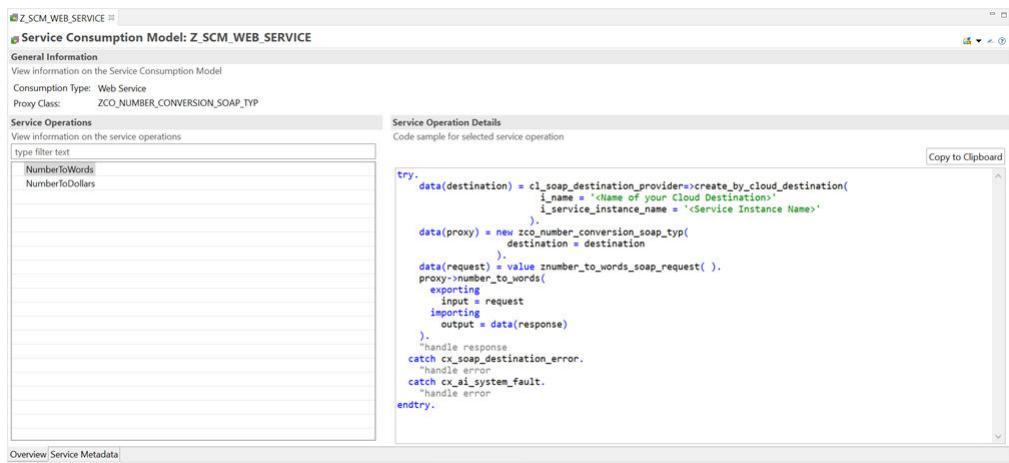


6. Choose *Next*.
7. Assign a transport request.
8. Choose *Finish*.

## Results

In the selected package, the ABAP back end system creates a service consumption model.

In the *Project Explorer*, the new service consumption model is added to the *Business Services* folder of the corresponding package node. As a result of this creation procedure, the form editor will be opened. Here, you can view the list of generated artifacts.



Service Consumption Model Editor

### i Note

Service Consumption Model for Web Service remote consumption is an Inactive object. To generate proxy artifacts, you must activate the Service Consumption Model. You can edit Service Consumption Model in the [Service Metadata](#) tab, here you can change the values specific to ABAP attributes or name attributes in ABAP elements.

## 5.1.8.4 Generating Proxies for Remote Function Call (RFC)

Create a service consumption model to generate proxies for remote function call (RFC).

### Prerequisites

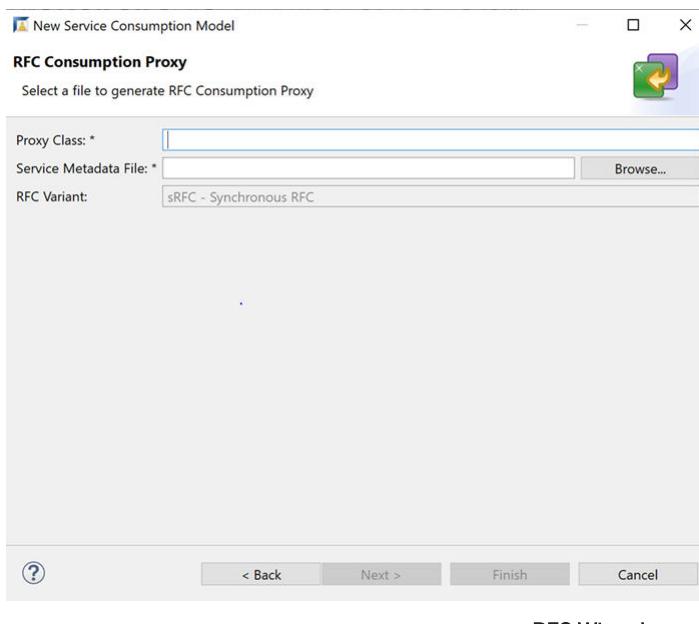
- You need the standard developer authorization profile to create ABAP development objects.
- You have the metadata for the remote function call stored in an XML file on your local system.

### Context

You can create a service consumption model using service metadata file.

## Procedure

1. In your ABAP project, open the context menu and choose **New** **Other ABAP Repository Object** **Business Services** **Service Consumption Model** to launch the creation wizard.
2. Enter a name and description. In addition to the *Project* and *Package*.
3. From the drop-down list of the *Remote Consumption Mode*, select **RFC**.
4. Choose **Next**.
5. Enter the *Proxy Class* and select the *Service Metadata File* to generate the RFC Consumption proxy.
6. Choose **Next**.

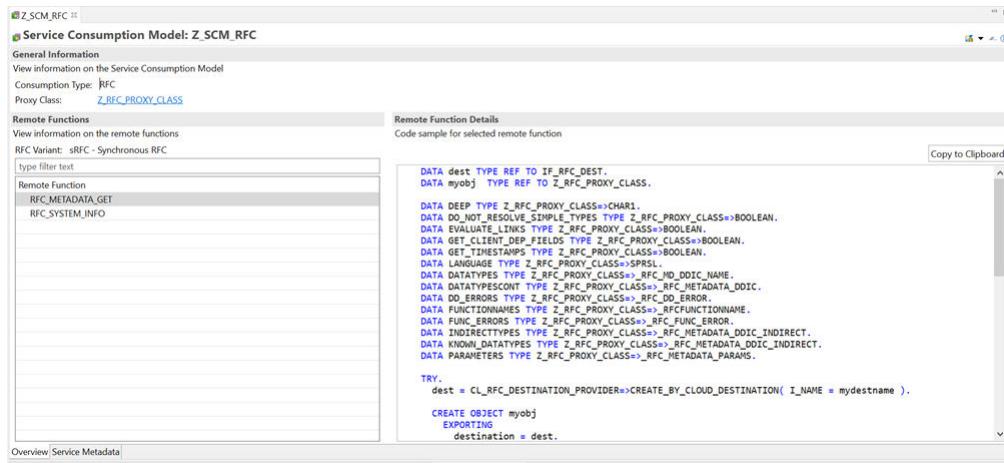


7. Assign a transport request.
8. Choose **Finish**.

## Results

In the selected package, the ABAP back-end system creates a service consumption model.

In the *Project Explorer*, the new service consumption model is added to the *Business Services* folder of the corresponding package node. As a result of this creation procedure, the form editor is opened. Here, you can view the list of remote functions.



Service Consumption Model Editor

## 5.1.8.5 Using Service Consumption Model Editor

Use the service consumption model editor to view the service metadata.

After you create a service consumption model, the editor is displayed. The following actions can be done here:

### For OData services

- View the service definition and the corresponding ABAP artifacts that have been generated.
- View the service metadata you provided using the EDMX file by selecting the *Service Metadata* tab.
- For EDMX files that contains service metadata for an OData service that supports transactional behavior (CRUD operations), the behavior definition is listed under the *Service Entity Set Details* section. For more information, see [.](#)
- View the code sample for performing CRUD operations on an entity set belonging to the remote OData service created through service consumption model. Each operation is displayed as code snippets which you can copy by choosing *Copy to Clipboard*.

For more information, see [Generating Proxies for Remote OData Service \[page 249\]](#)

### For Web Services

- View the generated proxy class. To generate and view the proxy artifacts, you must activate the Service Consumption Model.
- View and edit the service metadata you provided using the WSDL file by selecting the *Service Metadata* tab. You can only edit the ABAP related attributes and element tags.
- The *Service Metadata* tab displays WSDL file content with syntax coloring, outline view and marker occurrences
- View the code sample for performing operations related to the remote Web Service created through service consumption model. Each operation is displayed as code snippets which you can copy by choosing *Copy to Clipboard*.

For more information, see [Generating Proxies for Remote Web Service \[page 253\]](#)

## For Remote Function Call (RFC)

- View the generated proxy class. To generate and view the proxy artifacts, you must activate the Service Consumption Model.
- View the service metadata you provided using the XML file by selecting the *Service Metadata* tab.
- View the code sample to access the RFC. If the XML points to multiple functions, code sample is displayed for each function. Each operation is displayed as code snippets which you can copy by choosing *Copy to Clipboard*

For more information, see [Generating Proxies for Remote Function Call \(RFC\) \[page 255\]](#)

## 5.1.9 Documenting Development Objects

You can document development objects in the *Knowledge Transfer Document Editor*.

### Related Information

[Creating Knowledge Transfer Documents \[page 258\]](#)

[Editing Knowledge Transfer Documents \[page 259\]](#)

[Linking to Development Objects \[page 261\]](#)

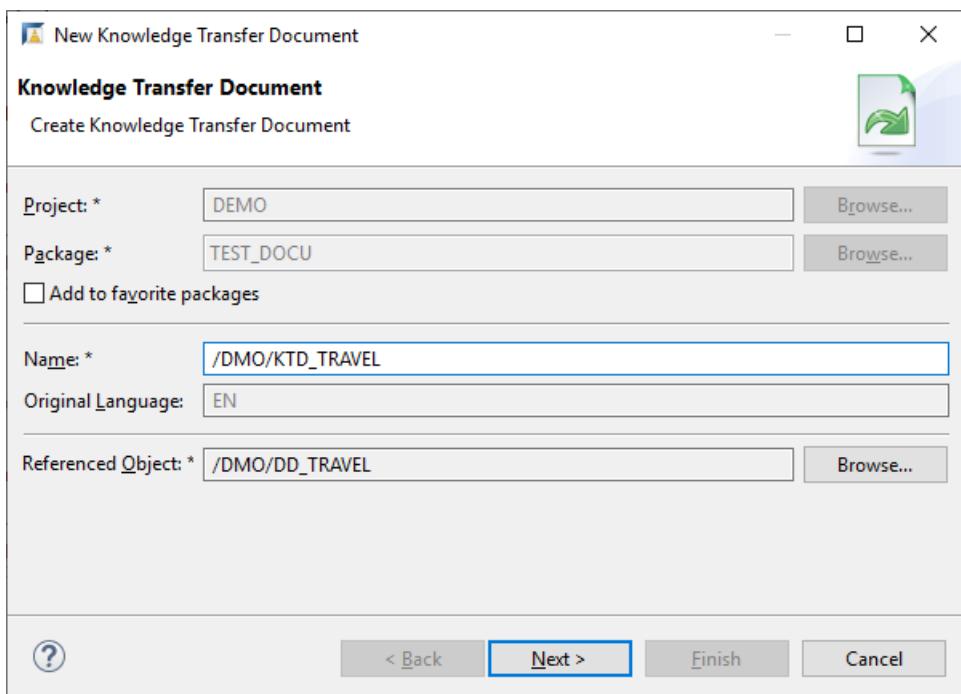
### 5.1.9.1 Creating Knowledge Transfer Documents

To create documentation for a development object, perform the following steps.

1. In the Project Explorer, select the object you want to document.
2. Open the context menu and select *New Knowledge Transfer Document*.
3. In the creation wizard, enter the name of the knowledge transfer document.

If needed, you can change the referenced object using the *Browse* button.

The knowledge transfer document must be created in the same package where the referenced object was created.



Creation Wizard of a Knowledge Transfer Document

4. Select *Next*.
5. Assign a transport request.
6. Select *Finish*.

#### Result

A knowledge transfer document is opened in the *Knowledge Transfer Document Editor*. You can now document the object and its element.

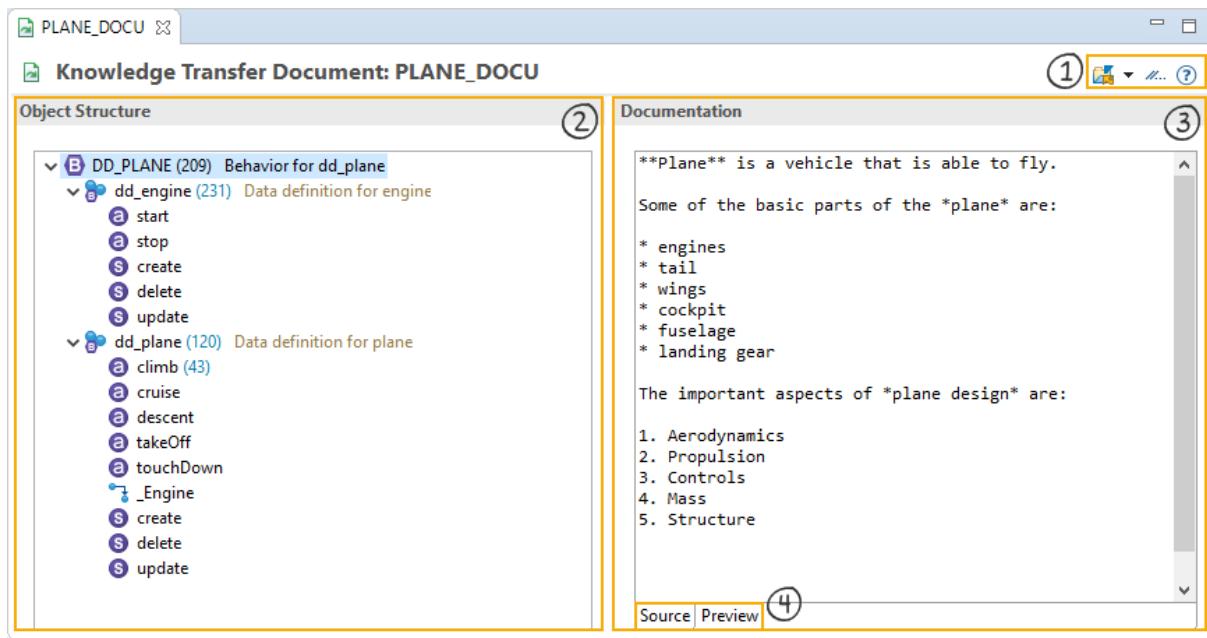
#### Related Information

[Editing Knowledge Transfer Documents \[page 259\]](#)

[Linking to Development Objects \[page 261\]](#)

[Knowledge Transfer Document \[page 30\]](#)

### 5.1.9.2 Editing Knowledge Transfer Documents



Knowledge Transfer Document Editor

## 1. Toolbar

In the toolbar, you can perform actions, such as opening an object in another project, sharing the link, or opening the help.

## 2. Object Structure

In this section, the object is displayed with its elements structured hierarchically as a tree.

The number in brackets next to an element describes the number of characters in the corresponding documentation and thus indicates that the element has already been documented. This number also gives you an impression about the amount of documentation provided.

From the context menu on each element, you can either open the element in its editor or copy the element name.

You can toggle a search field using **Ctrl** + **F**. Press **Ctrl** + **F** or **Esc** to hide the field again.

## 3. Documentation

This is where you can document the object and its elements using markdown syntax. Content assist functionality **Ctrl** + **Space** proposes available such as bold, italics, lists and links.

## 4. You can display an HTML preview of the documentation by selecting *Preview* tab.

```
**Plane** is a vehicle that is able to fly.

Some of the basic parts of the *plane* are:

* engines
* tail
* wings
* cockpit
* fuselage
* landing gear

The important aspects of *plane design* are:

1. Aerodynamics
2. Propulsion
3. Controls
4. Mass
5. Structure
```

[Source](#) [Preview](#)

**DD PLANE**

Plane is a vehicle that is able to fly.

Some of the basic parts of the *plane* are:

- engines
- tail
- wings
- cockpit
- fuselage
- landing gear

The important aspects of *plane design* are:

1. Aerodynamics
2. Propulsion
3. Controls
4. Mass
5. Structure

[Source](#) [Preview](#)

Display of Source and Preview

### Unassigned Texts

If a documented element was renamed or deleted, the documentation for the element is displayed under the *Unassigned Texts* node. From there, you can consider to copy the documentation to another element or to delete it using the context menu.

### Related Information

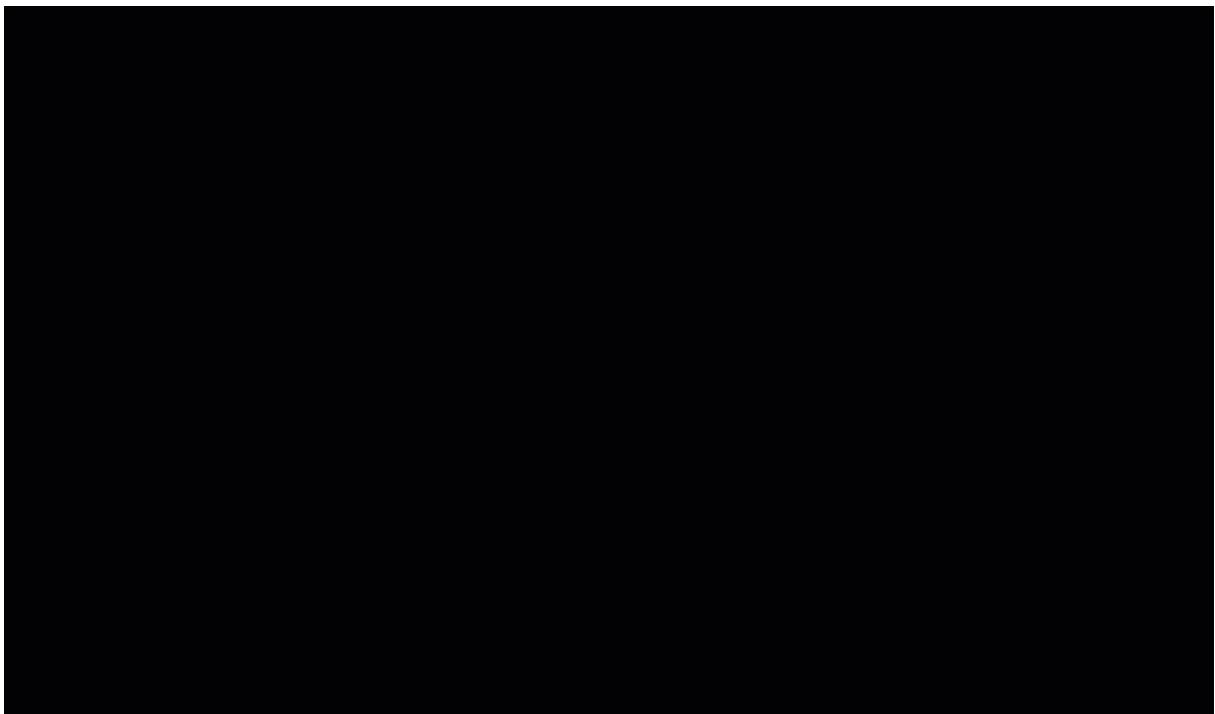
[Creating Knowledge Transfer Documents \[page 258\]](#)

[Linking to Development Objects \[page 261\]](#)

[Knowledge Transfer Document \[page 30\]](#)

### 5.1.9.3 Linking to Development Objects

In the *Knowledge Transfer Document Editor*, you can link to development objects and their elements. Use content assist functionality to complete the link as shown below.



Linking to development objects

## Related Information

[Creating Knowledge Transfer Documents \[page 258\]](#)

[Editing Knowledge Transfer Documents \[page 259\]](#)

[Knowledge Transfer Document \[page 30\]](#)

[Writing Texts Using Markdown \[page 32\]](#)

## 5.1.10 Working with Enhancements

### 5.1.10.1 Working with Business Add-Ins (BAdIs)

In this section you find information how to define BAdIs and how to implement BAdIs using an existing enhancement spot.

## Related Information

[Defining BAdIs \[page 263\]](#)

[Implementing BAdIs \[page 266\]](#)

## 5.1.10.1.1 Defining BAdls

You can create *BAdl Enhancement Spots* and define BAdls in the *BAdl Enhancement Spot Editor*.

### Related Information

[Creating BAdl Enhancement Spots \[page 263\]](#)

[Editing BAdl Enhancement Spots \[page 265\]](#)

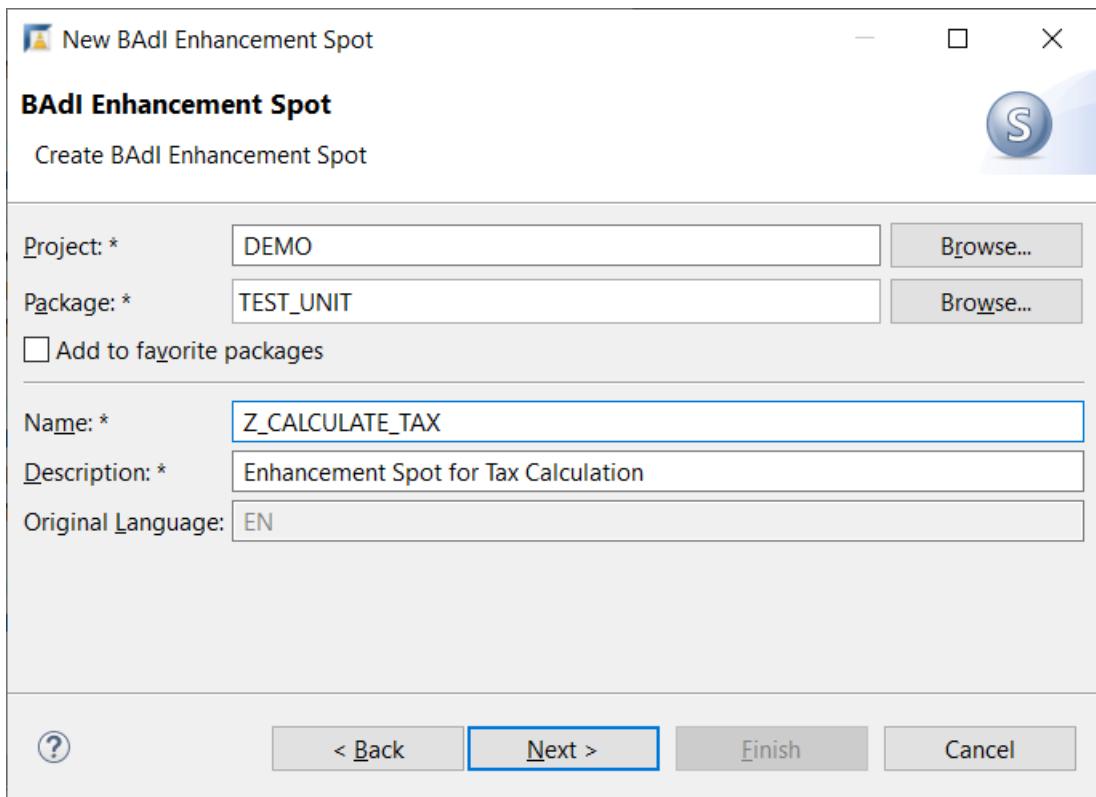
### 5.1.10.1.1.1 Creating BAdl Enhancement Spots

#### Context

BAdl definitions are defined in *BAdl Enhancement Spots*. You can create BAdl Enhancement Spots in ABAP Development Tools (ADT) using the creation wizard.

#### Procedure

1. To create a *BAdl Enhancement Spot*, start the creation wizard from the main menu, toolbar, Project Explorer, or using a shortcut. For more information, see [Creating Development Objects \[page 156\]](#).
2. In the creation wizard, fill in the project and package, if not already filled. Fill in the name and description of the BAdl Enhancement Spot.



Creation Wizard of BAdI Enhancement Spot

3. Select [Next](#).
4. Assign a transport request.
5. Select [Finish](#).

## Results

*BAdI Enhancement Spot* is created and opened in the *BAdI Enhancement Spot Editor*. In the editor, you can create new BAdI definitions.

## Related Information

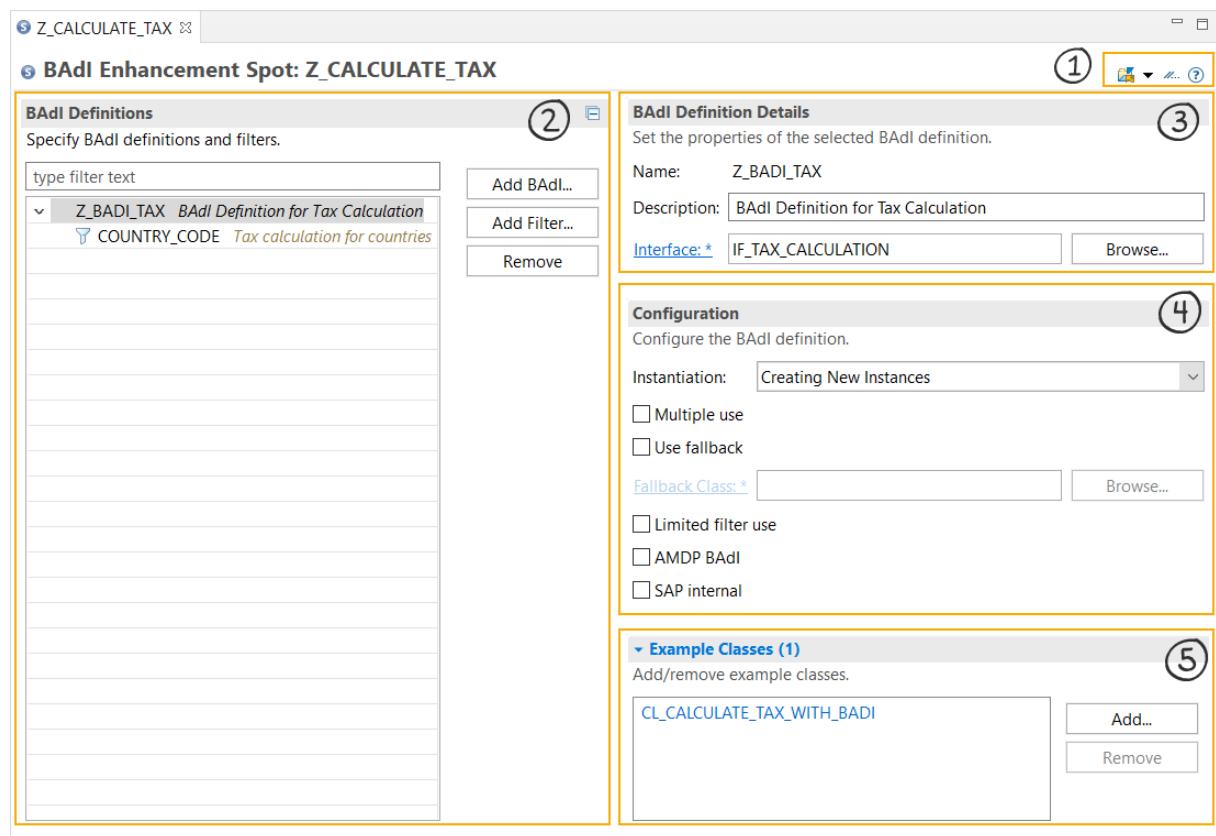
[Editing BAdI Enhancement Spots \[page 265\]](#)

[Creating BAdI Implementations \[page 267\]](#)

[Editing BAdI Implementations \[page 268\]](#)

## 5.1.10.1.1.2 Editing BAdI Enhancement Spots

You can edit BAdI spots in the *BAdI Enhancement Spot Editor*.



BAdI Enhancement Implementation Editor

The editor provides the following information about a BAdI spot:

### 1. Toolbar

In the toolbar, you can perform actions, such as opening an object in another project, sharing links, or opening the help menu.

### 2. BAdI Definitions

In this section, you can add BAdI definitions using the *Add BAdI...* button.

To add a filter, use the *Add Filter...* button in the *BAdI Definitions* section. After specifying the name for the filter, the *Filter Details* section is opened in the editor. Fill in the description and select the filter type.



Specifying Filter Details

To specify a constant filter value when using GET BADI, select the checkbox *Only constant filter values*.

In the section [Value Check](#) you can define how the filter values specified in the BAdl implementations are checked. If you select the check by data element or domain, domain values are used for the check. If you select the check by class, the interface `IF_FILTER_CHECK_AND_F4` must be implemented in the class.

### 3. BAdl Definition Details

This section provides the name and description of the BAdl definition. To select the interface, which defines the method signatures of the BAdl, use the [Browse](#) button.

### 4. Configuration

This is where you can configure the BAdl definition as follows.

- 1. [Instantiation](#). The following are possible specifications:
  - Creating new Instances
  - Reuse Instances
  - Context-Specific Instances
- 2. [Use](#). You can define whether a BAdl is to be provided for single or multiple use.
- 3. [Fallback Class](#). If no implementation is executed, select [Use fallback](#) and select a fallback class by clicking the [Browse](#) button.
- 4. [Specific Configurations](#). You can select specific configurations, such as [Limited filter use](#) and [AMDP BAdl](#).

### 5. Example Classes

In this section you can add and remove an example class by selecting [Add...](#) and [Remove](#) buttons.

## Related Information

[Creating BAdl Enhancement Spots \[page 263\]](#)

[Creating BAdl Implementations \[page 267\]](#)

## 5.1.10.1.2 Implementing BAdls

To implement an existing BAdl Enhancement Spot, create a BAdl implementation and edit it in the [BAdl Implementation Editor](#).

## Related Information

[Creating BAdl Implementations \[page 267\]](#)

[Editing BAdl Implementations \[page 268\]](#)

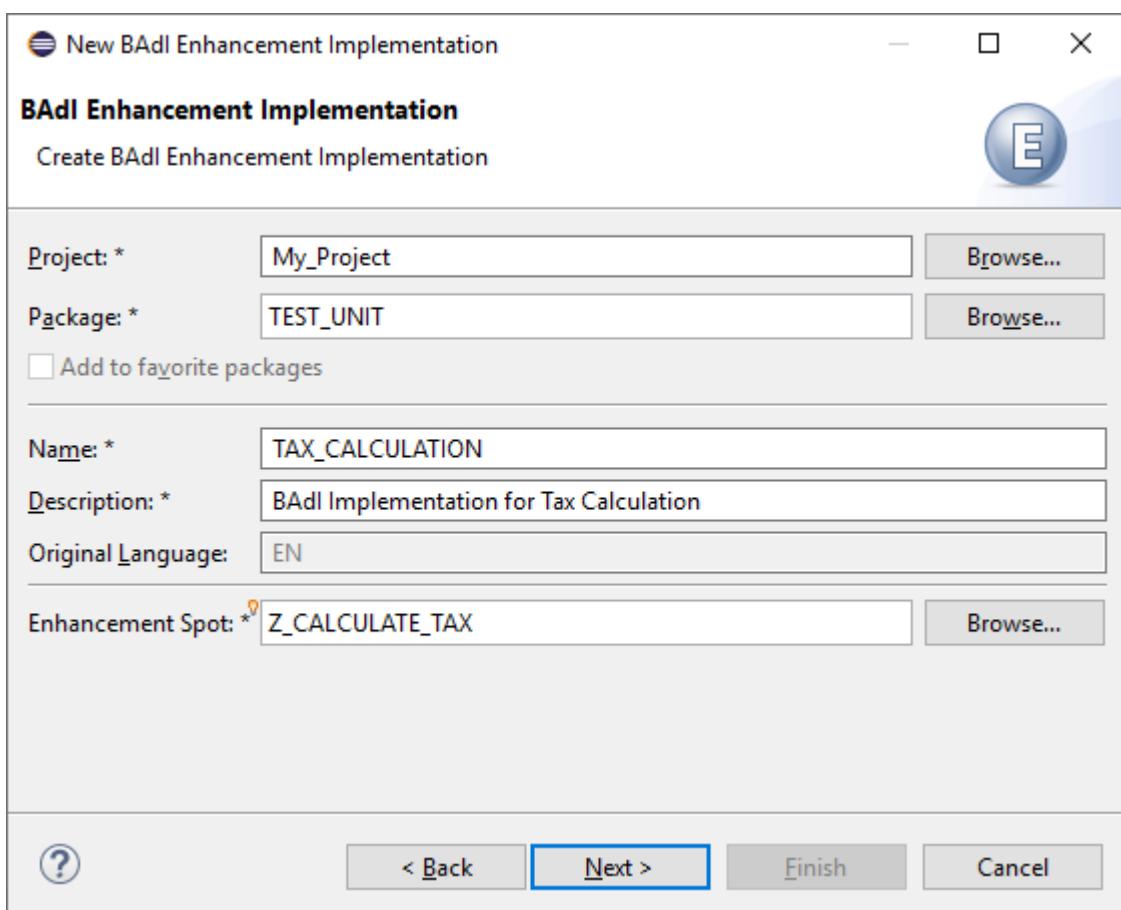
## 5.1.10.1.2.1 Creating BAdI Implementations

### Context

To create a BAdI implementation, you need to create a BAdI enhancement implementation first, where you can add BAdI implementations.

### Procedure

1. To create a *BAdI Enhancement Implementation*, start the creation wizard from the main menu, toolbar, Project Explorer, or using a shortcut. For more information, see [Creating Development Objects \[page 156\]](#).
2. In the creation wizard, fill in the project and package, if necessary. Enter the name and description of an enhancement implementation. Select an enhancement spot you want to enhance using the *Browse* button.



3. Select *Next*.

4. Assign a transport request.
5. Select *Finish*.

The *BAdI Enhancement Implementation Editor* is opened.

6. In the *BAdI Implementations* section of the editor, add a BAdI implementation using the  button.

## Results

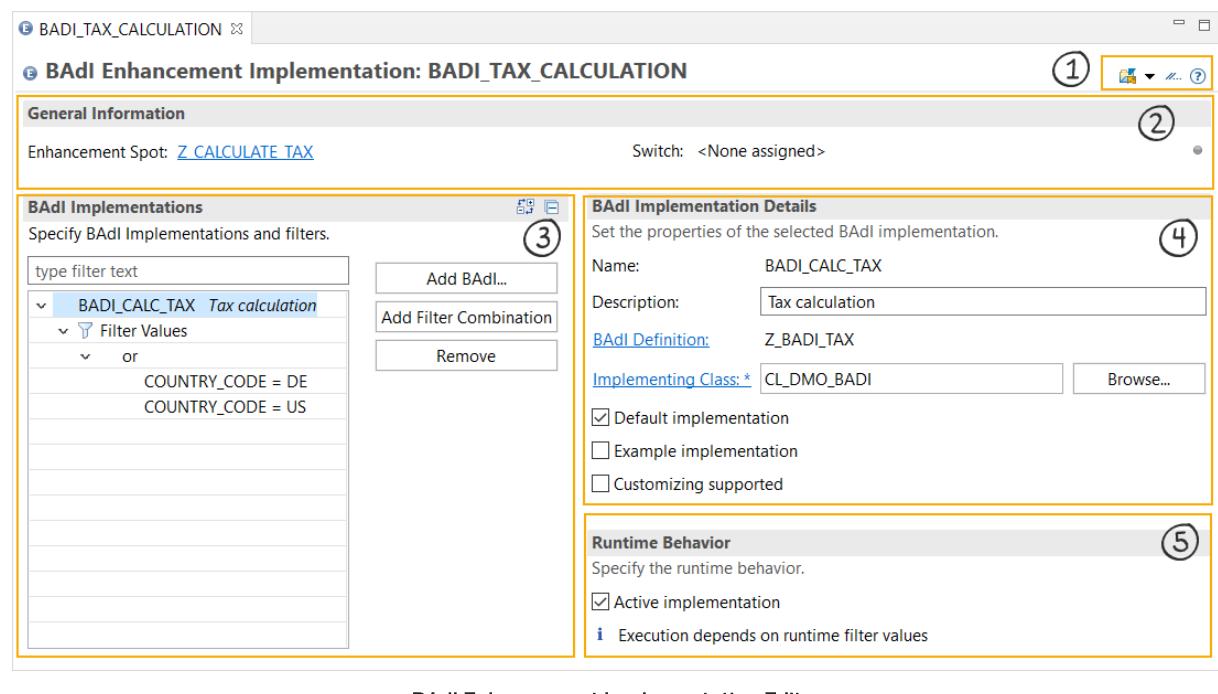
You can now edit the implementation using the *BAdI Enhancement Implementation Editor*.

## Related Information

- [Editing BAdI Implementations \[page 268\]](#)
- [Creating BAdI Enhancement Spots \[page 263\]](#)
- [Editing BAdI Enhancement Spots \[page 265\]](#)

### 5.1.10.1.2.2 Editing BAdI Implementations

You can edit BAdI implementations in the *BAdI Enhancement Implementation Editor*.



The editor provides the following information about a BAdI implementation:

## 1. Toolbar

In the toolbar, you can perform actions, such as opening an object in another project, sharing links, or opening the help menu.

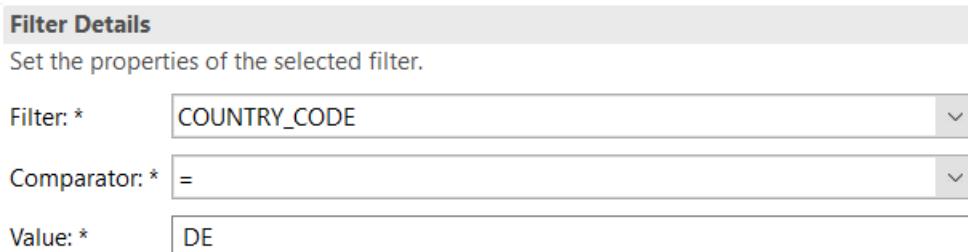
## 2. General Information

This is where you can find information about the name of the enhancement spot and whether a switch is assigned.

## 3. BAdI Implementations

This section contains a list of BAdI implementations and the filters with their values. Here you can add or remove implementations and filter values.

To add filter combinations, select the *Add Filter Combinations* button. The *Filter Details* section is opened in the editor. Select a comparator from the drop-down list and fill in the filter value.



The screenshot shows a configuration dialog for a filter. The title bar says "Filter Details" and the sub-instruction is "Set the properties of the selected filter." There are three fields: "Filter: \*" with "COUNTRY\_CODE" selected, "Comparator: \*" with "=" selected, and "Value: \*" with "DE" entered.

Filter Details	
Filter: *	COUNTRY_CODE
Comparator: *	=
Value: *	DE

### Filter Details Section

You can add more complex conditions from the context menu of the filter value.

## 4. BAdI Implementation Details

The BAdI Implementation details provide the name and description of the BAdI implementation, the BAdI definition, and the implementing class.

You can configure a BAdI implementation as follows:

- **Default implementation**

A default implementation is called at runtime. However, it can be overridden by an implementation that is not marked as default.

- **Example implementation**

An example implementation serves as an example for an implementation and is not executed itself.

- **Customizing supported**

If the checkbox *Customizing supported* is selected, the activation status of the BAdI implementation can be overridden in IMG.

## 5. Runtime Behavior

If you select the *Active Implementation* checkbox, the implementation is executed, assuming the filter conditions are fulfilled.

## Related Information

[Creating BAdI Implementations \[page 267\]](#)

[Creating BAdI Enhancement Spots \[page 263\]](#)

[Editing BAdI Enhancement Spots \[page 265\]](#)

## 5.1.11 Working with Transformations for XML

You can create and edit transformations to convert common data structures, such as structures, table types, and classes (if the `IF_SERIALIZABLE` interface is implemented) into XML format.

### Related Information

[Creating Transformations for XML \[page 270\]](#)

[Editing Transformations \[page 272\]](#)

[Debugging Simple Transformations \[page 274\]](#)

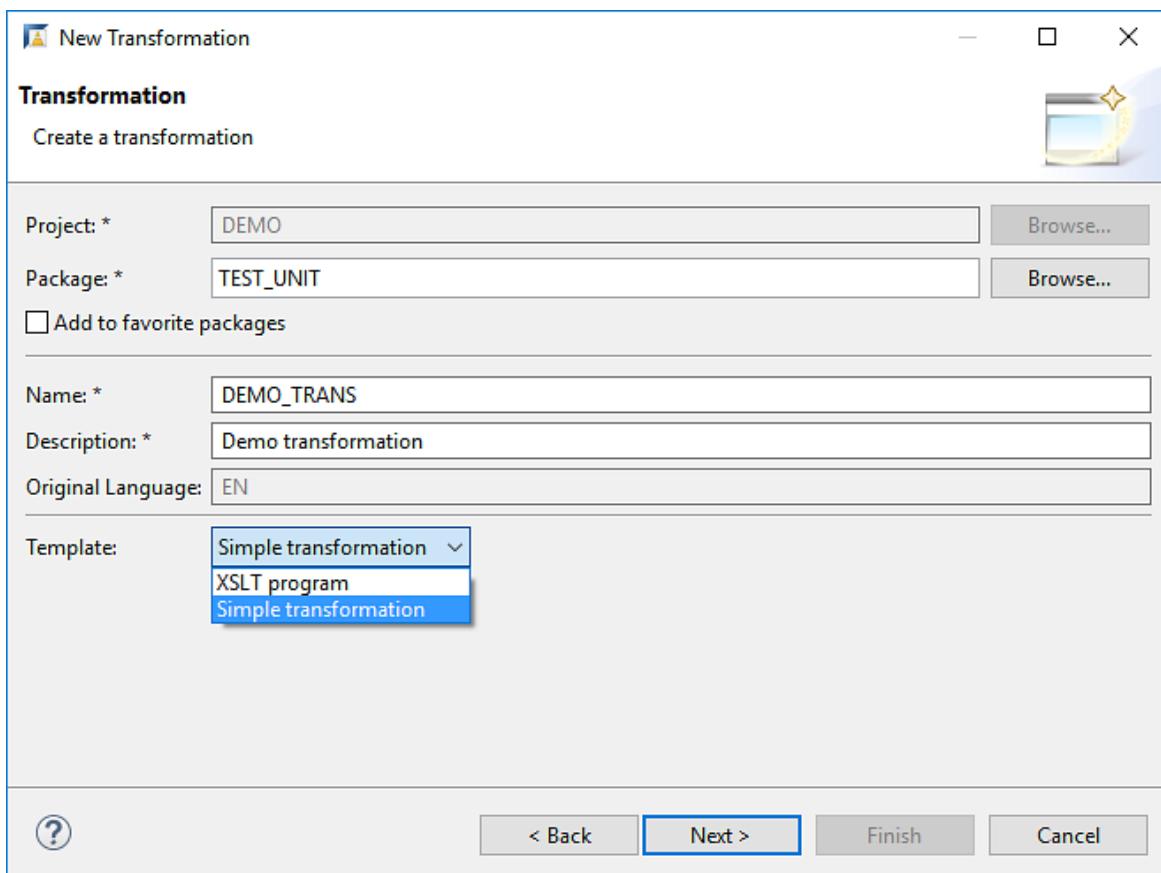
[Transformation Editor \[page 69\]](#)

### 5.1.11.1 Creating Transformations for XML

You can create a simple transformation or an XSLT program to describe transformations between ABAP data and XML data.

#### Procedure

1. In the *Project Explorer*, select the relevant package node.
2. Open the context menu and select     to launch the creation wizard.
3. *Project* and *Package* are automatically inserted. You can change *Package* if needed.
4. Enter the *Name* and *Description*.
5. Select one of the following *Templates*:
  - **XSLT program** to convert an XML document to another XML document
  - **Simple transformation** to convert the ABAP objects structures and table types into XML data



Creation wizard

6. Select *Next*.
7. Assign a transport request.
8. Select *Finish*.

## Results

The back-end system creates an inactive version of a transformation on the basis of the selected template in the selected package. In the *Project Explorer*, the new transformation is added to the corresponding package node. In the source code editor, the initially generated XML source code is displayed and ready for editing.

### Simple Transformation

If you have created a simple transformation, the following content is generated and added as source text:

```
<?sap.transform simple?>
<tt:transform xmlns:tt="http://www.sap.com/transformation-templates">
<tt:root name="ROOT"/>
<tt:template>
</tt:template>
</tt:transform>
```

For more information, see ABAP Keyword Documentation: [ST - Simple Transformations](#).

### XSLT Transformation

If you have created a XSLT transformation, the following content is generated and added as source text:

```
<xsl:transform version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:sap="http://www.sap.com/sapxsl"
>
<xsl:strip-space elements="*"/>
<xsl:template match="/">
</xsl:template>
</xsl:transform>
```

For more information, see ABAP Keyword Documentation: [XSL Transformations](#).

## Related Information

[Editing Transformations \[page 272\]](#)

## 5.1.11.2 Editing Transformations

### Context

The table below gives you an overview of the supported features.

Feature types	Features	Key Shortcuts	Availability
Standard	<a href="#">Editing [page 294]</a>		✓
	<a href="#">Deleting [page 183]</a>		✓
	<a href="#">Activation [page 70]</a>	<span style="border: 1px solid black; padding: 2px;">Ctrl</span> + <span style="border: 1px solid black; padding: 2px;">F3</span>	✓
	<a href="#">Duplicating [page 173]</a>		✓
Search	<a href="#">Searching Usages (Where-Used) [page 293]</a>	<span style="border: 1px solid black; padding: 2px;">Ctrl</span> + <span style="border: 1px solid black; padding: 2px;">Shift</span> + <span style="border: 1px solid black; padding: 2px;">G</span>	✓
		<span style="border: 1px solid black; padding: 2px;">Ctrl</span> + <span style="border: 1px solid black; padding: 2px;">H</span>	✓
Convenience	<a href="#">Navigation</a>	<span style="border: 1px solid black; padding: 2px;">F3</span>	✓

Feature types	Features	Key Shortcuts	Availability
	Formatting <a href="#">[page 307]</a>	<code>Shift</code> + <code>F1</code>	✓
	Outline <a href="#">[page 80]</a>		✓
	Quick Outline <a href="#">[page 338]</a>	<code>CTRL</code> + <code>O</code>	✓
	Code Completion <a href="#">[page 298]</a>	<code>Ctrl</code> + <code>Space</code>	✗
	Syntax Highlighting <a href="#">[page 176]</a>		✓
	Automatic Syntax Check		✓
	Element Info <a href="#">[page 33]</a>	<code>F2</code>	✗
	Quick Assists <a href="#">[page 342]</a>	<code>Ctrl</code> + <code>1</code>	✗
Others	Version History <a href="#">[page 208]</a>		✓
	Comparing Source Code <a href="#">[page 207]</a>		✓
	Breakpoints for Debugging <a href="#">[page 75]</a>		✓
	Unit Testing <a href="#">[page 71]</a> (via test relations) <a href="#">[page 561]</a>	<code>Ctrl</code> + <code>Shift</code> + <code>F10</code>	✓
	Share Link (ADT link only)		✓
	Context-sensitive syntax documentation	<code>F1</code>	✓

## Related Information

[Editing ABAP Source Code \[page 294\]](#)

[Debugging ABAP Code \[page 613\]](#)

### 5.1.11.3 Debugging Simple Transformations

Since The ABAP Debugger supports simple transformation (ST) in ABAP Development Tools (ADT). Hence, you can debug and test whether a simple transformation converts ABAP source code to XML as required.

#### Prerequisites

You can only debug activated development objects.

#### Context

You can call simple transformations, for example, in ABAP classes or programs. To test the behavior of a simple transformation in advance, proceed as follows:

#### Procedure

1. Open the simple transformation.
2. To set a breakpoint, click on the corresponding position within the `<tt:template>` section in the ruler bar.

The breakpoint is set and displayed with the  decorator.

3. Open, for example, the ABAP class or program where the ST is called.
4. Run, for example, an unit test or the ABAP program.

If the execution pointer of the ABAP Debugger points at a `CALL TRANSFORMATION` statement of a simple transformation, choose .

5. [Optional:] Confirm possible dialogs whenever you are asked to switch to the *Debug* perspective with *Yes*.

The *Debug* perspective is opened. The ABAP Debugger will stop directly at the breakpoint.

6. [Optional:] Simple transform template calls are reflected by the ABAP Debugger stack. Therefore, you can navigate to calling templates or ABAP methods on the ABAP stack. To do this, proceed as follows:
  - a. In the *Debug* view, double-click the relevant template or method.

The source code editor displays the corresponding source code and sets the execution pointer at the calling position. In addition, in the *Variables* view, the current value of the selected parameters, local variable, and data nodes are displayed.

7. [Optional:] To display the current value of variables, parameters, and data nodes of simple transformations, hover the relevant symbol on the simple transformation source or double-click it.

A popup is opened that displays the current value of the selected element.

## Related Information

[Debugging ABAP Code \[page 613\]](#)

## 5.1.12 Working with Messages and Message Classes

You can create and group messages in message classes in order to inform the user about an error or a status, or to issue a warning.

### Context

#### Note

You have to activate a message class after you have edited or created it.

## Related Information

[Messages and Message Classes \[page 66\]](#)  
[Message Types \[page 67\]](#)  
[Creating a Message Class \[page 275\]](#)  
[Adding a Message to a Message Class \[page 277\]](#)  
[Adding a Long Text to a Message \[page 278\]](#)  
[Filtering Messages in Message Classes \[page 282\]](#)  
[Opening a Message from the Source Code in the Message Class Editor \[page 283\]](#)  
[Opening in Another System \[page 163\]](#)  
[Previewing a Message Long Text \[page 280\]](#)  
[Sharing Links of Selected Messages \[page 284\]](#)  
[Searching the Usage \(Where-used\) of a Message \[page 286\]](#)

## 5.1.12.1 Creating a Message Class

### Context

You create a message class to group and reuse messages in a development object.

## Procedure

1. In the *ABAP* perspective, expand the corresponding package node.
2. In the context menu of the package, choose  *New*  *Other ABAP Repository Object* .
3. In the *New ABAP Repository Object* window, select the *Message Class* folder and expand it.
4. Choose *Message Class*.
5. Choose *Next*.
6. In the *Name* field, enter the *Name* of the new message class.

### Note

You can enter up to 20 digits, maximum.

7. In the *Description* field, enter a description that provides additional details.
8. Choose *Next*.
9. Assign a transport request.
10. Start the creation of the message class with *Finish*.

## Results

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

As a result of this creation procedure, the editor will be opened. You can now add messages to the message class.

## Related Information

- [Adding a Message to a Message Class \[page 277\]](#)
- [Adding a Long Text to a Message \[page 278\]](#)
- [Filtering Messages in Message Classes \[page 282\]](#)
- [Opening a Message from the Source Code in the Message Class Editor \[page 283\]](#)
- [Opening in Another System \[page 163\]](#)
- [Previewing a Message Long Text \[page 280\]](#)
- [Sharing Links of Selected Messages \[page 284\]](#)
- [Messages and Message Classes \[page 66\]](#)
- [Message Types \[page 67\]](#)
- [Transport Request \[page 108\]](#)
- [Adding a Development Object to a Transport Request \[page 164\]](#)

## 5.1.12.2 Adding a Message to a Message Class

### Prerequisites

An appropriate message class to which you want to add a message must exist.

### Context

You create or edit a short text that is going to be used by an application.

### Procedure

1. In the *ABAP perspective*, expand the corresponding package node.
2. Double-click the *Message Class* level.
3. In the menu bar of the editor, choose the **New** button or directly type the message number in the *<Enter new value>* field. NOTE: In order to continue a certain message number range, select the message with the message number that you want to continue. So, the editor creates the new message with the following message number. If you don't select a message, the editor automatically continues with the next available message number.
4. Enter the message number.

#### **i** Note

Enter a unique 3-digit message number that is not already used in the corresponding message class.

5. In the editor double-click in the *Short Text* field.

#### **i** Note

If you edit an existing message, the whole message class is blocked for other users. The blockage is closed after saving your changes.

6. Enter the message short text. NOTE: You can enter up to 73 digits, maximum.
7. Specify for each message if it is **Self Explanatory**.

#### **i** Note

Messages that are marked as self-explanatory need no further explanations. If this checkbox is not activated a long text has to be added.

8. Save your entries.

## Results

The system creates the message in the development object *Message Classes*. It is stored in table **T100** and can be displayed.

## Related Information

[Creating a Message Class \[page 275\]](#)

[Adding a Long Text to a Message \[page 278\]](#)

[Filtering Messages in Message Classes \[page 282\]](#)

[Opening a Message from the Source Code in the Message Class Editor \[page 283\]](#)

[Opening in Another System \[page 163\]](#)

[Previewing a Message Long Text \[page 280\]](#)

[Sharing Links of Selected Messages \[page 284\]](#)

[Messages and Message Classes \[page 66\]](#)

[Message Types \[page 67\]](#)

### 5.1.12.3 Adding a Long Text to a Message

#### Prerequisites

A message of an appropriate valid message class must exist in a program.

#### Context

If you need to display a message with more than 73 digits, enter a long text for the message.

##### Note

Long text can be added to every message which is not marked as self-explanatory. If you add long text to a message, the editor will automatically deactivate the checkbox.

#### Procedure

1. In the *ABAP perspective*, expand the corresponding package node.
2. Double-click the *Message Class* level.

3. Select the message to which you want to add a long text.

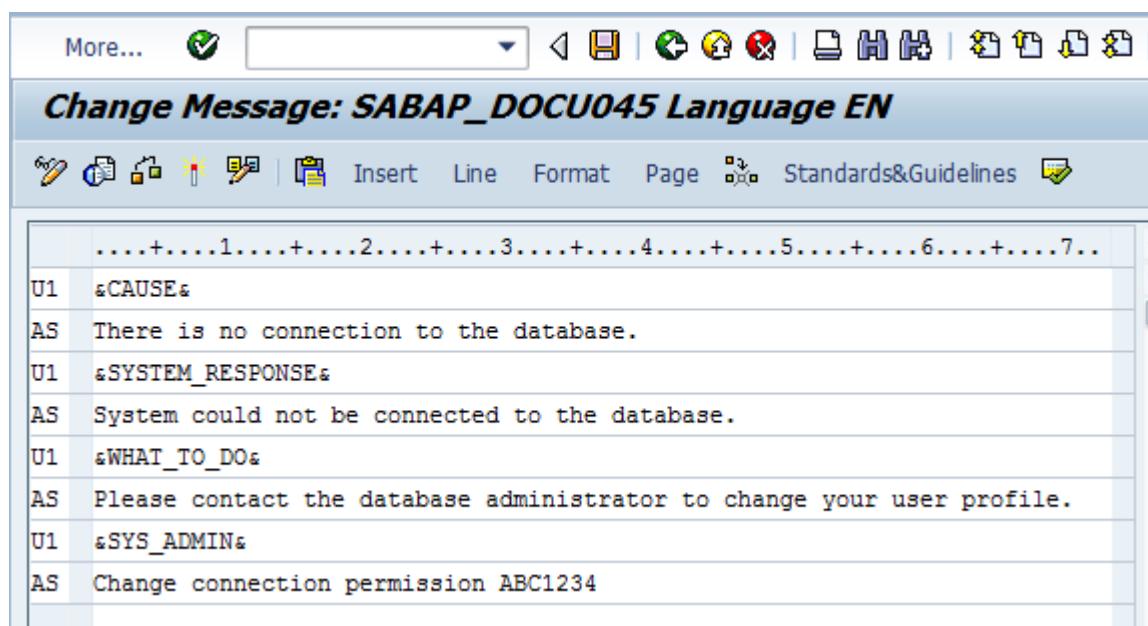
**i Note**

You can only add a long text to a message, if the message is not marked as *Self Explanatory*.

4. [Optional] Deactivate the checkbox **Self Explanatory**.
  5. Choose button **Long Text** in the message class editor.
- A SAP GUI will be opened where you can add a long text to the message.
6. Enter the long text in the SAP GUI editor by adding the text for the corresponding information level.

You can enter the following information levels, for example:

- Cause: Reason why the problem occurred.
- System Response: Details about the result of attempts made by the system to succeed.
- What to do: Procedure for next steps, to be performed by the user.
- Sys Admin: Procedure for next steps, to be performed by the system administrator.



Display of the Workbench editor where you add a long text to a message

7. Save the entries.
8. Activate the message class.

## Results

The entered text will be saved in the master language of the ABAP project.

**i Note**

All messages which have been created or changed will be locked until they are saved. So, no other user can edit them until they are saved.

## Related Information

[Creating a Message Class \[page 275\]](#)

[Adding a Message to a Message Class \[page 277\]](#)

[Filtering Messages in Message Classes \[page 282\]](#)

[Opening a Message from the Source Code in the Message Class Editor \[page 283\]](#)

[Opening in Another System \[page 163\]](#)

[Previewing a Message Long Text \[page 280\]](#)

[Sharing Links of Selected Messages \[page 284\]](#)

[Messages and Message Classes \[page 66\]](#)

[Message Types \[page 67\]](#)

### 5.1.12.4 Previewing a Message Long Text

#### Prerequisites

You can only preview the long text:

- if the ABAP project is opened in the original language. If no translation is available, a standard message is displayed.
- if the message is not activated as self-explanatory.

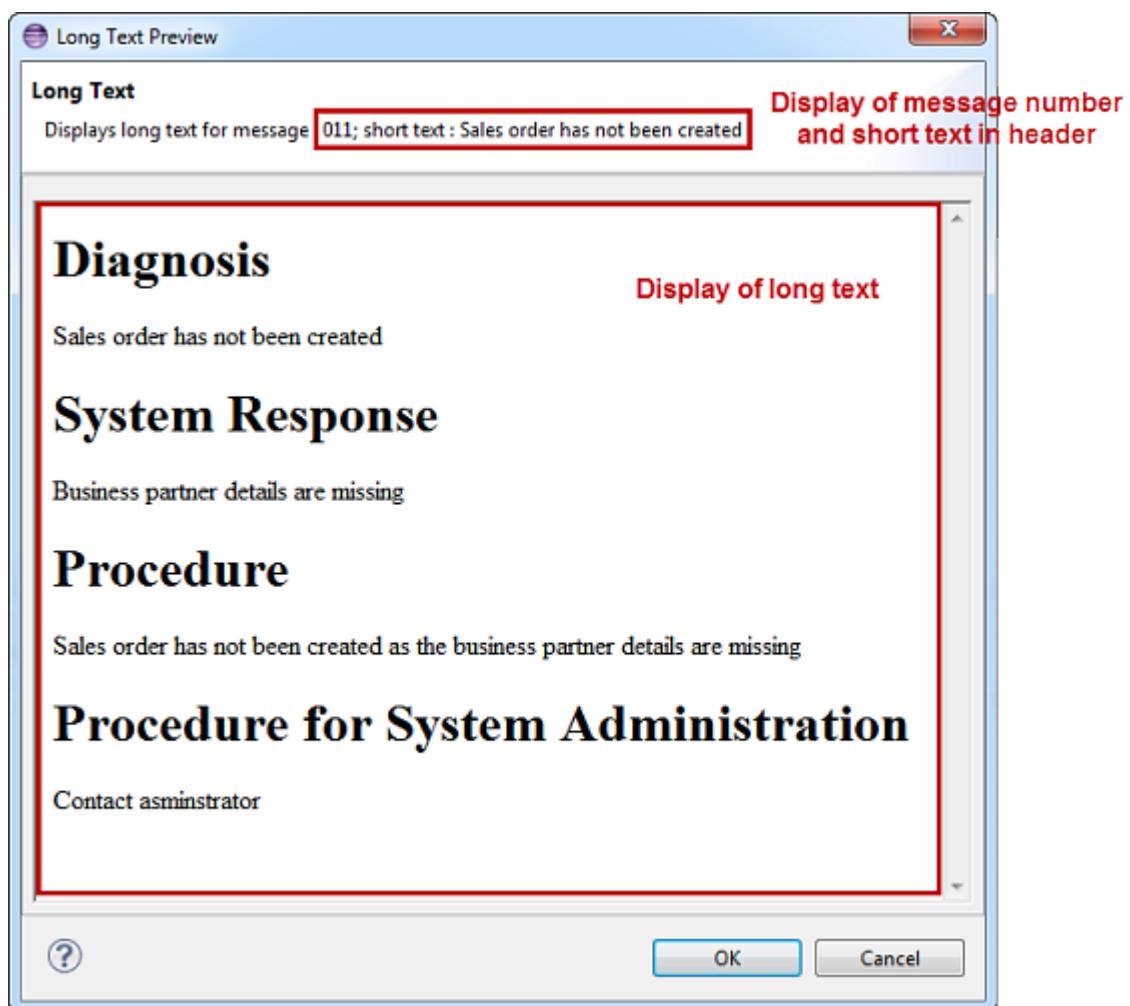
#### Context

In the *Long Text Preview* window, you can display the long text of a message that was added through the back end in an ABAP project.

This enables you to obtain the following information without opening the SAP GUI:

- Header: Message number and short text, so you can check that you have selected the desired message
- Main frame: Long text that was added to the message

#### Example



Long Text Preview window with the long text of the selected message

## Procedure

1. In the *ABAP* perspective, expand the corresponding package node.
2. Select the corresponding message class in the *Message Class* node.
3. In the *Message Class* editor, select the message class with the long text.
4. In the toolbar, select the **Preview** button.

## Results

The *Long Text Preview* window is opened, displaying the long text that was added to this message in the back end.

## Related Information

[Creating a Message Class \[page 275\]](#)

[Adding a Message to a Message Class \[page 277\]](#)

[Adding a Long Text to a Message \[page 278\]](#)

[Filtering Messages in Message Classes \[page 282\]](#)

[Opening a Message from the Source Code in the Message Class Editor \[page 283\]](#)

[Opening in Another System \[page 163\]](#)

[Sharing Links of Selected Messages \[page 284\]](#)

[Messages and Message Classes \[page 66\]](#)

[Message Types \[page 67\]](#)

### 5.1.12.5 Filtering Messages in Message Classes

#### Context

You filter for short texts and message numbers to search for particular messages of a message class. This functionality helps you to find and reuse messages which already exist. So you can prevent creating the almost identical message several times.

##### Note

The message class editor displays per default all messages which are defined in the message class.

##### Example

You want to check if there are messages which contain the combination 'airline carrier' in the short text. After you have entered this combination in the filter text field, the editor will automatically display the corresponding messages.

#### Procedure

1. In the [Project Explorer](#), expand the relevant package node.
2. Double-click the relevant [Message Class](#) node.
3. In the filter text field of the message class, enter the text or message number, which you want to search for.

Message Class: SABAP_DOCU [Total Messages: 3, With Long text: 2]					
Locked	Number	Short Text	Self Explanatory	Last Changed By	Changed On
	003	Choose an example program	<input type="checkbox"/>		2019-07-17
	045	No authorization for airline carrier &	<input type="checkbox"/>		2019-07-17
	047	No flight found for this date	<input checked="" type="checkbox"/>		2019-07-17
	<Enter new value>		<input checked="" type="checkbox"/>		

Message class editor with the filter text field

## Results

After entering the filter text the editor will automatically display the reduced set of messages.

→ Tip

Delete the filter text to display all messages of the message class.

## Related Information

[Creating a Message Class \[page 275\]](#)

[Adding a Message to a Message Class \[page 277\]](#)

[Adding a Long Text to a Message \[page 278\]](#)

[Opening a Message from the Source Code in the Message Class Editor \[page 283\]](#)

[Opening in Another System \[page 163\]](#)

[Previewing a Message Long Text \[page 280\]](#)

[Sharing Links of Selected Messages \[page 284\]](#)

[Searching the Usage \(Where-used\) of a Message \[page 286\]](#)

[Messages and Message Classes \[page 66\]](#)

[Message Types \[page 67\]](#)

## 5.1.12.6 Opening a Message from the Source Code in the Message Class Editor

### Context

You can open a message in the message class editor from the source code editor of any source-based objects such as classes, function modules, or ABAP programs.

## Procedure

In the source code editor, choose **Ctrl** + **left mouse button** or **F3** on the corresponding message number.

## Results

The message class editor is opened and the number of the selected message is highlighted.

## Related Information

[Creating a Message Class \[page 275\]](#)  
[Adding a Message to a Message Class \[page 277\]](#)  
[Adding a Long Text to a Message \[page 278\]](#)  
[Filtering Messages in Message Classes \[page 282\]](#)  
[Opening in Another System \[page 163\]](#)  
[Previewing a Message Long Text \[page 280\]](#)  
[Sharing Links of Selected Messages \[page 284\]](#)  
[Searching the Usage \(Where-used\) of a Message \[page 286\]](#)  
[Messages and Message Classes \[page 66\]](#)  
[Message Types \[page 67\]](#)

## 5.1.12.7 Sharing Links of Selected Messages

### Prerequisites

You can only copy the hyperlink if the message is saved.

### Context

You can initiate the creation of a hyperlink that refers to a specific message of a message class. The receiver can then open the referenced message directly in ABAP Development Tools (ADT) in the corresponding ABAP project.

For this purpose, you can generate an email where the hyperlink path is added or paste it from the clipboard manually into any document.

## Procedure

1. In the *Message Class* editor, select the message.
2. In the toolbar, select the *Share link for selection...* button.
3. In the *Share Link for Selection* dialog, select *ADT Link*.
4. [Optional:] Choose the *Copy link to clipboard* button and paste the link into any document.

## Results

The hyperlink path is then saved in temporary storage. If you paste it in your document or editor, the path is added as text.

### • Example

The following example displays the URL of a specific message:

```
adt://[ABAP system name]/sap/bc/adt/messageclass/demo_message_class/
messages/100
```

### i Note

To make a hyperlink click-sensitive, the receiver needs to define the application that is launched when opening a hyperlink.

## Related Information

[Linking to Open Development Objects in Another ADT Installation \[page 167\]](#)

[Opening in Another System \[page 163\]](#)

[Linking to Selected Parts of ABAP Source Code \[page 169\]](#)

[Creating a Message Class \[page 275\]](#)

[Adding a Message to a Message Class \[page 277\]](#)

[Adding a Long Text to a Message \[page 278\]](#)

[Filtering Messages in Message Classes \[page 282\]](#)

[Opening a Message from the Source Code in the Message Class Editor \[page 283\]](#)

[Previewing a Message Long Text \[page 280\]](#)

[Messages and Message Classes \[page 66\]](#)

[Message Types \[page 67\]](#)

## 5.1.12.8 Searching the Usage (Where-used) of a Message

### Context

In the message class editor, you can provide the where-used list in order to display the usage of a message.

#### i Note

See more in [Searching Usages \(Where-Used\) \[page 293\]](#)

### Related Information

[Searching Usages \(Where-Used\) \[page 293\]](#)

[Creating a Message Class \[page 275\]](#)

[Adding a Message to a Message Class \[page 277\]](#)

[Adding a Long Text to a Message \[page 278\]](#)

[Opening a Message from the Source Code in the Message Class Editor \[page 283\]](#)

[Messages and Message Classes \[page 66\]](#)

[Where-Used Function \[page 43\]](#)

## 5.1.13 Working with the HTTP Service Editor

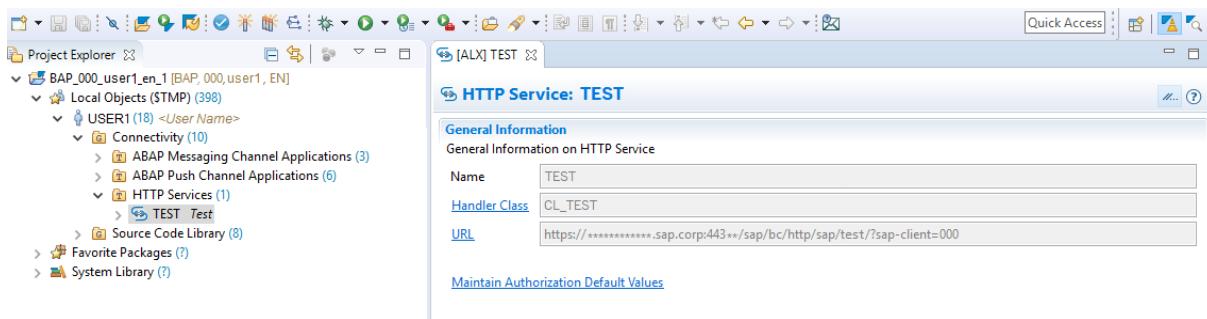
Use the HTTP service editor to create and configure HTTP services for your application.

You can use the HTTP service editor to create and activate an HTTP service based on the ABAP Internet Communication Framework (ICF).

To create the first HTTP service object, proceed as follows:

1. Right click on your user name (or, if available, on the *Connectivity* node) and choose   *New*  *Other ABAP Repository Object* .
2. In the creation wizard, enter `http` in the `<type filter text>` field, select *HTTP service* and choose *Next*.
3. Choose a `<Package>`, `<Name>` and `<Description>` for your service.
4. Select *Add to favorite packages* to store the new service in your favorites.
5. Choose *Finish*.

In the navigation tree, a new node *HTTP services* is created under    *<User>*  *Connectivity* . To create more HTTP service objects, simply right-click on the *HTTP services* node in the navigation tree and choose *Add new HTTP service*.



Once created, you can use this HTTP service to implement your application-specific functionality. Click on the link [Handler class](#) to open the class editor.

#### i Note

The handler class can be named in the HTTP service wizard, and is generated if not existing. Changing the class name outside the HTTP service wizard is not supported and would cause an invalid handler class.

To create authorization default values for the service, choose the link [Maintain Authorization Default Values](#) and follow the steps in the creation wizard.

To activate the service, choose the icon [Activate](#) from the top icon bar or right-click on the service name in the navigation tree and choose [Activate](#).

#### i Note

The corresponding SICF service node will be activated in the background as soon as you have assigned your HTTP service to a communication scenario, see also [Consuming Services in the Context of API with Technical Users \[page 127\]](#).

## Calling Your HTTP Service from the Browser

To call your HTTP service from the browser, choose the [URL](#) link in the service details (section [General Information](#)).

## 5.1.14 Working with the Relation Explorer

You have different options to select an object to display its related objects.

- **Using the Editor**

From the context menu in the Editor, select [Show In > Relation Explorer](#).

- **Using a Shortcut**

Press [Alt](#) + [Shift](#) + [W](#) in the Project Explorer or in the Editor. Select [Relation Explorer](#).

- **Drag and Drop**

You can drag and drop the object to the *Relation Explorer*.

- **From the History List**

You can select the object from the history list, using  in the toolbar.

- **Using the Other Object Option**

You can select another object by clicking  in the toolbar.

## Toolbar Features

The following toolbar features are available in the Relation Explorer:

Icon	Description	Name
	If there are different contexts available for the object, you can switch to another context.	Other Context
	You can reload the object relations.	Reload Relations

Icon	Description	Name
	The <a href="#">History list</a> button displays a history of previously displayed entry objects.	History
	You can select another object.	Object
	You can have the <a href="#">Relation Explorer</a> update automatically to the object in the active editor by enabling the <a href="#">Link with Editor</a> button.	Link with editor
	In the view menu, you can	<ul style="list-style-type: none"> <li>Select label decorations for an object such as a short description, its package, or no label decoration</li> <li>Show or hide the search filter</li> <li>Show or hide details, if the details are available</li> <li>Select the horizontal, vertical, or automatic layout of displaying the details</li> <li><a href="#">Create a new instance of Relation Explorer [page 290]</a></li> <li>Rename the view</li> <li>Enable the <a href="#">Link with Editor</a> option</li> <li>Return to the <a href="#">Getting Started</a> page of the Relation Explorer</li> </ul>

## Related Information

[Relation Explorer \[page 36\]](#)

[Creating a New Instance of the Relation Explorer \[page 290\]](#)

## 5.1.14.1 Creating a New Instance of the Relation Explorer

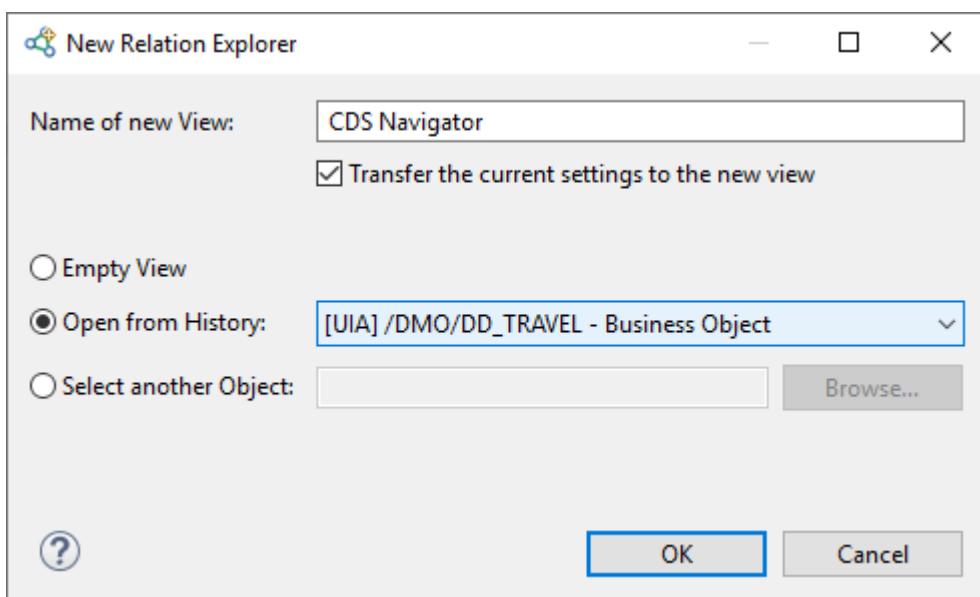
### Context

You can create a new instance of the Relation Explorer to work with multiple objects or with the same object in different contexts at once, instead of switching between them.

### Procedure

1. In the Relation Explorer, from the dropdown list of the view menu , select [New Relation Explorer...](#).

The creation dialog is opened.



**Creating a New Relation Explorer**

2. In the *Name of new View* field, you can type a name for a view. Select or deselect the *Transfer the current settings to the new view* checkbox to define the settings for the new view.
3. Select one of the following:
  - Empty View
  - [OPTIONAL] Open from History
  - Select another object by clicking the *Browse* button.
4. Select *OK*.

## Results

The Relation Explorer is created and opened.

## Related Information

[Relation Explorer \[page 36\]](#)

### 5.1.15 Searching in ABAP Projects

#### Context

In addition to the Eclipse search functionalities, the ABAP search provides powerful tools for finding versions of an ABAP development object in all of the back-end systems you work with.

The following functions enable you to search for ABAP project-specific content:

- [Searching Development Objects \[page 291\]](#) to find a certain ABAP development objects on base of the object name
- [Searching Usages \(Where-Used\) \[page 293\]](#) to find the usages of development objects and sub-objects

#### i Note

Use the ABAP Search function to find ABAP content. The Eclipse file search results may be incomplete.

## Related Information

[ABAP Search \[page 42\]](#)

[Opening Development Objects \[page 159\]](#)

### 5.1.15.1 Searching Development Objects

#### Context

You can search for ABAP development objects either in one particular ABAP project or system-wide across all of your ABAP projects.

## Procedure

1. In the menu bar, choose  **Search**  **Search...**  Alternatively, you can use the shortcut **Ctrl + H**.
2. In the following dialog box, choose the **ABAP Object Search** tab.

 **Tip**

If the recommended tab is not displayed, choose the **Customize...** button, to select the corresponding one.

3. In the **ABAP Object Search** tab, enter the **Object Name** you wish to find.

 **Note**

You can only search for main repository objects, such as ABAP classes, ABAP programs, or data dictionary objects. You cannot search for sub-objects such as methods or form routines.

4. Select **Workspace** to search in all your ABAP projects that are currently active and usable. Or select one of your ABAP projects.

 **Note**

You can restrict the search to finding your favorite packages by activating the **Restrict Search to Favorite Packages** checkbox.

5. Select **Search** to launch the search.

## Results

The repository objects that match the search scope are displayed in the **Search** view. The result is grouped by ABAP packages

 **Note**

From the results list, you can use the context menu to open each object in the editor.

## Related Information

[ABAP Development Objects \[page 19\]](#)

[ABAP Search \[page 42\]](#)

[Opening Development Objects \[page 159\]](#)

[Searching in ABAP Projects \[page 291\]](#)

## 5.1.15.2 Searching Usages (Where-Used)

The improvements in the *Where-used* view like the general and filter functionalities from the toolbar and context menu

### Context

You can investigate where a development object or element is used within an ABAP package.

You can execute the where-used search from the following positions:

- Project Explorer
- Outline
- Source code editor
- Another where-used result

### Example

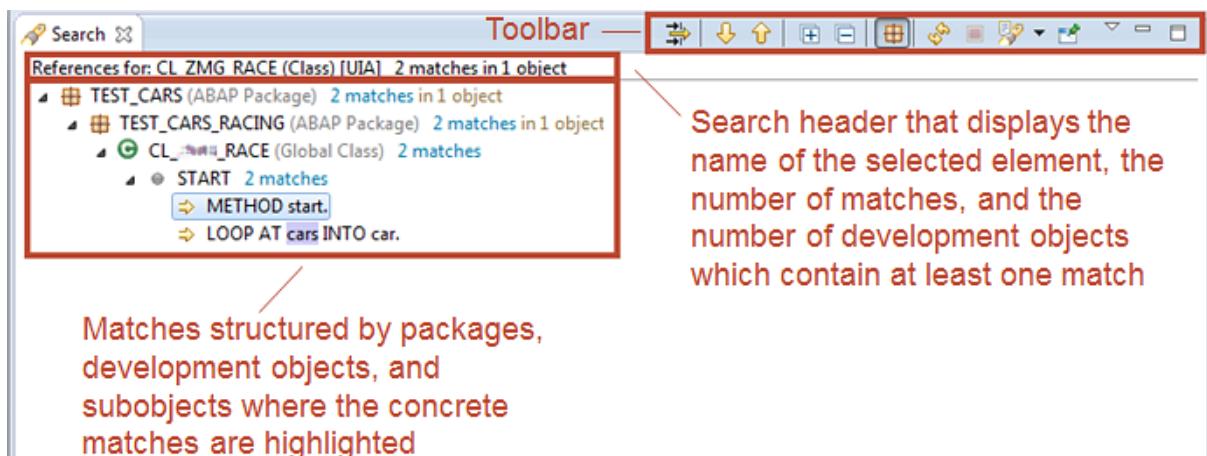
The following procedure describes how to search for the usage of a development object from the *Project Explorer*:

### Procedure

1. In the *Project Explorer*, select the relevant development object.
2. From the context menu or the toolbar, choose  **Get Where-Used List....**

### Results

The *Search* view is opened, and the relationships and usages of the development object and their elements are displayed.



Search header that displays the name of the selected element, the number of matches, and the number of development objects which contain at least one match

Matches structured by packages, development objects, and subobjects where the concrete matches are highlighted

Example of a search view that displays the hits and relationships of an ABAP class

To display the number of matches instead of a placeholder, expand the corresponding tree level.

From here you can add an object to the *Favorite Objects*, run an ABAP unit test or another where-used search for a development object or an element.

## Related Information

[ABAP Search \[page 42\]](#)

[Where-Used Function \[page 43\]](#)

[Searching in ABAP Projects \[page 291\]](#)

## 5.1.16 Editing ABAP Source Code

### Context

The ABAP IDE provides a number of functions and utilities for efficiently creating, editing, and navigating ABAP source code. These primarily comprise ABAP keyword completion, the source code templates, and the various formatting capabilities.

## Related Information

[ABAP Development Objects \[page 19\]](#)

[Getting Support from the Content Assist \[page 295\]](#)

[Working with Source Code Templates \[page 305\]](#)

[Closing Brackets and Literals Automatically \[page 307\]](#)

[Formatting ABAP Source Code \[page 307\]](#)

[Refactoring ABAP Source Code \[page 342\]](#)  
[Applying ABAP Quick Fixes \[page 402\]](#)  
[Changing Colors for ABAP Source Code \[page 309\]](#)  
[Marking Occurrences in the Source Code Editor \[page 331\]](#)  
[Viewing the Outline \[page 332\]](#)  
[Viewing the ABAP Type Hierarchy \[page 334\]](#)  
[Using Quick Views \[page 337\]](#)  
[Comparing Source Code \[page 207\]](#)  
[Managing Version Conflicts \[page 340\]](#)

## 5.1.16.1 Getting Support from the Content Assist

In the *ABAP source code editor*, the content assist proposes valid ABAP keywords and identifiers that are relevant at the current position. One of these proposals can be inserted to the source code.

### Context

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

The following content assist functionalities are provided:

List of Available ABAP Content Assist Functionalities

Content Assist	Description
<a href="#">Keyword Completion [page 296]</a>	The best matching ABAP keyword is automatically proposed as soon as you start typing.
<a href="#">Non-Keyword Completion [page 297]</a>	The best matching non-keyword, that is available for the corresponding source code section, is automatically proposed as soon as you start typing.
<a href="#">Code Completion [page 298]</a>	A list of all matching keywords, identifiers, components, and templates is displayed for the position where you have chosen the <code>Ctrl + Space</code> shortcut.

### Related Information

[ABAP Development Objects \[page 19\]](#)  
[Working with Source Code Templates \[page 305\]](#)

## 5.1.16.1.1 Keyword Completion

In the *ABAP source code editor*, the best matching ABAP keyword is displayed as soon as you start typing the keyword. If you choose the tabulator tab, the keyword is inserted at the current position.

### Prerequisites

To use this functionality, navigate to the  *Window*  *Preferences*  *ABAP Development*  *Editors*  *Source Code Editors*  *Code Completion*  preference page and select the  *Typing*  *Suggest keywords*  option.

### Context

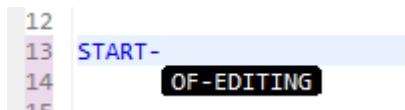
#### Note

Keyword completion is enabled by default.

### Procedure

1. In the source code, start typing a keyword.

The proposal for the most relevant keyword is displayed.



Display of a proposal for a certain keyword while typing

2. Adopt the displayed entry into your source code by choosing the *tabulator* key.

### Results

At the position where you have selected the proposal, the corresponding keyword is added to the source code.

## 5.1.16.1.2 Non-Keyword Completion

In the *ABAP source code editor*, the best matching non-keyword, such as an element name or type definition, is displayed when you start typing.

### Prerequisites

To use this functionality, open the  *ABAP Development*  *Editors*  *Source Code Editors*  *Code Completion*  *Typing*  preference page and select the *Also suggest non-keywords* option.

### Context

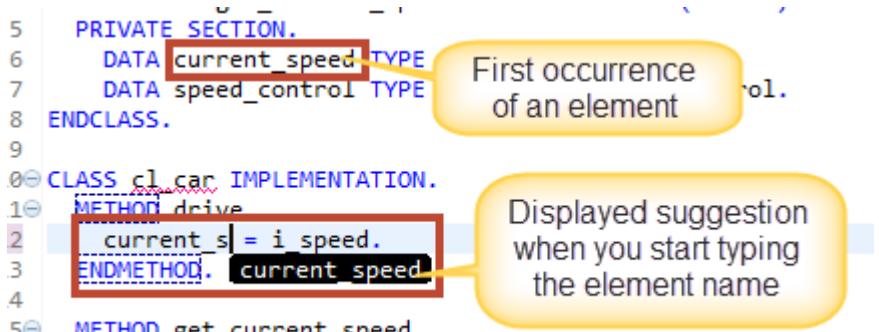
#### Note

Non-keyword completion is disabled by default.

### Procedure

1. In the source code, start typing a non-keyword.

The proposal for the most relevant non-keyword is displayed.



```
5 PRIVATE SECTION.
6   DATA current_speed TYPE
7   DATA speed_control TYPE
8 ENDCLASS.
9
0 CLASS cl_car IMPLEMENTATION.
1 METHOD drive
2   current_s = i_speed.
3 ENDMETHOD. current_speed
4
5 METHOD get_current_speed.
6   speed_control->set_max_speed( ).
```

Display of a proposal for a certain non-keyword while typing

#### Note

Non-keywords are only proposed if they are already used in the same development object.

2. Adopt the displayed entry into your source code by choosing the *tabulator* key.

## Results

At the position where you have selected the proposal, the corresponding non-keyword is added to the source code.

### 5.1.16.1.3 Code Completion

In the *ABAP source code editor*, you can open the code completion list manually at any position or open it automatically after you have typed a component selector such as `-`, `~`, `->`, `=>`.

## Context

This functionality enables you to:

- add possible keywords or identifiers such as variables, classes, interfaces, methods
- select and add the component of the related identifier, such as a field of a structure or a method of a class to your source code.

## Procedure

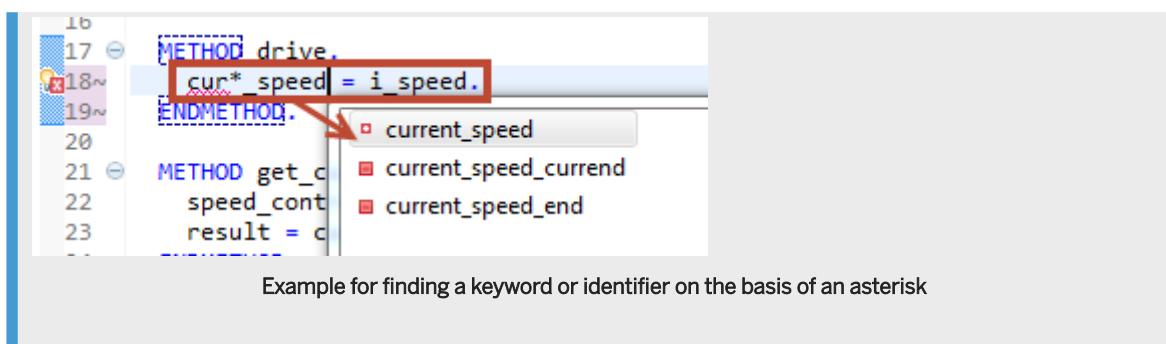
1. In the source code editor, type the beginning character(s).
2. To display the list of suggested entries at this position, choose `Ctrl` + `Space`.  
The most relevant proposals are displayed in the code completion list.
3. Choose `Enter` to add keywords or identifiers or `Shift` + `Enter` to add the full signature of a method or function module.

### i Note

If the code completion list is not displayed, check whether the related components are correctly defined in the respective development object.

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

If you use the asterisk, a list is opened that displays the relevant keywords and identifiers after the cursor position.



4. Selecting a proposal from the code completion list.

## Results

The corresponding keyword or identifier is added to the source code at this position.

## Related Information

- [Using Automatic Code Completion \[page 299\]](#)
- [Inserting a Full Signature Automatically \[page 301\]](#)
- [Inserting or Overwriting Source Code \[page 302\]](#)
- [Overriding Methods from Superclasses \[page 303\]](#)

### 5.1.16.1.3.1 Using Automatic Code Completion

In the *ABAP source code editor*, you want to quickly add a component (such as method of a class, field of a structure, etc.) to your code.

## Context

### i Note

Automatic code completion is selected by default. You can disable it here ► *Window* ► *Preferences* ► *ABAP Development* ► *Editors* ► *Source Code Editors* ► *Code Completion* ► *Automatically trigger code completion after typing -,-,>,>=,>,\* ►

## Procedure

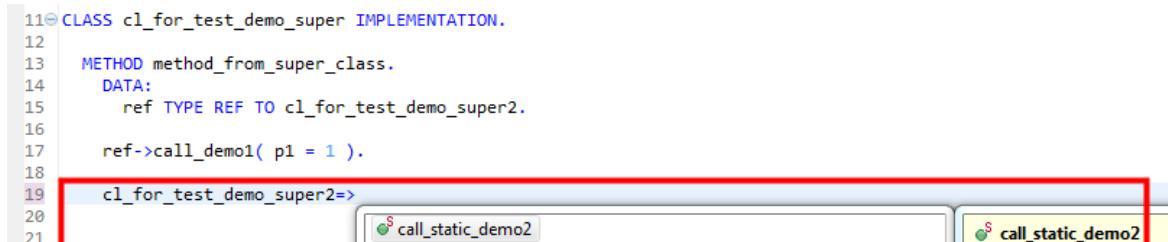
1. In the source code, start typing one of the following component selectors.

Component Selector	Displayed Proposals
-	Fields of structures
~	Components of interfaces
->	Instance components of object references
=>	Static components of classes / interfaces

### i Note

If you have enabled the following preference  *Window > Preferences > ABAP Development > Editors > Source Code Editors > Code Completion*  *Automatically replace '-' , '=>' , ' ~' , for class/interface components*  , you can just type '-'. This will be replaced by the appropriate component selector automatically.

The code completion list is automatically displayed.



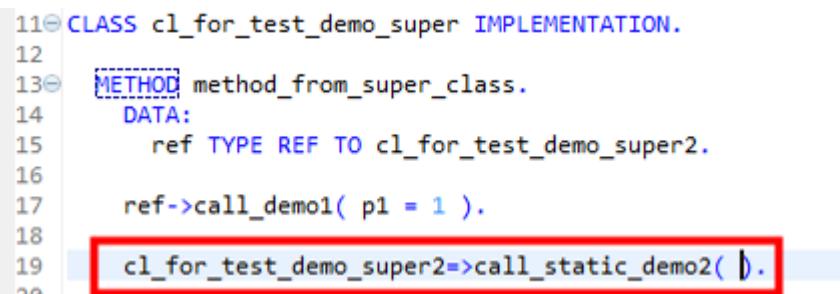
```
11 CLASS cl_for_test_demo_super IMPLEMENTATION.  
12  
13 METHOD method_from_super_class.  
14 DATA:  
15     ref TYPE REF TO cl_for_test_demo_super2.  
16  
17     ref->call_demo1( p1 = 1 ).  
18  
19     cl_for_test_demo2=>  
20  
21
```

Code completion list and popup according to the component

2. Select a component from the code completion list.

## Results

The component is added to the source code at the current position.



```
11 CLASS cl_for_test_demo_super IMPLEMENTATION.  
12  
13 METHOD method_from_super_class.  
14 DATA:  
15     ref TYPE REF TO cl_for_test_demo_super2.  
16  
17     ref->call_demo1( p1 = 1 ).  
18  
19     cl_for_test_demo2=>call_static_demo2( ).  
20
```

A call to the static call\_static\_demo2 method is inserted at your current position

## 5.1.16.1.3.2 Inserting a Full Signature Automatically

In the [ABAP source code editor](#) of a development object, you want to insert the full signature of a method, function module, or form routine automatically.

### Prerequisites

To use this function, open the  [ABAP Development](#)  [Editors](#)  [Source Code Editors](#)  [Code Completion](#)  [Typing](#)  preference page and select the *Always insert full signature on completion* option.

### Context

#### Note

Automatic insert of a full signature is disabled by default.

### Procedure

1. In the source code editor, type the name of a method, function module, or form routine.  
The code completion list with the available entries and full signature is displayed.
2. Select the relevant entry.

### Results

The full signature is inserted in the source code at the current position.

### 5.1.16.1.3.3 Inserting or Overwriting Source Code

In the *ABAP source code editor*, you can insert a keyword, identifier, or template from the code completion list at the current position. Optionally, you can configure the code completion to overwrite the keyword or identifier at the current position.

#### Prerequisites

To overwrite the keyword or identifier at the current position, choose the *ABAP Development* *Editors* *Source Code Editors* *Code Completion* *Content Assist* *Completion overwrites* preference in advance.

#### Context

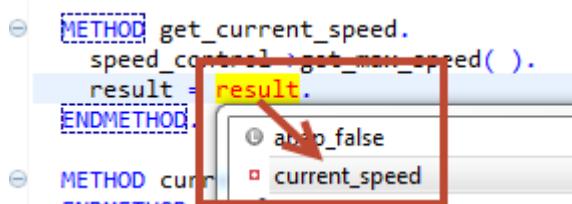
##### i Note

Code insertion is defined by default.

#### Procedure

1. In the source code editor, type the beginning character(s).
2. To display the list of suggested entries at this position, choose *Ctrl* + *Space*.

The code completion list is opened.



Example for inserting the identifier `current_speed` and replacing it with the existing identifier `result`

3. Adopt the displayed entry into your source code by choosing the *tabulator* key.

##### i Note

To toggle between inserting and overwriting while the content assist is active, choose *Ctrl*.

## Results

The selected keyword, identifier, or template is inserted and replaces the keyword or identifier at the current cursor position.

```
 METHOD get_current_speed.  
   speed_control->get_max_speed( ).  
   result = current_speed  
 ENDMETHOD.
```

Example after replacing the identifier result with current\_speed

### 5.1.16.1.3.4 Overriding Methods from Superclasses

You can generate a code snippet for the method implementation and its redefinition in one step.

#### Context

You want to override or extend a method from a superclass.

#### Procedure

1. In the implementation of an ABAP class, you can add the relevant code snippet at the following positions:
  - If the class does not yet contain any methods, position the cursor between the CLASS implementation and ENDCCLASS statement.
  - Otherwise, position the cursor between the
    - CLASS implementation and METHOD statement
    - ENDMETHOD and METHOD statement
    - ENDMETHOD and ENDCCLASS statement
2. Trigger code completion choosing **Ctrl** + **Space**.

A list with all methods that can be overridden is displayed.

The screenshot shows an ABAP code editor with the following code:

```
1 CLASS cl_car DEFINITION PUBLIC INHERITING FROM cl_demo_1
2
3   PUBLIC SECTION.
4
5     METHODS: drive, stop.
6
7   ENDCLASS.
8
9 CLASS cl_car IMPLEMENTATION
10
11   METHOD drive.
12     ENDMETHOD.
13
14
15
16
17 END
```

A code completion dropdown is open at line 9, position 11, with the text "METHOD drive." highlighted. The dropdown contains two items:

- Override 'START'
- Override 'MOVE'

Two red callout boxes are overlaid on the screenshot:

- A red callout box points to the line "9 CLASS cl\_car IMPLEMENTATION" with the text "Possible positions to trigger code completion".
- A red callout box points to the dropdown list with the text "Available methods to override".

At the bottom of the dropdown, the text "Press 'Shift+Enter' to insert full signature" is visible.

List that displays the superclass methods that can be overridden, triggered by code completion

3. Choose the relevant method.

## Results

The relevant method implementation, together with its redefinition, is added to your ABAP class. Its visibility is the same as the corresponding definition of the superclass.

```
► C CL_CAR ► START
1 CLASS cl_car DEFINITION PUBLIC INHERITING FROM cl_demo_
2
3 PUBLIC SECTION.
4 methods start redefinition.
5
6 METHODS: drive, stop.
7
8 ENDCLASS.
9
10 CLASS cl_car IMPLEMENTATION.
11
12 METHOD drive.
13 ENDMETHOD.
14
15 method start.
16
17 endmethod.
18
19 METHOD stop.
20 ENDMETHOD.
21
22 ENDCLASS.
```

Added  
redefinition

Added  
method block

Example of an ABAP class that now contains the redefinition and the method block of an inherited method from the superclass

#### i Note

The supercall is not added by default.

If you want to keep the behavior of the method as it is defined in the superclass, you have to add the supercall to the corresponding method implementation manually.

## 5.1.16.2 Working with Source Code Templates

### Context

Code templates reduce the time spent on routine coding. The new ABAP IDE provides a number of predefined code templates for ABAP. In addition, you can create further templates for your own use.

### Procedure

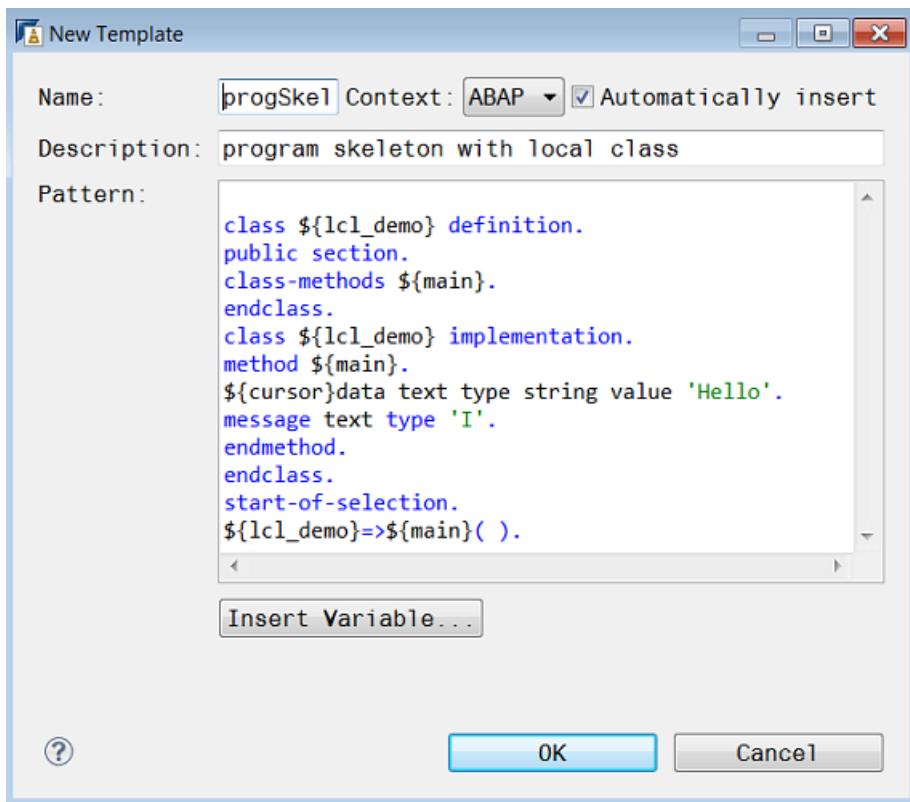
1. To search for all ABAP templates available...

- a. Open the preferences page    from the menu bar.
2. To use a template...
  - a. In the source code editor, write the beginning characters of the template.
  - b. Select **Ctrl + Space** (code completion).

→ Tip

Alternatively, you can use the *Templates* view to insert the template code through **Drag & Drop**.

3. To add a new template...
  - a. Open the ABAP source editor for any arbitrary development object.
  - b. Switch to *Templates* view.
  - c. In the toolbar of the *Templates* view, click the **Create a New Template** icon.
  - d. Specify the *Name*, the *Description* and the *Pattern* for the template.



Adding an new ABAP template

- e. Improve the format for the source code template manually.
- f. Save the new template with **OK**.

## Related Information

[ABAP Development Objects \[page 19\]](#)

[Adding Code Templates to an ABAP Exception Class \[page 190\]](#)

## 5.1.16.3 Closing Brackets and Literals Automatically

### Context

Whenever you enter an opening bracket such as:

- ( or { or [
- or a literal such as ", ``, ||,

the relevant closing character will be automatically inserted in the editor automatically directly behind the cursor, so you do not need to take care of it.

#### i Note

The feature for closing brackets and literals is enabled by default. If you wish to disable it, you have to switch off the corresponding setting in the preferences page:

► *Window* ► *Preferences* ► *ABAP Development* ► *Editors* ► *Source Code Editors* ► *Code Completion* ► *Automatically close brackets and literals* ▾

### Related Information

[ABAP Development Objects \[page 19\]](#)

## 5.1.16.4 Formatting ABAP Source Code

You use the ABAP formatter to keep ABAP source code in a correct format and well organized in order to develop new functions easier.

### Context

Formatting is always user-defined for a specific ABAP project. The formatting options are shared between the ABAP Workbench and ABAP Development Tools (ADT) in the same datasource.

#### i Note

In Eclipse, the name of the pretty printer functionality is *Source formatter*.

You can do the formatting from the editor of a development object in the following ways:

1. To format the whole ABAP source code, use the ► *Source Code* ► *Format* ▾ context menu or the `Shift` + `F1` shortcut.

2. To format selected source code blocks, select them and use the  context menu or the `Ctrl` + `Shift` + `F1` shortcut.

## Related Information

[Defining ABAP Formatting Options \[page 308\]](#)

### 5.1.16.4.1 Defining ABAP Formatting Options

You can define your own options for formatting ABAP keywords and identifiers from the source-based development objects of an ABAP project. These formatting options are user-specific and can be defined for each ABAP project.

#### Context

You can change these options or define your own formatting options in ABAP Development Tools (ADT) using the [ABAP Formatter](#) preference page.

#### Procedure

1. Open the [ABAP Formatter](#) preferences page using one of the following ways:
  - In the [Project Explorer](#), select the relevant ABAP project and select [Properties](#) from the context menu.
  - From the menu bar, choose  From here, open the  tree and select the [ABAP Formatter](#) link. You then have to choose the relevant ABAP project.

The [ABAP Formatter](#) preferences page is opened.

2. To specify the alignment of the source code, select the [Indent Lines](#) checkbox in the [Indention](#) area.
3. In the [Upper/Lower Case Conversion](#) area, you define the formatting of ABAP keywords and identifiers. Choose one of the following radio buttons:

- Select the [None](#) radio button to adopt other formatting options.
- Select the [Derived from First Statement](#) radio button to accept the formatting options used for the ABAP keywords and identifiers in the first statement of the development object you are currently working on. ADT then uses these formatting options for subsequent ABAP keywords and identifiers.
- Select the [Custom](#) radio button if you want to define your own formatting options.

If you have chosen the [Custom](#) radio button, the [Keywords](#) and [Identifiers](#) areas become active.

4. In each area, select the relevant radio button to format the relevant source code in upper or lower case.

The example code snippet in the *Preview* area is formatted in accordance to your selection.

5. Choose *OK* to confirm.

## Results

The formatting option is saved specifically for your user and the selected system connection in the back-end. This means, the formatting options you have modified in ADT will also be considered when you use the pretty printer functionality in the ABAP Workbench and vice versa.

## Related Information

[Formatting ABAP Source Code \[page 307\]](#)

## 5.1.16.5 Changing Colors for ABAP Source Code

### Context

You can define the color for how text or annotation types are displayed in ABAP source code.

For this, you have to set the corresponding preferences in order to:

- [Changing the Font Color of Texts \[page 310\]](#)
- [Changing The Color of Individual ABAP Keywords \[page 311\]](#)
- [Changing Colors of Annotation Types \[page 313\]](#)

#### i Note

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

The new color settings become effective as soon as you (re-)open the source code editor.

## Related Information

[Formatting ABAP Source Code \[page 307\]](#)

[Marking Occurrences in the Source Code Editor \[page 331\]](#)

## 5.1.16.5.1 Changing the Font Color of Texts

### Context

You can define the colors in which the following texts are to be displayed in ABAP source code:

Entry	Description
Search	To highlight search results with write access
Syntax Coloring	To highlight, for example, ABAP keywords, literals, identifiers in general, as well as the background color of ABAP method blocks and embedded language elements, such as SQL script
	<p><b>i Note</b></p> <p>To define colors for individual ABAP keywords, see the Related Information from below.</p>
Transformation	To highlight, for example, specific XML attributes, such as equal signs or values as well as comments that are defined for transformations
WebDynpro	To highlight, for example, generated code in general, mapping lines regarding context mapping, and WebDynpro error texts as well as their formatting

Using different colors makes it easier for you to skim through source code in order to distinguish, for example, between an ABAP keyword and elements.

### Procedure

1. Open the    preference page.
2. Expand the folders  
3. Select the text type whose color you want to change and click the *Edit...* button.
4. Select the color and confirm with *OK*.

**i Note**

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

5. To make the new color settings effective, (re-)open the source code editor.

## Results

In the source code editor, the corresponding source code parts are displayed or highlighted in the predefined color settings.

## Related Information

[Changing The Color of Individual ABAP Keywords \[page 311\]](#)

[Formatting ABAP Source Code \[page 307\]](#)

[Changing Colors of Annotation Types \[page 313\]](#)

### 5.1.16.5.2 Changing The Color of Individual ABAP Keywords

You can configure the color of an individual ABAP keyword or the sequence of several ABAP keywords, for example, `call method`, used within a statement. This enables you to highlight their occurrences and to improve readability when you skim through ABAP source code objects, such as ABAP classes, interfaces, programs, and so on.

## Procedure

1. Open the  [ABAP Development](#)  [Editors](#)  [Source Code Editors](#)  [ABAP Keyword Colors](#)  [Preferences](#) page.

The [ABAP Keyword Colors](#) preference page is opened. Here you find a list of default configurations that have already been configured by SAP.

2. To start configuring the highlighting of a new ABAP keyword or sequence, chose the [Add](#) button.

An empty entry is added at the end of the list. Its default color is black.

The cursor is automatically moved to the corresponding position.

3. Start typing the name of the ABAP keyword or sequence you want to configure.

#### Note

You can use the content assist functionality ( + ) to open a dialog. It lists all the available ABAP keywords. From here you can select the relevant one.

You can use the '\*' wildcard to search for any string.

4. To configure the color, choose the relevant ABAP keyword or sequence in the list and click the colored button in the [Color](#) section on the right.

#### Tip

You can group configurations that are used in the same context by using the same color for them.

The *Color* dialog is opened.

5. Select the color and confirm with *OK*.

In the list, your ABAP keyword or sequence is displayed in accordance with your selection.

6. To make your configuration(s) visible in the source code editor, choose *OK*.

Your color configurations become effective as soon as you (re-)open the source code editor.

```
1① CLASS cl_car_constructor_generator DEFINITION
2   PUBLIC
3   FINAL
4   CREATE PUBLIC .
5   PUBLIC SECTION.
6     CLASS-METHODS create
7       IMPORTING
8         i_current_speed TYPE i
9         value(test) TYPE i OPTIONAL
10      RETURNING
11        value(r_result) TYPE REF TO cl_car_constructor_generator
12        DATA current_speed TYPE i.
13     METHODS drive IMPORTING i_speed TYPE i.
14   PROTECTED SECTION.
15   PRIVATE SECTION.
16 ENDCLASS.
17
18② CLASS cl_car_constructor_generator IMPLEMENTATION.
19
20  METHOD create.
21    DATA: r_result_1 TYPE REF TO cl_car_constructor_generator.
22
23    CREATE OBJECT r_result_1
24    r_result_1->current_speed = i_current_speed.
25    r_result_1->value(test) = 0.
26
27  ENDMETHOD.
```

Example of an ABAP class where the syntax coloring of single ABAP keywords is configured

7. In addition, you have the following options:

- To make SAP's default configurations visible in the source code editor, select the relevant checkbox(es) in the list or choose the *Select all* button on the right. Then click the *Apply* button.

SAP's default color configurations become effective as soon as you (re-)open the source code editor.

- To highlight the style of ABAP keywords and sequences, select the relevant checkbox in the list and choose one of the following checkboxes on the right:

- Bold
- Italic
- Strikeout
- Underline

In the list, your ABAP keyword or sequence is displayed in accordance with your selection.

- c. To import or export a configuration as an XML file, choose the *Import* or *Export* button on the right.

The *Export color settings* dialog is opened.

- d. Choose your file location and confirm with *Open* or *Save*.

The configuration file is imported and its content is now displayed or the relevant file is exported accordingly.

- e. To take over the configuration of another ABAP keyword or sequence, select the relevant entry in the list and choose *Pick settings from* from the context menu.

A dialog with the already configured ABAP keywords or sequences is opened.

- f. From here, choose the relevant entry.

In the list, your ABAP keyword or sequence is displayed in accordance with your selection.

- g. To restore the default color settings, click the *Restore Defaults* button.

#### **i Note**

If you perform the restore action, all of your configurations will be deleted.

### **5.1.16.5.3 Changing Colors of Annotation Types**

#### **Context**

You can define the coloring of annotation types in the source code. These are, for example:

- ABAP block highlighting
- ABAP coverage
- ABAP debug exceptions
- Breakpoints
- And so on

This enables you, highlight annotation types in your personal color.

#### **Procedure**

1. Open the  *General*  *Editors*  *Text Editors*  preference page.
2. Select the annotation type you want to change and choose the **Color** button.
3. Select the color and confirm with **OK**.

#### **i Note**

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

## Results

The new color settings become effective as soon as you (re-)open the source code editor.

## Related Information

[Changing the Font Color of Texts \[page 310\]](#)

[Marking Occurrences in the Source Code Editor \[page 331\]](#)

[Changing Colors for ABAP Source Code \[page 309\]](#)

## 5.1.16.6 Editing Comments in ABAP Source Code

### Context

You have the following options for commenting source code:

- ABAP comments to explain a particular part or section of source code
- ABAP Doc comments to describe code elements (for example attributes, methods, local variables) in the code element information popup, ABAP Element Info view, and in the code completion list

## Related Information

[Element Info \[page 33\]](#)

[ABAP Doc Comments \[page 29\]](#)

[Editing ABAP Doc Comments \[page 314\]](#)

[Editing ABAP Comments \[page 328\]](#)

## 5.1.16.6.1 Editing ABAP Doc Comments

You can add, edit, format, as well as import or export ABAP Doc comments to document source code.

## Related Information

[ABAP Doc Comments \[page 29\]](#)

[Adding ABAP Doc Comments \[page 315\]](#)

[Adding Documentation of Parameters and Exceptions in ABAP Doc Comments \[page 317\]](#)

[Formatting ABAP Doc Comments \[page 318\]](#)

[Linking ABAP Repository Objects and Its Components \[page 320\]](#)

[Synchronizing ABAP Doc Comments \[page 321\]](#)

[Importing ABAP Doc Descriptions from the Class Builder \[page 323\]](#)

[Exporting ABAP Doc Comments \[page 325\]](#)

## 5.1.16.6.1.1 Adding ABAP Doc Comments

A single ABAP Doc comment or a block of several ABAP Doc comments is introduced by the character combination "!".

### Positioning ABAP Doc Comments in the Definition

In the editor, ABAP Doc comments are added one row above the related element in the definition part.

So, the ABAP Doc comment has to start before the element, directly in front of a declarative statement (for example, data declaration, method definition, class definition). Otherwise a warning will be displayed because the source code editor verifies the position and the content structure of ABAP Doc comments when you execute the **ABAP syntax check**. So, if comments are added at the wrong position or contain incorrect syntax, a warning is displayed in the *Problems* view.

#### • Example

Example of a single ABAP Doc comment:

```
!" ABAP Doc test
METHODS method_with_variable.
```

Example of an ABAP Doc of a constant:

```
!" This is documentation for the following constant
CONSTANTS co_initial_value TYPE i VALUE 0.
```

## Using a Quick Fix

In the implementation of a global class, you can document a method. You can use the *Add ABAP Doc* quick assist to add the `<p class="shorttext synchronized">` tag. Then an ABAP Doc comment block containing the relevant tag is added.

To do this, proceed as follows:

1. In the definition, position the cursor on the method name for which you want to perform `Ctrl + 1`.  
The *Quick Fix* dialog box is opened.

2. Choose [Add ABAP Doc](#).

An ABAP Doc comment block is added to your source code, which contains the relevant tag:

```
  !! <p class="shorttext synchronized" lang="en"></p>
  !!
```

Enter the relevant short text before the closing `</p>` tag.

## Adding Statements over Multiple Lines

You can use multiple lines to document source code elements. In this case, you have to add the character combination "!" in front of each line.

### Example

Example for a multi line constant definition:

```
  !! This documentation for the following constant is documented
  !! in multiple lines.
CONSTANTS co_initial_value TYPE i VALUE 0.
```

## Documenting a Block of Statements

If you want to document a block of statements using the ABAP colon comma semantics, the ABAP Doc comment block must be located in front of the identifier and after the colon.

### Example

```
CONSTANTS:
  !! Initial value
  co_initial_value TYPE i VALUE 0,
  !! Invalid value
  co_invalid_value TYPE i VALUE -1.
```

## Related Information

[Adding Documentation of Parameters and Exceptions in ABAP Doc Comments \[page 317\]](#)

[Synchronizing ABAP Doc Comments \[page 321\]](#)

## 5.1.16.6.1.2 Adding Documentation of Parameters and Exceptions in ABAP Doc Comments

You can document parameters and exceptions for methods, events, functions modules, and subroutines.

### Context

This function supports you in listing all undocumented parameters and exceptions that relate to the selected element.

#### i Note

If you have already added parameters in an ABAP Doc comment and made changes in the meantime, you can also add the new parameters with this function. Thus, only the new parameters will be added. Parameters and exceptions that are already documented will not be changed.

In the ABAP Language Help (through F1 in the source editor), you will find further information about using the syntax of ABAP Doc comments.

### Procedure

1. In the source code editor, navigate to the **definition part** of the element of the relevant code line.
2. [Optional]: If no ABAP Doc comment exists, enter "!" to start the comment line.

#### ❖ Example

Example of an empty ABAP Doc comment:

```
!"  
METHODS method_with_variable.
```

#### i Note

Choose [Enter](#) to add a new row for further comments. The editor will automatically start the new row with "!"

3. Place the cursor on the position where you want to add the missing parameters or exception.
4. In the context menu, choose [Quick Fix \(Ctrl 1\)](#), and select [Add missing parameters to documentation](#) to add the syntax of the missing parameters and exceptions.  
The name of the missing parameters are added and introduced by a @. You can describe each parameter or exception with the corresponding documentation after the |.

#### i Note

Every parameter or exception should occupy a separate line in order to keep the source code readable.

## • Example

Example of a type definition added as an ABAP Doc comment:

```
  !@parameter p1 |
METHODS method_with_variable.
```

5. Save the changes.

## Results

The documentation is saved and displayed in the source code editor of the ADT.

## Related Information

[Adding ABAP Doc Comments \[page 315\]](#)

[Importing ABAP Doc Descriptions from the Class Builder \[page 323\]](#)

[Creating ABAP Exception Classes \[page 187\]](#)

## 5.1.16.6.1.3 Formatting ABAP Doc Comments

You can format ABAP Doc comments with XHTML tags.

## Context

This enables you to structure and highlight parts of your comments that are rendered, for example, into HTML files or the [ABAP Element Info](#) view.

You can use the following XHTML tags to format text:

Formatting option	Tag	Alternative Tag
Line break	 	 </br>
Paragraph	<p>...</p>	
Emphasized text	<em>...</em>	
Strong emphasized text	<strong>...</strong>	
Unsorted lists	<ul><li>...</li></ul>	
Sorted lists	<ol><li>...</li></ol>	

Formatting option	Tag	Alternative Tag
Headers	<h1>...</h1><h2>...</h2> <h3>...</h3>	

### i Note

In addition, ABAP Development Tools (ADT) analyzes the content of ABAP Doc comments, converting it to HTML, and displays it appropriately. Consequently, special characters such as ", ', <, > and @ must be escaped using &quot;, &apos;, &lt;, &gt;, and &#64;.

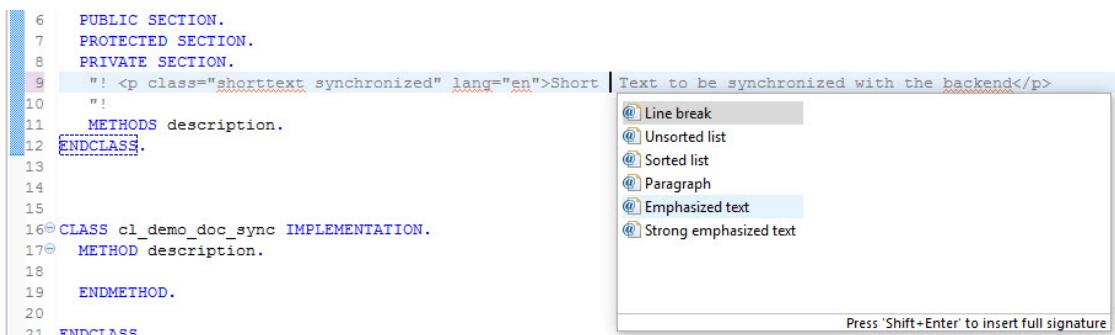
### • Example

"Test class" will be formatted as cursive:

```
!"<em>Test class</em> for demo
class CL_FOR_TEST_DEMO definition
public
final
create public.
```

## Procedure

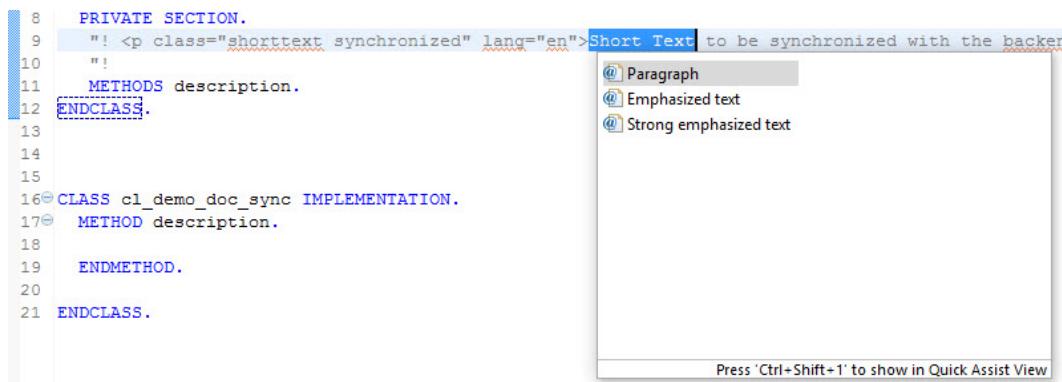
1. You can use code completion to add formatting to ABAP doc comments:
  - a. To do this, position the cursor at the corresponding position within the comment and press **Ctrl** + **Space**.
  - b. Choose then the relevant shortcut from the popup.



Example for adapting ABAP doc comments using code completion

As a result, the corresponding HTML tag is added to your comment.

2. You can also use quick fixes to adapt formatting of existing comments:
  - a. To do this, select the text to be formatted within the comment and press **Ctrl** + **1**.
  - b. Choose then the relevant quick fix from the popup.



```

8 PRIVATE SECTION.
9   " ! <p class="shorttext synchronized" lang="en">Short Text to be synchronized with the backend</p>
10  " !
11  METHODS description.
12  ENDCLASS.
13
14
15
16 CLASS cl_demo_doc_sync IMPLEMENTATION.
17   METHOD description.
18
19   ENDMETHOD.
20
21 ENDCLASS.

```

Press 'Ctrl+Shift+1' to show in Quick Assist View

Example for highlighting text in ABAP doc comments using quick fixes

As a result, the selected text is surrounded with the corresponding HTML tag.

## Related Information

[Adding ABAP Doc Comments \[page 315\]](#)

[Adding Documentation of Parameters and Exceptions in ABAP Doc Comments \[page 317\]](#)

## 5.1.16.6.1.4 Linking ABAP Repository Objects and Its Components

You can add links to ABAP repository objects or to the components of ABAP repository objects in ABAP Doc comments.

## Context

This enables you to open ABAP Doc comments directly from the *Element Information* popup and the *ABAP Element Info* view. In addition, you can trigger the where-used list from linked object or component names.

You can provide such links for the following repository objects and components:

- Global types, such as ABAP classes, interfaces, database tables, views, structures
- Components, such as methods,

### Code Syntax

local/public types

- Other objects, requiring a prefix, such as ABAP function groups, domains, XSLT transformations, and so on.

## Procedure

1. Add an ABAP Doc comment before the relevant statement using the " ! tags. Alternatively, you can position the cursor at the relevant position within an ABAP Doc comment.
2. Add the ... {@link [[kind:]name.]...}[kind:]name} ... syntax.
3. Enter the name of the ABAP repository object or its component to be opened.

## Related Information

 [ABAP Doc \(ABAP Keyword Documentation\)](#)  
[ABAP Doc Comments \[page 29\]](#)  
[Adding ABAP Doc Comments \[page 315\]](#)

## 5.1.16.6.1.5 Synchronizing ABAP Doc Comments

You can synchronize ABAP Doc Comments with the short descriptions.

## Prerequisites

- ABAP classes or ABAP interfaces must contain short descriptions to synchronize short descriptions with ABAP Doc Comments.
- You can only import and synchronize in the original language.

### → Recommendation

In accordance with SAP programming guidelines, you should write program comments in English. So you should be logged on in English as your ABAP project language.

- If ABAP Doc contains special characters such as @ " ! < > - >, you need to replace them by its ACSCII code. Otherwise, these special characters are not imported.
- Short texts may not contain more than 60 characters.

## Context

This enables you to keep the documentation of the ABAP source code between ADT and the ABAP Workbench consistent.

Method	Level	Visibility	M...	Description
DESCRIPTION	Instance	Method	Private	Short Text to be synchronized with the backend

Example of short text displayed in transaction SE24

You have the following possibilities to synchronize changes of ABAP Doc comments in ADT with the Class Builder in the back end:

- You can surround an ABAP Doc comment with the `<p class="short text synchronized">` tag in the ABAP source code editor.  
Then, ABAP Doc comments are directly synchronized when you save the corresponding object in ADT.

#### i Note

When documenting methods in the definition of ABAP classes, you can use the *Add ABAP Doc* quick assist for adding the relevant tag as ABAP Doc comment block. For further information, see the Related Information below.

- You can import ABAP Doc from descriptions into ADT.  
Then, the `<p class="short text synchronized">` tag is automatically added to the ABAP Doc comments when you import for the first time. So, the imported ABAP Doc comments are synchronized when you save changes.

#### ❖ Example

Example for providing a short text to be synchronized with the back end in the ABAP source code editor:

```
!<p class="short text synchronized" lang="en">Short Text to be synchronized
with the backend</p>
!"
```

## Related Information

[Adding ABAP Doc Comments \[page 315\]](#)

[Importing ABAP Doc Descriptions from the Class Builder \[page 323\]](#)

## 5.1.16.6.1.6 Importing ABAP Doc Descriptions from the Class Builder

In ABAP Development Tools (ADT), you can import descriptions of ABAP classes or ABAP interfaces (including their attributes, methods, parameters, and so on) that are entered in the form-based editor of the Class Builder. This enables you, to add, edit, or display them also in ABAP Development Tools (ADT).

### Prerequisites

- The descriptions of components for ABAP classes or ABAP interfaces are entered in the form-based Class Builder of the ABAP Workbench.
- You can only import and synchronize in the original language.

#### → Recommendation

In accordance with the SAP programming guidelines, you should write program comments in English. So, you should be logged on in English as your ABAP project language.

- If ABAP Doc contains special characters such as @ “ ! < > - > , you need to replace them by its ACSCII code. Otherwise these special characters are not imported.
- Short texts may not contain more than 60 characters.

### Context

#### i Note

You can import them only once when you edit them at the first time in ADT.

The source code editor does not overwrite existing ABAP Doc comments for class or interface components that are already documented with ABAP Doc.

#### • Example

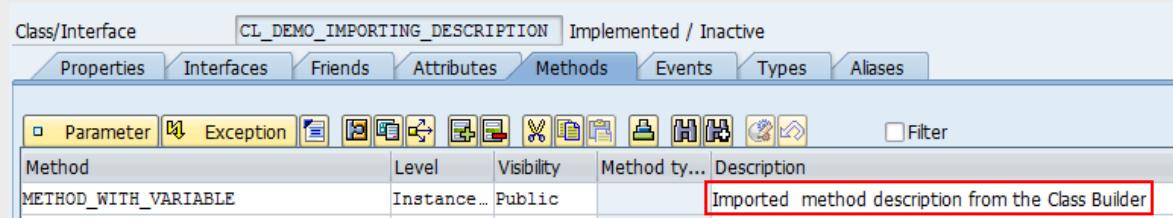
In the source code editor of ADT, the definition of the `cl_demo_importing_description` ABAP class is displayed as follows:

```
CLASS cl_demo_importing_description DEFINITION.
  PUBLIC
  FINAL
  CREATE PUBLIC.

  PUBLIC SECTION.

  PROTECTED SECTION.
  PRIVATE SECTION.
    DATA: myself TYPE REF TO cl_demo_importing_description.
    DATA any_string1 TYPE c.
    METHODS method_with_variable.
ENDCLASS.
```

In the form-based Class Builder of the ABAP Workbench, the following description is saved for the `METHOD_WITH_VARIABLE` method of the same class:



Method	Level	Visibility	Method ty...	Description
<code>METHOD_WITH_VARIABLE</code>	Instance...	Public		Imported method description from the Class Builder

Example of a method description that can be reused as ABAP Doc in ADT

## Procedure

1. In ADT, open the context menu in the source code editor of a class or interface.
2. Choose **Source Code** **Import ABAP Doc from Descriptions**.

## Results

The source code editor imports all descriptions of the class or interface components from the Class Builder and inserts them as ABAP Doc comments at the appropriate position in the source code editor of ADT.

If you import descriptions in SAP NetWeaver 7.5 SP00 or higher, the corresponding ABAP Doc comments are surrounded with a `<p class="shorttext synchronized">` tag. This tag defines possible changes are directly synchronized in the back-end when you save the object.

### Example

In the source code of the `cl_demo_importing_description` ABAP class, the imported descriptions of the components are added as ABAP Doc comments:

```

"! <p class="shorttext synchronized" lang="en">Imported class description</p>
CLASS cl_demo_importing_description DEFINITION.
  PUBLIC
  FINAL
  CREATE PUBLIC.

  PUBLIC SECTION.

  PROTECTED SECTION.
  PRIVATE SECTION.
  "! <p class="shorttext synchronized" lang="en">Imported attribute description</p>
  DATA: myself TYPE REF TO cl_demo_importing_description.
  "! <p class="shorttext synchronized" lang="en">Imported attribute description</p>
  DATA any_string1 TYPE c.
  "! <p class="shorttext synchronized" lang="en">Imported method description from the Class
Builder</p>
  METHODS method_with_variable.
ENDCLASS.

```

## Related Information

[Synchronizing ABAP Doc Comments \[page 321\]](#)

[Exporting ABAP Doc Comments \[page 325\]](#)

### 5.1.16.6.1.7 Exporting ABAP Doc Comments

You can export source code documentation of ABAP classes and ABAP interfaces, for example, in order to create a backup of this information for audits.

#### Context

For each class or interface you can create an HTML file as follows:

#### Procedure

1. Open the *Export* wizard.

Here you have the following options:

- In the *Project Explorer*, open the context menu in any ABAP project, package, class, or interface and choose *Export*.  
Note that you can also select several packages, classes, and interfaces.
- Choose  *File*  *Export*  from the menu bar.

The *Export* wizard is opened.

2. In the displayed tree, select  .
3. Choose *Next*.  
The second page is opened.
4. Choose an ABAP project.

Here you have the following options:

- Enter the name of the project in the *Project* field.

#### → Tip

When editing the project, you can avail of the content assistant functionality by pressing **Ctrl + Space**.

- Choose *Browse...* to search for a specific project.

5. Choose the ABAP package(s), class(es), or interface(s) you want to export.

Here you have the following options:

- To add, choose *Add....*
- To remove one or more entries, choose *Remove...*

### **i** Note

The maximum number of objects for one export job is 4000. If the maximum number is exceeded, no ABAP Doc comments are exported and an error is displayed in the export log.

6. To define the set of members to be exported for classes, select one option in the *Visibility of members to be exported* area.
7. To export short texts if no ABAP Doc comment is provided for an element, choose the corresponding checkbox in the *Export mode* area.
8. To define the directory where you want to save the export files, choose *Browse...* in the *Destination* area.
9. Choose *Finish* to start the export.

## Results

The export job is started. If you select a package with a huge number of classes and interfaces, for example, the job might last several minutes. You can check the state of the export job in the *Console* and *Progress* view.

When the job is finished, the export log is displayed in the *Console* view. At the same time, a new folder with a unique name is created in the chosen directory. The export files are stored here.

## • Example

The screenshot shows an ABAP export file with the following structure and annotations:

- Attributes | Methods**: Navigation bar.
- Class CL\_METHOD\_CHECK\_AREA**: Name of the selected ABAP class.
- Documentation**: This class checks the...
- Attributes**: Exported members according to the chosen visibility.
- Visibility and Level** table:

Visibility and Level	Name	Documentation
public instance	length type I	Length of the area
public instance	width type I	Width of the area
- Methods**: ABAP Doc or short text depending on the chosen export mode.
- Generated on 25.06.2015**

Example of an Export File that Contains ABAP Documentation

In addition, a popup is displayed from where you can open the export folder directly.

### i Note

To adopt the layout of the exported html file, open and edit the `stylesheet.css` file. It is saved in the `\htmldesign` path.

### i Note

If the exported ABAP Doc comments contain links like `{ @link cl_car.meth:drive }`, they will be rendered as simple text like `drive`.

Currently it is not possible to follow these links.

## Related Information

[Importing ABAP Doc Descriptions from the Class Builder \[page 323\]](#)

## 5.1.16.6.2 Editing ABAP Comments

### Context

ABAP comments start with \* or ".

#### i Note

You can also use the following shortcuts:

Shortcut	Function	Description
<b>Ctrl + &lt;</b>	Add Comment	The editor pastes an asterisk (*) at the beginning of the line.
<b>Ctrl + &gt;</b>	Remove Comment	The editor removes the existing asterisk (*) at the beginning of the line.
<b>Ctrl + 7</b>	Toggle Comment	The editor pastes or removes an asterisk (*) at the beginning of the line.

#### i Note

In addition, you have the following options for working with ABAP comments:

- In the context menu of the editor, choose *Source Code* and select the corresponding entry.
- In an ABAP class and procedure, enter an asterisk (\*) to split source code into logical sections.
- Enter an asterisk (\*) to disable ABAP statements.
- Enter double quotation marks ("") to describe source code within a line.

### Related Information

[Editing ABAP Doc Comments \[page 314\]](#)

## 5.1.16.7 Displaying Element Info in Source Code Editors

To display *Element Info* in source code editors, select an identifier such as the name of a method, a database table, or an interface and press **F2**.

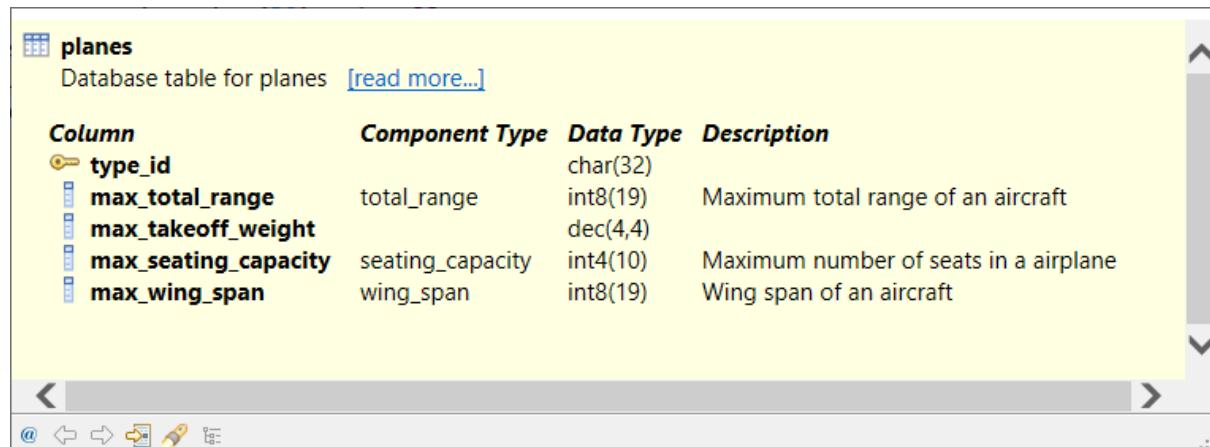
Alternatively, you can select:

- **Source Code** **Show Code Element Information** from the context menu on the element.
- **Window** **Show View** **Other...** **ABAP Element Info** from the menu bar.

#### i Note

To display *Element Info* for a method, press **Alt** + **F2** at any position in the method implementation.

The *Element Info* popup opens as shown in the following example.



Column	Component Type	Data Type	Description
type_id		char(32)	
max_total_range	total_range	int8(19)	Maximum total range of an aircraft
max_takeoff_weight		dec(4,4)	
max_seating_capacity	seating_capacity	int4(10)	Maximum number of seats in a airplane
max_wing_span	wing_span	int8(19)	Wing span of an aircraft

Element Info Popup

The *Element Info* popup provides the following features.

#### Element Info Popup Features

 @	Open the same information in the <a href="#">ABAP Element Info View</a> <a href="#">[page 329]</a>
Related object	Navigate to the related object. Related object name becomes a link if you hover over it
 (Alt + Left)	Go back to the previous display
 (Alt + Right)	Go forward to the next display
 (F3)	Open an element in the editor
 (Ctrl + F)	Search within the Element Info popup Press <b>Enter</b> or  to find the next occurrence of your search string Press <b>Shift</b> + <b>Enter</b> or  to find the previous occurrence of your search string
 <b>H</b>	Open hierarchical view to display includes and append structures
<b>Ctrl</b> + <b>+</b>	Increase font size
<b>Ctrl</b> + <b>-</b>	Reduce font size

## ABAP Element Info View

*ABAP Element Info* view gives you the possibility to display the *Element Info* permanently. This view also provides additional features, such as linking with selections , pinning views , and printing content .

ABAP Element Info

**planes**  
Database table for planes [\[read more...\]](#)

Column	Component Type	Data Type	Description
type_id		char(32)	
max_total_range	total_range	int8(19)	Maximum total range of an aircraft
max_takeoff_weight		dec(4,4)	
max_seating_capacity	seating_capacity	int4(10)	Maximum number of seats in a airplane
max_wing_span	wing_span	int8(19)	Wing span of an aircraft

ABAP Element Info View

## Code Completion/Content Assist

*Element Info* is displayed as a side view if you trigger code completion by **Ctrl** + **Space** on a relevant element.

\*CL\_DEMO\_PLANE

CL\_DEMO\_PLANE > TAKEOFF

```
70 METHOD takeoff.
71
72   SELECT max_takeoff_weight
73   FROM pla
```

Ctrl + space

planes

planets

plays

Press 'Shift+Enter' to insert full signature

planes

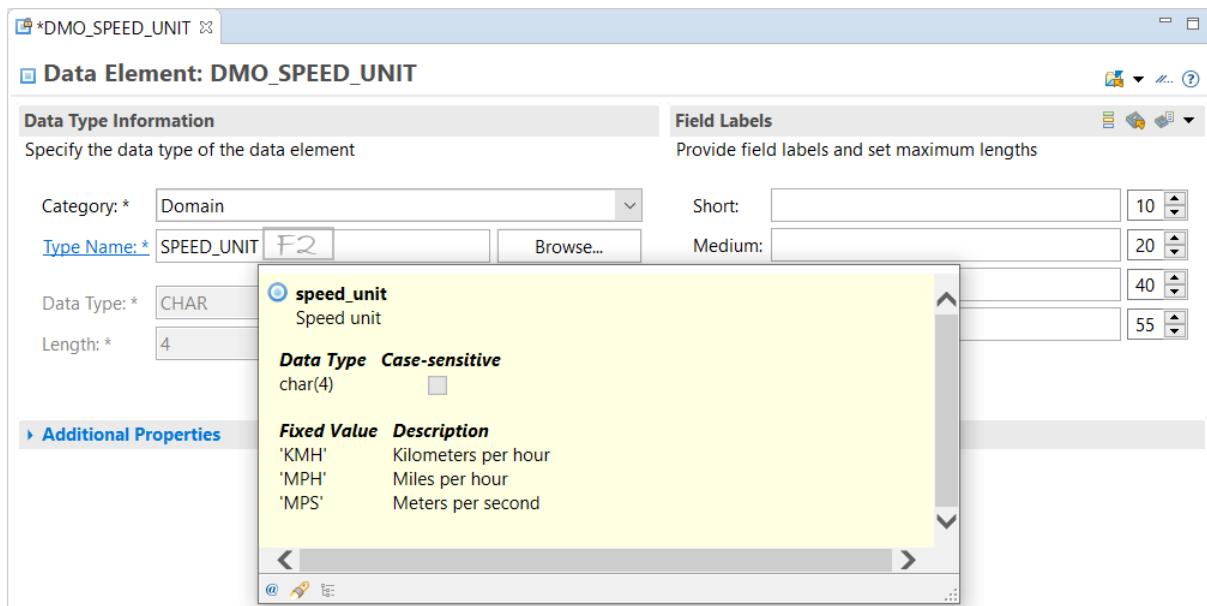
Database table for planes

Column	Component Type	Data Type
type_id		char(32)
max_total_range	total_range	int8(19)
max_takeoff_weight		dec(4,4)
max_seating_capacity	seating_capacity	int4(10)
max_wing_span	wing_span	int8(19)

Element Info Triggered by Code Completion

## Element Info in Form Editors

In form editors, *Element Info* can be displayed from the context menu entry *Show Element Information* or by pressing **F2** on the element.



Element Info in Data Element Editor

## Related Information

[Element Info \[page 33\]](#)

## 5.1.16.8 Marking Occurrences in the Source Code Editor

### Context

The ABAP source code editor provides a highlighting feature for occurrences of local variables and parameters. It allows you to perform the following activities:

- **To mark** the occurrences, click on the local variable or parameter in the editor.

```

76  valid = abap_true.
77  RETURN.
78  ENDIF.
79  IF strlen( object ) <= 4.
80  valid = abap_false.
81  RETURN.
82  ENDIF.
83  IF object(4) <> 'ABAP' AND
84    object(4) <> 'ABEN' AND
85    object(4) <> 'DYNP'.
86  valid = abap_false.
87  RETURN.
88  ENDIF.
89  cl_abap_docu_tree=>get_node_table( 'D' ).
90  READ TABLE cl_abap_docu_tables_broker=>root->node_table
91    WITH KEY node_key = object
92    TRANSPORTING NO FIELDS.
93  IF sy-subrc = 0.
94  valid = abap_true.
95  ENDIF.
96 ENDMETHOD.

```

Highlighting occurrences of local variables in the source editor

- **To navigate** between the occurrences choose the toolbar icons  (Next Annotation) and  (Previous Annotation).
- **To search for** occurrences, starting from the current position, open the context menu and choose the menu  **Source Code**  **Occurrences in File**  or press **Ctrl + Shift + U**.
- **To disable or enable** this feature, use the marker icon  in the toolbar.
- **To specify the preferences** for marking, choose  **Window**  **Preferences**  **ABAP Development**  **Source Code Editor**  **Mark Occurrences** 

## Related Information

[ABAP Development Objects \[page 19\]](#)

## 5.1.16.9 Viewing the Outline

### Context

The *Outline* view displays the internal structure of an ABAP class, interface, or program that is currently opened in the ABAP source code editor.

#### Activities from the Outline View Toolbar

The following activities are provided in the toolbar:

Icon	Description
	If you activate this filter, only public members of classes will be displayed as an outline. All non-public members of ABAP classes will remain hidden.
	If you activate this filter, only methods of ABAP classes or interfaces will be displayed as an outline. All members that are not methods, such as attributes, data types, and events will remain.

### Activities from the Context Menu of a Structuring Element

The following activities are provided from the context menu of an element in the outline tree:

Function	Description
Navigate to Declaration	Navigates to the declaration part in the source code editor.
Navigate to Implementation	Navigates to the implementation part of a class or test class in the source code editor.
Run As	Enables you to directly execute programs (F8) or launch test environments for classes, or to start ABAP unit tests. <b>See also:</b> <a href="#">Launching ABAP Unit Tests [page 551]</a>
Debug As	Enables you to start debugging ABAP code after having set a breakpoint. <b>See also:</b> <a href="#">Debugging ABAP Code [page 613]</a>
Profile As	Enables you to start profiling (ABAP Traces) for the ABAP program. <b>See also:</b> <a href="#">Profiling ABAP Code [page 640]</a>
Coverage As	Enables you to start a code coverage measurement. <b>See also:</b> <a href="#">Evaluating ABAP Unit Code Coverage Results [page 571]</a>
Get Where-used List	Enables you to determine the development objects that use a specific object. <b>See also:</b> <a href="#">Where-Used Function [page 43]</a> <a href="#">Searching Usages (Where-Used) [page 293]</a>
Rename	Enables you to find and rename variables, methods, and form routine names, as well as other elements and form routine names, as well as other elements of an ABAP program. ABAP classes and ABAP interfaces can be renamed as well. <b>See also:</b> <a href="#">Renaming Identifiers [page 343]</a>

## Related Information

[Outline View \[page 80\]](#)

[Advantages of Using the Outline View \[page 82\]](#)

[Icons in the Outline View \[page 83\]](#)

[Tools for ABAP Development \[page 19\]](#)

[Using Quick Views \[page 337\]](#)

[Viewing the ABAP Type Hierarchy \[page 334\]](#)

## 5.1.16.10 Viewing the ABAP Type Hierarchy

### Context

You can open the *ABAP Type Hierarchy* view to display the inheritance tree of a selected ABAP class or interface.

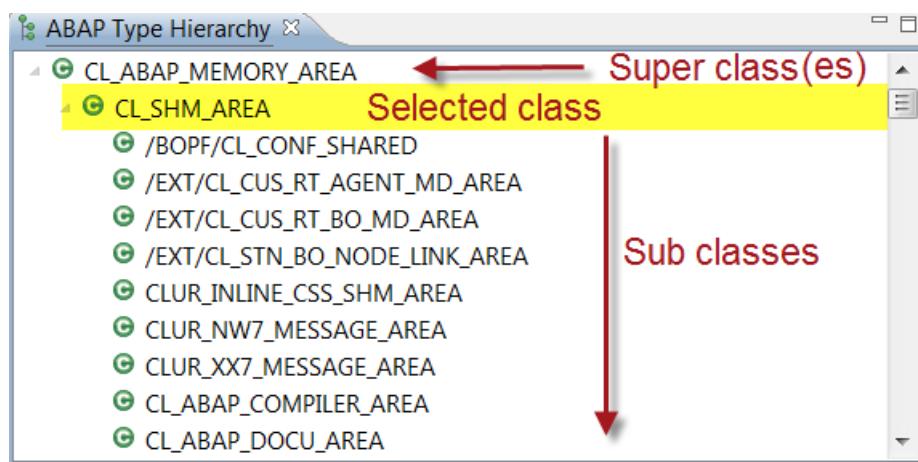
#### i Note

In contrast to the corresponding quick view (**Ctrl + T**), you can look at the ABAP type hierarchy for a selected class or interface independently from the current editor content. The hierarchy of a class or interface is still displayed in the *ABAP Type Hierarchy* view even if the source editor has been closed, in the meantime.

### Procedure

1. Select either the name of the class in the currently opened ABAP source editor or the corresponding node in the *Project Explorer*.
2. Open the context menu and then choose **Open ABAP Type Hierarchy** or press **T**.

The *ABAP Type Hierarchy* view appears that displays the class hierarchy including the superclasses and subclasses.



ABAP Type Hierarchy view that displays a class hierarchy

### Related Information

[ABAP Type Hierarchy View \[page 46\]](#)

[Using Quick Views \[page 337\]](#)

[Viewing the Outline \[page 332\]](#)

[Getting Orientation in the Source Code Using Breadcrumbs \[page 335\]](#)

## 5.1.16.11 Getting Orientation in the Source Code Using Breadcrumbs

### Prerequisites

Select the  breadcrumb icon from the tool bar to display the breadcrumb bar in the source code editor.

### Context

The breadcrumb functionality:

- Helps you to orientate and to navigate within the source code of a development object.
- Provides you with the same structural information as the *Outline* view.
- Provides you with information about the control structures (such as branch or loop statements) in which your cursor is currently positioned.

The screenshot shows the SAP ABAP source code editor with the following code:

```

10      RETURNING
11      result) TYPE
12      IMPORTING
13      .
14
15      .
16  ENDCLASS.
17
18 CLASS cl_car_constructor_generator
19   METHOD create.
20     DATA: r_result_1 TYPE REF
21     CREATE OBJECT r_result.
22     r_result_1 = r_result.
23     r_result_1->current_speed = i_current_speed.
24     test = 0.
25
26 METHOD drive
27   DATA: temp TYPE i.
28   IF i_speed > 240.
29     current_speed = 240.
30   ELSE.

```

The breadcrumb bar at the top shows the path: CL\_CAR\_CONSTRUCTOR\_GENERATOR > DRIVE. A red box highlights the word 'DRIVE' in the breadcrumb bar. A callout bubble points to this box with the text: "Breadcrumb bar with the occurrence of the selected element".

A dropdown menu is open next to the word 'DRIVE' in the breadcrumb bar. The menu contains the following items:

- CREATE
- CURRENT\_SPEED
- DRIVE

A red box highlights the word 'CREATE' in the dropdown menu. A callout bubble points to this box with the text: "Siblings of the element that is chosen from the dropdown list".

The word 'drive' in the code is highlighted with a red box. A callout bubble points to this box with the text: "Selected element".

Displaying siblings of an element in the breadcrumb bar

You can display the breadcrumb bar in the source code editor of the following source code-based development objects:

- ABAP classes
- ABAP interfaces
- ABAP programs
- ABAP includes
- ABAP function group main includes
- ABAP function group includes
- ABAP function modules

#### • Example

You are working on a long method where you cannot see the start statement. But you want to know the name of the method. The breadcrumb displays the names of the corresponding class and the method.

## Procedure

1. In the source code editor, position the cursor where you need further information.

The name of the method or development object is displayed in the breadcrumb bar..

2. Select the  icon.

A drop-down list is opened where the name of the relevant development objects and subcomponents are displayed.

3. Select the corresponding entry by double-clicking it.

## Results

The corresponding development object or subcomponent is opened and the cursor is positioned on the element's name (for example, method name or class name).

## Related Information

[Icons in the Outline View \[page 83\]](#)

[Using Quick Views \[page 337\]](#)

[Viewing the Outline \[page 332\]](#)

[Viewing the ABAP Type Hierarchy \[page 334\]](#)

## 5.1.16.12 Using Quick Views

### Context

So-called quick views are in-place views that are shown on top of the source code editor area and can be easily opened using keyboard shortcuts.

In the context of ABAP source code navigation, the following views are available:

- [Quick Outline \[page 338\]](#) (Ctrl + O shortcut) - displays the internal structure of a class, interface, or program.
- [Quick Type Hierarchy \[page 339\]](#) (Ctrl + T shortcut) - displays the type hierarchy of a class or interface.

## Related Information

[Viewing the Outline \[page 332\]](#)

[Viewing the ABAP Type Hierarchy \[page 334\]](#)

[Getting Orientation in the Source Code Using Breadcrumbs \[page 335\]](#)

## 5.1.16.12.1 Quick Outline

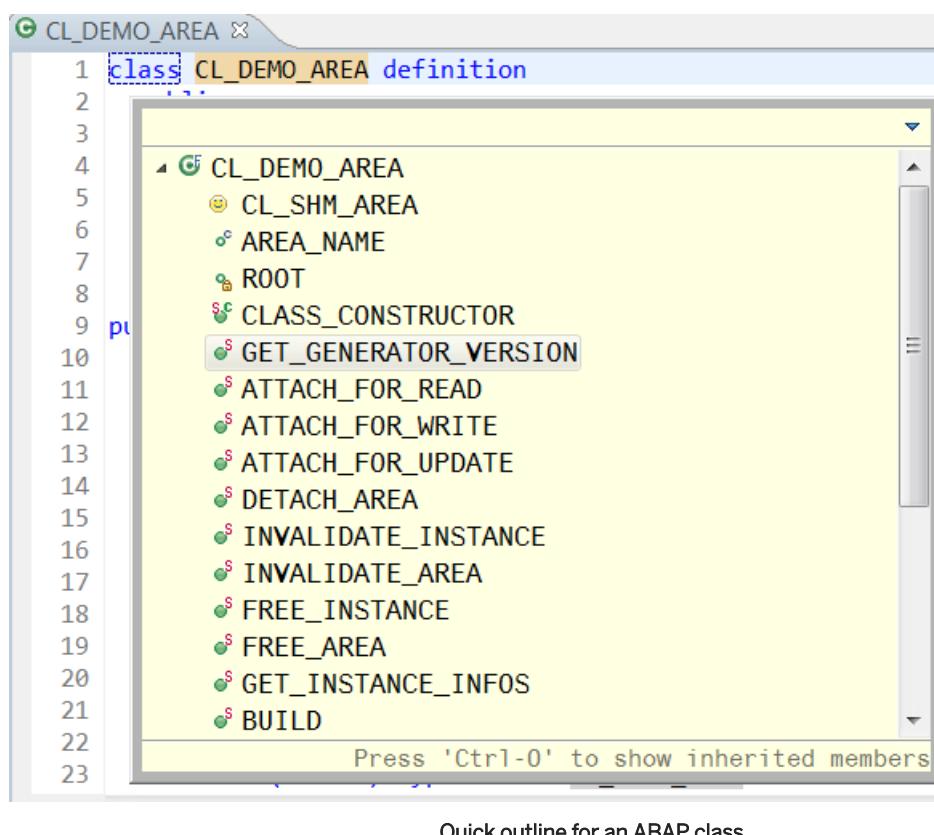
### Context

To use the quick outline in the currently opened ABAP source editor:

### Procedure

1. Press **Ctrl + O** or open the context menu of the editor and choose **Quick Outline**.

The in-place outline view appears for the currently opened development object.



2. Press **Ctrl + O** once again.

The *Outline* view shows also all subcomponents of all superclasses and all implemented interfaces (in blue font).

3. Start typing while the Outline view is shown to filter the list of elements.

```
1 class CL_DEMO_AREA definition
2   public
3   loc ← type in to filter elements
4   ▲ CL_DEMO_AREA
5     ⚋ LOCK_KIND_OUT_OF_MEMORY - CL_SHM_AREA
6     ⚋ LOCK_KIND_COMPLETION_ERROR - CL_SHM_AREA
7     ⚋ LOCK_KIND_DETACHED - CL_SHM_AREA
8     ⚋ LOCK_KIND_READ - CL_SHM_AREA
9     ⚋ LOCK_KIND_UPDATE - CL_SHM_AREA
10    ⚋ LOCK_KIND_WRITE - CL_SHM_AREA
11
12
13
14
15
16
```

Press 'Ctrl-0' to hide inherited members

Filtering elements in the quick outline

## Related Information

[Quick Type Hierarchy \[page 339\]](#)

[Using Quick Views \[page 337\]](#)

## 5.1.16.12.2 Quick Type Hierarchy

### Context

To use the quick view for type hierarchy:

### Procedure

1. Select the name of the class or interface in the currently opened ABAP source editor.
2. Press **Ctrl + T** or open the context menu of the editor and choose **Quick Type Hierarchy**.

The in-place type hierarchy view appears for the selected class or interface.

```
1 class CL_DEMO_AREA definition
2 public
3 inheritance from CL_ABAP_MEMORY_AREA
4 final
5 create
6 global
7
8
9 public
10
```

Types implementing or defining CL\_DEMO\_AREA->

- CL\_ABAP\_MEMORY\_AREA super class
- CL\_SHM\_AREA selected class
- CL\_DEMO\_AREA sub class

Quick view displaying the class hierarchy

## Related Information

[Quick Outline \[page 338\]](#)

[Using Quick Views \[page 337\]](#)

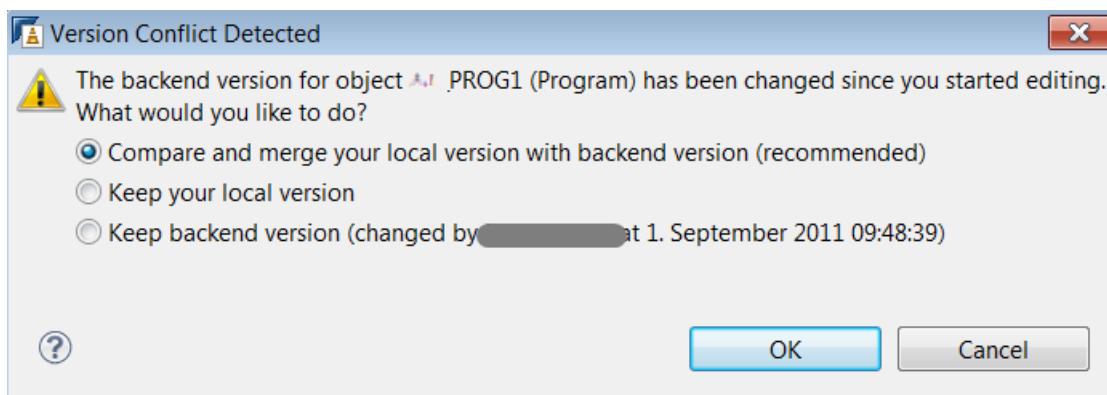
## 5.1.16.13 Managing Version Conflicts

### Context

In systems with an automatic logoff mechanism, in particular, it can happen that your lock gets lost. This can lead to the situation whereby other users may modify the same development object in the backend even if your changes have not yet been stored.

The merging function allows you to decide which version should be stored if another user has changed the same source in the backend in parallel. The merge dialog offers you options to:

- Compare and merge the versions available: This option opens the *Compare* editor where you can track the changes first.
- Keep your local version: This will overwrite the backend version of your source code with those from your ABAP project.
- Keep the backend version: This will overwrite your local source code with those from the ABAP Repository (backend).



Dialog that appears when version conflict has been detected

### 5.1.16.14 Switching between Inactive and Active Versions of a Source-based Object

You can toggle between the inactive and active version of any source-based objects such as classes, function modules, or ABAP programs. This functionality enables you to read the inactive version that is saved by another user and not available in your working area.

#### Prerequisites

You have made changes in the source code and saved them.

#### Procedure

1. In the source code editor, open the corresponding development object.
2. In the context menu, choose **Switch to active/inactive version** or select **Edit > Switch to active/inactive version** from the menu.
3. [Optional:] Repeat step 2 to switch back to the opposite version.

#### Results

In the source code editor, the inactive or active version of a development object is displayed.

##### Note

In the tab of the current source code editor, the status of a development object is indicated by a decorator.

Inactive development objects are highlighted with a .

## Related Information

[Comparing Source Code \[page 207\]](#)

[Status of a Development Object \[page 20\]](#)

## 5.1.17 Applying Quick Assists

### Context

ABAP Development Tools (ADT) provides a set of quick assists that help you to change ABAP source code in a semi-automated way.

## Related Information

[Refactoring ABAP Source Code \[page 342\]](#)

[Applying ABAP Quick Fixes \[page 402\]](#)

[Applying Other Quick Assists \[page 422\]](#)

[Quick Assists \[page 85\]](#)

[Quick Assist View \[page 91\]](#)

[ABAP Unit Test Classes \[page 97\]](#)

## 5.1.17.1 Refactoring ABAP Source Code

### Context

ABAP Development Tools provides a number of functions to refactor ABAP source code.

## Related Information

[Renaming Identifiers \[page 343\]](#)

[Extracting Methods \[page 347\]](#)

[Extracting Constants \[page 353\]](#)

[Extracting and Converting Variables \[page 357\]](#)

[Moving Members \[page 377\]](#)

[Exception Handling \[page 384\]](#)

[Quick Assists \[page 85\]](#)

[ABAP Refactorings \[page 86\]](#)

[Extracting Constants from Literals \[page 87\]](#)

[Extracting Variables from Literals \[page 89\]](#)

[Quick Assist View \[page 91\]](#)

## 5.1.17.1.1 Renaming Identifiers

### Context

ABAP Development Tools (ADT) provides an intelligent search and renaming function that can find and rename variables, methods, and form routine names, as well as other elements of an ABAP program. ABAP classes and ABAP interfaces can be renamed as well.

In respective of your current back end version, following renaming functions are provided:

- [Renaming Elements in a Single Source Unit \[page 344\]](#)
- [Renaming Elements in Multiple Source Units \[page 345\]](#)

### Other Supported Releases

In addition, you can rename global ABAP classes, interfaces, or their members (such as fields or methods of classes or interfaces) that are used globally in other programs. All external references in other classes, interfaces, or other program types are then automatically updated accordingly.

### Not Supported Object Types

#### ⚠ Caution

This kind of automatic update of external references is currently not supported in simple transformations, package interfaces, transactions, data elements, structures, table types, eCatt objects, and so on.

### Related Information

[Extracting Methods \[page 347\]](#)

[Extracting Constants \[page 353\]](#)

[Extracting and Converting Variables \[page 357\]](#)

[Moving Members \[page 377\]](#)

[Exception Handling \[page 384\]](#)

[Quick Assists \[page 85\]](#)

[Quick Assist View \[page 91\]](#)

## 5.1.17.1.1.1 Renaming Elements in a Single Source Unit

### Context

You can change all occurrences of an element (except ABAP keywords) in an ABAP program at the same time.

### Procedure

1. In the source code editor, position the cursor on the element to be renamed.
2. Start the Rename function function using the context menu and then *Rename x in source unit (Ctrl 2, R)*.

A frame appears around each occurrence of the element.

```
data: docu_langu type doku_langu,
      dokil_lines type standard table of dokil,
      work_output_string type string,
      dokil_line type dokil.

convert_isolangu_to_dokulangu(
  exporting
    iso_langu = iso_langu
  importing
    doku_langu = docu_langu ).

search_dokil_for_document(
  exporting
    dokil_object = search_string
    dokil_langu = docu_langu
  importing
    dokil_lines = dokil_lines
).


```

Highlighted position of the element that is edited and all further occurrences of the element

3. Type in your change. The ADT makes the change simultaneously in all occurrences of the element in the source code unit you are currently working in. This technique is only allowed for named elements that occur only in your current source code unit. For example, you cannot use this technique to rename elements in a method implementation if the named element is also used in your ABAP Unit tests.

### Results

You have renamed an element in your source code throughout the source code unit you are currently working in.

## Related Information

[Renaming Identifiers \[page 343\]](#)

[Renaming Elements in Multiple Source Units \[page 345\]](#)

## 5.1.17.1.1.2 Renaming Elements in Multiple Source Units

### Context

You can use this function to rename elements in all parts of the ABAP source code.

### Procedure

1. In the source code editor, position the cursor on the element to be renamed. ADT marks the element.

```
data: dokulangu type dokulangu,  
      dokil_lines type standard table of dokil,  
      work_output_string type string,  
      dokil_line type dokil.  
  
convert_isolangu_to_dokulangu(  
  exporting  
    iso_langu = isolangu  
  importing  
    dokulangu = dokulangu ).
```

Position the cursor on the variable to be changed

Highlighted positions where an element is renamed

2. Start the    from the context menu of the element to be renamed). Or use the *Quick Fix* menu: **Ctrl 1** and then *Rename...*
3. In the dialog box that follows, enter the new name for the element and choose *Next*.

#### i Note

You can also choose *Finish* to complete the renaming right away.

New name:

Input field to enter the new element name

4. The **Rename** function shows you the changes that will be made. Use the navigation buttons to step through the changes.

## Rename: Field

The following changes are necessary to perform the refactoring.



Changes to be performed

Rename  
 CL\_DOCU\_ADT\_CALL\_TOOLSDOCU\_SRV (Global Class)

ABAP Compare

Original Source	Refactored Source
350 data: doku_langu type doku_langu, 351 dokil_lines type standard table of dokil, 352 work_output_string type string, 353 dokil_line type dokil_line, 354 355 convert_isolangu_to_dokulangu( 356 exporting 357 iso_langu = iso_langu 358 importing 359 doku_langu = doku_langu, 360	350 data: docu_langu type docu_langu, 351 dokil_lines type standard table of dokil, 352 work_output_string type string, 353 dokil_line type dokil_line, 354 355 convert_isolangu_to_dokulangu( 356 exporting 357 iso_langu = iso_langu 358 importing 359 doku_langu = docu_langu, 360

Changes highlighted in a compare window

5. Choose *Finish* to complete the renaming.

## Results

The wizard renames the selected element and updates the references.

```
data: docu_langu type doku_langu,  
      dokil_lines type standard table of dokil,  
      work_output_string type string,  
      dokil_line type dokil_line.  
  
convert_isolangu_to_dokulangu(  
  exporting  
  iso_langu = iso_langu  
  importing  
  doku_langu = docu_langu ).
```

**Renaming is intelligent:  
Types and parameter  
names are untouched**

Display of the renamed elements and updated references

### i Note

You can undo the renaming by choosing *Undo Rename (Ctrl Z)* from the context menu.

## Related Information

[Renaming Identifiers \[page 343\]](#)

[Renaming Elements in a Single Source Unit \[page 344\]](#)

### 5.1.17.1.2 Extracting Methods

#### Context

In the source code editor, you have the following options for extracting methods:

Function	Description
<a href="#">Extracting Methods from Statements [page 348]</a>	Creating a new method in the current class. The selected code is moved into the body of the new method and replaced with a call to the new method. Further occurrences of similar code are not replaced.
<a href="#">Extracting Methods from Expressions [page 351]</a>	Creating a new method in the implementation of an ABAP class that returns the result of a selected expression. The selected expression is replaced with the call to the new method. Further occurrences of similar expressions are not replaced.

## Related Information

[Extracting Constants \[page 353\]](#)

[Extracting and Converting Variables \[page 357\]](#)

[Moving Members \[page 377\]](#)

[Exception Handling \[page 384\]](#)

[Creating Method Implementations from the Method Definition \[page 404\]](#)

[Creating Method Definitions from Implementation Parts \[page 406\]](#)

[Creating Implementation Parts for Unimplemented Methods \[page 409\]](#)

[Creating Methods from Method Calls \[page 414\]](#)

[Quick Assists \[page 85\]](#)

[Quick Assist View \[page 91\]](#)

## 5.1.17.1.2.1 Extracting Methods from Statements

### Prerequisites

You must be editing a global or local ABAP class in a class pool, a program, or another type of ABAP program. Method extraction is not supported in code outside of classes. Also, you can only extract a method within the start and end area of another method.

Your code should be free of syntax errors.

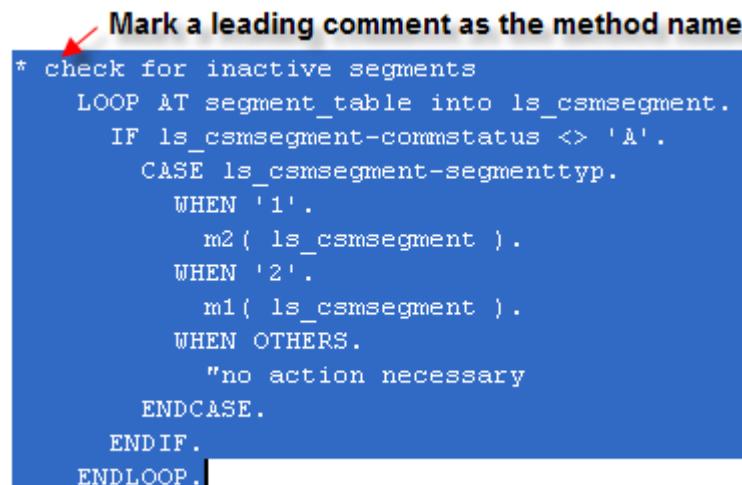
### Context

In the implementation of an ABAP class, you can create a new method. The selected code is moved into the body of the new method and replaced with a call to the new method. Further occurrences of similar code are not replaced.

#### i Note

The name of the new method is derived from the first code line that is selected. To this end, you can add a comment at the beginning of the source code that contains the keywords you want to reuse in the name of the new method.

**Mark a leading comment as the method name**



```
* check for inactive segments
  LOOP AT segment_table into ls_csmsegment.
    IF ls_csmsegment-commstatus <> 'A'.
      CASE ls_csmsegment-segmenttyp.
        WHEN '1'.
          m2( ls_csmsegment ).
        WHEN '2'.
          m1( ls_csmsegment ).
        WHEN OTHERS.
          "no action necessary
      ENDCASE.
    ENDIF.
  ENDLOOP.
```

Example of selected source code that is extracted into a new method

### Procedure

1. In the source code editor, select the code that you want to move to a new method.
2. In the context menu, choose  *Source Code*  *Extract Method ...*  or the shortcut (Alt Shift M) to start the dialog box.

Class/Interface: CL\_GUI\_ALV\_GRID

Pattern:  Method  Event

Name: `check_for_inactive_segments`

Access:  Static

Visibility:  Private  Protected  Public

Parameters:

Name	Direction	By ...	Typing	Data Type
c_segments_table	Changing	<input type="checkbox"/>	Type	ty_segment_table

**You can adjust the suggested method name and parameter attributes**

**Input field to edit the name of the method to be extracted**

You can change a suggested method name as well as the suggested name and attributes of a method parameter.

#### i Note

You cannot add a parameter to the proposed method signature, nor can you delete one.

3. Choose [Next](#) to open the transport dialog:
4. [Optional:] If the changed object is not assigned to a transport request, choose [Change Transport](#) to select an existing or create a new transport request.
  - Select [Error handling](#) if you want to ignore syntax errors. Consequently, no error messages will be displayed.
  - Select [Activation](#) if you want to activate the affected objects after refactoring. As a result, the changes will also be available in the related objects.

#### i Note

The method extractor presents the ABAP Compare editor to display the changes that will be made.

5. Choose [Next](#) to check the changes that will be made. The method extractor presents the [ABAP Compare](#) editor to display the changes that will be made.
6. Use the [ABAP Compare](#) navigation buttons to review the changes. These are all shown in the [Refactored Source](#) editor to the right.

```

ABAP Compare

Original Source
15  PROTECTED SECTION.
16  PRIVATE SECTION.
17    DATA a_private_counter TYPE i.
18    methods m4 importing ls_segment type csm
19      returning value(status) type csmcomsta
20
21  ENDMETHOD.
22
23
24
25 CLASS cl_test_intelligent_renaming IMPLEMENT
26
27  method m4.
28    status = 'A
29
30    DATA: 1
31      link_
32      object_name_length TYPE i,
33      type_1 TYPE
34
35  ENDMETHOD.

Refactored Source
15  PROTECT
16  PRIVATE
17    DATA a_private_counter TYPE i.
18    methods m4 importing ls_segment type csm
19      returning value(status) type csmcomsta
20
21
22
23
24
25 TYPES:
26   ty_segment_table TYPE STANDARD TAB
27
28 METHODS check_for_inactive_segments
29   CHANGING
30     c_segment_table TYPE ty_segment
31
32  ENDMETHOD.

CLASS cl_test_intelligent_renaming IMPLEMENT

```

Navigation between original and refactored changes

### i Note

- **Comment as default name:** Alternatively, you can start your selection with a comment. The comment is then automatically suggested as the name of the new method. But you can also change the name or write in your own name from scratch.
- **Method signature:** The code extractor is very intelligent and usually does a good job of parsing the code to extract and suggest useful method parameters. However, it can – under rare circumstances – be misled. In this case, you can adjust the extracted code after you are finished with the method extraction. You can also, within reason, change the suggested direction (exporting/changing...) of a parameter.
- **Method visibility:** By default, the new method is a private method in the local or global class you are currently working in. You cannot have the method created in another class.

7. Choose *Finish* to complete the method extraction.

## Results

The method extractor inserts a call to the new method. Using *Undo Extract Method* (Ctrl + Z) in the editor context menu, you can cancel the method extraction and revert to the old code.

In addition to moving the extracted code and replacing it with a method call, the method extractor has done the following:

- Implemented the signature – with your changes – that it suggested for the new method
- Declared any local variables that are needed in the new method
- Exposed class-based or classic exceptions in the method signature
- Suggested a simple, recommended naming convention for parameters and variables in the new method

You may wish to delete unused variables in the class from which you extracted the method. The method extractor does not change the data declarations in your code, so the method extraction may leave an orphaned data declaration behind.

## Related Information

[Extracting Methods from Expressions \[page 351\]](#)

[Extracting Methods \[page 347\]](#)

[Quick Assists \[page 85\]](#)

[Quick Assist View \[page 91\]](#)

### 5.1.17.1.2.2 Extracting Methods from Expressions

#### Context

In the implementation part of an ABAP class, you can create a new method that returns the result of a selected expression. The tool declares this returning parameter with type Integer and the selected expression is replaced with a call to the new method.

##### i Note

The editor only replaces the selected expression. If the same expression is used several times in the same repository object, only the selected occurrence is replaced. This means you have to extract all remaining occurrences individually.

#### Example

Here, the expression `length * width` is extracted into a method:

```
CLASS CL_METHOD_CHECK_AREA DEFINITION PUBLIC.
  " This class describes the 'length * width' extraction of an expression to a
  new method
  PUBLIC SECTION.
  DATA:
    length TYPE i,
    width TYPE i.
  METHODS:
    check_area
    RETURNING
      VALUE(r_result) TYPE abap_bool.
ENDCLASS.

CLASS CL_METHOD_CHECK_AREA IMPLEMENTATION.
  METHOD check_area.
    IF length * width > 100.
      r_result = abap_false.
    ELSE.
      r_result = abap_true.
    ENDIF.
  ENDMETHOD.
ENDCLASS.
```

## Procedure

1. In the implementation part, select the entire expression that you want to add to a new method.
2. In the context menu, choose **Quick Fix** or use the shortcut (Ctrl + 1).
3. In the Quick Fix dialog box, double-click **Extract method from expression**.

## Results

The extracted method is created with the name `new_method`.

In the private section of the definition part, the name of the extracted method (`new_method`) is added and the appropriate parameters are declared.

In the implementation part, the selected expression is added in a new method. In the former call, the expression is replaced with the name of the extracted method (`new_method`).

Example

ABAP class after extraction:

```
CLASS CL_METHOD_CHECK_AREA DEFINITION PUBLIC.
  "This class describes the 'length * width' extraction of an expression for a
  new method.
  PUBLIC SECTION.
  DATA:
    length TYPE i,
    width TYPE i.
  METHODS:
    check_area
    RETURNING
      VALUE(r_result) TYPE abap_bool.
PRIVATE SECTION.
METHODS: new_method
RETURNING
  VALUE(r_result) TYPE i.
ENDCLASS.

CLASS CL_METHOD_CHECK_AREA IMPLEMENTATION.
METHOD check_area.
  IF new_method( ) > 100.
    r_result = abap_false.
  ELSE.
    r_result = abap_true.
  ENDIF.
ENDMETHOD.
METHOD new_method.
  r_result = length * width.
ENDMETHOD.
ENDCLASS.
```

## Related Information

[Extracting Methods from Statements \[page 348\]](#)

[Extracting Methods \[page 347\]](#)

[Quick Assists \[page 85\]](#)

[Quick Assist View \[page 91\]](#)

### 5.1.17.1.3 Extracting Constants

#### Context

In the source code editor, you have the following options for extracting constants:

Function	Description
<a href="#">Extracting Constants from Literals [page 353]</a>	Creating a constant with the value of the literal and replacing all occurrences.
<a href="#">Reusing Existing Constants [page 355]</a>	Replacing a literal with an existing constant. All other occurrences of the same literals remain unchanged.
<a href="#">Converting Locals to Class Members [page 360]</a>	Converting a local constant, local variable, or local type to a class member such as a member constant, attribute, or member type.

Example

- Characters are indicated with the ' quotation mark: '240'
- Strings are indicated with the ` quotation mark: `240`

#### Related Information

[Extracting Methods \[page 347\]](#)

[Extracting and Converting Variables \[page 357\]](#)

[Moving Members \[page 377\]](#)

[Exception Handling \[page 384\]](#)

[Extracting Constants from Literals \[page 87\]](#)

[Extracting Variables from Literals \[page 89\]](#)

[Quick Assists \[page 85\]](#)

[Quick Assist View \[page 91\]](#)

#### 5.1.17.1.3.1 Extracting Constants from Literals

You can extract local and member constants, local variables, attributes, as well as importing parameters from the source code in order to substitute a literal in all or selected methods of an ABAP class. Afterwards, the tool adds the corresponding identifier in the private or public section of the current class. So, the new identifier is available in the entire ABAP class or in the corresponding method.

## Context

### Example

Before Execution	After Execution
<pre>CLASS cl_car_extract_constant DEFINITION PUBLIC CREATE PUBLIC . PUBLIC SECTION .   DATA current_speed TYPE i.   METHODS: drive IMPORTING i_speed TYPE i.   PRIVATE SECTION . ENDCLASS.  CLASS cl_car_extract_constant METHOD drive.   IF i_speed &gt; 240.     current_speed = 240.   ENDIF. ENDMETHOD. ENDCLASS.</pre>	<pre>CLASS cl_car_extract_constant DEFINITION PUBLIC CREATE PUBLIC . PUBLIC SECTION .   DATA current_speed TYPE i.   METHODS: drive IMPORTING i_speed TYPE i.   PRIVATE SECTION . <b>CONSTANTS _240 TYPE i VALUE 240.</b> ENDCLASS.  CLASS cl_car_extract_constant METHOD drive.   IF i_speed &gt; <b>_240</b>.     current_speed = <b>_240</b>.   ENDIF. ENDMETHOD. ENDCLASS.</pre>

To copy the source code example, click here [Code Example](#)

[Before Execution \[page 355\]](#)

The literal '240' is to be extracted into a member constant.

In the private section of the ABAP class, the new constant (\_240) is defined as a string type with the value of the literal ('240'). Also, all occurrences of the literal are replaced with the new constant.

## Procedure

1. In the implementation part, select the literal that is to be extracted as a member constant.
2. Get the quick assist proposals by choosing **Quick Fix** from the context menu (shortcut **Ctrl + 1**) or by using the **Quick Assist** view.
3. In the **Quick Fix** dialog box, double-click **Extract local constant** or **Extract member constant**.

## Related Information

[Reusing Existing Constants \[page 355\]](#)

[Converting Locals to Class Members \[page 360\]](#)

[Extracting Constants from Literals \[page 87\]](#)

[Quick Assists \[page 85\]](#)

[Quick Assist View \[page 91\]](#)

### 5.1.17.1.3.1.1 Code Example Before Execution

```
CLASS cl_car_extract_constant DEFINITION PUBLIC CREATE PUBLIC .
  PUBLIC SECTION .
    DATA current_speed TYPE i.
    METHODS: drive IMPORTING i_speed TYPE i.
  PRIVATE SECTION .
ENDCLASS.

CLASS cl_car_extract_constant
  METHOD drive.
    IF i_speed > 240.
      current_speed = 240.
    ENDIF.
  ENDMETHOD.
ENDCLASS.
```

### 5.1.17.1.3.2 Reusing Existing Constants

You can reuse a member or local constant that has already been generated. Note that all occurrences of the same literal remain unchanged.

#### Prerequisites

An existing constant is already defined for the same literal in a certain ABAP class.

#### Context

Example

## Before Execution

```
CLASS cl_car DEFINITION.
  PUBLIC SECTION .
    METHODS: drive
      IMPORTING
        i_speed TYPE i.
    PRIVATE SECTION .
      CONSTANTS max_speed TYPE i VALUE
220.
      DATA: current_speed TYPE i.
    ENDCLASS.

CLASS cl_car IMPLEMENTATION.
  METHOD drive.
    IF i_speed > 220.
      current_speed = 220.
    ELSE.
      current_speed = i_speed.
    ENDIF.
  ENDMETHOD.
ENDCLASS.
```

## After Execution

```
CLASS cl_car DEFINITION.
  PUBLIC SECTION .
    METHODS: drive
      IMPORTING
        i_speed TYPE i.
    PRIVATE SECTION .
      CONSTANTS max_speed TYPE i VALUE
220.
      DATA: current_speed TYPE i.
    ENDCLASS.

CLASS cl_car IMPLEMENTATION.
  METHOD drive.
    IF i_speed > max_speed.
      current_speed = max_speed.
    ELSE.
      current_speed = i_speed.
    ENDIF.
  ENDMETHOD.
ENDCLASS.
```

To copy the source code example, click here[Code Example](#)

[Before Execution \[page 356\]](#)

The literal '200' is to be replaced with an existing constant

In the source code of the ABAP class, all occurrences of the literal are replaced with the name of the existing constant.

## Procedure

1. In the source code editor, select the literal that is to be replaced by an existing constant.
2. Get the quick assist proposals by choosing **Quick Fix** from the context menu (shortcut **Ctrl + 1**) or by using the **Quick Assist** view.
3. In the **Quick Fix** dialog box, double-click **Use local constant** or **Use member constant**.

## Related Information

[Extracting Constants from Literals \[page 353\]](#)

[Converting Locals to Class Members \[page 360\]](#)

[Extracting Constants from Literals \[page 87\]](#)

[Quick Assists \[page 85\]](#)

[Quick Assist View \[page 91\]](#)

## 5.1.17.1.3.2.1 Code Example Before Execution

```
CLASS cl_car DEFINITION.
```

```

PUBLIC SECTION .
METHODS: drive
  IMPORTING
    i_speed TYPE i.
PRIVATE SECTION .
  CONSTANTS max_speed TYPE i VALUE 220.
  DATA: current_speed TYPE i.
ENDCLASS.

CLASS cl_car IMPLEMENTATION.
METHOD drive.
  IF i_speed > 220.
    current_speed = 220.
  ELSE.
    current_speed = i_speed.
  ENDIF.
ENDMETHOD.
ENDCLASS.

```

## 5.1.17.1.4 Extracting and Converting Variables

### Context

In the source code editor, you have the following options for extracting variables:

Function	Description
<a href="#">Extracting Local Variables from Expressions [page 358]</a>	Assigning the selected expression to a new local variable. The selected expression is replaced with the new local variable.
<a href="#">Assigning a Statement to a New Variable [page 363]</a>	Assigning the value of the selected statement to a new local variable or attribute.
<a href="#">Converting Locals to Class Members [page 360]</a>	Converting a local constant, local variable, or local type to a class member such as a member constant, attribute, or member type of the current class.
<a href="#">Converting Local Variables to Parameters [page 362]</a>	In a certain method, converting an existing local variable to a new parameter.
<a href="#">Declaring Variables from Usage [page 368]</a>	Creating a declaration for an attribute within a method.
<a href="#">Declaring Inline Variables Explicitly [page 365]</a>	In the method signature, converting an existing inline declaration of a local variable to an explicit declaration.
<a href="#">Using Similar Variables [page 369]</a>	You have following options for using similar variables: <ul style="list-style-type: none"> <li>• <a href="#">Correcting Misspelled Variables [page 370]</a></li> <li>• <a href="#">Correcting an Interface Attribute Access [page 371]</a></li> </ul>
<a href="#">Deleting Unused Variables [page 375]</a>	Supported deletion of unused data declarations and variables.

### Related Information

[Extracting Methods \[page 347\]](#)

[Extracting Constants \[page 353\]](#)

[Moving Members \[page 377\]](#)

[Quick Assists \[page 85\]](#)

[Exception Handling \[page 384\]](#)

[Extracting Variables from Literals \[page 89\]](#)

[Quick Assist View \[page 91\]](#)

## 5.1.17.1.4.1 Extracting Local Variables from Expressions

In a method of an ABAP class, you can create a new variable on the basis of an expression. This means, the tool adds a new local variable and assigns the expressions. Note that other occurrences of the same expression are not replaced with the new variable.

### Context

Example

Before Execution

```
01 CLASS cl_car DEFINITION.
02   PUBLIC SECTION.
03     METHODS: drive
04       IMPORTING i_speed TYPE i.
05   PRIVATE SECTION.
06     DATA: speed TYPE i.
07 ENDCLASS.
08
09 CLASS cl_car IMPLEMENTATION.
10   METHOD drive.
11     speed = speed + i_speed * '1.6'.
12   ENDMETHOD.
13 ENDCLASS.
```

After Execution

```
01 CLASS cl_car DEFINITION.
02   PUBLIC SECTION.
03     METHODS: drive
04       IMPORTING i_speed TYPE i.
05   PRIVATE SECTION.
06     DATA: speed TYPE i.
07 ENDCLASS.
08
09 CLASS cl_car IMPLEMENTATION.
10   METHOD drive.
11     DATA: kmh TYPE i.
12     kmh = i_speed * '1.6'.
13     speed = speed + i_speed * '1.6'.
14     speed = speed + kmh.
15   ENDMETHOD.
16 ENDCLASS.
```

To copy the source code example, click here [Code Example](#)

[Before Execution \[page 359\]](#)

The expression `i_speed * '1.6'` is to be extracted into a new local variable.

In the implementation part of the method, the new local variable (`kmh`) is defined. The expression is assigned to the new local variable. In the `speed` statement, the expression is replaced with the new variable.

## Procedure

1. In the implementation part, select the entire expression that you want to replace with a new variable.
2. Get the quick assist proposals by choosing **Quick Fix** from the context menu (shortcut **Ctrl** + **1**) or by using the *Quick Assist* view.
3. Apply *Extract local variable*.

## Related Information

[Extracting and Converting Variables \[page 357\]](#)  
[Converting Locals to Class Members \[page 360\]](#)  
[Converting Local Variables to Parameters \[page 362\]](#)  
[Assigning a Statement to a New Variable \[page 363\]](#)  
[Declaring Inline Variables Explicitly \[page 365\]](#)  
[Declaring Variables from Usage \[page 368\]](#)  
[Using Similar Variables \[page 369\]](#)  
[Deleting Unused Variables \[page 375\]](#)  
[Extracting Variables from Literals \[page 89\]](#)  
[Quick Assists \[page 85\]](#)  
[Quick Assist View \[page 91\]](#)

## 5.1.17.1.4.1.1 Code Example Before Execution

```
CLASS cl_car DEFINITION.  
  PUBLIC SECTION.  
    METHODS: drive  
      IMPORTING i_speed TYPE i.  
  PRIVATE SECTION.  
    DATA: speed TYPE i.  
ENDCLASS.  
CLASS cl_car IMPLEMENTATION.  
  METHOD drive.  
    speed = speed + i_speed * '1.6'.  
  ENDMETHOD.  
ENDCLASS.
```

## 5.1.17.1.4.2 Converting Locals to Class Members

You can convert definitions of a local variable, local constant, or local type and paste them into the private section of the current class. Therefore, you can make certain local elements of individual methods available for all methods within an ABAP class.

### Context

#### i Note

In the following table, you can see the local elements that can be converted to the corresponding member:

Local Elements	Members
Local variables	Attributes
Local constants	Member constants
Local types	Member types
Field symbols	Cannot be converted to members

### Example

Before Execution	After Execution
<pre>01 CLASS cl_car DEFINITION. 02 PUBLIC SECTION. 03 METHODS: drive. 04 PRIVATE SECTION. 05 ENDCCLASS. 06 07 CLASS cl_car IMPLEMENTATION. 08 METHOD drive. 09 DATA: speed TYPE i. 10 ... 11 ENDMETHOD. 12 ENDCCLASS.</pre>	<pre>01 CLASS cl_car DEFINITION. 02 PUBLIC SECTION. 03 METHODS: drive. 04 PRIVATE SECTION. 05 DATA: speed TYPE i. 06 ENDCCLASS. 07 08 CLASS cl_car IMPLEMENTATION. 09 METHOD drive. 10 DATA: speed TYPE i. 11 ... 12 ENDMETHOD. 13 ENDCCLASS.</pre>

To copy the source code example, click here [Code Example](#)

[Before Execution \[page 361\]](#)

The speed variable is declared local variable inside the implementation of the `drive` method.

The local declaration has been removed and replaced by an attribute declaration.

The attribute has been added to the private section.

## Procedure

1. In the implementation part, position the cursor on the corresponding local declaration (variable, constant, or type).
2. Get the quick assist proposals by choosing *Quick Fix* from the context menu (shortcut  $\text{Ctrl} + \text{1}$ ) or by using the *Quick Assist* view.
3. Apply *Convert 'local variable' to attribute*, *Convert 'local constant' to constant*, or *Convert 'local type' to class type*.

## Related Information

- [Extracting and Converting Variables \[page 357\]](#)  
[Extracting Local Variables from Expressions \[page 358\]](#)  
[Converting Local Variables to Parameters \[page 362\]](#)  
[Assigning a Statement to a New Variable \[page 363\]](#)  
[Declaring Inline Variables Explicitly \[page 365\]](#)  
[Declaring Variables from Usage \[page 368\]](#)  
[Using Similar Variables \[page 369\]](#)  
[Deleting Unused Variables \[page 375\]](#)  
[Extracting Variables from Literals \[page 89\]](#)  
[Quick Assists \[page 85\]](#)  
[Quick Assist View \[page 91\]](#)

### 5.1.17.1.4.2.1 Code Example Before Execution

```
CLASS cl_car DEFINITION.
  PUBLIC SECTION.
    METHODS: drive.
  PRIVATE SECTION.
ENDCLASS.

CLASS cl_car IMPLEMENTATION.
  METHOD drive.
    DATA: speed TYPE i.
  ...
ENDMETHOD.
ENDCLASS.
```

## 5.1.17.1.4.3 Converting Local Variables to Parameters

You can convert a local variable into a parameter create one of the following parameter types: IMPORTING, RETURNING, CHANGING, or EXPORTING.

### Context

Example

Before Execution	After Execution
<pre>01 CLASS cl_car DEFINITION. 02 PUBLIC SECTION. 03 METHODS: drive. 04 PRIVATE SECTION. 05 ENDCCLASS. 06 07 CLASS cl_car IMPLEMENTATION. 08 METHOD drive. 09   DATA: speed TYPE i. 10   ... 11 ENDMETHOD. 12 ENDCCLASS.</pre>	<pre>01 CLASS cl_car DEFINITION. 02 PUBLIC SECTION. 03 METHODS: drive. 04 IMPORTING 05   speed TYPE i. 06 PRIVATE SECTION. 07 ENDCCLASS. 08 09 CLASS cl_car IMPLEMENTATION. 10 METHOD drive. 11 DATA: speed TYPE i. 12 ... 13 ENDMETHOD. 14 ENDCCLASS.</pre>

To copy the source code example, click here[Code Example](#)

[Before Execution \[page 363\]](#)

The local variable speed is declared inside the drive method.

The local declaration has been removed and the speed parameter has been added to the signature of the drive method.

### Procedure

1. In the source code editor, navigate to the method and position the cursor on the definition of the local variable.
2. Get the quick assist proposals by choosing **Quick Fix** from the context menu (shortcut **Ctrl + 1**) or by using the *Quick Assist* view.
3. Apply **Convert 'local variable' to [importing / changing / exporting / returning] parameter**.

### Related Information

[Extracting and Converting Variables \[page 357\]](#)

[Extracting Local Variables from Expressions \[page 358\]](#)

[Converting Locals to Class Members \[page 360\]](#)

[Assigning a Statement to a New Variable \[page 363\]](#)

[Declaring Inline Variables Explicitly \[page 365\]](#)

[Declaring Variables from Usage \[page 368\]](#)

[Using Similar Variables \[page 369\]](#)

[Deleting Unused Variables \[page 375\]](#)

[Extracting Variables from Literals \[page 89\]](#)

[Quick Assists \[page 85\]](#)

[Quick Assist View \[page 91\]](#)

### 5.1.17.1.4.3.1 Code Example Before Execution

```
CLASS cl_car DEFINITION.
  PUBLIC SECTION.
    METHODS: drive.
  PRIVATE SECTION.
ENDCLASS.

CLASS cl_car IMPLEMENTATION.
  METHOD drive.
    DATA: speed TYPE i.
    ...
  ENDMETHOD.
ENDCLASS.
```

### 5.1.17.1.4.4 Assigning a Statement to a New Variable

You can create an attribute or local variable in order to assign the result of a functional method call. A functional method provides a returning parameter.

#### Prerequisites

This quick assist is provided if the statement has a result.

#### Context

##### Example

## Before Execution

```
01 CLASS cl_car DEFINITION.
02 PUBLIC SECTION.
03 CLASS-METHODS: create
04 RETURNING*
05 r_result TYPE REF TO cl_car.
06 ENDCCLASS.
07
08 CLASS cl_car IMPLEMENTATION.
09 METHOD create.
10 CREATE OBJECT r_result.
11 ENDMETHOD.
12 ENDCCLASS.
13
14 CLASS ltc_car DEFINITION.
15 PRIVATE SECTION.
16 METHODS: test FOR TESTING.
17 ENDCCLASS.
18
19 CLASS ltc_car IMPLEMENTATION.
20 METHOD test.
21 cl_car=>create( ).
```

## After Execution

```
01 CLASS cl_car DEFINITION.
02 PUBLIC SECTION.
03 CLASS-METHODS: create
04 RETURNING*
05 r_result TYPE REF TO cl_car.
06 ENDCCLASS.
07
08 CLASS cl_car IMPLEMENTATION.
09 METHOD create.
10 CREATE OBJECT r_result.
11 ENDMETHOD.
12 ENDCCLASS.
13
14 CLASS ltc_car DEFINITION.
15 PRIVATE SECTION.
16 METHODS: test FOR TESTING.
17 ENDCCLASS.
18
19 CLASS ltc_car IMPLEMENTATION.
20 METHOD test.
21 DATA: car TYPE REF TO cl_car.
22 car = cl_car=>create( ).
```

To copy the source code example, click here[Code Example](#)

[Before Execution \[page 365\]](#)

The new variable `car` is to be added to replace the call of the `create` method

In the implementation part of the `test` method, the new `car` variable is declared.

At the cursor position, the new variable is added and assigned to the existing statement. Now, you can, for example, change its name

## Procedure

1. In the implementation part, position the cursor in front of the functional method call for which you want to create a new variable or attribute.
2. Get the quick assist proposals by choosing **Quick Fix** from the context menu (shortcut **Ctrl** + **1**) or by using the **Quick Assist** view.
3. Apply **Assign statement to new local variable** or **Assign statement to new attribute**.

## Related Information

[Extracting and Converting Variables \[page 357\]](#)

[Extracting Local Variables from Expressions \[page 358\]](#)

[Converting Locals to Class Members \[page 360\]](#)

[Converting Local Variables to Parameters \[page 362\]](#)

[Declaring Inline Variables Explicitly \[page 365\]](#)

[Declaring Variables from Usage \[page 368\]](#)

[Using Similar Variables \[page 369\]](#)

[Deleting Unused Variables \[page 375\]](#)

[Extracting Variables from Literals \[page 89\]](#)

[Quick Assists \[page 85\]](#)

[Quick Assist View \[page 91\]](#)

### 5.1.17.1.4.4.1 Code Example Before Execution

```
CLASS cl_car DEFINITION.
  PUBLIC SECTION.
    CLASS-METHODS: create
      RETURNING*
        r_result TYPE REF TO cl_car.
  ENDCLASS.

  CLASS cl_car IMPLEMENTATION.
    METHOD create.
      CREATE OBJECT r_result.
    ENDMETHOD.
  ENDCLASS.

  CLASS ltc_car DEFINITION.
    PRIVATE SECTION.
      METHODS: test FOR TESTING.
  ENDCLASS.

  CLASS ltc_car IMPLEMENTATION.
    METHOD test.
      cl_car=>create( ).
    ENDMETHOD.
  ENDCLASS.
```

### 5.1.17.1.4.5 Declaring Inline Variables Explicitly

In a method implementation, you can convert inline declarations of one or several local variables into explicit declarations.

#### Prerequisites

The local variable is declared inline using the statement `DATA (temp)`.

## Context

Example

Before Execution	After Execution
<pre>01 CLASS cl_car DEFINITION. 02 PUBLIC SECTION. 03 METHODS: drive 04 IMPORTING 05 i_speed TYPE i. 06 PRIVATE SECTION. 07 DATA: speed TYPE i. 08 ENDCCLASS. 09 10 CLASS cl_car IMPLEMENTATION. 11 METHOD drive. 12 DATA(temp) = i_speed * '1.6'. 13 DATA(engine) = 'on'. 14 speed = speed + temp 15 ENDMETHOD. 16 ENDCCLASS.</pre>	<pre>01 CLASS cl_car DEFINITION. 02 PUBLIC SECTION. 03 METHODS: drive 04 IMPORTING 05 i_speed TYPE i. 06 PRIVATE SECTION. 07 DATA: speed TYPE i. 08 ENDCCLASS. 09 10 CLASS cl_car IMPLEMENTATION. 11 METHOD drive. 12 DATA: temp TYPE i. 13 engine TYPE string. 14 DATA(temp) = i_speed * '1.6'. 15 DATA(engine) = 'on'. 16 speed = speed + temp. 17 ENDMETHOD. 18 ENDCCLASS.</pre>

To copy the source code example, click here [Code Example](#)

[Before Execution \[page 367\]](#)

The inline declaration DATA (temp) is removed and an explicit declaration is inserted in the method implementation.

In the corresponding method, the new local variables (temp and engine) are generated and replace the undeclared inline variable. The type (i and string) are derived from the usage of the variable.

## Procedure

1. In the implementation part, position the cursor on the inline declared variable.
2. Get the quick assist proposals by choosing **Quick Fix** from the context menu (shortcut **Ctrl** + **1**) or by using the *Quick Assist* view.
3. Apply **Declare local variable 'variable' explicitly**.

## Related Information

[Extracting and Converting Variables \[page 357\]](#)

[Extracting Local Variables from Expressions \[page 358\]](#)

[Converting Locals to Class Members \[page 360\]](#)

[Converting Local Variables to Parameters \[page 362\]](#)

[Assigning a Statement to a New Variable \[page 363\]](#)

[Declaring Variables from Usage \[page 368\]](#)  
[Using Similar Variables \[page 369\]](#)  
[Deleting Unused Variables \[page 375\]](#)  
[Extracting Variables from Literals \[page 89\]](#)  
[Quick Assists \[page 85\]](#)  
[Quick Assist View \[page 91\]](#)

## 5.1.17.1.4.5.1 Code Example Before Execution

```
CLASS cl_car DEFINITION.  
  PUBLIC SECTION.  
    METHODS: drive  
      IMPORTING  
        i_speed TYPE i.  
  PRIVATE SECTION.  
    DATA: speed TYPE i.  
  ENDCLASS.  
  
CLASS cl_car IMPLEMENTATION.  
  METHOD drive.  
    DATA(temp) = i_speed * '1.6'.  
    DATA(engine) = 'on'.  
    speed = speed + temp  
  ENDMETHOD.  
ENDCLASS.
```

## 5.1.17.1.4.6 Declaring Variables from Usage

You can create an attribute, local variable, or a parameter (importing, returning, changing, or exporting) on the basis of a usage in a method implementation.

### Context

#### Example

##### Before Execution

```
01 CLASS cl_car DEFINITION.
02   PUBLIC SECTION.
03     METHODS drive.
04   PRIVATE SECTION.
05 ENDCLASS.
06
07 CLASS cl_car IMPLEMENTATION.
08   METHOD drive.
09     speed = 50.
10     engine_on = abap_true.
11   ...
12 ENDMETHOD.
13 ENDCCLASS.
```

##### After Execution

```
01 CLASS cl_car DEFINITION.
02   PUBLIC SECTION.
03     METHODS drive.
04   PRIVATE SECTION.
05 ENDCLASS.
06
07 CLASS cl_car IMPLEMENTATION.
08   METHOD drive.
09     DATA: speed TYPE i,
10       engine_on TYPE abap_bool.
11     speed = 50.
12     engine_on = abap_true.
13   ...
14 ENDMETHOD.
15 ENDCCLASS.
```

To copy the source code example, click here [Code Example Before Execution \[page 369\]](#)

The variable speed is used but not yet declared.

In the signature of the drive method, the attributes (speed and engine\_on) are added and declared.

### Procedure

1. In the implementation part, select the usage that you want to replace, for example, with a local variable.
2. Get the quick assist proposals by choosing *Quick Fix* from the context menu (shortcut **Ctrl** + **1**) or by using the *Quick Assist* view.
3. Apply *Declare [importing / changing / exporting / returning] parameter from usage* or *Declare local variable from usage*.

### Related Information

[Extracting and Converting Variables \[page 357\]](#)  
[Extracting Local Variables from Expressions \[page 358\]](#)  
[Converting Locals to Class Members \[page 360\]](#)  
[Converting Local Variables to Parameters \[page 362\]](#)  
[Assigning a Statement to a New Variable \[page 363\]](#)  
[Declaring Inline Variables Explicitly \[page 365\]](#)  
[Using Similar Variables \[page 369\]](#)  
[Deleting Unused Variables \[page 375\]](#)  
[Extracting Variables from Literals \[page 89\]](#)  
[Quick Assists \[page 85\]](#)  
[Quick Assist View \[page 91\]](#)

### 5.1.17.1.4.6.1 Code Example Before Execution

```
CLASS cl_car DEFINITION.
  PUBLIC SECTION.
    METHODS drive.
  PRIVATE SECTION.
ENDCLASS.

CLASS cl_car IMPLEMENTATION.
  METHOD drive.
    speed = 50.
    engine_on = abap_true.
    ...
  ENDMETHOD.
ENDCLASS.
```

### 5.1.17.1.4.7 Using Similar Variables

You can reuse the name of a variable or interface attribute that is already defined by applying the following quick fixes:

- [Correcting Misspelled Variables \[page 370\]](#) to replace an undeclared variable with the name of a variable already defined.
- [Correcting an Interface Attribute Access \[page 371\]](#) to replace an undeclared variable with a similar constant or attribute of an implemented ABAP interface.

### Related Information

[Extracting and Converting Variables \[page 357\]](#)  
[Extracting Local Variables from Expressions \[page 358\]](#)  
[Converting Locals to Class Members \[page 360\]](#)  
[Converting Local Variables to Parameters \[page 362\]](#)

[Assigning a Statement to a New Variable \[page 363\]](#)

[Declaring Inline Variables Explicitly \[page 365\]](#)

[Declaring Variables from Usage \[page 368\]](#)

[Deleting Unused Variables \[page 375\]](#)

[Extracting Variables from Literals \[page 89\]](#)

[Quick Assists \[page 85\]](#)

[Quick Assist View \[page 91\]](#)

## 5.1.17.1.4.7.1 Correcting Misspelled Variables

You can correct misspelled variables using a quick fix if another variable with a similar name exists.

### Prerequisites

The source code contains an undeclared variable, but a variable with a similar name already exists.

### Context

Example

Before Execution	After Execution
<pre>01 CLASS cl_car DEFINITION. 02   PUBLIC SECTION. 03     METHODS drive. 04   PRIVATE SECTION. 05     DATA speed TYPE i. 06 ENDCLASS. 07 08 CLASS cl_car IMPLEMENTATION. 09   METHOD drive. 10     sped = 50. 11     ... 12 ENDMETHOD. 13 ENDCCLASS.</pre>	<pre>01 CLASS cl_car DEFINITION. 02   PUBLIC SECTION. 03     METHODS drive. 04   PRIVATE SECTION. 05     DATA speed TYPE i. 06 ENDCLASS. 07 08 CLASS cl_car IMPLEMENTATION. 09   METHOD drive. 10     spedspeed = 50. 11     ... 12 ENDMETHOD. 13 ENDCCLASS.</pre>

To copy the source code example, click here [Code Example](#)

[Before Execution \[page 371\]](#)

The sped variable has been used but it is misspelled with a typo. Only the speed variable is defined.

The misspelled sped variable has been replaced by the existing speed attribute.

## Procedure

1. Position the cursor on the underlined name of the variable.
2. Get the quick assist proposals by choosing **Quick Fix** from the context menu (shortcut `Ctrl 1`) or by using the *Quick Assist* view.
3. Apply **Use similar attribute 'variable'**.

## Related Information

[Correcting an Interface Attribute Access \[page 371\]](#)

[Using Similar Variables \[page 369\]](#)

### 5.1.17.1.4.7.1.1 Code Example Before Execution

```
CLASS cl_car DEFINITION.
  PUBLIC SECTION.
    METHODS drive.
  PRIVATE SECTION.
    DATA speed TYPE i.
ENDCLASS.

CLASS cl_car IMPLEMENTATION.
  METHOD drive.
    sped = 50.
    ...
  ENDMETHOD.
ENDCLASS.
```

### 5.1.17.1.4.7.2 Correcting an Interface Attribute Access

You want to use an attribute of an interface. But you have forgotten to use the name of the interface as a prefix, followed by `~`.

## Context

You have the following options for correcting an access to the interface attributes by:

- [Adding the Interface Name \[page 372\]](#)
- [Generating an Alias \[page 373\]](#)

## Related Information

[Correcting Misspelled Variables \[page 370\]](#)

[Using Similar Variables \[page 369\]](#)

### 5.1.17.1.4.7.2.1 Adding the Interface Name

You can reuse existing interface members by adding the corresponding interface name and the '~' component selector.

## Context

Example

Before Execution	After Execution
<pre>01 INTERFACE if_car. 02   DATA: speed TYPE i. 03 ENDINTERFACE. 04 05 CLASS cl_car DEFINITION. 06   PUBLIC SECTION. 07     INTERFACES: if_car. 08     METHODS: drive 09       IMPORTING 10         i_speed TYPE i. 11   ENDCLASS. 12 13 CLASS cl_car IMPLEMENTATION. 14   METHOD drive. 15     IF speed = 0. 16     ... 17   ENDIF. 18   ENDMETHOD. 19 ENDCLASS.</pre>	<pre>01 INTERFACE if_car. 02   DATA: speed TYPE i. 03 ENDINTERFACE. 04 05 CLASS cl_car DEFINITION. 06   PUBLIC SECTION. 07     INTERFACES: if_car. 08     METHODS: drive 09       IMPORTING 10         i_speed TYPE i. 11   ENDCLASS. 12 13 CLASS cl_car IMPLEMENTATION. 14   METHOD drive. 15     IF if_car~speed = 0. 16     ... 17   ENDIF. 18   ENDMETHOD. 19 ENDCLASS.</pre>

[To copy the source code example, click here \[page 373\]](#)

The used speed variable is not declared in the cl\_car class but in the if\_car interface.

The speed variable has been replaced by if\_car~speed.

## Procedure

1. In the implementation part, position the cursor on the attribute that is defined in the interface.
2. Get the quick assist proposals by choosing **Quick Fix** from the context menu (shortcut **Ctrl** + **1**) or by using the **Quick Assist** view.
3. Apply [\*Use similar attribute ...\*](#).

### i Note

In this example, you can also choose [\*Use similar parameter ...\*](#) to use the importing parameter.

## 5.1.17.1.4.7.2.1.1 Code Example Before Execution

```
INTERFACE if_car.  
  DATA: speed TYPE i.  
ENDINTERFACE.  
  
CLASS cl_car DEFINITION.  
  PUBLIC SECTION.  
    INTERFACES: if_car.  
    METHODS: drive  
      IMPORTING  
        i_speed TYPE i.  
  ENDCLASS.  
  
CLASS cl_car IMPLEMENTATION.  
  METHOD drive.  
    IF speed = 0.  
    ...  
  ENDIF.  
  ENDMETHOD.  
ENDCLASS.
```

## 5.1.17.1.4.7.2.2 Generating an Alias

You generate and reuse an alias from a variable that points to the ~ component selector of an interface.

## Context

### Example

## Before Execution

```
01 INTERFACE if_car.
02 DATA: speed TYPE i.
03 ENDINTERFACE.
04
05 CLASS cl_car DEFINITION.
06 PUBLIC SECTION.
07 INTERFACES: if_car.
08 METHODS: drive
09 IMPORTING
10 i_speed TYPE i.
11 ENDCCLASS.
12
13 CLASS cl_car IMPLEMENTATION.
14 METHOD drive.
15 IF speed = 0.
16 ...
17 ENDIF.
18 ENDMETHOD.
19 ENDCCLASS.
```

## After Execution

```
01 INTERFACE if_car.
02 DATA: speed TYPE i.
03 ENDINTERFACE.
04
05 CLASS cl_car DEFINITION.
06 PUBLIC SECTION.
07 INTERFACES: if_car.
08 ALIASES:
09 speed FOR if_car~speed.
10 METHODS: drive
11 IMPORTING
12 i_speed TYPE i.
13 ENDCCLASS.
14
15 CLASS cl_car IMPLEMENTATION.
16 METHOD drive.
17 IF speed = 0.
18 ...
19 ENDIF.
20 ENDMETHOD.
21 ENDCCLASS.
```

To copy the source code example, click here [page 374]

The used speed variable is not declared in the cl\_car class but in the if\_car interface.

The name speed has been declared as alias for the if\_car~speed attribute in the public section of the cl\_car class where the interface is implemented.

This alias can now be used in the implementation. This enables you to access the defined attribute directly without the if\_car~ prefix.

### i Note

ADT automatically uses the name of the attribute for the alias. You can rename the alias after execution.

## Procedure

1. In the implementation part, position the cursor on the attribute for which you want use an alias.
2. Get the quick assist proposals by choosing *Quick Fix* from the context menu (shortcut **Ctrl** + **1**) or by using the *Quick Assist* view.
3. Apply *Declare alias* ....

### 5.1.17.1.4.7.2.2.1 Code Example Before Execution

```
INTERFACE if_car.
DATA: speed TYPE i.
```

```

ENDINTERFACE

CLASS cl_car DEFINITION.
  PUBLIC SECTION
    INTERFACES: if_car.
    METHODS: drive
      IMPORTING
        i_speed TYPE i.
  ENDCLASS.

CLASS cl_car IMPLEMENTATION.
  METHOD drive.
    IF speed = 0.
    ...
  ENDIF
  ENDMETHOD.
ENDCLASS.

```

## 5.1.17.1.4.8 Deleting Unused Variables

In a source code-based development object, you can delete unused variables in the entire object or only in marked source code. This enables you to automatically find and delete unused variable declarations in order to clean up source code.

### Context

This function deletes the following types of unused variables:

- Local variables within the scope of a method, function module, or form routine
- Global variables within a class, function group, or report.

#### i Note

In a class, no variables that are declared in the PUBLIC or PROTECTED sections are deleted. The deletion does not delete seemingly unused variables that could be referenced from outside.

#### i Note

Deleting an unused variable can cause another variable to become unused if it was referenced. The deletion of unused variables does not perform recursive checks for such variables. It can therefore be useful to run the function more than once in order to find and delete unused variables in such cases. You can undo the deletions using the shortcut **Ctrl** + **Z**.

Example

## Before Execution

```
01 CLASS cl_car DEFINITION.
02 PUBLIC SECTION.
03 DATA name TYPE string.
04 METHODS drive
05 IMPORTING
06 i_speed TYPE i.
07 PRIVATE SECTION.
08 DATA current_speed TYPE i.
09 ENDCCLASS.
10
11 CLASS cl_car IMPLEMENTATION.
12 METHOD drive.
13 DATA temp TYPE i.
14 DATA delta TYPE i.
15 CONSTANTS max TYPE i VALUE 220.
16 temp = 0.
17 ENDMETHOD.
18 ENDCCLASS.
```

## After Execution

```
01 CLASS cl_car DEFINITION.
02 PUBLIC SECTION.
03 DATA name TYPE string.
04 METHODS drive
05 IMPORTING
06 i_speed TYPE i.
07 PRIVATE SECTION.
08 DATA current_speed TYPE i.
09 ENDCCLASS.
10
11 CLASS cl_car IMPLEMENTATION.
12 METHOD drive.
13 DATA temp TYPE i.
14 DATA delta TYPE i.
15 CONSTANTS max TYPE i VALUE 220.
16 temp = 0.
17 ENDMETHOD.
18 ENDCCLASS.
```

To copy the source code example, click here [Code Example](#)

Before Execution [page 377]

The cl\_car class contains the name and i\_speed as well as the private variables current\_speed, delta, temp, and max public variables.

Except the temp variable, all other variables are not used.

Only the unused current\_speed, delta, and max private variables have been removed.

### i Note

To see which unused variables have been deleted, hover the marker in the *mark bar* and check the displayed tooltip.

## Procedure

1. You can delete all unused variables of a source code-based development object:
  - a. In the source code editor, position the cursor anywhere within the source code.
  - b. Get the quick assist proposals by choosing **Quick Fix** from the context menu (shortcut **Ctrl** + **1**) or by using the **Quick Assist** view.
  - c. Apply **Source Code** **Delete Unused Variables (All)** or shortcut (**Alt** + **U**).
2. You can delete all unused variables in a selected section of source code:
  - a. In the source code editor, select the source code in the method where you want to perform the deletion.
  - b. Get the quick assist proposals by choosing **Quick Fix** from the context menu (shortcut **Ctrl** + **1**) or by using the **Quick Assist** view.
  - c. Apply **Source Code** **Delete Unused Variables (Selection)** or shortcut (**Alt** + **Shift** + **U**).

## Related Information

[Extracting and Converting Variables \[page 357\]](#)  
[Extracting Local Variables from Expressions \[page 358\]](#)  
[Converting Locals to Class Members \[page 360\]](#)  
[Converting Local Variables to Parameters \[page 362\]](#)  
[Assigning a Statement to a New Variable \[page 363\]](#)  
[Declaring Inline Variables Explicitly \[page 365\]](#)  
[Declaring Variables from Usage \[page 368\]](#)  
[Using Similar Variables \[page 369\]](#)  
[Extracting Variables from Literals \[page 89\]](#)  
[Quick Assists \[page 85\]](#)  
[Quick Assist View \[page 91\]](#)

### 5.1.17.1.4.8.1 Code Example Before Execution

```
CLASS cl_car DEFINITION.
  PUBLIC SECTION.
    METHODS drive
      IMPORTING i_speed TYPE i.
  PRIVATE SECTION.
    DATA current_speed TYPE i.
ENDCLASS.

CLASS cl_car IMPLEMENTATION.
  METHOD drive.
    DATA temp TYPE i.
    DATA delta TYPE i.
    CONSTANTS max TYPE i VALUE 220.
    temp = 0.
  ENDMETHOD.
ENDCLASS.
```

### 5.1.17.1.5 Moving Members

#### Context

In the source code editor, you can substitute members of an ABAP class in order to perform any of the following functions:

Function	Description
<a href="#">Changing Visibility of Members [page 378]</a>	Changing the visibility of a member by moving it into the public, protected, or private section.

Function	Description
<a href="#">Converting Locals to Class Members [page 360]</a>	Converting a local constant, local variable, or local type to a class member such as a member constant, attribute, or member type of the current class.
<a href="#">Pull-up Members to Superclass [page 380]</a>	Removing member definitions from a subclass and adding them to the superclass.

#### i Note

- Characters are indicated with the ' quotation mark: 'Maximum speed'
- Strings are indicated with the ` quotation mark: `Maximum speed`

## Related Information

[Extracting Methods \[page 347\]](#)

[Extracting Constants \[page 353\]](#)

[Extracting and Converting Variables \[page 357\]](#)

[Exception Handling \[page 384\]](#)

[Quick Assists \[page 85\]](#)

[Quick Assist View \[page 91\]](#)

## 5.1.17.1.5.1 Changing Visibility of Members

In an ABAP class or ABAP interface, you can move a member definition between the public, protected, and private sections in order to switch its availability.

## Context

This refactoring option can be triggered for the following member types:

- Types
- Constants
- Attributes
- Methods
- Events
- Aliases

#### i Note

Depending on the current section of the selected member, two refactoring aids are available as shown in the following table:

Current Section	Option 1	Option 2
Private	Make protected	Make public
Protected	Make private	Make public
Public	Make private	Make protected

Example

Before Execution	After Execution
<pre> 01 CLASS cl_car DEFINITION. 02 PUBLIC SECTION. 03 PRIVATE SECTION. 04 DATA: speed TYPE i. 05 ENDCCLASS. 06 07 CLASS cl_car IMPLEMENTATION. 08 ... 09 ENDCCLASS. </pre>	<pre> 01 CLASS cl_car DEFINITION. 02 PUBLIC SECTION. 03 DATA: speed TYPE i. 04 PRIVATE SECTION. 05 DATA: speed TYPE i. 06 ENDCCLASS. 07 08 CLASS cl_car IMPLEMENTATION. 09 ... 10 ENDCCLASS. </pre>

To copy the source code example, click here [Code Example](#)

[Before Execution \[page 380\]](#)

The speed attribute is declared in the private section.

The declaration of the speed attribute has been removed from the private section and added in the public section.

## Decreasing Visibility

Some refactoring aids decrease (public > protected > private > local) the visibility of members of a class.

So, if the visibility of the target section is more restrictive than in the current section, the refactoring will check whether any usages exist that will no longer be available after the change. In this case, the developer is informed and can decide to reject or apply the change.

## Increasing Visibility

Some refactoring aids increase (local > private > protected > public) the visibility of members or locals inside a class.

So, if the visibility of a member (or local) is increased, subsequent processing might depend on other declarations that are no longer visible for the member. In this case, the developer is informed and can decide to reject or apply the change.

## Procedure

1. In the definition part, position the cursor on the member you want to move to another section.
2. Get the quick assist proposals by choosing **Quick Fix** from the context menu (shortcut **Ctrl + 1**) or by using the **Quick Assist** view.

3. Apply **Make 'member' protected**, **Make 'member' public**, or **Make 'member' private**. Note that the corresponding refactoring options for the possible target sections will be displayed depending on the current cursor position.

## Related Information

[Pull-up Members to Superclass \[page 380\]](#)  
[Pull-up Members to Interface \[page 382\]](#)  
[Converting Locals to Class Members \[page 360\]](#)  
[Converting Locals to Class Members \[page 360\]](#)  
[Quick Assists \[page 85\]](#)  
[Quick Assist View \[page 91\]](#)

### 5.1.17.1.5.1.1 Code Example Before Execution

```
CLASS cl_car DEFINITION.  
  PUBLIC SECTION.  
  PRIVATE SECTION.  
    DATA: speed TYPE i.  
  ENDCLASS.  
  
CLASS cl_car IMPLEMENTATION.  
  ...  
  ENDCLASS.
```

### 5.1.17.1.5.2 Pull-up Members to Superclass

You can move member definitions (for example, constants, attributes, methods, types, and events) from a subclass and to its superclass.

## Context

Example

## Before Execution

```
01 CLASS cl_vehicle DEFINITION.
02 ENDCCLASS.
03
04 CLASS cl_vehicle IMPLEMENTATION.
05 ENDCCLASS.
06
07 CLASS cl_car DEFINITION
08 INHERITING FROM cl_vehicle.
09 PRIVATE SECTION.
10 DATA: speed TYPE i.
11 ENDCCLASS.
12
13 CLASS cl_car IMPLEMENTATION.
14 ENDCCLASS.
```

## After Execution

```
01 CLASS cl_vehicle DEFINITION.
+ 02 PROTECTED SECTION.
+ 03 DATA: speed TYPE i.
04 ENDCCLASS.
05
06 CLASS cl_vehicle IMPLEMENTATION.
07 ENDCCLASS.
08
09 CLASS cl_car DEFINITION
10 INHERITING FROM cl_vehicle.
11 PRIVATE SECTION.
* 12 DATA: speed TYPE i.
13 ENDCCLASS.
14
15 CLASS cl_car IMPLEMENTATION.
16 ENDCCLASS.
```

To copy the source code example, click here [Code Example](#)

Before Execution [page 382]

The `cl_car` class contains the `speed` attribute and implements the `cl_vehicle` superclass.

The declaration of the `speed` attribute has been removed from the `cl_car` class and added to the `cl_vehicle` superclass.

The visibility of the attribute has been changed from private to protected in order to make it available for the `cl_car` subclass.

## Procedure

1. In the implementation, select the member that you want to move to the protected section of the superclass.
2. Get the quick assist proposals by choosing **Quick Fix** from the context menu (shortcut `Ctrl + 1`) or by using the *Quick Assist* view.
3. Apply **Pull-up Attribute to Super Class**.

## Related Information

[Changing Visibility of Members \[page 378\]](#)

[Pull-up Members to Interface \[page 382\]](#)

[Converting Locals to Class Members \[page 360\]](#)

[Converting Locals to Class Members \[page 360\]](#)

[Quick Assists \[page 85\]](#)

[Quick Assist View \[page 91\]](#)

### 5.1.17.1.5.2.1 Code Example Before Execution

```
CLASS cl_vehicle DEFINITION.  
ENDCLASS.  
  
CLASS cl_vehicle IMPLEMENTATION.  
ENDCLASS.  
  
CLASS cl_car DEFINITION  
  INHERITING FROM cl_vehicle.  
  PRIVATE SECTION.  
  DATA: speed TYPE i.  
ENDCLASS.  
  
CLASS cl_car IMPLEMENTATION.  
ENDCLASS.
```

### 5.1.17.1.5.3 Pull-up Members to Interface

You can move member definitions from a class and add them to the implemented interface. To avoid invalidation of existing usages, aliases are declared.

#### Context

Example

## Before Execution

```
01 INTERFACE if_vehicle.
02 ENDINTERFACE.
03
04 CLASS cl_car DEFINITION.
05 PUBLIC SECTION.
06 INTERFACES: if_vehicle.
07 METHODS: move.
08 ENDCCLASS.
09
10 CLASS cl_car IMPLEMENTATION.
11 METHOD move.
12 ENDMETHOD.
13 ENDCCLASS.
```

## After Execution

```
01 INTERFACE if_vehicle.
+ 02 METHODS: move.
03 ENDINTERFACE.
04
05 CLASS cl_car DEFINITION.
06 PUBLIC SECTION.
07 INTERFACES: if_vehicle.
* 08 METHODS: move.
+ 09 ALIASES: move FOR if_vehicle~move.
10 ENDCCLASS.
11
12 CLASS cl_car IMPLEMENTATION.
13 METHOD move.
14 ENDMETHOD.
15 ENDCCLASS.
```

To copy the source code example, click here [Code Example](#)

[Before Execution \[page 384\]](#)

The `cl_car` class contains the `move` method and implements the `if_vehicle` interface.

The declaration of the `move` method has been removed from the `cl_car` class and added to the `if_vehicle` interface.

Additionally, an alias definition has been added to the `cl_car` class to avoid invalidations of existing usages.

## Procedure

1. In the definition part, select the member that you want to move to the interface.
2. Get the quick assist proposals by choosing **Quick Fix** from the context menu (shortcut `Ctrl + 1`) or by using the **Quick Assist** view.
3. Apply **Pull-up Member with Alias**.

## Related Information

- [Changing Visibility of Members \[page 378\]](#)  
[Pull-up Members to Superclass \[page 380\]](#)  
[Pull-up Members to Interface \[page 382\]](#)  
[Converting Locals to Class Members \[page 360\]](#)  
[Converting Locals to Class Members \[page 360\]](#)  
[Quick Assists \[page 85\]](#)  
[Quick Assist View \[page 91\]](#)

### 5.1.17.1.5.3.1 Code Example Before Execution

```
INTERFACE if_vehicle.  
ENDINTERFACE.  
  
CLASS cl_car DEFINITION.  
  PUBLIC SECTION.  
    INTERFACES: if_vehicle.  
    METHODS: move.  
  ENDCLASS.  
  
CLASS cl_car IMPLEMENTATION.  
  METHOD move.  
  ENDMETHOD.  
ENDCLASS.
```

### 5.1.17.1.6 Exception Handling

If exceptions are propagated from a called procedure or raised by a `RAISE EXCEPTION` statement, they must be explicitly declared in the signature of the calling method or handled by an appropriate `TRY CATCH` block.

In the source code editor, several functionalities are available for exception handling and these can be triggered at the following positions:

- Cursor inside statement with procedure call
- Cursor inside `RAISE EXCEPTION` statement
- Selection of multiple statements with procedure calls and/or `RAISE EXCEPTION` statements

#### Source Code Selection That Includes Raised Exceptions

The exception(s) are added to the signature of the calling method or surrounded with a `TRY CATCH` block.

Function	Description
<a href="#">Propagating All Exceptions [page 386]</a>	Adding the unhandled exceptions of the selected block to the signature of the calling method in order to propagate them.
<a href="#">Surrounding with TRY CATCH [page 389]</a>	<b>Single catches:</b> Surrounding the selected block of statements with a try catch statement to handle raised exception(s). Each exception is handled in a separate catch block
	<b>Multi catch:</b> Surrounding the selected block of statements with a try catch statement to handle raised exception(s). One catch block handles all exceptions.

#### Existing TRY CATCH Block

The statement(s) are surrounded by a `TRY CATCH` block in order to handle the exceptions.

Function	Description
<a href="#">Extracting a Catch Variable [page 393]</a>	Adding a local variable and adding the INTO clause to an existing catch block.
<a href="#">Extending a TRY CATCH Statement [page 397]</a>	Adding a new catch block to an existing try catch statement.
<a href="#">Removing a TRY CATCH Statement [page 401]</a>	Removing the entire try catch statement.
<a href="#">Splitting a MULTI CATCH Block [page 399]</a>	Replacing the existing multi catch block by individual catch blocks per exception.

## RAISE EXCEPTION Statement

Function	Description
<a href="#">Propagating an Exception [page 388]</a>	Adding an exception class to the signature of a method signature that is based on an existing RAISE EXCEPTION statement.
<a href="#">Extracting an Exception Variable from a RAISE Statement [page 395]</a>	Adding a variable declaration for the exception and source code to instantiate the exception before raising it.

## Related Information

- [Extracting Methods \[page 347\]](#)
- [Extracting Constants \[page 353\]](#)
- [Extracting and Converting Variables \[page 357\]](#)
- [Moving Members \[page 377\]](#)
- [Quick Assists \[page 85\]](#)
- [Quick Assist View \[page 91\]](#)

## 5.1.17.1.6.1 Propagating All Exceptions

You can add the unhandled exceptions of a selected block to the signature of the calling method in order to propagate them.

### Context

#### Example

##### Before Execution

```
01 CLASS cl_car DEFINITION.
02 PUBLIC SECTION.
03 METHODS:
04   drive,
05   check_fuel RAISING cx_no_fuel.
06 DATA:
07   engine_ok TYPE abap_bool.
08 ENDCLASS.
09
10 CLASS cl_car IMPLEMENTATION.
11 METHOD drive.
12   check_fuel( ).
13   IF engine_ok = abap_false.
14     RAISE EXCEPTION TYPE cx_failure.
15   ENDIF.
16 ENDMETHOD.
17 ...
18 ENDCLASS.
```

##### After Execution

```
01 CLASS cl_car DEFINITION.
02 PUBLIC SECTION.
03 METHODS:
04   drive RAISING cx_no_fuel cx_failure,
05   check_fuel RAISING cx_no_fuel.
06 DATA:
07   engine_ok TYPE abap_bool.
08 ENDCLASS.
09
10 CLASS cl_car IMPLEMENTATION.
11 METHOD drive.
12   check_fuel( ).
13   IF engine_ok = abap_false.
14     RAISE EXCEPTION TYPE cx_failure.
15   ENDIF.
16 ENDMETHOD.
17 ...
18 ENDCLASS.
```

To copy the source code example, click here [Code Example Before Execution \[page 387\]](#)

The selection contains two statements that can raise exceptions. `cx_no_fuel` and `cx_failure` have either to be handled inside the method or propagated by adding them to the method signature. For both cases, ABAP Development Tools (ADT) provides a quick assist.

The `cx_failure` and `cx_no_fuel` exceptions are added to the signature of the `drive` method. The exception will now be propagated to the caller of the `drive` method automatically at runtime.

To propagate exceptions, proceed as follows:

#### i Note

To handle them inside the method, proceed as described in [Surrounding with TRY CATCH \[page 389\]](#)

### Procedure

1. In the implementation part, select the lines where the `RAISE` statement is called in a method.

2. Get the quick assist proposals by choosing *Quick Fix* from the context menu (shortcut **Ctrl** + **1**) or by using the *Quick Assist* view.
3. Apply *Propagate All Exceptions*.

## Related Information

[Exception Handling \[page 384\]](#)

[Surrounding with TRY CATCH \[page 389\]](#)

[Propagating an Exception \[page 388\]](#)

[Extracting a Catch Variable \[page 393\]](#)

[Extracting an Exception Variable from a RAISE Statement \[page 395\]](#)

[Extending a TRY CATCH Statement \[page 397\]](#)

[Removing a TRY CATCH Statement \[page 401\]](#)

[Splitting a MULTI CATCH Block \[page 399\]](#)

### 5.1.17.1.6.1.1 Code Example Before Execution

```

CLASS cx_no_fuel DEFINITION
  INHERITING FROM cx_static_check.
ENDCLASS.

CLASS cx_failure DEFINITION
  INHERITING FROM cx_static_check.
ENDCLASS.

CLASS cl_car DEFINITION.
  PUBLIC SECTION.
    METHODS:
      drive,
      check_fuel RAISING cx_no_fuel.
    DATA:
      engine_ok TYPE abap_bool.
ENDCLASS.

CLASS cl_car IMPLEMENTATION.
  METHOD drive.
    check_fuel( ).
    IF engine_ok = abap_false.
      RAISE EXCEPTION TYPE cx_failure.
    ENDIF.
  ENDMETHOD.

  METHOD check_fuel.
    RAISE EXCEPTION TYPE cx_no_fuel.
  ENDMETHOD.
ENDCLASS.

```

## 5.1.17.1.6.2 Propagating an Exception

You can add an exception to the method signature based on an existing `RAISE EXCEPTION` statement of the method implementation.

### Context

Example

Before Execution	After Execution
<pre>01 CLASS cl_car DEFINITION. 02 PUBLIC SECTION. 03 METHODS: drive 04 IMPORTING 05 i_speed TYPE i. 06 ENDCCLASS. 07 08 CLASS cl_car IMPLEMENTATION. 09 METHOD drive. 10 IF i_speed &lt; 0. 11 RAISE EXCEPTION TYPE cx_error. 12 ENDIF. 13 ... 14 ENDMETHOD. 15 ENDCCLASS.</pre>	<pre>01 CLASS cl_car DEFINITION. 02 PUBLIC SECTION. 03 METHODS: drive 04 IMPORTING 05 i_speed TYPE i. 06 RAISING 07 cx_error. 08 ENDCCLASS. 09 10 CLASS cl_car IMPLEMENTATION. 11 METHOD drive. 12 IF i_speed &lt; 0. 13 RAISE EXCEPTION TYPE cx_error. 14 ENDIF. 15 ... 16 ENDMETHOD. 17 ENDCCLASS.</pre>

To copy the source code example, click here [Code Example](#)

[Before Execution \[page 389\]](#)

In the implementation of the `drive` method, the `check_fuel` method that can raise an exception is called. In order to propagate the exception, you need to define it in the public section.

In the signature of the `drive` method, the `cx_no_fuel` exception class is added to the `RAISING` clause.

### Procedure

1. In the implementation part, select the exception class that you want to define in the method signature.
2. Get the quick assist proposals by choosing **Quick Fix** from the context menu (shortcut `Ctrl 1`) or by using the **Quick Assist** view.
3. Apply **Add raising declaration**.

## Related Information

[Exception Handling \[page 384\]](#)  
[Propagating All Exceptions \[page 386\]](#)  
[Surrounding with TRY CATCH \[page 389\]](#)  
[Extracting a Catch Variable \[page 393\]](#)  
[Extracting an Exception Variable from a RAISE Statement \[page 395\]](#)  
[Extending a TRY CATCH Statement \[page 397\]](#)  
[Removing a TRY CATCH Statement \[page 401\]](#)  
[Splitting a MULTI CATCH Block \[page 399\]](#)

### 5.1.17.1.6.2.1 Code Example Before Execution

```
CLASS cx_error DEFINITION
  INHERITING FROM cx_static_check.
ENDCLASS.
CLASS cl_car DEFINITION.
  PUBLIC SECTION.
    METHODS: drive
      IMPORTING
        i_speed TYPE i.
  ENDMETHOD.
CLASS cl_car IMPLEMENTATION.
  METHOD drive.
    IF i_speed < 0.
      RAISE EXCEPTION TYPE cx_error.
    ENDIF.
  ENDMETHOD.
ENDCLASS.
```

### 5.1.17.1.6.3 Surrounding with TRY CATCH

You can surround the selected block of statements with a TRY CATCH statement in order to handle raised exception(s) as a single or multiple TRY CATCH block.

## Context

Example with Single CATCH

## Before Execution

```
01 CLASS cl_car DEFINITION.
02 PUBLIC SECTION.
03 METHODS:
04   drive,
05   check_fuel RAISING cx_no_fuel,
06   check_engine RAISING cx_failure.
07 ENDCLASS.
08
09 CLASS cl_car IMPLEMENTATION.
10 METHOD drive.
11   check_fuel( ).
12   check_engine( ).
13 ENDMETHOD.
14 ...
15 ENDCCLASS.
```

## After Execution

```
01 CLASS cl_car DEFINITION.
02 PUBLIC SECTION.
03 METHODS:
04   drive,
05   check_fuel RAISING cx_no_fuel,
06   check_engine RAISING cx_failure.
07 ENDCLASS.
08
09 CLASS cl_car IMPLEMENTATION.
10 METHOD drive.
11 TRY.
12   check_fuel( ).
13   check_engine( ).
14 CATCH cx_no_fuel.
15   "handle exception
16 CATCH cx_failure.
17   "handle exception
18 ENDTRY.
19 ENDMETHOD.
20 ...
21 ENDCCLASS.
```

To copy the source code example, click here [Code Example](#)

### Before Execution [page 392]

In the implementation of the `drive` method, the `check_fuel` and `check_engine` methods are called. The `check_fuel` and `check_engine` methods raise an exception of the `cx_failure` and `cx_no_fuel` types. The signature of the `drive` method does not contain any exception.

In the `drive` method, the selection contains the call of the `check_fuel` and `check_engine` methods that can raise exceptions. You want to handle the `cx_no_fuel` and `cx_failure` exceptions that might occur in the `drive` method.

In the `drive` method, the selection contains the call of the `check_fuel` and `check_engine` methods that can raise exceptions. You want to handle the `cx_no_fuel` and `cx_failure` exceptions that might occur in the `drive` method.

### Example with Multi CATCH

## Before Execution

```
01 CLASS cl_car DEFINITION.
02 PUBLIC SECTION.
03 METHODS:
04   drive,
05   check_fuel RAISING cx_no_fuel,
06   check_engine RAISING cx_failure.
07 ENDCCLASS.
08
09 CLASS cl_car IMPLEMENTATION.
10 METHOD drive.
11   check_fuel( ).  
12   check_engine( ).  
13 ENDMETHOD.
14 ...
15 ENDCCLASS.
```

## After Execution

```
01 CLASS cl_car DEFINITION.
02 PUBLIC SECTION.
03 METHODS:
04   drive,
05   check_fuel RAISING cx_no_fuel,
06   check_engine RAISING cx_failure.
07 ENDCCLASS.
08
09 CLASS cl_car IMPLEMENTATION.
10 METHOD drive.
11 TRY.
12   check_fuel( ).  
13   check_engine( ).  
14 CATCH cx_no_fuel cx_failure.
15   "handle exception"
16 ENDTRY.
17 ENDMETHOD.
18 ...
19 ENDCCLASS.
```

To copy the source code example, click here [Code Example](#)

[Before Execution \[page 392\]](#)

In the implementation of the `drive` method, the `check_fuel` and `check_engine` methods are called. The `check_fuel` and `check_engine` methods raise an exception of the `cx_failure` and `cx_no_fuel` types. The signature of the `drive` method does not contain any exception.

A `TRY CATCH` block is added to the `drive` method. If you have selected the call, a `CATCH` block for each exception (`cx_no_fuel` and `cx_failure`) is added. Inside each `CATCH` block, you can add code to handle the exceptions.

To handle this, ABAP Development Tools (ADT) provides two options:

## Procedure

1. In the implementation part, select the source code that can raise one or more exceptions.
2. Get the quick assist proposals by choosing *Quick Fix* from the context menu (shortcut `Ctrl 1`) or by using the *Quick Assist* view.
3. Apply *Surround all with TRY CATCH*. If you have selected *Surround all with TRY MULTI CATCH*, all raised exceptions are added to one `CATCH` block.

### i Note

You can also select several methods that are called. You then need to choose **Surround all with TRY MULTI CATCH** in order to handle all raised exceptions that are defined for the selected method(s) in the public section. Thus, all of the raised exceptions are added and handled in one `CATCH` block.

## Related Information

[Exception Handling \[page 384\]](#)  
[Propagating All Exceptions \[page 386\]](#)  
[Propagating an Exception \[page 388\]](#)  
[Extracting a Catch Variable \[page 393\]](#)  
[Extracting an Exception Variable from a RAISE Statement \[page 395\]](#)  
[Extending a TRY CATCH Statement \[page 397\]](#)  
[Removing a TRY CATCH Statement \[page 401\]](#)  
[Splitting a MULTI CATCH Block \[page 399\]](#)

### 5.1.17.1.6.3.1 Code Example Before Execution

```
CLASS cx_failure DEFINITION
  INHERITING FROM cx_static_check.
ENDCLASS.

CLASS cx_no_fuel DEFINITION
  INHERITING FROM cx_static_check.
ENDCLASS.

CLASS cl_car DEFINITION.
  PUBLIC SECTION.
    METHODS:
      drive,
      check_fuel RAISING cx_no_fuel,
      check_engine RAISING cx_failure.
  ENDCLASS.

CLASS cl_car IMPLEMENTATION.
  METHOD drive.
    check_fuel( ).
    check_engine( ).
  ENDMETHOD.

  METHOD check_fuel.
    RAISE EXCEPTION TYPE cx_no_fuel.
  ENDMETHOD.

  METHOD check_engine.
    RAISE EXCEPTION TYPE cx_failure.
  ENDMETHOD.
ENDCLASS.
```

## 5.1.17.1.6.4 Extracting a Catch Variable

You can add a local variable to the method signature. Its type refers to the raised exception of the calling method. In addition, this local variable is added as an `INTO` clause to the already existing `CATCH` block. This enables you to handle the result of the exception in order, for example, to display it in a message.

### Context

Example

Before Execution	After Execution
<pre>01 CLASS cl_car DEFINITION. 02 PUBLIC SECTION. 03 METHODS: 04   drive, 05   check_fuel RAISING cx_no_fuel. 06 ENDCCLASS. 07 08 CLASS cl_car IMPLEMENTATION. 09 METHOD drive. 10   TRY. 11     check_fuel( ). 12     CATCH cx_no_fuel. 13       "handle exception 14     ENDTRY. 15   ENDMETHOD. 16 ... 17 ENDCCLASS.</pre>	<pre>01 CLASS cl_car DEFINITION. 02 PUBLIC SECTION. 03 METHODS: 04   drive, 05   check_fuel RAISING cx_no_fuel. 06 ENDCCLASS. 07 08 CLASS cl_car IMPLEMENTATION. 09 METHOD drive. +10   DATA: exc TYPE REF TO cx_no_fuel. 11   TRY. 12     check_fuel( ). 13     CATCH cx_no_fuel INTO exc. 14       "handle exception 15     ENDTRY. 16   ENDMETHOD. 17 ... 18 ENDCCLASS.</pre>

To copy the source code example, click here [Code Example](#)

[Before Execution \[page 394\]](#)

In the implementation of the `drive` method, the `check_fuel` method is called in a `TRY CATCH` block. Here, the `cx_no_fuel` exception that is raised in the public section is handled.

The result of the exception now needs be handled as a local variable after it is raised.

In the signature of the `drive` method, the `exc` local variable is added with `TYPE REF TO` of the selected exception class. In addition, this local variable is added as an `INTO` clause at the end of the `CATCH` statement.

### Procedure

1. In the implementation part, select the `CATCH` statement where the exception class is called.
2. Get the quick assist proposals by choosing **Quick Fix** from the context menu (shortcut `Ctrl + 1`) or by using the [Quick Assist](#) view.

3. Apply **Extract Catch Variable**.

## Related Information

[Exception Handling \[page 384\]](#)  
[Propagating All Exceptions \[page 386\]](#)  
[Surrounding with TRY CATCH \[page 389\]](#)  
[Propagating an Exception \[page 388\]](#)  
[Extracting an Exception Variable from a RAISE Statement \[page 395\]](#)  
[Extending a TRY CATCH Statement \[page 397\]](#)  
[Removing a TRY CATCH Statement \[page 401\]](#)  
[Splitting a MULTI CATCH Block \[page 399\]](#)

## 5.1.17.1.6.4.1 Code Example Before Execution

```
CLASS cx_no_fuel DEFINITION
  INHERITING FROM cx_static_check.
ENDCLASS.

CLASS cl_car DEFINITION.
  PUBLIC SECTION.
  METHODS:
    drive,
    check_fuel RAISING cx_no_fuel.
ENDCLASS.

CLASS cl_car IMPLEMENTATION.
  METHOD drive.
    TRY.
      check_fuel( ).
      CATCH cx_no_fuel.
        "handle exception
    ENDTRY.
  ENDMETHOD.

  METHOD check_fuel.
    RAISE EXCEPTION TYPE cx_no_fuel.
  ENDMETHOD.
ENDCLASS.
```

## 5.1.17.1.6.5 Extracting an Exception Variable from a RAISE Statement

You can add a variable declaration for an exception to the method signature as well as to source code in order to instantiate the exception before raising it. This enables you to handle the result of the exception, for example, to display it in a message.

### Context

Example

Before Execution	After Execution
<pre>01 CLASS cl_car DEFINITION. 02 PUBLIC SECTION. 03 METHODS: drive 04 IMPORTING 05 i_speed TYPE i 06 RAISING 07 cx_error. 08 ENDCLASS. 09 10 CLASS cl_car IMPLEMENTATION. 11 METHOD drive. 12 IF i_speed &lt; 0. 13 RAISE EXCEPTION TYPE cx_error. 14 ENDIF. 15 ... 16 ENDMETHOD. 17 ENDCLASS.</pre>	<pre>01 CLASS cl_car DEFINITION. 02 PUBLIC SECTION. 03 METHODS: drive 04 IMPORTING 05 i_speed TYPE i 06 RAISING 07 cx_error. 08 ENDCLASS. 09 10 CLASS cl_car IMPLEMENTATION. 11 METHOD drive. 12 DATA: exc TYPE REF TO cx_error. 13 IF i_speed &lt; 0. 14 CREATE OBJECT exc. 15 RAISE EXCEPTION TYPE cx_error. 16 RAISE EXCEPTION exc. 17 ENDIF. 18 ... 19 ENDMETHOD. 20 ENDCLASS.</pre>

To copy the source code example, click here [Code Example](#)

[Before Execution \[page 396\]](#)

In the implementation of the `drive` method, an exception with the `cx_error` type is raised when the `i_speed` variable is less than zero.

In the signature of the `drive` method, the `exc` exception variable is added with `TYPE REF TO` of the selected exception class. In addition, this exception variable is added in a `CREATE` statement and called in a `RAISE` statement. The former `RAISE EXCEPTION TYPE` statement is deleted.

You can now reuse or process the value of the exception variable in order to handle this case of an exception.

## Procedure

1. In the implementation part, select the exception class where it is called in the RAISE EXCEPTION statement.
2. Get the quick assist proposals by choosing **Quick Fix** from the context menu (shortcut **Ctrl 1**) or by using the *Quick Assist* view.
3. Apply **Extract Raise Variable**.

## Related Information

[Exception Handling \[page 384\]](#)

[Propagating All Exceptions \[page 386\]](#)

[Surrounding with TRY CATCH \[page 389\]](#)

[Propagating an Exception \[page 388\]](#)

[Extracting a Catch Variable \[page 393\]](#)

[Extending a TRY CATCH Statement \[page 397\]](#)

[Removing a TRY CATCH Statement \[page 401\]](#)

[Splitting a MULTI CATCH Block \[page 399\]](#)

## 5.1.17.1.6.5.1 Code Example Before Execution

```
CLASS cx_error DEFINITION
  INHERITING FROM cx_static_check.
ENDCLASS.
CLASS cl_car DEFINITION.
  PUBLIC SECTION.
    METHODS: drive
      IMPORTING
        i_speed TYPE i
      RAISING
        cx_error.
  ENDCLASS.
CLASS cl_car IMPLEMENTATION.
  METHOD drive.
    IF i_speed < 0.
      RAISE EXCEPTION TYPE cx_error.
    ENDIF.
  ENDMETHOD.
ENDCLASS.
```

## 5.1.17.1.6.6 Extending a TRY CATCH Statement

You can add a new CATCH block to an existing TRY CATCH statement. This enables you to handle another exception in an existing TRY CATCH statement.

### Context

Example

Before Execution	After Execution
<pre>01 CLASS cl_car DEFINITION. 02 PUBLIC SECTION. 03 METHODS: 04   drive, 05   check_fuel RAISING cx_no_fuel, 06   check_engine RAISING cx_failure. 07 ENDCLASS. 08 09 CLASS cl_car IMPLEMENTATION. 10 METHOD drive. 11   TRY. 12     check_fuel( ). 13     check_engine( ). 14   CATCH cx_no_fuel. 15     "handle exception 16     CLEANUP. 17     "cleanup 18   ENDTRY. 19   ENDMETHOD. 20 ... 21 ENDCLASS.</pre>	<pre>01 CLASS cl_car DEFINITION. 02 PUBLIC SECTION. 03 METHODS: 04   drive, 05   check_fuel RAISING cx_no_fuel, 06   check_engine RAISING cx_failure. 07 ENDCLASS. 08 09 CLASS cl_car IMPLEMENTATION. 10 METHOD drive. 11   TRY. 12     check_fuel( ). 13     check_engine( ). 14   CATCH cx_no_fuel. 15     "handle exception 16   CATCH cx_failure. 17     "handle exception 18     CLEANUP. 19     "cleanup 20   ENDTRY. 21   ENDMETHOD. 22 ... 23 ENDCLASS.</pre>

To copy the source code example, click

here [Code Example Before Execution](#)

[page 398]

In the implementation of the drive method, the check\_engine method is called. This method can raise exceptions of the type cx\_failure.

The exception now needs be handled in a new CATCH block.

In the drive method, the new CATCH block is added where the cx\_failure exception can now be handled.

### Procedure

1. In the implementation part, select the method that can raise an exception that needs to be handled in a new CATCH block.
2. Get the quick assist proposals by choosing **Quick Fix** from the context menu (shortcut **Ctrl + 1**) or by using the **Quick Assist** view.
3. Apply **Extend Try Catch**.

## Related Information

[Exception Handling \[page 384\]](#)  
[Propagating All Exceptions \[page 386\]](#)  
[Surrounding with TRY CATCH \[page 389\]](#)  
[Propagating an Exception \[page 388\]](#)  
[Extracting a Catch Variable \[page 393\]](#)  
[Extracting an Exception Variable from a RAISE Statement \[page 395\]](#)  
[Removing a TRY CATCH Statement \[page 401\]](#)  
[Splitting a MULTI CATCH Block \[page 399\]](#)

### 5.1.17.1.6.6.1 Code Example Before Execution

```
CLASS cx_failure DEFINITION
  INHERITING FROM cx_static_check.
ENDCLASS.

CLASS cx_no_fuel DEFINITION
  INHERITING FROM cx_static_check.
ENDCLASS.

CLASS cl_car DEFINITION.
  PUBLIC SECTION.
    METHODS:
      drive,
      check_fuel RAISING cx_no_fuel,
      check_engine RAISING cx_failure.
  ENDCLASS.

CLASS cl_car IMPLEMENTATION.
  METHOD drive.
    TRY.
      check_fuel( ).
      check_engine( ).
      CATCH cx_no_fuel.
        "handle exception
      CLEANUP.
        "cleanup
    ENDTRY.
  ENDMETHOD.

  METHOD check_fuel.
    RAISE EXCEPTION TYPE cx_no_fuel.
  ENDMETHOD.

  METHOD check_engine.
    RAISE EXCEPTION TYPE cx_failure.
  ENDMETHOD.
ENDCLASS.
```

## 5.1.17.1.6.7 Splitting a MULTI CATCH Block

You can replace an existing MULTI CATCH block by individual CATCH blocks for each exception separately.

### Context

Example

Before Execution	After Execution
<pre>01 CLASS cl_car DEFINITION. 02 PUBLIC SECTION. 03 METHODS: 04   drive, 05   check_fuel RAISING cx_no_fuel, 06   check_engine RAISING cx_failure. 07 ENDCCLASS. 08 09 CLASS cl_car IMPLEMENTATION. 10 METHOD drive. 11   TRY. 12     check_fuel( ). 13     check_engine( ). 14     CATCH cx_no_fuel cx_failure. 15     "handle exception 16   ENDTRY. 17 ENDMETHOD. 18 ... 19 ENDCCLASS.</pre>	<pre>01 CLASS cl_car DEFINITION. 02 PUBLIC SECTION. 03 METHODS: 04   drive, 05   check_fuel RAISING cx_no_fuel, 06   check_engine RAISING cx_failure. 07 ENDCCLASS. 08 09 CLASS cl_car IMPLEMENTATION. 10 METHOD drive. 11   TRY. 12     check_fuel( ). 13     check_engine( ). 14     CATCH cx_no_fuel cx_failure. 15     "handle exception 16     CATCH cx_no_fuel. 17     "handle exception 18     CATCH cx_failure. 19     "handle exception 20   ENDTRY. 21 ENDMETHOD. 22 ... 23 ENDCCLASS.</pre>

To copy the source code example, click here [Code Example](#)

[Before Execution \[page 400\]](#)

In the implementation of the drive method, a MULTI CATCH block handles the cx\_no\_fuel and cx\_failure exceptions.

You now want the exceptions to be handled differently in their own CATCH block.

In the drive method, the MULTI CATCH block is divided into two new single CATCH blocks for each exception.

### Procedure

1. In the implementation part, select the MULTI CATCH block where you want to catch the exceptions separately.
2. Get the quick assist proposals by choosing **Quick Fix** from the context menu (shortcut **Ctrl + 1**) or by using the **Quick Assist** view.

3. Apply **Split Multi Catch**.

## Related Information

[Exception Handling \[page 384\]](#)  
[Propagating All Exceptions \[page 386\]](#)  
[Surrounding with TRY CATCH \[page 389\]](#)  
[Propagating an Exception \[page 388\]](#)  
[Extracting a Catch Variable \[page 393\]](#)  
[Extracting an Exception Variable from a RAISE Statement \[page 395\]](#)  
[Extending a TRY CATCH Statement \[page 397\]](#)  
[Removing a TRY CATCH Statement \[page 401\]](#)

## 5.1.17.1.6.7.1 Code Example Before Execution

```
CLASS cx_failure DEFINITION
  INHERITING FROM cx_static_check.
ENDCLASS.
CLASS cx_no_fuel DEFINITION
  INHERITING FROM cx_static_check.
ENDCLASS.
CLASS cl_car DEFINITION.
  PUBLIC SECTION.
  METHODS:
    drive,
    check_fuel RAISING cx_no_fuel,
    check_engine RAISING cx_failure.
ENDCLASS.
CLASS cl_car IMPLEMENTATION.
  METHOD drive.
    TRY.
      check_fuel( ).
      check_engine( ).
    CATCH cx_no_fuel cx_failure.
      "handle exception
    ENDTRY.
  ENDMETHOD.
  METHOD check_fuel.
    RAISE EXCEPTION TYPE cx_no_fuel.
  ENDMETHOD.
  METHOD check_engine.
    RAISE EXCEPTION TYPE cx_failure.
  ENDMETHOD.
ENDCLASS.
```

## 5.1.17.1.6.8 Removing a TRY CATCH Statement

You can remove an entire TRY CATCH statement if you no longer want to handle the exceptions.

### Context

Example

Before Execution	After Execution
<pre>01 CLASS cl_car DEFINITION. 02 PUBLIC SECTION. 03 METHODS: 04   drive, 05   check_fuel RAISING cx_no_fuel, 06   check_engine RAISING cx_failure. 07 ENDCCLASS. 08 09 CLASS cl_car IMPLEMENTATION. 10 METHOD drive. 11   TRY. 12     check_fuel( ). 13     check_engine( ). 14     CATCH cx_no_fuel. 15     "handle exception 16     CATCH cx_failure. 17     "handle exception 18   ENDTRY. 19 ENDMETHOD. 20 ... 21 ENDCCLASS.</pre>	<pre>01 CLASS cl_car DEFINITION. 02 PUBLIC SECTION. 03 METHODS: 04   drive, 05   check_fuel RAISING cx_no_fuel, 06   check_engine RAISING cx_failure. 07 ENDCCLASS. 08 09 CLASS cl_car IMPLEMENTATION. 10 METHOD drive. 11   TRY. 12     check_fuel( ). 13     check_engine( ). 14     CATCH cx_no_fuel. 15     "handle exception 16     CATCH cx_failure. 17     "handle exception 18   ENDTRY. 19 ENDMETHOD. 20 ... 21 ENDCCLASS.</pre>

To copy the source code example, click here [Code Example](#)

[Before Execution \[page 402\]](#)

In the implementation of the `drive` method, the `cx_no_fuel` and `cx_failure` exceptions are handled.

The TRY CATCH statement should now be removed because the exceptions are no longer to be handled.

In the `drive` method, the TRY CATCH statement – including the CATCH blocks – is removed. Only the content of the former TRY block remains.

### Procedure

1. In the implementation part, select the TRY statement of the TRY CATCH statement you want to remove.
2. Get the quick assist proposals by choosing **Quick Fix** from the context menu (shortcut `Ctrl + 1`) or by using the *Quick Assist* view.
3. Apply **Remove Try Catch**.

## Related Information

[Exception Handling \[page 384\]](#)  
[Propagating All Exceptions \[page 386\]](#)  
[Surrounding with TRY CATCH \[page 389\]](#)  
[Propagating an Exception \[page 388\]](#)  
[Extracting a Catch Variable \[page 393\]](#)  
[Extracting an Exception Variable from a RAISE Statement \[page 395\]](#)  
[Extending a TRY CATCH Statement \[page 397\]](#)  
[Splitting a MULTI CATCH Block \[page 399\]](#)

### 5.1.17.1.6.8.1 Code Example Before Execution

```
CLASS cx_failure DEFINITION
  INHERITING FROM cx_static_check.
ENDCLASS.

CLASS cx_no_fuel DEFINITION
  INHERITING FROM cx_static_check.
ENDCLASS.

CLASS cl_car DEFINITION.
  PUBLIC SECTION.
  METHODS:
    drive,
    check_fuel RAISING cx_no_fuel,
    check_engine RAISING cx_failure.
ENDCLASS.

CLASS cl_car IMPLEMENTATION.
  METHOD drive.
    TRY.
      check_fuel( ).
      check_engine( ).
      CATCH cx_no_fuel.
        "handle exception
      CATCH cx_failure.
        "handle exception
    ENDTRY.
  ENDMETHOD.

  METHOD check_fuel.
    RAISE EXCEPTION TYPE cx_no_fuel.
  ENDMETHOD.

  METHOD check_engine.
    RAISE EXCEPTION TYPE cx_failure.
  ENDMETHOD.
ENDCLASS.
```

### 5.1.17.2 Applying ABAP Quick Fixes

#### Context

In the source code editor, the following ABAP Quick Fixes are provided:

Function	Description
<a href="#">Creating Method Implementations from the Method Definition [page 404]</a>	Creating an empty method implementation from an existing declaration in an ABAP class.
<a href="#">Creating Method Definitions from Implementation Parts [page 406]</a>	Creating the empty method implementations of the methods that are defined in an implemented ABAP interface and of other unimplemented methods within an ABAP class.
<a href="#">Creating Implementation Parts for Unimplemented Methods [page 409]</a>	Promoting the implementation part of methods that are defined in an ABAP interface and method stubs of other unimplemented methods.
<a href="#">Creating Methods from Method Calls [page 414]</a>	Creating a method from the method call. The signature is derived from the existing method call.
<a href="#">Creating ABAP Classes or ABAP Interfaces from Usage [page 417]</a>	Starting the creation wizard of a global ABAP class or interface directly from the name of the missing repository object.
<a href="#">Creating ABAP Includes from Usage [page 421]</a>	Starting the creation wizard of an ABAP include from the name of the missing repository object.
<a href="#">Creating ABAP Function Modules from Usage [page 420]</a>	Starting the creation wizard of an ABAP function module from the name of the missing repository object.

You have the following options for executing quick fixes:

- Shortcut `Ctrl 1`
- Entry   in the menu bar
- Entry **Quick Fix** in the context menu
- Entry in *Quick Assist* view

## Related Information

- [Creating Definitions and Implementations of Methods \[page 404\]](#)
- [Creating ABAP Classes or ABAP Interfaces from Usage \[page 417\]](#)
- [Creating ABAP Function Modules from Usage \[page 420\]](#)
- [Creating ABAP Includes from Usage \[page 421\]](#)
- [Refactoring ABAP Source Code \[page 342\]](#)
- [Applying Other Quick Assists \[page 422\]](#)
- [Quick Assists \[page 85\]](#)
- [Quick Assist View \[page 91\]](#)

## 5.1.17.2.1 Creating Definitions and Implementations of Methods

### Context

ABAP Development Tools (ADT) provides the following quick fix tools to generate and synchronize definitions and implementations of methods:

- [Creating Method Implementations from the Method Definition \[page 404\]](#)
- [Creating Method Definitions from Implementation Parts \[page 406\]](#)
- [Creating Implementation Parts for Unimplemented Methods \[page 409\]](#)
- [Creating Methods from Method Calls \[page 414\]](#)

#### i Note

The quick fixes can be opened at the following positions:

- Corresponding position within the definition or implementation
- Error symbol or warning in the ruler of the source code editor
- Context menu in the *ABAP Problems* view

### Related Information

[Creating ABAP Classes or ABAP Interfaces from Usage \[page 417\]](#)

[Creating ABAP Function Modules from Usage \[page 420\]](#)

[Creating ABAP Includes from Usage \[page 421\]](#)

[Quick Assists \[page 85\]](#)

[Quick Assist View \[page 91\]](#)

### 5.1.17.2.1.1 Creating Method Implementations from the Method Definition

#### Prerequisites

In the ABAP class where you want to create the empty implementation part of a method, the declaration of the method must exist in the definition part.

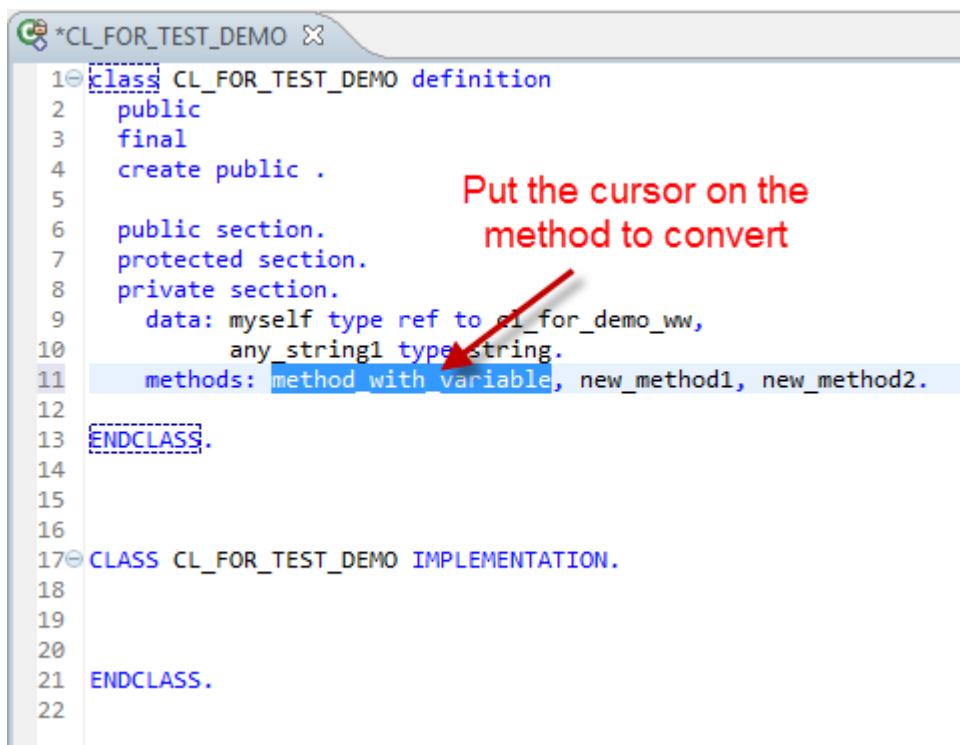
The ABAP class must be free of syntax errors, other than those for the missing method implementation(s).

## Context

In an ABAP class, you can generate the implementation part of a method from an existing declaration in the definition part. The added implementation part is empty and can be edited for development activities.

## Procedure

1. In the source code editor, position the cursor on a method definition that has no implementation part yet.



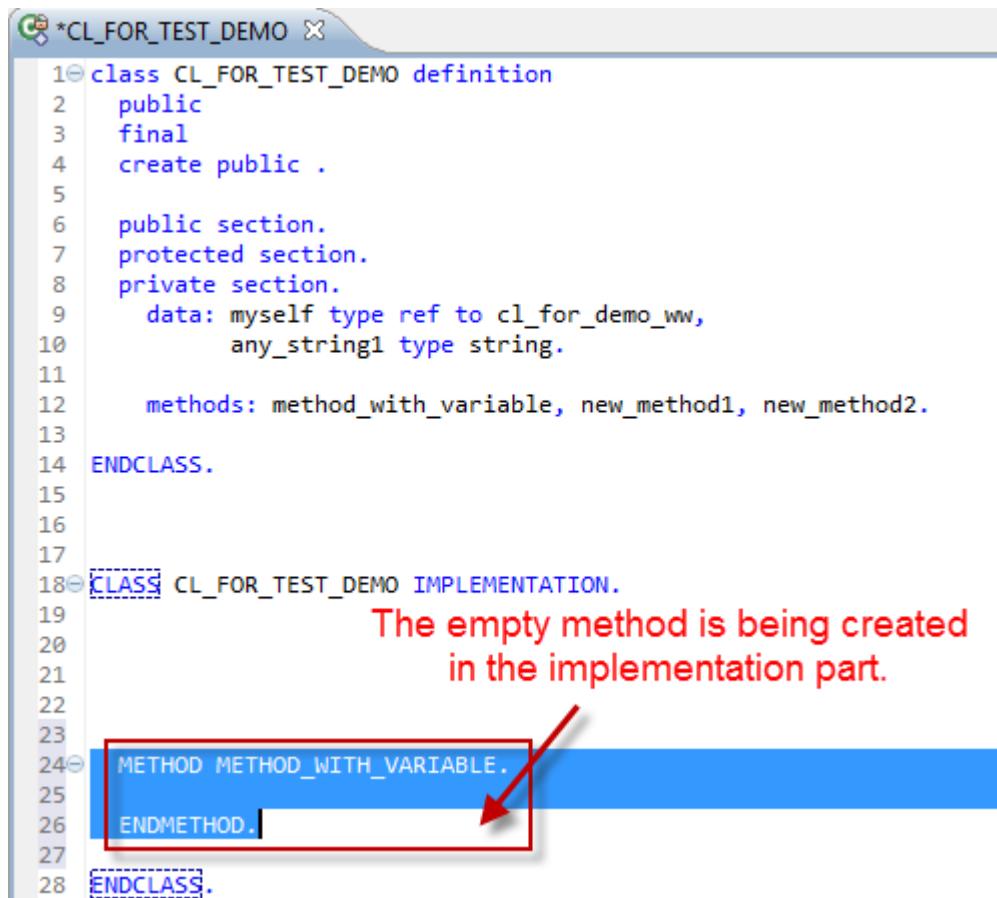
```
1 class CL_FOR_TEST_DEMO definition
2   public
3   final
4   create public .
5
6   public section.
7   protected section.
8   private section.
9     data: myself type ref to cl_for_demo_ww,
10        any_string1 type string.
11   methods: method with variable, new_method1, new_method2.
12
13 ENDCCLASS.
14
15
16
17 CLASS CL_FOR_TEST_DEMO IMPLEMENTATION.
18
19
20
21 ENDCCLASS.
22
```

Creating the implementation part out of the method definition

2. Choose **Quick Fix** (Ctrl 1) from the context menu.
3. In the Quick Fix dialog, double-click **Create implementation 'method\_with\_variable'**.

## Results

ABAP Development Tools inserts the corresponding (empty) implementation part of the method into the existing implementation of the ABAP class.



```

1① class CL_FOR_TEST_DEMO definition
2  public
3  final
4  create public .
5
6  public section.
7  protected section.
8  private section.
9    data: myself type ref to cl_for_demo_ww,
10       any_string1 type string.
11
12   methods: method_with_variable, new_method1, new_method2.
13
14 ENDCCLASS.
15
16
17
18② CLASS CL_FOR_TEST_DEMO IMPLEMENTATION.
19
20
21
22
23
24③ METHOD METHOD_WITH_VARIABLE.
25
26 ENDMETHOD.
27
28 ENDCCLASS.

```

Adding of the empty implementation part

## Related Information

- [Creating Method Definitions from Implementation Parts \[page 406\]](#)
- [Creating Implementation Parts for Unimplemented Methods \[page 409\]](#)
- [Creating Methods from Method Calls \[page 414\]](#)
- [Quick Assists \[page 85\]](#)
- [Quick Assist View \[page 91\]](#)

### 5.1.17.2.1.2 Creating Method Definitions from Implementation Parts

#### Prerequisites

In the ABAP class where you want to create the corresponding declaration in the definition part, the method must exist in the implementation part.

The ABAP class must be free of syntax errors other than those for the missing method definition(s).

## Context

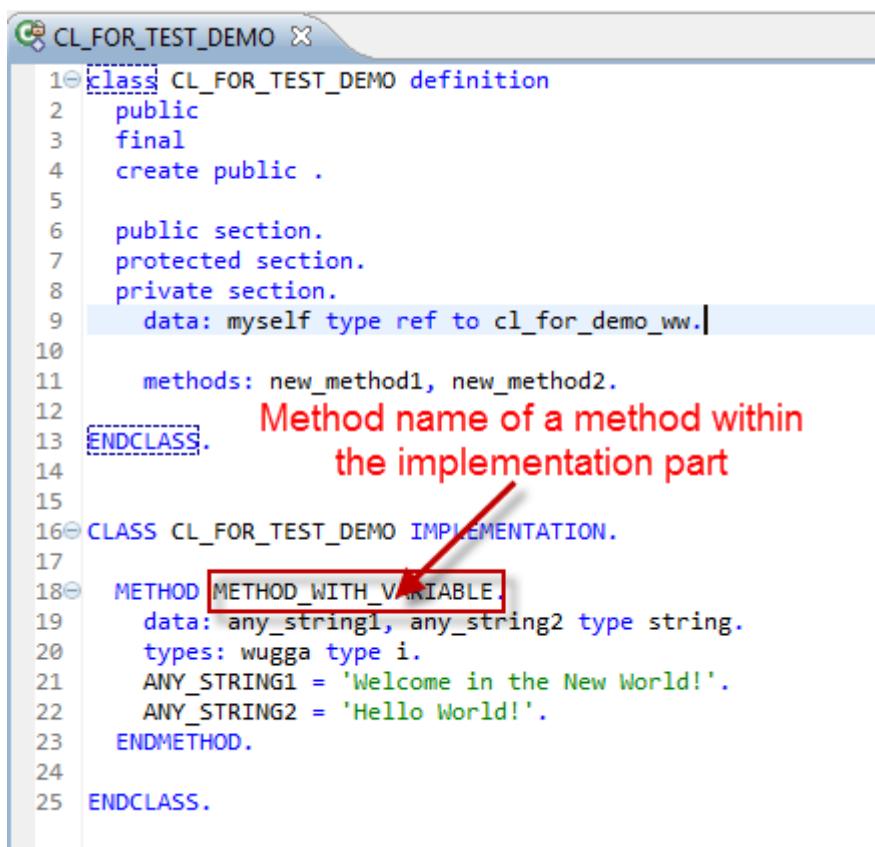
You can generate the declaration of a method in the definition part of an ABAP class directly out of the implementation part of the corresponding method.

### i Note

If you use the dialog and add parameters, the corresponding signature will be added in the definition part.

## Procedure

1. In the source code editor, double-click the method name in the implementation part that has no definition yet.



```
1① class CL_FOR_TEST_DEMO definition
2   public
3   final
4   create public .
5
6   public section.
7   protected section.
8   private section.
9     data: myself type ref to cl_for_demo_ww.| Method name of a method within the implementation part
10
11   methods: new_method1, new_method2.
12
13 ENDCLASS. Method name of a method within the implementation part
14
15
16② CLASS CL_FOR_TEST_DEMO IMPLEMENTATION.
17
18③ METHOD METHOD_WITH_VARIABLE.
19   data: any_string1, any_string2 type string.
20   types: wugga type i.
21   ANY_STRING1 = 'Welcome in the New World!'.
22   ANY_STRING2 = 'Hello World!'.
23 ENDMETHOD.
24
25 ENDCLASS.
```

Creating the method definition from the method name in the implementation

2. Choose **Quick Fix** (Ctrl 1) from the context menu.
3. In the *Quick Fix* dialog box, select the option relevant for continuing.

### i Note

You can use the following options:

- **Create definition 'METHOD\_WITH\_VARIABLE'** to create an instance method definition without parameters in the private section.

- **Create definition 'METHOD\_WITH\_VARIABLE' using wizard** to create a method via the dialog. Here, for example, you can add parameters or define the visibility of the method.
- **Create definition 'METHOD\_WITH\_VARIABLE' for testing** to create a test method. Note, this option is only available for test classes.

## Results

The source code editor inserts the definition from the method in the existing class implementation

```

1① class CL_FOR_TEST_DEMO definition
2   public
3   final
4   create public .
5
6   public section.
7   protected section.
8   private section.
9   data: myself type ref to cl_for_demo.
10  methods: METHOD_WITH_VARIABLE.           The method name is being added in the definition part.
11
12 ENDCCLASS.
13
14
15② CLASS CL_FOR_TEST_DEMO IMPLEMENTATION.
16
17③   METHOD METHOD_WITH_VARIABLE.
18     data: any_string1, any_string2 type string.
19     types: wugga type i.
20     ANY_STRING1 = 'Welcome in the New World!'.
21     ANY_STRING2 = 'Hello World!'.
22   ENDMETHOD.
23
24
25
26 ENDCCLASS.

```

Adding the method definition in the private section

## Related Information

- [Creating Method Implementations from the Method Definition \[page 404\]](#)
- [Creating Implementation Parts for Unimplemented Methods \[page 409\]](#)
- [Creating Methods from Method Calls \[page 414\]](#)
- [Quick Assists \[page 85\]](#)
- [Quick Assist View \[page 91\]](#)

### 5.1.17.2.1.3 Creating Implementation Parts for Unimplemented Methods

#### Prerequisites

- The name of the interface must be declared in the public section of the definition part of the class.
- The class must be free of syntax errors, other than those for the missing method implementation(s).

#### Context

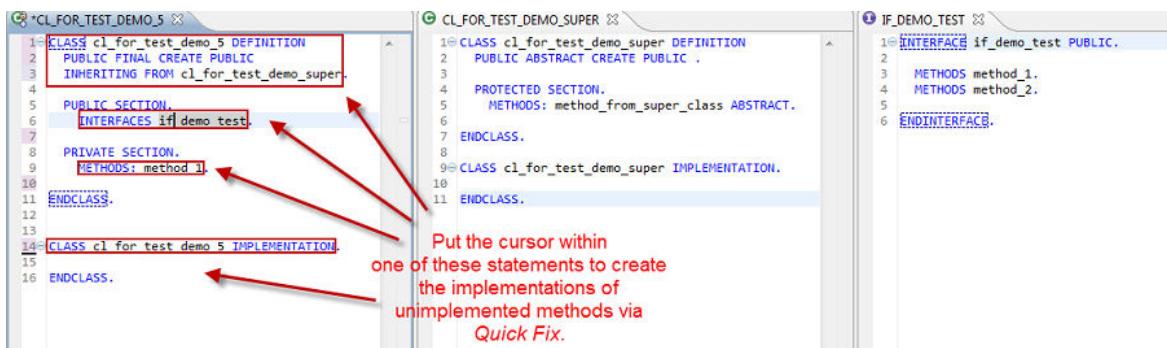
In an ABAP class, you can generate the implementation part of methods that are defined in an ABAP interface as well as method stubs of other unimplemented methods.

You can create method implementations in the following situations:

- Unimplemented interface methods
- Unimplemented abstract methods from super-classes
- Locally defined methods that are not yet implemented

#### Procedure

1. Position the cursor on one of the following statements within the class:
  - Class definition statement
  - Interface declaration that has no implementation part yet
  - Method definition of an unimplemented method
  - Class implementation statement



##### Creating method implementation stubs for unimplemented methods in a class

2. In the context menu, choose *Quick Fix* or use the shortcut (**Ctrl** + **1**).
3. In the Quick Fix dialog box, select *Add unimplemented methods*.

## Results

ABAP Development Tools generates stubs for all missing method implementations regardless of where the method is defined.

```
*CL_FOR_TEST_DEMO_5
1 CLASS cl_for_test_demo_5 DEFINITION
2   PUBLIC FINAL CREATE PUBLIC
3   INHERITING FROM cl_for_test_demo_super.
4
5   PUBLIC SECTION.
6     INTERFACES if_demo_test.
7   PROTECTED SECTION.
8     METHODS method_from_super_class REDEFINITION.
9
10  PRIVATE SECTION.
11    METHODS: method_1.
12
13  ENDCLASS.
14
15
16
17 CLASS cl_for_test_demo_5 IMPLEMENTATION.
18
19 METHOD method_1.
20
21 ENDMETHOD.
22
23
24 METHOD method_from_super_class.
25
26 ENDMETHOD.
27
28
29 METHOD if_demo_test~method_1.
30
31 ENDMETHOD.
32
33
34 METHOD if_demo_test~method_2.
35
36 ENDMETHOD.
37
38 ENDCLASS.
```

```
CL_FOR_TEST_DEMO_SUPER
1 CLASS cl_for_test_demo_super DEFINITION
2   PUBLIC ABSTRACT CREATE PUBLIC .
3
4   PROTECTED SECTION.
5     METHODS: method_from_super_class ABSTRACT.
6
7   ENDCLASS.
8
9 CLASS cl_for_test_demo_super IMPLEMENTATION.
10
11 ENDCLASS.
```

```
IF_DEMO_TEST
1 INTERFACE if_demo_test PUBLIC.
2
3 METHODS method_1.
4 METHODS method_2.
5
6 ENDINTERFACE.
```

1. The redefinition for the abstract method from the super class is created in the definition part.  
2. The empty method implementations are created in the implementation part.

Adding empty method stubs of unimplemented methods in a class

## Related Information

- [Creating Method Implementations from the Method Definition \[page 404\]](#)
- [Creating Method Definitions from Implementation Parts \[page 406\]](#)
- [Creating Methods from Method Calls \[page 414\]](#)
- [Quick Assists \[page 85\]](#)
- [Quick Assist View \[page 91\]](#)

## 5.1.17.2.1.3.1 Creation in SAP NetWeaver 7.3 EHP1 SP4-10

## Context

### Limitations

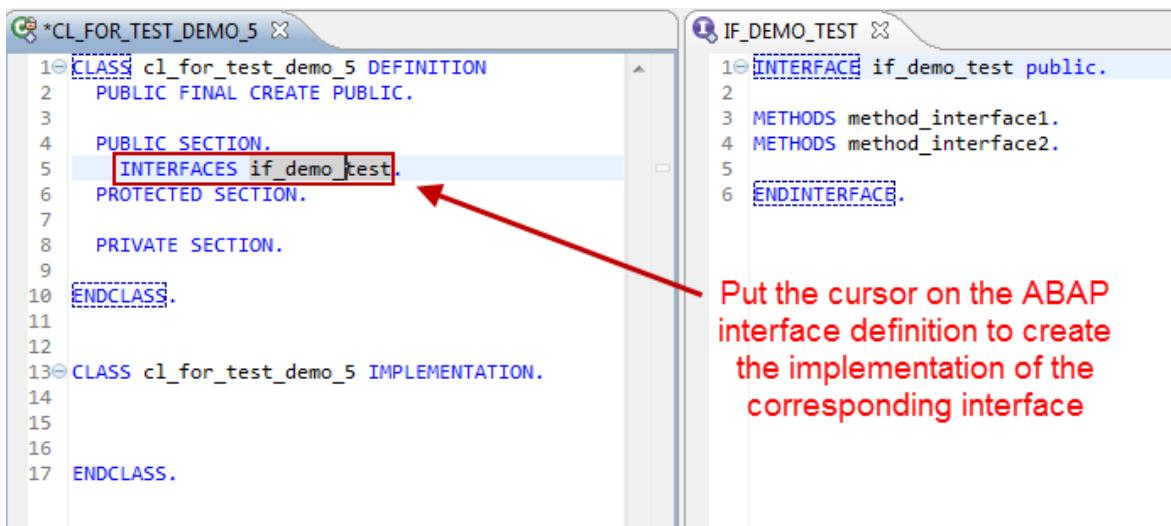
You can only create the implementations for unimplemented interface methods.

### i Note

In SAP NetWeaver 7.3 EHP1 SP11 and higher, the same behavior as in SAP NetWeaver 7.4 SP2 and subsequent releases is provided.

## Procedure

1. In the source code editor of the class, enter the statement with the name of the interface in the public section.
2. Position the cursor on the interface declaration that has no implementation part yet.



```
*CL_FOR_TEST_DEMO_5
1 CLASS cl_for_test_demo_5 DEFINITION
2   PUBLIC FINAL CREATE PUBLIC.
3
4   PUBLIC SECTION.
5   INTERFACES if_demo_test.
6   PROTECTED SECTION.
7
8   PRIVATE SECTION.
9
10  ENDCCLASS.
11
12
13 CLASS cl_for_test_demo_5 IMPLEMENTATION.
14
15
16
17 ENDCCLASS.
```

```
IF_DEMO_TEST
1 INTERFACE if_demo_test public.
2
3 METHODS method_interface1.
4 METHODS method_interface2.
5
6 ENDINTERFACE.
```

Put the cursor on the ABAP interface definition to create the implementation of the corresponding interface

Creating the implementation part of an interface in a class

3. In the context menu, choose **Quick Fix** or use the shortcut (**Ctrl** + **1**).
4. In the Quick Fix dialog box, select *Add unimplemented interface methods*.

## Results

ABAP Development Tools inserts the implementations of the missing interface methods in the existing implementation part of your class.

```

*CL_FOR_TEST_DEMO_5
1 CLASS cl_for_test_demo_5 DEFINITION
2   PUBLIC FINAL CREATE PUBLIC.
3
4   PUBLIC SECTION.
5     INTERFACES if_demo_test.
6   PROTECTED SECTION.
7
8   PRIVATE SECTION.
9
10  ENDCLASS.
11
12
13 CLASS cl_for_test_demo_5 IMPLEMENTATION.
14
15 METHOD if_demo_test~method_interface2.
16
17 ENDMETHOD.
18
19
20 METHOD if_demo_test~method_interface1.
21
22 ENDMETHOD.
23
24 ENDCLASS.

```

```

IF_DEMO_TEST
1 INTERFACE if_demo_test public.
2
3 METHODS method_interface1.
4 METHODS method_interface2.
5
6 ENDINTERFACE.

```

The empty method implementations of the ABAP interface are being created in the implementation part of the ABAP class.

Adding empty method stubs of an interface into the implementation part of a class

## Related Information

[Creating Implementation Parts for Unimplemented Methods \[page 409\]](#)  
[Creation in SAP NetWeaver 7.4 SP2 and Subsequent Releases \[page 412\]](#)

## 5.1.17.2.1.3.2 Creation in SAP NetWeaver 7.4 SP2 and Subsequent Releases

### Prerequisites

The class must be free of syntax errors, other than those for the missing method implementation.

### Context

You can create method implementations in the following situations:

- Unimplemented interface methods

- Unimplemented abstract methods from super-classes
- Locally defined methods that are not yet implemented

## Procedure

1. Position the cursor on one of the following statements within the class:

- Class definition statement
- Interface declaration that has no implementation part yet
- Method definition of an unimplemented method
- Class implementation statement

```

*CL_FOR_TEST_DEMO_5
1 CLASS cl_for_test_demo_5 DEFINITION
2 PUBLIC FINAL CREATE PUBLIC
3 INHERITING FROM cl_for_test_demo_super.
4
5 PUBLIC SECTION.
6  INTERFACES if_demo_test.
7
8 PRIVATE SECTION.
9  METHODS: method_1.
10
11 ENDCCLASS.
12
13
14 CLASS cl_for_test_demo_5 IMPLEMENTATION.
15
16 ENDCCLASS.

```

```

CL_FOR_TEST_DEMO_SUPER
1 CLASS cl_for_test_demo_super DEFINITION
2 PUBLIC ABSTRACT CREATE PUBLIC .
3
4 PROTECTED SECTION.
5  METHODS: method_from_super_class ABSTRACT.
6
7 ENDCCLASS.
8
9 CLASS cl_for_test_demo_super IMPLEMENTATION.
10
11 ENDCCLASS.

```

```

IF_DEMO_TEST
1 INTERFACE if_demo_test PUBLIC.
2
3 METHODS method_1.
4 METHODS method_2.
5
6 ENDINTERFACE.

```

Creating method implementation stubs for unimplemented methods in a class

2. In the context menu, choose *Quick Fix* or use the shortcut (**Ctrl** + **1**).
3. In the Quick Fix dialog box, select *Add unimplemented methods*.

## Results

ABAP Development Tools generates stubs for all missing method implementations regardless of where the method is defined.

```

*CL_FOR_TEST_DEMO_5
1 CLASS cl_for_test_demo_5 DEFINITION
2 PUBLIC FINAL CREATE PUBLIC
3 INHERITING FROM cl_for_test_demo_super.
4
5 PUBLIC SECTION.
6   INTERFACES if_demo_test.
7
8   PROTECTED SECTION.
9
10  METHODS method_from_super_class REDEFINITION.
11
12  PRIVATE SECTION.
13    METHODS: method_1.
14
15  ENDCLASS.
16
17 CLASS cl_for_test_demo_5 IMPLEMENTATION.
18
19 METHOD method_1.
20
21 ENDMETHOD.
22
23 METHOD method_from_super_class.
24
25 ENDMETHOD.
26
27
28 METHOD if_demo_test~method_1.
29
30 ENDMETHOD.
31
32
33 METHOD if_demo_test~method_2.
34
35 ENDMETHOD.
36
37
38 ENDCCLASS.

```

```

*CL_FOR_TEST_DEMO_SUPER
1 CLASS cl_for_test_demo_super DEFINITION
2 PUBLIC ABSTRACT CREATE PUBLIC .
3
4 PROTECTED SECTION.
5   METHODS: method_from_super_class ABSTRACT.
6
7 ENDCLASS.
8
9 CLASS cl_for_test_demo_super IMPLEMENTATION.
10
11 ENDCCLASS.

```

```

*IF_DEMO_TEST
1 INTERFACE if_demo_test PUBLIC.
2
3 METHODS method_1.
4 METHODS method_2.
5
6 ENDINTERFACE.

```

1. The redefinition for the abstract method from the super class is created in the definition part.
2. The empty method implementations are created in the implementation part.

Adding empty method stubs of unimplemented methods in a class

## Related Information

[Creation in SAP NetWeaver 7.3 EHP1 SP4-10 \[page 410\]](#)

[Creating Implementation Parts for Unimplemented Methods \[page 409\]](#)

### 5.1.17.2.1.4 Creating Methods from Method Calls

#### Prerequisites

Your code must be free from syntax errors.

#### Context

In an ABAP class, you can generate the method definition from the method call in the implementation part.

##### i Note

You can also create the method definition if the method is used in an ABAP class other than the one you are now working in.

## Procedure

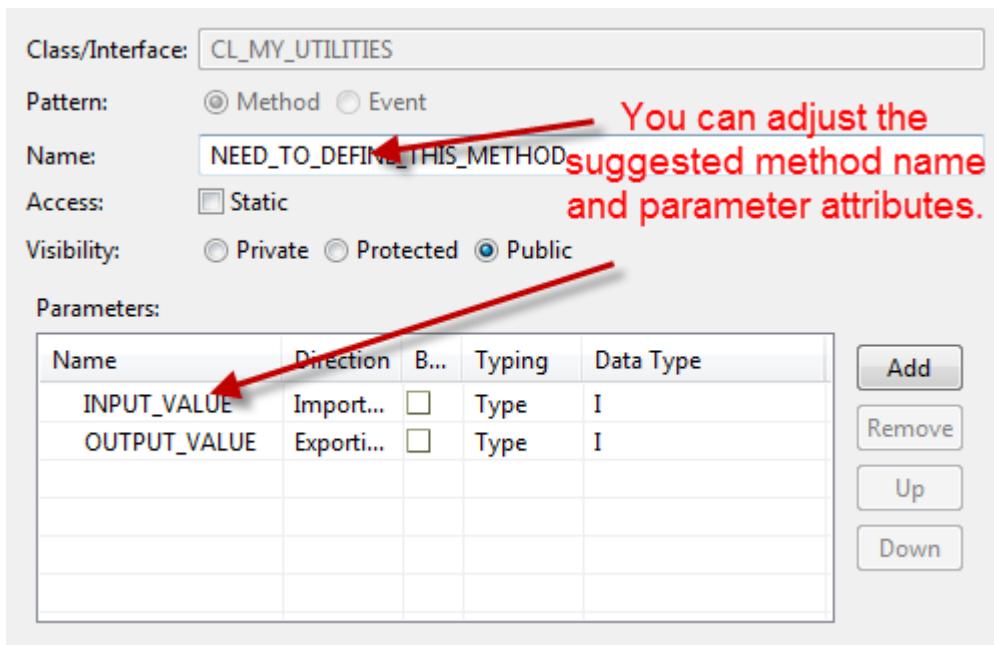
1. In the source code editor of the ABAP class where the method is implemented, position the cursor on a method call to create the definition in another ABAP class.

```
CL_MY_UTILITIES
1 class CL_MY_UTILITIES definition
2   public
3   final
4   create public .
5
6   public section.
7   protected section.
8   private section.
9  ENDCLASS.
10
11
12
13 CLASS CL_MY_UTILITIES IMPLEMENTATION.
14
15
16 ENDCCLASS.
```

```
CL_THIS_IS_GREAT
1 class CL_THIS_IS_GREAT definition
2   public
3   final
4   create public .
5
6   public section.
7   protected section.
8   private section.
9   METHODS GREATEST_METHOD_EVER.
10 ENDCCLASS.
11
12
13
14 CLASS CL_THIS_IS_GREAT IMPLEMENTATION.
15
16 METHOD Greatest_method_ever.
17   DATA: my_helper type ref to cl_my_utilities,
18         value_to_process type i,
19         processed_value type i.
20
21   my_helper->need_to_define_this_method
22     exporting
23       input_value = value_to_process
24     importing
25       output_value = processed_value .
26
27
28 ENDMETHOD.
29 ENDCCLASS.
```

Creating the method definition from the method call in another class

2. Choose **Quick Fix** (Ctrl 1) from the context menu.
3. In the *Quick Fix* dialog box, double-click **Create method**.
4. In the dialog, you can change a suggested method name as well as the suggested name and attributes of a method parameter.



Adding parameters that are to be declared in the signature of the method definition

5. In the dialog box, click **Finish**.

## Results

The source code editor inserts the definition of the method in the public section of the existing ABAP class – here with `IMPORTING` and `EXPORTING` parameters – using the wizard, and it creates the empty method in the implementation.

```

*CL_MY_UTILITIES
1 class CL_MY_UTILITIES definition
2   public
3   final
4   create public .
5
6   public section.
7     METHODS NEED_TO_DEFINE_THIS_METHOD
8       IMPORTING
9         INPUT_VALUE TYPE I
10      EXPORTING
11        OUTPUT_VALUE TYPE I.
12   protected section.
13   private section.
14 endclass.

18 CLASS CL_MY_UTILITIES IMPLEMENTATION.
19
20
21
22 METHOD NEED_TO_DEFINE_THIS_METHOD.
23
24 ENDMETHOD.
25
26 endclass.

CL_THIS_IS_GREAT
1 class CL_THIS_IS_GREAT definition
2   public
3   final
4   create public .
5
6   public section.
7   protected section.
8   private section.
9     METHODS GREATEST_METHOD_EVER.
10    ENDMETHOD.
11
12
13
14 CLASS CL_THIS_IS_GREAT IMPLEMENTATION.
15
16 METHOD Greatest_method_ever.
17   DATA: my_helper type ref to cl_my_utilities,
18         value_to_process type i,
19         processed_value type i.
20
21   my_helper->need_to_define_this_method(
22     exporting
23       input_value = value_to_process
24     importing
25       output_value = processed_value ).
26
27
28 ENDMETHOD.
29

```

Adding the method definition in the public section and creating the empty implementation part in another ABAP class

## Related Information

[Creating Method Implementations from the Method Definition](#) [page 404]

[Creating Method Definitions from Implementation Parts](#) [page 406]

[Creating Implementation Parts for Unimplemented Methods](#) [page 409]

[Quick Assists](#) [page 85]

[Quick Assist View](#) [page 91]

## 5.1.17.2.2 Creating ABAP Classes or ABAP Interfaces from Usage

### Prerequisites

In order to make this option available in the *Quick Fix* dialog, note the following prerequisites:

- The definition statement must be completed.
- No class or interface with the given name must exist in the ABAP system.

## Context

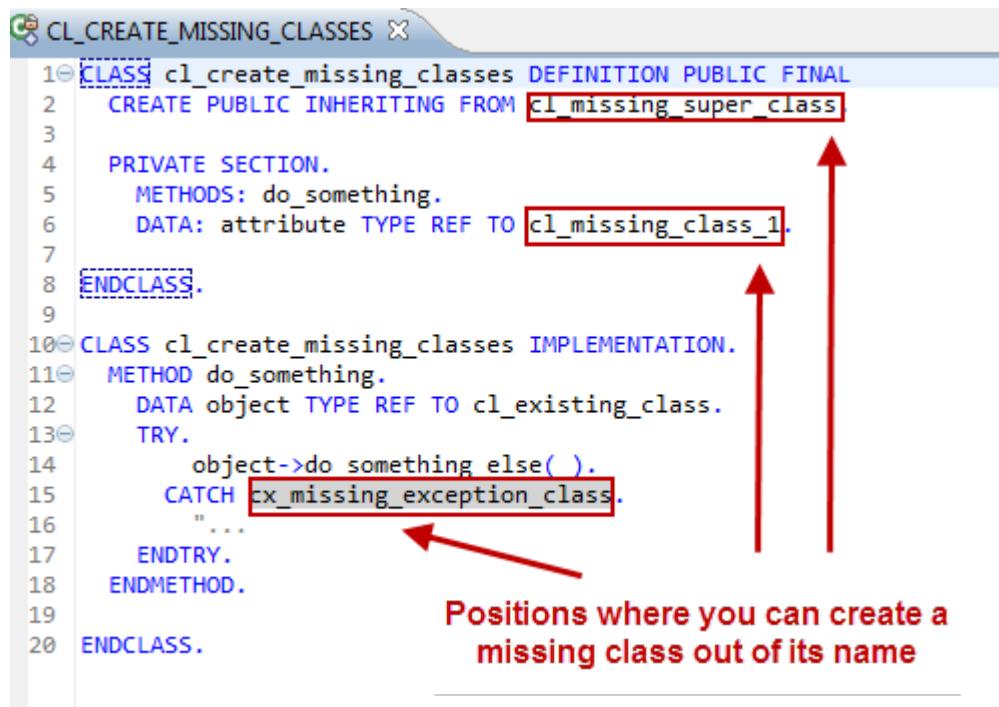
In the source code of an ABAP program, you can create a local and global ABAP class / interface directly from the name of the missing class or interface.

### ABAP Classes

ABAP classes can be generated from the following statements:

- Definition statements, where the `type ref to` addition is used to specify a variable, constant type, or method parameter
- Catch statements
- Raise exception statements
- Class definition statements within the `inheriting from` addition
- Create object statements with a `type` addition

Example



```
1 CLASS cl_create_missing_classes DEFINITION PUBLIC FINAL
2   CREATE PUBLIC INHERITING FROM cl_missing_super_class.
3
4   PRIVATE SECTION.
5     METHODS: do_something.
6     DATA: attribute TYPE REF TO cl_missing_class_1.
7
8   ENDCLASS.
9
10 CLASS cl_create_missing_classes IMPLEMENTATION.
11   METHOD do_something.
12     DATA object TYPE REF TO cl_existing_class.
13     TRY.
14       object->do_something_else( ).
15     CATCH cx_missing_exception_class.
16     " ...
17   ENDTRY.
18 ENDMETHOD.
19
20 ENDCCLASS.
```

Positions where you can create a missing class out of its name

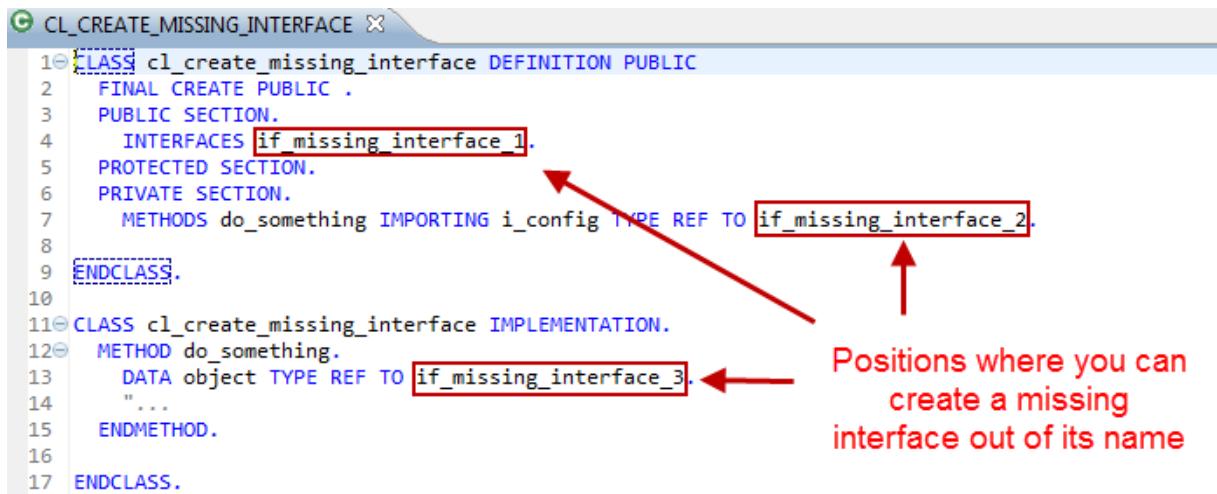
Example of an ABAP class that contains references to missing classes

### ABAP Interfaces

ABAP interfaces can be generated from the following statements:

- Definition statements, where the `type ref to` addition is used to specify a variable, constant type, or method parameter
- Interface implementation statements in the public section of a class

Example



```

1 CLASS cl_create_missing_interface DEFINITION PUBLIC
2   FINAL CREATE PUBLIC .
3   PUBLIC SECTION.
4     INTERFACES if_missing_interface_1.
5   PROTECTED SECTION.
6     PRIVATE SECTION.
7     METHODS do_something IMPORTING i_config TYPE REF TO if_missing_interface_2.
8
9   ENDCCLASS.
10
11 CLASS cl_create_missing_interface IMPLEMENTATION.
12   METHOD do_something.
13     DATA object TYPE REF TO if_missing_interface_3.
14     ...
15   ENDMETHOD.
16
17 ENDCCLASS.

```

Positions where you can create a missing interface out of its name

Example of an ABAP class that contains references to missing interfaces

## Procedure

1. In the source code editor, position the cursor on the corresponding class or interface name for which you want to create a class or interface.
2. In the context menu, choose **Quick Fix** or use the shortcut (Ctrl + 1).
3. In the *Quick Fix* dialog box, double-click the required entry.

### i Note

The entries are displayed in reference to the element that you have selected.

The following entries are provided:

- Create global class
- Create global interface

## Results

If you create a global class or interface, the corresponding creation wizard is opened. Here you define further creation details.

## Related Information

[Creating Definitions and Implementations of Methods \[page 404\]](#)

[Creating ABAP Includes from Usage \[page 421\]](#)

[Creating ABAP Function Modules from Usage \[page 420\]](#)

[Creating ABAP Classes \[page 185\]](#)

[Quick Assists \[page 85\]](#)

## 5.1.17.2.3 Creating ABAP Function Modules from Usage

### Context

You can create an ABAP function module that does not exist yet, based on the `CALL FUNCTION` statement.

#### i Note

- Parameter types are not considered
- The quick fix is only proposed if the name of the function module is set in hyphens.

#### Example

The function module `NEW_FUNCTION_MODULE` is called in an ABAP class as follows:

```
CALL FUNCTION 'NEW_FUNCTION_MODULE'.
```

If the function module does not exist yet, you can create it using a quick fix.

### Procedure

1. In the source code editor, position the cursor on the corresponding function module name you want to create.
2. In the context menu, choose **Quick Fix** or use the shortcut (Ctrl + 1).
3. In the *Quick Fix* dialog box, double-click **Create function module**.

### Results

The creation wizard is opened. Here you enter the data for creating the missing function module.

### Related Information

[Creating Definitions and Implementations of Methods \[page 404\]](#)

[Creating ABAP Classes or ABAP Interfaces from Usage \[page 417\]](#)

[Creating ABAP Includes from Usage \[page 421\]](#)

[Creating an ABAP Function Module \[page 202\]](#)

[Quick Assists \[page 85\]](#)

## 5.1.17.2.4 Creating ABAP Includes from Usage

### Context

In the source code of an ABAP Include, you can start the creation wizard of an include directly from the name of its call.

Example

An include that does not exist yet is used.

For this purpose, the `demo_include` include is created:

```
REPORT demo.
INCLUDE demo_include.
```

### Procedure

1. In the source code editor, position the cursor on the corresponding include name you want to create.
2. In the context menu, choose **Quick Fix** or use the shortcut (Ctrl + 1).
3. In the *Quick Fix* dialog box, double-click **Create include**.

### Results

The creation wizard is opened. Here you enter the data in order to create the missing include.

### Related Information

[Creating Definitions and Implementations of Methods \[page 404\]](#)

[Creating ABAP Classes or ABAP Interfaces from Usage \[page 417\]](#)

[Creating ABAP Function Modules from Usage \[page 420\]](#)

[Creating ABAP Includes from Usage \[page 421\]](#)

[Creating an ABAP Function Group Include \[page 199\]](#)

[Creating ABAP Classes \[page 185\]](#)

[Quick Assists \[page 85\]](#)

[Quick Assist View \[page 91\]](#)

## 5.1.17.3 Applying Other Quick Assists

### Context

In addition to refactorings and quick fixes, the following other quick assists are provided:

Function	Description
<a href="#">Generating Class Constructor Methods [page 423]</a>	Creating an empty <code>class constructor</code> method in the public section of the current ABAP class.
<a href="#">Generating Constructor Methods [page 424]</a>	Creating a constructor in the public section of the current ABAP class.
	<p><b>i Note</b></p> <p>If the class has attributes, a dialog is opened where you can select the attributes that should be instantiated by the constructor.</p>
<a href="#">Generating Factory Methods [page 425]</a>	Creating a static <code>create</code> method in the public section of the current ABAP class.
	<p><b>i Note</b></p> <p>If the class has attributes, a dialog is opened where you can select the attributes that should be instantiated by the factory method.</p>
<a href="#">Creating a Text Symbol in the Text Pool [page 431]</a>	Creating a text symbol in the text pool of an ABAP program.
<a href="#">Editing Text Symbols [page 434]</a>	Changing a text symbol in the text pool of an ABAP program.
<a href="#">Correcting Inconsistencies Within Text Symbols [page 435]</a>	Balancing mismatches between source code and text pool.
<a href="#">Switching Notations [page 436]</a>	Adopting text changes of an existing text symbol in the text pool or the source code and vice versa.
<a href="#">Generating Getters and Setters [page 427]</a>	
<a href="#">Regenerating a Constructor for Exception Classes [page 428]</a>	

You have the following options for executing the generating functions:

- Entry   in the menu bar
- Entry **Quick Fix** in the context menu
- Entry in *Quick Assist* view
- Shortcut **Ctrl 1**

### Related Information

[Refactoring ABAP Source Code \[page 342\]](#)

[Applying ABAP Quick Fixes \[page 402\]](#)

[Quick Assists \[page 85\]](#)

[Quick Assist View \[page 91\]](#)

## 5.1.17.3.1 Generating Class Constructor Methods

### Context

You can create a public method called `class_constructor` with an empty implementation.

Example

Here, the `class_constructor` method is added:

```
CLASS cl_car_constructor_generator DEFINITION PUBLIC CREATE PUBLIC .  
  PUBLIC SECTION.  
ENDCLASS.  
  
CLASS cl_car_constructor_generator IMPLEMENTATION.  
ENDCLASS.
```

### Procedure

1. In the definition part, select the name of the class.
2. In the context menu, choose **Quick Fix** or use the shortcut (Ctrl 1).
3. In the *Quick Fix* dialog box, double-click **Generate class constructor**.

### Results

In the definition, the `class_constructor` method is declared as **CLASS-METHODS**.

In the implementation, the empty body of the `class_constructor` method is added.

Example

ABAP class after generating

```
CLASS cl_car_constructor_generator DEFINITION PUBLIC CREATE PUBLIC .  
  PUBLIC SECTION.  
  CLASS-METHODS class_constructor.  
ENDCLASS.  
  
CLASS cl_car_constructor_generator IMPLEMENTATION.  
  METHOD class_constructor.  
  ENDMETHOD.  
ENDCLASS.
```

## Related Information

[Generating Constructor Methods \[page 424\]](#)

[Generating Factory Methods \[page 425\]](#)

[Applying Other Quick Assists \[page 422\]](#)

[Outline View \[page 80\]](#)

[Quick Assist View \[page 91\]](#)

### 5.1.17.3.2 Generating Constructor Methods

#### Context

From the class name in the definition of an ABAP class, you can create a public instance method called `constructor`.

If the class has attributes, a dialog is opened where you can select the attributes that should be instantiated by the constructor. Afterwards, the constructor is created with the selected parameters.

Example

The `car` class has the `name` attribute:

```
CLASS cl_car_constructor_generator DEFINITION PUBLIC CREATE PUBLIC .  
  PUBLIC SECTION.  
ENDCLASS.  
  
CLASS cl_car_constructor_generator IMPLEMENTATION.  
ENDCLASS.
```

#### Procedure

1. In the definition part, select the name of the class.
2. In the context menu, choose **Quick Fix** or use the shortcut (Ctrl 1).
3. In the *Quick Fix* dialog box, double-click **Generate constructor**.
4. A dialog is opened where you can select the attributes which should be initialized by the `constructor`. Select the desired attributes and confirm with **OK**.

#### Results

In the definition, the `constructor` method is added. For each selected attribute, the corresponding parameter is created.

In the implementation part, each parameter is assigned to the corresponding attribute.

## Example

ABAP class after generating

```
CLASS cl_car_constructor_generator DEFINITION PUBLIC CREATE PUBLIC .
  PUBLIC SECTION.
    CLASS-METHODS class_constructor.
  ENDCLASS.

  CLASS cl_car_constructor_generator IMPLEMENTATION.
    METHOD class_constructor.
  ENDMETHOD.
  ENDCLASS.
```

## Related Information

[Generating Class Constructor Methods \[page 423\]](#)

[Generating Factory Methods \[page 425\]](#)

[Quick Assists \[page 85\]](#)

[Quick Assist View \[page 91\]](#)

### 5.1.17.3.3 Generating Factory Methods

#### Context

In the definition of an ABAP class, you can add the `static create` method from the class name. If the class has attributes, a dialog is opened where you can select the attributes that should be instantiated by the create method. Then, in the public section, the definition of this method is added. In addition, an importing as well as a returning parameter that refers to the same class are added.

#### Example

The `car` class has the `name` attribute:

```
CLASS cl_car DEFINITION.
  PUBLIC SECTION.
  PRIVATE SECTION.
    DATA name TYPE string.
  ENDCLASS.

  CLASS cl_car IMPLEMENTATION.
  ENDCLASS.
```

#### Procedure

1. In the definition part, select the name of the class.

2. In the context menu, choose **Quick Fix** or use the shortcut (Ctrl + 1).
3. In the *Quick Fix* dialog box, double-click **Generate factory method**.

## Results

In the definition part, the `create` method is added. For each selected attribute, the corresponding `r_result` returning parameter and `i_current_speed` importing parameter is created.

In the implementation part, the body of the `create` class-method is added. Here, the `r_result` returning parameter is created and assigned to the `i_name` importing parameter.

Example

ABAP class after generating

```
CLASS cl_car DEFINITION.
  PUBLIC SECTION.
    CLASS-METHODS create
    IMPORTING
      i_name TYPE string.
    RETURNING
      VALUE(r_result) TYPE REF TO cl_car.
  PRIVATE SECTION.
    DATA name TYPE string.
ENDCLASS.

CLASS cl_car IMPLEMENTATION.
  METHOD create
    CREATE OBJECT r_result.
    r_result->name = i_name.
  ENDMETHOD.
ENDCLASS.
```

## Related Information

[Generating Class Constructor Methods \[page 423\]](#)

[Generating Constructor Methods \[page 424\]](#)

[Quick Assists \[page 85\]](#)

[Quick Assist View \[page 91\]](#)

## 5.1.17.3.4 Generating Getters and Setters

You can generate the getter and/or setter method stubs from an attribute in the definition of an ABAP class in the implementation. In addition, the corresponding importing and/or returning parameter(s) are set.

### Context

Example

Before Execution	After Execution
<pre>01 CLASS cl_car DEFINITION. 02   PUBLIC SECTION. 03   PRIVATE SECTION. 04     DATA: speed TYPE i. 05   ENDCLASS. 06 07 CLASS cl_car IMPLEMENTATION. 08   ... 09 ENDCLASS.</pre>	<pre>01 CLASS cl_car DEFINITION. 02   PUBLIC SECTION. 03   METHODS: 04     get_speed 05       RETURNING 06         VALUE(r_speed) TYPE i, 07       set_speed 08         IMPORTING 09           i_speed TYPE i. 10   PRIVATE SECTION. 11   DATA: speed TYPE i. 12 ENDCLASS. 13 14 CLASS cl_car IMPLEMENTATION. 15   ... 16   METHOD get_speed. 17     r_speed = speed. 18   ENDMETHOD. 19   METHOD set_speed. 20     speed = i_speed. 21   ENDMETHOD. 22 ENDCLASS.</pre>

To copy the source code example, click here [page 428]

In the local `cl_car` class of an ABAP program, the `speed` attribute is defined. For the latter, the getter and setter methods and the corresponding returning and importing parameters need to be generated. The reason for this is because their value should be accessed and encapsulated.

In the definition part of the `cl_car` class, the `get_speed` and `set_speed` methods are added. For the `get_speed` method, the `r_speed` returning parameter is set and for the `set_speed` method the `i_speed` importing parameter is set.

In the implementation part, the corresponding method stubs and their returning and importing parameters are generated on the basis of the selected attribute.

## Procedure

1. In the definition part, select the variable for which you want to create the getter and setter methods.
2. Get the quick assist proposals by choosing **Quick Fix** from the context menu (shortcut **Ctrl 1**) or by using the *Quick Assist* view.
3. Apply one of the following options.
  - *Generate Getter and Setter for speed* to add the `get_speed` and `set_speed` method stubs as well as its `i_speed` importing and `r_speed` returning parameters. In addition, the corresponding definitions are also added.
  - *Generate Getter for speed* to add the `get_speed` method stub where the `r_speed` returning parameter is set. The corresponding definitions are also added.
  - *Generate Setter for speed* to add the `set_speed` method stub where the `i_speed` importing parameter is set. The corresponding definitions are also added.

## Related Information

[Quick Assists \[page 85\]](#)

### 5.1.17.3.4.1 Code Example Before Execution

```
CLASS cl_car DEFINITION.  
  PUBLIC SECTION.  
  PRIVATE SECTION.  
  DATA: speed TYPE i.  
ENDCLASS.  
  
CLASS cl_car IMPLEMENTATION.  
  ...  
ENDCLASS.
```

### 5.1.17.3.5 Regenerating a Constructor for Exception Classes

Exception classes have special constructor methods. If they implement the `IF_T100_MESSAGE` interface, the signature and the implementation have a special logic.

## Context

You should regenerate the constructor whenever you add or remove:

- the interface `IF_T100_MESSAGE`, or

- public or protected attributes.

## • Example

### Before Execution

```

01 CLASS cx_speed_error DEFINITION
02   INHERITING FROM cx_static_check.
03   PUBLIC SECTION.
04     INTERFACES if_t100_message.
05     METHODS constructor
06       textid LIKE textid OPTIONAL
07       previous LIKE previous OPTIONAL.
08     DATA: speed TYPE i.
09   ENDCLASS.
10
11 CLASS cx_speed_error IMPLEMENTATION.
12   METHOD constructor.
13     super->constructor(
14       textid = textid
15       previous = previous ).
16   ENDMETHOD.
17 ENDCLASS.

```

### After Execution

```

01 CLASS cx_speed_error DEFINITION
02   INHERITING FROM cx_static_check.
03   PUBLIC SECTION.
04     INTERFACES if_t100_message.
05     METHODS constructor
06       textid LIKE textid OPTIONAL
07       textid LIKE *=>t100key OPTIONAL
08       previous LIKE previous OPTIONAL.
09     DATA: speed TYPE i OPTIONAL.
10
11 ENDCLASS.
12
13 CLASS cx_speed_error IMPLEMENTATION.
14   METHOD constructor.
15     super->constructor(
16       textid = textid
17       previous = previous ).
18     me->speed = speed.
19     CLEAR me->textid.
20     IF textid IS INITIAL.
21       *~t100key = *=>default_textid.
22     ELSE.
23       *~t100key = textid.
24   ENDIF.
25
26 ENDCLASS.

```

To copy the source code example, click here [page 430]

You have added the interface `IF_T100_MESSAGE` and an additional attribute `SPEED` to an exception class. Therefore, the constructor is no longer up to date.

The signature and the implementation of the constructor have been modified in accordance with `T100` semantics and the new attribute.

\* Note that the asterisk stands as a placeholder for the interface `IF_T100_MESSAGE`.

## Procedure

1. In the definition or the implementation, select the name of the exception class or the constructor method for which you want to regenerate the constructor.
2. Get the quick assist proposals by choosing **Quick Fix** from the context menu (shortcut `Ctrl 1`) or by using the *Quick Assist* view.
3. Apply *Regenerate constructor*.

## Related Information

[Quick Assists \[page 85\]](#)

### 5.1.17.3.5.1 Code Example Before Execution

```
CLASS cx_speed_error DEFINITION
  INHERITING FROM cx_static_check.
  PUBLIC SECTION.
    INTERFACES if_t100_message.
    METHODS constructor
      IMPORTING
        textid LIKE textid OPTIONAL
        previous LIKE previous OPTIONAL.
    DATA: speed TYPE i.
  ENDCLASS.

  CLASS cx_speed_error IMPLEMENTATION.
    METHOD constructor.
      super->constructor(
        textid = textid
        previous = previous ).
    ENDMETHOD.
  ENDCLASS.
```

### 5.1.17.3.6 Creating and Editing Text Symbols (Quick Assists)

#### Context

Using a text symbol makes a development object translateable and easier to maintain.

You can use various quick assits to create and edit text symbols directly in the source code editor.

#### • Example

You can use text symbols in the following use cases:

- WRITE statement: `WRITE:/ text-001.`
- Variable assignment: `greeting = 'hello'(B01).`
- Method call: `print ('hello').`

The following other quick assists are available for working with text symbols:

- [Creating a Text Symbol in the Text Pool \[page 431\]](#)
- [Editing Text Symbols \[page 434\]](#)
- [Correcting Inconsistencies Within Text Symbols \[page 435\]](#)

- [Switching Notations \[page 436\]](#)

The table below shows the available quick assists, depending on the way a text is addressed in the source code and on whether a corresponding text symbol already exists in the text pool.

Precondition Checks for Providing Other Quick Assists

Source code provided	Text symbol does not exist	Text symbol already exists
text-001	Create	Edit Add_literal => 'hello' (001)
'hello' (001)	Create	Edit => 'goodbye' (001) Align source code with text pool Remove_literal => text-001
'hello'	Create => 'hello' (001)	X
'HELLO'	No proposal for texts in all uppercase	X

**i Note**

Text in uppercase is not considered for translation. In this case, no quick assist is provided.

#### Limitation

Text symbols are not considered whenever you copy a development object.

### 5.1.17.3.6.1 Creating a Text Symbol in the Text Pool

#### Context

In your program, you want to use a text that is translatable.

**❖ Example**

You want to create a text symbol from the following write statement:

```
WRITE text-001.
```

#### Procedure

1. In the source code editor, write `text-xxxx` where `xxx` is the placeholder for the text key of your text symbol.

### i Note

The text key has three characters that can be letters and / or numbers. However, it must not start with %\_ or contain empty spaces.

2. Position the cursor somewhere on `text-xxx`.
3. In the context menu, choose *Quick Fix* or use the shortcut (`Ctrl` + `1`).

### i Note

Text in uppercase is not considered for translation. In this case, no quick assist is provided.

4. In the *Quick Fix* dialog box, double-click *Create text 001 in the text pool*.

The *New Text Symbol* creation wizard is opened.

### i Note

Following entries are write-protected:

- ADT automatically derives the name of the *Text Pool* from the name of the development object you are currently working on.
- The value of the *Original Language* is derived from the original language of the ABAP program.
- The **Text Key** identifies the text symbol in the text pool.

5. In the *Text Content* input field, add the information you want to display in the program.

### i Note

You can enter a maximum of 132 characters.

6. Enter the *Maximum Length* for the text content to be translated. You can also use the slider in order to determine the number for the *Maximum Length*.

### i Note

Text translations in other languages may need more space due to an increased number of characters.

In order to provide the required space for such translations, the following automatic rules are defined in ADT:

- If the actual length is less than 20 characters, ADT adds 10 characters to the actual length.
- If the actual length is greater than or equal to 20 characters, ADT doubles the number of the actual length.

7. Choose *Next*.
8. Assign a transport request.
  - Select the *Choose from requests in which I am involved* radio button to pick an already existing transport request.
  - Select the *Create a new request* radio button in order to generate a new transport request. In the *Request description* field, type in further information.
  - Select the *Enter a request number* in order to add your class to an existing request. Choose the *Browse...* button if you want to select a request that has already been created by a certain user.
9. Start the creation with **Finish**.

## Results

The text symbol is created in the text pool and activated.

Example

The write statement is the same as it was before execution because the text content has been added in the creation wizard.

In the *Quick Assist* view, the following quick assists are provided:

- **Add literal for text** to change the notation from `text-xxx` to 'hello' (xxx)
- **Edit text** to change the text of the text symbol

### → Tip

This procedure is also provided for similar use cases. In addition, take a look at the following differences:

#### Comparing Similar Use Cases for Creating Text Symbols

Description	Code Example before Execution	Code Example after Execution	Procedure	Differences
You can write a text followed by a certain text key enclosed in brackets.	<pre>greeting = 'hello'(B01).</pre>	<pre>greeting = 'hello'(B01).</pre>	Use quick assists to create or edit the text symbol.	<ul style="list-style-type: none"><li>• In the Text Content input field, the given text 'hello' is already added.</li></ul>
You can write a text without providing any text key.	<pre>print( 'Page was printed!' ).</pre>	<pre>print( 'Page was printed!' (100) ).</pre>	Use the quick assist to create the text symbol.	<ul style="list-style-type: none"><li>• ADT automatically uses a text key that has not been used so far.</li><li>• In the Text Content input field, the given text 'Page was printed!' is already added.</li></ul>

### i Note

If the same already exists as a text symbol, you will get a quick assist proposal to reuse the text symbol.

## Related Information

[Creating and Editing Text Symbols \(Quick Assists\) \[page 430\]](#)

[Editing Text Symbols \[page 434\]](#)

[Correcting Inconsistencies Within Text Symbols \[page 435\]](#)

[Switching Notations \[page 436\]](#)

## 5.1.17.3.6.2 Editing Text Symbols

### Prerequisites

The text symbol exists in the text pool.

### Context

You can edit text symbols in order to modify the text content or the maximum length.

Example

You want to update the text content of a text symbol:

```
greeting = 'hello' (B01) .
```

### Procedure

1. In the source code, select the text content.
2. In the context menu, choose *Quick Fix* or use the shortcut (ctrl 1).
3. In the *Quick Fix* dialog box, double-click *Edit text B01 in text pool*.  
The *Edit Text Symbol* wizard is opened.
4. Change the text symbol and choose *Next*.
5. Assign a transport request.
6. Choose *Finish* to complete editing.

### Results

The text symbol is adopted in the text tool and activated.

Example

In the source code editor, the text content is updated:

```
greeting = 'good morning' (B01) .
```

## Related Information

[Creating and Editing Text Symbols \(Quick Assists\) \[page 430\]](#)

[Creating a Text Symbol in the Text Pool \[page 431\]](#)

[Correcting Inconsistencies Within Text Symbols \[page 435\]](#)

[Switching Notations \[page 436\]](#)

[Adding Code Templates to an ABAP Exception Class \[page 190\]](#)

### 5.1.17.3.6.3 Correcting Inconsistencies Within Text Symbols

#### Prerequisites

The text in the source code differs from the text stored in the text pool.

#### Context

You can automatically correct inconsistencies of an existing text symbol in the text pool or the source code editor.

Example

Here, the text in the source code is 'hello', but the text pool contains 'goodbye' for the text key '001'.

```
REPORT adt_text_symbols.  
WRITE:/ 'hello'(001).
```

#### Procedure

1. Position the cursor on the corresponding text or the text key.
2. In the context menu, choose *Quick Fix* or use the shortcut (**Ctrl** + **1**).
3. In the *Quick Fix* dialog box, you can select the following quick fixes:
  1. **<- Replace the literal by text 001** to substitute the literal in the source code with the text content of the text pool
  2. **-> Replace text 001 in the text pool with the literal** to substitute the text content of the text pool with the literal from the source code.

#### **i** Note

For this quick assist, an editor dialog is opened to verify or adjust the text that will be stored in the text pool.

## Results

### Example

Depending on your selection, the following results are achieved:

1. ABAP programm if you have replaced the literal by a text key from the text pool:  
Here, the existing text content is updated with the content provided by the text tool.

```
REPORT adt_text_symbols.  
WRITE:/ 'goodbye'(001).
```

2. ABAP programm if you have replaced the existing text content of the text pool:  
Here, the text content remains in the source code editor. The text content in the text pool is updated.

```
REPORT adt_text_symbols.  
WRITE:/ 'hello'(001).
```

### → Tip

If the texts are consistent and you want to change them, you can just edit the text in the source code and use the quick assits to adjust the text in the text pool without opening the wizard.

## Related Information

[Creating and Editing Text Symbols \(Quick Assists\) \[page 430\]](#)

[Creating a Text Symbol in the Text Pool \[page 431\]](#)

[Editing Text Symbols \[page 434\]](#)

[Switching Notations \[page 436\]](#)

## 5.1.17.3.6.4 Switching Notations

### Context

There are two ways to switch notations in the source code:

1. You want to remove the fully qualified literal of a text symbol to display a notation.
2. You want to add the fully qualified literal of a text symbol without displaying a notation.

### • Example

In the source code, you want to replace the text content of a text symbol with a notation.

```
REPORT adt_text_symbols.  
WRITE:/ 'This text needs to be replaced with another text ID'(001).
```

## Procedure

1. In the source code, select the literal or the text key in brackets.
2. In the context menu, choose *Quick Fix* or use the shortcut (`Ctrl` + `1`).
3. In the *Quick Fix* dialog box, select *Remove literal for text 001* to replace the text content in the source code editor with a notation.  
The *Preview Changes* dialog is opened.
4. Check the preview and select *Finish*.

## Results

### • Example

Here, the literal is substituted with a notation.

```
REPORT adt_text_symbols.  
WRITE:/ text-001.
```

## Related Information

[Creating and Editing Text Symbols \(Quick Assists\) \[page 430\]](#)

[Creating a Text Symbol in the Text Pool \[page 431\]](#)

[Editing Text Symbols \[page 434\]](#)

[Correcting Inconsistencies Within Text Symbols \[page 435\]](#)

## 5.1.18 Working with Transport Organizer

### Context

The Transport Organizer in the ABAP Development Tools (ADT) contains the basic transport functions for the SAP GUI-based Transport Organizer.

## Related Information

[Icons and Decorators in the Transport Organizer \[page 110\]](#)

[Filtering Transport Requests and Tasks \[page 439\]](#)

[Configuring Tree \[page 440\]](#)

[Adding Users to a Transport Request \[page 442\]](#)  
[Changing the Owner of Transport Requests and Tasks \[page 443\]](#)  
[Checking Consistency of Objects \[page 445\]](#)  
[Releasing in the Transport Organizer \[page 446\]](#)  
[Deleting in the Transport Organizer \[page 449\]](#)  
[Transport Organizer \[page 104\]](#)  
[Transport Layer \[page 107\]](#)  
[Transport Request \[page 108\]](#)

## 5.1.18.1 Creating a Transport Request

You can create a new transport request from the Transport Organizer view.

### Procedure

1. Open the Transport Organizer view.
2. In the context menu, select [New Transport Request](#) or Click the [New Transport Request](#) in the toolbar of the view.

#### i Note

You can also create a new transport request through  [File](#)  [New](#)  [Other...](#)  [ABAP](#)  [Transport Request](#)

3. In the dialog box, provide the following details:
  1. Choose the [Request Type](#).
  2. Enter a [Short Description](#).
  3. (Optional) Enter a [Target](#) or search using [Browse](#).... Alternately, use content assist to search for a Target.

#### i Note

The Request will be `LOCAL` if a [Target](#) is not provided.

4. (Optional) Enter [CTS Project](#) or search using [Browse](#).... Alternately, use content assist to search for the CTS project.
5. Use [Add...](#) to add users to the task.
4. Choose [Finish](#).

### Results

The new transport request will be created with tasks assigned in the Transport Organizer view to the logged on user. The Request is opened in the ADT Editor.

## 5.1.18.2 Filtering Transport Requests and Tasks

### Context

The *Transport Organizer* view displays, by default, all transport requests and tasks of the logged on user. You can set a filter to display objects that meet the filter criteria.

#### i Note

This process simultaneously filters the following content in all configured back-end systems:

- Transport requests
- Tasks
- Owners
- Descriptions
- Object types

### Procedure

1. Open the *Transport Organizer* view.
2. Enter the search criteria in the *Filter Text* field.

#### i Note

You can enter letters, a name, special characters, or a number in the filter text field.

### Results

The reduced set of elements in the *Transport Organizer* view is displayed. TIP: Delete the filter text to display all transport requests, tasks, and objects.

### Related Information

- [Configuring Tree \[page 440\]](#)
- [Adding Users to a Transport Request \[page 442\]](#)
- [Changing the Owner of Transport Requests and Tasks \[page 443\]](#)
- [Checking Consistency of Objects \[page 445\]](#)
- [Releasing in the Transport Organizer \[page 446\]](#)
- [Deleting in the Transport Organizer \[page 449\]](#)
- [Icons and Decorators in the Transport Organizer \[page 110\]](#)

[Transport Request \[page 108\]](#)

[Transport Organizer \[page 104\]](#)

### 5.1.18.3 Configuring Tree

#### Context

You can set a filter for transport requests, tasks, and objects relating to a specific user.

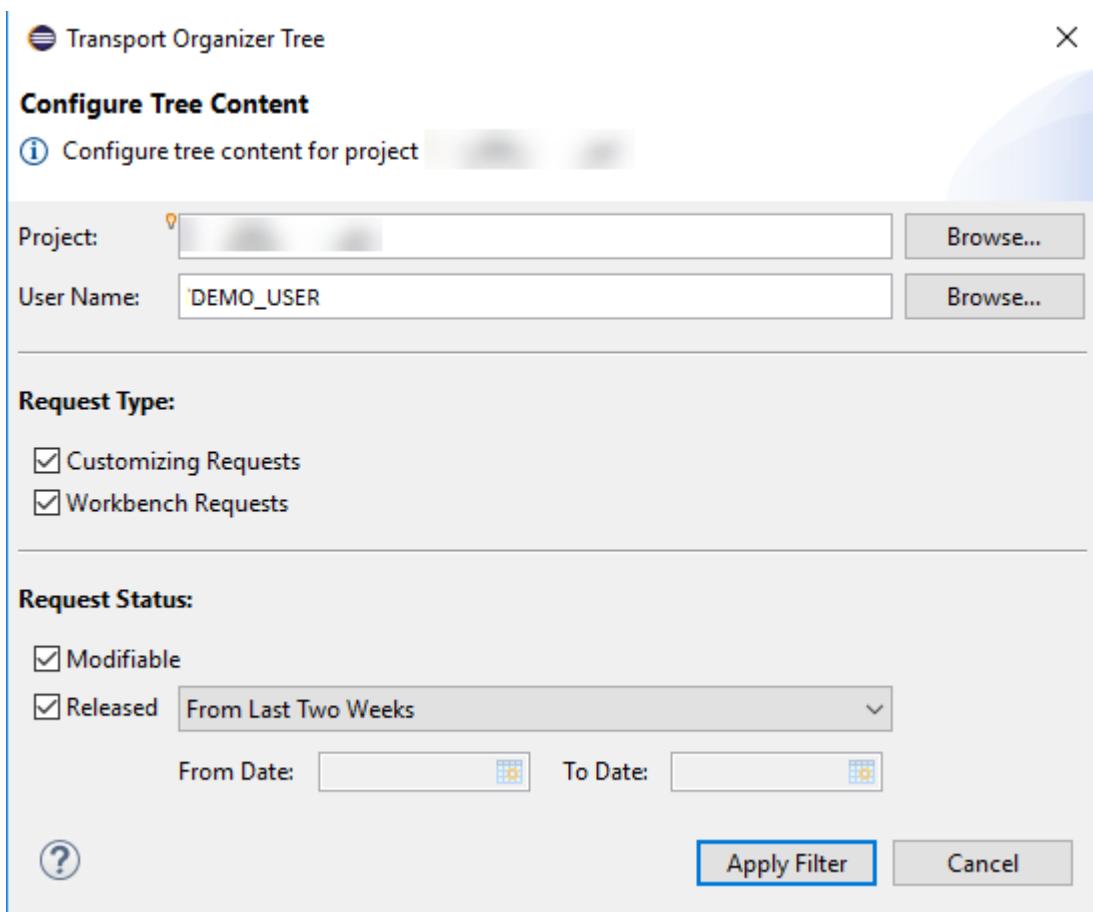
##### i Note

The *Transport Organizer* view displays, by default, all modifiable and released (last two weeks) transport requests, tasks under workbench and customizing types in accordance with the owner status of the current user for the selected project.

#### Procedure

1. Open the *Transport Organizer* view.
2. In the context menu of the ABAP project, select *Configure Tree...*
3. In the following dialog box, change preferences according to your requirement.

Configure Tree Content



#### i Note

At least one Request Type and Request Status must be set. Using the drop-down for 'Released' status you can make a selection.

4. Choose *Apply Filter*.

#### i Note

You can use *Reset Tree* in the context menu to reset to default preferences.

## Results

The reduced set of transport requests and tasks that match the selected preference are displayed.

## Related Information

[Filtering Transport Requests and Tasks \[page 439\]](#)

[Adding Users to a Transport Request \[page 442\]](#)

[Changing the Owner of Transport Requests and Tasks \[page 443\]](#)

[Checking Consistency of Objects \[page 445\]](#)

[Releasing in the Transport Organizer \[page 446\]](#)

[Deleting in the Transport Organizer \[page 449\]](#)

[Icons and Decorators in the Transport Organizer \[page 110\]](#)

[Transport Request \[page 108\]](#)

[Transport Organizer \[page 104\]](#)

## 5.1.18.4 Adding Users to a Transport Request

### Prerequisites

You can only add a user if the corresponding transport request exists in the folder *Modifiable*.

This feature is supported as of Application Server ABAP 7.53 SPO0.

### Context

You can create a new task to add users to a transport request. Consequently, you can ensure split development activities amongst several users and ensure that a transport request can only be released if all involved users have finished their work.

You can create a transport request within the *Transport Organizer* view of ABAP Development Tools. For more information, see [Creating a Transport Request \[page 438\]](#).

You can add users to a Transport Request in the following ways:

Method	Procedure
Transport Organizer View	<ol style="list-style-type: none"><li>1. Open the <i>Transport Organizer</i> view.</li><li>2. Expand the relevant project node until you have found the requested transport request.</li><li>3. In the context menu of the transport request, select <i>Add User</i>.</li><li>4. In the following dialog box, enter the user name of the <i>Owner</i> or select the button <i>Browse...</i> to search for the user name.</li><li>5. Choose <i>Finish</i>.</li></ol>

Method	Procedure
Transport Request Editor	<ol style="list-style-type: none"> <li>1. Open the Request Editor. For more information, see <a href="#">Opening a Request or Task Editor [page 453]</a>.</li> <li>2. From the context menu on the Request in the Objects section, select <a href="#">Add User</a>. Alternately, Select the <a href="#">Add User</a> icon in the toolbar of the editor.</li> <li>3. In the following dialog box, enter the user name of the <i>Owner</i> or select the button <a href="#">Browse...</a> to search for the user name.</li> <li>4. Choose <a href="#">Finish</a>.</li> </ol>

## Related Information

- [Filtering Transport Requests and Tasks \[page 439\]](#)  
[Configuring Tree \[page 440\]](#)  
[Changing the Owner of Transport Requests and Tasks \[page 443\]](#)  
[Checking Consistency of Objects \[page 445\]](#)  
[Releasing in the Transport Organizer \[page 446\]](#)  
[Deleting in the Transport Organizer \[page 449\]](#)  
[Icons and Decorators in the Transport Organizer \[page 110\]](#)  
[Transport Request \[page 108\]](#)  
[Transport Organizer \[page 104\]](#)

### 5.1.18.5 Changing the Owner of Transport Requests and Tasks

#### Context

You assign the owner of a transport request or task to make it available for a particular user.

You can change the owner of transport requests and tasks in the following ways:

Method	Procedure
Transport Organizer View	<ol style="list-style-type: none"><li>1. Open the <i>Transport Organizer</i> view.</li><li>2. Expand the <i>Workbench</i> folder.</li><li>3. Expand the <i>Modifiable</i> folder until you have found the requested transport request or task.</li><li>4. In the context menu, select <i>Change Owner</i>.</li><li>5. In the following dialog box, enter the user name of the <i>New Owner</i> or select the button <i>Browse...</i> to search for the user name.</li><li>6. Choose <i>Finish</i>.</li></ol>
Transport Request/Task Editor	<ol style="list-style-type: none"><li>1. Open the Request Editor. For more information, see <a href="#">Opening a Request or Task Editor [page 453]</a>.</li><li>2. In the Properties section, select <i>Change....</i> Alternatively, select <i>Change Owner</i> in the context menu of the <i>Objects</i> section.</li><li>3. In the following dialog box, enter the user name of the New Owner or select the button <i>Browse...</i> to search for the user name.</li><li>4. Choose <i>Finish</i>.</li></ol>

### i Note

This feature is supported as of Application Server ABAP 7.53 SP00.

## Results

The transport request or task is assigned to the new user. Only the new user can edit or release the corresponding transport request, tasks, and objects.

## Related Information

- [Filtering Transport Requests and Tasks \[page 439\]](#)
- [Configuring Tree \[page 440\]](#)
- [Adding Users to a Transport Request \[page 442\]](#)
- [Checking Consistency of Objects \[page 445\]](#)
- [Releasing in the Transport Organizer \[page 446\]](#)
- [Deleting in the Transport Organizer \[page 449\]](#)
- [Icons and Decorators in the Transport Organizer \[page 110\]](#)
- [Transport Request \[page 108\]](#)
- [Transport Organizer \[page 104\]](#)

## 5.1.18.6 Checking Consistency of Objects

### Context

You can run various consistency checks on objects that are assigned to a transport request in one operation. The consistency check will also automatically be executed when you release a transport request. Therefore, you can check in advance if any problems have occurred before releasing.

You can check the consistency of objects in the following ways:

Method	Procedure
Transport Organizer View	<ol style="list-style-type: none"><li>1. Open the <i>Transport Organizer</i> view.</li><li>2. Expand the <i>Workbench</i> folder.</li><li>3. Expand the <i>Modifiable</i> folder until you have found the requested transport request or task.</li><li>4. In the context menu of the transport request or task, choose <i>Run Consistency Checks</i>.</li></ol>
Transport Request/Task Editor	<ol style="list-style-type: none"><li>1. Open the Request Editor. For more information, see <a href="#">Opening a Request or Task Editor [page 453]</a>.</li><li>2. Select the <i>Run Consistency Checks</i> icon in the toolbar of the editor. Alternately, Under Objects section, select <i>Run Consistency Checks</i> in the context menu of transport request or task.</li></ol>

#### i Note

This feature is supported as of Application Server ABAP 7.53 SP00.

### Results

If an inconsistency exists, ABAP Development Tools displays an error message in the status bar.

### Related Information

[Filtering Transport Requests and Tasks \[page 439\]](#)

[Configuring Tree \[page 440\]](#)

[Adding Users to a Transport Request \[page 442\]](#)

[Changing the Owner of Transport Requests and Tasks \[page 443\]](#)

[Releasing in the Transport Organizer \[page 446\]](#)

[Deleting in the Transport Organizer \[page 449\]](#)

[Icons and Decorators in the Transport Organizer \[page 110\]](#)

[Transport Request \[page 108\]](#)

[Transport Organizer \[page 104\]](#)

## 5.1.18.7 Releasing in the Transport Organizer

### Context

The *Transport Organizer* view and the Request/Task editors contain basic functions for releasing [transport requests \[page 448\]](#) and [tasks \[page 446\]](#). Use this action to release the tasks. After releasing all tasks, you can also release the transport request.

### Related Information

[Filtering Transport Requests and Tasks \[page 439\]](#)  
[Configuring Tree \[page 440\]](#)  
[Adding Users to a Transport Request \[page 442\]](#)  
[Changing the Owner of Transport Requests and Tasks \[page 443\]](#)  
[Checking Consistency of Objects \[page 445\]](#)  
[Deleting in the Transport Organizer \[page 449\]](#)  
[Icons and Decorators in the Transport Organizer \[page 110\]](#)  
[Transport Request \[page 108\]](#)  
[Transport Organizer \[page 104\]](#)

## 5.1.18.7.1 Releasing Tasks

### Prerequisites

You can only release a transport task if you are the owner.

### Context

You release a task when you have finished the development activities and only when a certain development status is to be released.

You can release tasks in the following ways:

Method	Procedure
Transport Organizer View	<ol style="list-style-type: none"><li>1. Open the <i>Transport Organizer</i> view.</li><li>2. Expand the <i>Workbench</i> folder.</li><li>3. Expand the <i>Modifiable</i> folder until you have found the requested task.</li><li>4. <i>Release</i>.</li></ol>
Transport Request/Task Editor	<ol style="list-style-type: none"><li>1. Open the Task Editor. For more information, see <a href="#">Opening a Request or Task Editor [page 453]</a>.</li><li>2. Select the <i>Release</i> icon in the toolbar of the task editor. Alternately, Under Objects section, select <i>Release</i> in the context menu of transport task.</li></ol>

## Results

A task is released. If error exists, *Release Transport Request* error message appears. Click *Problems view* to view detailed error message information. The objects are still locked. This means that only those users involved in the change request can edit these objects.

If there are any ATC errors, the result can be checked under *ATC Result Browser* view.

### i Note

In the context menu of the transport task, choose *Release*. The objects are copied to the object list of its change request. Therefore, in the *Transport Organizer* view the corresponding objects and tasks are displayed at the same folder level as the  decorator. In the Transport Request Editor, Tasks are displayed with  decorator and objects are displayed on the right pane of objects section. Released objects are appended to the All Objects node.

## Related Information

[Releasing Transport Requests \[page 448\]](#)

[Releasing in the Transport Organizer \[page 446\]](#)

## 5.1.18.7.2 Releasing Transport Requests

### Prerequisites

You can only release a transport request if you are the owner and only if all containing tasks are released.

### Context

After you have released all tasks, you can release the transport request itself. Consequently, all objects of the object list that have been changed are available for import into another system.

You can release transport requests in the following ways:

Method	Procedure
Transport Organizer View	<ol style="list-style-type: none"><li>1. Open the <i>Transport Organizer</i> view.</li><li>2. Expand the <i>Workbench</i> folder.</li><li>3. Expand the <i>Modifiable</i> folder until you have found the requested transport request.</li><li>4. In the context menu of the transport task, choose <i>Release</i>.</li></ol>
Transport Request Editor	<ol style="list-style-type: none"><li>1. Open the Request Editor. For more information, see <a href="#">Opening a Request or Task Editor [page 453]</a>.</li><li>2. Select the <i>Release</i> icon in the toolbar of the transport request editor. Alternately, Under Objects section, select <i>Release</i> in the context menu on the request.</li></ol>

#### i Note

This feature is supported as of Application Server ABAP 7.53 SP00.

### Results

A transport request is released. If error exists, *Release Transport Request* error message appears. Click [Problems view](#) to view detailed error message information.

If there are any ATC errors, the result can be checked under [ATC Result Browser](#) view.

### Related Information

[Releasing Tasks \[page 446\]](#)

[Releasing in the Transport Organizer \[page 446\]](#)

## 5.1.18.8 Deleting in the Transport Organizer

### Context

You delete [transport requests \[page 449\]](#) and [tasks \[page 450\]](#) to prevent the transfer of objects into another system or client. The objects still remain in the system and will not be deleted.

### Related Information

[Filtering Transport Requests and Tasks \[page 439\]](#)  
[Configuring Tree \[page 440\]](#)  
[Adding Users to a Transport Request \[page 442\]](#)  
[Changing the Owner of Transport Requests and Tasks \[page 443\]](#)  
[Checking Consistency of Objects \[page 445\]](#)  
[Releasing in the Transport Organizer \[page 446\]](#)  
[Icons and Decorators in the Transport Organizer \[page 110\]](#)  
[Transport Request \[page 108\]](#)  
[Transport Organizer \[page 104\]](#)

### 5.1.18.8.1 Deleting Transport Requests

#### Prerequisites

The transport request has the request status *Modifiable* and has not been released yet.

### Context

You can delete a transport request that is displayed in the *Modifiable* folder.

#### i Note

You cannot delete transport requests that contain locked objects.

## Procedure

1. Open the *Transport Organizer* view.
2. Expand the *Workbench* folder.
3. Expand the *Modifiable* folder until you have found the transport request.
4. In the context menu, select **Delete**.  
The *Delete Request* dialog box appears.
5. Confirm the deletion.

## Results

The transport request and all containing tasks are removed and they disappear from the *Transport Organizer* view.

## Related Information

[Deleting Tasks \[page 450\]](#)

[Deleting in the Transport Organizer \[page 449\]](#)

## 5.1.18.8.2 Deleting Tasks

### Prerequisites

The task is modifiable and the objects assigned to the task are unlocked.

### Context

You can remove classified and unclassified tasks from a transport request. You cannot delete tasks that contain locked objects.

You can delete tasks in the following ways:

Method	Procedure
Transport Organizer View	<ol style="list-style-type: none"><li>1. Open the <i>Transport Organizer</i> view.</li><li>2. Expand the <i>Modifiable</i> folder until you have found the requested task.</li><li>3. In the context menu, select <i>Delete</i>. The Delete Task dialog box appears.</li><li>4. Choose <i>Confirm</i>.</li></ol>
Transport Request Editor	<ol style="list-style-type: none"><li>1. Open the Request Editor. For more information, see <a href="#">Opening a Request or Task Editor [page 453]</a>.</li><li>2. Under Objects section, select <i>Delete</i> in the context menu of the task.</li><li>3. Choose <i>Confirm</i>.</li></ol>

The task will be removed from the corresponding transport request.

## Related Information

[Deleting Transport Requests \[page 449\]](#)

[Deleting in the Transport Organizer \[page 449\]](#)

## 5.1.18.8.3 Removing ABAP Objects from Request or Task

### Prerequisites

This feature is supported as of Application Server ABAP 7.53 SP00.

## Context

You can remove objects from a request or task in the following ways:

Method	Procedure
Transport Organizer View	<ol style="list-style-type: none"><li>1. Open the <a href="#">Transport Organizer</a> view.</li><li>2. Expand the <a href="#">Modifiable</a> folder until you have found the object.</li><li>3. In the context menu, select <a href="#">Remove</a>. The Remove ABAP Object dialog box appears.</li><li>4. Choose <a href="#">Confirm</a>.</li></ol>
Transport Request/Task Editor	<ol style="list-style-type: none"><li>1. Open the Request Editor. For more information, see <a href="#">Opening a Request or Task Editor [page 453]</a>.</li><li>2. Select the Request or Task under the Objects section of editor.</li><li>3. In the context menu of the Object list on the right pane, select <a href="#">Remove</a>. The Remove ABAP Object dialog box appears.</li><li>4. Choose <a href="#">Confirm</a>.</li></ol>

### 5.1.18.9 Reassigning Tasks to a Transport Request

You can assign a task from an existing transport request to any other transport request assigned to your user.

#### Prerequisites

This feature is supported as of Application Server ABAP 7.53 SP00.

#### Procedure

1. Open the Transport Organizer view.
2. Expand the relevant project node and select the required task.
3. In the context menu, select [Reassign Task....](#).
4. In the dialog box, enter the Request Number or use [Browse...](#) to select an existing request.
5. Choose [Finish](#).

The Task will be moved from the source request to the newly selected transport request

## 5.1.18.10 Opening a Request or Task Editor

You can navigate to the selected request or task or the ABAP Object.

### Prerequisites

This feature is supported as of Application Server ABAP 7.53 SP00.

### Context

You can open the editor in the following ways:

Method	Procedure
Context Menu	<ol style="list-style-type: none"><li>1. Open the <i>Transport Organizer</i> view.</li><li>2. Expand the node till you find the requested request, task or ABAP Object.</li><li>3. Double click or in the context menu, select <i>Open</i>.</li><li>4. Editor of selected object will be opened.</li></ol>
Toolbar icon	<ol style="list-style-type: none"><li>1. Open <i>Transport Organizer</i> view.</li><li>2. Click <i>Open Request/Task</i>.</li><li>3. In the dialog box, enter the Request or Task.</li><li>4. Choose Open.</li></ol>
Request or Task Editor	<ol style="list-style-type: none"><li>1. Double click or click <i>Open</i> context menu of the Tasks/ABAP objects present in the Object List section of the editor.</li><li>2. To open the Request Editor from the Task Editor, click the Request link in the properties section of the Task Editor.</li></ol>

## 5.1.18.11 Editing a Transport Request or a Task

You can edit the transport request or task in the editor.

### Context

Open the Editor to edit a transport request or task. For more information, see [Opening a Request or Task Editor \[page 453\]](#).

The tasks that can be performed in the Request Editor are:

1. Changing Short and long description
2. [Changing the Owner of Transport Requests and Tasks \[page 443\]](#)
3. Changing Target
4. Changing CTS Project
5. [Releasing Tasks \[page 446\]](#)
6. [Releasing Transport Requests \[page 448\]](#)
7. [Adding Users to a Transport Request \[page 442\]](#)
8. Checking Consistency of Objects [page 445]
9. [Deleting Tasks \[page 450\]](#)
10. [Removing ABAP Objects from Request or Task \[page 451\]](#)
11. Share links
- 12.
- 13.
14. [Including Objects in a Request Manually \[page 456\]](#)
15. [Navigating to Revision History \[page 458\]](#)
16. [Comparing Across ABAP Projects \[page 458\]](#)
17. [Comparing with Previous Revision \[page 459\]](#)

The tasks that can be performed in the Task Editor are:

1. Changing Short and long description
2. [Changing the Owner of Transport Requests and Tasks \[page 443\]](#)
3. Checking Consistency of Objects [page 445]
4. [Releasing Tasks \[page 446\]](#)
5. [Including Objects in a Request Manually \[page 456\]](#)
6. [Navigating to Revision History \[page 458\]](#)
7. [Comparing Across ABAP Projects \[page 458\]](#)
8. [Comparing with Previous Revision \[page 459\]](#)

## 5.1.18.12 Protecting a Transport Request

To ensure that only the owner of a request can add more users, you can protect the request.

### Context

Other users cannot add users if the request protection is enabled.

### Procedure

1. Open *Transport Organizer* view.

2. Choose a modifiable request.
3. In the context menu choose *Protect*.

**i Note**

To remove the protection, choose *Remove Protection*.

This option is only available in the Transport Organizer view.

### 5.1.18.13 Changing a Task Type

You can change the task type of a task that is added under a transport request.

#### Context

Method	Procedure
Transport Organizer View	<ol style="list-style-type: none"> <li>1. Open the <i>Transport Organizer</i> view.</li> <li>2. Expand the relevant project node until you find the requested transport task.</li> <li>3. In the context menu, select <i>Change Task Type</i></li> <li>4. In the sub-menu, select the relevant task type :           <ul style="list-style-type: none"> <li>○ Development/Correction</li> <li>○ Repair</li> <li>○ Unclassified</li> </ul> </li> </ol>
Transport Task Editor	<ol style="list-style-type: none"> <li>1. Open the <i>Task Editor</i>. For more information, see <a href="#">Opening a Request or Task Editor [page 453]</a>.</li> <li>2. To change task type, in the <i>Properties</i> section, choose <i>Browse</i>.</li> <li>3. In the context menu, select <i>Change Task Type</i></li> <li>4. In the sub-menu, select the relevant task type :           <ul style="list-style-type: none"> <li>○ Development/Correction</li> <li>○ Repair</li> <li>○ Unclassified</li> </ul> </li> <li>5. Choose <i>Save</i>.</li> </ol>

**i Note**

You cannot add any objects if the task type is *Unclassified*. You need to change the task type to *Development/Correction* or *Repair*.

## 5.1.18.14 Using Sort and Compress

You can delete one of several identical entries in a transport request or task.

### Procedure

1. Open the Transport Organizer view.
2. Expand the relevant project node and select the required transport request, task or ABAP object.
3. In the context menu, select *Sort and Compress*.

### Results

Duplicate object entries will be removed from the transport request or task.

## 5.1.18.15 Including Objects in a Request Manually

To include objects in a request, you have the following options:

1. [Add Object... \[page 456\]](#) which allows selection of objects of your choice by providing the object name, type, and the program ID.
  2. [Add objects from request... \[page 457\]](#) which copies objects from another request.
- In both the cases, the objects are added to the request without locks.  
These objects get locked when the request is being released; if the same objects are locked for editing by other change requests, then these change requests should be released first to avoid any conflicts.  
This is to make sure that the requests do not transport objects in an undefined intermediate state.

### 5.1.18.15.1 Including required Object to the Request

Through this feature, you can assign objects of your choice to a request

### Procedure

1. Open the *Request Editor*.
2. Under the *Objects* section, choose *Include Objects* from the context menu of the request.
3. Select [Add Object...](#) from the sub-context menu.
4. In the dialog box, select the *Object Name*, *Object Type* and *Program ID*.

5. Choose *Finish*.

## Results

The Object is successfully included in the request.

### i Note

Objects can also be added to a task in the *Task Editor* by choosing the *Add Object* tool item in the *Objects* section.

## 5.1.18.15.2 Including the Object Lists from a Request

You can assign objects from one request to another request.

## Procedure

1. Open the *Request Editor*.
2. Under the *Objects* section, select the request to which you need to add objects and choose *Include Objects* from the context menu.
3. Select *Add objects from request...* from the sub-context menu.
4. In the dialog box, select the source request from which the objects must be copied.
5. Choose *Finish*.

## Results

All the objects from the source request is copied to the target request.

## 5.1.18.16 Comparing the Change in Objects

You can compare the changes in an object from a transport request or task editor.

There are two ways to compare the objects:

1. [Navigating to Revision History of Transport Objects \[page 458\]](#) : which shows changes associated with transport request within a single ABAP project
2. [Comparing across ABAP Projects \[page 458\]](#): which compares the changes in two different ABAP projects

## 5.1.18.16.1 Navigating to Revision History

You can open the History view from the transport request or task editor. These revisions show changes over a much longer period. You can compare the changes from one transport of source code to another.

### Procedure

1. Open the Request Editor. For more information, see [Opening a Request or Task Editor \[page 453\]](#)
2. Select the Request or Task under the Objects section of editor.
3. In the context menu of the object list, choose  [Compare With](#) .

### Results

This will open the [History](#) view with the revision details of the object.

## 5.1.18.16.2 Comparing Across ABAP Projects

You can compare the newest version of an object in each of the two ABAP projects. You can use this comparison to check code-level changes in separate ABAP back-end systems.

### Procedure

1. Open the [Request Editor](#).  
For more information, see [Opening a Request or Task Editor \[page 453\]](#)
2. Select the [Request](#) or [Task](#) under the Objects section of editor.
3. In the context menu of the object list, choose  [Compare With](#)  where ABAP project is one of the projects defined in your ADT.

### Results

A compare editor opens with the two versions of the object corresponding to each system.

## 5.1.18.16.3 Comparing with Previous Revision

### Context

You can compare the changes made to the source code with the last revised version of the code that was transported.

### Procedure

From the context menu of an object in the *Project Explorer*, source code editor or Transport Request editor, choose  *Compare With Previous Revision*

### Results

A new compare editor window opens, showing the current version of the code. The last transported version of the code is shown with its timestamp in an adjacent view. Changes are highlighted in the two windows. Locations or changes are shown as icons at the right edge of the saved code version.

## 5.1.18.17 Specifying Table Keys

### Prerequisites

Ensure you have necessary authorization to edit the objects.

### Context

You can use the object list of a Transport Request editor to edit one or more entries of a table that is locked in a transport request. You need to first specify the table (TABU) or a generic transport object (TDAT, VDAT, CDAT), and edit the individual key fields.

## Procedure

1. Open the Transport Request editor.
2. Create a new object entry with *Object Name*, *Object Type* and *Program ID*. Check **Including Objects in a Request Manually** [page 456]
3. Choose *Open* from the context menu to specify the key fields of the object. You can also double click on the object.
4. The *Object Keys* editor opens.
5. Choose *Add* to add an empty row to the *Table Keys* section.
6. Enter a value in the *Table Key* field under *Table Key Details* section. The table key value is the concatenated version of the key elements of the table. Alternatively, you can also enter the individual key field values in the table.
7. Save the changes.

### i Note

Object types like CDAT, VDAT, and TDAT are **read only**.

## Results

The changes to the table are successfully saved.

### i Note

- You can also modify the table key value.
  - Select the key value from the *Table Keys* section.
  - Edit the value under the *Table Key Details* section.
- If the key ends with an asterisk (\*), all keys that match up to the asterisk are specified. If the key description contains more than one asterisk, the initial asterisks are interpreted as characters.  
Example
  - 000Dabc\* specifies all keys that start with 000Dabc.
  - 000Dabc\*\* specifies all keys that start with 000Dabc\*.

## 5.1.18.18 Merge Requests

This function moves all the objects and tasks that are under one request to another request.

## Procedure

1. Open the *Transport Organizer* View

2. Expand the relevant project node and select the required request.
3. In the context menu, select *Merge Requests...*
4. In the dialog box, enter the target request into which the source request should be merged.
5. Choose *Finish*.

## Results

All the objects and tasks of source request are moved into the target request. The empty request (source request) is then deleted.

## 5.1.19 Accessing ABAP Keyword Documentation

The ABAP Keyword Documentation describes the syntax and meaning of the keywords of the ABAP language and its object-oriented part ABAP Objects.

### Prerequisites

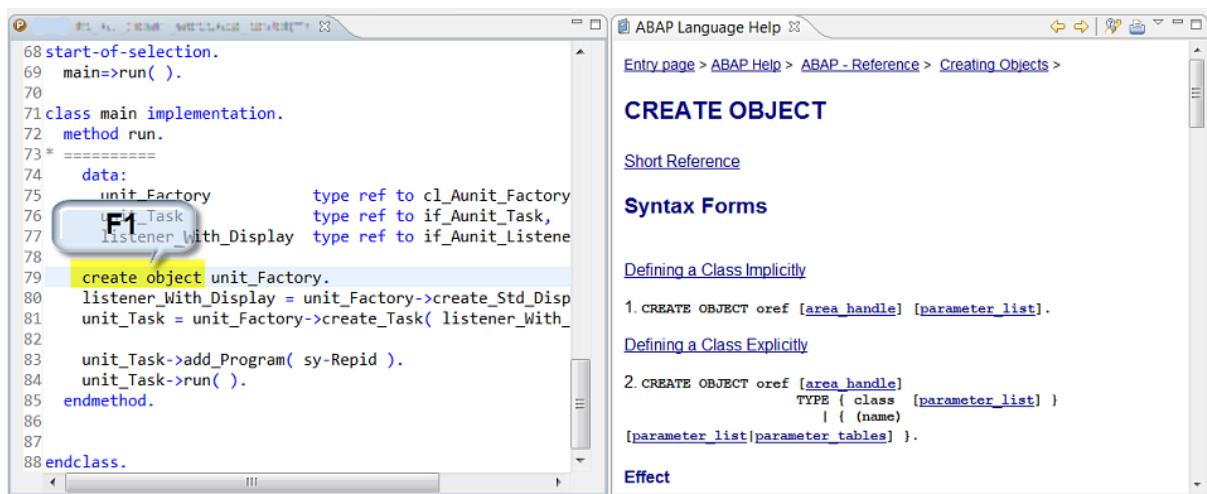
### Context

While editing ABAP source code, you will generally need to fall back on the ABAP language help. Here, you can to find more information about the syntax and meaning an ABAP keyword. For this, ABAP Development Tools provides a context-sensitive link from within the ABAP source editor. The help content is displayed in a separate help view.

The content is stored in the back-end system, so it is in sync with the release of the ABAP project in which you are working.

### Procedure

To access the ABAP Keyword Documentation, position the cursor on an ABAP statement or ABAP keyword for which you need help and choose **F1**.



Context-sensitive ABAP language help in the source code editor

## 5.1.20 Working with Released APIs

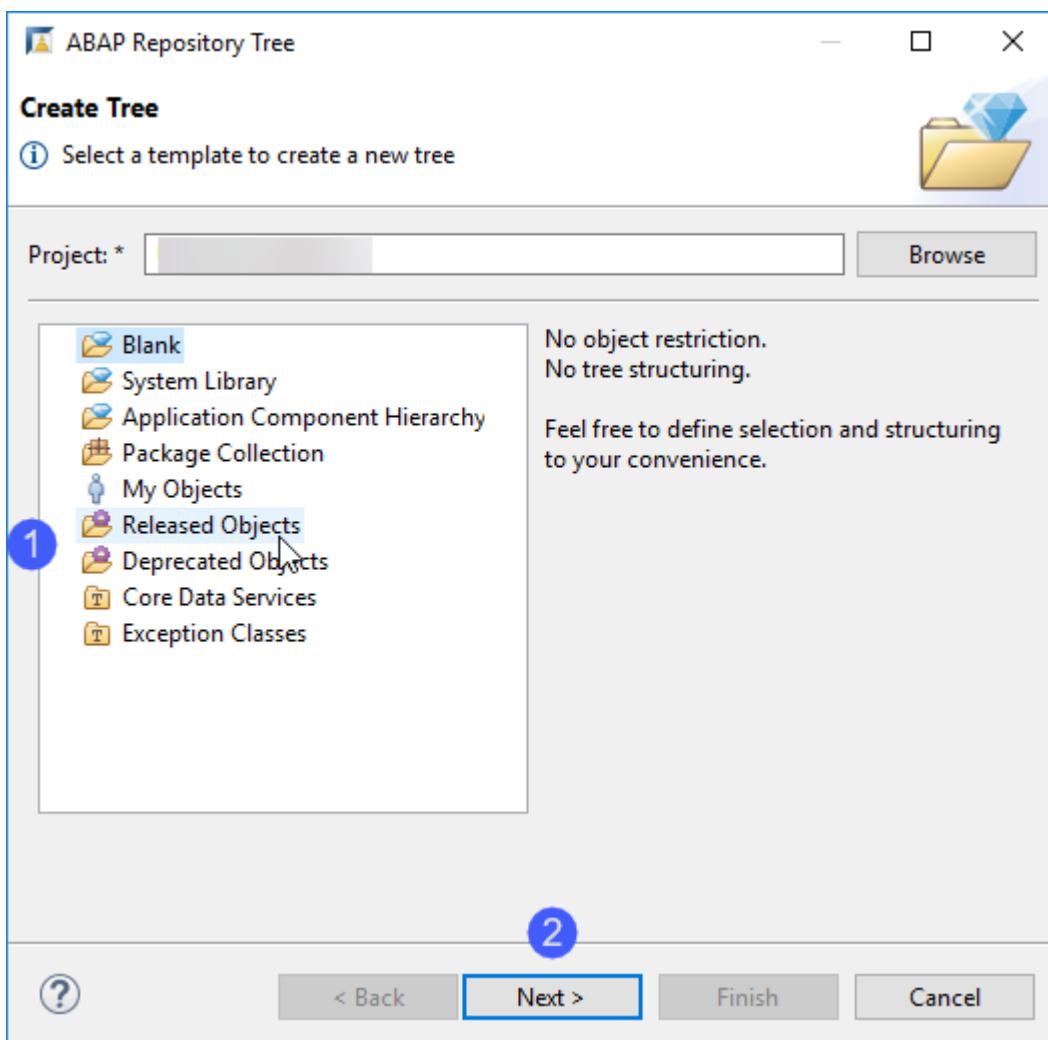
### 5.1.20.1 Finding Released APIs and Deprecated Objects

You have the following options to search and group released APIs belonging to a certain contract:

- Using the *Repository Tree*
- Using the *Open ABAP Development Object* dialog

#### Repository Tree

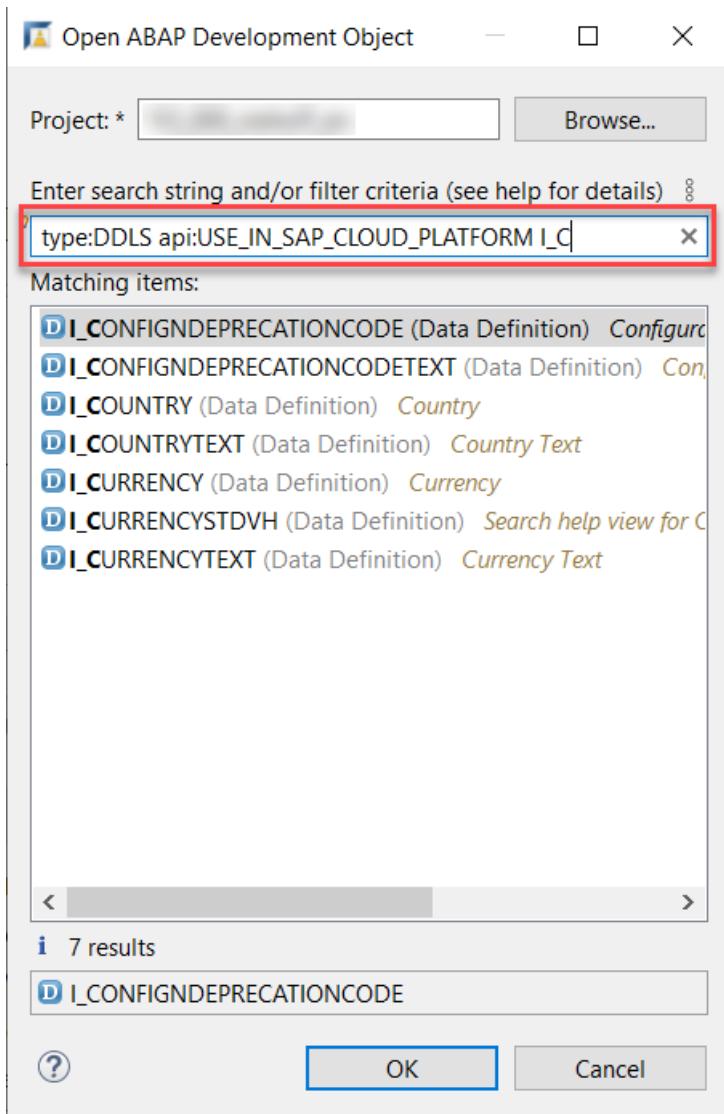
1. In the Project Explorer, right-click on an ABAP Project and choose **New > ABAP Repository Tree**.
2. In the Create Tree dialog, choose *Released Objects* as filter criteria and click *Next* as shown below. Alternatively, you can choose *Deprecated Objects*.



3. In the input field *Property Filter*, type **api** and press **Ctrl** + **space** to open the list of different use types for which development objects can be released. Alternatively, you can search for the following key words when searching for following release states :
  - **USE\_AS\_REMOTE\_API** (Use as Remote API)
  - **DEPRECATED** (Deprecated)
  - **DEPRECATED\_AS\_REMOTE\_API** (Deprecated for Use as Remote API)
  - **USE\_IN\_SAP\_CLOUD\_PLATFORM** (Use in SAP Cloud Platform)
  - **DEPRECATED\_FOR\_CLOUD\_PLATFORM** (Deprecated for Use in SAP Cloud Platform)
4. In the Project Explorer, the **ABAP Repository Tree** does now appear as additional tree as part of the ABAP project, containing the development objects which were released as API for the selected types of use. If you have selected all release states, and there are no objects for a respective release state, it will not be visible in the **ABAP Repository Tree**.

## Open ABAP Development Object Dialog

1. 
2. Click **Ctrl** + **shift** to open the *Open ABAP Development Object* dialog.
3. Search for *api: + release state* as shown in the following example. In this example, the filter api is being used to find CDS views (filter type:DDLS) with name pattern *I\_C* which were released as APIs for use in SAP Cloud Platform (filter api:USE\_IN\_SAP\_CLOUD\_PLATFORM):



## 5.1.20.2 Setting the API Release State

You can set the release state either in the properties tab or by right-clicking on the object in the project explorer. Once an ABAP development was opened, its API state is displayed in the API State tab of the Properties view. The API State tab is visible only for types of objects which can be released as APIs.

### Procedure

You can change the API state of your own development object as follows:

#### → Remember

If you want to make objects available in any of your other software components, you need to release these objects with visibility Use in SAP Cloud Platform.

#### i Note

Keep in mind that if you use a released object in another software component, the two software components become dependent on each other.

#### i Note

Released APIs mustn't be changed incompatibly. For details, see [Released APIs \[page 101\]](#). Otherwise, for example, runtime errors might occur or an upgrade might fail.

1. Open the Properties tab for the object you want to set the API release status for.  
Alternatively, you can change the release state by right-clicking on the object in the *Project Explorer*. You can then select the *Change API State* option and open the wizard from there.
2. Click *Add Release Contract* to open the API Change Wizard. An object can only be released for one API state, not two at once.  
The *Change API State* wizard is opened.
3. Select *Released* for the API state you want to release the object for. The available release contracts depend on the type of the object to be released.
4. Choose *Next*.
5. Select the release state. On the second screen, release state *Released* is already set as default value. The following release states are available:  
The following release states are available:
  - *Released*: Choose this release state to make your development object available for customers in a restricted Cloud environment.
  - *Not to Be Released*: Choose this release state to document the decision that your development object isn't suitable for releasing.
  - *Not to Be Released, Stable*: Choose this release state if your development must not be released, but nevertheless needs to remain stable.
6. You can enter a comment if you want to provide details about your release decision. This comment is stored only locally in the source system and won't be transported.

7. *(Only for Release Contract C1)*: Define the visibility.
8. On the *Validation* screen, you can check if any errors have occurred.
9. On the *Selection of Transport Request* tab, select the respective transport request.

## Results

The object is now released for the selected release contract.

## Related Information

[Released APIs \[page 101\]](#)

### 5.1.20.3 Deprecating Development Objects

You want to deprecate a development object and define a successor.

## Context

You can mark a development object as deprecated, for example if you've defined an improved successor that is to be used instead.

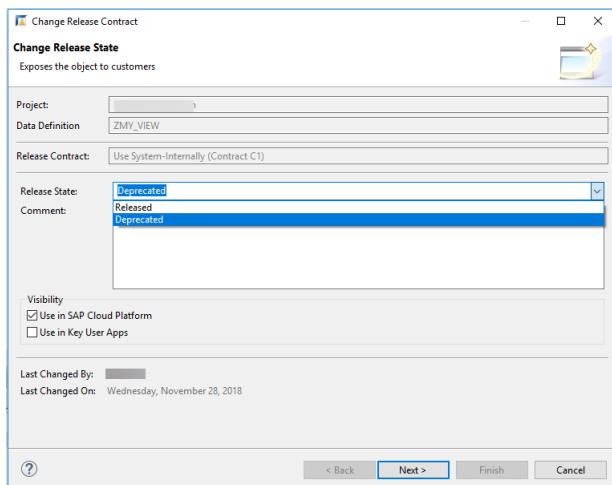
### i Note

Your development object must have the release State **Released**.

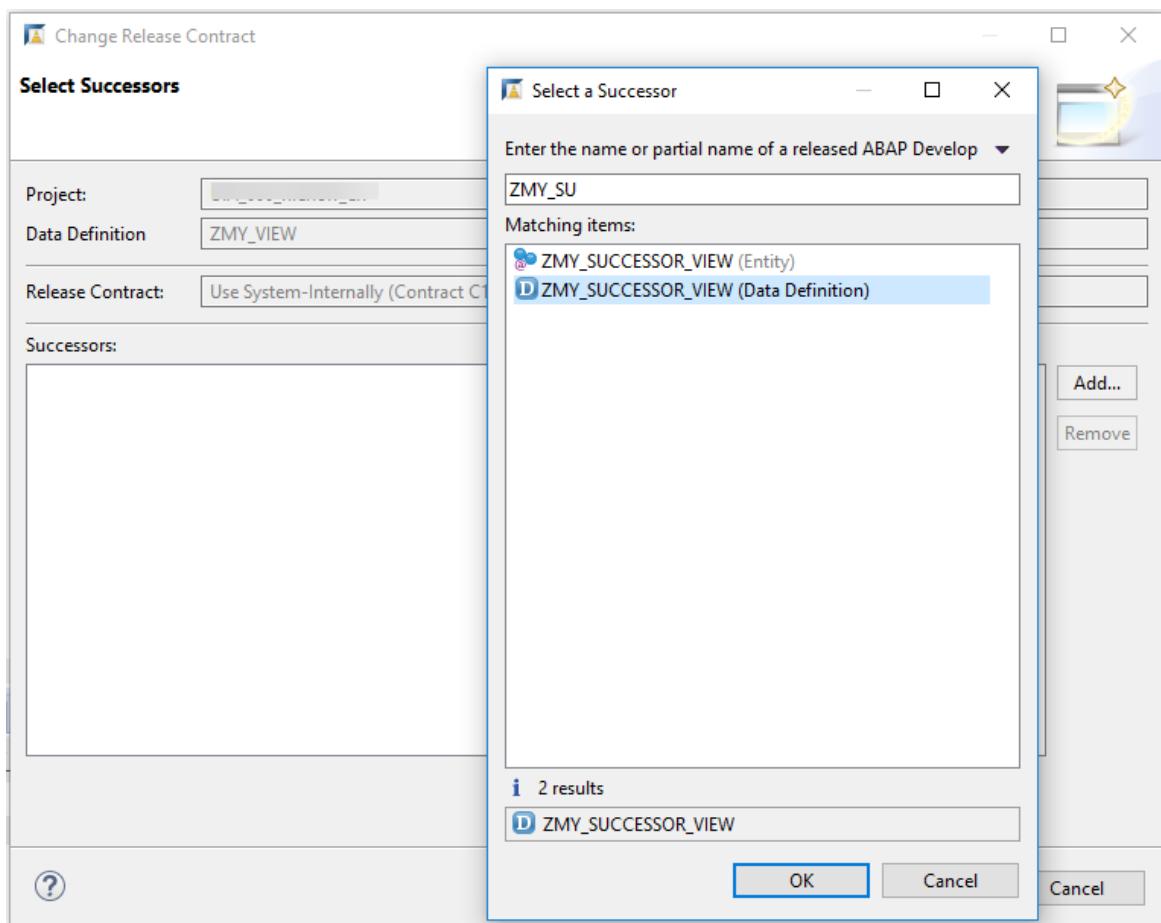
## Procedure

1. Open the Properties tab for the object you want to set the API release status for.
2. Click *Change* to open the API State Wizard.
3. Change the release state from *Released* to *Deprecated* and click button *Next*.





4. Click **Add** to select at least one successor. This is mandatory: A development object can only be deprecated if a successor was defined.
5. Select a successor. The successor must be a released API, which can serve as a replacement in all places where the deprecated object was used before.



6. Click **Next** to validate both the release state change and the selected successor objects.
7. Select a transport request.

## Results

The development object is now deprecated and a successor is in place. Using a deprecated object leads to a syntax warning that points to the successor.

### 5.1.20.3.1 Deprecating on CDS Entity Element Level

You want to deprecate an element in a CDS entity and you want to define a successor.

#### Context

 Note

Your CDS entity has the release State *Released*.

#### Procedure

Set the following annotations for the element you want to deprecate:

- `@API.element.releaseState: #DEPRECATED`
- `@API.element.successor: 'SucceedingReleasedElement'`  
( It's mandatory to select a successor for an element).

#### Results

The element is now deprecated and a successor is in place.

### 5.1.21 Working with Data Preview

#### 5.1.21.1 Accessing Data Preview

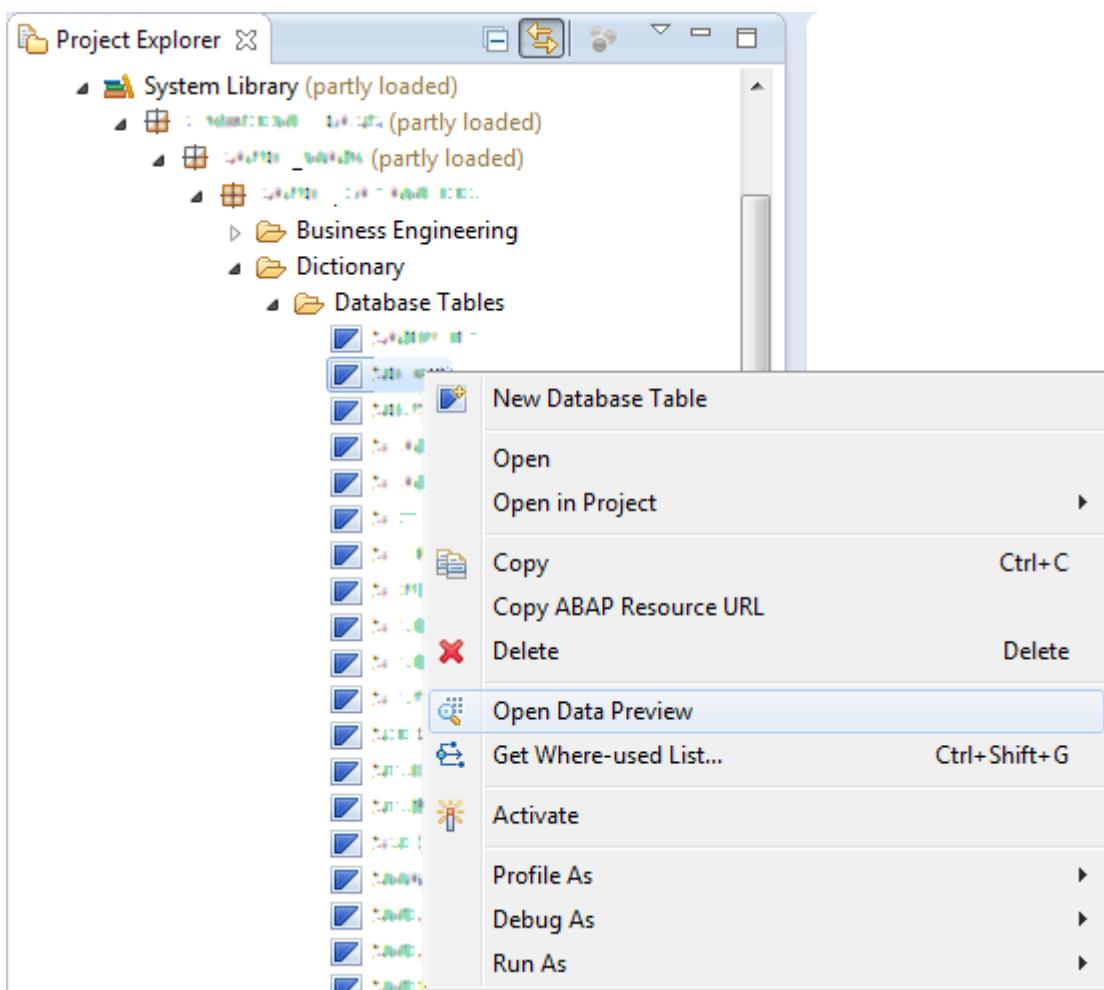
##### Prerequisites

- You have already created an ABAP cloud project. For information, see [Creating an ABAP Cloud Project \[page 118\]](#).

- You have added ABAP packages required for your work in the favorite list. For information, see [Adding a Favorite Package \[page 137\]](#).

## Procedure

1. In the *Project Explorer* view, choose the required ABAP project.
2. Choose the required ABAP package.
3. Navigate to the required database table or view in the ABAP Dictionary.
4. In the context menu, choose *Open Data Preview*.



Project Explorer context menu with the open data preview option

You can also use the combination key **Alt** + **F8** to open a database table or a view in the ABAP Data Dictionary.

## Results

The Data Preview tool displays the top 100 records from the selected database table or view by default.

## Related Information

[Data Preview \[page 79\]](#)  
[Setting Result-Set Size \[page 470\]](#)  
[Configuring Columns \[page 471\]](#)  
[Adding or Removing Filters \[page 473\]](#)  
[Viewing Logs \[page 480\]](#)  
[Viewing Distinct Column Values \[page 481\]](#)  
[Saving Result Set \[page 482\]](#)  
[Viewing Frequently Accessed Data \(Data Aging\) \[page 479\]](#)

### 5.1.21.2 Setting Result-Set Size

#### Prerequisites

You have already opened the Data Preview for the required table.

#### Context

The Data Preview tool allows you to set the size of a result-set to retrieve records from ABAP Data Dictionary tables, Views, external Views, and ABAP DDL Sources. The result-set size determines the number of records that the Data Preview tool can retrieve. The default value is 100 and the maximum value is 5000.

##### i Note

Using the value in the *Max rows* field, you can restrict the number of records for display in the Data Preview. The number you set is the number of records displayed, even if the filter criteria has fetched more records. The *Number of Entries* button displays the total number of records that match the filter criteria. If you have not applied any filter, the button displays the total number of records in the database.

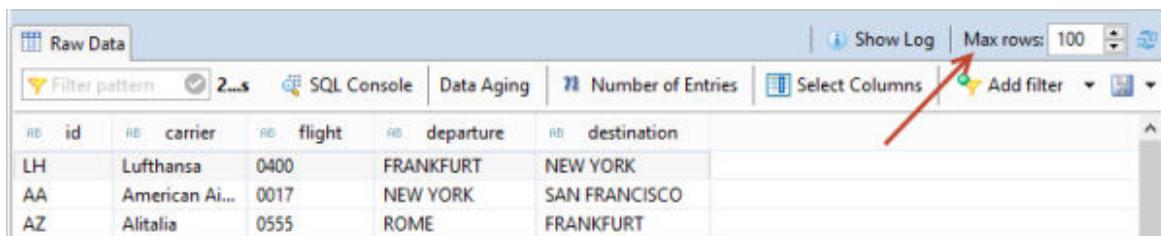
#### Procedure

1. In the top-right of the Data Preview tool, enter or select a value in the *Max rows* field.

The default value is 100.

2. Press **Enter** or press (Refresh).

The Data Preview tool fetches the number of records according to the value entered in the **Max rows** field.



Raw Data						Show Log	Max rows: 100	Print
Filter pattern		2...s	SQL Console	Data Aging	Number of Entries	Select Columns	Add filter	...
id	carrier	flight	departure	destination				
LH	Lufthansa	0400	FRANKFURT	NEW YORK				
AA	American Ai...	0017	NEW YORK	SAN FRANCISCO				
AZ	Alitalia	0555	ROME	FRANKFURT				

## Related Information

[Saving Result Set \[page 482\]](#)

[Viewing Logs \[page 480\]](#)

[Adding or Removing Filters \[page 473\]](#)

[Configuring Columns \[page 471\]](#)

[Setting Result-Set Size \[page 470\]](#)

[Accessing Data Preview \[page 468\]](#)

## 5.1.21.3 Configuring Columns

### Prerequisites

You have already opened the Data Preview tool for the Data Dictionary table or Data Dictionary View.

### Context

You can configure Data Preview to view or work with specific columns. By default, Data Preview displays the first 20 columns of an ABAP Data Dictionary table or View. This option is not available in ABAP DDL Source and external Views.

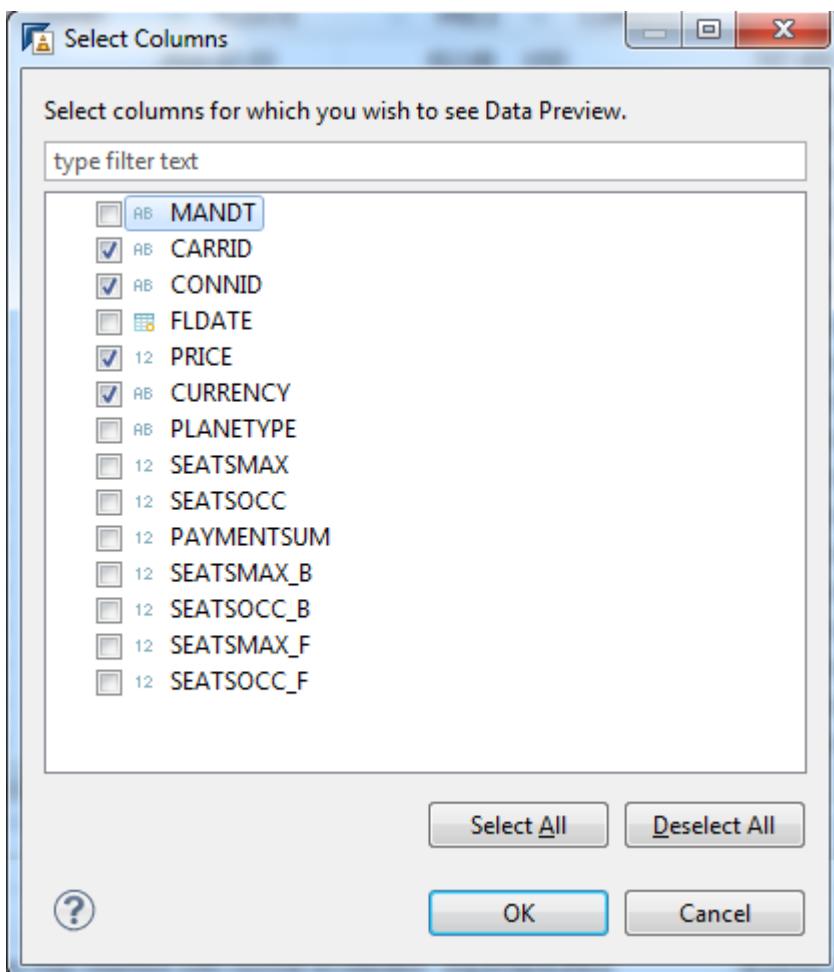
### Procedure

1. In the top-right of the Data Preview tool, choose **Select Columns**.



Raw Data														
100 rows retrieved - 382 ms (partial result)														
	AB	MANDT	AB	CARRID	AB	CONNID	AB	FLDATE	12	PRICE	AB	CURRENCY	AB	12
000		UA		3517		3517		2014-02-03		612.68		USD		747-40
000		UA		3517		3517		2014-01-27		612.68		USD		747-40
000		UA		3517		3517		2014-01-20		612.68		USD		747-40
000		UA		3517		3517		2014-01-13		612.68		USD		747-40
000		UA		3517		3517		2014-01-06		612.68		USD		747-40
		...		...		...		...		...		...		...

2. In the **Select Columns** wizard, only select columns that you want to display in Data Preview.



3. Choose **OK**.

## Results

The selected columns appear in Data Preview.

## Related Information

[Accessing Data Preview \[page 468\]](#)  
[Setting Result-Set Size \[page 470\]](#)  
[Adding or Removing Filters \[page 473\]](#)  
[Viewing Logs \[page 480\]](#)  
[Viewing Distinct Column Values \[page 481\]](#)  
[Saving Result Set \[page 482\]](#)

### 5.1.21.4 Adding or Removing Filters

#### Prerequisites

You have already opened the Data Preview tool for the ABAP Data Dictionary tables, Views, external Views, or ABAP DDL Sources.

#### Context

To identify and list specific column entries, the Data Preview tool allows you to set filters. You can set a filter on a single column or on multiple columns. The Data Preview tool allows you to set filters using any of the following options:

- [Using the Add Filter Button \[page 474\]](#)
- [Using Quick Filter \[page 477\]](#)
- [Using Local Filter \[page 478\]](#)

## Related Information

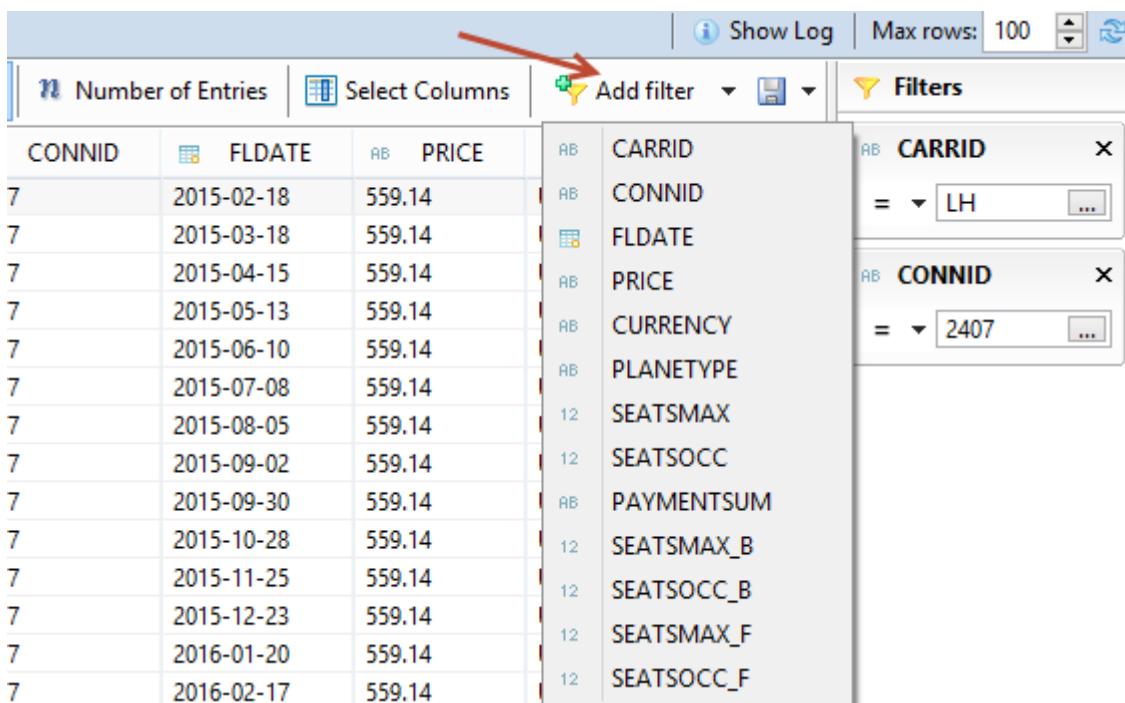
[Saving Result Set \[page 482\]](#)  
[Viewing Distinct Column Values \[page 481\]](#)  
[Viewing Logs \[page 480\]](#)  
[Configuring Columns \[page 471\]](#)  
[Setting Result-Set Size \[page 470\]](#)  
[Accessing Data Preview \[page 468\]](#)

## 5.1.21.4.1 Using the Add Filter Button

### Context

### Procedure

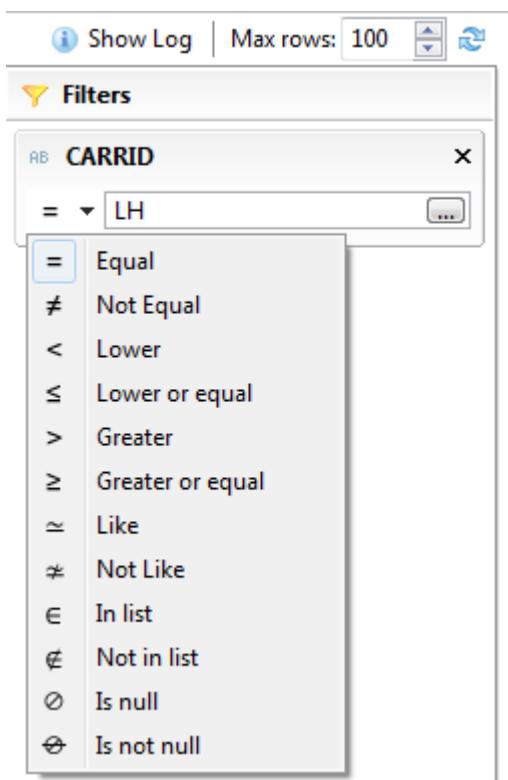
1. In the top-right of the Data Preview tool, identify the *Add Filter* button.
2. Choose *Add Filter* to list the columns.



The screenshot shows the SAP Data Preview tool interface. On the left is a table with columns: CONNID, FLDATE, and PRICE. On the right is a list of columns with their data types and lengths: CARRID (AB, 10), CONNID (AB, 10), FLDATE (AB, 10), PRICE (AB, 10), CURRENCY (AB, 10), PLANETYPE (AB, 10), SEATSMAX (12), SEATSOCC (12), PAYMENTSUM (12), SEATSMAX\_B (12), SEATSOCC\_B (12), SEATSMAX\_F (12), and SEATSOCC\_F (12). A red arrow points to the 'Add filter' button in the top right of the interface. A dropdown menu is open, showing two filter criteria: 'CARRID = LH' and 'CONNID = 2407'. The 'Filters' button is also visible in the top right.

3. Choose a column from the list.

A section to set the filter criteria appears.



The table below describes some filter criteria that you can use:

Operator	Description
Equal	<p>The Equal operator selects all records with value equal to the value specified in the text field. This operator works with column data types such as:</p> <ul style="list-style-type: none"> <li>○ String</li> <li>○ Numeric</li> <li>○ Timestamp</li> </ul>
Not Equal	<p>The Not Equal operator selects all records with value not equal to the value specified in the text field. This operator works with column data types such as:</p> <ul style="list-style-type: none"> <li>○ String</li> <li>○ Numeric</li> <li>○ Timestamp</li> </ul>
Lower	<p>The Lower operator selects all records with value lower than the value specified in the text field. For columns with String data type, Data Preview selects values with lexical order lower than the value specified in the text field. This operator works with column data types such as:</p> <ul style="list-style-type: none"> <li>○ String</li> <li>○ Numeric</li> <li>○ Timestamp</li> </ul>

Operator	Description
Lower or equal	<p>The Lower or equal operator selects all records with value lower than or equal to the value specified in the text field. For columns with String data type, Data Preview selects values with lexical order lower than or equal to the value specified in the text field. This operator works with column data types such as:</p> <ul style="list-style-type: none"> <li>○ String</li> <li>○ Numeric</li> <li>○ Timestamp</li> </ul>
Greater	<p>The Greater operator selects all records with value greater than the value specified in the text field. For columns with String data type, Data Preview selects values with lexical order greater than the value specified in the text field. This operator works with column data types such as:</p> <ul style="list-style-type: none"> <li>○ String</li> <li>○ Numeric</li> <li>○ Timestamp</li> </ul>
Greater or equal	<p>The Greater or equal operator selects all records with value greater than or equal to the value specified in the text field. For columns with String data type, Data Preview selects values with lexical order greater than or equal to the value specified in the text field. This operator works with column data types such as:</p> <ul style="list-style-type: none"> <li>○ String</li> <li>○ Numeric</li> <li>○ Timestamp</li> </ul>
Like	<p>The Like operator selects all records with value that matches the regular expression value specified in the text field. This operator works with columns of String data type.</p>
Not Like	<p>The Not Like operator selects all records with value that does not match the regular expression value specified in the text field. This operator works with columns of String data type.</p>
In List	<p>The In List operator selects all records with value that is equal to any of the values specified in the text field. This operator works with columns of String and Numeric data types.</p>
Not In List	<p>The Not In List operator selects all records with value that is not equal to any of the values specified in the text field. This operator works with columns of String and Numeric data types.</p>
Is Null	<p>The Is Null operator selects all records with the value "is null". This operator works with column data types such as:</p> <ul style="list-style-type: none"> <li>○ String</li> <li>○ Numeric</li> <li>○ Timestamp</li> </ul>

Operator	Description
Is Not Null	<p>The Is Not Null operator selects all records with the value "is not null". This operator works with column data types such as:</p> <ul style="list-style-type: none"> <li>○ String</li> <li>○ Numeric</li> <li>○ Timestamp</li> </ul>

For information about data types, see

4. Set filter criteria using the available options.

The filtered value appears in the column.

## Related Information

[Using Quick Filter \[page 477\]](#)

[Using Local Filter \[page 478\]](#)

[Adding or Removing Filters \[page 473\]](#)

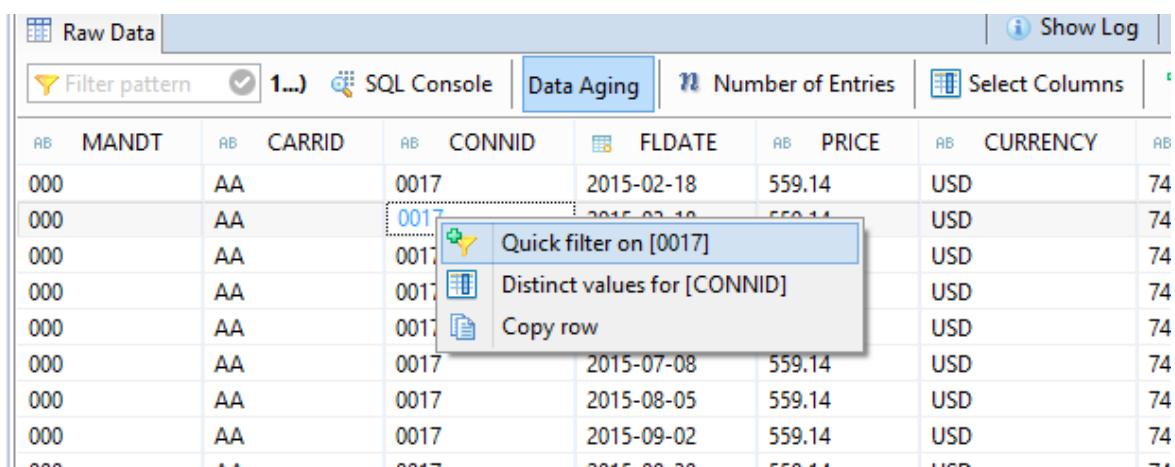
### 5.1.21.4.2 Using Quick Filter

## Context

## Procedure

1. Choose a column value.
2. In the context menu, choose *Quick Filter on*.

The Data Preview tool displays records that match the Quick Filter criteria.



The screenshot shows the SAP Data Preview tool interface. The top navigation bar includes 'Raw Data', 'Show Log', 'Filter pattern', '1...', 'SQL Console', 'Data Aging', 'Number of Entries', 'Select Columns', and a refresh icon. The main area is a grid of data with columns: MANDT, CARRID, CONNID, FLDATE, PRICE, CURRENCY, and a numerical key. A context menu is open over a row where the CONNID value is '0017'. The menu options are: 'Quick filter on [0017]', 'Distinct values for [CONNID]', and 'Copy row'. The grid data is as follows:

MANDT	CARRID	CONNID	FLDATE	PRICE	CURRENCY
000	AA	0017	2015-02-18	559.14	USD
000	AA	0017	2015-02-18	559.14	USD
000	AA	0017	2015-02-18	559.14	USD
000	AA	0017	2015-02-18	559.14	USD
000	AA	0017	2015-07-08	559.14	USD
000	AA	0017	2015-08-05	559.14	USD
000	AA	0017	2015-09-02	559.14	USD
000	AA	0017	2015-09-02	559.14	USD

### i Note

To remove a filter, close the section used to set the filter criteria

## Related Information

[Using the Add Filter Button \[page 474\]](#)

[Using Local Filter \[page 478\]](#)

[Adding or Removing Filters \[page 473\]](#)

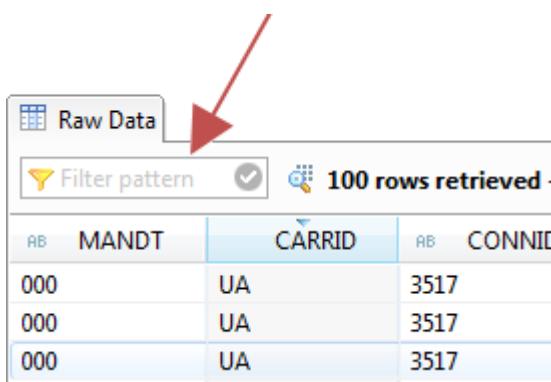
### 5.1.21.4.3 Using Local Filter

## Context

The local filter highlights all values that match the filter criteria in the record-set fetched from the backend. The local filter does not make any backend connection.

## Procedure

1. In the top-left of the Data Preview tool, identify the Filter Pattern field.
2. Enter a value.



AB	MANDT	CARRID	AB	CONNID
000	UA			3517
000	UA			3517
000	UA			3517

## Results

The Data Preview tool highlights all values that match the criteria.

## Related Information

[Using the Add Filter Button \[page 474\]](#)

[Using Quick Filter \[page 477\]](#)

[Adding or Removing Filters \[page 473\]](#)

## 5.1.21.5 Viewing Frequently Accessed Data (Data Aging)

The Data Preview tool also contains the Data Aging option for viewing the most frequently accessed data (hot data) in your database.

### Prerequisites

- You are using a database that supports data aging.
- You have switched on the Data Aging business function (DAAG\_DATA\_AGING ).

### Context

This option supports all databases that support data aging. By default, Data Preview displays the most frequently accessed data. For more information about data aging, see Data Aging under Related Information.

### Procedure

Choose the *Data Aging* button to switch the option off or on.

If you switch off the Data Aging option, Data Preview displays the less frequently used data (cold data).

## 5.1.21.6 Viewing Logs

### Prerequisites

You have already opened the Data Preview tool for the ABAP Data Dictionary tables, Views, external Views, or ABAP DDL Sources.

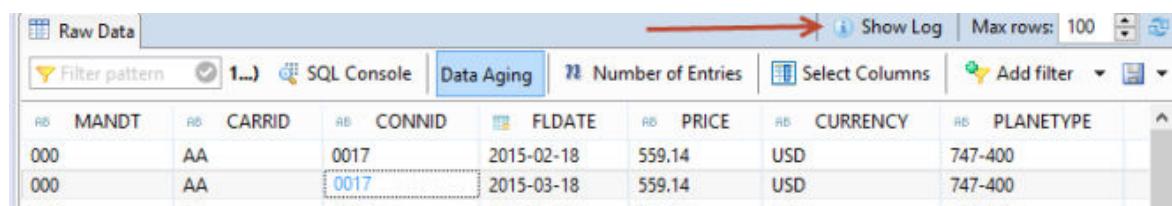
### Context

The Data Preview tool generates logs containing SQL queries. You can reuse queries from logs in your application if required.

### Procedure

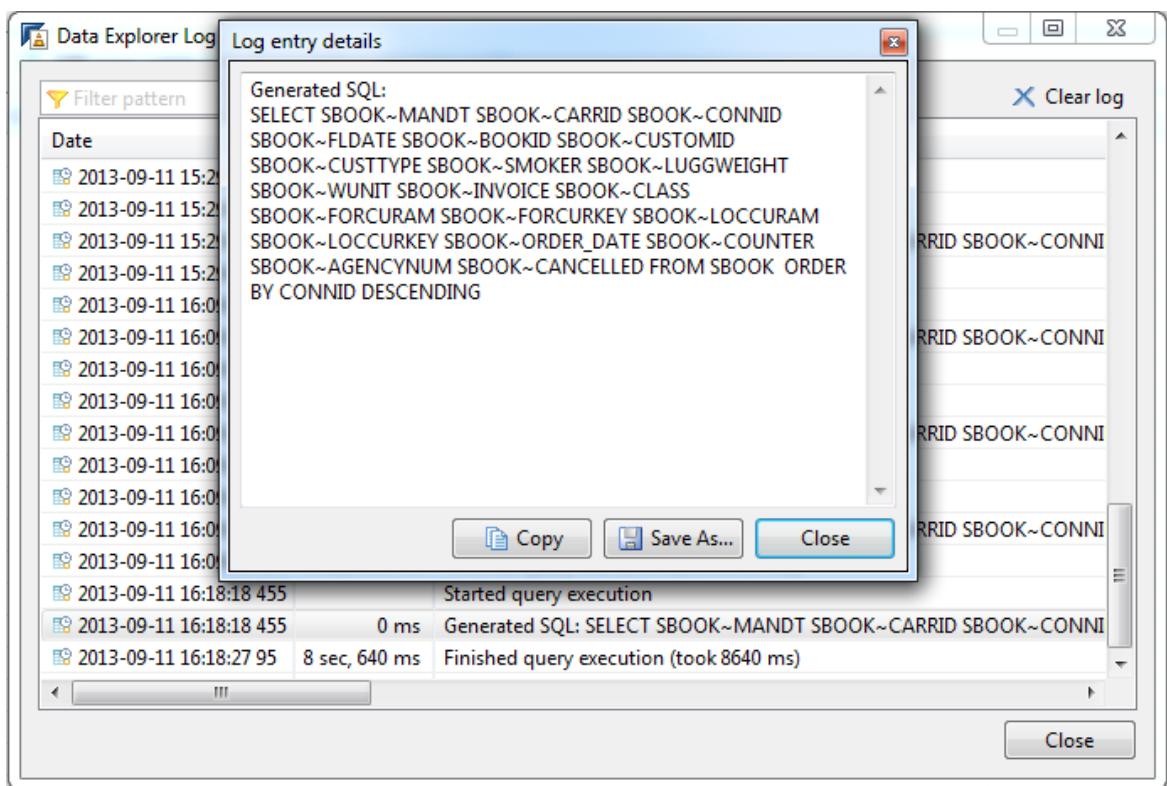
1. In the top-right of the Data Preview tool, choose the *Show Log* button.

The *Data Explorer Log* window appears and lists the log entries.



MANDT	CARRID	CONNID	FLDATE	PRICE	CURRENCY	PLANETYPE
000	AA	0017	2015-02-18	559,14	USD	747-400
000	AA	0017	2015-03-18	559,14	USD	747-400

2. To copy a log entry, double-click an entry.
3. In the *Log entry details* window, you can either copy or save a log entry.



## Related Information

[Accessing Data Preview \[page 468\]](#)  
[Setting Result-Set Size \[page 470\]](#)  
[Configuring Columns \[page 471\]](#)  
[Adding or Removing Filters \[page 473\]](#)  
[Viewing Distinct Column Values \[page 481\]](#)  
[Saving Result Set \[page 482\]](#)

### 5.1.21.7 Viewing Distinct Column Values

#### Prerequisites

You have already opened the Data Preview tool for the ABAP Data Dictionary tables, Views, external Views, or ABAP DDL Sources.

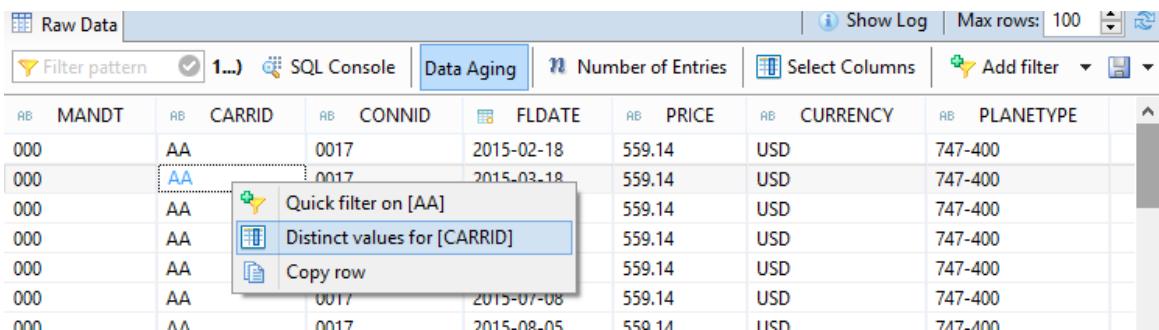
## Context

The Data Preview tool enables listing of unique or distinct values in a column that contains duplicate values.

## Procedure

1. Choose a column value.
2. In the context menu, choose *Distinct Values for*.

A dialog displays records with the list of all distinct values.



The screenshot shows a data preview table with columns: MANDT, CARRID, CONNID, FLDAT, PRICE, CURRENCY, and PLANETYPE. A row for CARRID 'AA' is selected. A context menu is open over this row, with the 'Distinct values for [CARRID]' option highlighted. Other options in the menu include 'Quick filter on [AA]' and 'Copy row'.

MANDT	CARRID	CONNID	FLDATE	PRICE	CURRENCY	PLANETYPE
000	AA	0017	2015-02-18	559.14	USD	747-400
000	AA	0017	2015-03-18	559.14	USD	747-400
000	AA	0017	2015-03-18	559.14	USD	747-400
000	AA	0017	2015-07-08	559.14	USD	747-400
000	AA	0017	2015-08-05	559.14	USD	747-400
000	AA	0017	2015-08-05	559.14	USD	747-400

## Related Information

[Accessing Data Preview \[page 468\]](#)

[Setting Result-Set Size \[page 470\]](#)

[Configuring Columns \[page 471\]](#)

[Adding or Removing Filters \[page 473\]](#)

[Viewing Logs \[page 480\]](#)

[Saving Result Set \[page 482\]](#)

## 5.1.21.8 Saving Result Set

### Prerequisites

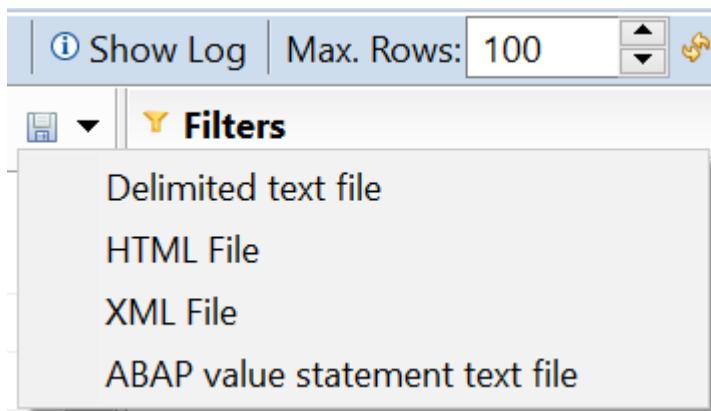
You have already opened the Data Preview tool for the ABAP Data Dictionary tables, Views, external Views, or ABAP DDL Sources.

## Context

The Data Preview tool provides different file formats to save analyzed records.

## Procedure

1. In the top-right of the Data Preview tool, identify the (Save as file) icon.



2. Choose the save icon.

The following file options appear:

- Delimited text file
- HTML File
- XML File
- ABAP value statement text file

3. Choose the required file option to save the result set.

## Related Information

[Using the Add Filter Button \[page 474\]](#)

[Setting Result-Set Size \[page 470\]](#)

[Configuring Columns \[page 471\]](#)

[Adding or Removing Filters \[page 473\]](#)

[Viewing Logs \[page 480\]](#)

[Viewing Distinct Column Values \[page 481\]](#)

## 5.1.21.9 Copying Rows as ABAP Value Statement

### Prerequisites

You have opened the Data Preview tool for the ABAP Data Dictionary tables, Views, external Views, or ABAP DDL Sources.

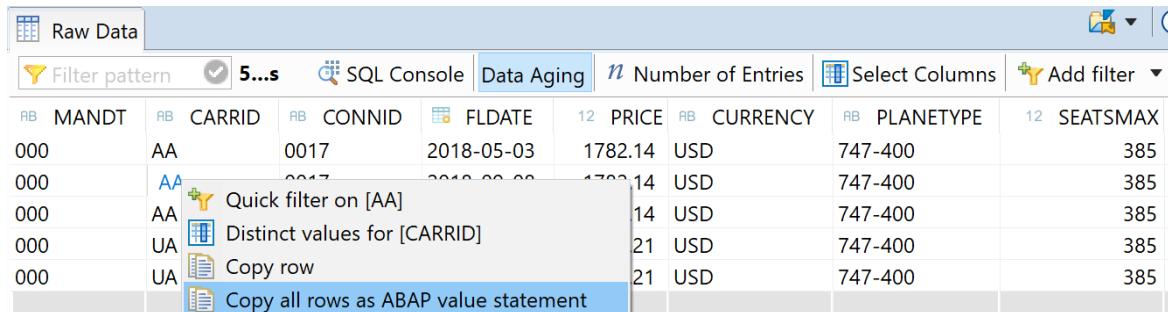
### Context

You can copy all the rows as an ABAP value statement and use it in the ABAP code to work with internal tables.

### Procedure

1. Right-click on the data table.
2. In the context menu, choose *Copy all rows as ABAP value statement*.

The rows are copied to the clipboard as ABAP value statement.



RB	MANDT	RB	CARRID	RB	CONNID	FLDATE	12	PRICE	RB	CURRENCY	RB	PLANETYPE	12	SEATSMAX
000		AA	0017			2018-05-03	1782.14	USD		747-400			385	
000		AA	0017			2018-02-22	1782.14	USD		747-400			385	
000		AA					14	USD		747-400			385	
000		UA					21	USD		747-400			385	
000		UA					21	USD		747-400			385	

You can use the generated source code in the ABAP code to work with internal tables.

```
► ZSFLIGHT_DATA_PREVIEW ►
1  *&-----*
2  *& Report zsflight_data_preview
3  *&-----*
4  *&
5  *&-----*
6  REPORT zsflight_data_preview.
7
8  data lt_sflight type STANDARD TABLE OF sflight.
9
10 lt_sflight = VALUE #( ( MANDT = '000' CARRID = 'AA' CONNID = '0017' FLDATE = '20180503' PRICE = '1782.14' CURRENCY = 'USD' PLANETYPE = '')
11   ( MANDT = '000' CARRID = 'AA' CONNID = '0017' FLDATE = '20180908' PRICE = '1782.14' CURRENCY = 'USD' PLANETYPE = '')
12   ( MANDT = '000' CARRID = 'AA' CONNID = '0017' FLDATE = '20181111' PRICE = '1782.14' CURRENCY = 'USD' PLANETYPE = '')
13   ( MANDT = '000' CARRID = 'UA' CONNID = '3517' FLDATE = '20180602' PRICE = '1970.21' CURRENCY = 'USD' PLANETYPE = '')
14   ( MANDT = '000' CARRID = 'UA' CONNID = '3517' FLDATE = '20181008' PRICE = '1970.21' CURRENCY = 'USD' PLANETYPE = '')
15 ).|
```

## 5.1.21.10 Working with SQL Console

SQL Console allows you to write an SQL statement and analyze the query performance.

### Context

SQL Console uses the most current query used in Data Preview. SQL Console supports:

- only new ABAP SQL syntax
- most of the ABAP source code features while working with queries. The table below lists features specific to SQL Console and how to use them:

Feature	How to use?
Check	<p>In SQL Console, choose Check. This feature verifies the syntax of a query.</p> <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;"><p><b>i Note</b></p><p>SQL Console also performs an automatic syntax check and instantly highlights any errors in syntax. For proper functioning of the automatic syntax check, ensure that the option <i>Automatic Syntax check on all open ABAP source editors</i> is selected in Eclipse. This option is available under the Eclipse menu option  <i>Windows</i> <i>Preferences</i> </p></div>
Run	<p>In SQL Console, choose <i>Run</i> to execute a query. In an SQL statement with multiple queries, if you want to execute a specific query, highlight the query and choose <i>Run</i>.</p> <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;"><p><b>i Note</b></p><p>The <i>Run</i> drop-down menu contains following options:</p><ul style="list-style-type: none"><li>• <i>History</i>: This option lists previously executed queries in SQL Console. Use this option to remove specific queries from the list or to define the number of queries to be displayed in the list.</li><li>• <i>Clear History</i>: This option removes all the previously executed queries, except the current query, from the History list.</li><li>• <i>Add To Favorites</i>: This option allows you to add a query in the favorites section and reuse it for future reference.</li><li>• <i>Organize Favorites</i>: This option allows you to work with queries in your favorites. You can add, remove, or move a query in the list.</li></ul></div>
Data Aging	<p>The Data Aging option allows you to view the most frequently accessed data (hot data) in your database. This feature is available only for the HANA database. The Data</p>

Feature	How to use?
	Aging option does not appear if you open Data Preview in other database.
Max Rows	In SQL Console, enter a value for <i>Max Rows</i> . Data Preview considers only this value while displaying records. Maximum row value provided in the SQL statement is not considered.
Content Assist	In SQL Console, press <b>Ctrl</b> + <b>Space</b> .
Tooltip Support	In SQL Console, press <b>F2</b> on any DDIC artifact or DDIC component.
Query Performance	SQL Console is split into two windows. The top window provides you the option of entering an SQL statement and the bottom window provides query related statistics.

## Procedure

1. In the *Project Explorer* view, choose an ABAP project.
2. In the context menu of the selected project, choose *SQL Console*.

The SQL Console appears and displays the executed query, query results, and query statistics.

### 5.1.22 Previewing Analytical Queries

You can preview the data of analytical queries with the Preview Analytical Queries app.

## Prerequisites

- You have created an ABAP cloud project. For information, see [Creating an ABAP Cloud Project \[page 118\]](#).
- You have added ABAP packages required for your work in the favorite list. For information, see [Adding a Favorite Package \[page 137\]](#).
- You have created a CDS based analytical query.

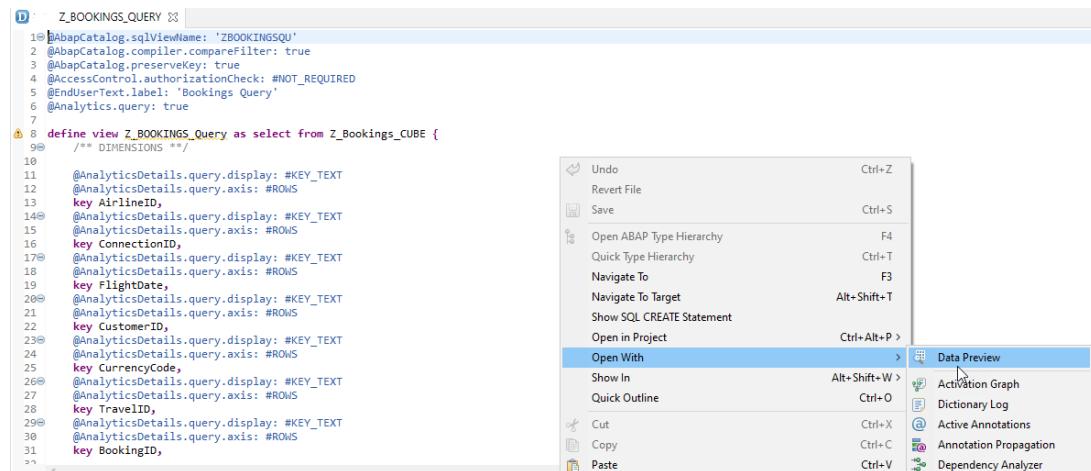
## Procedure

1. In the *Project Explorer* view, choose the required ABAP project.
2. Choose the required ABAP package.
3. Once the CDS View is annotated with `@Analytical.query: true`, you can open the app by right-clicking on CDS view listed in *Package Explorer*,  *Open With Data Preview*.

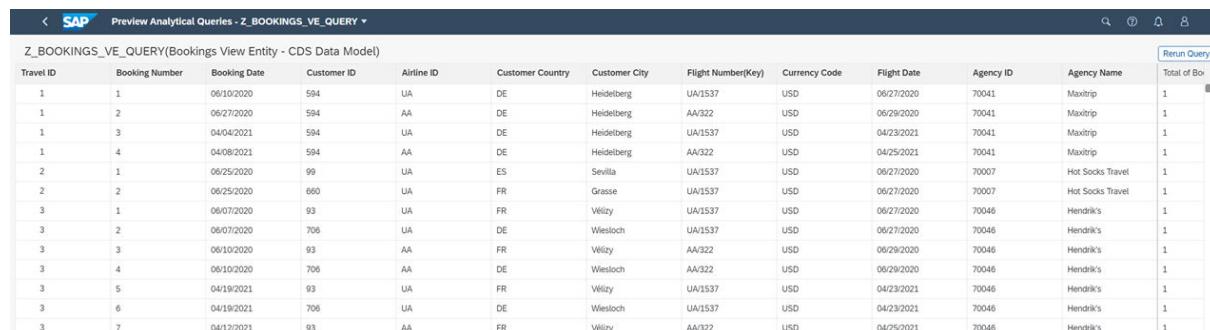
4. You can also open the app by right-clicking on CDS View editor and choose  **Open With > Data Preview**, or by pressing F8 in the CDS View editor.

## Results

The Preview Analytical Queries app opens in your browser.



Open with Data Preview



Z_BOOKINGS_VE_QUERY(Bookings View Entity - CDS Data Model)											Run Query	
Travel ID	Booking Number	Booking Date	Customer ID	Airline ID	Customer Country	Customer City	Flight Number(Key)	Currency Code	Flight Date	Agency ID	Agency Name	Total of B0
1	1	06/10/2020	594	UA	DE	Heidelberg	UA1537	USD	06/27/2020	70041	Maxtrip	1
1	2	06/27/2020	594	AA	DE	Heidelberg	AA322	USD	06/29/2020	70041	Maxtrip	1
1	3	04/04/2021	594	UA	DE	Heidelberg	UA1537	USD	04/23/2021	70041	Maxtrip	1
1	4	04/08/2021	594	AA	DE	Heidelberg	AA322	USD	04/25/2021	70041	Maxtrip	1
2	1	06/25/2020	99	UA	ES	Sevilla	UA1537	USD	06/27/2020	70007	Hot Socks Travel	1
2	2	06/25/2020	660	UA	FR	Grasse	UA1537	USD	06/27/2020	70007	Hot Socks Travel	1
3	1	06/07/2020	93	UA	FR	Vézelay	UA1537	USD	06/21/2020	70046	Hendrik's	1
3	2	06/07/2020	706	UA	DE	Westloch	UA1537	USD	06/21/2020	70046	Hendrik's	1
3	3	06/10/2020	93	AA	FR	Witzig	AA322	USD	06/29/2020	70046	Hendrik's	1
3	4	06/10/2020	706	AA	DE	Westloch	AA322	USD	06/29/2020	70046	Hendrik's	1
3	5	04/19/2021	93	UA	FR	Vézelay	UA1537	USD	04/23/2021	70046	Hendrik's	1
3	6	04/19/2021	706	UA	DE	Westloch	UA1537	USD	04/23/2021	70046	Hendrik's	1
3	7	04/12/2021	93	AA	FR	Vézelay	AA322	USD	04/25/2021	70046	Hendrik's	1

Preview Analytical Queries app

## 5.1.23 Getting Feeds

### Context

ABAP Development Tools (ADT) offers you a variety of **feeds from the ABAP repository**, including these:

- [ABAP Runtime Errors and Short Dumps \[page 46\]](#)  
The following feeds are created by default:
  - Abortions related to your user
  - Abortions that occurred in development objects which are assigned to your user
- System messages, for example, from the system administrator about downtimes, and so on  
This feed is also created by default.

## Related Information

[Subscribing to Feeds from ABAP Repository \[page 488\]](#)

[Subscribing to an Atom or RSS Feed \[page 489\]](#)

[Editing Feeds \[page 490\]](#)

[Canceling a Feed \[page 490\]](#)

[Feed Reader View \[page 80\]](#)

### 5.1.23.1 Subscribing to Feeds from ABAP Repository

#### Context

Here is how to subscribe to various types of feeds from ABAP Repository.

**i** Note

When you create an ABAP project, some feeds are set up for you automatically.

#### Procedure

1. Open the *Feed Reader* view.
2. [Optional:] Type `Ctrl` + `3` to display available views, and select the *Feed Reader*, if it is not already visible.
3. Choose **Add feed from repository...** 

The *New Feed Query* view opens your *ABAP Projects* from the *Project Explorer* to show the repositories.

4. Choose an ABAP Project to display and choose from the available feeds.

You can see (and change, if you want to) the default refresh interval – the frequency with which ABAP Development Tools checks for new feed items.

Mark the *Display notification messages* check box to have a pop-up dialog notify you when there is new content for a feed. Notifications offer a nice feature: The dialog tells you how old the feed event is. That is useful, for example, for screening out old system messages if you open an ABAP project that you haven't used for a while.

You can [Selecting ABAP Runtime Errors for a Feed \[page 714\]](#) for an error feed.

The new feed is added to the *Feed Reader* view.

5. Select a feed to see the ABAP project from which the feed comes, the refresh interval, and other information.
6. Select a feed item to display the item in the right-hand pane of the *Feed Reader* view.
7. Double-click on an item to display the complete item or to navigate to the relevant editor.

### i Note

This navigation feature is not available for all feeds.

If you haven't chosen *Display notification messages*, then monitor the **# Unread** column in the *Feed Reader* view to see when new feed items have arrived.

## Related Information

[Getting Feeds \[page 487\]](#)

[Canceling a Feed \[page 490\]](#)

[Editing Feeds \[page 490\]](#)

[Subscribing to an Atom or RSS Feed \[page 489\]](#)

## 5.1.23.2 Subscribing to an Atom or RSS Feed

### Prerequisites

You have obtained the URL of the Atom feed from the provider of the feed.

### Context

In the ABAP Development Tools, you can subscribe to any feed that is published in Atom (Atom Syndication Format) or RSS (Really Simple Syndication) format.

### Procedure

1. Type `Ctrl` + `3` to display available views, and select the *Feed Reader*, if it is not already visible.
2. Click on the pull-down menu of *Add feed...*  in the *Feed Reader* view.
3. From the pull-down menu, choose *Add Atom / RSS feed from URL*.
4. In the following window, enter the URL of the Atom or RSS feed and choose *Finish*.

### Results

Your Atom/RSS feed appears in the *Feed Reader*. Click on an entry in the feed for a quick look at the content in the right frame of the *Feed Reader* view. Double-click on an entry to open it in its own window.

## Related Information

[Getting Feeds \[page 487\]](#)  
[Canceling a Feed \[page 490\]](#)  
[Editing Feeds \[page 490\]](#)  
[Subscribing to Feeds from ABAP Repository \[page 488\]](#)

### 5.1.23.3 Editing Feeds

#### Context

#### Procedure

1. Mark a feed in the *Feed Reader* view.
2. Choose **Edit Feed...** from the context menu of the feed.

You can change any of the selection specifications of the feed. Changes take effect immediately.

## Related Information

[Getting Feeds \[page 487\]](#)  
[Canceling a Feed \[page 490\]](#)  
[Subscribing to an Atom or RSS Feed \[page 489\]](#)  
[Subscribing to Feeds from ABAP Repository \[page 488\]](#)

### 5.1.23.4 Canceling a Feed

#### Procedure

To cancel a feed, or also to change the settings of an existing feed, mark the feed in the *Feed Reader* view. Then choose **Delete** from the context menu of the feed.

## Related Information

[Getting Feeds \[page 487\]](#)

[Editing Feeds \[page 490\]](#)

[Subscribing to an Atom or RSS Feed \[page 489\]](#)

[Subscribing to Feeds from ABAP Repository \[page 488\]](#)

## 5.1.24 Working with Bookmarks

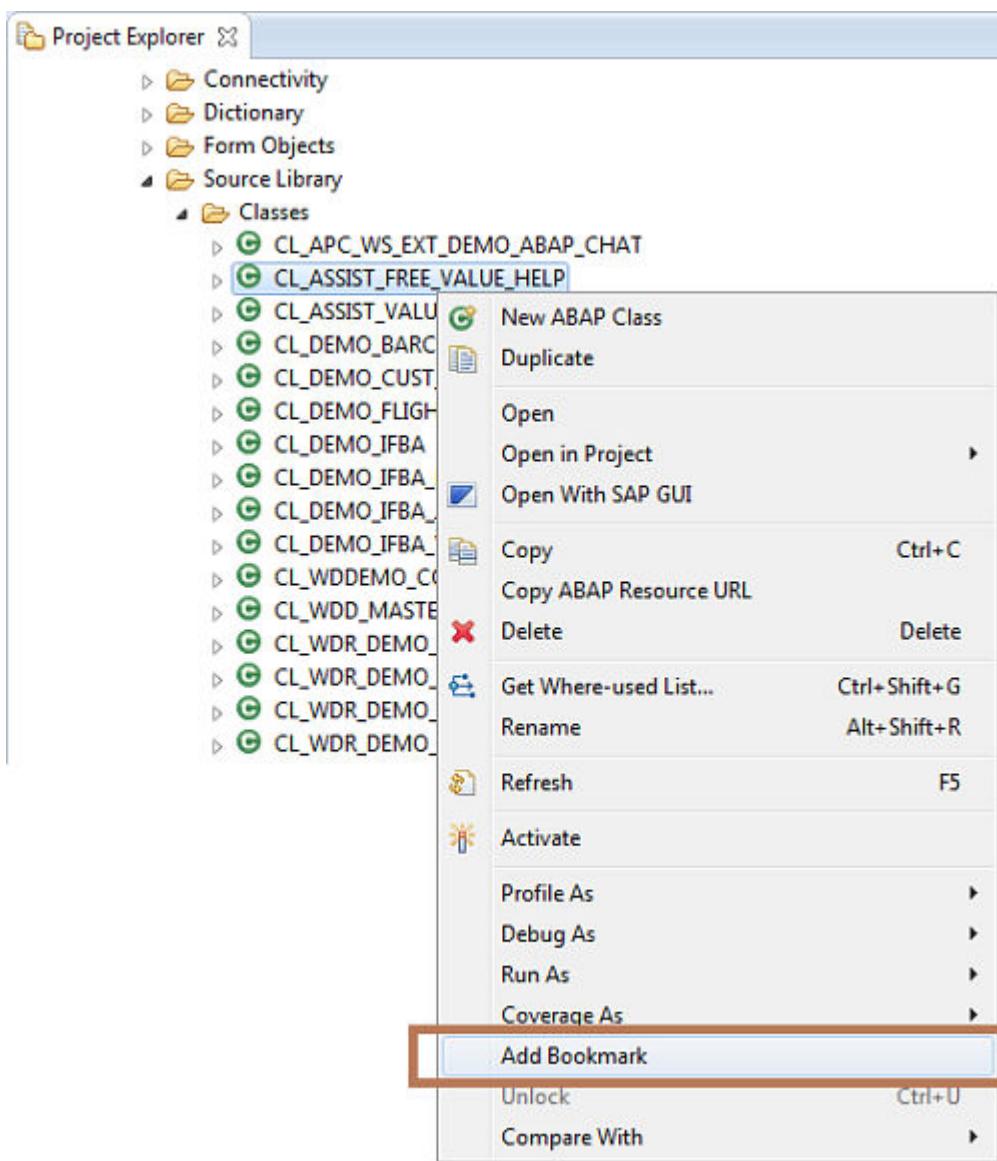
### Context

Use bookmark to navigate to ABAP Objects that are frequently required.

You can create bookmarks to any ABAP objects.

### Procedure

1. In **Project Explorer**, select the corresponding development object.
2. In context menu, select **Add Bookmark**



## Results

The **Bookmarks** view displays all bookmarks in the ABAP Workbench.

Bookmarks			
Task Repositories			
Task List			
Problems			
Properties			
Templates			
ABAP Communication Log			
2 items			
Description	Resource	Path	Location
Database Tables SAIRPORT	.wingui	/UIA_000_br_en	Unknown
Global Class CL_ASSIST_FREE_VALUE_HELP	cl_assist_free_value_help.apclass	/UIA_000_br_en/.adt/classlib/classes/cl_assist_free...	line 1

## 5.1.25 Working with Authorization Objects and Fields

### Context

To protect certain activities in your business processes and to enable authorization checks in your area, you can create authorization fields and bundle them in authorization objects.

As a next step, you define default authorizations on service level when you use the authorization objects in your implementation. Finally, when you create an app, the default authorizations of the included services are added automatically and you can adjust them if required.

You need to complete the following steps:

### Procedure

1. [Defining Authorization Fields \[page 493\]](#)
2. [Defining Authorization Objects \[page 494\]](#)
3. [Maintaining Authorization Default Values \[page 495\]](#)
4. [Creating an App \[page 124\]](#)

### Related Information

[Authorization Fields and Objects \[page 114\]](#)

## 5.1.25.1 Defining Authorization Fields

Authorization fields represent the values to be tested during authorization checks.

### Prerequisites

A suitable data element and domain exist.

## Context

You can use the fields defined by SAP in your own authorization objects. If you create a new authorization object, you do not need to define your own fields. For example, you can use the SAP field `<ACTVT>` in your own authorization objects to represent a wide variety of actions in the system.

To create an authorization field, proceed as follows:

## Procedure

1. In the *Project Explorer*, select the relevant package node.
2. Open the context menu and choose  *New*  *Other ABAP Repository Object*  *Authorizations*  *Authorization Field*  to launch the creation wizard.
3. Enter the name of the authorization field. Field names must be unique and must begin with the letter Y or Z.
4. Assign a transport request.
5. Start the creation with *Finish*.
6. Assign a data element from the ABAP Dictionary to the field. You must define the data element by a domain.
7. If desired, attach a check table for the possible entries. The link provides possible field values. You can also define a value range by way of the area with which a field is associated.
8. Save your entries.

## 5.1.25.2 Defining Authorization Objects

An authorization object groups up to 10 authorization fields that are related.

## Context

You create authorization objects to bundle authorization fields to be tested in authorization checks.

## Procedure

1. In the *Project Explorer*, select the relevant package node.
2. Open the context menu and choose  *New*  *Other ABAP Repository Object*  *Authorizations*  *Authorization Object*  to launch the creation wizard.

3. Enter a unique object name and a description. Object names must begin with the letter Y or Z in accordance with the naming convention for customer-specific objects.
4. Assign a transport request.
5. Start the creation with *Finish*.
6. Add the authorization fields that belong to the object. You can use own authorization fields or those that have been whitelisted by SAP, such as *Activity (ACTVT)*. The *Activity Field* is automatically selected when you create a new authorization object.
7. Define the authorization field that acts as activity field by selecting the *Activity Field* checkbox.
8. Define the permitted activities (such as *Change* or *Display*) and select the required access category (*Read*, *Write*, or *Value Help*).
9. Save your entries.

## 5.1.25.3 Maintaining Authorization Default Values

Default values represent the automatically assigned authorizations per object.

### Context

After having defined the authorization objects for your area, you can use them in your implementation when you create your service. On service level, you can classify the authorization objects by defining authorization default values for each field.

Please note that for OData V2 services, you need to maintain the values for IWSG.

### Procedure

1. Open the service binding and click the *Maintain Authorization Default Values* link. The system displays a list of all included authorization objects and the authorization fields per object.
2. Click the *Insert Before* or *Insert After* icon to include additional authorization objects in the list.
3. Click the *Delete Selected Item* icon to remove the assignment to an authorization object if required.
4. To change a default status, select a line in the authorization objects list and choose the required value from the dropdown list.

The following options are available:

- *Default with Field Values* (Selected by default)  
An authorization check is required, the values are taken over to the IAM app.
- *Default Without Field Values*  
An authorization check is required, the values are not taken over to the IAM app.
- *Default Inactive*  
Values have been maintained and are taken over but remain inactive.
- *No Default*

- No values have been maintained.
  - *Synchronize*
    - Allows to synchronize the standard values with authorization default values.
5. Define the required authorization values in the *From* and *To* rows of the *Authorization Default Values* list of each authorization field included in the respective authorization objects.
  6. Save your entries.

## 5.1.25.4 Defining Restriction Fields

Restriction fields are used to restrict access to a specific business object (for example *Organizational Area*) controlled by business roles.

### Prerequisites

A corresponding authorization field exists.

### Context

For each authorization field you want to expose and maintain in a business role, you need to create a corresponding restriction field and assign it to a restriction type.

### Procedure

1. In the *Project Explorer*, select the relevant package node.
2. Open the context menu and choose       to launch the creation wizard.
3. Enter the name of the restriction field. Restriction field names must be unique.
4. Assign a transport request.
5. Start the creation with *Finish*.
6. Assign an authorization field.
7. Select the *Supports Ranges* checkbox if you want the restriction field to support ranges.
8. Save your entries.

## 5.1.25.5 Defining Restriction Types

Restriction types bundle the available restriction fields to a logical definition (for example [Company Code](#)) controlled by business roles.

### Prerequisites

At least one corresponding restriction field and authorization object exist.

### Context

For the authorization objects you want to expose and maintain in a business role, you need to create a corresponding restriction type and assign the previously created restriction fields. Then you need to assign the authorization object that defines the required authorizations. As a last step, you need to assign the restriction type to the required business catalog. For more information, see [Creating a Business Catalog \[page 126\]](#).

Please note that as soon as the restriction type is defined, all the corresponding authorizations in the IAM app are overwritten with the values defined in the business role.

To create a restriction type, proceed as follows:

### Procedure

1. In the [Project Explorer](#), select the relevant package node.
2. Open the context menu and choose  [New](#)  [Other](#)  [ABAP Repository Object](#)  [Cloud Identity and Access Management](#)  [Restriction Type](#)  [Next](#)  to launch the creation wizard.
3. Enter the name of the restriction type. Restriction type names must be unique.
4. Assign a transport request.
5. Start the creation with [Finish](#).
6. Assign one or more restriction fields.
7. Assign one or more authorization objects.
8. Save your entries.

## 5.2 Ensuring Quality of ABAP Code

### Context

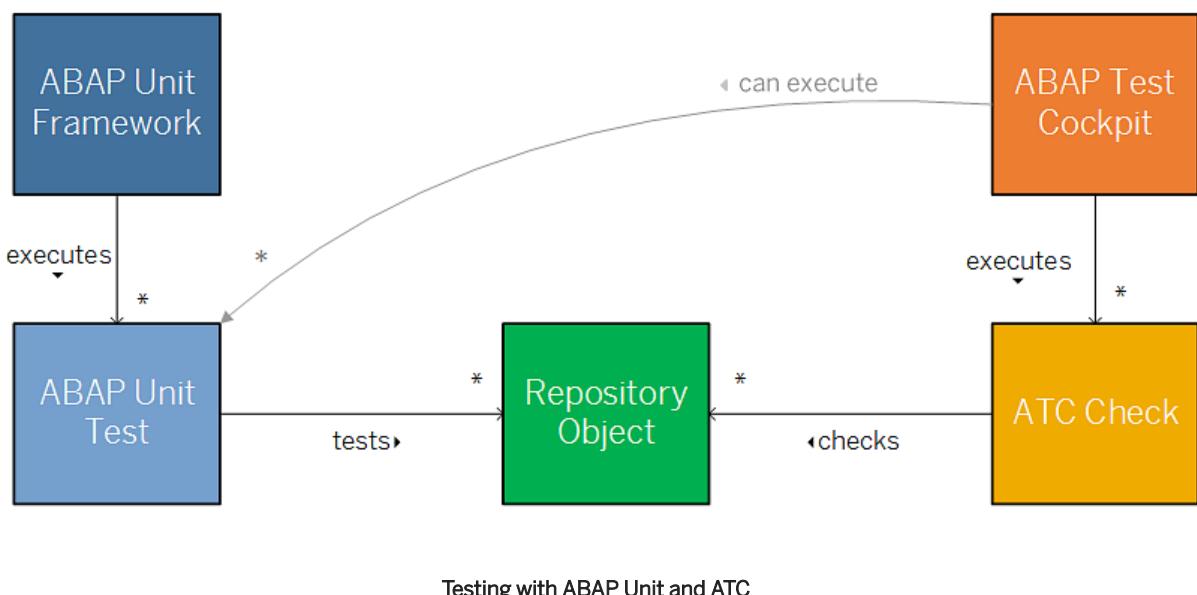
There are two central tools in ADT for ensuring the quality of a repository object: ABAP Unit Framework and ATC (ABAP Test Cockpit). Although these tools are self-contained, there is a strong integrational relationship between them. ATC allows you to execute ABAP Unit tests together with other quality checks.

ABAP Unit Framework checks the functional correctness of the tested object and verifies the intended program behavior at runtime.

ATC Framework measures the quality of the ABAP code and detects respective issues. There is a variety of checks, you can execute within the ATC Framework, for example:

- Security check
- Accessibility check
- Usability check
- Functionality check
- ABAP Unit tests

You can configure, which tests you want to execute in ATC check.



### Transport Release

When you want to release a transport request, ATC checks are executed to verify the specified quality attributes.

Depending on the used ATC check variant, this includes execution of unit tests. Be aware that unit test execution might be switched off or restricted to a maximum duration of 'medium' or 'short'.

### i Note

The concept of [test relations \[page 550\]](#) is **not** considered during ATC check. This means that ATC doesn't search for the foreign tests and doesn't execute them.

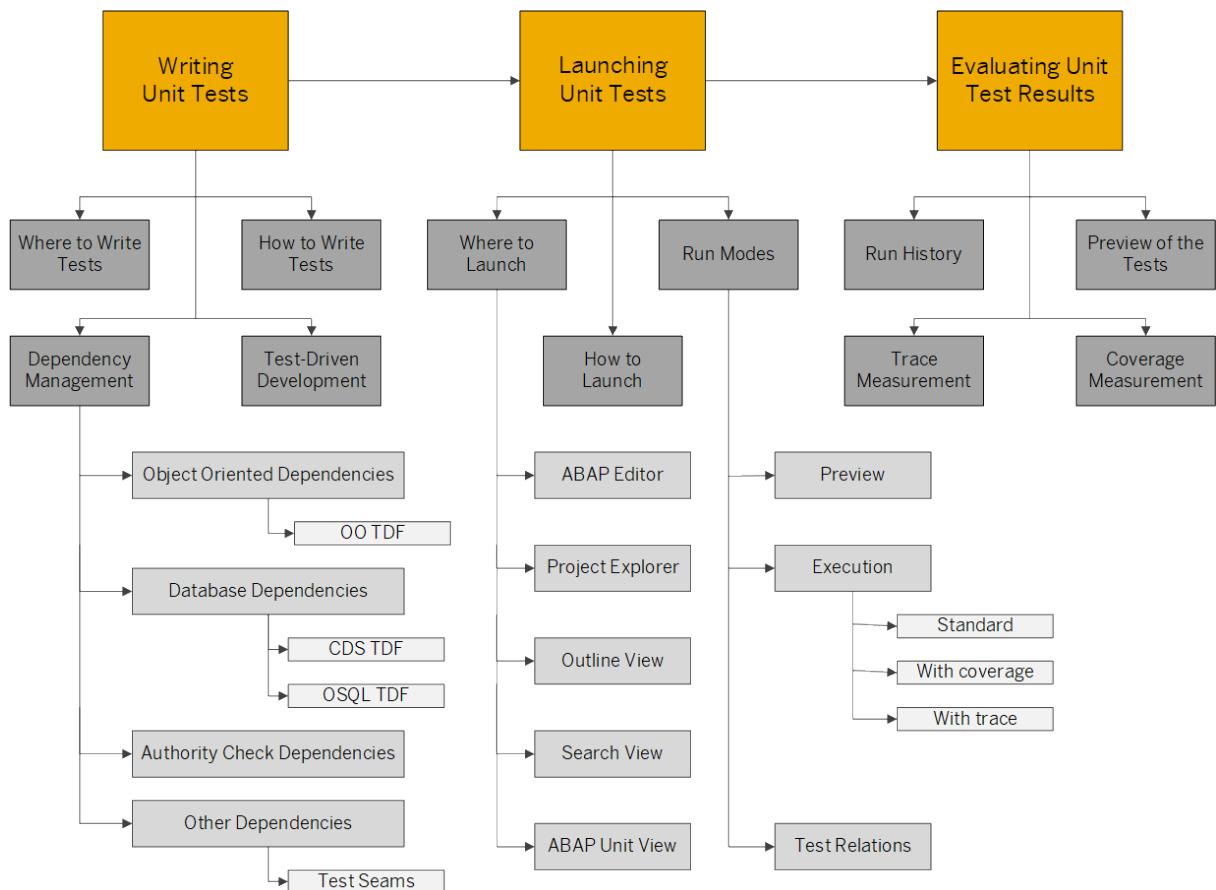
## Related Information

[Checking Quality of ABAP Code with ATC \[page 579\]](#)

### 5.2.1 Unit Testing with ABAP Unit

Writing ABAP Unit tests is the way to provide high quality software, which can be easily evolved over time without introducing regressions. In methodologies, like extreme programming and test driven development, the role of unit testing is even more important.

ABAP unit is the state-of-the-art unit testing framework for ABAP. It's embedded into the ABAP programming language which supports you in writing unit tests. In ADT you have various possibilities to execute the unit tests and to evaluate the results concerning functional correctness and code coverage.



- [Writing ABAP Unit Tests \[page 501\]](#)
- [Launching ABAP Unit Tests \[page 551\]](#)
- [Evaluating ABAP Unit Test Results \[page 569\]](#)
- [Where to Write ABAP Unit Test Classes \[page 502\]](#)
- [How to Write ABAP Unit Test Classes \[page 505\]](#)
- [Managing Dependencies with ABAP Unit \[page 509\]](#)
- [Test-Driven Development with ABAP Unit \[page 506\]](#)
- [Managing Object Oriented Dependencies with ABAP Unit \[page 512\]](#)
- [Managing Database Dependencies with ABAP Unit \[page 515\]](#)
- [Managing Other Dependencies with ABAP Unit \[page 548\]](#)
- [ABAP Object Oriented Test Double Framework \[page 514\]](#)
- [ABAP CDS Test Double Framework \[page 516\]](#)
- [ABAP SQL Test Double Framework \[page 539\]](#)
- [Managing Other Dependencies with ABAP Unit \[page 548\]](#)
- [Launching ABAP Unit Tests \[page 551\]](#)
- [Launching ABAP Unit Tests \[page 551\]](#)
- [Run Modes of ABAP Unit Tests \[page 564\]](#)
- [Launching ABAP Unit Tests from ABAP Editor \[page 553\]](#)

- [Launching ABAP Unit Tests from the Project Explorer \[page 552\]](#)
- [Launching ABAP Unit Tests from the Outline View \[page 555\]](#)
- [Launching ABAP Unit Tests from the Search View \[page 557\]](#)
- [Launching ABAP Unit Tests from the ABAP Unit View \[page 556\]](#)
- [Launching a Preview of the ABAP Unit Tests \[page 559\]](#)
- [ABAP Unit Launch Dialog \[page 565\]](#)
- [Launching ABAP Unit with Test Relations \[page 561\]](#)
- [Using the History to Switch Between Results of ABAP Unit Tests \[page 574\]](#)
- [Evaluating Preview Results of ABAP Unit Tests \[page 575\]](#)
- [Evaluating ABAP Unit Trace Results \[page 573\]](#)
- [Evaluating ABAP Unit Code Coverage Results \[page 571\]](#)
- [Managing Dependencies on ABAP Authority Checks with ABAP Unit \[page 545\]](#)

## Related Information

[Unit Testing in ABAP \[page 71\]](#)

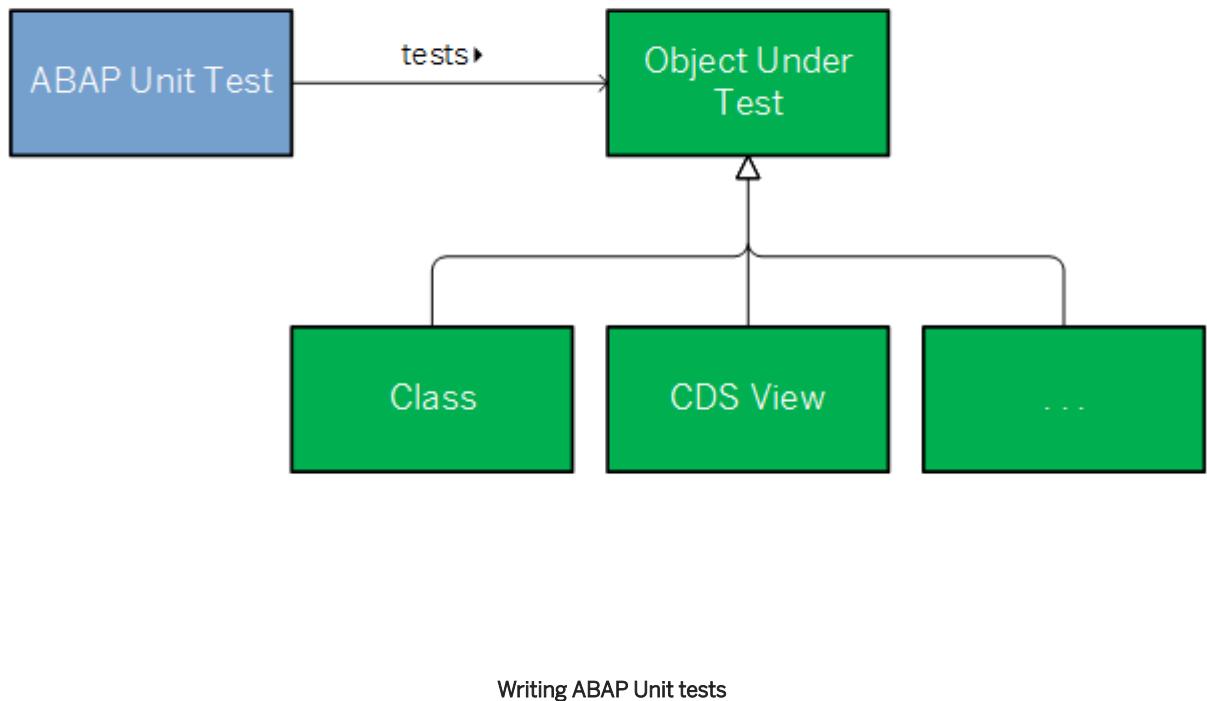
[Writing ABAP Unit Tests \[page 501\]](#)

[Launching ABAP Unit Tests \[page 551\]](#)

[Evaluating ABAP Unit Test Results \[page 569\]](#)

### 5.2.1.1 Writing ABAP Unit Tests

ABAP Unit is a framework for executing automated tests written in local classes. These tests usually check the functional correctness of ABAP objects like classes and CDS views.



## Related Information

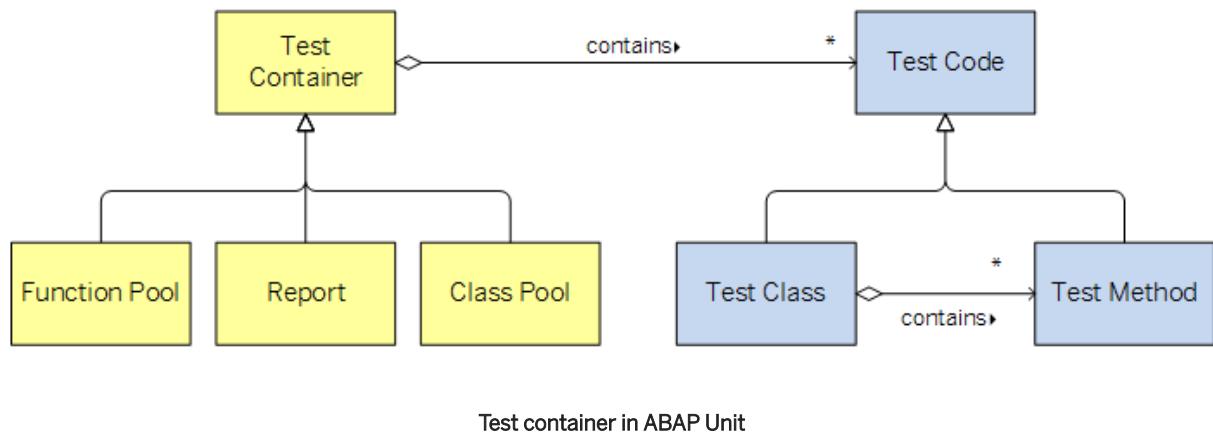
- [How to Write ABAP Unit Test Classes \[page 505\]](#)
- [Where to Write ABAP Unit Test Classes \[page 502\]](#)
- [Test-Driven Development with ABAP Unit \[page 506\]](#)
- [Managing Dependencies with ABAP Unit \[page 509\]](#)

### 5.2.1.1.1 Where to Write ABAP Unit Test Classes

Depending on the program object with which you are working, there are different recommendations as to where you should implement ABAP Unit test classes.

## Context

In classes, for example, there is a special include for ABAP Unit test classes. Function modules also provide a special include for ABAP Unit tests.



This table shows you how ABAP Development Tools supports you in creating and navigating to these ABAP Unit includes, where they are available, and where to write your ABAP Unit source code.

Type of Repository Object	Where to Write Your ABAP Unit Test Classes
ABAP Classes	<p>Just choose the <i>Test Classes</i> tab at the bottom of the ABAP class editor.</p> <p>ABAP Development Tools creates the include for ABAP Unit tests automatically and you can program your ABAP unit test classes here.</p>

## ABAP Function Groups

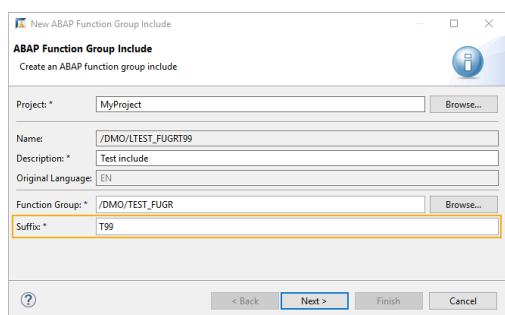
- Create the '*T99*' include for ABAP Unit classes from the embedded Java GUI or SAP GUI:
  1. Open the function module in the ABAP **Object Navigator** (transaction SE80).
  2. Start editing mode (**Ctrl** + **F1**).
  3. In the main menu select ► **Goto** ► **Local Test Classes** ► 
  4. The creation dialog appears. Confirm the creation of the empty include for the test class by selecting **Yes**.
  5. In the appeared dialog select **Continue**.

### Result

The *<function\_group\_name>T99* include is created for ABAP unit test classes. The wizard also generates stubs for the test methods that you specify.

In ADT, you can navigate to the '*T99*' include and edit the ABAP unit classes by opening the function group from Project Explorer. This opens the master program of the function group. You can then navigate to the ABAP unit source code in the *T99* include.

- Create the test include in ADT:
  1. Open the function group in ADT.
  2. Select the function group in Project Explorer. From context menu select ► **New** ► **ABAP Function Group Include** 
  3. In the Dialog enter **Description**. Enter *T99* in the **Suffix** field.



## ABAP Programs, Reports, and Subroutine Pools

Create ABAP unit local classes and test methods at the end of the ABAP program.

There is no reserved include for test classes. You can locate test classes in the report or a separate include.

### Note

Be aware that [ABAP Test Seams \[page 548\]](#) can only be applied within class pools and function groups.

## Related Information

[Writing ABAP Unit Tests \[page 501\]](#)

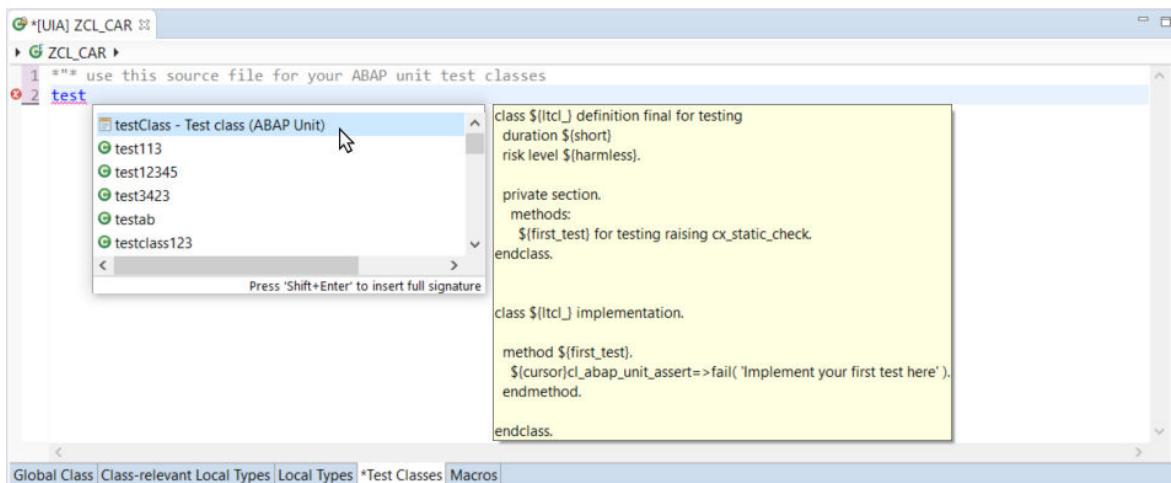
[How to Write ABAP Unit Test Classes \[page 505\]](#)

[Split Screen Editing \[page 508\]](#)

### 5.2.1.1.2 How to Write ABAP Unit Test Classes

To write ABAP Unit tests, create tests methods in local test classes.

1. Open the [Test Classes](#) tab at the bottom of your ABAP Editor.
2. Write a test class using the predefined [language elements](#). Alternatively, you can insert a template for ABAP Unit test classes. Just type `test` in your editor, then press `Ctrl` + `Space` for a code-completion list.



Writing test classes with a template

3. In the pop-up menu with suggested entries, select [testClass - Test class \(ABAP Unit\)](#) to insert the template.

```

1 *** use this source file for your ABAP unit test classes
2@class ltcl_definition final for testing
3 duration short
4 risk level harmless.
5
6 private section.
7   methods:
8     first_test for testing raising cx_static_check.
9   endclass.
10
11
12@class ltcl_implementation.
13
14@  method first_test.
15    cl_abap_unit_assert=>fail( 'Implement your first test here' ).
16  endmethod.
17
18 endclass.

```

#### A template for test classes

You can now create the first test method.

The static methods of the class `CL_ABAP_UNIT_ASSERT` check the test assumptions.

### Related Information

[Writing ABAP Unit Tests \[page 501\]](#)

[Where to Write ABAP Unit Test Classes \[page 502\]](#)

[Test-Driven Development with ABAP Unit \[page 506\]](#)

### 5.2.1.1.3 Test-Driven Development with ABAP Unit

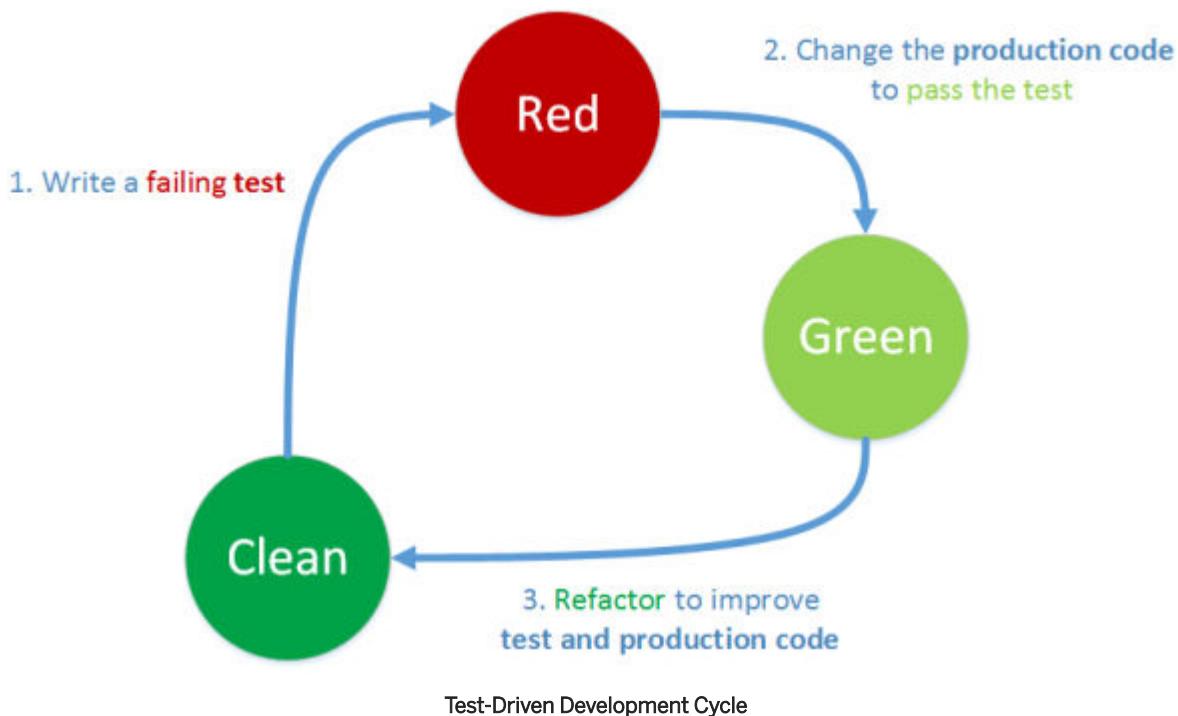
Test-Driven Development (TDD) is a way to develop software driven by automated tests which are written earlier than the production code itself. ABAP Unit provides excellent support for TDD in ABAP.

### Context

ABAP Unit is integrated into the ABAP language, which has the following benefits:

- ABAP Unit testing is always available. You do not need to switch to a special operation mode or use a specially prepared server.
- You can run tests quickly and easily right from the development environment.
- You can navigate back and forth between a test class and the production code.

ADT supports you in the following cycle with the ABAP Unit Test Framework together with other tools such as Coverage Measurements, Quickfixes, Refactoring Tools, Dependency Isolation Frameworks, and Splitscreen Editing.



#### Supporting Tools

ABAP Unit	Using ABAP Unit and ADT you can easily <a href="#">write [page 501]</a> unit tests, <a href="#">execute [page 551]</a> them any time you want, and <a href="#">evaluate [page 569]</a> the results with respect to functional correctness and code coverage.
Quickfixes	<p>Just call a production method that does not yet exist and let the quickfix <a href="#">Create Method from Call [page 414]</a> create the entire method signature together with an implementation stub.</p> <p>To write a new test, you can copy an existing test, give it a new name, and let the quickfix <a href="#">Create Method Definition [page 412]</a> create the method definition.</p>
Refactoring Tools	<p>Refactoring tools complete the toolset and let you quickly improve the structure of your code.</p> <p><a href="#">Refactoring ABAP Source Code [page 342]</a></p>
Dependency Isolation Support	<p>Especially in the context of TDD, unit tests need to be stable and fast. Therefore, it is essential to replace test-unfriendly code with test-friendly code when running tests.</p> <p>For this purpose, you can make use of the following:</p> <ul style="list-style-type: none"> <li>• <a href="#">ABAP Test Seams [page 548]</a></li> <li>• <a href="#">OO Test Double Framework [page 512]</a> (see class <code>CL_ABAP_TESTDOUBLE</code>)</li> <li>• <a href="#">ABAP SQL Test Double Framework [page 539]</a> (see class <code>CL_OSQL_TEST_ENVIRONMENT</code>)</li> <li>• <a href="#">ABAP CDS Test Double Framework [page 516]</a> (see class <code>CL_CDS_TEST_ENVIRONMENT</code>)</li> </ul>
Coverage Measurement	Making coverage measurements while executing the unit tests helps you to find production code which is not motivated by an existing unit test and helps you to get back on track.
Splitscreen Editing	Use <a href="#">Split Screen Editing [page 508]</a> to edit test code and production code side-by-side.

## 5.2.1.1.3.1 Split Screen Editing

### Context

You can switch to test code for ABAP classes by choosing the *Test Classes* tab at the bottom of ABAP Editor. To use ABAP Unit in Test-Driven Development, edit the source code and test the code side-by-side. You can configure the split screen editor as follows..

### Procedure

1. Open your ABAP object source code in ABAP editor.
2. Select **Window > Editor > Clone** from the ADT tool bar.

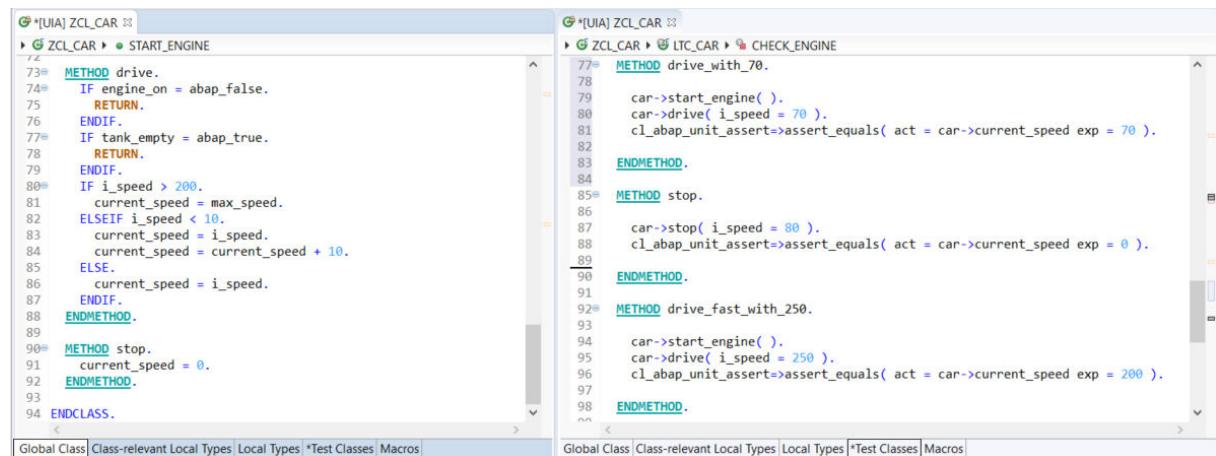
The ADT opens a second ABAP editor for your class. You can switch to the test class code, while the other editor displays the source code of the class.

3. Pull the class name tab of one of the editors to the right, so that you get a split screen editor.
4. Select the *Test Classes* tab of one of the editors.

Leave split-screen mode by pulling one of the split-screen class name tabs back up to the row of class name tabs at the top of the editor screen. Alternatively, close one of the editors.

### Results

You can now write test methods and implement your source code displayed side-by-side, following the Test-Driven Development programming concept.



```
72* METHOD drive.
73*   IF engine_on = abap_false.
74*     RETURN.
75*   ENDIF.
76*   IF tank_empty = abap_true.
77*     RETURN.
78*   ENDIF.
79*   IF i_speed > 200.
80*     current_speed = max_speed.
81*   ELSEIF i_speed < 10.
82*     current_speed = i_speed.
83*   ELSE.
84*     current_speed = i_speed.
85*   ENDIF.
86*   current_speed = i_speed.
87* ENDMETHOD.
88*
89* METHOD stop.
90*   current_speed = 0.
91* ENDMETHOD.
92*
93 ENDCCLASS.
```

```
77* METHOD drive_with_70.
78*   car->start_engine( ).
79*   car->drive( i_speed = 70 ).
80*   cl_abap_unit_assert=>assert_equals( act = car->current_speed exp = 70 ).
81*
82 ENDMETHOD.
83*
84 METHOD stop.
85*   car->stop( i_speed = 80 ).
86*   cl_abap_unit_assert=>assert_equals( act = car->current_speed exp = 0 ).
87*
88 ENDMETHOD.
89*
90 METHOD drive_fast_with_250.
91*   car->start_engine( ).
92*   car->drive( i_speed = 250 ).
93*   cl_abap_unit_assert=>assert_equals( act = car->current_speed exp = 250 ).
94*
95 ENDMETHOD.
```

Working with split screen editor

## Related Information

[Test-Driven Development with ABAP Unit \[page 506\]](#)

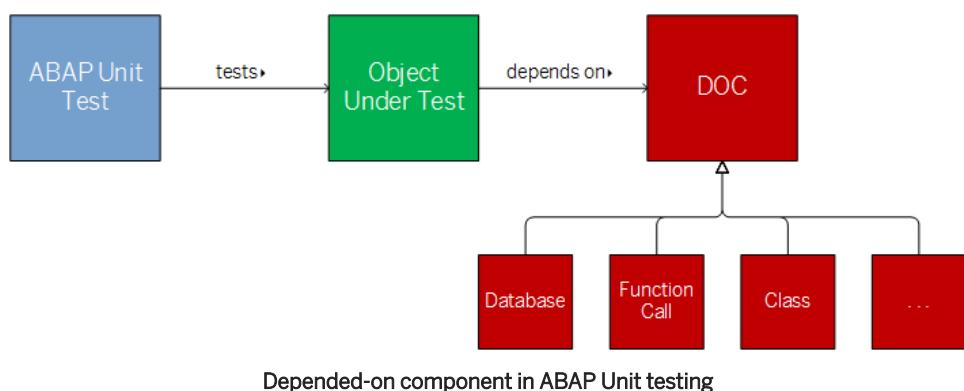
[How to Write ABAP Unit Test Classes \[page 505\]](#)

[Where to Write ABAP Unit Test Classes \[page 502\]](#)

[Launching ABAP Unit Tests \[page 551\]](#)

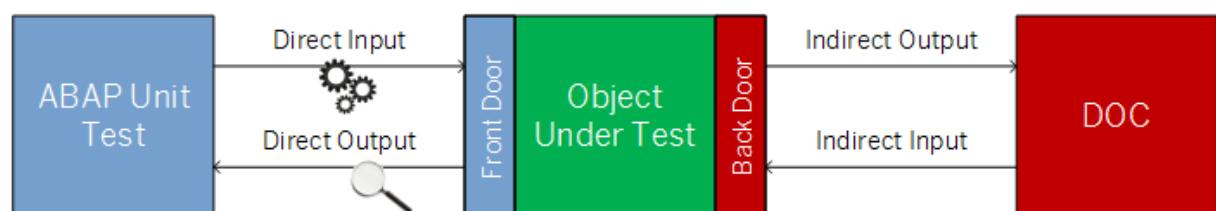
### 5.2.1.1.4 Managing Dependencies with ABAP Unit

ABAP Unit is used to test production code that is called **object under test**. The object under test may depend on other production code, known as the **depended-on component** or **DOC**. Examples of depended-on components are classes, function calls, and databases.



Depended-on component in ABAP Unit testing

A test may interact with the object under test directly using its **front door** or indirectly using its **back door**. The front door can be viewed as the public API of the object under test while the back door contains calls to any depended-on components.



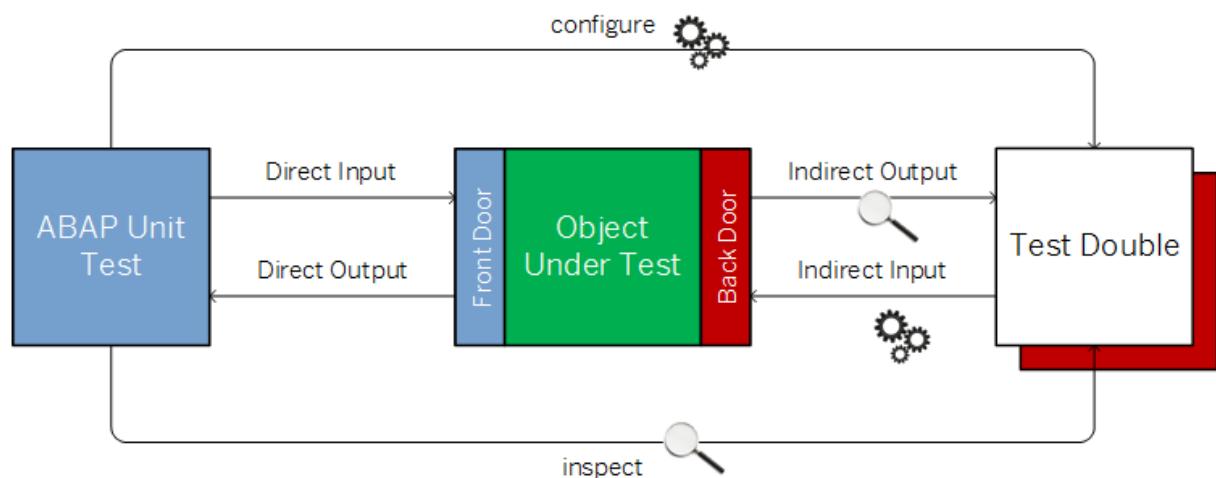
Interaction of the object under test using front and back doors.

It is relatively simple to control and observe what happens at the front door, but more complicated to do the same at the back door.

## Input and Output at the Front and Back Doors

	Input	Output
Direct (front door)	Direct calls to the object under test, including the parameters of these calls.	Responses received by the test from the object under test like return values or exceptions.
Indirect (back door)	Responses from the DOC, like return values or exceptions.	Calls from the object under test to the DOC, including the parameters of these calls.

**Test Doubles** are the key to get access to the backdoor. They replace the DOCs when tests are executed and help you to control the indirect input and to observe the indirect output.

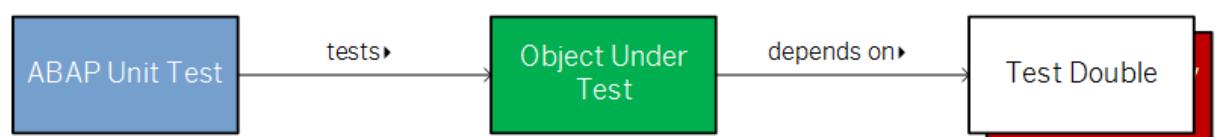


### Access to the backdoor with the test double

To use a test double you need to:

1. **Create** the test double
2. **Configure** the test double [optional]
3. **Inject** the test double (this means replacing the DOC with the test double)

As a result, the test double replaces the depended-on component and behaves in such a way that the test can rely on it.



### The result of replacing a DOC with a test double

ABAP provides several frameworks and concepts to get access to the back door. Which approach you choose, depends on the kind of DOC and what you want to control and observe.

- **Object Oriented Dependencies** (DOC = Class)
  - **HM TD**: Handmade Test Double Class
  - **OO TDF [page 514]**: ABAP Test Double Framework
- **Database Dependencies** (DOC = DB Table or CDS View)
  - **CDS TDF [page 516]**: CDS Test Double Framework (Read Access)
  - **OSQL TDF [page 539]**: ABAP SQL Test Double Framework (Read and Write Access)
- **Other Dependencies** (DOC = anything else)
  - **ATS [page 548]**: ABAP Test Seams.
- **Authority Check Dependencies**
  - **Authority Check Test Helper [page 545]**

Dependency Management Technique	Depended-On Component	Double
OO TDF	Object	Object
CDS TDF	Database table, CDS view	Database table
Test Seams	Statement	Statement
Authority Check Test Helper	Authority Object	Modified Authorizations

In the table below you can see what you can control and/or observe using the test double approach.

Purpose of Test Double	HM TD	OO TDF	CDS TDF	OSQL TDF	ATS	Authority Check Test Helper
Control indirect input (Stub)	✓	✓	✓	✓	✓	✓
Verify indirect output (Mock)	!	✓	✗	✗	✓	✗
Log indirect output (Spy)	✓	✗	✗	✗	✓	✗
Pretend existence (Dummy)	!	✓	n/a	n/a	✗	✗



Well supported



Possible, but not recommended (there is a better solution)



Not possible

n/a

Not applicable

## Related Information

[Managing Object Oriented Dependencies with ABAP Unit \[page 512\]](#)

[Managing Database Dependencies with ABAP Unit \[page 515\]](#)

[Managing Other Dependencies with ABAP Unit \[page 548\]](#)

[ABAP SQL Test Double Framework \[page 539\]](#)

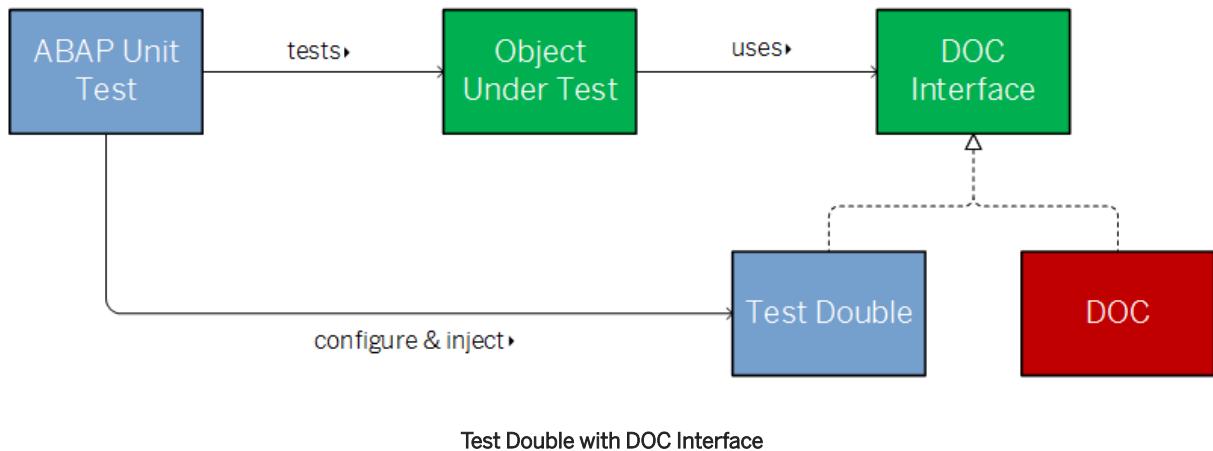
[Creating a Test Class Using a Wizard \[page 519\]](#)

### 5.2.1.1.4.1 Managing Object Oriented Dependencies with ABAP Unit

Whether you prefer handmade or tool-supported object-oriented test doubles, both approaches are based on the same principle. First, you need an ABAP class that implements the same interface as the DOC (depended-on component) you want to replace. Second, you need a mechanism to actually make the replacement. Use object-oriented test doubles if the object under test uses the DOC through an ABAP interface. In this case, the interface can be implemented by a test double class.

#### i Note

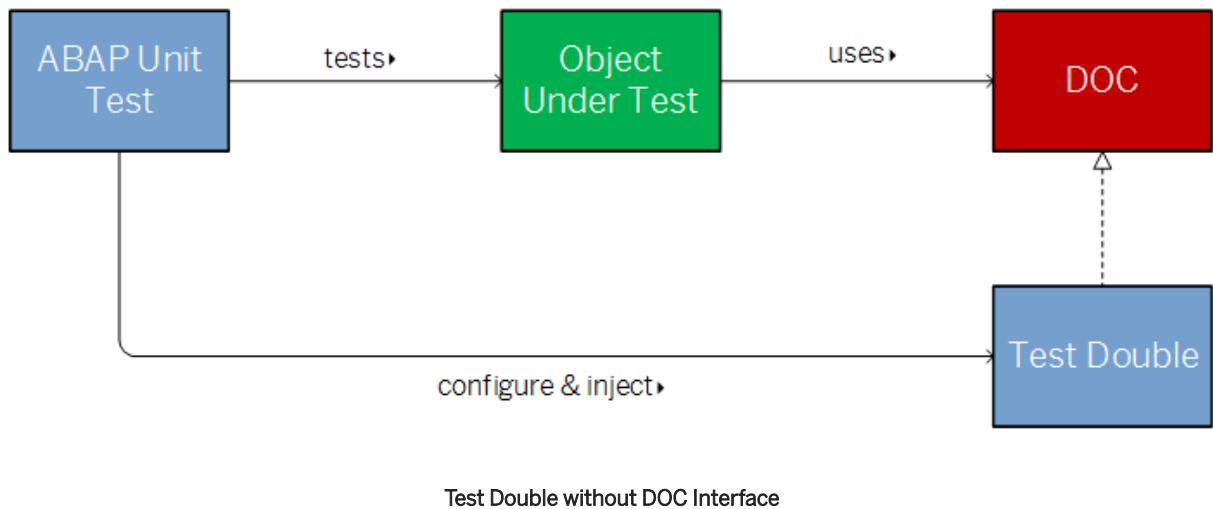
There is no need to implement all methods of the interface in the test double class. Use the keyword `partially implemented` and implement only those methods that you call when executing tests.



A slightly different approach is possible if the object under test directly uses the DOC. The DOC has to be a nonfinal ABAP class. In this case it is possible to implement a test-specific subclass as a test double.

**i Note**

Be aware that you can only overwrite public and protected methods. Private methods cannot be replaced with this approach.



There are several approaches to replacing the original class with the test double class:

- **Public injection:** Use the public interface of the object under test to replace the dependency
  - **Constructor Injection:** Pass the test double as a parameter of the constructor
  - **Setter Injection:** Pass the test double as a parameter of a setter method
  - **Parameter Injection:** Pass the test double as a parameter of the method under test
- **Private Injection:** In the object under test overwrite the dependency
  - **Test Class as Friend:** Declare the test class as a friend and overwrite the private attribute that refers to the DOC interface
  - **ABAP Test Seams:** Exchange the code where the DOC is created and create the test double instead

Instead of building the test double classes manually, use the ABAP Test Double Framework to create and configure the test double instances.

## Related Information

[Writing ABAP Unit with Test Relations \[page 550\]](#)

[Managing Dependencies with ABAP Unit \[page 509\]](#)

### 5.2.1.1.4.1.1 ABAP Object Oriented Test Double Framework

The ABAP Test Double Framework has been designed to simplify and standardize the creation and configuration of test double behavior.

Test doubles are used in unit test environments to replace depended-on components that are used by the methods being tested.

The output of a method call can be configured easily using the framework API. The values of returning, exporting and changing parameters, as well as any exceptions and events can be configured for each method call.

Additionally, the framework provides functions for verifying any interactions on the double object, such as the number of times a method was called with specific input parameters.

For creating a test double, use the `CL_ABAP_TESTDOUBLE` class and call its static methods `create`, `configure_call`, and `verify_expectations` in the test class.

#### ! Restriction

The following use cases are not supported:

- Local interfaces and local classes
- Classic exceptions
- Classes marked as `FINAL`, `STATIC`, or `FOR TESTING`
- Classes with `CREATE PRIVATE` as part of the definition
- Classes with the following constructor implementations:
  - Having mandatory parameters
  - Having logic which would raise an exception or an event or any form of termination
  - Defined in the `PRIVATE SECTION`
- Configuration of methods marked as `FINAL`, `STATIC`, or `PRIVATE`

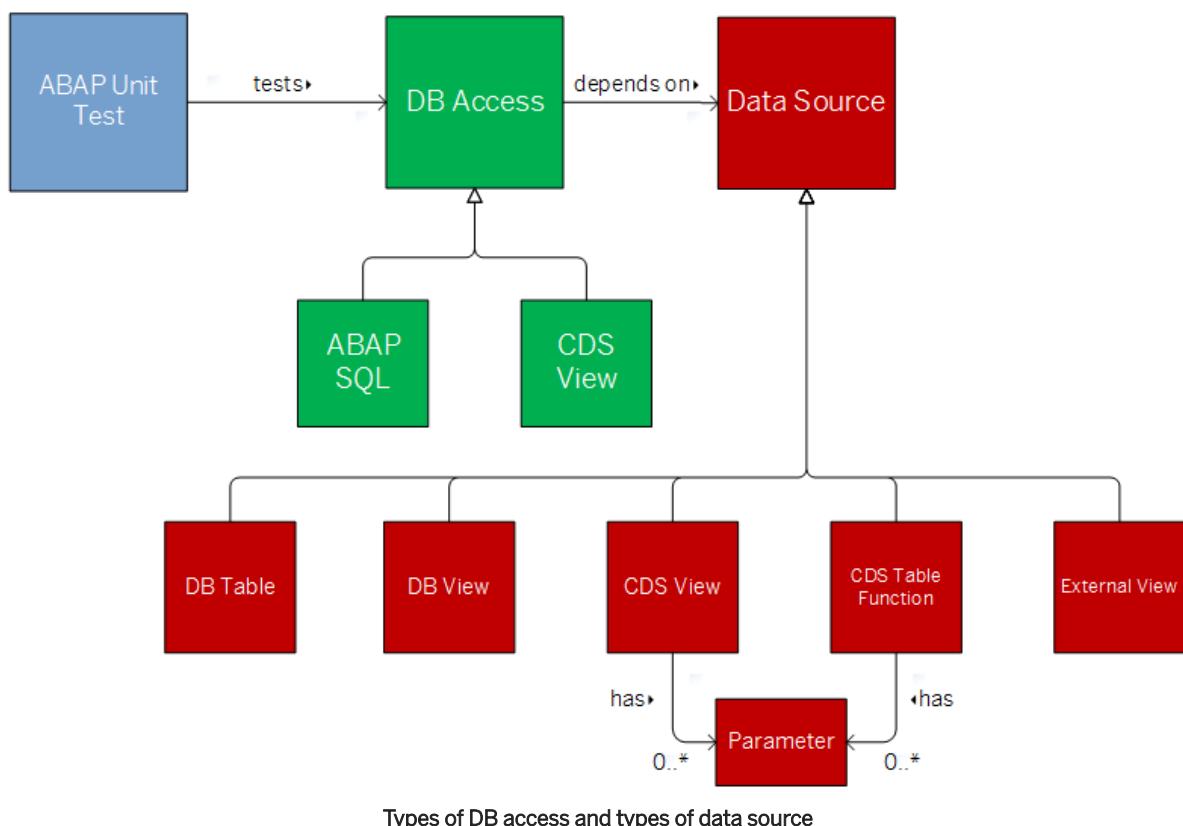
## Related Information

[Managing Object Oriented Dependencies with ABAP Unit \[page 512\]](#)

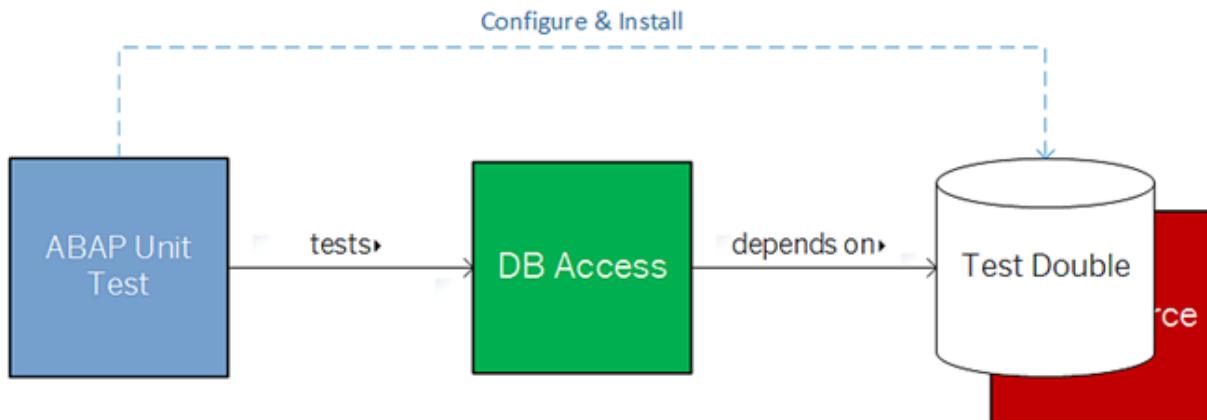
## 5.2.1.1.4.2 Managing Database Dependencies with ABAP Unit

You can manage database dependencies with ABAP CDS Test Double Framework and ABAP SQL Test Double Framework.

If you write tests that access a database, you might encounter problems with the consistency of the data. The data in databases can change, whereas the data for the tests needs to be stable. To take control of the data, you can manage dependencies between the object under test and the data source. A data source can be a database table, a database view, a CDS view, a CDS table function, or an external view.



To manage dependencies in CDS views, use ABAP CDS Test Double Framework. For database dependencies in ABAP programs, use ABAP SQL Test Double Framework. The frameworks support you in replacing data sources with test doubles, so that the object under test accesses the test double instead of the real depended-on component.



Replacing a data source with a test double

## Related Information

[ABAP CDS Test Double Framework \[page 516\]](#)

[ABAP SQL Test Double Framework \[page 539\]](#)

### 5.2.1.1.4.2.1 ABAP CDS Test Double Framework

CDS Test Double Framework enables you to test the logic expressed in CDS entities in an automated way.

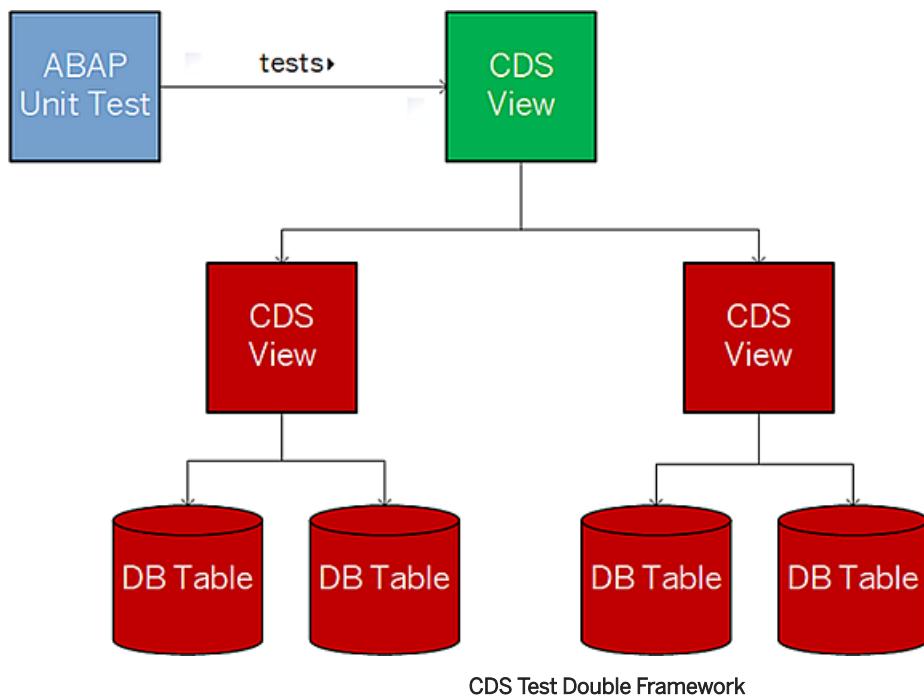
### Challenges with Conventional ABAP Solutions

The logic in a CDS entity is executed in the underlying database, which is independent of the ABAP runtime, which means it is not possible to use conventional ABAP solutions for inserting dependencies. The depended-on components (DOC) of the CDS need to be doubled in the database. It needs to be ensured that the database engine calls or executes these Test Doubles instead of the dependent component when the CDS is tested.

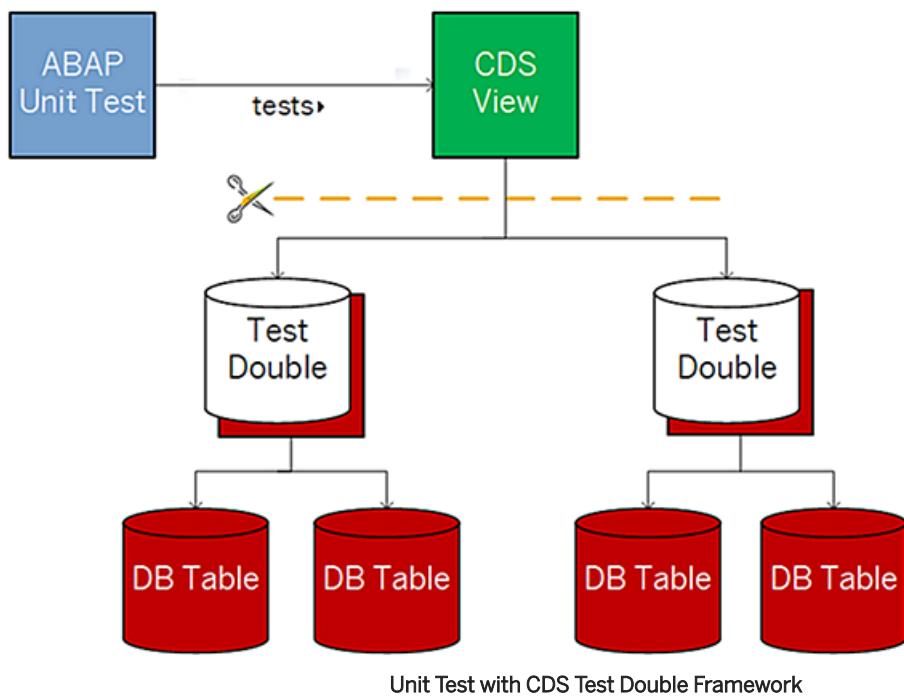
To test the logic in CDS entity under test (CUT), test data must be inserted in the double. This data is returned by the Test Doubles when the CUT is executed. CUT must not be modified by the test framework to enable unit or hierarchical testing.

## CDS Test Double Framework

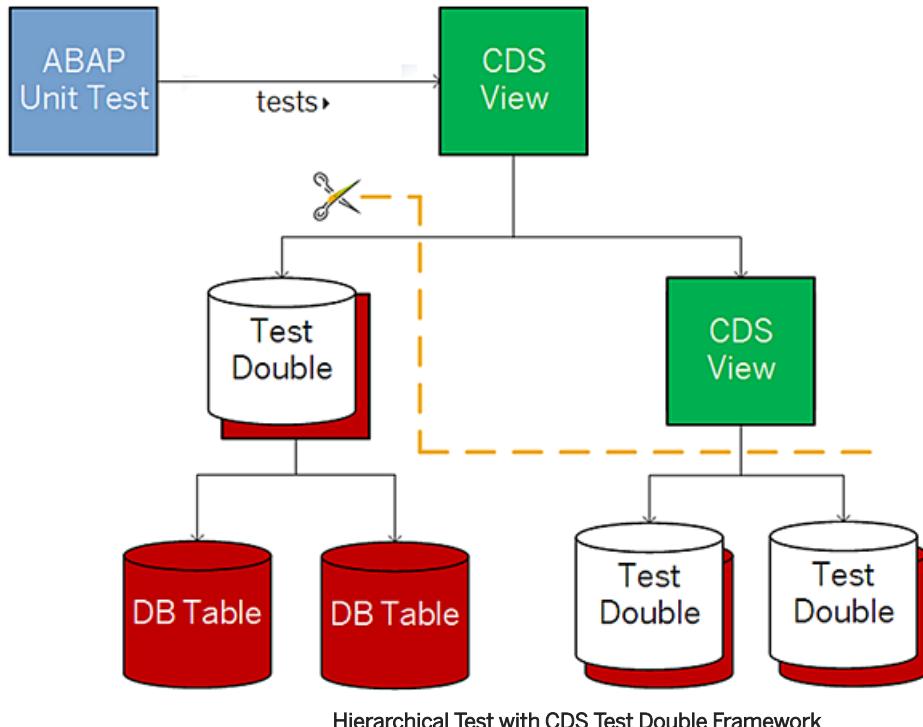
Use CDS Test Double Framework to address these challenges. It enables unit or hierarchical testing of CDS views by automatically creating "updatable" Test Doubles for each dependent component in the same database schema. The framework double that is created is a stub which has the same structure as the original dependent component.



The figure below shows an ABAP Unit test for a CDS view. The dependent CDS Views have been replaced with test double tables. By imitating the behavior of the original CDS views, test double tables provide a CDS view under test with the data created with a test class wizard.



The figure below shows the hierarchical testing of two CDS views. To test the views, a dependent CDS View and two dependent tables are replaced by the appropriate test double tables.



## Supported Test Scenarios

CDS Test Double framework supports creation of test doubles for the following dependent components:

- DDIC tables
- DDIC views
- CDS views
- CDS views with parameters
- CDS view entities
- CDS view entities with parameters
- Projection views
- External views
- Table functions
- Special CDS functions (CURRENCY\_CONVERSION and UNIT\_CONVERSION)
- Shared tables (For all share types like R, W, T, and S)
- Session variables

CDS test double framework supports the following entities as code under test:

- CDS views
- CDS views with parameters
- CDS views with DCL
- CDS view entities
- CDS view entities with parameters
- CDS view entities with DCL
- Projection views

## Related Information

[Creating a Test Class Using a Wizard \[page 519\]](#)

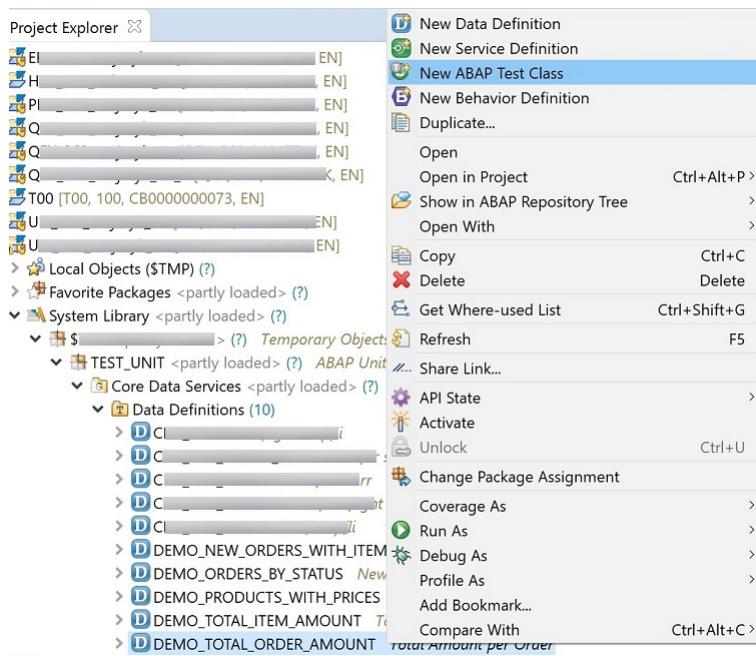
## 5.2.1.1.4.2.1.1 Creating a Test Class Using a Wizard

### Context

Create a new test class with boiler plate code to reduce the entry barrier of writing tests for CDS views.

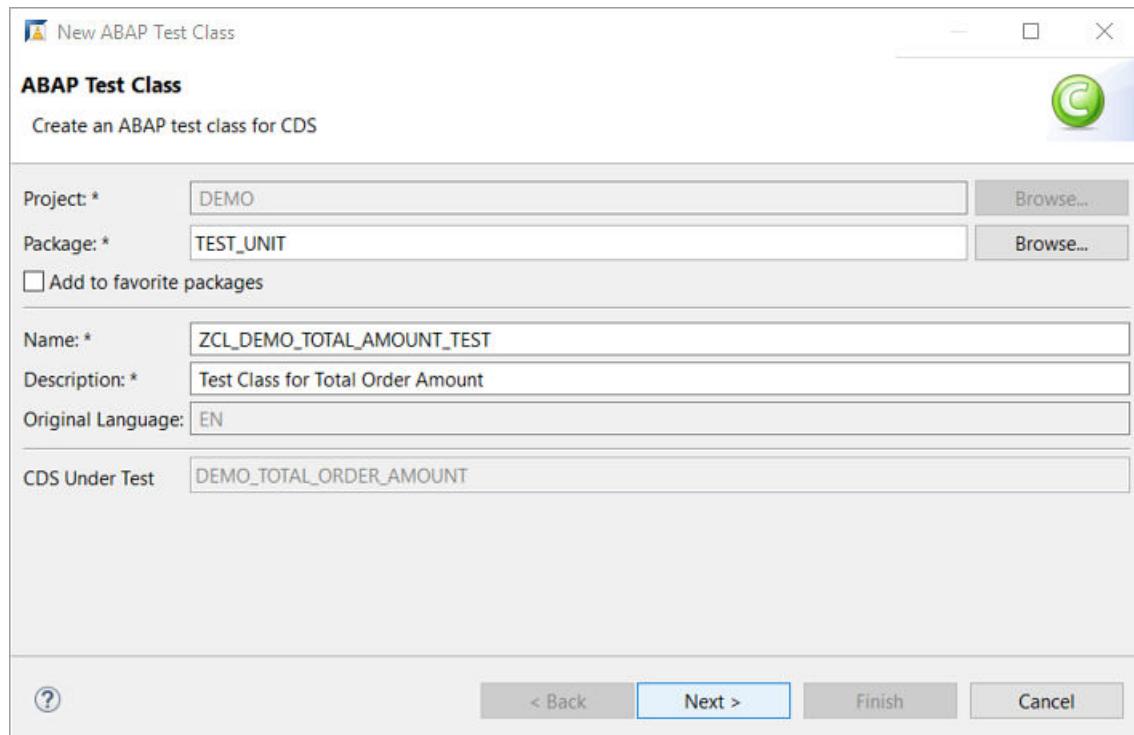
### Procedure

1. In the *Project Explorer*, select the data definition that contains the CDS entity you want to test.
2. Open the context menu and select *New Test Class* to launch the creation wizard.



Creating a new test class from the context menu

- Specify the *Package*, *Name*, and *Description* for the ABAP test class you want to create. Select *Next*



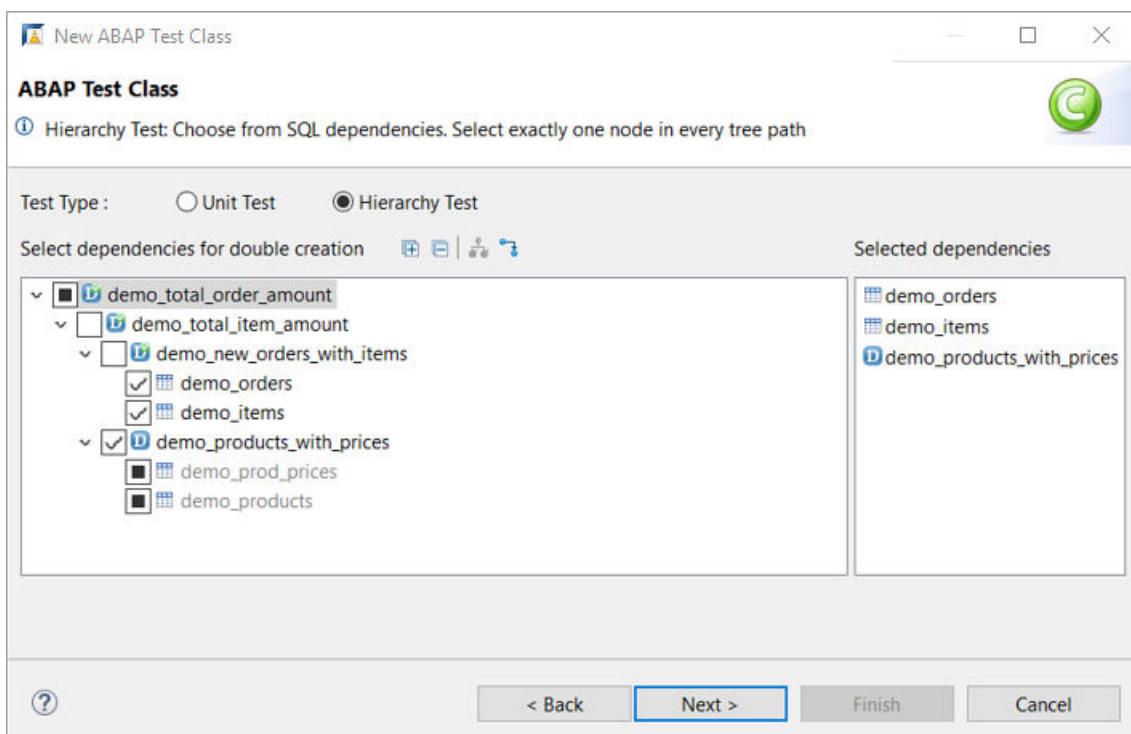
Creating a new ABAP class

- Select *Test Type*.
  - Unit Test**: Use this option to create a new test class by replacing all first level dependencies.
  - Hierarchy Test**: Use this option to replace dependencies at a deeper level. For example, you can test all involved CDS entities by replacing all dependencies on the lowest level. This means you have to select all leaf nodes. You can do this by choosing the  icon.

- Ensure at least one node is selected from each tree path for the selection to be valid.
- If you select a node in the tree, its child nodes are disabled.

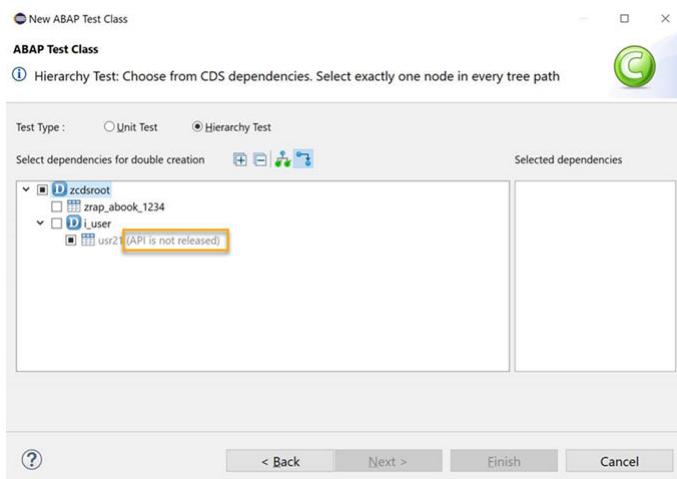
### i Note

If you want to test associations that are exposed by the CDS entity but are themselves not used in the entity, you can do this by adding these associations to the dependency list. Use the  icon to do this. Then you can also select the target data sources of these associations to create doubles and to specify the test data.



### Selecting a test type

While creating a new ABAP test class for a CDS entity, the CDS dependency tree displays the API release state of the dependent objects. You can only select the dependencies that have released APIs for creating test doubles.



#### Displaying API Release State of Dependent Objects

5. Select *Next*.
6. Enter the test method name.

CLIENT	ORDER_ID	PRODUCT_ID	QUANTITY
000	0001	FB	10
000	0002	KT	5
000	0003	TK	2
000	0004	FS	3

#### Entering test method name

The list of dependencies is displayed on the left pane. By clicking these dependencies, you can view the corresponding column names on the right pane.

7. Select one of them and enter the test data.

If the dependency contains a parameter, the parameter is visible as a column with the symbol . Enter the parameters in the respective column.

You can use other options such as *Add Row* and *Delete Row(s)* for adding a new row or deleting one or more rows. Use *Select Columns* to filter the required columns from the dependency table.

-  (Add Row) to add a new row.
  -  (Delete Rows) to delete one or more rows.
  -  (Select Columns) to filter the required columns from the dependency table.
  -  (Import Test Data) is used to upload the test data from a file that was exported from Data Preview. You can select the required columns from the uploaded file that need to be imported into the table.
8. Select *Next*.
  9. Assign a transport request and select *Finish*.
  10. Switch to the test class tab of the new class and finalize the test code.

## Related Information

[ABAP CDS Test Double Framework \[page 516\]](#)

## 5.2.1.1.4.2.1.2 Writing a Test for a CDS Entity Using Quick Assist Proposals

Quick Assist View provides you proposals for context specific source code generation.

### Prerequisites

Ensure the following prerequisites are met:

- The test class that is created must have [Test References](#) to link the test class or test methods with the repository objects.

#### Note

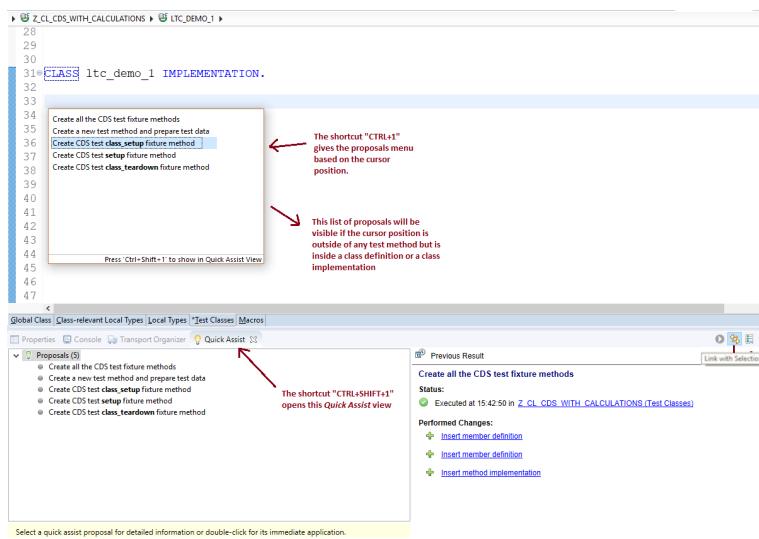
Select [Table of contents](#) and navigate to the path: *ABAP - Keyword Documentation > ABAP - Reference > ABAP Syntax > Program Directives > Test Reference*.

- Class must be activated for quick assist proposals to appear in ADT.

### Context

In ABAP Development Tools (ADT), you can open [Quick Assist](#) prompt using **CTRL** + **1** keys. If you want to dock this view to your ADT workspace perspective, use **CTRL+SHIFT+1** keys.

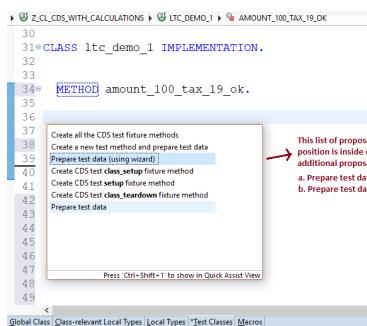
Currently, the following quick assist proposals are offered while writing a test for CDS entity.



The screenshot shows the SAP IDE interface with the following details:

- Project Path:** Z\_CL\_CDS\_WITH\_CALCULATIONS > LTC\_DEMO\_1
- Code Area:** A code editor showing a class implementation. The cursor is positioned within a class definition block.
- Quick Assist View:** A floating window titled "Quick Assist" is displayed, listing proposals for CDS test fixture methods. One proposal, "Create CDS test class\_setup fixture method", is highlighted.
- Annotations:**
  - A red arrow points to the "Create CDS test class\_setup fixture method" proposal with the text: "The shortcut 'CTRL+1' gives the proposals menu based on the cursor position."
  - A red arrow points to the floating window with the text: "This list of proposals will be visible if the cursor position is outside of any test method but is inside a class definition or a class implementation".
  - A note at the bottom of the Quick Assist window says: "Press 'Ctrl+Shift+1' to show in Quick Assist View".

Quick Assist proposals when the cursor is outside any test method but within a test class definition or implementation



The screenshot shows the SAP IDE interface with the following details:

- Project Path:** Z\_CL\_CDS\_WITH\_CALCULATIONS > LTC\_DEMO\_1 > AMOUNT\_100\_TAX\_19\_OK
- Code Area:** A code editor showing a method implementation. The cursor is positioned within a method block.
- Quick Assist View:** A floating window titled "Quick Assist" is displayed, listing proposals for CDS test fixture methods. One proposal, "Create CDS test class\_setup fixture method", is highlighted.
- Annotations:**
  - A red arrow points to the "Create CDS test class\_setup fixture method" proposal with the text: "This list of proposals will be visible if the cursor position is inside of any test method. There are 2 additional proposals that can see in this context:
  - A. Prepare test data
  - B. Prepare test data (using wizard)
- A note at the bottom of the Quick Assist window says: "Press 'Ctrl+Shift+1' to show in Quick Assist View".

Quick Assist proposals when the cursor is inside any test method

## Procedure

1. Select **Create CDS test class\_setup fixture method** proposal.

For creating the class\_setup method, you can select the CTRL+1 or CTRL+SHIFT+1 keys in the ABAP unit class to view the code generation proposals.

```
METHOD class_setup.  
  "corresponding doubles and clone(s) for the CDS view under test and its  
  dependencies, are created here  
  environment = cl_cds_test_environment->create( 'CDSFRWK_OPEN_SO_ITEMS' ).  
ENDMETHOD.
```

2. Select **Create CDS test setup fixture method** proposal.

```
METHOD setup.  
  "clear_doubles clears the test data for all the doubles used in the test  
  method before each test method execution.  
  environment->clear_doubles( ).  
ENDMETHOD.
```

3. Select **Create CDS class\_teardown fixture method** proposal.

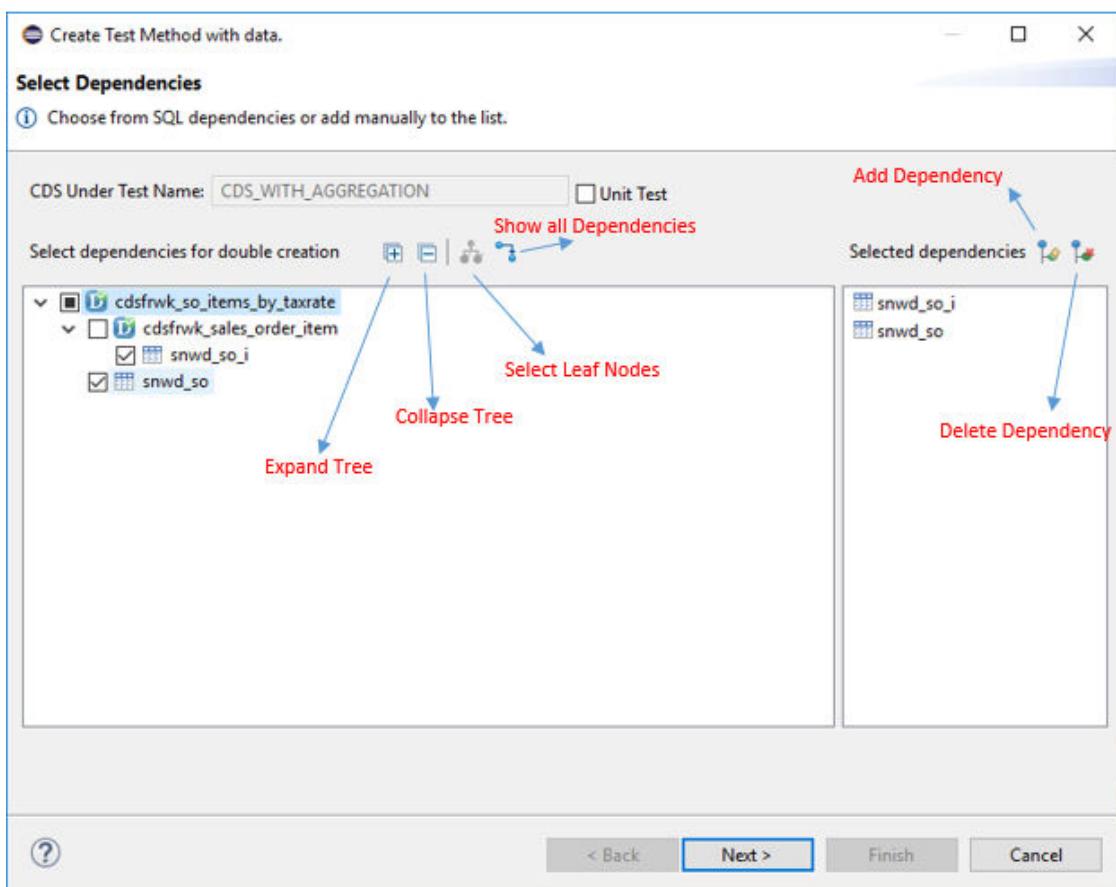
```
METHOD class_teardown.  
  "Generated database entities (doubles & clones) should be deleted at the  
  end of test class execution.  
  environment->destroy( ).  
ENDMETHOD.
```

4. Select **Create all the CDS test fixture methods** proposal.

```
METHOD class_setup.  
  "corresponding doubles and clone(s) for the CDS view under test and its  
  dependencies, are created here  
  environment = cl_cds_test_environment->create( 'CDSFRWK_OPEN_SO_ITEMS' ).  
ENDMETHOD.  
METHOD setup.  
  "clear_doubles clears the test data for all the doubles used in the test  
  method before each test method execution.  
  environment->clear_doubles( ).  
ENDMETHOD.  
METHOD class_teardown.  
  "Generated database entities (doubles & clones) should be deleted at the  
  end of test class execution.  
  environment->destroy( ).  
ENDMETHOD.
```

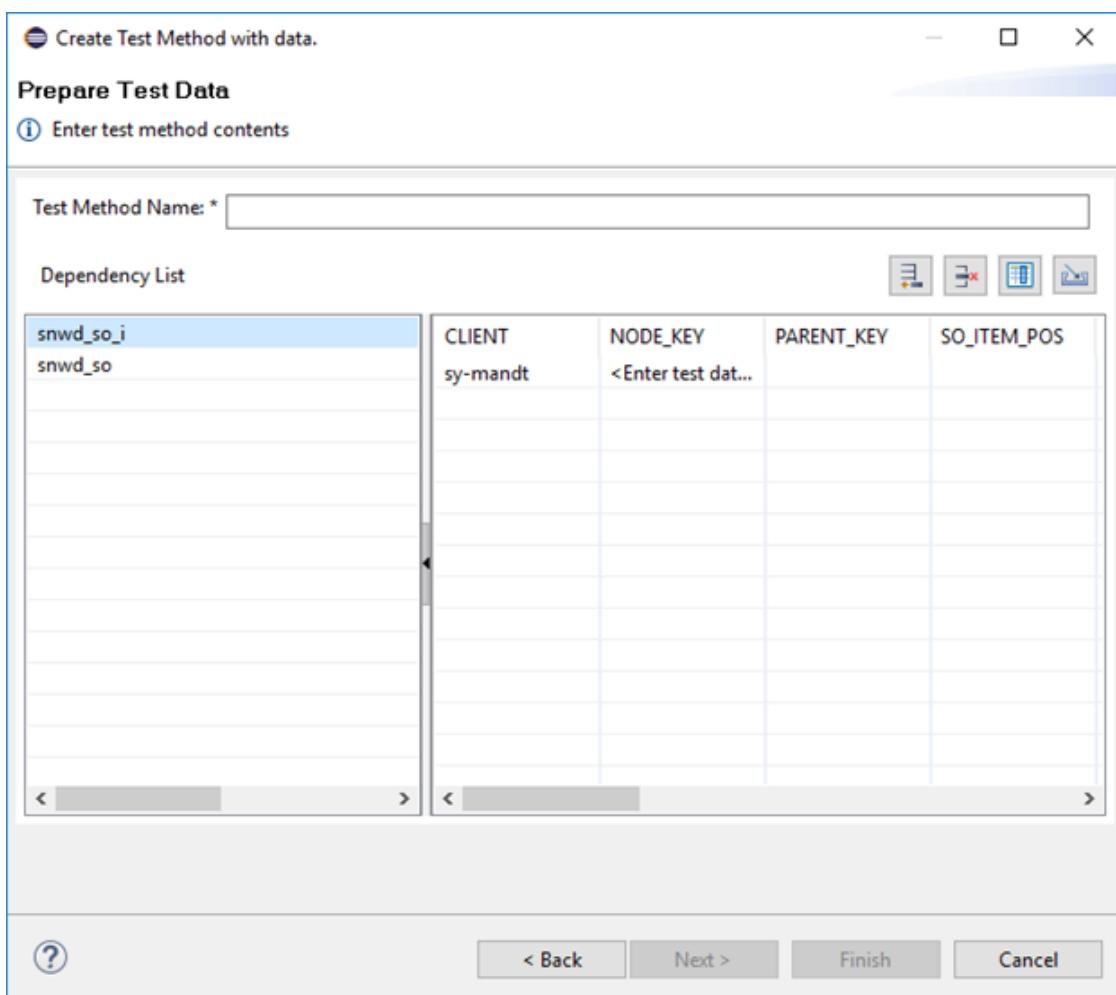
5. (Using wizard) Select **Create a new test method and prepare test data** proposal.

- *Unit Test* checkbox is selected by default to test first level dependencies. Clear the selection to choose dependencies.



### i Note

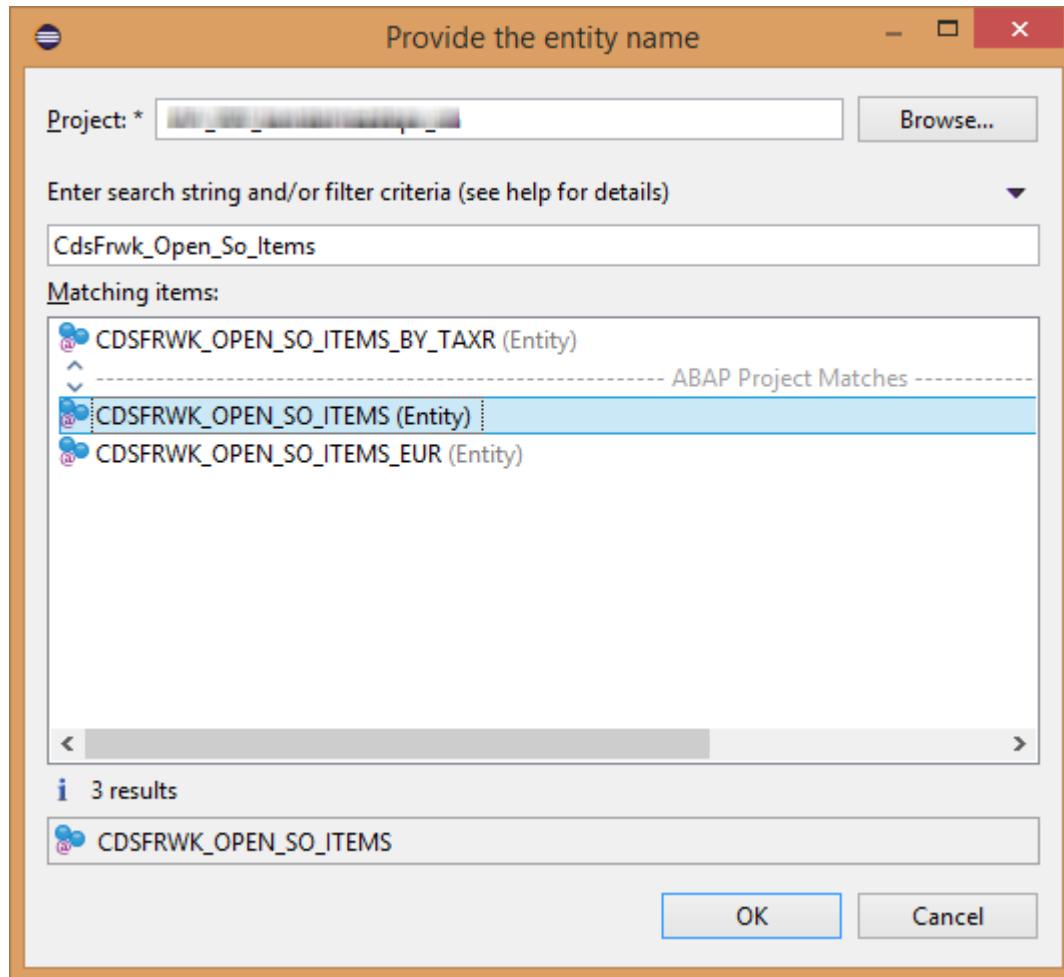
- Dependencies can be added/deleted manually by selecting and icons respectively.
  - All the leaf nodes of the tree can be selected using button.
  - In the wizard, the dependency tree shows all the SQL dependencies i.e. the dependencies (data sources) in those CDS associations which are “used” in the CDS Entity. Additionally, there is also an option to show all the first level CDS dependencies which means showing dependencies from “Exposed” and BUT “Not Used” associations as well, along with all the SQL dependencies.
  - These dependencies can also be included for test double creation
  - - This icon can be used for the above-mentioned use case.
- Choose Next.
  - Enter Test Method Name.



The list of dependencies is displayed on the left pane. On clicking these dependencies, you can view the corresponding column names on the right-hand side.

- Enter the test data for the dependencies.  
If the dependency contains a parameter, the parameter is visible as a column with the symbol Enter the parameters in the respective column.  
You can use other options
    - (Add Row) for adding a new row.
    - (Delete Rows) for deleting one or multiple rows.
    - (Select Columns) to filter the required columns from the dependency table.
    - (Import Test Data) helps to upload the test data from a file which has been exported from Data Preview. You can select the required columns from the uploaded file which needs to be imported into the table.
    - Select *Finish*.
6. Select **Prepare test data** proposal.

For example, if the cursor position is inside a test method, say `mixed_guids_and_cuco`, then this proposal is visible. If use this proposal, then the following code is generated inside the test method at the cursor position:



```

"TODO: Provide the test data here
cdsfrwk_demo_3_data = VALUE #( (
  MANDT = ' '
  SO_GUID = ''
  SO_ID = ''
  CURRENCY_CODE = ''
  SUM_GROSS_AMOUNT = ''
  TAX_RATE = ''
) ).

"TODO: Provide the input parameter data here
cdsfrwk_demo_3_params = VALUE #( (
  parameter_name = 'PCUCO'
  parameter_value = ''
) ).

  environment->insert_test_data( i_data = act_results i_parameter_values =
i_param_vals ).
```

7. Select **Prepare test data (using wizard)** proposal.

Allows you to set data for the selected dependencies using the wizard for an existing test method and generates the code accordingly. To execute this proposal, you can follow the same steps as in the [Create a new test method and prepare test data](#) proposal.

## Related Information

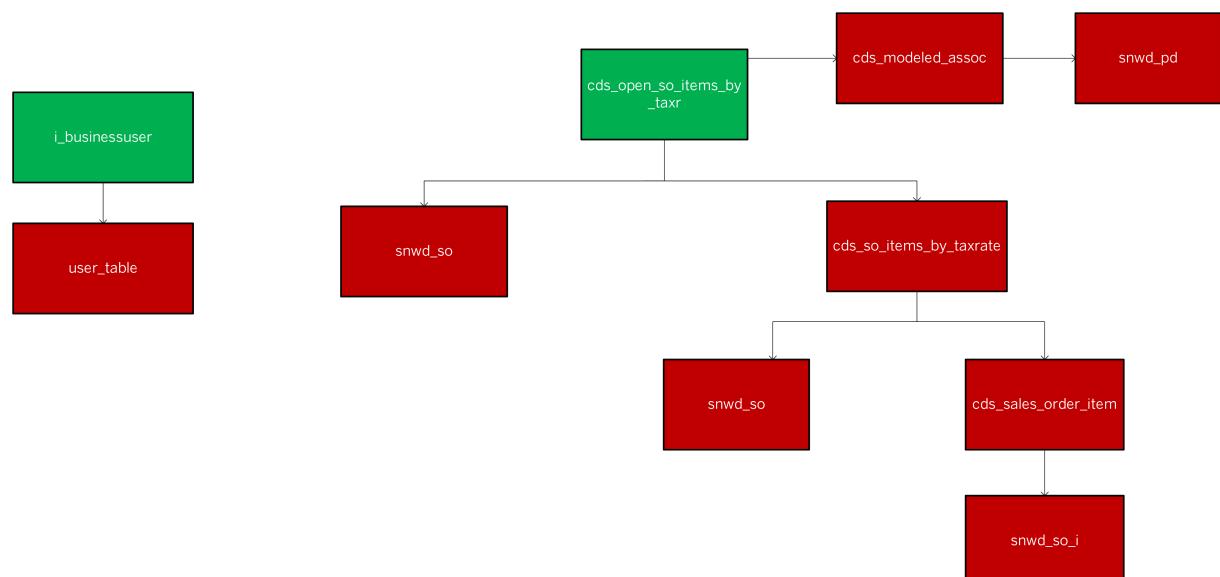
[ABAP Unit Test Classes \[page 97\]](#)

### 5.2.1.1.4.2.1.3 Writing a Test for CDS Entity

You can write unit tests to test the logic in a CDS entity using the CDS Test Double Framework.

## Context

The different testing scenarios are explained using the diagram:



## Code Under Test

### Sample Code

```
@AbapCatalog.sqlViewName: 'CDS_DEMO_1'
@EndUserText.label: 'Calculations in SELECT list'
@AbapCatalog.compiler.compareFilter: true
define view Cds_Sales_Order_Item
  as select from snwd_so_i
    association [1] to snwd_pd as _product on snwd_so_i.product_guid =
    product.node_key
  {
```

```

key snwd_so_i.node_key
parent_key
snwd_so_i.product_guid,
snwd_so_i.so_item_pos,
currency_code,
gross_amount,
case net_amount
when 0 then 0
else
  cast( ( division( tax_amount, net_amount, 4 ) * 100 ) as
abap.dec( 15, 2 ) )
end
  as tax_rate,
division( tax_amount, net_amount, 4 ) * 100 as demo,
_product
}

```

1. Create an ABAP test class.

#### « Sample Code

```

CLASS ltc_demo_1 DEFINITION FINAL FOR TESTING DURATION SHORT RISK LEVEL
HARMLESS.

PRIVATE SECTION.
CLASS-DATA: environment TYPE REF TO if_cds_test_environment.
...
ENDCLASS.

CLASS ltc_demo_1 IMPLEMENTATION.
...
ENDCLASS.

```

2. Define fixture methods.

The fixture method `class_setup` creates test doubles and the test environment. This test environment is used to insert test data in the doubles.

#### « Sample Code

```

METHOD class_setup.
  environment = cl_cds_test_environment->create( i_for_entity =
'Cds_Sales_Order_Item' ).
  "Test double would be created for the table 'snwd_so_i'. Executes once
per test class.
ENDMETHOD.

METHOD setup.
  environment->clear_doubles( ).
  "Ensures fresh data for each test method. Executes once before each
test method execution
ENDMETHOD.

METHOD calculate_tax_rate.
...
ENDMETHOD.

METHOD class_teardown.
  environment->destroy( ).
  "Destroys test environment & test doubles created as part of the test.
Executes once per test class.
ENDMETHOD.

```

3. Implement unit test method.

#### ↳ Sample Code

```
METHOD calculate_tax_rate.  
  
    "Prepare and insert test data  
    sales_order_items = VALUE #( ( client = sy-mandt net_amount = 333  
tax_amount = 111 ) ).  
    environment->insert_test_data( i_data = sales_order_items ).  
    "Execute the CDS under test  
    SELECT * FROM cds_sales_order_item INTO TABLE @act_results.  
    "Verify the result with the expectation  
    cl_abap_unit_assert=>assert_equals( act = lines( act_results ) exp =  
1 ).  
    cl_abap_unit_assert=>assert_equals( act = act_results[ 1 ]-tax_rate  
exp = '33.33' ).  
  
ENDMETHOD.
```

#### i Note

You can find detailed examples in the package `SABP_UNIT_DOUBLE_CDS_DEMO`.

### 5.2.1.1.4.2.1.3.1 Writing Unit Test for a CDS Entity

During the unit testing of a CDS entity using CDS test double framework, test doubles are created for all the first-level dependencies.

The different scenarios for unit testing a CDS entity:

#### Unit Testing a CDS View

#### ↳ Sample Code

```
METHOD class_setup.  
  
    environment = cl_cds_test_environment=>create( i_for_entity =  
'Cds_Open_So_Items_By_TaxR' ).  
    "Test doubles would be created for 'Cds_So_Items_By_TaxRate' and  
'snwd_so'.  
  
ENDMETHOD.
```

#### Unit Testing a CDS View - Modeled Associations

You can test modeled associations. Only the first-level associations of the CUT are considered.

#### ↳ Sample Code

```
METHOD class_setup.  
  environment = cl_cds_test_environment->create( i_for_entity =  
  'Cds_Open_So_Items_By_TaxR' test_associations = 'X').  
  "Test doubles would be created for 'Cds_So_Items_By_TaxRate', 'snwd_so'  
  and 'Cds_Modeled_Assoc'.  
ENDMETHOD.
```

### 5.2.1.4.2.1.3.2 Writing Hierarchical Test for a CDS Entity

In hierarchical testing of CDS entity, you must specify the list of dependencies from the hierarchy of the CDS entity under test for which you want to create the test doubles.

In the CDS hierarchy, ensure that one dependency is selected from each tree path for the selection to be valid. Once the dependencies are selected, the test doubles for these dependencies are created. All the entities above the selected dependencies in every tree path can be tested using ABAP SQL statements.

The different scenarios for hierarchical testing of CDS entity are:

#### Hierarchy Test - Specify All Dependencies

#### ↳ Sample Code

```
METHOD class_setup.  
  environment = cl_cds_test_environment->create( i_for_entity =  
  'Cds_Open_So_Items_By_TaxR'  
  i_dependency_list =  
  VALUE #( ( name =  
  'Cds_Sales_Order_Item' type = 'CDS_VIEW' )  
  ( name =  
  'SNWD_SO' type = 'TABLE' )  
  )  
  ).  
  "Test doubles would be created for 'Cds_Sales_Order_Item', 'snwd_so'  
  "and the logic inside the entities  
  'Cds_Open_So_Items_By_TaxR', 'Cds_So_Items_By_TaxRate'  
  "can be tested using ABAP SQL statements in the test methods.  
ENDMETHOD.
```

#### Hierarchy Test - Select Leaf Nodes

You can specify dependencies for few tree paths and select the leaf nodes (base dependencies) for others.

## ↳ Sample Code

```
METHOD class_setup.  
  environment = cl_cds_test_environment->create( i_for_entity =  
  'Cds_Open_So_Items_By_TaxR'  
  i_dependency_list =  
  VALUE #( ( name =  
  'Cds_Sales_Order_Item' type = 'CDS_VIEW' )  
  )  
  i_select_base_dependencies  
  = abap_true  
  ).  
  "Test doubles would be created for 'Cds_Sales_Order_Item', 'snwd_so'  
  "and the logic inside the entities  
  'Cds_Open_So_Items_By_TaxR', 'Cds_So_Items_By_TaxRate'  
  "can be tested using ABAP SQL statements in the test methods.  
ENDMETHOD.
```

## Hierarchy Test - Select All Leaf Nodes

You can select the leaf nodes (base dependencies) for all the tree paths.

## ↳ Sample Code

```
METHOD class_setup.  
  environment = cl_cds_test_environment->create( i_for_entity =  
  'Cds_Open_So_Items_By_TaxR'  
  i_select_base_dependencies  
  = abap_true  
  ).  
  "Test doubles would be created for 'snwd_so', 'snwd_so_i'  
  "and the logic inside the entities  
  'Cds_Open_So_Items_By_TaxR', 'Cds_So_Items_By_TaxRate', 'Cds_Sales_Order_Item'  
  "can be tested using ABAP SQL statements in the test methods.  
ENDMETHOD.
```

## Hierarchy Test – Modeled Associations

You can test modeled associations. Only the first-level associations of the CUT are considered.

## ↳ Sample Code

```
METHOD class_setup_4.  
  environment = cl_cds_test_environment->create( i_for_entity =  
  'Cds_Open_So_Items_By_TaxR'  
  i_select_base_dependencies  
  = abap_true  
  test_associations = 'X'  
  ).  
  "Test doubles would be created for 'snwd_so', 'snwd_so_i', 'snwd_pd'  
  "and the logic inside the entities 'Cds_Open_So_Items_By_TaxR',  
  'Cds_So_Items_By_TaxRate',
```

```

    "'Cds_Sales_Order_Item', 'Cds_Modeled_Assoc' can be tested using ABAP SQL
statements in the test methods.

ENDMETHOD.
```

### 5.2.1.1.4.2.1.3.3 Writing Test for Multiple CDS Entities

You can use the CDS test double framework to test multiple CDS entities in the same test class.

For each CDS entity, you can specify the unit test or hierarchy test.

#### ↳ Sample Code

```

METHOD class_setup.
  environment = cl_cds_test_environment->create_for_multiple_cds(
    i_for_entities = VALUE #(
      (
        i_for_entity = 'Cds_Open_So_Items_By_TaxR'
        i_select_base_dependencies = abap_true
        i_dependency_list =
          VALUE #(
            ( 'Cds_Sales_Order_Item' )
          )
      )
      (
        i_for_entity = 'I_BUSINESSUSER'
      )
    )
  .
  "Hierarchy testing is enabled for 'Cds_Open_So_Items_By_TaxR'.
  "Test double would be created for 'Cds_Sales_Order_Item' as it is
  mentioned in the i_dependency_lists parameter
  "and as i_select_base_dependencies is set to true, doubles would be
  created for all the remaining base tables which is 'snwd_so'.
  "Unit testing is enabled for 'I_BUSINESSUSER'.
  "As the parameters i_dependency_list and i_select_base_dependencies are
  not supplied
  "Test doubles would be created for all the first level dependencies i.e
  'user_table'.
ENDMETHOD.
```

### 5.2.1.1.4.2.1.3.4 Writing Test for a CDS Entity with Access Control

Access control enables you to limit the results returned by CDS entity from the database based on static values or user authorization (classical authorization objects) conditions.

While testing, the CDS entity under test must be isolated from all its dependencies. When the test environment is created using CDS test double framework, the access control acting on the CUT is disabled by default.. Hence, during test execution, the CUT is isolated from access control acting on it, unless it is enabled explicitly.

You can enable the access control acting on the CUT and provide the test data for the user role authorization objects that are used during evaluation of the access control.

## Example

The following example shows a CDS view and an access control document defined for it. It also shows the different test scenarios possible with access control while testing using CDS test double framework.

### ↳ Sample Code

#### CDS View

```
@AbapCatalog.sqlViewName: 'CDS_DEMO_14B'
@AbapCatalog.compiler.compareFilter: true
@AccessControl.authorizationCheck: #CHECK
@EndUserText.label: 'CDS based on DCL with PFBCG conditions'
define view Cds_With_Dcl_Pfcg
  as select from sflight
{
  key sflight.carrid,
  key sflight.connid,
  key sflight.fldate,
  sflight.currency,
  sflight.planetype,
  sflight.seatsmax,
  sflight.seatsocc
}
```

### ↳ Sample Code

#### Access control for the above CDS view

```
@EndUserText.label: 'DCL for CDS_WITH_DCL_PFCG (Demo)'
@MappingRole: true
define role cds_DCL_role_PFCG {
  grant
    select
  on
    CDS_WITH_DCL_PFCG
  where

    -- For flights without booked seats no restriction
    seatsocc = 0

    OR

    -- Otherwise, the CARRID must be allowed via S_CARRID with "Display"-authorization
    ( carrid ) = aspect pfbcg_auth( S_CARRID, CARRID, ACTVT = '03' )

    AND

    -- And the combination of CURRENCY and PLANETYPE is authorized via
    authorization object S_ACMDemo
    ( currency, planetype ) = aspect pfbcg_auth( S_ACMDemo, SACMTSOID,
    SACMORGUID )

  ;
}
```

Here, the access control acting on the CDS view grants access to it using static condition SEATSOCC = 0 and classical PFBCG authorization objects. The carrid field of the CDS view is mapped to the authorization field

CARRID of the authorization object S\_CARRID with *display* authorizations. The *currency* and *planetype* fields of the CDS view are mapped to SACMTSOID and SACMORGUID fields of the authorization object S\_ACM\_DEMO.

After the test environment is created, the access control on the CDS view is disabled.

#### « Sample Code

```
METHOD class_setup.  
  environment = cl_cds_test_environment->create( i_for_entity =  
  'Cds_With_Dcl_Pfcg' )."Unit testing is enabled for the CDS  
ENDMETHOD.
```

Insert the test data into the test double for each test method implementation.

#### « Sample Code

```
METHOD insert_test_data.  
  DATA:  
    lt_sflight TYPE TABLE OF sflight.  
    lt_sflight = VALUE #(  
      ( mandt = sy-mandt carrid = 'AA' connid = '1234' currency = 'USD'  
      planetype = 'B737' seatsocc = 132 )  
      ( mandt = sy-mandt carrid = 'AA' connid = '2454' currency = 'USD'  
      planetype = 'A300' seatsocc = 220 )  
      ( mandt = sy-mandt carrid = 'AA' connid = '4534' currency = 'USD'  
      planetype = 'A320' seatsocc = 0 )  
      ( mandt = sy-mandt carrid = 'LH' connid = '0815' currency = 'EUR'  
      planetype = 'A330' seatsocc = 0 )  
      ( mandt = sy-mandt carrid = 'LH' connid = '4711' currency = 'EUR'  
      planetype = 'A320' seatsocc = 145 )  
    ).  
  environment->insert_test_data( i_data = lt_sflight ).  
ENDMETHOD.
```

## Disable Access Control During Testing

When access control is disabled, all the 5 records that are inserted in the test double are returned by the CDS.

#### « Sample Code

```
METHOD disable_access_control.  
  insert_test_data( ).  
  environment->get_access_control_double( )->disable_access_control( ).  
  SELECT * FROM cds_with_dcl_pfcg INTO TABLE @DATA(lt_result).  
  cl_abap_unit_assert=>assert_equals( act = lines( lt_result ) exp = 5 ).  
ENDMETHOD.
```

## Enable Access Control without User Authorization

Here access control is enabled for the CDS view but the user role authorization objects are not provided with the test data. So, the user doesn't have required authorizations.

The two records with SEATSOCC= 0 are returned because access control provides access to them even without authorizations.

#### ↳ Sample Code

```
METHOD enable_access_control_without_auth.

  insert_test_data( ).
  DATA(acm_data_no_authorization) =
  cl_cds_test_data->create_access_control_data( i_role_authorizations = VALUE
#( ) ).
  environment->get_access_control_double( )-
>enable_access_control( i_access_control_data = acm_data_no_authorization ).
  SELECT * FROM cds_with_dcl_pfcg INTO TABLE @DATA(lt_result).
  cl_abap_unit_assert=>assert_equals( act = lines( lt_result ) exp = 2 ).

ENDMETHOD.
```

## Enable Access Control with User Authorizations for Authorization Object, S\_CARRID

Access control is enabled for the CDS view with test data only provided for one authorization object, S\_CARRID. Here, the user has authorizations to access the records for which carrid field has value LH with display authorization.

Only the two records with SEATSOCC = 0 are returned even though the authorizations for S\_CARRID authorization object are provided. This is because there are no authorizations for S\_ACMDemo object. Hence, the condition in the access control is evaluated to FALSE.

#### ↳ Sample Code

```
METHOD enable_access_control_with_1_auth_object.

  insert_test_data( ).
  DATA(acm_data_1_auth_object_authorization) =
  cl_cds_test_data->create_access_control_data(
    i_role_authorizations = VALUE #(

      (
        object      = 'S_CARRID'
        authorizations = VALUE #(
          (
            VALUE #(
              (
                fieldname    = 'CARRID'
                fieldvalues = VALUE #(
                  ( lower_value = 'LH' )  " User is authorized for CARRID
= 'LH' ...
                  ) " Field values
                ) " First field CARRID
              (
                fieldname    = 'ACTVT'
                fieldvalues = VALUE #(
                  ( lower_value = '03' )  " ... with display authorization
                  ) " Field values
                ) " Second field ACTVT
              )
            )
          )
        )
      )
    )
  )
ENDMETHOD.
```

```

        )
    )
).

environment->get_access_control_double( )-
>enable_access_control( i_access_control_data =
acm_data_1_auth_object_authorization ).
    SELECT * FROM cds_with_dcl_pfcg INTO TABLE @DATA(lt_result).
    cl_abap_unit_assert=>assert_equals( act = lines( lt_result ) exp = 2 ).

ENDMETHOD.

```

## Enable Access Control with User Authorizations for Authorization Objects,

S\_CARRID, S\_ACMDemo

The access control is enabled in CDS with test data provided for two the authorization objects, S\_CARRID and S\_ACMDemo. The user has authorizations to access the records for which `carrid` field has value `LH` and display authorization. There are no restrictions on the fields `currency` and `planetype`. In addition to the records with `SEATSOCC = 0`, one record with `carrid = LH` and `connid = 4711` is also returned.

### Sample Code

```

METHOD enable_access_control_with_both_auth_objects.

    insert_test_data( ).

    DATA (acm_data_2_auth_object_authorization) =
cl_cds_test_data=>create_access_control_data(
        i_role_authorizations = VALUE #(
            (
                object          = 'S_CARRID'
                authorizations = VALUE #(
                    (
                        VALUE #(
                            (
                                fieldname    = 'CARRID'
                                fieldvalues = VALUE #(
                                    ( lower_value = 'LH' )  " User is authorized for CARRID
= 'LH' ...
                                )
                            )
                        )
                    )
                )
            )
        )
        (
            object          = 'S_ACMDemo'
            authorizations = VALUE #(
                (
                    VALUE #(
                        (
                            fieldname    = 'SACMTSOID'
                            fieldvalues = VALUE #(
                                ( lower_value = '*' )  " All fields values permitted
for currency

```

```

        )
    )
(
    fieldname  = 'SACMORGUID'
    fieldvalues = VALUE #((
        ( lower_value = '*' )    " All fields values permitted
for planetype
        )
    )
)
)
)
)
)
)
)
)
).
environment->get_access_control_double( )-
>enable_access_control( i_access_control_data =
acm_data_2_auth_object_authorization ).
    SELECT * FROM cds_with_dcl_pfcg INTO TABLE @lt_result.
    cl_abap_unit_assert=>assert_equals( act = lines( lt_result ) exp = 3 ).

ENDMETHOD.

```

## 5.2.1.1.4.2.2 ABAP SQL Test Double Framework

You can test the code which accesses the database, using ABAP SQL Test Double Framework.

This section tells you how to write unit tests for the logic written using ABAP SQL statements.

An ABAP SQL statement depends on one or more data sources. A data source can be a table, a view, a CDS view, a CDS table function, a CDS view entities, a CDS projection view or an external view.

### Code Under Test

```

CLASS cl_order IMPLEMENTATION.

METHOD get_amount.

    SELECT * FROM demo_new_orders_with_items AS orders
    INNER JOIN demo_prod_prices AS prices ON orders~product_id =
prices~product_id
    INTO TABLE @DATA(entries).

    r_result = entries[ 1 ]-prices-amount * entries[ 1 ]-orders-quantity.

ENDMETHOD.

ENDCLASS.

```

1. Create an ABAP test class.

```

CLASS ltc_item_amount DEFINITION FINAL FOR TESTING DURATION SHORT RISK LEVEL
HARMLESS.

PRIVATE SECTION.
CLASS-DATA: environment TYPE REF TO
if_osql_test_environment.
...
ENDCLASS.

CLASS ltc_item_amount IMPLEMENTATION.

```

```
...
ENDCLASS.
```

## 2. Define fixture methods.

Define the following API calls provided by ABAP SQL Test Double Framework, within the respective fixture methods of the test class.

Executing the method `cl_osql_test_environment->create( i_dependency_list = '<dependency_list>' )` creates doubles for all the dependent components specified by you. This method should be called only once per test class.

```
METHOD class_setup.
  environment = cl_osql_test_environment->create(
    i_dependency_list = VALUE #( ( 'demo_new_orders_with_items' )
      ( 'demo_prod_prices' ) ) .
ENDMETHOD.

"Fixture method setup is executed once before each test method execution to
ensure fresh test data.
METHOD setup.
  environment->clear_doubles( ) .
ENDMETHOD.

METHOD calculate_amount.
...
ENDMETHOD.

METHOD class_teardown.
  environment->destroy( ) .
ENDMETHOD.
```

## 3. Implement unit test method.

```
METHOD calculate_amount.

* 1. Fill test data
DATA orders TYPE STANDARD TABLE OF demo_new_orders_with_items.
DATA prices TYPE STANDARD TABLE OF demo_prod_prices.
DATA: amount TYPE dsag_amount.

orders = VALUE #( ( product_id = 'TK' quantity = 5 status = 'new' ) ) .
prices = VALUE #( ( product_id = 'TK' amount = '11.00' ) ) .

environment->insert_test_data( orders ) .
environment->insert_test_data( prices ) .

* 2. Call method under test
amount = cl_znl_item_amount->get_amount( ) .

* 3. Verify result
cl_abap_unit_assert->assert_equals( act = amount exp = '55.00' ) .

ENDMETHOD.
```

You must provide proper test data for the respective key fields of the entity in which test data is being inserted. Otherwise an exception can be raised by the framework.

For all the actual database dependencies in the ABAP SQL statement for which test doubles are created, the ABAP SQL TDF ensures that they are replaced by the respective test doubles at runtime. Now the ABAP SQL statements fetch the data from the doubles rather than fetching it from the actual database artifacts.

## Other types of the data source:

- **Dependent component is a CDS view with parameters**

If the dependent component is a CDS view with parameters, set the test data for CDS parameter fields while also providing the test data for normal fields.

```
DATA parameters TYPE if_osql_param_values_config=>ty_parameter_value_pairs.  
parameters = VALUE #( ( parameter_name = 'p_status' parameter_value =  
'new' ) ).  
...  
environment->insert_test_data( i_data = orders i_parameter_values =  
parameters ).
```

- **Dependent component is a table function**

Doubles of the type Table Functions are handled in the same manner as any CDS view.

- **Dependent component is a projection view**

If the dependent component is a projection view, you can pass the projection view name as part of the i\_dependency\_list parameter.

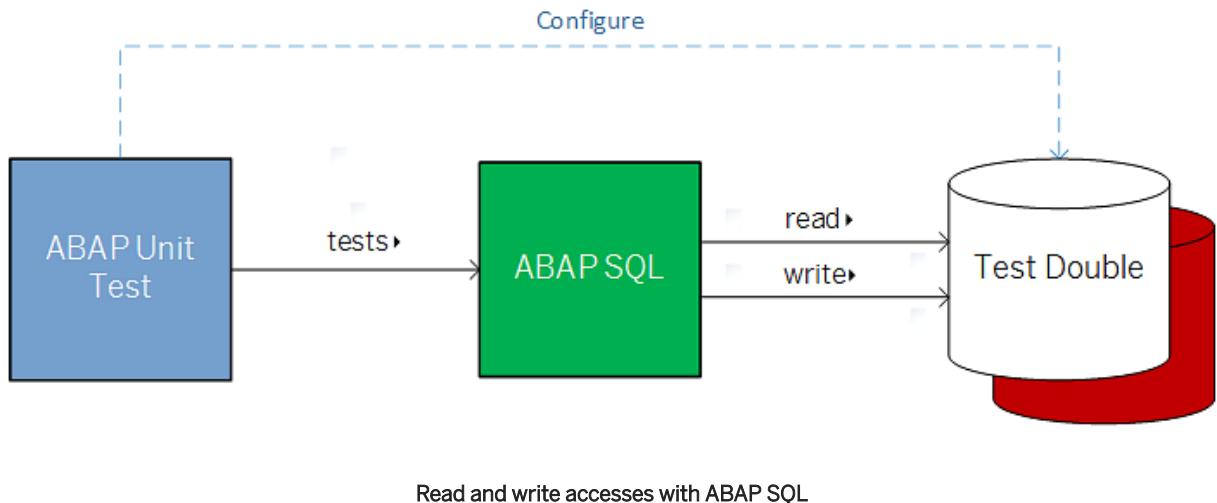
## NULL values

NULL values can be inserted explicitly in the double fields as follows:

```
DATA currency_is_null TYPE if_osql_null_values_config=>ty_element_names.  
currency_is_null = VALUE #( (`currency` ) ).  
...  
environment->insert_test_data( i_data = prices i_null_values =  
currency_is_null ).
```

## Write access

If the dependent object is a database table, both read and write accesses are redirected to the test double table.



## DCL access control

Access control logic acting on the dependent CDS is always disabled when tests are running.

## Example

You can find more examples in the package `SABP_UNIT_DOUBLE_OSQL_DEMO`.

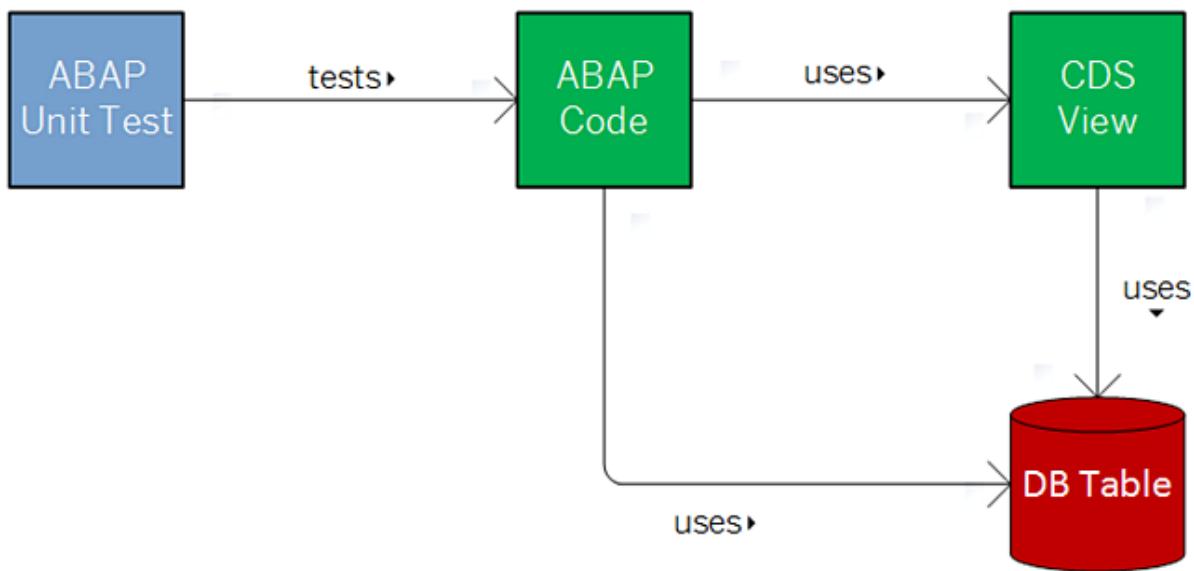
## Related Information

[Managing Database Dependencies with ABAP Unit \[page 515\]](#)

## 5.2.1.1.4.2.3 ABAP SQL TDF and CDS TDF

## Challenges

In some situations, you might want to test a component while knowing that it uses a certain database table multiple times in different parts of your software. This sounds simple because you can just tell the Test Double Framework (TDF) to replace this table and fill in some data. Here is an example:

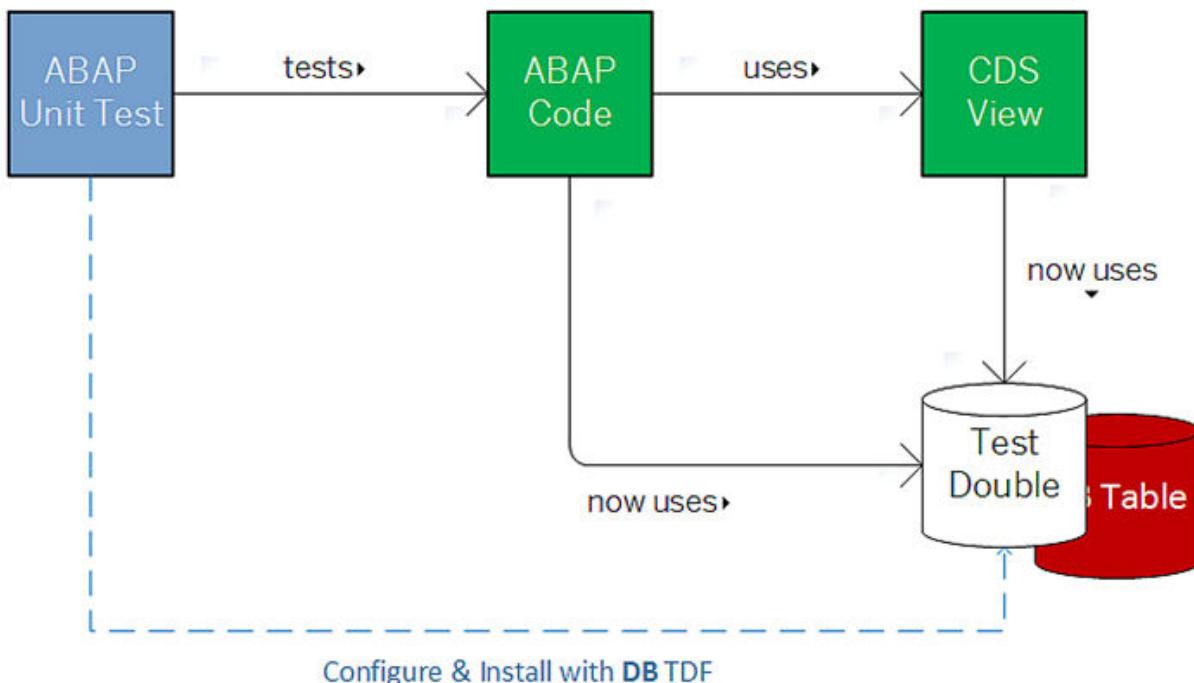


#### Source Code

```

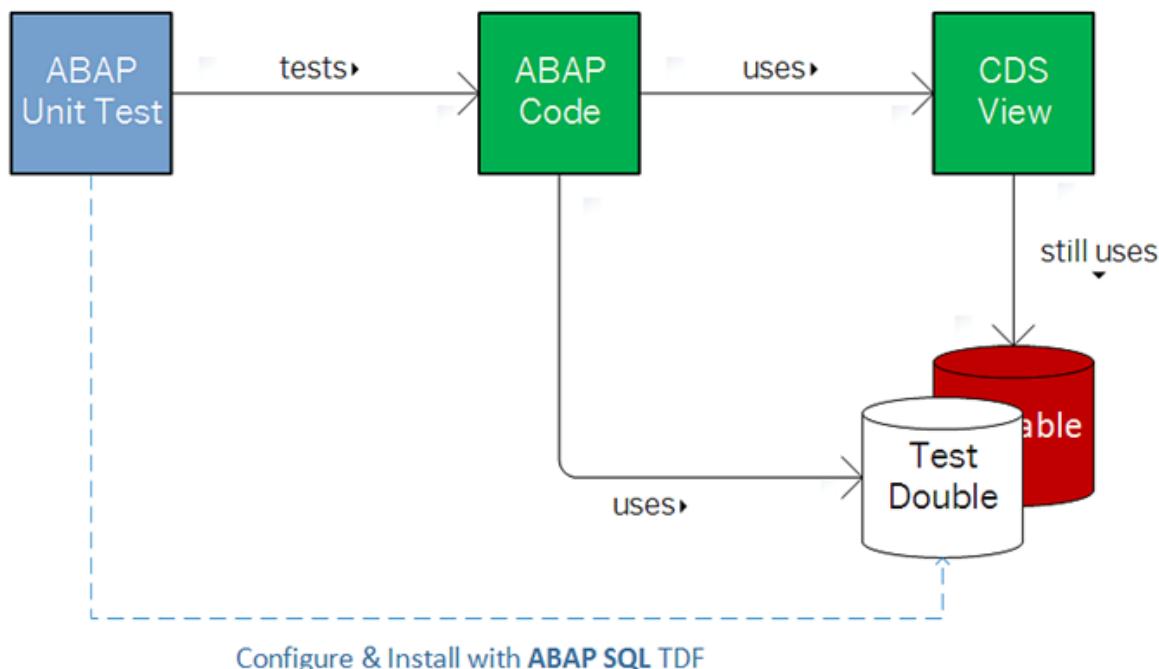
environment = cl_osql_test_environment->create( VALUE #(( 'DB_TABLE' ))).
environment->insert_test_data( <your_test_data> ).
```

This should redirect any access to the database table to the new and configured test double table as shown in the diagram below.



But unfortunately this does not always work as expected. A problem occurs if the table is not always accessed using ABAP SQL but by using CDS views (and vice versa).

For example, when you use ABAP SQL TDF, select statements of ABAP code are redirected to a test double table, whereas the CDS view still accesses the real object, as shown in the figure below.



## Solution

It is recommended that you enable double redirections with CDS TDF as shown below. This redirects any database accesses to a new test double table.

### Source Code

```
environment = cl_cds_test_environment->create(
    i_for_entity = 'CDS_VIEW'
    i_dependency_list = VALUE #( ( type = 'TABLE' name = 'DB_TABLE' ) ) .
environment->enable_double_redirection( ).
environment->insert_test_data( <your_test_data> ).
```

## Related Information

[ABAP CDS Test Double Framework \[page 516\]](#)

[ABAP SQL Test Double Framework \[page 539\]](#)

### 5.2.1.1.4.3 Managing Dependencies on ABAP Authority Checks with ABAP Unit

The ABAP Authority Check Test Helper API is an API-based approach to configure single or multiple users with authorizations for the test environment.

#### Challenges with Authority Checks

There was limited support for testing role-based functionality using AUTHORITY-CHECK statement in ABAP. Writing unit tests by configuring various users with the required roles and authorizations is complex. Configuring user roles and profiles for different user types requires considerable effort and technical understanding. Setting up test data for user roles and profiles is difficult in both single and multiuser environments. During central test runs all tests are executed by a test user, which has SAP\_ALL authorizations. Hence, the functional flow for business users is not evaluated. Running the tests in a separate session via RFC call for a user with limited rights is complicated to set up and maintain.

The authorizations for the users in the test environment are provided to the test helper APIs through an auth object. The auth objects (instances containing authorizations) support the test developers in handling test data.

#### Features

- Provides improved security by encapsulating the handler classes.
- Supports the FOR USER variant of the AUTHORITY-CHECK statement.
- Supports setting of expectations such as positive, negative or a combination of positive and negative. These expectations can be set against the tests that are evaluated.
- Supports asserting or checking of the expectations that are set previously, thus passing and failing tests wherever necessary.
- Provides comprehensive log results of AUTHORITY-CHECK statements that get executed in the test session.
  - Pass or failed executions
  - Deviations from expectations such as, expected to pass but failed and vice versa

The main API class provides the following options:

#### Creating an Authorization Object Set

Code sample

```
" Define a role with DISPLAY authorizations for authorization object S_DEVELOP.  
DATA(role_may_display) = VALUE  
cl_aunit_auth_check_types_def->role_auth_objects(  
                                         ( object = 'S_DEVELOP'
```

```

VALUE #(                                     authorizations =
#( ( fieldname    = 'ACTVT'                  ( VALUE
                                         fieldvalues   =
VALUE #( ( lower_value = '03' ) ) ) ) ) ) .
                                         ) .

DATA(usrarl_may_display) = VALUE
cl_aunit_auth_check_types_def=>user_role_authorizations( ( role_authorizations =
role_may_display ) ).

" Create an auth object set containing display authorizations.
DATA(auth_objset_with_disp_auth) =
cl_aunit_authority_check=>create_auth_object_set( usrarl_may_display ).
```

## Restricting Authorizations for Test User

### Example

There is an authorization object set `auth_objset` with authorizations that grant permissions for display of classes and query flight data of `LH` for user 1 and user 2. User 1 and user 2 must be modeled via the API for the test session.

```

" Set up environment - Get an instance of the test controller and set the user
configurations.
DATA(auth_controller) = cl_aunit_authority_check=>get_controller( ).

" Set up environment - Configure users with the intended authorizations via the
auth_objset for the test session.
auth_controller->restrict_authorizations_to( auth_objset ).
```

## Setting Positive and Negative Test Expectations

### Example

The current user must have only display authorizations during the test session via the controller `auth_controller`. Additionally, there are two authorization object sets `auth_objset_with_display_auth` which checks for display authorizations for the user in the test session and the other `auth_objset_with_create_auth` which checks for create authorizations for the user in the test session.

```

" Configure test expectations - Positive and negative expectations.
auth_controller->authorizations_expected_to(
  EXPORTING
    pass_execution = auth_objset_with_display_auth
    fail_execution = auth_objset_with_create_auth
).
```

## Evaluating Test Executions against Previously Set Expectations

### Example

The current user is restricted to only have display authorizations during the test session via the controller `auth_controller`. Additionally, there are two authorization object sets. The `auth_objset_with_display_auth` checks for display authorizations for the user in the test session. The other `auth_objset_with_create_auth` checks for create authorizations for the user in the test session, which was set as positive and negative expectations. The code under test is executed before calling assert/check APIs.

```
DATA : failed_expectations TYPE cl_aunit_auth_check_types_def=>auth_ctxtset_msgs.  
" Assert/Check configured test expectations is met.  
" Assert expectations.  
    auth_controller->assert_expectations( ).  
  
" Check expectations.  
    DATA(check_passed) = auth_controller->check_expectations(  
                           IMPORTING  
                           failed_expectations =  
                           failed_expectations  
                           ).
```

## Log Results

### Example

The current user is restricted to only have display authorizations during the test session via the controller `auth_controller`. Additionally, there are two authorization object sets. The `auth_objset_with_display_auth` probes for display authorizations for the user in the test session. The other `auth_objset_with_create_auth` checks for create authorizations for the user in the test session, which has set as positive and negative expectations. Also, the code under test is executed before calling assert/check APIs.

```
" Analyze Execution logs  
" Get the execution log and get to understand the various aspects of test,  
" This step is not mandatory and can be used to get detailed logs of :  
"   - executed authority checks,  
"   - expectation that were met and not met,  
"   - authorizations executed that are not a part of expectations.  
    auth_controller->get_auth_check_execution_log( )->get_execution_status(  
                           IMPORTING  
                           passed_execution = DATA(passed_execution)  
                           failed_execution = DATA(failed_execution)  
                           ).  
  
" Get a log of executions that were :  
"   - expected_to_pass_but_failed,  
"   - expected_to_fail_but_passed,  
"   - expected_to_pass_not_executed,  
"   - expected_to_fail_not_executed.
```

```

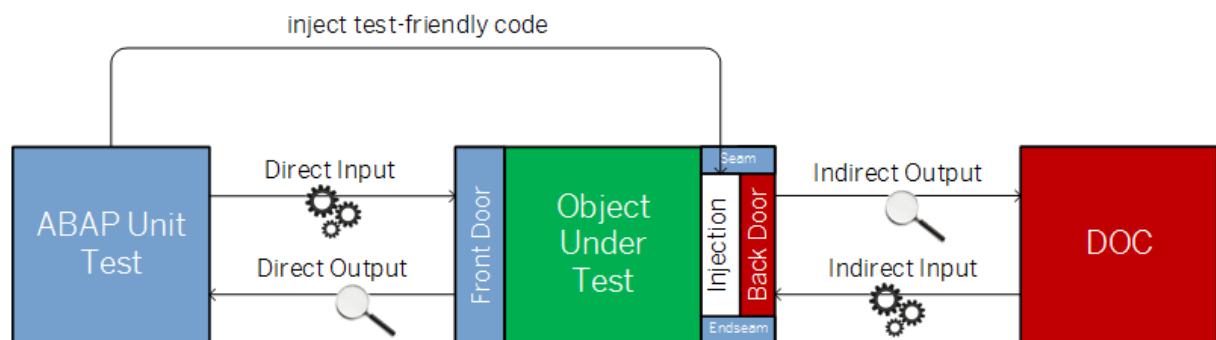
auth_controller->get_auth_check_execution_log( )->get_failed_expectations(
    IMPORTING
        expected_to_pass_but_failed    = DATA(expected_to_pass_but_failed)
        expected_to_fail_but_passed   = DATA(expected_to_fail_but_passed)
        expected_to_pass_not_executed =
    DATA(expected_to_pass_not_executed)
        expected_to_fail_not_executed = DATA(expected_to_fail_not_executed)
    ).

" Get a log of executions that passed_but_not_expected, failed_but_not_expected.
auth_controller->get_auth_check_execution_log( )->get_unexpected_executions(
    IMPORTING
        passed_but_not_expected = DATA(passed_but_not_expected)
        failed_but_not_expected = DATA(failed_but_not_expected)
    ).

```

#### 5.2.1.1.4.4 Managing Other Dependencies with ABAP Unit

You can manage dependencies using `test_seams`, a technique that facilitates the stability and independence of tests from any changes made in dependent components.



##### Writing Unit Tests with Test Seams

To use test seams, you enclose the test-unfriendly code with `TEST-SEAM <seamname> - END-TEST-SEAM`. In your test, you can specify the code that is used instead within `TEST-INJECTION - END-TEST-INJECTION`.

At runtime, the code is replaced accordingly. The replacement is valid until the end of the test method in which the injection is defined.

If you need the same injection for all test methods of your test class, you can put the `INJECTION` statement into the setup method of your test class.

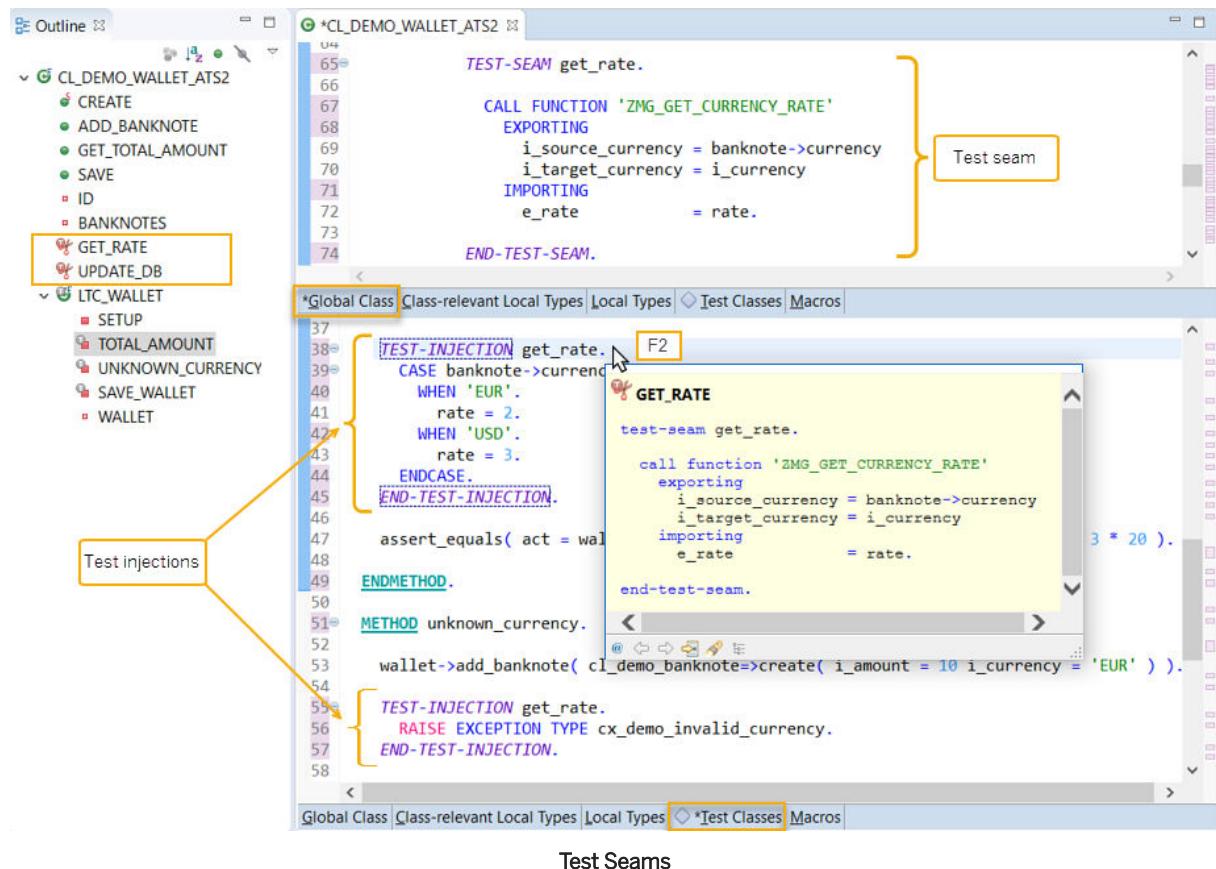
##### Scope

Within the `INJECTION` statement, you have access to all variables that are in the scope of the `TEST-SEAM` statement.

If you need access to variables or methods from your test class inside of the injection code, you have to provide these as static members of your test class.

Test seams are displayed in the *Outline* view by choosing the  icon, followed by the test seam name.

Clicking a test seam name in a test injection and pressing F2 opens a pop-up window with the corresponding test seam.



The screenshot shows the SAP ABAP Development Workbench interface. On the left, the *Outline* view is open, showing a tree structure of class members. A node labeled **Test injections** is highlighted with an orange box. A callout line points from this node to a specific line of code in the main editor window. The code editor window displays class **\*CL\_DEMO\_WALLET\_ATS2**. The line of code being pointed to is **TEST-SEAM get\_rate.**. The code injection for **get\_rate** is as follows:

```

TEST-SEAM get_rate.

CALL FUNCTION 'ZMG_GET_CURRENCY_RATE'
  EXPORTING
    i_source_currency = banknote->currency
    i_target_currency = i_currency
  IMPORTING
    e_rate            = rate.

END-TEST-SEAM.

```

The code injection is enclosed in a yellow box labeled **Test seam**. The F2 key icon is shown above the code editor. The code editor also shows other parts of the class, including a **get\_rate** method and a **unknown\_currency** method.

### ! Restriction

- Test injections can only refer to `test_seams` located in the same function pool.
- Code completion in test injections is available to a limited extent.

## Related Information

[Managing Dependencies with ABAP Unit \[page 509\]](#)

[Writing ABAP Unit Tests \[page 501\]](#)

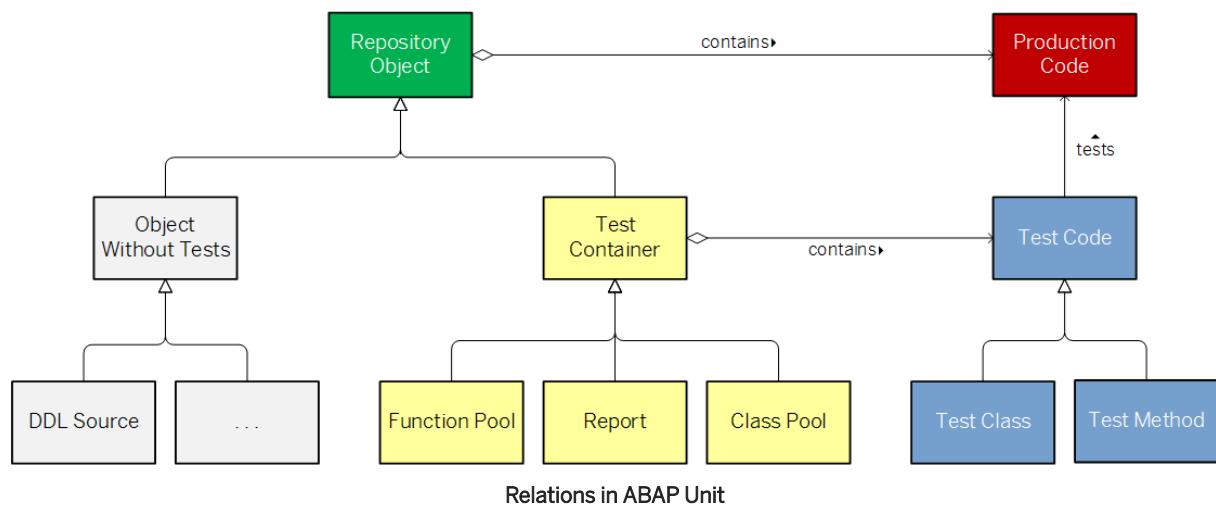
[ABAP CDS Test Double Framework \[page 516\]](#)

[ABAP SQL Test Double Framework \[page 539\]](#)

## 5.2.1.1.5 Writing ABAP Unit with Test Relations

You can use ABAP Unit to easily write automated tests for your **repository objects**, particularly if the **production code** you want to test is contained in a **class pool**, **function pool** or **report**. These object types are the only ones that can contain **test code** and therefore we call them **test containers**.

Other object types, like DDL sources or simple transformations, cannot contain any test code. If you want to test a CDS view contained in a DDL source you have to use another repository object (a test container) and write the tests for the CDS view there.



This works fine, but there is one issue: the test code is not related to the production code. Therefore, it is difficult to find the relevant tests for a certain object and execute them. To solve this issue, a relation between a test and a piece of production code has to be modeled. To model a connection, write a specific `@testing` link as [ABAP doc comment](#) in front of the test method or test class.

### i Note

The concept of test relations has been available since back-end version AS ABAP 7.53.

## Related Information

[Launching ABAP Unit with Test Relations \[page 561\]](#)

## 5.2.1.2 Launching ABAP Unit Tests

### Context

The maxim in unit testing is 'test early and often'. Here are the many convenient ways to launch your ABAP Unit tests in the ABAP Development Tools (ADT).

First, define a **test suite** by selecting one or more tests you want to execute. You can do this directly by selecting test classes or test methods or indirectly by selecting development objects.

- Direct selection: select one or more test classes or test methods in *Project Explorer*, in *Outline* view, or in *ABAP Unit View*.

#### i Note

If you want to select a test method in ABAP Editor, make sure you put your cursor on the method name in the method statement of the implementation part.

- Indirect selection: select one or more development objects in *Project Explorer*, *Outline* view, *Search* view, or *ABAP Unit* view.

The table below provides an overview of direct and indirect selection for running ABAP Unit tests.

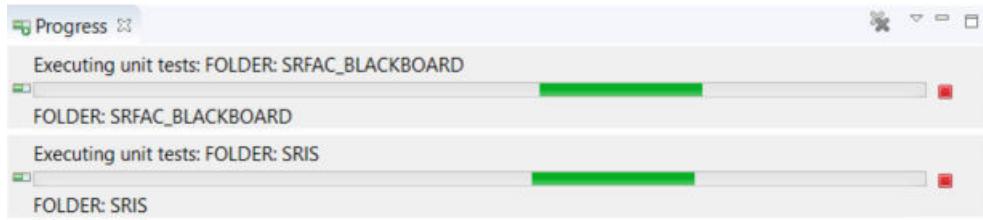
Location of Selection	Development Object	Test Class	Test Method
<a href="#">Project Explorer</a> [page 552]	✓	✓	✓
<a href="#">ABAP Editor</a> [page 553]	✓	✓	✓
<a href="#">Outline View</a> [page 555]	✓	✓	✓
<a href="#">ABAP Unit View</a> [page 556]	✓	✓	✓
<a href="#">Search View</a> [page 557]	✓	✗	✗

No matter which way you choose, ABAP unit results are displayed automatically for you in the *ABAP Unit* view.

#### i Note

ABAP Development Tools (ADT) allows you to execute tests in parallel. This means that you can execute another test while the first one is still running. The result of the previous test run will be overwritten in *ABAP Unit View* view. To see the results of the previous test runs, use [ABAP Unit Test History](#) [page 574].

The ADT shows you that the tests are running with the progress indicator  displayed in the bottom right corner of the workbench window. Double clicking on this progress indicator icon will show you the background operations in *Progress View*.



Parallel execution of tests in the Progress view

**i** Note

If you cannot execute ABAP unit tests, it can be due to a missing authorization or the system settings do not allow you to execute the tests.

## Related Information

[Unit Testing in ABAP \[page 71\]](#)

[Run Modes of ABAP Unit Tests \[page 564\]](#)

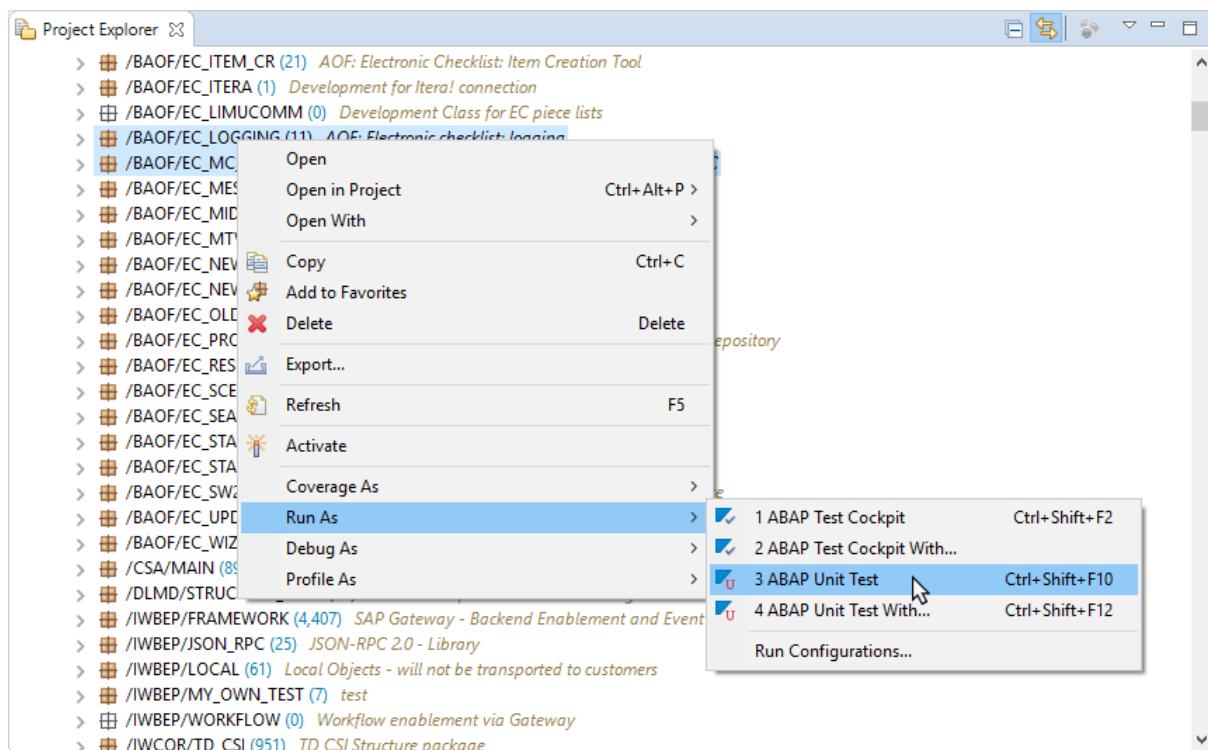
[Test-Driven Development with ABAP Unit \[page 506\]](#)

### 5.2.1.2.1 Launching ABAP Unit Tests from the Project Explorer

You can launch ABAP unit tests from the Project Explorer.

## Context

Launching tests from the Project Explorer allows you to make a multiple selection of the development objects you want to run tests for. For example, you can launch tests by selecting one or more classes, programs, folders, or packages.



Launching ABAP unit tests from the Project Explorer for multiple packages

## Related Information

- [Launching ABAP Unit Tests from ABAP Editor \[page 553\]](#)
- [Launching ABAP Unit Tests from the Outline View \[page 555\]](#)
- [Launching ABAP Unit Tests from the Search View \[page 557\]](#)
- [Launching ABAP Unit Tests from the ABAP Unit View \[page 556\]](#)
- [Launching a Preview of the ABAP Unit Tests \[page 559\]](#)
- [Run Modes of ABAP Unit Tests \[page 564\]](#)

### 5.2.1.2.2 Launching ABAP Unit Tests from ABAP Editor

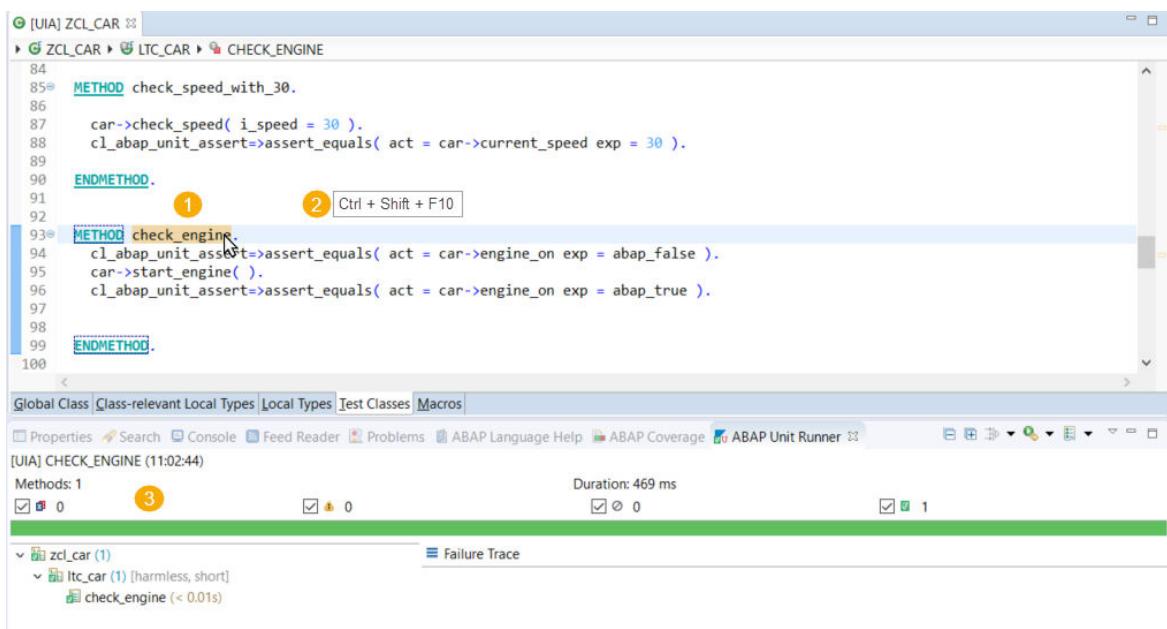
You can launch ABAP Unit tests from ABAP Editor.

## Context

While writing or changing the source code, you might want to execute all tests.

## Procedure

- Wherever you are working on your source code in ABAP Editor, the easiest way to launch all tests is by pressing a key shortcut **Ctrl** + **Shift** + **F10**.
- If you want to select a certain test method in ABAP Editor on the *Test Classes* tab, make sure that you put your cursor on the method name in the method statement in the definition or implementation part.
- If you want to launch tests for a certain test class, put your cursor on the class name in the definition or implementation part.



Launching a test method from ABAP Editor

## Related Information

- [Launching ABAP Unit Tests from the Project Explorer \[page 552\]](#)
- [Launching ABAP Unit Tests from the Outline View \[page 555\]](#)
- [Launching ABAP Unit Tests from the Search View \[page 557\]](#)
- [Launching ABAP Unit Tests from the ABAP Unit View \[page 556\]](#)
- [Launching a Preview of the ABAP Unit Tests \[page 559\]](#)
- [Run Modes of ABAP Unit Tests \[page 564\]](#)

### 5.2.1.2.3 Launching ABAP Unit Tests from the Outline View

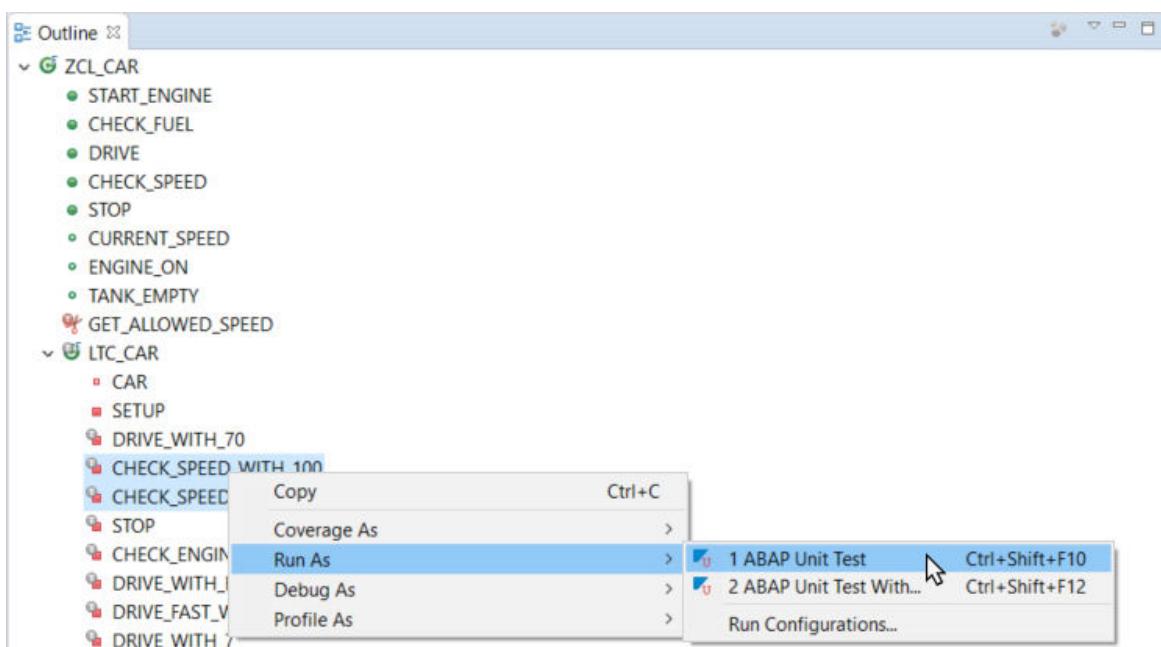
You can launch ABAP Unit tests from the *Outline* view.

#### Context

You can launch all of the ABAP Unit tests by selecting a development object, for example a class, indirectly in the *Outline* view. To launch one or more test classes directly, do the following:

#### Procedure

1. In the *Outline* view, select a test class or test method you want to launch.
2. Select **Run As > ABAP Unit Test** from the context menu to run the selected method or class. Alternatively, use key shortcut **Ctrl + Shift + F10**.



#### Related Information

[Launching ABAP Unit Tests from the Project Explorer \[page 552\]](#)

[Launching ABAP Unit Tests from ABAP Editor \[page 553\]](#)

[Launching ABAP Unit Tests from the Search View \[page 557\]](#)

[Launching ABAP Unit Tests from the ABAP Unit View \[page 556\]](#)

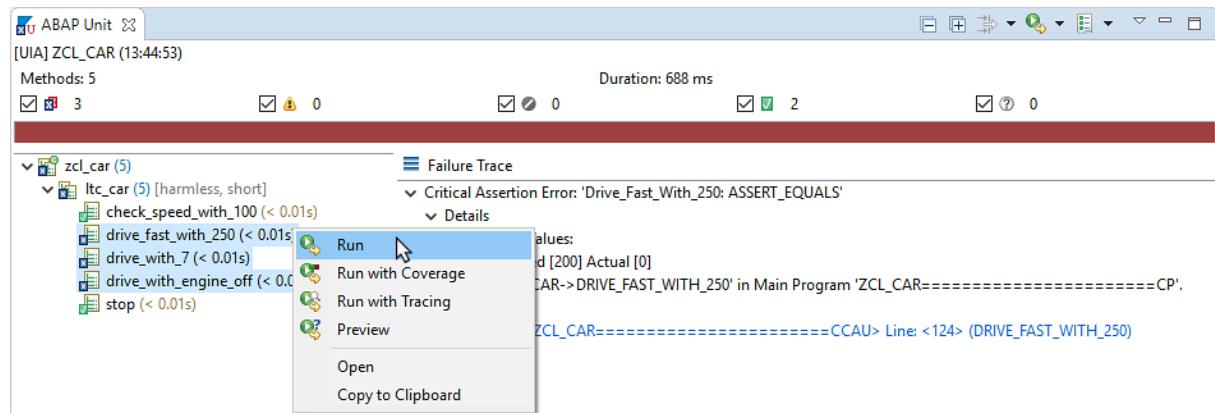
[Launching a Preview of the ABAP Unit Tests \[page 559\]](#)

[Run Modes of ABAP Unit Tests \[page 564\]](#)

## 5.2.1.2.4 Launching ABAP Unit Tests from the ABAP Unit View

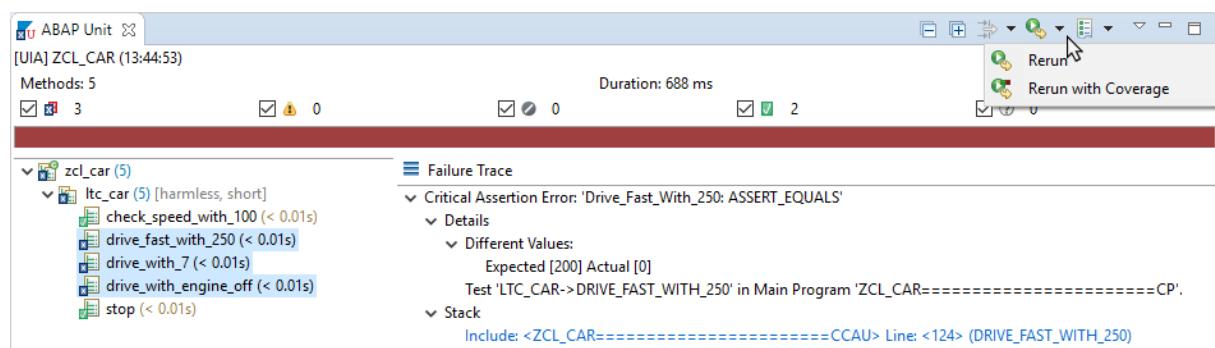
You can launch ABAP Unit tests from the *ABAP Unit* view.

If you launch your tests from the context menu, you can make a selection of test methods to run these methods only (for example, if the tests have been executed with errors). To analyze the errors, you can set a breakpoint and start debugging.



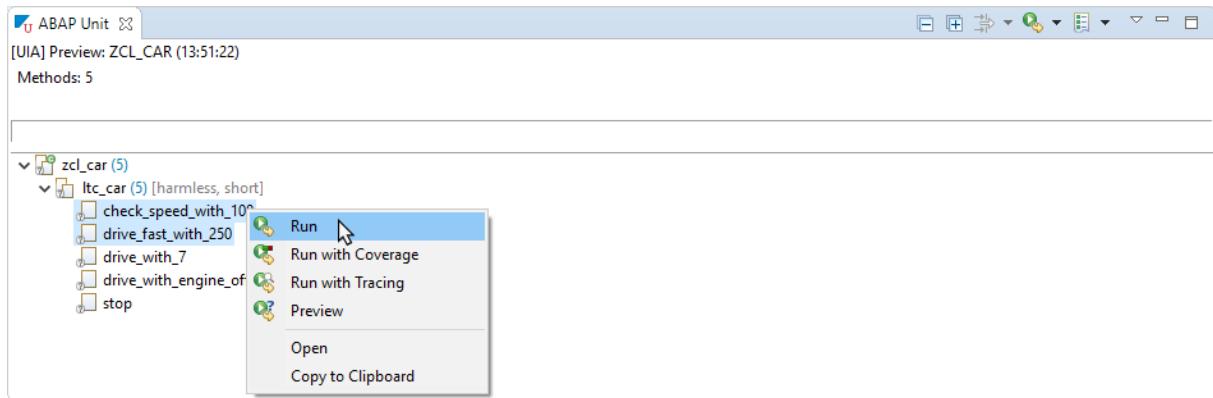
Launching a selection of tests with errors from the context menu

If you want to rerun all the tests, select in the tool bar of the *ABAP Unit* view. Alternatively, select *Rerun* or *Rerun with Coverage* in the dropdown menu of the *Rerun tool bar* button .



Launching a selection of tests with errors from the tool bar

If you run a preview for the tests, the results appear in the *ABAP Unit* view as well. To launch your tests after running a preview, select the tests that you want to run. Launch the tests from the context menu or with a key shortcut. Alternatively, you can launch all the tests using tool bar button. If you want to run the tests with the code coverage, use the dropdown menu of tool bar button..



Launching a test after running a preview

## Related Information

[Launching ABAP Unit Tests from the Project Explorer \[page 552\]](#)

[Launching ABAP Unit Tests from ABAP Editor \[page 553\]](#)

[Launching ABAP Unit Tests from the Search View \[page 557\]](#)

[Launching ABAP Unit Tests from the Outline View \[page 555\]](#)

[Launching a Preview of the ABAP Unit Tests \[page 559\]](#)

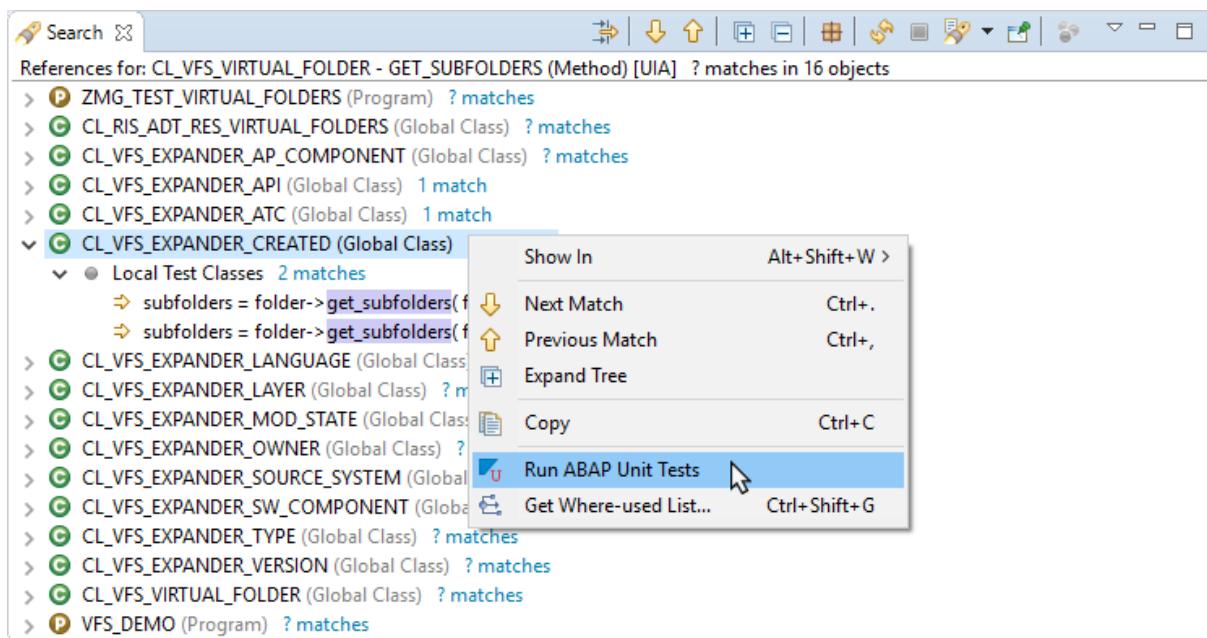
[Run Modes of ABAP Unit Tests \[page 564\]](#)

## 5.2.1.2.5 Launching ABAP Unit Tests from the Search View

You can launch ABAP Unit tests from the *Search* view.

### Context

After applying a *Where-Used* search function to an object, using the context menu or key shortcut `Ctrl` + `Shift` + `G`, you get a *Where-Used List* in the *Search* view. You can now launch unit tests. Select one or more development objects, for example classes or packages. Launch the tests from the context menu *Run ABAP Unit Tests* or by a key shortcut `Ctrl` + `Shift` + `F10`.



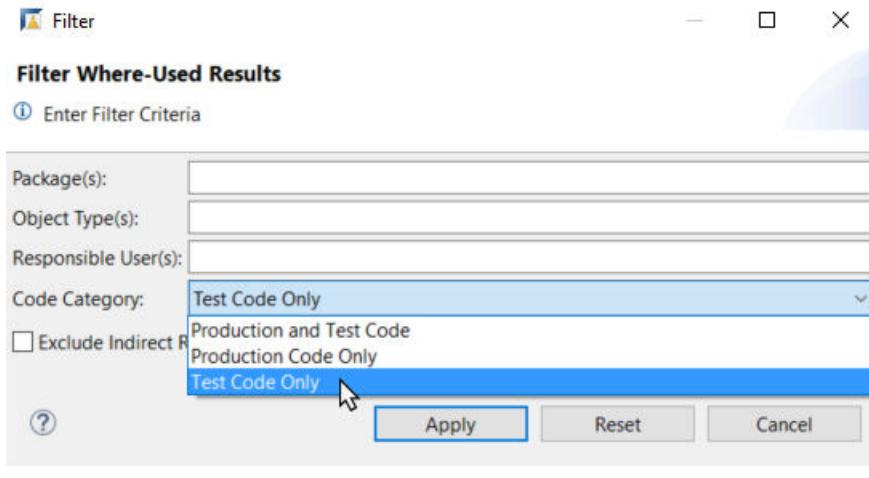
Launching unit tests from the Search view

You may also be interested in the locations in the production code where the related development object is used. Using [Where-Used List](#) in ADT, you can determine the dependencies between the objects and analyze the use of the related object.

Additionally, you might want to find the locations in test code where the object is used. You can display a list of tests that use the object as follows.

## Procedure

1. Select the target object and display the [Where-Used List](#), using context menu or key shortcut **Ctrl** + **Shift** + **G**.  
In the [Search](#) view, you get a list of occurrences where the target object is used.
2. You can filter the results using filter button  from the tool bar of the [Search](#) view.
3. In the wizard, select [Test Code Only](#) in the dropdown menu of [Code Category](#) to set the filter.



Filtering where-used results

The filter is applied now and the results in the [Search](#) view are restricted to test code.

## Results

You can now launch unit tests for the target object from the [Search](#) view.

## Related Information

- [Launching ABAP Unit Tests from the Search View \[page 557\]](#)
- [Launching ABAP Unit Tests from ABAP Editor \[page 553\]](#)
- [Launching ABAP Unit Tests from the Outline View \[page 555\]](#)
- [Launching ABAP Unit Tests from the ABAP Unit View \[page 556\]](#)
- [Launching a Preview of the ABAP Unit Tests \[page 559\]](#)
- [Run Modes of ABAP Unit Tests \[page 564\]](#)

### 5.2.1.2.6 Launching a Preview of the ABAP Unit Tests

You can launch a preview of the ABAP Unit tests to obtain an overview of the existing tests.

## Context

Wherever you can launch ABAP Unit tests, you can launch a preview of the tests as well. Even if you are not permitted to execute ABAP Unit tests, you can still run a preview.

- While working with a large amount of development objects, you might want to execute all the tests for example in a package. Running the tests can be time-consuming, depending on the amount of existing tests. Instead of launching all the tests, you can launch a preview to obtain an overview.
- If your tests are separated from the object being tested you can link them together using test relations. A preview shows you all related tests, known as foreign tests. You can get an overview of these tests and you then decide which of them you want to execute. This is especially the case if you are working with objects without their own tests, for example CDS views or simple transformations.

To launch a preview of unit tests, select one of the following options:

## Procedure

- The easiest way to launch a preview is by a key shortcut `Ctrl` + `Shift` + `F9`. Alternatively you can press `Ctrl` + `Shift` + `F12`. In the wizard you can specify whether you want to run a preview for your own tests or foreign tests, then select *Preview*.
- To launch a preview from the context menu, select  *Run As*  *ABAP Unit Test With*  In the wizard you can specify whether you want to run a preview for your own tests or foreign tests, then select *Preview*.

## Results

The results of preview runs are displayed in the *ABAP Unit* view.

## Related Information

[Launching ABAP Unit Tests from the ABAP Unit View \[page 556\]](#)

[Evaluating Preview Results of ABAP Unit Tests \[page 575\]](#)

[Launching ABAP Unit Tests \[page 551\]](#)

[Run Modes of ABAP Unit Tests \[page 564\]](#)

## 5.2.1.2.7 Launching ABAP Unit Tests Using Launch Configurations

### Context

To relaunch ABAP Unit tests for a saved list of ABAP programs, do the following:

### Procedure

1. Choose  **Run As**  from the context menu in ABAP Editor or Project Explorer. You see lists of previous runs of all types, with a section for ABAP Unit test runs. The name of the ABAP Unit test run tells you the repository object for which tests were run. A package name shows you that the saved configuration runs all ABAP unit tests in this package.
2. Double-click an entry in the ABAP unit list to repeat the test run for the saved list of ABAP programs. Or mark the *Run Configuration* and select *Run*.

### Results

A *Run Configuration* saves only the list of tested ABAP programs.

 **Note**

Delete *Run Configurations* from the list if you no longer need them. At this time, the ADT does not delete old *Run Configurations* automatically.

### Related Information

[Launching ABAP Unit Tests \[page 551\]](#)

[Evaluating ABAP Unit Test Results \[page 569\]](#)

[Evaluating ABAP Unit Code Coverage Results \[page 571\]](#)

[Test-Driven Development with ABAP Unit \[page 506\]](#)

## 5.2.1.2.8 Launching ABAP Unit with Test Relations

ABAP Unit is a framework for testing ABAP code like classes and function modules.

You can test other development objects, that cannot have **own tests**, like simple transformations and CDS views. For example, while working with a CDS view, you might want to test it by launching ABAP unit tests

directly from the CDS view. Since it is not possible to write tests in CDS views, related tests known as **foreign tests**, which are stored outside of CDS views, are used. To launch the tests from a CDS view, a relation between test code and production code has to be modeled.

To model a connection, write a specific @testing link as [ABAP doc comment](#) in front of the test method or test class.

- To test objects like classes and function modules, which can have both their own tests and foreign tests, use a key shortcut **Ctrl** + **Shift** + **F12**. In the launch dialog, go to the **Scope** section and you can select **own tests** and/or **foreign tests** to be executed. Then select [Execute](#). You can select [Preview](#) in the wizard to get an overview of the tests.
- To test objects like CDS views or simple transformations, which can have only foreign tests, use a shortcut **Ctrl** + **Shift** + **F10** to launch the tests directly from the object. You can press **Ctrl** + **Shift** + **F9** to get an overview of the tests.

When tests are selected indirectly by choosing a set of repository objects (**object selection**) it is possible not only to execute the objects' **own tests**, but also any **foreign tests** that are not contained in the selection but have a test relation pointing to an object inside the selection.

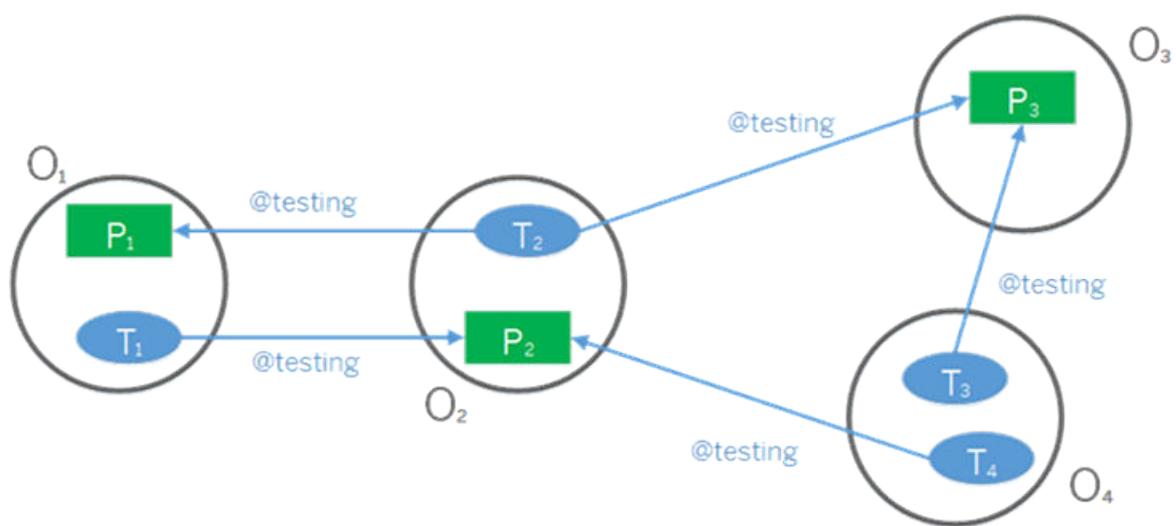
Tests	Inside object selection	Related to an object inside the selection
Own tests	✓	Does not matter
Foreign tests	✗	✓

Foreign tests can be executed in different ways, depending on the type of selected object.

## Example

### No object selected

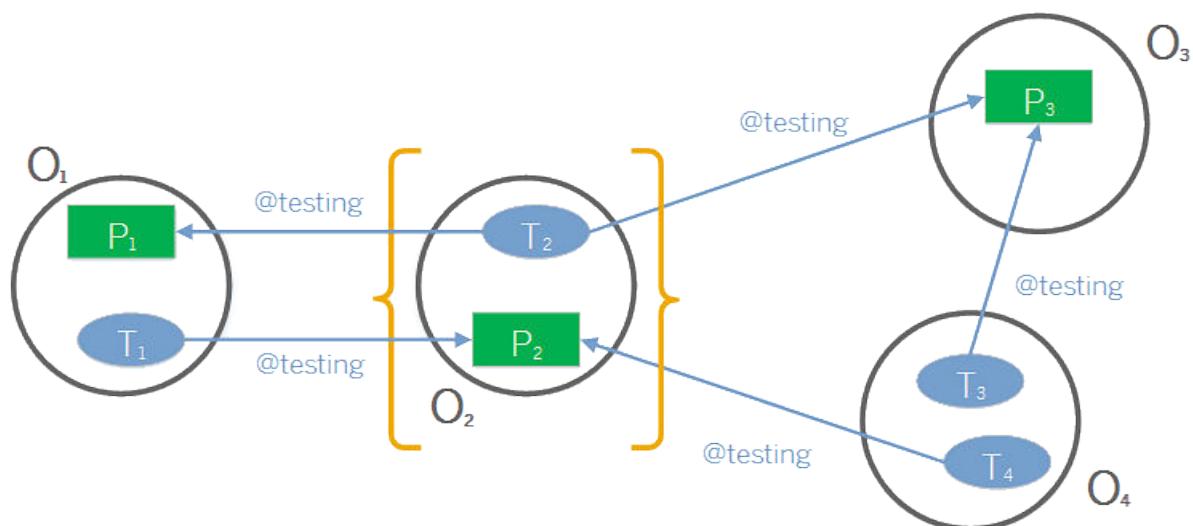
In the figure below you see some repository objects ( $O_i$ ) containing test code ( $T_i$ ) and production code ( $P_i$ ) with some test relations ( $\langle @testing \rangle$ ) defined. But you cannot decide whether one of the tests is an object's own test or a foreign test, because there is no current object selection.



#### Selection of a single object

In the following figure there is a current selection  $\{O_2\}$ . Test  $T_2$  is contained in the selection (object's own test) while  $T_1$  and  $T_4$  are not but do have test relations pointing to  $P_2$ , which is inside the selection.

- Object's own tests =  $\{T_2\}$
- Foreign tests =  $\{T_1, T_4\}$

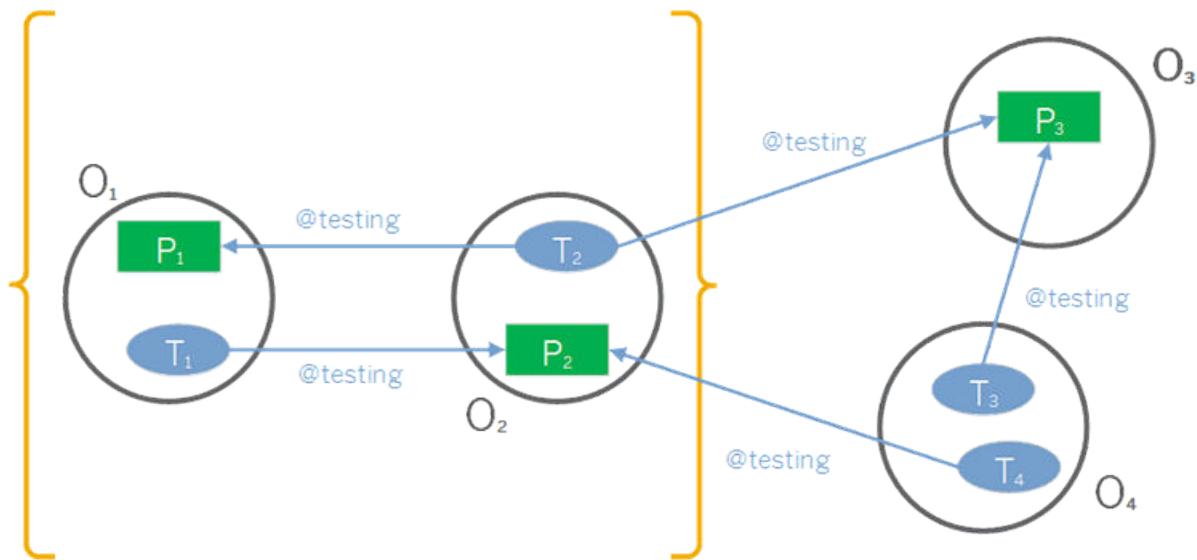


#### Selection of two objects

Finally, you can widen the selection to  $\{O_1, O_2\}$  and get a smaller set of foreign tests.

- Object's own tests =  $\{T_1, T_2\}$

- Foreign tests = {T<sub>4</sub>}



## Related Information

[Writing ABAP Unit with Test Relations \[page 550\]](#)

[ABAP Unit Launch Dialog \[page 565\]](#)

### 5.2.1.2.9 Run Modes of ABAP Unit Tests

There are several modes available for running ABAP Unit tests.

A mode of execution is chosen depending on how informative you want the results of the tests to be.

You can see the different modes of ABAP Unit tests with their possible execution variants in the following table.

Launching Unit Tests in Different Modes

Modes of ABAP Unit	Executed From				
	Test	Key Shortcut	Context Menu	Tool Bar	Menu Bar
Unit Test		[Ctrl] + [Shift] + [F10]	[Run As] > ABAP Unit Test	[Run As] > Run ABAP Unit Test	[Run As] > Run ABAP Unit Test

Modes of ABAP Unit	Executed From				
	Test	Key Shortcut	Context Menu	Tool Bar	Menu Bar
Unit Test Preview		<code>Ctrl</code> + <code>Shift</code> +  <code>ABAP Unit</code> <code>F9</code>			
Unit Test With Coverage		<code>Ctrl</code> + <code>Shift</code> +  <code>Coverage As</code> <code>F11</code>		  	 
Unit Test With Trace	Not assigned		 <code>Profile As ABAP</code> 	 <code>ABAP Unit Test</code> 	 <code>ABAP Unit Test</code> 
Launch Dialog With All Options		<code>Ctrl</code> + <code>Shift</code> +  <code>ABAP Unit</code> <code>F12</code>			

## Related Information

[ABAP Unit Launch Dialog \[page 565\]](#)

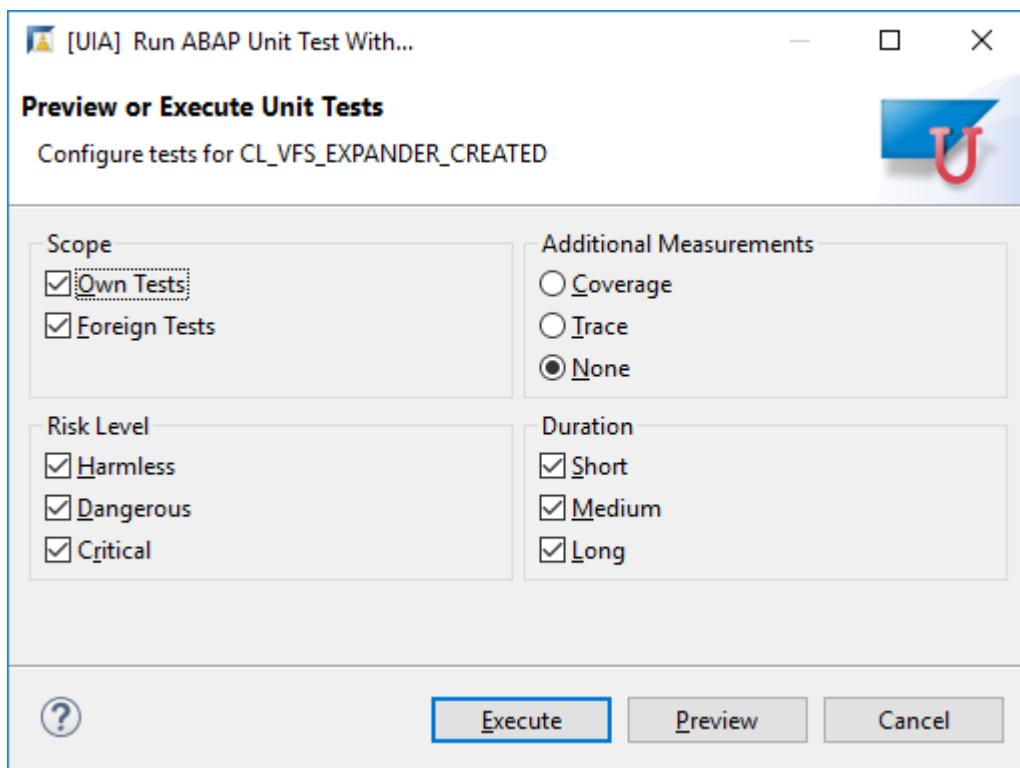
[Launching ABAP Unit Tests \[page 551\]](#)

### 5.2.1.2.9.1 ABAP Unit Launch Dialog

To configure your test execution, start a launch dialog with the key shortcut `Ctrl` + `Shift` + `F12`.

- Filter the amount of tests:
  - In the wizard, you can use the *Scope* section to include foreign tests to the test suite. Foreign tests can be defined by using [test relations \[page 561\]](#).
  - In the *Risk Level* section, you can reduce the amount of executed tests based on certain risk levels.
  - In the *Duration* section, you can reduce the amount of executed tests based on certain duration categories.
- Enable additional measurements:
  - In the *Additional Measurements* section, you can enable coverage measurement or tracing.

Finally, you can choose to execute the tests or just get a [preview \[page 575\]](#) of the tests.



Launch Dialog

## Related Information

- [Run Modes of ABAP Unit Tests \[page 564\]](#)
- [Launching ABAP Unit Tests \[page 551\]](#)
- [Launching ABAP Unit with Test Relations \[page 561\]](#)
- [Evaluating ABAP Unit Code Coverage Results \[page 571\]](#)
- [Evaluating ABAP Unit Trace Results \[page 573\]](#)

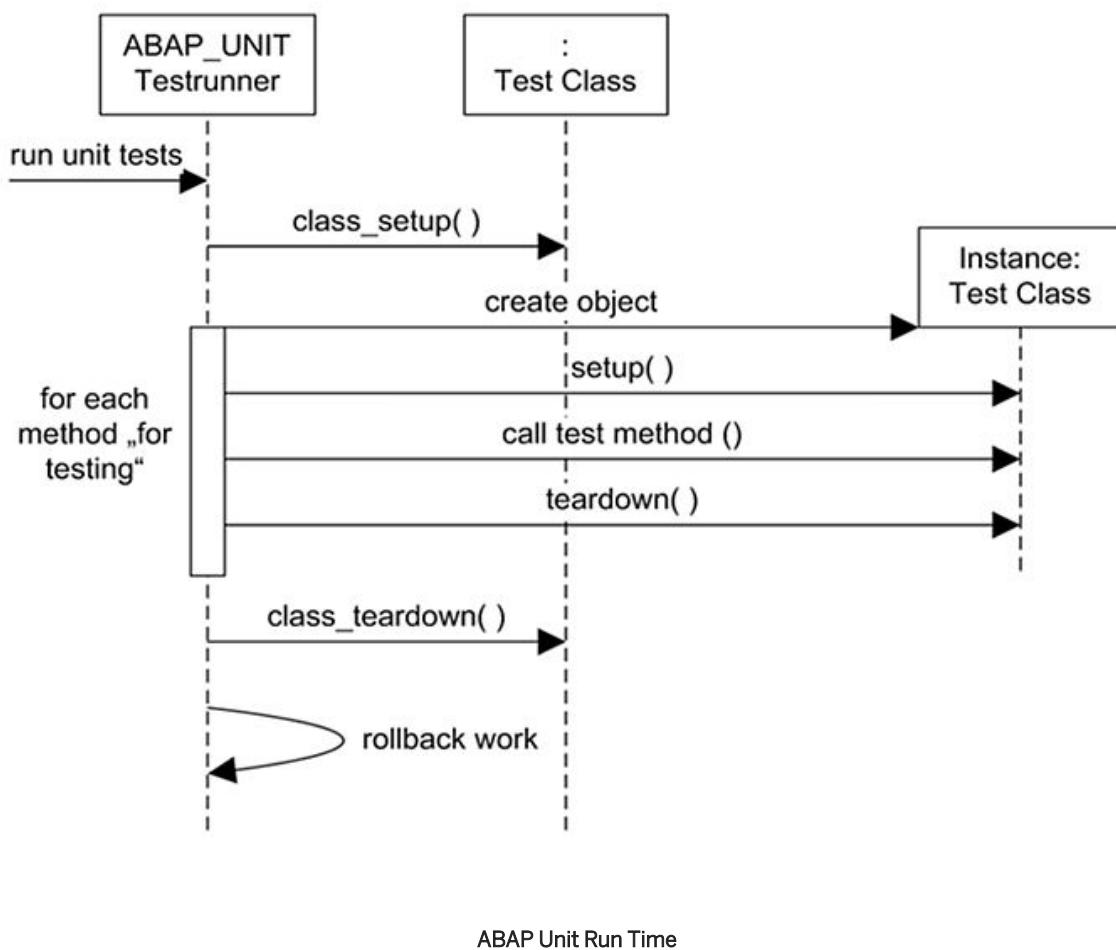
## 5.2.1.2.10 ABAP Unit Test Execution Sequence

A local test class can contain the private instance methods `setup` and `teardown`, which are executed before and after each test method. Besides this, it can also contain the private static methods `class_setup` and

`class_teardown`, which are executed at the beginning and at the end of each test class. For more information, see [CLASS - FOR TESTING \(ABAP Keyword Documentation\)](#).

When executing ABAP Unit tests, consider the following:

- All test methods are executed independently from each other.
- The test methods are executed in an undefined order.
- A new instance of the test class is created for each test method.
- [ROLLBACK WORK \(ABAP Keyword Documentation\)](#) is called at the end of each test class.

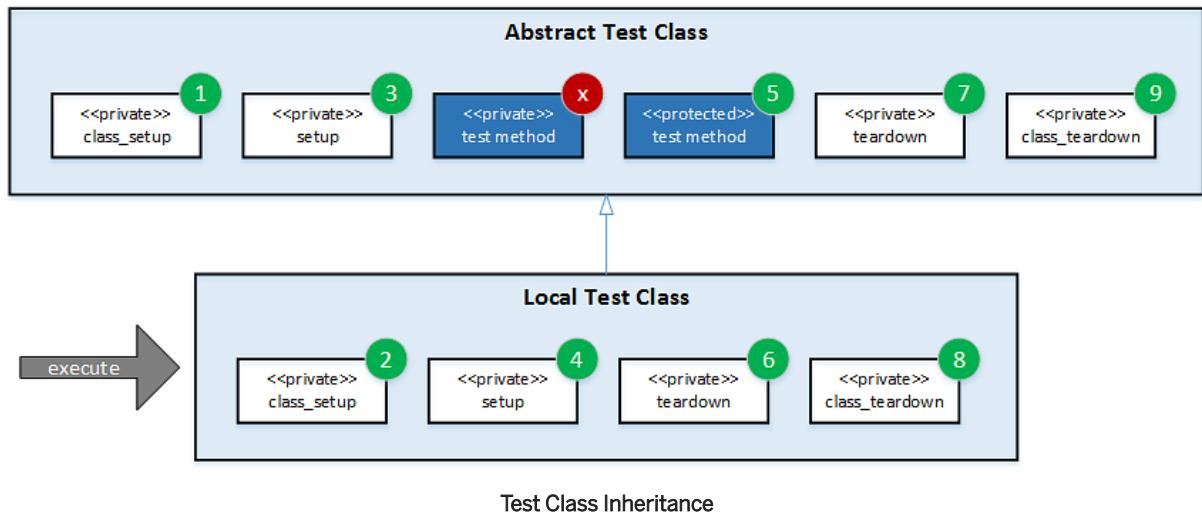


## Test Class Inheritance

When ABAP Unit executes a test class, methods from its superclass are executed as well. The figure below shows which methods of subclasses and superclasses are executed and in which order.

First, the `class_setup` method of the superclass is called, followed by the `class_setup` method of the subclass. Then the `setup` method of the subclass is called for each test method. After the execution of the test

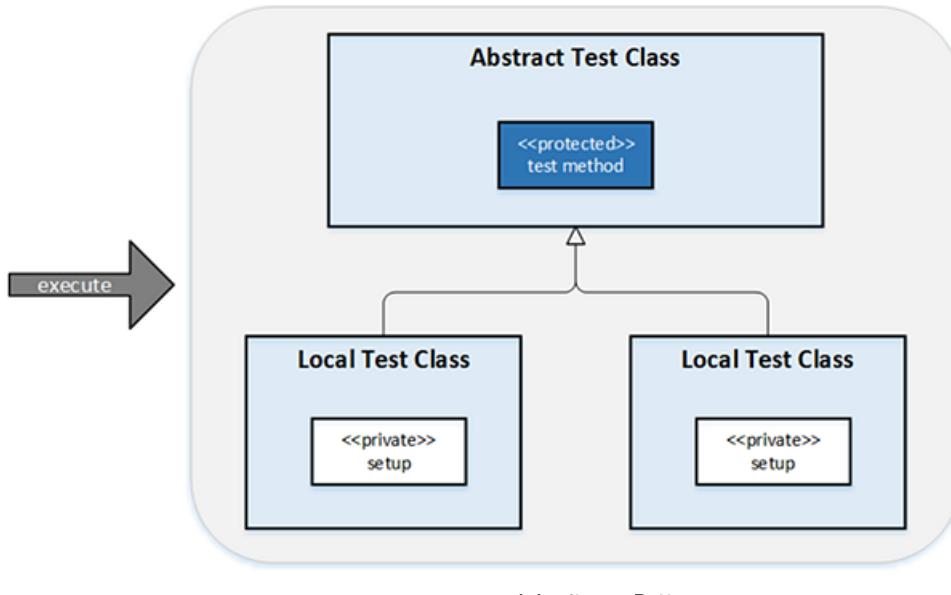
methods, the `teardown` methods of the sub class and the superclass are called. Finally, the `class_teardown` method of the subclass is called, followed by the `class_teardown` method of the superclass.



### Execute the same test method with different setups

If you want to execute the same test with different configurations, you can use the following pattern:

1. Write your tests as protected methods in the abstract superclass.
2. Create a subclass for each configuration and establish this configuration in the respective `setup` method.



## Related Information

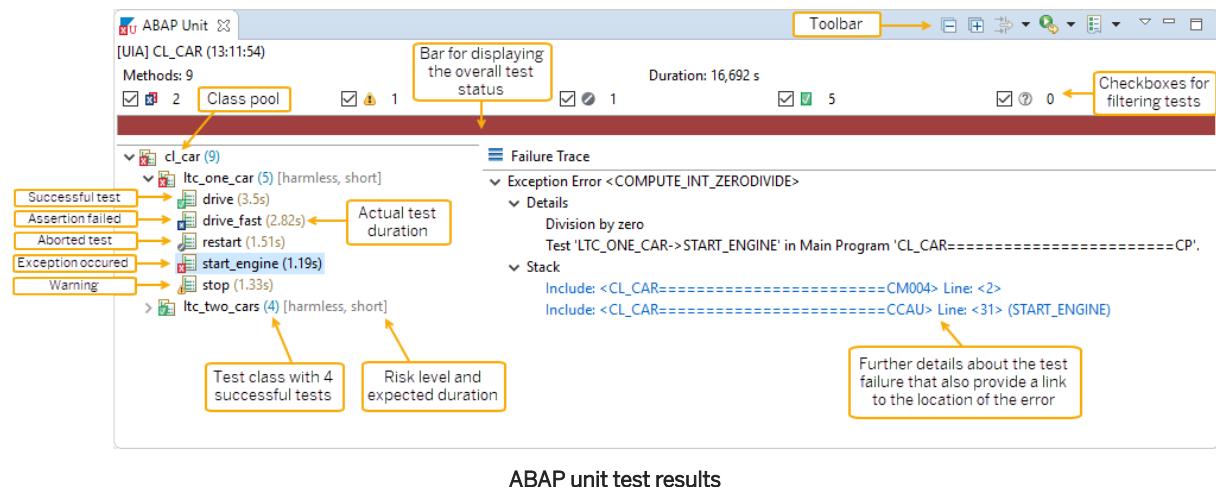
[Launching ABAP Unit Tests \[page 551\]](#)

### 5.2.1.3 Evaluating ABAP Unit Test Results

You evaluate the results of ABAP Unit tests to find out whether your objects under tests behave as expected.

#### Context

When you execute ABAP unit tests, ABAP Development Tools (ADT) automatically opens the *ABAP Unit* view.



The *ABAP Unit* view provides various information at the level of program, test class, test method, and total information. Total information is displayed at the top of the *ABAP Unit* view. The program, test class, and test method levels are displayed in the result tree. The table below shows what information you can see at these levels.

Information at various levels

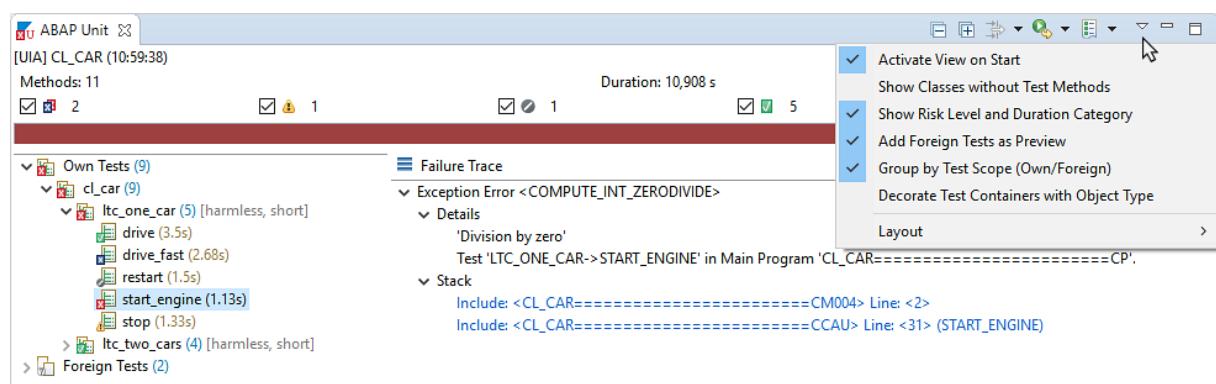
Levels	Counter	Actual test duration	Risk level	Expected test duration
Program	✓	✗	✗	✗
Test class	✓	✗	✓	✓
Test method	✗	✓*	✗	✗
Total information	✓	✓	✗	✗

## i Note

\* The detailed duration of a test run is only displayed if it is longer than 0.01 seconds. Otherwise, ADT displays generic information, such as "< 0.01s".

The status of the test method execution in the result tree is projected onto the status of the test class and program. For example, if at least one test method produces an error, the corresponding test class is displayed as failed.

By clicking the drop-down menu in the upper right corner of the *ABAP Unit* view you can show classes without test methods, hide the risk level and duration of the tests, hide foreign tests, group tests by the test scope, and decorate test containers with object type.



Hiding the risk level and duration of the tests

**Filtering checkboxes:** Restrict the number of displayed unit tests, for example display only those that have an error. To do this, select one or more of the following checkboxes from the bar above the result tree.

## → Tip

If you right-click one of the selected checkboxes, the other checkboxes are deselected. If you right-click again, all checkboxes are selected. This enables you to deselect the other checkboxes or select all checkboxes in one step.

## i Note

If you execute tests for objects like classes, function groups, or reports that have both own tests and foreign tests assigned to the objects using [test relations \[page 561\]](#), the results of the foreign test run are displayed as a preview in the *ABAP Unit* view besides the results of the own test run.

**Detailed error analysis:** The information about errors, warnings, terminations, or failures are shown in the *Failure Trace*. Here you can see the following information:

- Failure description

- The position in the source code where the problem occurred. To get to the position, select the link under *Stack* at the bottom of the *Failure Trace*. There you can analyze the error, for example by toggling a breakpoint and starting to debug.
- If a runtime error occurred during the ABAP Unit test run, the link *Show Runtime Error* appears in the *Failure Trace*. By clicking the link you can start [analyzing the runtime error \[page 710\]](#) in the *ABAP Runtime Error Viewer*.

## Related Information

- [Evaluating ABAP Unit Code Coverage Results \[page 571\]](#)
- [Evaluating ABAP Unit Trace Results \[page 573\]](#)
- [Using the History to Switch Between Results of ABAP Unit Tests \[page 574\]](#)
- [Evaluating Preview Results of ABAP Unit Tests \[page 575\]](#)
- [Debugging ABAP Code \[page 613\]](#)
- [Analyzing ABAP Runtime Errors \[page 710\]](#)

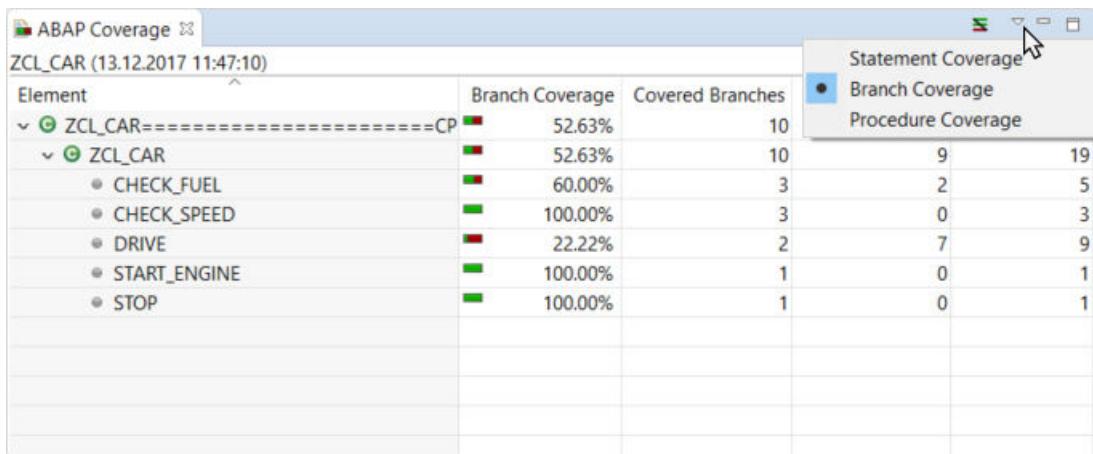
### 5.2.1.3.1 Evaluating ABAP Unit Code Coverage Results

#### Context

To run ABAP Unit tests with code coverage measurement, do the following:

#### Procedure

1. Start a code coverage measurement. Choose  *Coverage As*  from the context menu of ABAP Editor or Project Explorer.  
The coverage display appears separately from the ABAP Unit results display, in the *ABAP Coverage* view.  
The view shows you the code coverage that was achieved by running the ABAP Unit tests.
2. Use the **View Menu** of the **ABAP Coverage** view to switch to branch coverage or procedure (processing block) coverage.



Selection of a code coverage measurement

3. Double-click a class, method, or other processing block in the ABAP Coverage view to navigate to the code.

Here you can see the coverage at the source-code level. The source code display is related to the type of coverage you are displaying in the *ABAP Coverage* view. You can easily detect any gaps in your testing:

The meaning of shading in the source code

**Type of Coverage Measure-**

ment	Green Shading	Yellow Shading	Red Shading
Statement coverage	The statement has been executed	not applicable	The statement has not been executed
Branch coverage	All branches have been executed	At least one branch has not been executed	No branch has been executed
Procedure coverage	The procedure has been called	not applicable	The procedure has not been called

```

87
88
89 METHOD drive.
90
91 IF engine_on = abap_false.
92   RETURN.
93 ENDIF.
94
95 IF tank_empty = abap_false.
96   RETURN.
97 ENDIF.
98
99 IF i_speed > 200.
100   current_speed = max_speed.
101
102

```

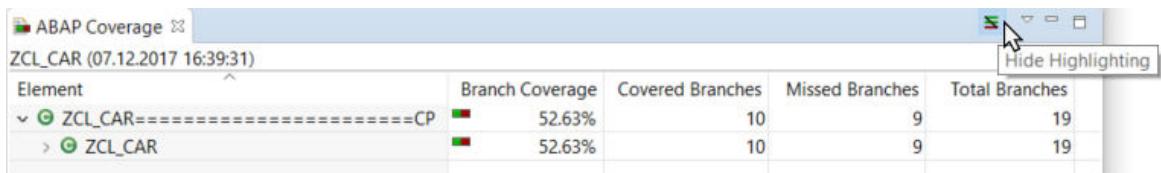
Branch coverage at the source-code level

4. Move the cursor over a line of code to see how many times the line was executed during the coverage measurement. In branch coverage, you see how often the branch resolved to 'true' and 'false'.

Once you are in the code-level coverage display, you can navigate as usual (with F3). The code coverage shadings are applied to the code to which you navigate too.

#### → Tip

You can hide the shading by editing your source code or clicking the *Hide Highlighting* icon in the tool bar of the ABAP Coverage view.



Element	Branch Coverage	Covered Branches	Missed Branches	Total Branches
✓ ZCL_CAR=====CP	52.63%	10	9	19
> ZCL_CAR	52.63%	10	9	19

Hiding the shading of the source code

## Results

#### i Note

- If there is an ABAP Unit assertion, ADT shows you the *ABAP Unit* view first, instead of the *ABAP Coverage* view. Click the *ABAP Coverage* icon to open the coverage view, if it is not open. In this case, the coverage measurement is incomplete.
- By default, code coverage is shown to you as statement coverage by program and by processing block (method, function module, or subroutine).
- In the processing block view, coverage is shown for all processing blocks of the tested ABAP programs, regardless of whether code in a processing block was actually executed by a test. You can therefore see where you still need test.

## Related Information

[Changing Colors of Annotation Types \[page 313\]](#)

### 5.2.1.3.2 Evaluating ABAP Unit Trace Results

You can use ABAP Unit with a trace for runtime analysis.

If you execute ABAP Unit tests with a trace, the results are displayed automatically in the *ABAP Traces* view. A trace file for each test class is generated.

## Related Information

[Displaying the Trace Overview \[page 649\]](#)

[Profiling ABAP Code \[page 640\]](#)

[Run Modes of ABAP Unit Tests \[page 564\]](#)

### 5.2.1.3.3 Using the History to Switch Between Results of ABAP Unit Tests

You can display previous test runs in the *ABAP Unit Test History*.

#### Context

The ABAP Unit test history enables you to switch between the results of different test runs without performing them again.

This saves you time and gives you immediate access to test runs that have already been evaluated.

#### i Note

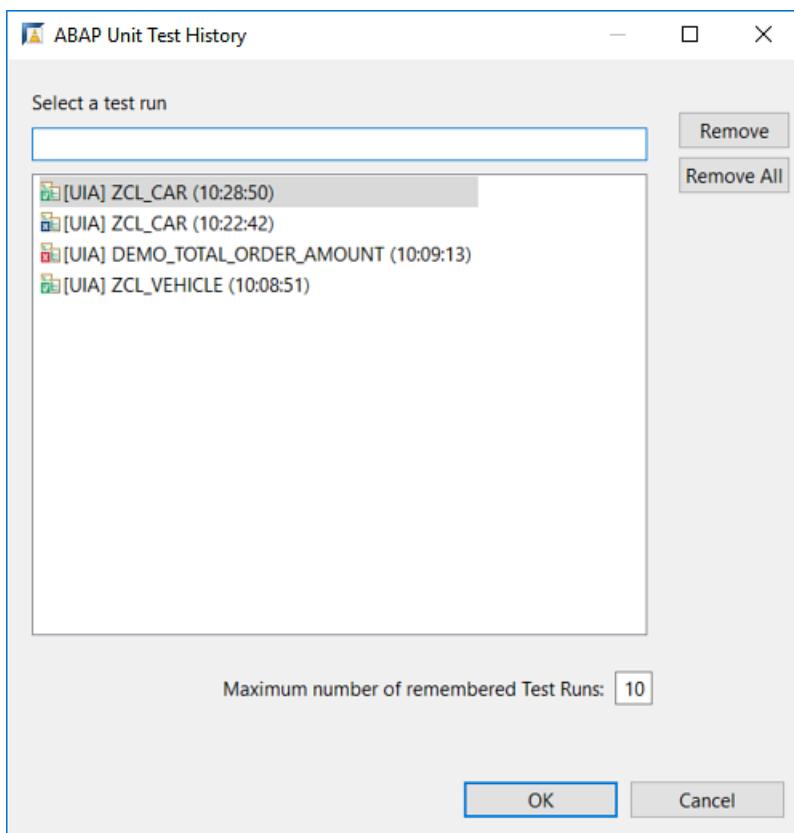
When you perform a test run, the previous result is overwritten but not deleted. The history is only deleted when you close or restart the IDE.

#### Procedure

To display the *ABAP Unit Test History*, proceed as follows:

1. Choose the arrow in the  *History...* icon displayed in the toolbar in the *ABAP Unit* view.
2. Choose *History...* from the menu.

The *ABAP Unit Test History* is opened.



#### Example of an ABAP unit test history

3. To open the relevant result, double-click it.

You can preconfigure the history to display up to the last 99 test runs. To do this, enter the required number in the *Maximum number of remembered test runs* input field.

The result of the selected test run is displayed in the *ABAP Unit* view.

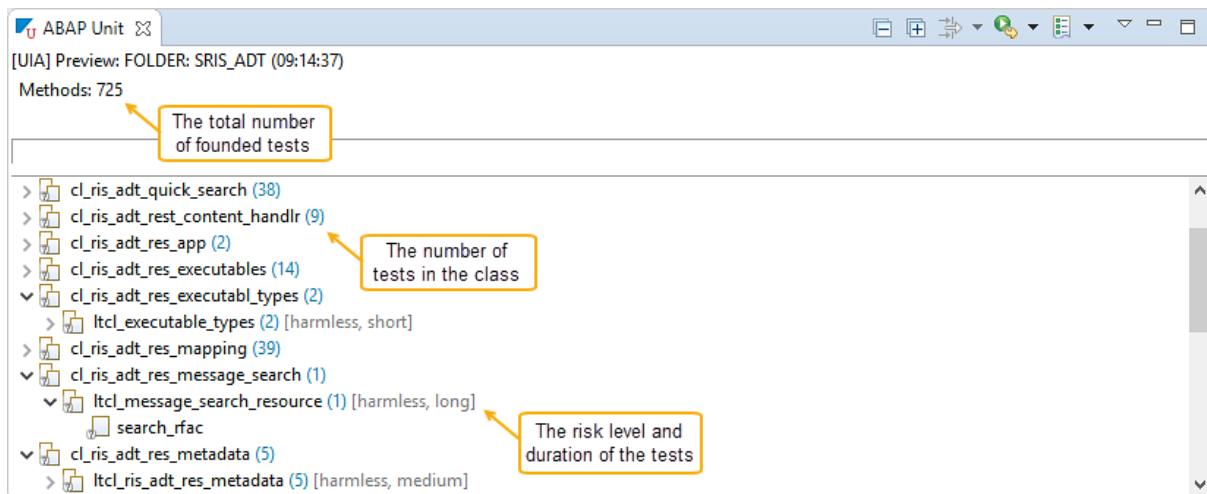
4. To remove a specific test run from the history, select it and choose *Remove*. To remove all results from the history, choose *Remove all*.

### 5.2.1.3.4 Evaluating Preview Results of ABAP Unit Tests

After a preview of the tests is run, the results are displayed automatically in the *ABAP Unit* view, just like the results of any executed ABAP Unit tests. This means that you can see all test methods grouped by test classes and programs. As a next step, you can choose any subset or even all of the tests to execute them.

The preview results differ from execution results in the following aspects:

- It is quicker to run a preview than to execute ABAP Unit tests.
- In the preview results, you do not see how long it takes to run a preview.
- In the preview results, there is no colored status bar.
- In the preview results, the tests are not filtered by error category. Every test has the same status, 'not executed'.



The results of the preview launch

You can now get an overview of the total number of tests under the object, as well as of the number of test classes. If you expand the classes you can navigate to local test classes and see the number, the risk level, and the duration of the tests.

## Related Information

[Launching a Preview of the ABAP Unit Tests \[page 559\]](#)

[Launching ABAP Unit Tests from the ABAP Unit View \[page 556\]](#)

## 5.2.1.4 ABAP Unit Glossary

### depended-on component

Component on which the objects under test depend. Examples of depended-on components are classes, function calls, and database tables. For more information, see [Depended-on components \[page 509\]](#).

### foreign test

Related tests, located outside of the objects under test, contrary to own tests. For more information, see [Foreign tests \[page 561\]](#).

## object under test

Tested development objects such as ABAP programs, classes, or CDS objects. For more information, see [Object under test \[page 509\]](#).

## own test

Tests located in the same repository object as the object under test. For more information, see [Own tests \[page 561\]](#).

## production code

We use the term **production code** for any code that is not test code.

## repository object

Development objects such as ABAP programs, classes, or CDS objects.

## test

We use the term **test** for a test method.

## test class

Local class that contains test methods for testing production code. For more information, see [Where to Write ABAP Unit Test Classes \[page 502\]](#).

## test code

Code written to test an object under test. For more information, see [Writing ABAP Unit Tests \[page 501\]](#).

## **test container**

Generic term for repository objects that can contain tests. For more information, see [Writing ABAP Unit with Test Relations \[page 550\]](#)

## **test double**

Object created to replace a depended-on component. It provides the same API as the real object so that the object under test can use it. For more information, see [Test Double with ABAP Unit \[page 510\]](#).

## **test method**

A method of a test class.

## **test relation**

A link between a foreign test and an object under test. It is especially important for objects that cannot have own tests, such as CDS views. For more information, see [Test relations \[page 561\]](#).

## **test run**

A test or test suite can be executed many times, each time potentially yielding a different test result. We use the term test run for any execution of tests, whether they are still running or already finished.

## **test suite**

A collection of tests that are executed together. For more information, see [Launching ABAP Unit Tests \[page 551\]](#).

## 5.2.2 Checking Quality of ABAP Code with ATC

The ABAP Test Cockpit (ATC) is the standard tool for checking the quality of ABAP development objects using static checks.

As an ABAP developer, you can use ATC tools in the Eclipse-based ABAP Development Tools (ADT) to find potential bugs already during the development phase.

In particular, you can check the ABAP development objects for many types of problems, including syntax errors, potential problems in performance, suboptimal or potentially faulty programming, adherence to standards, errors in ABAP Unit testing, among others.

### Related Information

[Introduction \[page 579\]](#)

[Working with ATC During Development \[page 581\]](#)

[Working with ATC During Transport Release \[page 600\]](#)

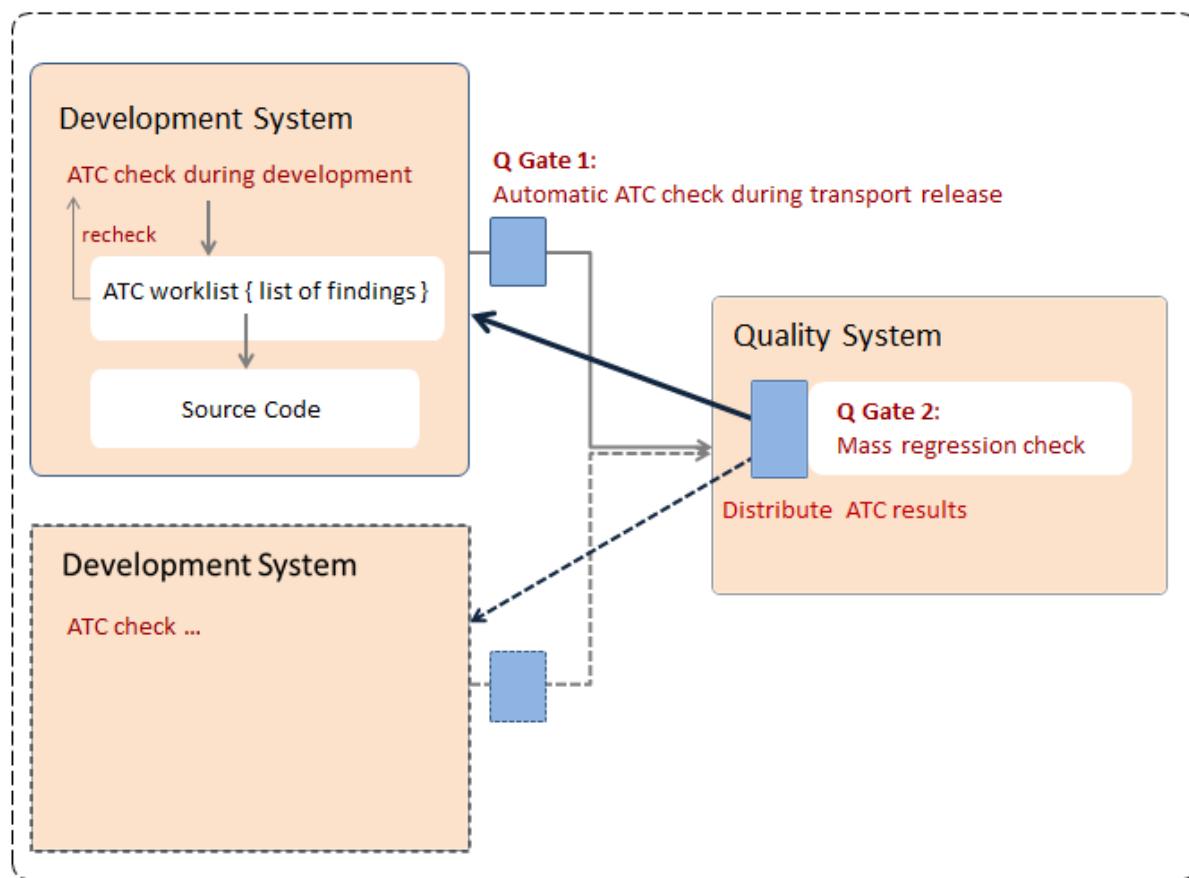
[Working with Central ATC Results \[page 602\]](#)

### 5.2.2.1 Introduction

#### Prerequisites

- Using ATC tools in your development landscape requires that the setup for ATC is completed.
- To run the ATC check, you need the respective authorizations for the ABAP Test Cockpit.
- You need at least ABAP Development Tools client version 2.31.

## Usage Scenarios



ATC quality check as an integral part of the development and delivery process

ABAP developers have various options for using ATC tools. In the current shipment, they can use ATC in the following use cases:

1. Running ATC checks during development  
Developers run the ATC checks to find potential bugs already during the development and launch of ATC from within their IDE.  
**More on this:**[Working with ATC During Development \[page 581\]](#)
2. Running ATC checks during transport release  
During the transport release, the ATC implicit check run for all development objects that are included in the transport request provides the first Q Gate (quality gate).  
**More on this:**[Working with ATC During Transport Release \[page 600\]](#)
3. Working with central ATC results from mass regression check runs that have been scheduled in a central quality system  
In the target quality system, the mass regression check runs are scheduled and serve as the final Q Gate before the code changes are released.  
**More on this:**[Working with Central ATC Results \[page 602\]](#)
4. Handling exemptions (false positives) for ATC findings  
ATC tools integrate an exemption process for false positives in order to handle findings that cannot be cleaned up. In a separate process, the developer requests an exemption for a finding. The quality manager then approves or rejects the request.

More on this: [Requesting Exemptions for ATC Findings \[page 611\]](#)

## 5.2.2.2 Working with ATC During Development

### Use Case

(1) The developer runs ATC checks frequently during development to find and immediately clean up defects in smaller development object units, like in a single ABAP class for example.

(2) The developer runs ATC checks up until the implementation of a certain functionality is complete and the ATC result is then used to clean up the code in bigger development object units, like a package or transport request.

### Activities Relevant for Developers

- [Launching ATC Check Run from the Project Explorer \[page 581\]](#)
- [Launching ATC Check Run from the ABAP Editor \[page 583\]](#)
- [Working with the ATC Problems View \[page 584\]](#)
- [\[Optional\] Selecting Check Variant \[page 597\]](#)
- [\[Optional\] Requesting Exemptions for ATC Findings \[page 611\]](#)

### Related Information

[ATC Quality Checking \[page 72\]](#)

#### 5.2.2.2.1 Launching ATC Check Run from the Project Explorer

Each ATC check run uses a global check variant that contains a set of relevant checks. For each system, a system default check variant is already predefined.

##### → Remember

The standard check variant in your development system may have been set by the central ATC administrator. If no such setting has been made, then it is the `DEFAULT` check variant of the *Code Inspector* (transaction `SCI`).

However, you (as developer) have the option to change the predefined check variant at the project's level. **See also:** [Selecting Check Variant \[page 597\]](#). If you then start an ATC check run with [Run As > ABAP Test Cockpit](#), the project-specific check variant determines which checks are run.

In addition, you can start an ATC check run with the [Run As > ABAP Test Cockpit With...](#) to specify an arbitrary global check variant. You can use this option to override the predefined ATC check variant for that specific check run. That way, you can run your own set of checks, perhaps a more stringent set as defined in the corporate standard.

## Procedure

### To run ATC checks with a predefined check variant:

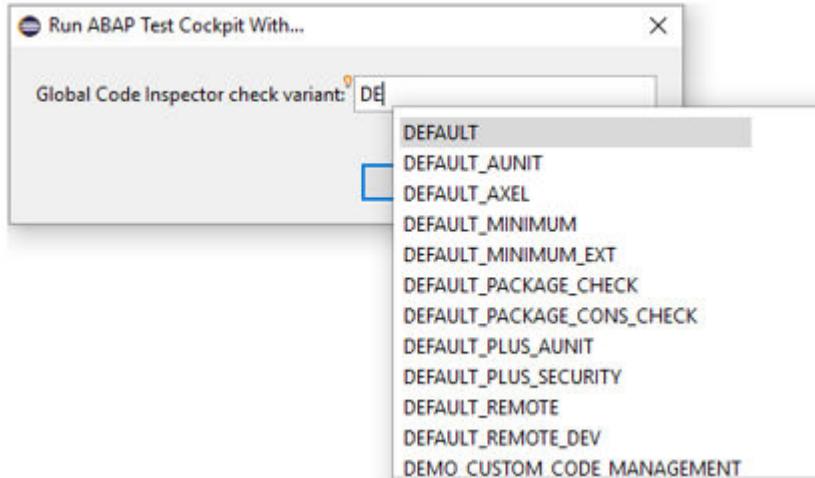
1. Select the relevant ABAP project in ADT.
2. In the [Project Explorer](#), select one or more development objects to be checked.
3. Choose [Run As > ABAP Test Cockpit](#) from the context menu or alternatively use the key shortcut [`Ctrl + Shift + F2`](#).

### To override the predefined check variant for an ATC check run:

1. Select the relevant ABAP project in ADT.
2. In the [Project Explorer](#), select one or more development objects to be checked.
3. Choose [Run As > ABAP Test Cockpit With...](#) from the context menu.
4. Specify the desired check variant and start the check run with [OK](#).

#### → Tip

When editing the name of the check variant, you can avail of the content assistant functionality by pressing [`Ctrl + Space`](#) in the corresponding field.



Selecting a check variant for the current ATC check run

#### → Remember

The check variant that you have chosen is used only for this ATC check run. Other runs revert to the predefined check variant - unless you once again choose a different check variant.

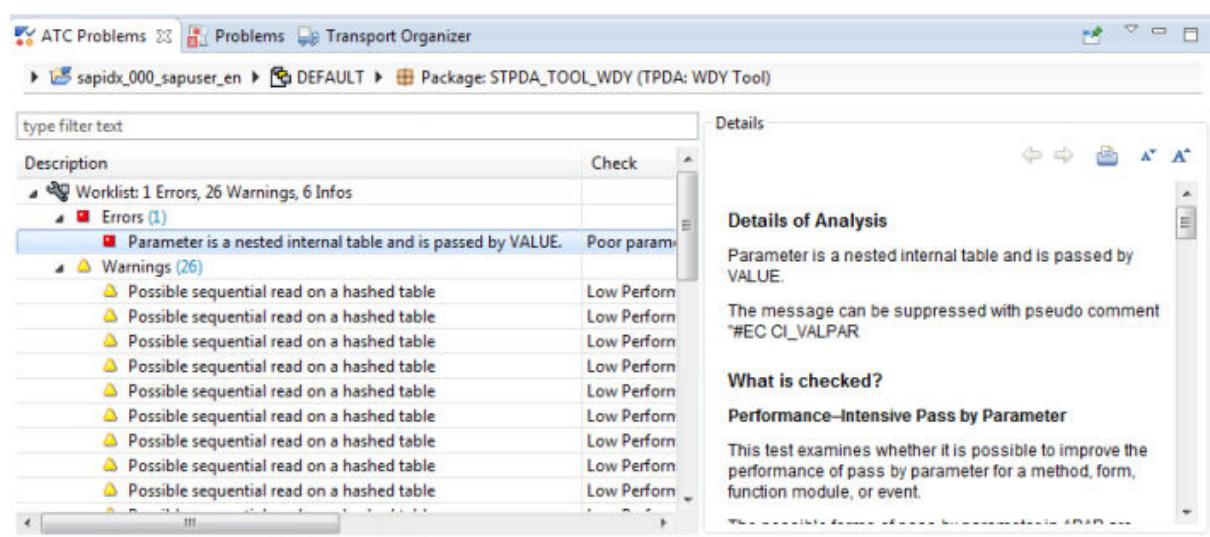
## Results

The ATC checks all of the repository objects contained in the elements you selected in the *Project Explorer* tree. Doing this for a package runs all ATC checks for all of the objects in the package.

The ADT shows you that the checks are running with *Running ATC checks* progress indicator. After completion of the ATC check run, the new ATC check result (set of all findings) is displayed in the *ATC Problems* view and added to the developer's *Worklist*.

### → Remember

You can think of a **Worklist** as a set of all findings for all ATC check runs that have been executed by the developer in the local system.



ATC Problems view displays the worklist with ATC findings

## Related Information

[Launching ATC Check Run from the ABAP Editor \[page 583\]](#)

### 5.2.2.2.2 Launching ATC Check Run from the ABAP Editor

## Context

You can run ATC quality checks on the class or other development object that you are editing in the ABAP editor. No matter how you start ATC checks, the ADT automatically displays the findings in the *ATC Problems* view.

## Procedure

1. Open the relevant development object in the ABAP editor.
2. Place the cursor at any position in the editor.
3. Choose the menu item  **ABAP Test Cockpit**  from the context menu in the ABAP editor or alternatively use the key shortcut **Ctrl** + **Shift** + **F2**.

### Note

Alternatively, you can start an ATC check run with the  **ABAP Test Cockpit With...**  to specify an arbitrary global check variant. You can use this option to override the predefined ATC check variant for that specific check run. **See also:** [Launching ATC Check Run from the Project Explorer \[page 581\]](#)

## Results

If your object has passed all of the checks included in one ATC check run, you will see an information message in the status message area of the ADT. After completion of the ATC check run, the new ATC check result is displayed as a *Worklist* in the *ATC Problems* view.

## Related Information

[Working with the ATC Problems View \[page 584\]](#)

### 5.2.2.2.3 Working with the ATC Problems View

Each ATC check run creates new ATC check results, which include a list of ATC findings. If a local ATC check run finds errors or warnings, the ABAP Development Tools (ADT) automatically shows these to you in the *Worklist* of the *ATC Problems* view.

The main purpose of the Worklist is to collect the results of all local ATC check runs that have been triggered in your IDE (during one and the same logon session) and keep the status of ATC findings up-to-date. This allows you, as a developer, to solve the ATC errors (or warnings) and immediately update the ATC status of the checked development objects – until the worklist is empty when all findings are fixed.

Here is how to work in this view and with ATC findings displayed:

- [Using the Filter \[page 585\]](#)
- [Configuring Columns and Grouping ATC Findings \[page 586\]](#)
- [Displaying Details of ATC Findings \[page 588\]](#)
- [Fixing ATC Findings \[page 590\]](#)
- [Applying Quick Fixes for ATC Findings in the ATC Problems View \[page 592\]](#)

- [Rechecking ATC Findings \[page 597\]](#)

## Related Information

[Selecting Check Variant \[page 597\]](#)

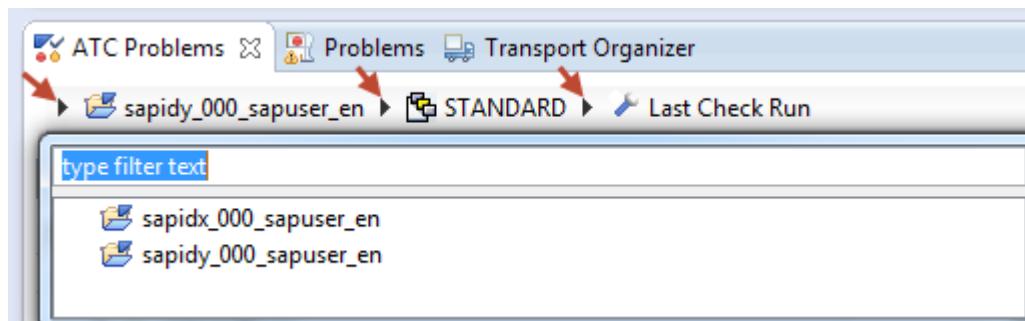
[Launching ATC Check Run from the Project Explorer \[page 581\]](#)

[Launching ATC Check Run from the ABAP Editor \[page 583\]](#)

### 5.2.2.2.3.1 Using the Filter

The ATC Problems view provides you with a filter to quickly switch between various views. In particular, you can switch...

- Between multiple ABAP projects from your workspace where local ATC check runs already have been performed during the IDE session
- For each project, between various check variants used for the ATC check run
- For each project and each check variant, between various object sets.



Using a filter for projects

## Related Information

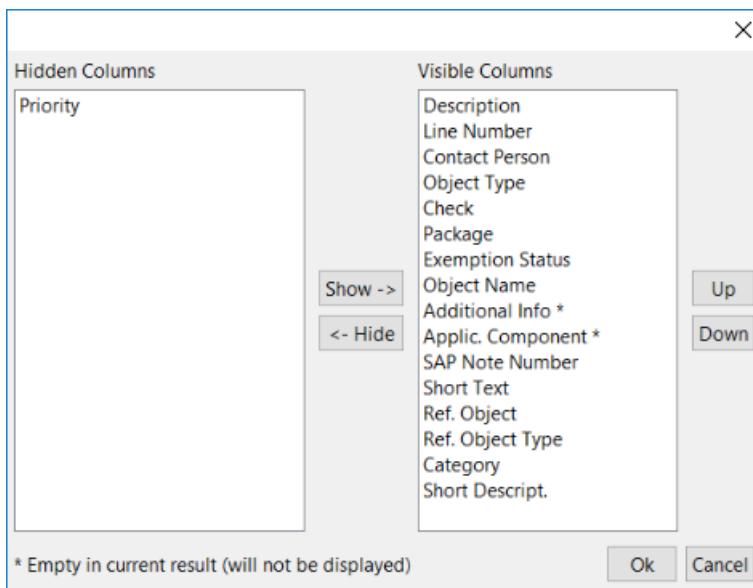
[Working with the ATC Problems View \[page 584\]](#)

## 5.2.2.2.3.2 Configuring Columns and Grouping ATC Findings

In the *ATC Problems* view, you can configure the columns of the view and group the ATC findings of an ATC result.

### Configuring Columns of the ATC Problems View

1. Choose *Configure Columns...* in the toolbar of the *ATC Problems* view.



Configuring columns of the ATC Problems view

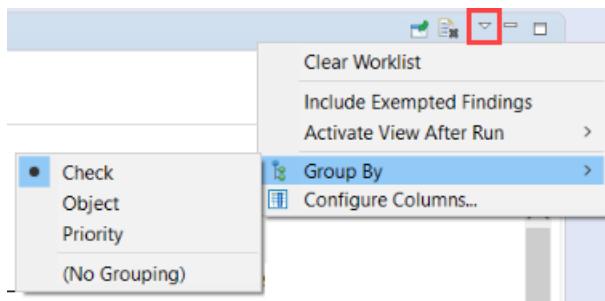
2. In the view that is opened, add or remove the columns you want to display or hide in the *Worklist* by selecting the name of the column in question and choosing ATC Problems *Show ->* or *<- Hide*.

#### i Note

Columns that are marked with an asterisk (\*) will not be displayed in the worklist, since they're empty for the current ATC check result.

### Grouping ATC Findings

Choose *Group By* in the toolbar of the *ATC Problems* view and select the option for which you want to group the ATC findings.



Grouping ATC findings in the worklist of the ATC Problems view

By default, the *ATC Problems* view groups your ATC findings by check.

**i Note**

For some check variants (for example the check variant S4HANA\_READINESS), the *ATC Problems* view displays more columns and grouping options to give more detailed information about the individual ATC findings.

## Related Information

[Working with the ATC Problems View \[page 584\]](#)

### 5.2.2.2.3.3 Changing the Contact Person of ATC Findings

If you want to assign ATC findings of an ATC run result to a certain developer, you can do that by changing the contact person of the ATC findings.

#### Prerequisites

You need the authorization object `S_Q_GOVERN` (`ACTVTT = 03` and `ATC_OTYPCGO = 02`) to change the contact person.

In addition, you need one of the following authorization objects:

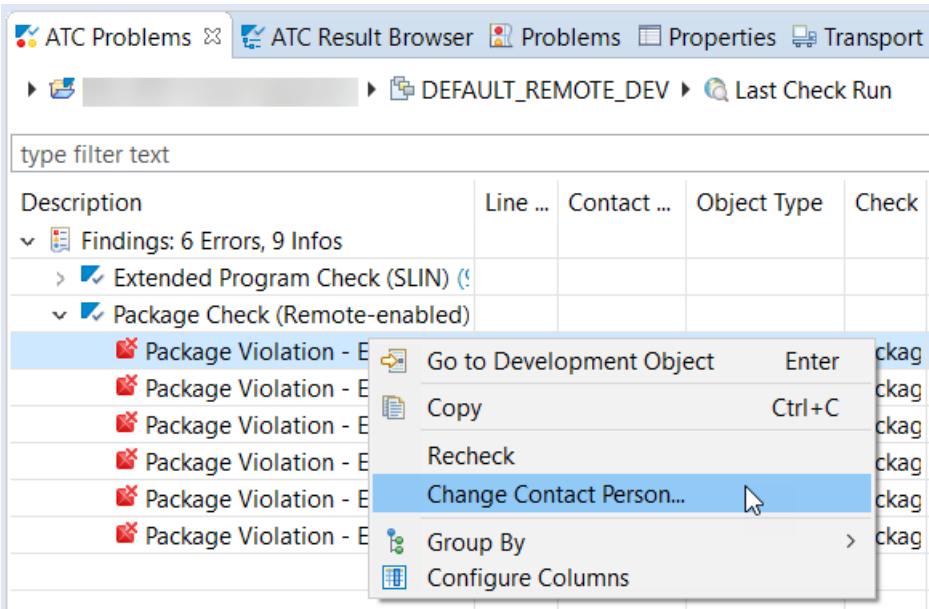
- `S_Q_GOVERN` (`ACTVTT = 01` and `ATC_OTYPCGO = 01`) or
- `S_DEVELOP` (`ACTVTT = 02`)

**i Note**

You can change the contact person only for local check runs.

## Procedure

1. In the *ATC Problems* view, select the ATC findings in question and choose *Change Contact Person...* in the context menu.



2. In the *Change Contact Person* view in the *New Contact Person* field, enter the user name of the developer to whom you want to assign the ATC findings.

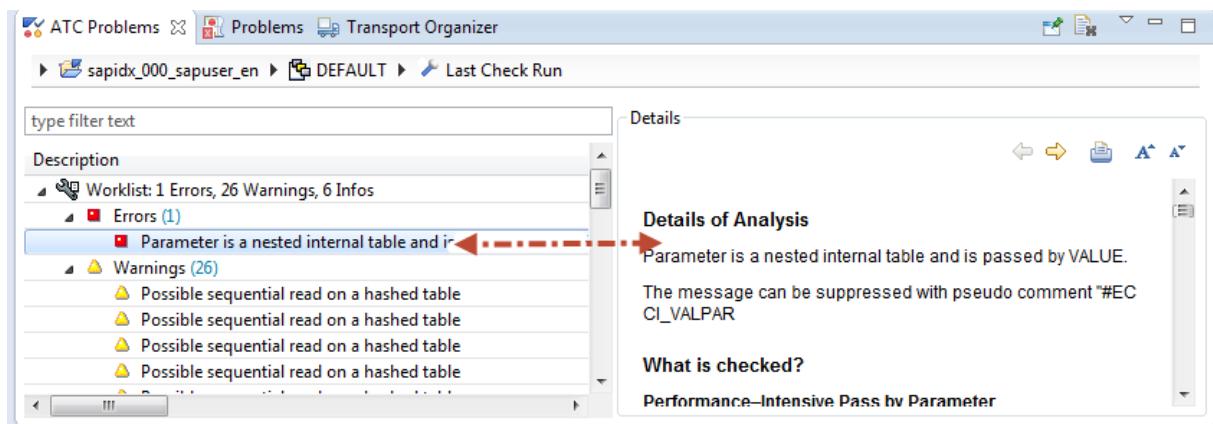
## Results

You have changed the contact person and the ATC findings are now assigned to the determined developer.

### 5.2.2.2.3.4 Displaying Details of ATC Findings

The short description of the ATC findings might not provide you with sufficient details about the problem. In the *ATC Problems* view, you can directly display the details of an ATC finding by selecting the ATC finding.

The *Details* are especially important for understanding ATC findings and fixing problems quickly. For example, you might see these details for an instance of the ATC finding 'Parameter is a nested internal table and passed by VALUE'.



The context-sensitive help for a selected ATC finding is automatically displayed in the ATC Problem view

#### i Note

The *Details* content automatically refreshes when you select a new finding in the *ATC Problems* view.

## Related Information

[Working with the ATC Problems View \[page 584\]](#)

### 5.2.2.2.3.5 Displaying Local Active Results in the ATC Problems View

## Prerequisites

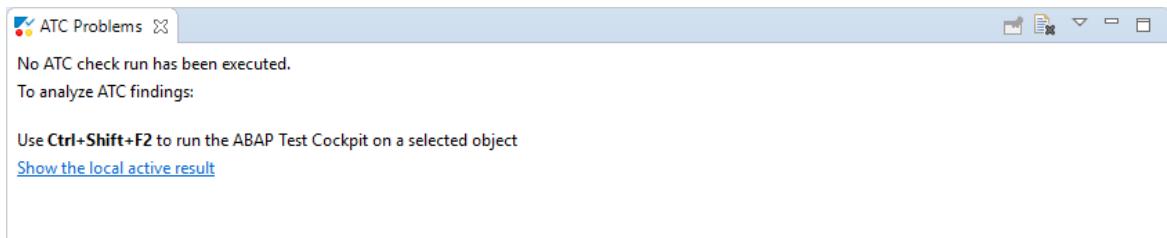
There is a local active result in the respective development system.

## Context

As a developer, you want to analyze and fix ATC findings that are assigned you.

## Procedure

1. On the onboarding page of the *ATC Problems* view, click the link as shown in the figure.

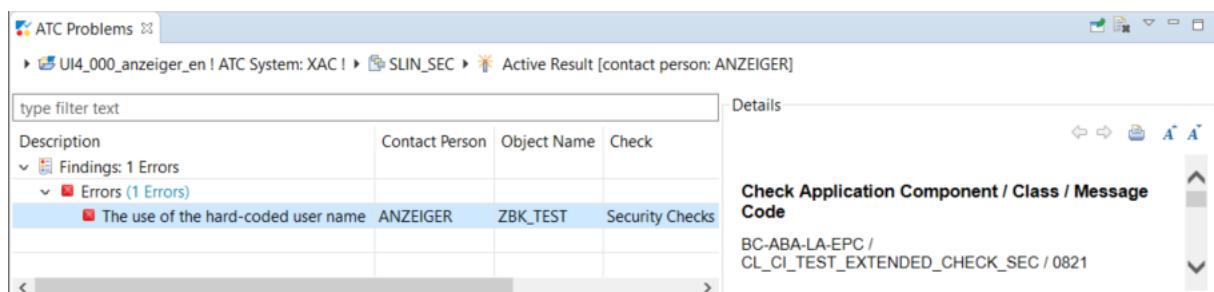


Onboarding page of ATC Problems view

2. In the dialog that opens, do one of the following:
  - select a project. Click *OK*
  - select *New* and follow the steps of the wizard to create a new project.

## Results

The ATC findings from local active results that are assigned to you are displayed in the *ATC Problems* view.



Local active results in ATC Problems view

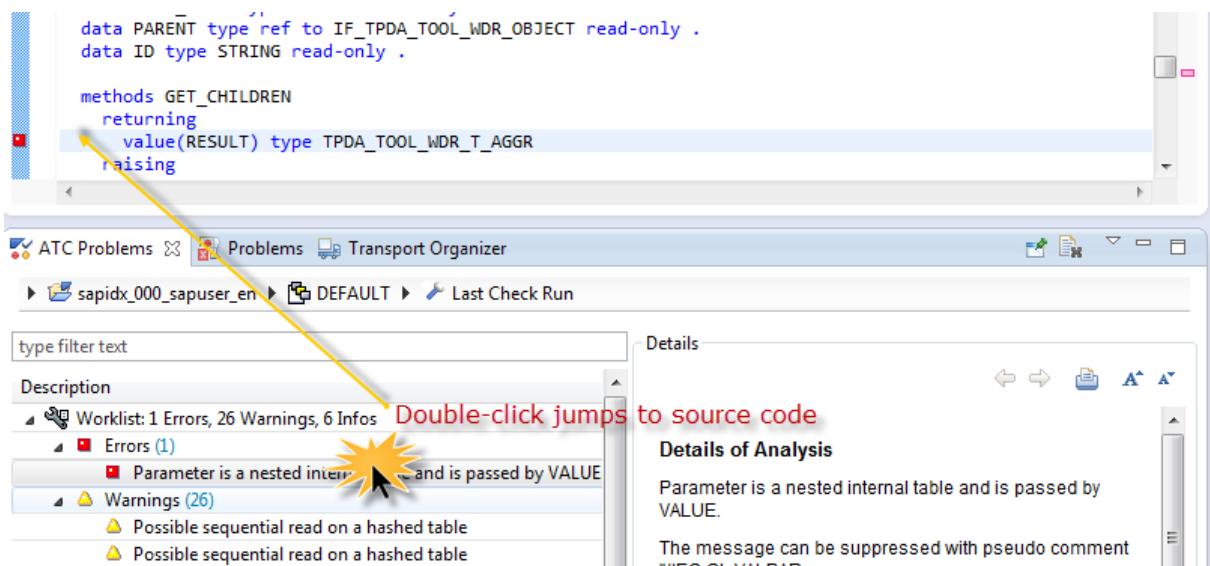
You can now fix ATC issues for example by [applying quick fixes \[page 592\]](#).

## Related Information

[Analyzing Local ATC Results from the ATC Result Browser \[page 607\]](#)

### 5.2.2.2.3.6 Fixing ATC Findings

If you are going to directly fix the ATC findings listed in the *Worklist* in the *ATC Problems* view, double-click an ATC finding to jump to your source code at the position where the ATC found a statement with a problem and adapt your source code.



Jumping from the worklist into your source code

If you want to locate the development object or object component that is responsible for an ATC finding, then:

- Open the *Outline* view to jump to the *Outline* at the development object or component in which the error was found.
- Choose the menu path **▶ Show In ▶ Project Explorer** in the editor to display in the *Project Explorer* view, the development object or object component in which the error was found.

#### i Note

You can also fix certain ATC findings with Quick Fixes. For more information, see [Applying Quick Fixes for ATC Findings \[page 591\]](#).

## Related Information

[Working with the ATC Problems View \[page 584\]](#)

### 5.2.2.2.3.7 Applying Quick Fixes for ATC Findings

You can fix certain ATC findings with Quick Fixes. These Quick Fixes provide functions that enable you to resolve errors and warnings without adapting your source code manually.

#### i Note

ATC findings that can be fixed with a Quick Fix are displayed with a lightbulb icon .

Quick Fixes for ATC findings can be applied in two different ways:

- Applying Quick Fixes for ATC Findings in the ATC Problems View [page 592]
  - Applying Quick Fixes for ATC Findings in the Source Code [page 593]

You can also fix several ATC findings at once with the *Recommended Quick Fixes* wizard. For more information, see [Applying Recommended Quick Fixes for Multiple ATC Findings \[page 594\]](#).

## Related Information

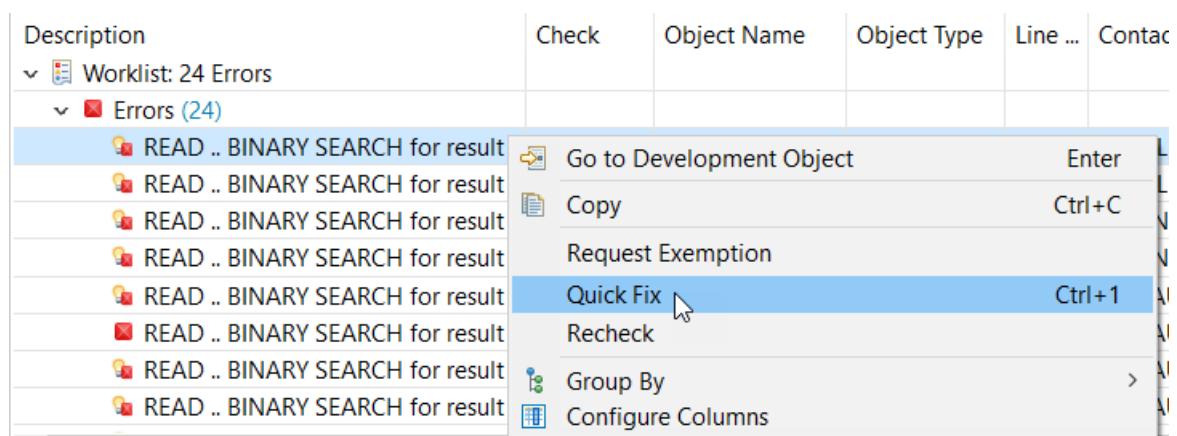
Working with the ATC Problems View [page 584]

### **5.2.2.2.3.7.1 Applying Quick Fixes for ATC Findings in the ATC Problems View**

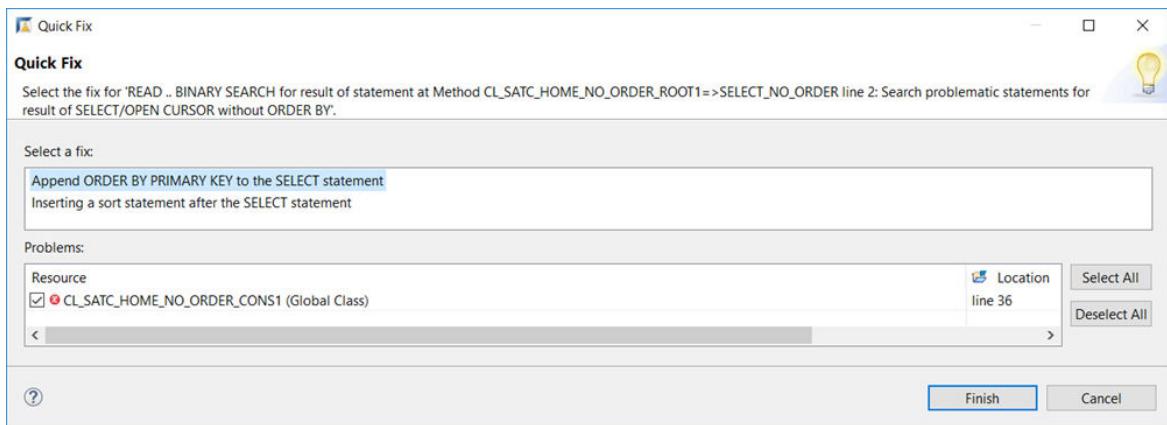
Quick Fixes for ATC findings can be applied in the [ATC Problems](#) view.

## Procedure

1. Select an ATC finding with a lightbulb icon in the *ATC Problems* view.
  2. Right-click the ATC finding and choose *Quick Fix*.



The Social Function of Art



3. Select the displayed Quick Fix and choose *Finish*.

#### → Recommendation

If there is more than one Quick Fix available for an ATC finding, we recommend that you select the first Quick Fix displayed.

## Related Information

[Applying Quick Fixes for ATC Findings \[page 591\]](#)

[Applying Quick Fixes for ATC Findings in the Source Code \[page 593\]](#)

## 5.2.2.2.3.7.2 Applying Quick Fixes for ATC Findings in the Source Code

Quick Fixes for ATC findings can be applied in the source code.

### Procedure

1. Double-click an ATC finding in the *ATC Problems* view.

The affected development object is opened in the source code editor.

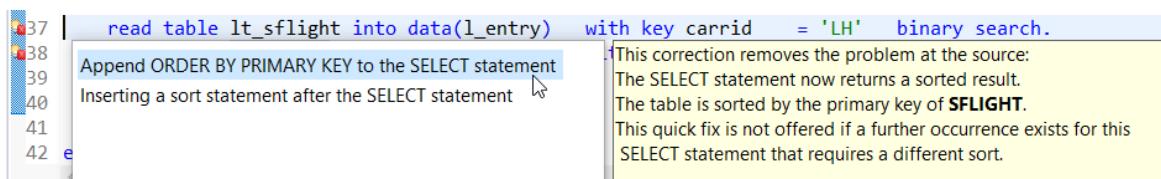
2. Place the cursor in a code line where a Quick Fix can be applied.

#### i Note

ATC findings that can be fixed with a Quick Fix are displayed with a lightbulb icon .

3. Open the context menu and choose *Quick Fix*.

The *Quick Fix Assist* view is opened.



#### Quick Fix Assist view

4. Double-click the Quick Fix to apply the Quick Fix to the affected code line.

#### → Recommendation

If there is more than one Quick Fix available for an ATC finding, we recommend that you select the first Quick Fix displayed.

## Related Information

[Applying Quick Fixes for ATC Findings \[page 591\]](#)

[Applying Quick Fixes for ATC Findings in the ATC Problems View \[page 592\]](#)

## 5.2.2.2.3.7.3 Applying Recommended Quick Fixes for Multiple ATC Findings

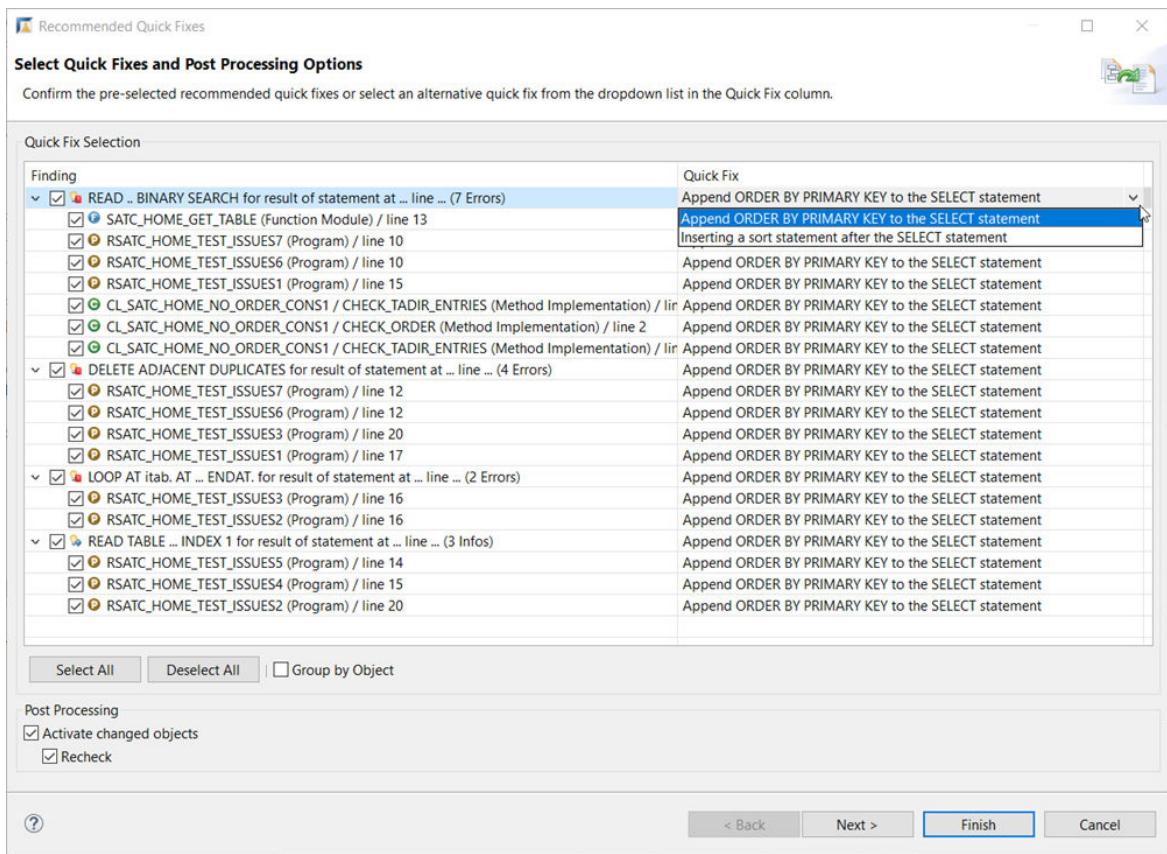
You can fix multiple ATC findings at once with the [Recommended Quick Fixes](#) wizard.

## Procedure

To fix multiple ATC findings at once, proceed as follows:

1. Select multiple ATC findings in the [ATC Problems](#) view.
2. Open the context menu and choose [Recommended Quick Fixes....](#)

The [Recommended Quick Fixes](#) wizard is opened.



#### Recommended Quick Fixes wizard

3. In the *Finding* column in the *Quick Fix Selection* frame, select the ATC findings and the affected objects. By default, all ATC findings and their affected objects are selected.

#### → Tip

If you want to display all affected objects and their respective ATC findings, choose *Group by Object*.

4. In the *Quick Fix* column in the *Quick Fix Selection* frame, the recommended Quick Fixes for the ATC findings are displayed by default. Select a Quick Fix to open a dropdown list with alternative Quick Fixes.

#### → Recommendation

We recommend applying the Quick Fixes displayed by default.

5. In the *Post Processing* frame, you can specify that the changed objects are activated after you apply the Quick Fixes. If this option is selected, you can specify that the selected ATC findings are rechecked after you finish the wizard.

#### i Note

If you do not select any post processing options, the initial ATC result is displayed after you finish the wizard. In this case, you have to activate and recheck the ATC findings manually.

6. Choose *Next*.
7. Select a transport request if required.
8. Choose *Next*.

- Review the changes. Here, a comparison editor is displayed where you can review the refactored source code. The code line where the source code has been refactored is highlighted.

The screenshot shows the SAP Recommended Quick Fixes 'Review Changes' wizard. The 'Changes to be performed' section lists two objects: 'CL\_SATC\_HOME\_NO\_ORDER\_ROOT1 (Global Class)' and 'SATC\_HOME\_GET\_TABLE (Function Module)'. The main area displays a comparison editor for the 'UIA\_SSO' object. The 'Original Source' tab shows the initial ABAP code, and the 'Refactored Source' tab shows the code with a specific line highlighted in blue: '24 adir into table result where object = 'RRST' ORDER BY PRIMARY KEY.'. This indicates that a fix has been applied to ensure the query is ordered by the primary key.

Review changes with the compare editor in the Recommended Quick Fixes wizard

In this example, an `order by primary key` statement was added to the source code to fix the ATC finding.

- Choose *Finish*.

## Results

You applied Quick Fixes for multiple ATC findings at once.

### i Note

Sometimes it can happen, that some ATC findings could not be fixed with recommended Quick Fixes after finishing the wizard. Open the [ABAP Log](#) view to see detailed information on these ATC findings.

### 5.2.2.2.3.8 Rechecking ATC Findings

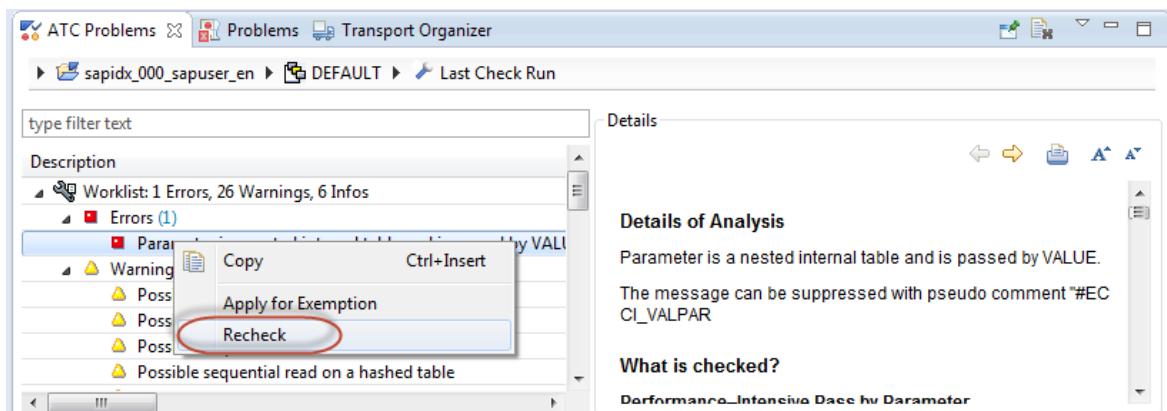
After fixing the ATC finding in your source code, the currently displayed ATC result is no longer valid. You can then run a recheck for the corrected ATC findings in order to check if they have disappeared from the worklist.

#### Context

To run a recheck for one or multiple findings (within the entire worklist), proceed as follows:

#### Procedure

1. (Multi-)Select the corrected ATC finding(s) in the *ATC Problems* view.
2. Choose *Recheck* from the context menu.



Running a Recheck for a corrected ATC finding

#### Related Information

[Working with the ATC Problems View \[page 584\]](#)

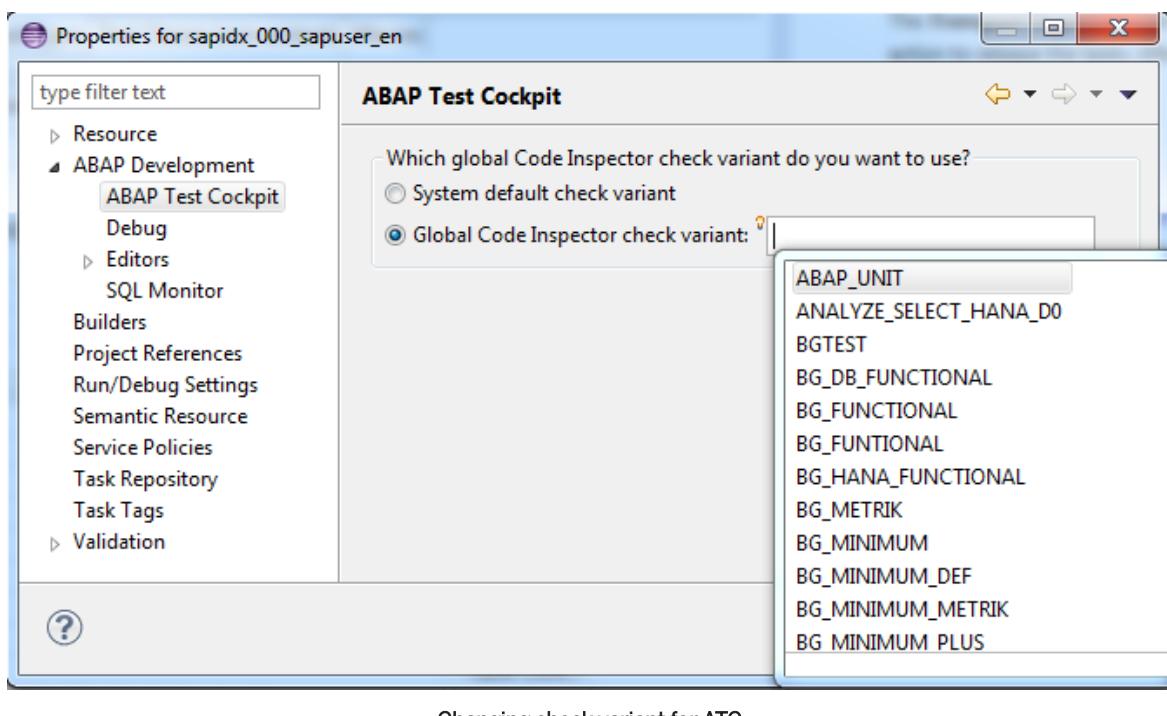
### 5.2.2.2.4 Selecting Check Variant

#### Context

Each ATC check run uses a global Code Inspector variant that contains a set of relevant checks. For each system, a default check variant is already predefined. In some cases, you may want to change the global Code Inspector variant that is used for ATC check runs in your development system.

## Procedure

1. In the *Project Explorer*, select the relevant ABAP project (that represents your ABAP development system).
2. Open the context menu and choose *Properties*.
3. In the *Properties* dialog, select the entry *ABAP Test Cockpit*.
4. Now you can enter the new *Global Code Inspector check variant*.



Changing check variant for ATC

→ Tip

When editing the name of the check variant, you can avail of the content assistant functionality by pressing *Ctrl + Space* in the corresponding field.

### 5.2.2.2.5 Check Variant Editor

The check variant editor enables you to display parameters and available documentation for ABAP Test Cockpit check variants.

i Note

The check variant editor is **only** available in display mode.

## Related Information

[Displaying Check Variants \[page 599\]](#)

### 5.2.2.2.5.1 Displaying Check Variants

#### Context

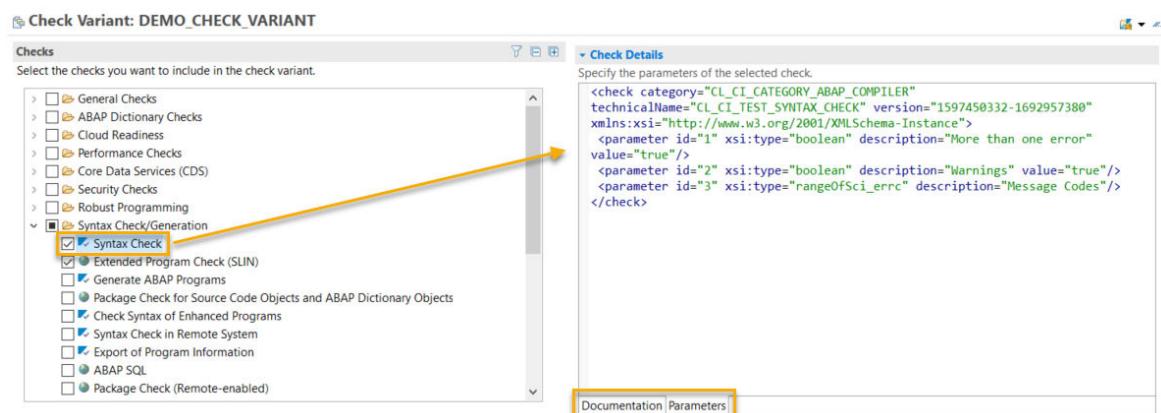
You can display parameters and available documentation for ABAP Test Cockpit check variants with the check variant editor in ABAP Development Tools (ADT).

##### i Note

The check variant editor is **only** available in display mode.

#### Procedure

1. In ADT, press Ctrl+Shift+A to open the search dialog for ABAP Development Objects.
  2. To search for all available check variants in your system, enter `type:chkv` as search string.
  3. Select the check variant you want to display, then choose *OK*.
- The check variant editor is opened.
4. Choose an entry from one of the folders in the list to display its parameters on the *Parameter* tab in the detail window.
  5. Choose the *Documentation* tab in the detail window to display available documentation for a check entry.



## 5.2.2.3 Working with ATC During Transport Release

### Use Case

(1) The developer runs the ATC check explicitly in the *Transport Organizer* tool before releasing a transport request or task. - **Recommended Step!**

(2) The developer releases a transport request or task and the ATC check is launched automatically. The automatic transport check provides a first **Q Gate** when the code leaves the development system at the time of transport release. ATC findings may stop the transport release.

### Activities Relevant for Developers

- [Launching ATC Check Run Explicitly \[page 600\]](#)
- [Launching ATC Check Implicitly \[page 601\]](#)
- [Working with the ATC Problems View \[page 584\]](#)

### Related Information

[Working with Transport Organizer \[page 437\]](#)

## 5.2.2.3.1 Launching ATC Check Run Explicitly

### Context

A transport request (or transport task) that contains changes for several development objects is about to be released. However, before releasing the transport, you want to check if the transport still has errors or warnings.

### Procedure

1. If you have not yet already done so, open the *Transport Organizer* view using the tool bar menu  [Window](#)  [Show View](#)  [Other...](#)
2. In the *Transport Organizer* view, expand the *Workbench* folder.

3. Expand the *Modifiable* folder until you have found the requested transport request or task.
4. Choose  from the context menu of the request or task or alternatively use the key shortcut `Ctrl + Shift + F2`.

#### Note

Alternatively, you can start an ATC check run with the  to specify an arbitrary global check variant. You can use this option to override the predefined ATC check variant for that specific check run. **See also:** [Launching ATC Check Run from the Project Explorer \[page 581\]](#)

## Results

The ATC checks all of the repository objects contained in the selected transport request or task. After completion of the ATC check run, the new ATC check result is displayed as a *Worklist* in the *ATC Problems* view.

## Related Information

[Working with Transport Organizer \[page 437\]](#)  
[Working with the ATC Result Browser \[page 605\]](#)

### 5.2.2.3.2 Launching ATC Check Implicitly

## Context

You, as an ABAP developer, are going to release a transport request that contains changes for several development objects. When you release a transport, the ABAP backend system automatically launches the ATC check run in the background to check all development objects in the transport request.

#### Recommendation

We recommend running an ATC check explicitly before releasing a transport request. For more information, see [Launching ATC Check Run Explicitly \[page 600\]](#).

To release a transport request in ADT, proceed as follows:

## Procedure

1. If you have not yet already done so, open the *Transport Organizer* view using the tool bar menu  *Window*  *Show View*  *Other...* 
2. In the *Transport Organizer* view, expand the *Workbench* folder.
3. Expand the *Modifiable* folder until you have found the requested transport request or task.
4. Choose *Release* from the context menu of the request.

### Note

If ATC findings are still reported in the transport request, a popup dialog is displayed. Choose *Display ATC Findings* to display the ATC findings in the *ATC Problems* view.

## Related Information

[Working with Transport Organizer \[page 437\]](#)

### 5.2.2.4 Working with Central ATC Results

## Use Case

The transports with code changes from the development systems reach the quality system during the consolidation phase. The quality manager runs mass regression checks with ATC in the quality system and then distributes the ATC results to the corresponding development systems as active results. This ensures that code changes do not leave a consolidation or quality system until all critical findings have been cleaned up (second **Q Gate**).

In ABAP Development Tools, the developers typically get a feed notification informing them about new central ATC results. Developers can analyze these central ATC results in the ATC Result Browser and immediately clean up the code in the development system.

## Activities Relevant for Developers

- [Opening the ATC Result Browser \[page 605\]](#)

- [Working with the ATC Result Browser \[page 605\]](#)
- [Optional] [Requesting Exemptions for ATC Findings \[page 611\]](#)

## Related Information

[ATC Quality Checking \[page 72\]](#)

### 5.2.2.4.1 Getting Feeds with Active Results

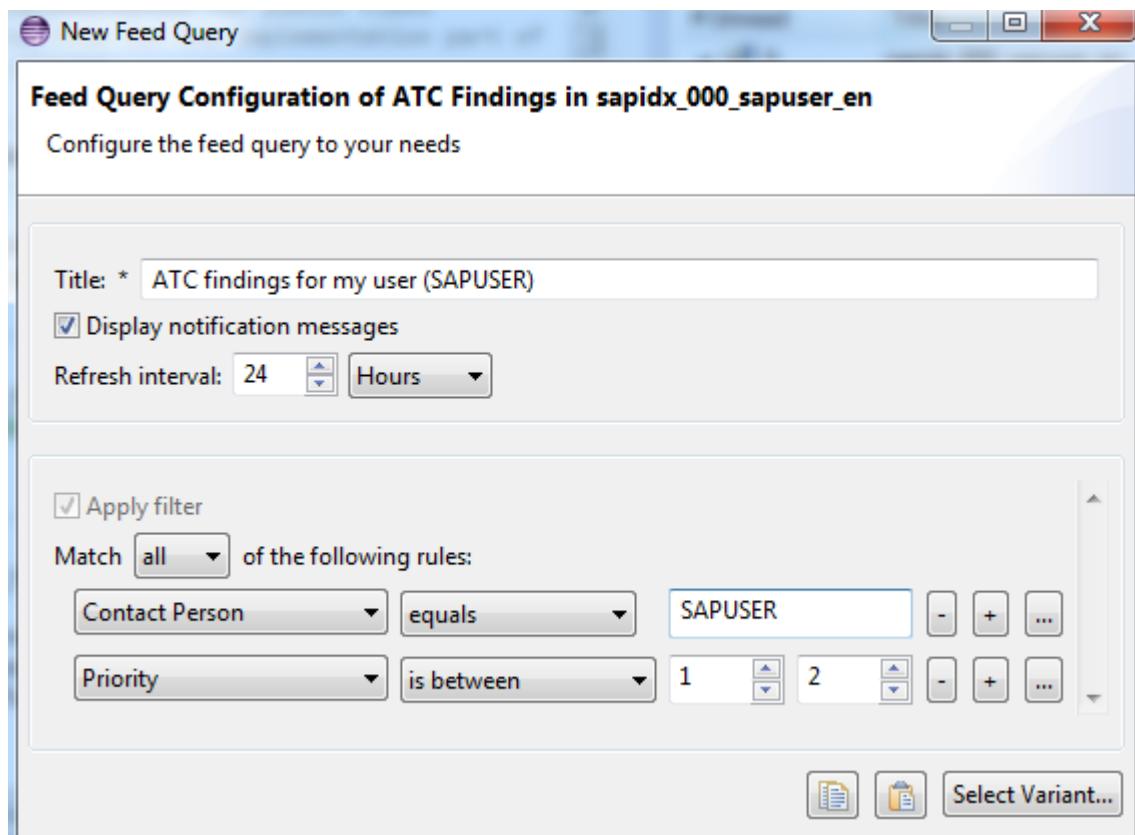
#### Prerequisites

- To get notice of your own active ATC findings, you must first subscribe to feeds from the ABAP Repository. In particular, when adding a new query, you must select *ATC Finding* as a feed query.

**See also:** [Subscribing to Feeds from ABAP Repository \[page 488\]](#)

→ Tip

You can configure the findings displayed in the active ATC result according to various filter criteria such as, for example, refresh interval, priority, or contact person.



Configuring the feed query for ATC findings

## Context

If your organization is using the ATC to run central quality checks, then you can receive notification of your own priority 1 and 2 ATC findings in the active ATC central result (entitled "ATC Messages --Prio 1 + 2 -- of Logon User in your ABAP Projects"). For this purpose, ADT offers an ATC feed that provides you with access to findings in the active ATC result.

Once you have the ATC feed, you can perform the following with the messages in the feed:

## Procedure

1. Display details of the ATC finding by clicking on the finding.
2. Jump to the location of a problem in your source code by double-clicking a finding.

## Related Information

[Feed Reader View \[page 80\]](#)

[ATC Quality Checking \[page 72\]](#)

## 5.2.2.4.2 Opening the ATC Result Browser

### Context

In ABAP Development Tools (ADT), the developer receives a notification with new incoming results from ATC quality checks originating from a central quality system. [More on this: Getting Feeds with Active Results \[page 603\]](#) You, as a developer, can use the *ATC Result Browser* to look at the current active results and also to get an overview of your local ATC check runs.

#### i Note

By default, the *ATC Result Browser* is not included in the *ABAP perspective*.

To add the *ATC Result Browser* view, proceed as follows:

### Procedure

1. Choose the tool bar menu  *Window*  *Show View*  *Other...* 
2. In the *ABAP* folder, select *ATC Result Browser*.

### Results

ADT opens the *ATC Result Browser* that displays, for each ABAP project (development system), the active result replicated from the central quality system and also provides an overview about local ATC check runs.

### Related Information

[Working with the ATC Result Browser \[page 605\]](#)

## 5.2.2.4.3 Working with the ATC Result Browser

Here is how to work with ATC Results and ATC Findings in the *ATC Result Browser* view:

- [Browsing ATC Results \[page 606\]](#)
- [Navigating Within the ATC Result Browser \[page 608\]](#)
- [Rechecking an ATC Result \[page 610\]](#)
- [Requesting Exemptions for ATC Findings \[page 611\]](#)

## 5.2.2.4.3.1 Browsing ATC Results

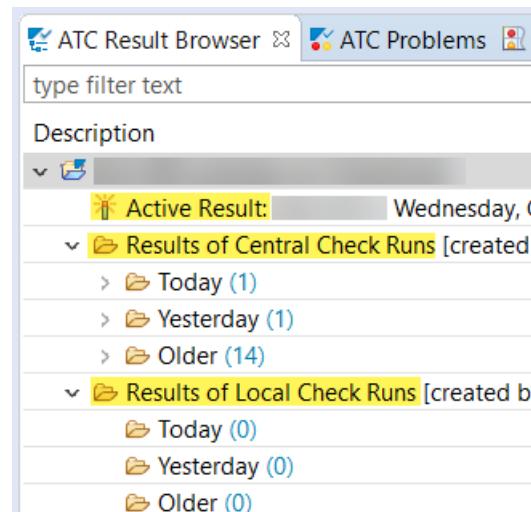
The *ATC Result Browser* lets you browse through ATC results for each ABAP Project (representing a connection to an ABAP system) created in your ADT workspace.

### Context

On the top level, the view displays the *Active Result*. In addition, the view displays *Results of Central Check Runs* and *Results of Local Check Runs*. You can also display ATC check results from other users.

#### → Remember

The ATC findings in the active result may be outdated when the code in the development system has been corrected in the meantime.

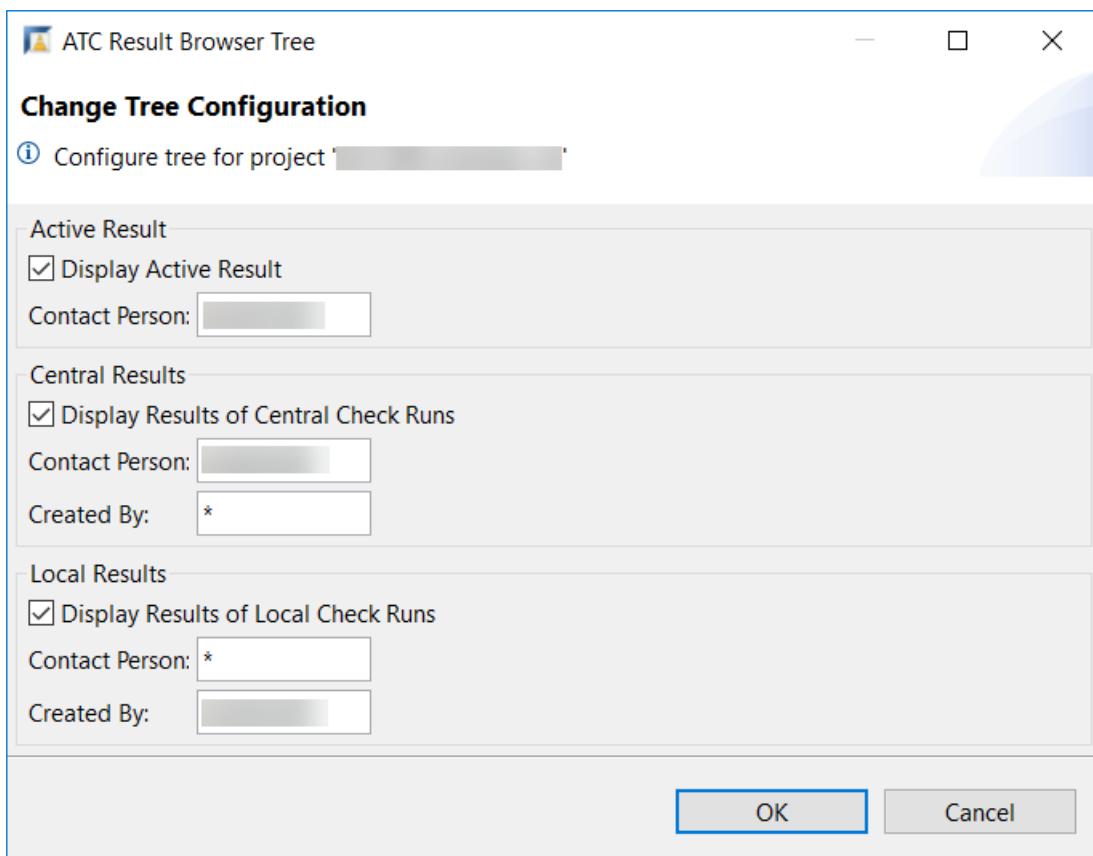


The ATC Result Browser displays the Active Result, results of central check runs, and results of local check runs

To browse ATC results from other users, proceed as follows:

### Procedure

1. Select the ABAP Project in the *ATC Result Browser* and choose *Configure Tree...* in the context menu.
2. In the *ATC Result Browser Tree* view, you can configure the display of the *Active Result*, of *Central Results*, and of *Local Results*. You can also enter the user name of whom you want to display the ATC results.



Changing the ATC Result Browser tree configuration

### 5.2.2.4.3.2 Analyzing Local ATC Results from the ATC Result Browser

#### Prerequisites

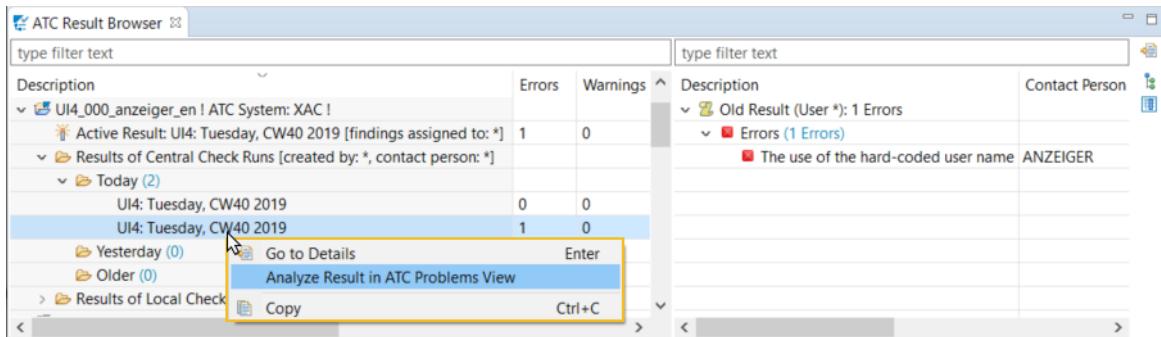
There is a local ATC Result in the respective development system.

#### Context

As a developer, you want to fix ATC issues that are displayed in the *ATC Result Browser* for example by applying a quick fix.

## Procedure

1. In the *ATC Result Browser*, select a result you want to analyze.
2. From the context menu, select *Analyze Result in ATC Problems View*.



## Results

The filtered result is opened in the *ATC Problems* view. Now, you can fix the issues, for example, by [applying a quick fix](#) [page 592].

## Related Information

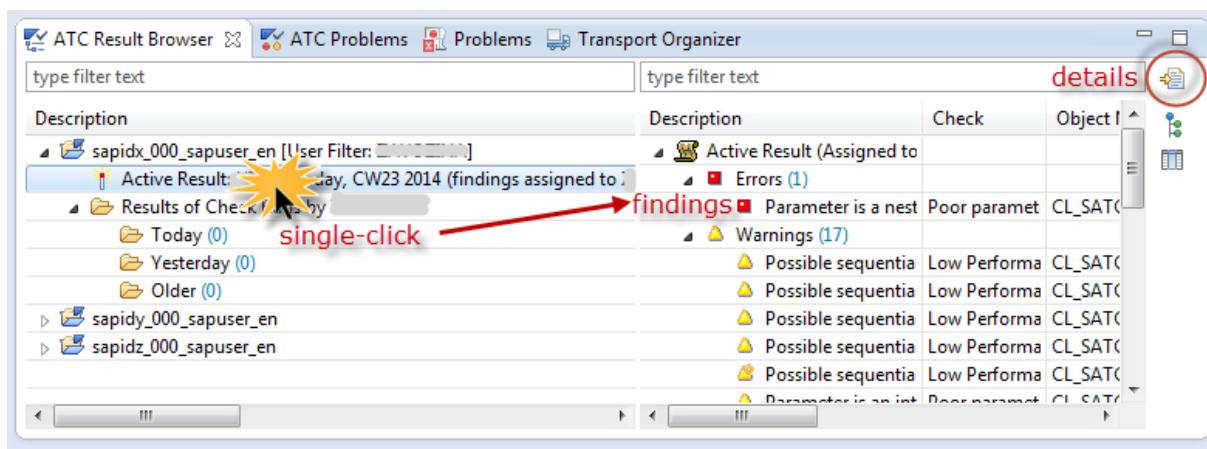
[Displaying Local Active Results in the ATC Problems View](#) [page 589]

### 5.2.2.4.3.3 Navigating Within the ATC Result Browser

In the *ATC Result Browser* view, you can also navigate from an ATC result to its ATC findings, display the details of an ATC finding, and jump from ATC findings to the effected source code.

#### Navigating From ATC Result to its ATC Findings

Select an ATC result in the *ATC Result Browser* view to display the list of ATC findings that belong to the corresponding ATC result.



Navigating from ATC result to its ATC findings

## Jumping from an ATC Finding to the Effected Source Code

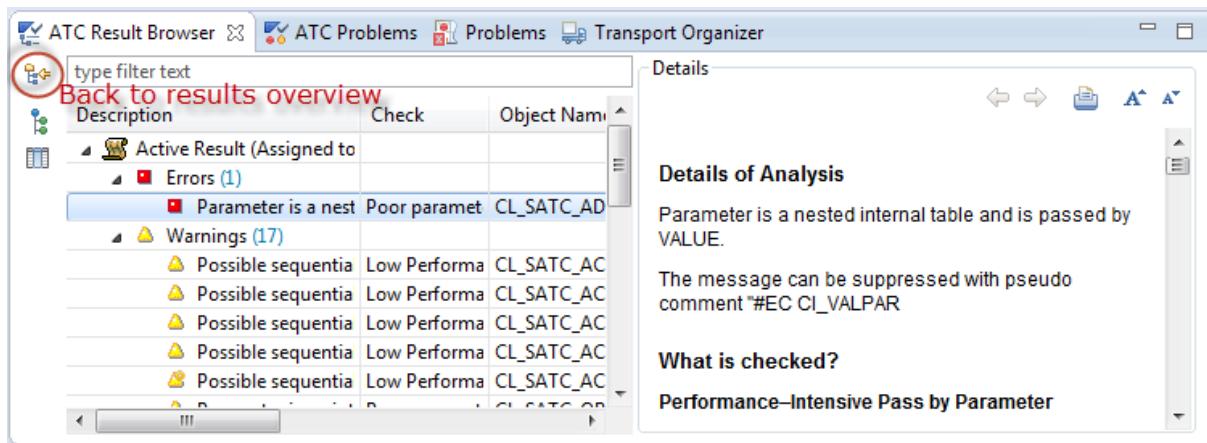
Double-click an ATC finding to jump to the effected source code at the position where the ATC found a problem.

## Displaying Details of an ATC Finding

Select an ATC finding and choose *Go to Details* in the context menu or choose the corresponding icon in the toolbar.

## Navigating Back to the ATC Results Overview

Select an ATC finding and choose *Back to Results Overview* in the context menu or click the corresponding icon in the toolbar.



Navigating back to the ATC results overview

## 5.2.2.4.3.4 Rechecking an ATC Result

If you, as a developer, want to know if the ATC findings from the replicated active result or the central or local ATC check runs are available in the relevant development system, you can recheck the ATC result in the [ATC Result Browser](#) view.

### Context

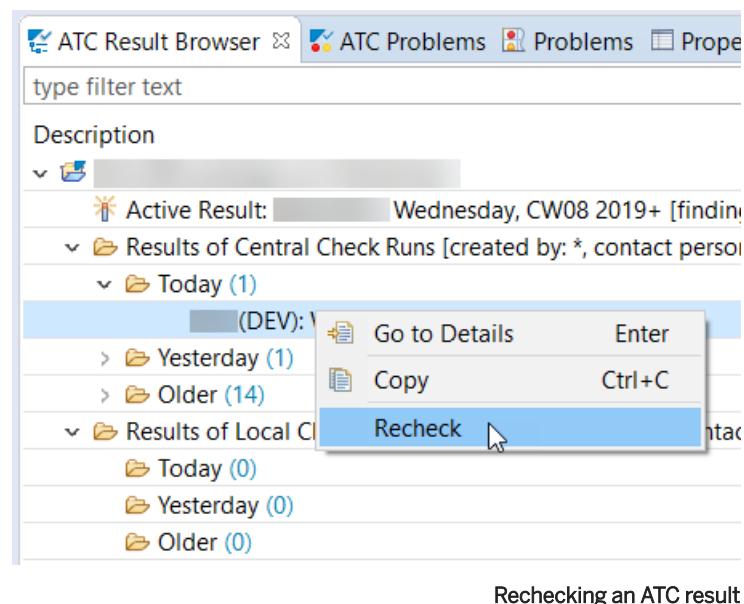
To recheck an ATC result in your development system, proceed as follows:

### Procedure

Select an ATC result and choose [Recheck](#) from the context menu.

#### → Remember

When running a recheck, ATC uses a Code Inspector variant that is predefined in your local system.



## 5.2.2.4.3.5 Requesting Exemptions for ATC Findings

Sometimes ATC findings cannot be corrected. It is possible that such a problem is planned for correction in an upcoming development cycle. Or perhaps only a test program is affected.

### Context

If you can't clear an ATC finding by correcting the underlying problem, you can still clear it by requesting an exemption. Once approved by a quality manager, an exemption masks an ATC error or warning message. The finding then does not appear as an open issue in the new ATC results anymore.

To request exemptions for ATC findings from within ABAP Development Tools (ADT), proceed as follows:

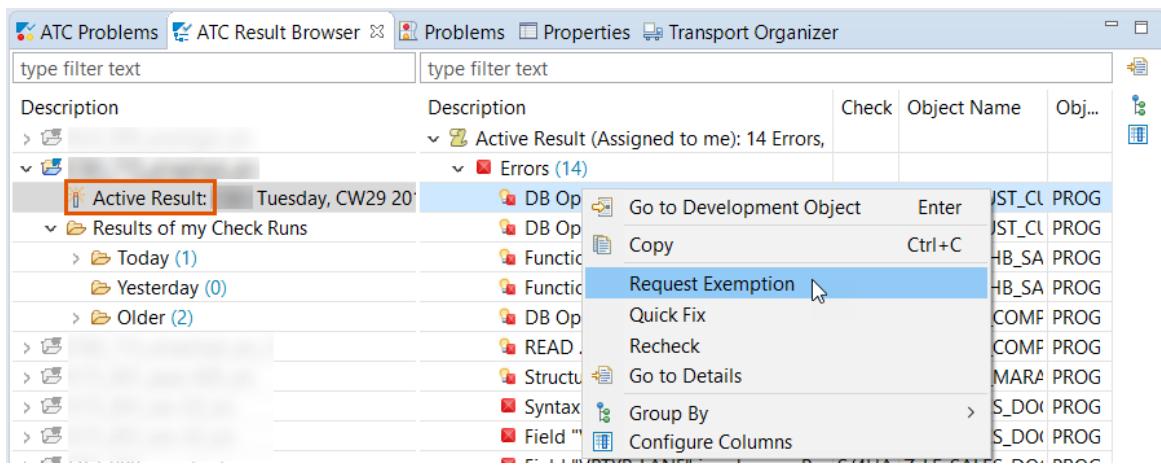
### Procedure

1. In the *ATC Result Browser* view, select the relevant ATC finding.

#### i Note

Be sure to request exemptions only for ATC findings from the *Active Result* - marked with  icon - in the *ATC Result Browser*. Only exemptions that you request against the active result are considered in future central ATC runs. It is not sensible to request exemptions for a local result that you generated yourself in your development environment.

2. Choose *Request Exemption* in the context menu of the ATC finding.



Requesting an Exemption in the context menu of an ATC finding

ADT starts the corresponding wizard.

3. In the first wizard, specify the granularity and the scope of the exemption.

### i Note

The minimum granularity and scope of an exemption restricts the ATC check to all instances of a particular check message in a single subobject, such as an ABAP include or a method (in a class).

4. Choose *Next*.
5. In the second wizard page, enter at least an approver, and specify the reason for the exemption.
6. Choose *Finish*.

## Results

The approver receives an e-mail notification with the exemption request. If the exemption is approved by the quality manager, the ATC finding is marked as inactive, and the ATC generates no new findings for the underlying ATC problem.

## Related Information

[Working with the ATC Result Browser \[page 605\]](#)

[Getting Feeds with Active Results \[page 603\]](#)

## 5.3 Using Troubleshooting Tools

### Context

The ABAP Debugger lets you stop a program during runtime and examine the flow and results of each statement during execution. Stepping through a program with the debugger helps you to detect and correct errors in ABAP source code.

The ABAP Profiler tools show you where runtime is being consumed, and where effort for refactoring and optimization can best be applied. They also let you analyze and understand program flow, which is useful when you are trying to understand a problem or learn about code you need to analyze or maintain.

If an ABAP program encounters a runtime error, the program writes an ABAP short dump to document the error and help in its analysis. An **ABAP short dump** provides a great deal of well-structured information for localizing and understanding a runtime error, including a textual description of the error and its likely cause, an excerpt showing the location of the error in the source code, tables of variables and their values, and more.

Dynamic logpoints have been introduced in order to support logging in quality or productive systems where the source code cannot be changed. For example, you have found a location in the source code for which you need additional monitoring details, or you may be facing a poorly localizable error situation that happens from time to time in a batch job and cannot be debugged.

The AMDP Debugger allows you to debug the embedded AMDP code within ABAP Development Tools (ADT).

## Related Information

[Debugging ABAP Code \[page 613\]](#)

[Profiling ABAP Code \[page 640\]](#)

[Analyzing ABAP Runtime Errors \[page 710\]](#)

### 5.3.1 Debugging ABAP Code

#### Context

The ABAP Debugger can satisfy most of your debugging requirements. Start the debugger by setting a breakpoint in your ABAP source code and then running the ABAP program.

The debugger offers, among other features

- [Setting and Managing Breakpoints \[page 614\]](#) in the debugger.
- [Breakpoints in the Debugger - Characteristics \[page 75\]](#), so that you can capture external calls via HTTP or RFC into the ABAP back end system in the ADT debugger
- [Using Watchpoints \[page 618\]](#)
- [Managing Variables \[page 622\]](#)
- [Debugging Dynpro Flow Logic \[page 631\]](#)
- [Debugging Enhancement Implementation \[page 633\]](#)
- [Displaying Exceptions \[page 630\]](#)
- [Navigating in the Debugger \[page 635\]](#)
- Direct editing in the debugger, so that you can correct a mistake right when you find it, as well as many other comfort and efficiency features

It's an ABAP debugger, but the look, feel, and functionality are all Eclipse, with all of the comfort and feature-richness that one associates with Eclipse. For a quick orientation to the Debug perspective, follow this [ABAP Debugger \[page 73\]](#).

## Related Information

[Debugging Modes \[page 76\]](#)

[Setting ABAP Debugging Preferences \[page 637\]](#)

[Setting ABAP Project-Specific Debug Settings \[page 638\]](#)

## 5.3.1.1 Setting and Managing Breakpoints

### Context

Here is how to set and manage breakpoints:

- [Setting Breakpoints at a Line in Code \[page 614\]](#)
- [Setting Dynamic Breakpoints \[page 615\]](#)
- [Adding Conditions to Breakpoints \[page 616\]](#)

### Related Information

[Setting ABAP Project-Specific Debug Settings \[page 638\]](#)

[Breakpoints in the Debugger - Characteristics \[page 75\]](#)

### 5.3.1.1.1 Setting Breakpoints at a Line in Code

#### Procedure

##### Setting Breakpoint from the Ruler in the Editor Using the Context Menu

1. Position the cursor within the ruler (left bar) of the source editor at the line that contains the executable ABAP statement of your interest.
2. Choose *Toggle Breakpoint* from the context menu.

##### Setting Breakpoint from the Ruler in the Editor Using Double-Click .

1. Double-click within the ruler of the source editor at the line that contains the executable ABAP statement of your interest.

##### Setting Breakpoint At Lines of Code

1. Position the cursor on the ABAP statement at which you want to stop.
2. Then choose the menu option  *Run*  *ABAP Breakpoints*  *Toggle Breakpoint* .

### Results

A breakpoint stays with the line of code at which you set it. If you change the code above the breakpoint, it slides along with the relocated code.

#### Further Activities

Toggle a breakpoint again to delete it. You can also deactivate breakpoints, without deleting them.

You can display and manage breakpoints in the *Breakpoints* view in the *Debug* perspective.

## Related Information

[Adding Conditions to Breakpoints \[page 616\]](#)

[Breakpoints in the Debugger - Characteristics \[page 75\]](#)

[Setting Dynamic Breakpoints \[page 615\]](#)

### 5.3.1.1.2 Setting Dynamic Breakpoints

#### Context

You can set dynamic breakpoints. These stop execution whenever a specified token – an ABAP statement or an exception class – is reached.

#### Procedure

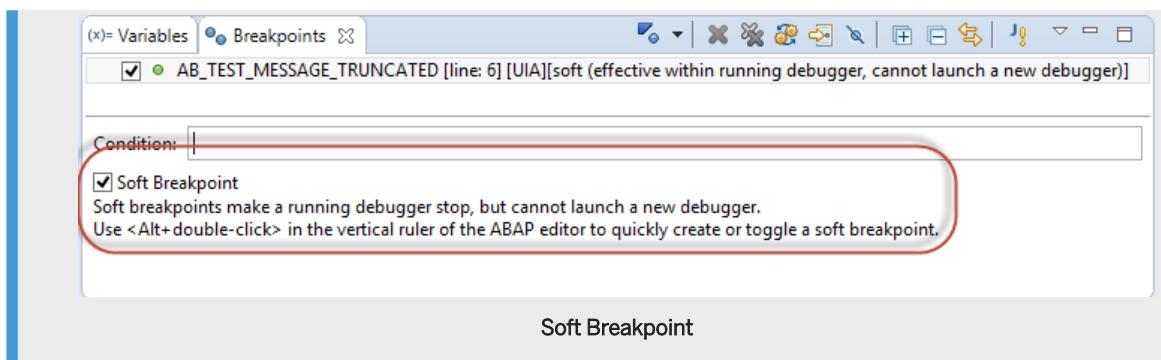
1. From the menu, choose **Run > ABAP Breakpoints > Add Statement Breakpoint** or **Add Exception Breakpoint**.
2. If the ABAP project that you are working in is not selected, then choose the project from the dialog that appears.
3. Choose from the list of ABAP statements. Or enter an ABAP exception class. The function searches incrementally for exception classes as you type.

#### i Note

- **Alternative way to set a dynamic breakpoint:** Open the *Debug* perspective. From the tool bar of the *Breakpoints* view, choose *Add ABAP Breakpoint*.



- **Breakpoint scope:** For dynamic breakpoints we recommend keeping the default property *Soft breakpoints*. Thus only running debug sessions if they have been started for example, using a standard line-breakpoint, will consider this dynamic breakpoints. This allows you to focus on the relevant code area and avoids unwanted debug sessions.



## Results

You have set a dynamic breakpoint that is active for any program that you run from within the current ABAP project.

You can display and manage dynamic breakpoints from the *Breakpoint* view in the *Debug* perspective. In particular, you can change the token that triggers the breakpoint.

## Related Information

[Adding Conditions to Breakpoints \[page 616\]](#)

[Setting ABAP Project-Specific Debug Settings \[page 638\]](#)

[Breakpoints in the Debugger - Characteristics \[page 75\]](#)

### 5.3.1.1.3 Adding Conditions to Breakpoints

#### Context

You have the option to specify a condition for an ABAP breakpoint. The condition is evaluated at runtime whenever the source code position of the related breakpoint is reached. If the condition is fulfilled (evaluates to true), the debugger will stop execution at the breakpoint. Otherwise, if the condition is not fulfilled, the debugger will not stop at the breakpoint.

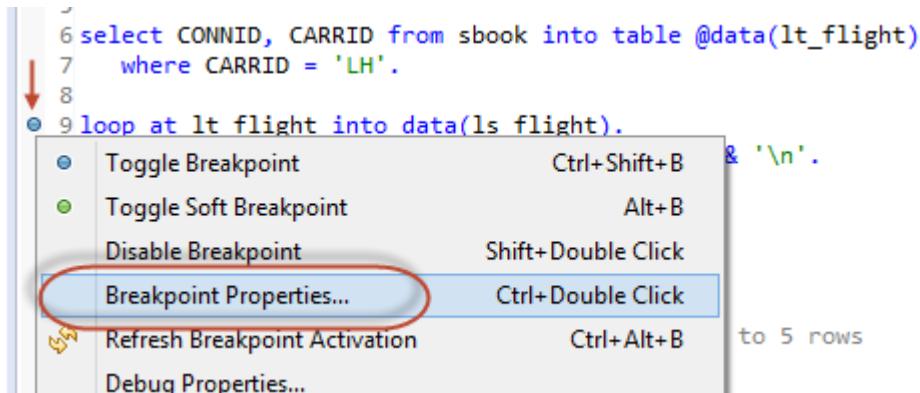
#### i Note

An empty condition always evaluates to true.

## Procedure

To add a condition to an existing ABAP breakpoint, proceed as follows:

1. Position the cursor in the ruler (left bar) of the source editor at the breakpoint position.
2. Open the context menu and choose *Breakpoint Properties...*



Using properties dialog to add condition to a breakpoint

### → Tip

Alternatively, you can specify a breakpoint condition in the Breakpoints view of the Debug perspective.

3. In the screen that appears, edit a condition in the corresponding entry field:

More on this: [Syntax for Breakpoint Conditions \[page 800\]](#)

### i Note

The maximum length of a breakpoint condition is 255 characters.



Editing breakpoint condition

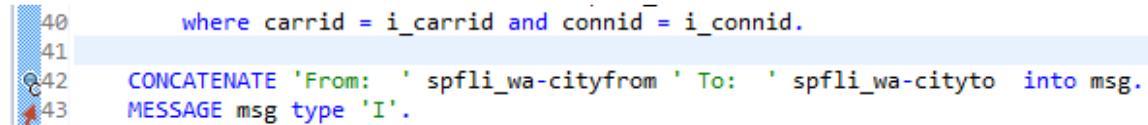
4. Choose *Apply* to save the new breakpoint's property.

### i Note

The validation of the condition syntax takes place at design time. The actual existence and validity of the operands (for example, variable symbols) however is checked at runtime as they cannot be checked properly at the time when they are created. If an operand is not valid, the program stops when it reaches the breakpoint and a corresponding error message occurs.

## Results

You have added a condition that is now active for the specified breakpoint. This is indicated by the new breakpoint decorator that is visible in the ruler of the source editor.



```
40     where carrid = i_carrid and connid = i_connid.
41
C42 CONCATENATE 'From: ' spfli_wa-cityfrom ' To: ' spfli_wa-cityto into msg.
43 MESSAGE msg type 'I'.
```

The symbol C is added to the breakpoint's decorator in the editor ruler

### Related Topics

- [Syntax for Breakpoint Conditions \[page 800\]](#)
- [Variables in Breakpoint Conditions \[page 801\]](#)
- [Literals in Breakpoint Conditions \[page 802\]](#)
- [Built-in Functions in Breakpoint Conditions \[page 803\]](#)
- [Operators in Breakpoint Conditions \[page 804\]](#)

## 5.3.1.2 Using Watchpoints

### Prerequisites

You can use watchpoints for variables only during a running ABAP debug session.

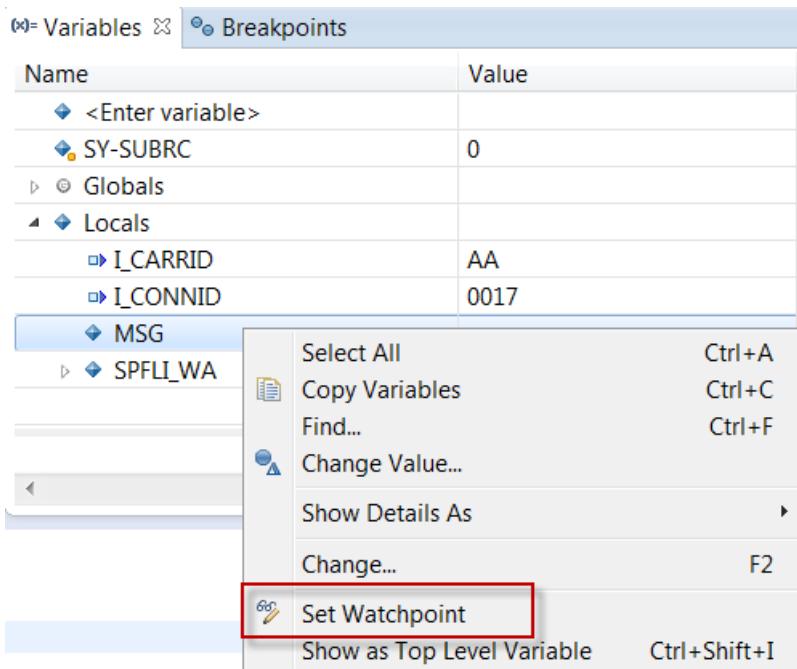
### Context

When debugging ABAP code, you can use watchpoints to track the value of individual ABAP variables. The ABAP Debugger stops as soon as the value of a watched variable has changed. Furthermore, you can specify conditions for watchpoints. The debugger then additionally checks whether this condition is fulfilled. [More on this: Watchpoints \[page 76\]](#)

### Procedure

1. Setting Watchpoints for Currently Selected Field in the ABAP source code editor:
  - a. In the ABAP source code editor of the *Debug* perspective, position the cursor on the field in question.
  - b. Open the context menu and select the option *Set Watchpoint*.
2. Setting Watchpoints in the *Variables* view:
  - a. In the Variables view of the *Debug* perspective, select the node for the variable in question.

- b. Open the context menu and select the option *Set Watchpoint*.

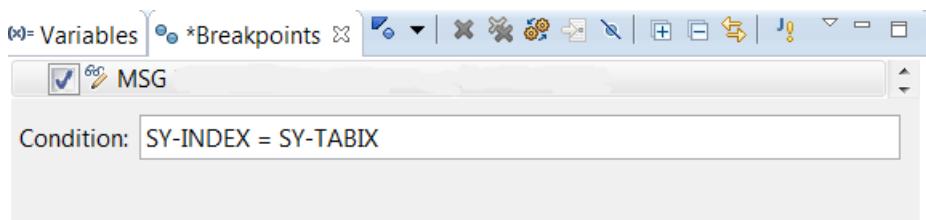


Setting watchpoints in the Variables view of the Debug perspective

3. Specifying Conditions for Watchpoints:

- Open the *Breakpoints* view of the *Debug* perspective.
- Select the watchpoint.
- Enter a valid condition in the corresponding entry field.

See also: [Conditions for Watchpoints \[page 620\]](#)

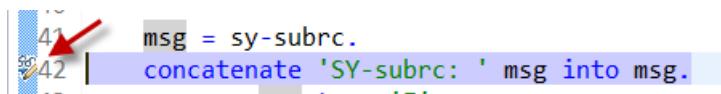


Editing a watchpoint condition

## Results

Once the watchpoint has been created for the debug session, the value of the specified variable will be monitored. The ABAP Debugger stops as soon as the value of the variable is changed after a debug step and the specified condition for the watchpoint is fulfilled.

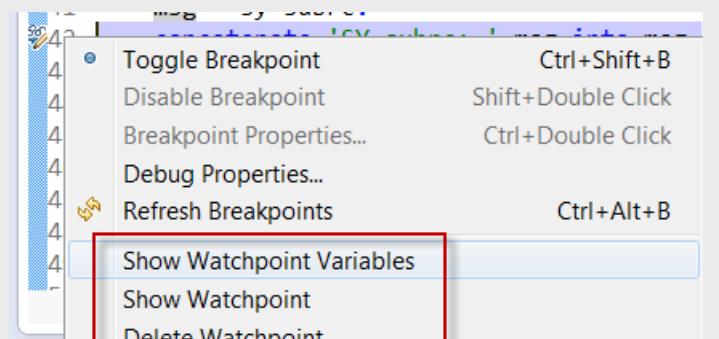
Once the watchpoint is reached, the editor range indicator displays a small watchpoint decorator.



Watchpoint decorator in the source editor

→ Tip

You can use the context menu of the watchpoint to access the watchpoint-related actions.



Watchpoint-related actions

i Note

ABAP watchpoints are always short-living in the *Debug* perspective of ABAP Development Tools.

They can only be active within a suspended debug session and are removed automatically when that debug session is terminated.

## Related Information

[Watchpoints \[page 76\]](#)

### 5.3.1.2.1 Conditions for Watchpoints

#### Context

You can specify additional conditions for watchpoints. The Debugger will then only stop if the value of the watchpoint variable changes AND the condition is fulfilled.

#### Syntax

A condition must have the following syntax:

```
<Function(Variable) or Variable or Literal> Operator <Function(Variable) or  
Variable or Literal>
```

The following functions are available when using watchpoint conditions:

#### For internal tables:

- `LINES( ITAB )` - Number of rows of an internal table ITAB
- `KEY_STATUS( ITAB KEY_NAME )` - Status of a secondary key KEY\_NAME of an internal table ITAB

#### For strings:

- `STRLEN( S )` - Current length of a string S

#### To identify inexact arithmetic operations :

- `INEXACT_DF( )` - Operation in which a rounding error occurred. The function `INEXACT_DF( )` returns the value 'X' or ' ', depending on whether the final result of an operation (for example, `COMPUTE` or `MOVE`) is inexact (has been rounded) or not.

#### Examples for Valid Conditions

- `sy-index > 5`
- `sy-index = sy-tabix`
- `lines( itab ) > 0`
- `lines( itab ) <> sy-tabix`
- `lines( itab ) < lines( itab2 )`
- `strlen( s ) >= sy-index`
- `KEY_STATUS( <itab> <key_name> ) = 1`
- `INEXACT_DF = 'X'`

#### Special Usage of Functions

The functions `KEY_STATUS( <itab> <key_name> )` as well as `LINES( <itab> )` are valid Debugger symbols. They can therefore also be used as watchpoint variables. So the Debugger would only stop for a watchpoint at `LINES( <itab> )` if the number of rows had not changed but the content of the table `<itab>` had.

Using `KEY_STATUS( <itab> <key_name> )`, you can monitor and query the secondary key of an internal table. The return values of this function are 1 (secondary key is out-of-date) or 0 (secondary key is up-to-date).

The (secondary) key (or index) of a table is not necessarily updated immediately for performance reasons; it is updated at the next read or write access to the table. This makes it difficult to determine which operations on the table lead to a given index needing to be refreshed. This is the situation where "`KEY_STATUS( <itab> <key_name> )`" can be used.

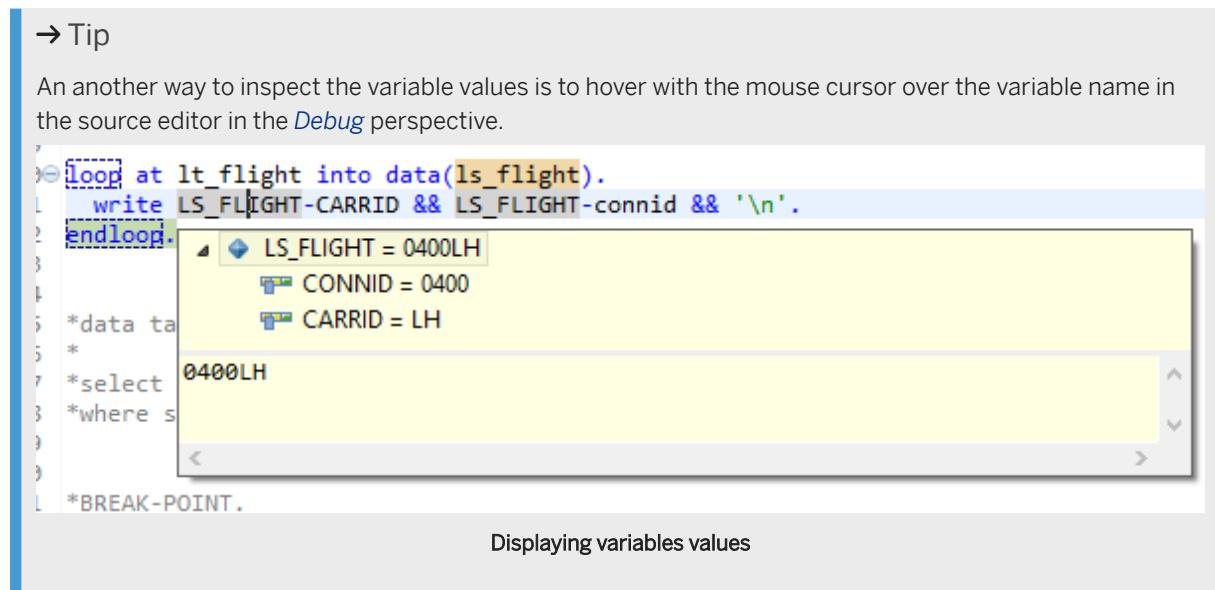
## Related Information

[Watchpoints \[page 76\]](#)

### 5.3.1.3 Managing Variables

#### Context

Double-click on any variable in the editor to open it for display in the *Variables* view. Internal tables are automatically also opened in the *ABAP Internal Table* view.



#### 5.3.1.3.1 Displaying Variables by Name

Alternatively, write the name of a variable in the *Variables* view or the name of an internal table in the *ABAP Internal Table* view to open the variable for display.

#### 5.3.1.3.2 Displaying Internal Tables

#### Context

You can automatically display the content of internal tables in the *ABAP Internal Table (Debugger)* view. This view lets you work comfortably with the rows in an internal table.

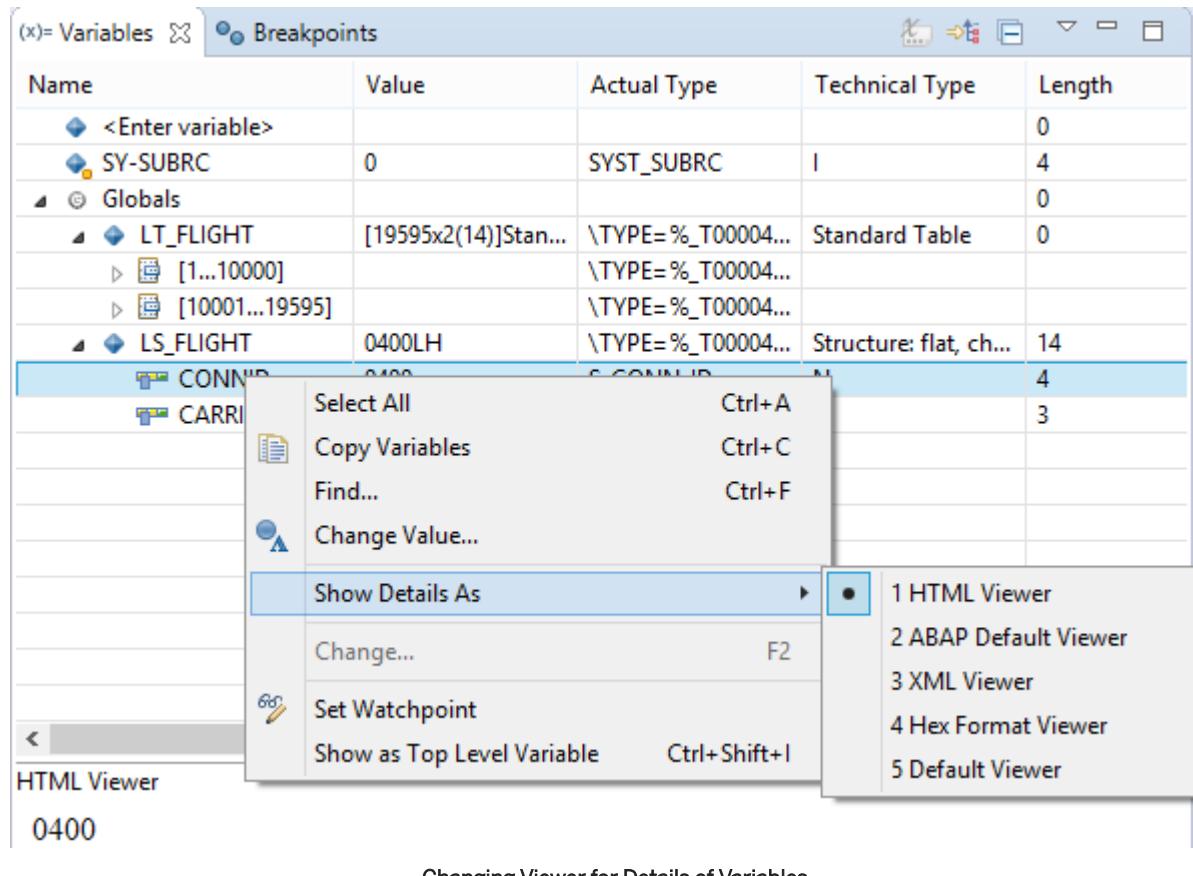
You can quickly scroll through long tables and rearrange table columns for better display with drag-and-drop.

More on this: [Analyzing Internal Tables \[page 625\]](#)

### 5.3.1.3.3 Displaying Hexadecimal, HTML, and XML Data

Are you confronted with unreadable hexadecimal data, or a string that contains HTML or XML tags?

Then choose **Show Details As** **HTML Viewer** **XML Viewer** to translate the data to text and to display the data using its HTML or XML formatting. You will find these functions in the context menu of unstructured variables in the *Variables* view.



### 5.3.1.3.4 Displaying Objects and Their Attributes

Analyzing a complex ABAP object and displaying its attributes becomes easier if you try the following:

- Expand the size of the *Variables* view while you are working with the object.
- Move a subordinate object to the top of the variables display to give yourself more display room. Choose *Show as Top Level Variable* from the context menu.
- If your object attributes are internal tables, display them in the *ABAP Internal Table* view by putting the cursor on the table and choosing *Show in Table View*.

### 5.3.1.3.5 Setting Variables

Type the value that a variable should have in the *Value* field in the *Variables* view, or in any field of a structure or row of an internal table.

The variable immediately assumes the new value in the debugger context.

You can also use the *Change Value* entry in the context menu to set the value of a variable. The function has an added value: It is inactive if the cursor is on a variable whose value cannot be changed.

### 5.3.1.3.6 Global and Local Variables

You may have noticed the folders *Locals* and *Globals* in the *Variables* view. These folders offer a fast way to find and display any active variable.

You have following possibilities:

- Select open *Locals* to see a list of all of the data variables that are defined locally in the procedure that is currently executing in the debugger.
- Select open *Globals* to see the globally defined data variables of the program that you are debugging.

### 5.3.1.3.7 Favorite Variables

#### Context

By default, ABAP Development Tools displays the `SY-SUBRC` system variable as a favorite, a variable that is automatically displayed in the *Variables* view.

You can add your own favorites, or even get rid of `SY-SUBRC`.

#### Procedure

1. Type in the name of the variable in the *Enter variable* field in the *Variables* view or double-click on a variable in the source code that you are debugging to add it to the *Variables* view.
2. Put the cursor on the variable name in the *Variables* view.
3. From the context menu of the variable, choose:
  - *Keep as Favorite* to toggle the status of the variable – automatically displayed favorite or not.
  - *Change* to type in a new variable name in place of the existing variable.

- [Remove](#) to delete the variable from the *Variables* view and from the list of favorite variables.

### 5.3.1.4 Analyzing Internal Tables

Internal tables open automatically in the [ABAP Internal Table \(Debugger\)](#) view. This view lets you work comfortably with the rows of an internal table. You can quickly scroll through long tables and use drag and drop to rearrange table columns for better display.

You have the following possibilities to analyze internal tables while debugging ABAP source code:

- [Displaying Internal Tables \[page 625\]](#)
- [Filtering \[page 626\]](#)
- [Sorting \[page 627\]](#)
- [Switching off Automatic Refresh for Faster Stepping \[page 627\]](#)
- [Using Column Configuration \[page 628\]](#)
- [Reusing the Table Status \[page 630\]](#)

### Displaying Internal Tables

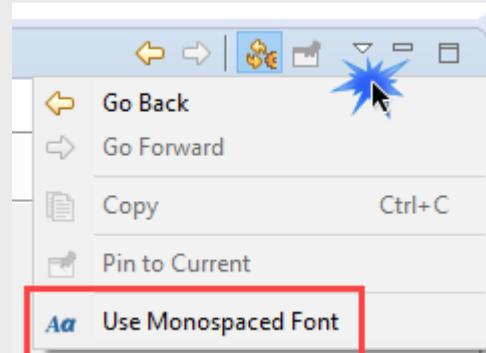
You have the following possibilities to display the content of internal tables from the [Debug](#) perspective:

1. Double-click the variable name from the ABAP source code editor.
2. Select an internal table in the source code that you are debugging and choose [Inspect Variable](#) from the context menu.
3. In the *Variables* view, double-click the node for the internal table in question.
4. Open the [ABAP Internal Table \(Debugger\)](#) view from the [Quick Access](#) and enter the name of the variable in the input field.

The content of the internal table is then opened in the [ABAP Internal Table \(Debugger\)](#) view.

#### i Note

For a precise string analysis, you can display the content of an internal table in monospaced fonts. To do this, choose [Use Monospaced Font](#) from the toolbar menu in the [ABAP Internal Table \(Debugger\)](#) view.



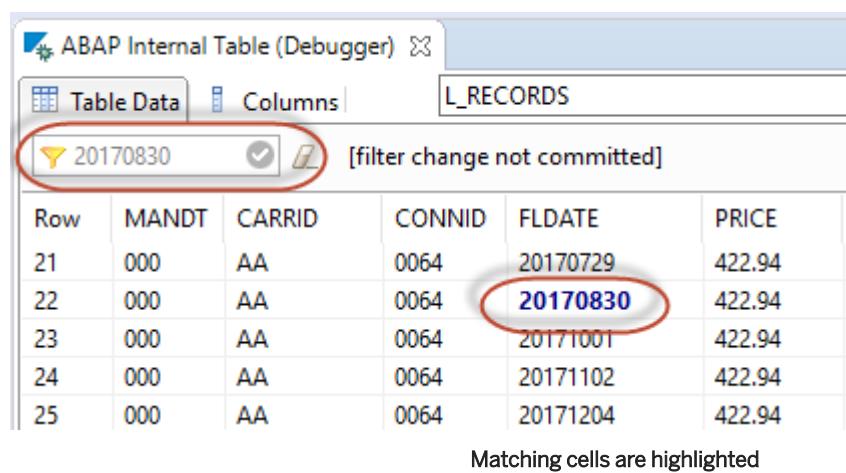
Opening the toolbar menu to display the content in monospaced font

## Filtering

The *ABAP Internal Table* view provides you with a quick filtering functionality. Once you start editing the filter text on the top left corner of the table view, you will immediately notice that the matching cells are highlighted in blue. The highlighting is adjusted as you type.

### → Remember

An asterisk (\*) is added implicitly to the beginning and the end of the filter text, unless you have already used \* or ? (? = any single character) within your filter text. Note that the filtering influences all columns of the table view.



Row	MANDT	CARRID	CONNID	FLDATE	PRICE
21	000	AA	0064	20170729	422.94
22	000	AA	0064	20170830	422.94
23	000	AA	0064	20171001	422.94
24	000	AA	0064	20171102	422.94
25	000	AA	0064	20171204	422.94

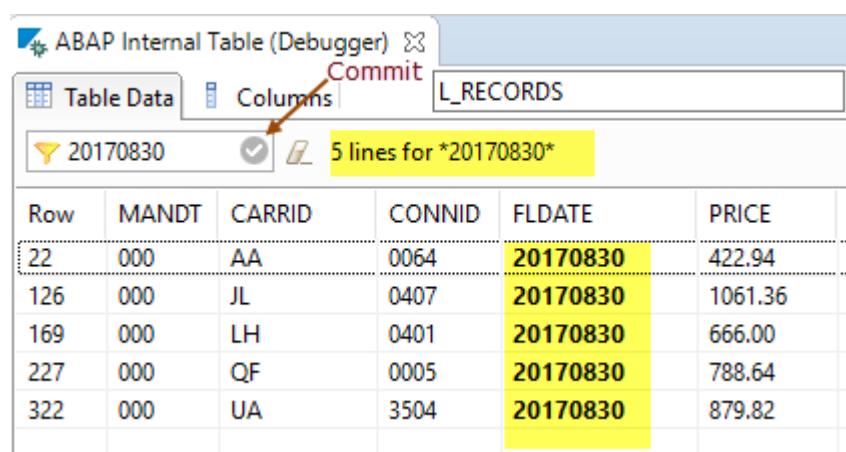
Matching cells are highlighted

### Reducing number of records displayed on UI

Once you have committed the filter text (by pressing **Enter** or by clicking the small gray *Commit* icon), only the matching records will remain visible, while all non-matching records will disappear.

### i Note

The filter simply reduces the number of records shown on the UI (in our example from 1000 to 5 lines). No records have actually been deleted from the internal table.



Row	MANDT	CARRID	CONNID	FLDATE	PRICE
22	000	AA	0064	20170830	422.94
126	000	JL	0407	20170830	1061.36
169	000	LH	0401	20170830	666.00
227	000	QF	0005	20170830	788.64
322	000	UA	3504	20170830	879.82

Reducing results after commit

## Resetting the filter

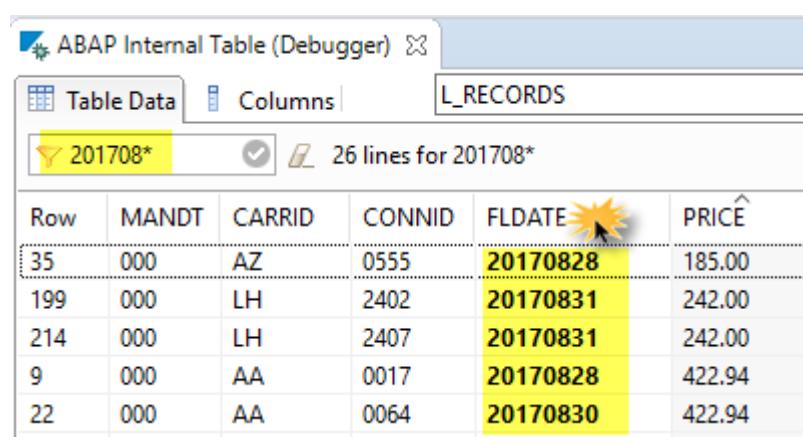
To reset the filter, choose *Clear Filter* (a single click on the eraser icon). All records will then be visible again.

## Sorting

You can also sort internal tables by clicking on one of the column headers. This also works in combination with the filter.

### → Remember

The sorting and filtering functionality also effects sorted tables and hashed tables.



The screenshot shows the ABAP Internal Table (Debugger) interface. The title bar says 'ABAP Internal Table (Debugger)'. Below it, there are tabs for 'Table Data' (which is selected), 'Columns', and 'L\_RECORDS'. Under 'Table Data', there is a filter bar with a funnel icon and the text '201708\*'. To the right of the filter bar, it says '26 lines for 201708\*'. The main area is a table with the following data:

Row	MANDT	CARRID	CONNID	FLDATE	PRICE
35	000	AZ	0555	20170828	185.00
199	000	LH	2402	20170831	242.00
214	000	LH	2407	20170831	242.00
9	000	AA	0017	20170828	422.94
22	000	AA	0064	20170830	422.94

Sorting records

### i Note

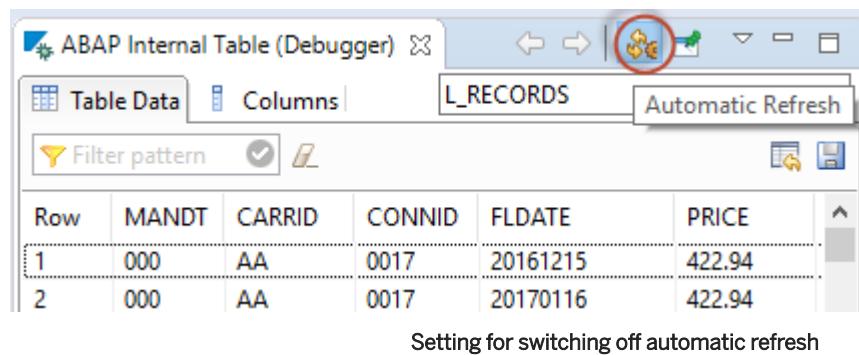
The sorting functionality only affects the UI presentation of the internal table, and does not actually sort the internal table in the context of the currently debugged process. The actual technical index of the records within the internal table is still visible on the left-most column called *Row*.

In the example above, this means that the record shown on top is at position 35 in the internal table. In the currently debugged ABAP program this record can still be addressed using the variable `L_RECORDS[ 35 ]`.

## Switching off Automatic Refresh for Faster Stepping

During debugging it is necessary to refresh all variable values after each (single) step. This is necessary because it is impossible for the ABAP debugger to know whether variable values have been changed during the step. For internal tables, this also means that the described sorting and filtering must be re-executed after each single step, since changed values can influence the sort order and the filtering. For small tables, the re-sorting / re-filtering is performed very quickly and you will not even know it is happening. For very large tables, it can take up to a couple of seconds however. Since this slows down the stepping speed significantly, there is an option for switching off the automatic refresh in the internal table tool.

To switch off automatic refresh, choose the corresponding Automatic Refresh icon in the toolbar in the table view.

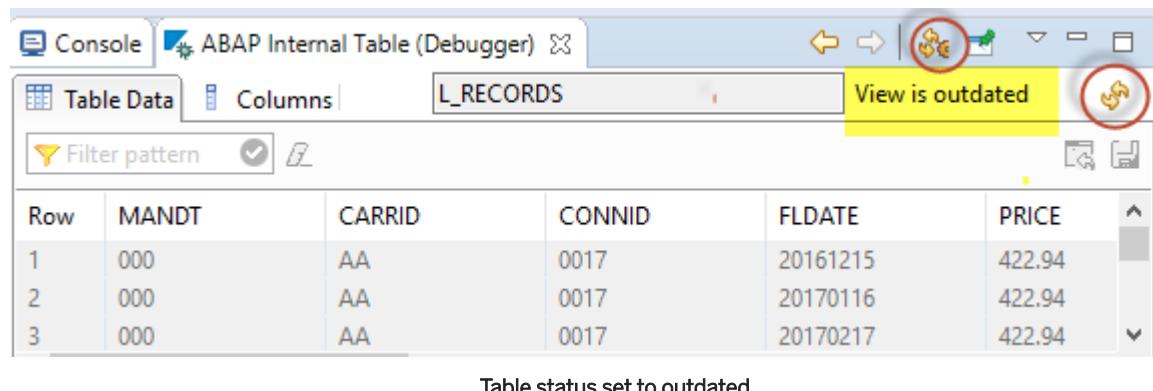


The screenshot shows the ABAP Internal Table (Debugger) interface. The toolbar has an icon with a circular arrow and a refresh symbol, which is highlighted with a red circle. The table view shows two rows of data:

Row	MANDT	CARRID	CONNID	FLDATE	PRICE
1	000	AA	0017	20161215	422.94
2	000	AA	0017	20170116	422.94

Below the table, the text "Setting for switching off automatic refresh" is displayed.

After the next debug step, the table view will switch the table status to *outdated*.



The screenshot shows the ABAP Internal Table (Debugger) interface. The toolbar has an icon with a circular arrow and a refresh symbol, which is highlighted with a red circle. The table view shows three rows of data, and the status bar at the top right says "View is outdated". The status bar also has a red circle around the refresh icon. The table data is identical to the previous screenshot:

Row	MANDT	CARRID	CONNID	FLDATE	PRICE
1	000	AA	0017	20161215	422.94
2	000	AA	0017	20170116	422.94
3	000	AA	0017	20170217	422.94

Below the table, the text "Table status set to outdated" is displayed.

The shown values are now potentially invalid (the internal table might not even exist anymore) and you will not be able to scroll or perform any actions on the data. The view will stay in this state until you perform a manual refresh (see figure above) or you re-activate automatic refresh.

#### i Note

To avoid confusion: There are no recognizable performance drawbacks unless you work with very large tables **and** you use sorting / filtering **and** you try to perform stepping at the same time. For the vast majority of situations, you can therefore simply use the default setting (*Automatic Refresh* is active).

## Using Column Configuration

The table view provides you with an additional *Columns* tab. You can use this tab for column configuration. It also offers a way to find columns faster and navigate to them directly, which is especially helpful for broad tables (internal tables with more than 100 columns).

### Using the filter for broad tables

To find certain columns faster, you can use the column filter at the top left.

ABAP Internal Table (Debugger)

Table Data    Columns    L\_RECORDS

seat

Positi...	Component	ABAP Type	Length
8	SEATSMAX	I	4
9	SEATSOCC	I	4
11	SEATSMAX_B	I	4
12	SEATSOCC_B	I	4
13	SEATSMAX_F	I	4

Using the filter function for columns in broader internal tables

## Navigating to data

A double-click on one of the listed columns lets you navigate directly to the data and show the clicked column on the left side.

## Changing column configuration

You have the option of changing the position of columns using drag and drop or the corresponding context menu functions.

ABAP Internal Table (Debugger)

Table Data    Columns    L\_RECORDS

Filter pattern

Positi...	Component	ABAP ...	Length	
1	MANDT	C	3	
2	CARRID	C	3	
3	CONNID	N	4	
4	FLDATE	D	8	
5	PRICE	P	8	
6	CURRENCY	C	5	
7	PLANETYPE	C	10	
8	SEATSMAX	I	4	
9	SEATSOCC	I	4	
10	PAYMENTS			
11	SEATSMAX			
12	SEATSOCC			
13	SEATSMAX			
14	SEATSOCC			

Options for configuring table columns

ABAP Internal Table (Debugger) X

Table Data Columns L\_RECORDS

Filter pattern ✖

Positi...	Component	ABAP ...	Length	
1	SEATSMAX	I	4	
2	SEATSOCC	I	4	
3	MANDT	C	3	
4	CARRID	C	3	
5	CONNID	N	4	

Re-arranged columns

## Reusing the Table Status

In earlier ADT versions, it was already possible to make certain rudimentary adjustments in the table tool, for example to, change the width of the columns. One single step was enough to completely discard the personal adjustments. With the current version of ADT, the table status (sum of all personal adjustments, including column positions, width of columns, sort column and filter) will remain during stepping, and will not become lost if you switch to a different variable.

### Using the column configuration

#### Example

1. You display the first variable `first_itab`. The default configuration is used.
2. Change the column configuration and enter a filter. This changes the column configuration.
3. Display the second variable `second_itab`. The default configuration is used again.
4. Display the first variable `first_itab` again. The previous (changed) column configuration and the filter from step 2 will now be restored automatically.

The table status is reset once the debug session ends or the table view is closed. You can also reset the status manually with a single click by choosing the *Reset Column Configuration*  icon.

## 5.3.1.5 Displaying Exceptions

### Context

Use the following functions to display the location and attributes of an exception object.

#### i Note

If an exception occurs when [Navigating in the Debugger \[page 635\]](#), the debugger notifies you of the exception by displaying a red '!' exclamation point in the left hand gutter of the editor window.

## Procedure

Use the context menu of the line marked with the '!' exclamation point to use these functions:

- *Go to Exception Raise Location*: Jump to the location of the exception
- *Show Exception Variable*: Display the attributes of the exception object.

### 5.3.1.6 Debugging Dynpro Flow Logic

Starting with ABAP Development Tools (ADT) client version 2.83, you have the option of debugging the flow logic of dynpro screens (PBO, PAI, and so on). Dynpro debugging enables you to step from the ABAP code to the screen flow logic, and vice versa. Both dynpro and ABAP entries are represented in a common call stack of the current debug session.

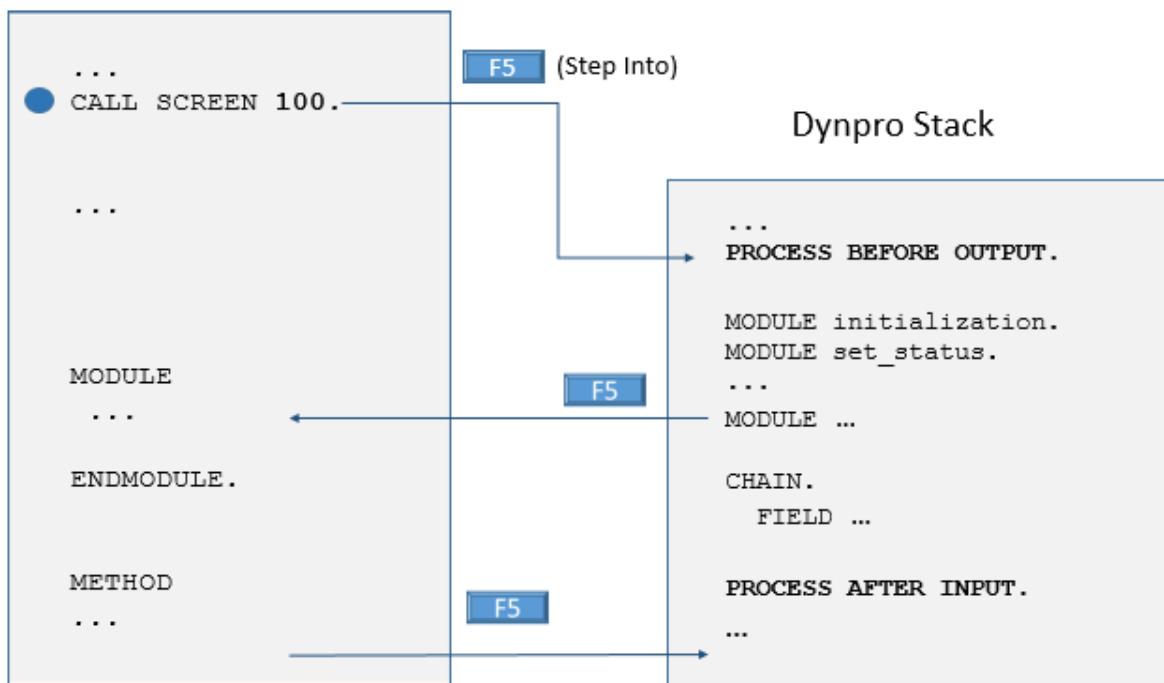
#### → Remember

In previous ADT versions, you were not able to view the dynpro code during stepping or in the call stack view of the ABAP debugger. A single step at an ABAP statement such, as `CALL SCREEN` for example, made the debugger stop at the next ABAP statement after the dynpro screen was processed.

## Stepping and Call Stack Navigation

The following figure demonstrates the stepping in a mixed call stack that includes ABAP and dynpro entries. In our example, the debugger can stop at the `CALL SCREEN` statement, which serves as an ABAP stack entry. To step into the called screen 100, you use the `F5` key. The debugger then navigates to the PBO of the screen and you can now debug the flow logic within the corresponding dynpro stack entry. And, vice versa, starting from the dynpro code, you can call entries on the ABAP stack – again by pressing `F5`.

## ABAP Stack

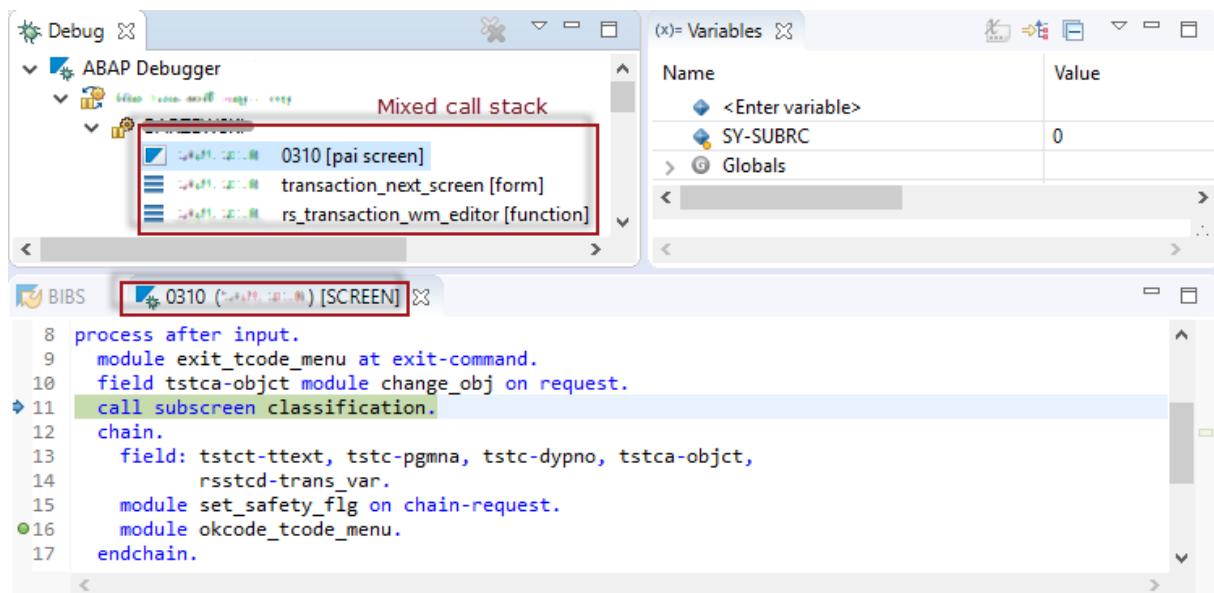


Mixed call stack with dynpro and ABAP stack entries

### i Note

The ABAP Debugger editor displays the dynpro flow logic in pure read-only mode. It does not support any code changes).

As illustrated in the figure below, the small debugger icon on the tab header indicates the use of the debugger editor. In addition, you can see the screen number (0310) and the name of the ABAP program that the dynpro screen belongs to (SAPLSEUK). The *Variables* view provides access to the variables of the underlying ABAP program.

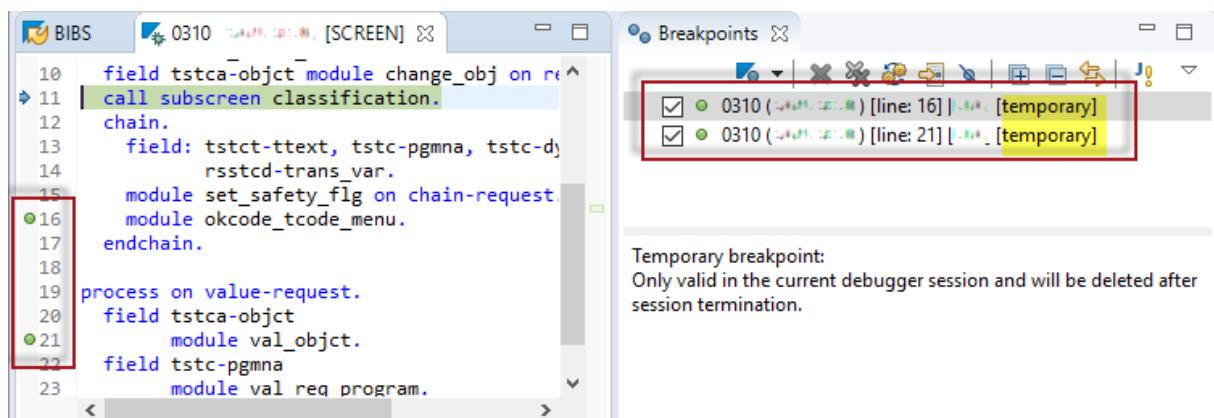


Analyzing dynpro flow logic in the ABAP Debugger of ADT

## Adding Temporary Breakpoints

When stepping through debugged code, you have the option of adding further breakpoints in the debugger editor. You add these breakpoints on demand in the debugger editor. They are green in color and are decorated with the tag *[temporary]* in the *Breakpoints* view.

Note that these temporary breakpoints are only valid during the current session and will disappear automatically after termination of the debug session.



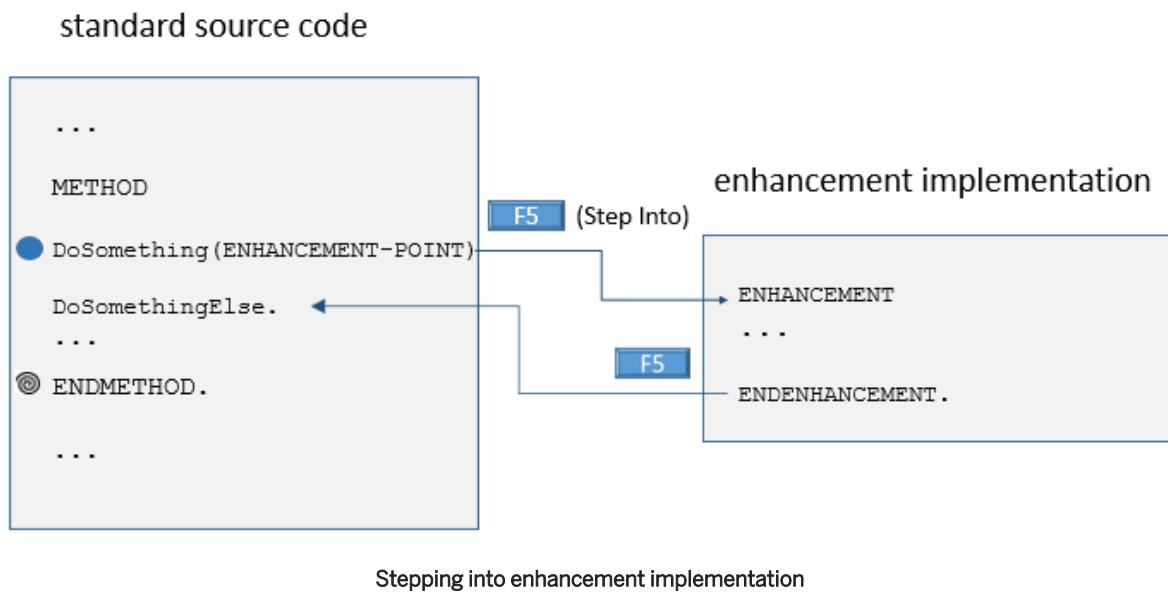
Temporary breakpoints are only valid during the current debug session

### 5.3.1.7 Debugging Enhancement Implementation

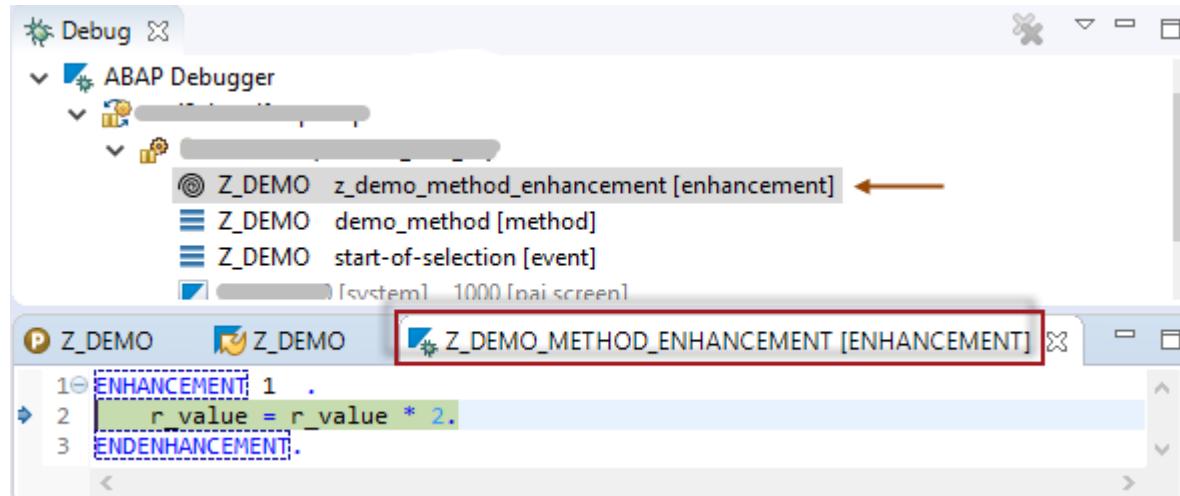
Starting with ABAP Development Tools (ADT) client version 2.83, you have the option of debugging the enhancement implementation in the ABAP debugger. Just like stepping into dynpro flow logic by pressing **F5**, it is also possible to step into enhancement implementation (implicit and explicit).

## Stepping and Call Stack Navigation

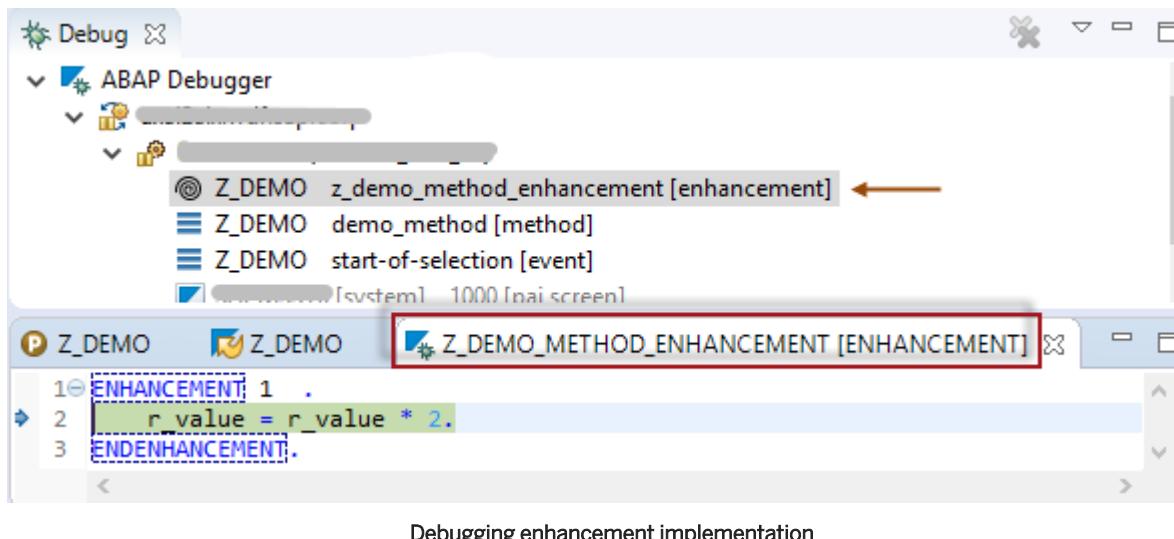
The following figure demonstrates stepping from standard source code into enhancement implementation, and vice versa. In our example, the debugger can stop in the standard source code at an enhancement-point within a method implementation. After pressing **F5** (Step Into), the debugger will reach the implicit enhancement implementation.



The debugger editor displays the enhancement code in pure read-only mode and has the same characteristics as known from dynpro debugging. **More on this:** [Debugging Dynpro Flow Logic \[page 631\]](#). The enhancement implementation is part of the call stack and is indicated with the tag `[enhancement]`. The *Variables* view provides access to the related enhancement implementation.



Debugging standard source code that includes an enhancement-point

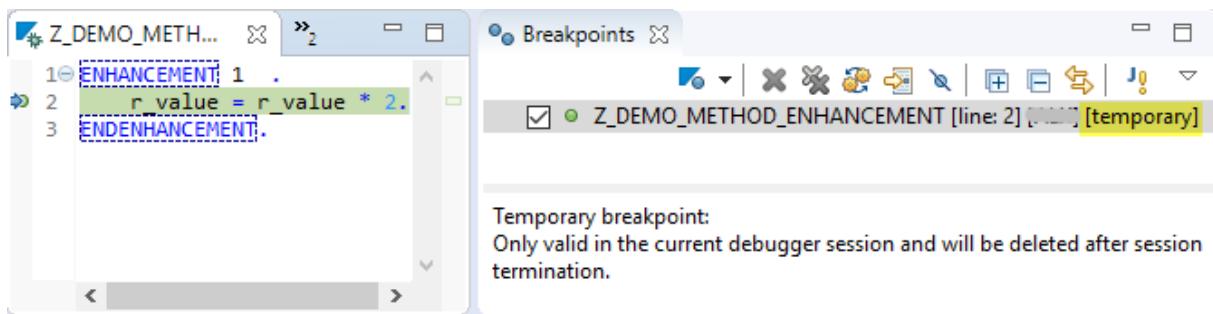


## Adding Temporary Breakpoints

When stepping through debugged code, you can – on demand – create temporary breakpoints in the debugger editor. These breakpoints are green in color and are decorated with the tag *[temporary]* in the *Breakpoints* view.

### i Note

Temporary breakpoints are only valid during the current session and will disappear automatically after termination of the debug session.



Temporary breakpoints when debugging enhancement implementation

### 5.3.1.8 Navigating in the Debugger

#### Context

You will find tools for stepping through and navigating within code in the *Debug* perspective. By default, the corresponding buttons are available in the toolbar at the top of the perspective.

This section briefly explains the navigation options that you have.

Button / Shortcut	Description
Resume	Run to the next breakpoint or to the end of the program.
Terminate	Abort the execution of the program in the debugger. Program execution ends.
Disconnect	Run to the end of the program, ignoring any intervening breakpoints.
Step Into (F5)	Execute the next single ABAP instruction in the program in the debugger. Step into a called procedure.
Step Over (F6)	Execute the next ABAP statement. If the next step is a procedure call, run the entire procedure.
Step Return (F7)	Run until the current procedure returns to its caller or until the program ends.
Run to Line (Shift F8)	Run to the statement on which the cursor is positioned.
Jump to Line (Shift F12)	Put the execution pointer to the statement on which the cursor is positioned without executing of any other statement in between. Consider that variables are not reset to reflect a restart from an earlier code position.
Jump to code in call stack	Click on an entry in the call stack in the <a href="#">Debug</a> view to open the code in the editor. The cursor is positioned at the point at which the call happened.

**i** Note

The **Step Filter** function is shown as active in the ABAP debugger, but it can only be used with Java coding. For an equivalent functionality, see layer-aware debugging in the back end debugger.

### 5.3.1.9 Managing Technical Conflicts During Debugging

In the special case that **multiple people** are using **the same user name and password** (group users), some technical conflicts can arise.

#### SAP Cloud Platform ABAP Environment

The first one is the option [This project only](#) within the project-specific settings which allows you to work isolated for exactly one certain ABAP project. If this option is chosen you will not experience any of the above mentioned issues. But you can only debug requests that are started from this certain project and no external requests (RFC, HTTP).

The second way to avoid the mentioned issues is that only one project at a time is allowed to debug external requests for a certain user. Other projects that are up to debug external requests for the same user are suspended. Suspended projects show their status via different UI elements (for example a red cross icon on

every breakpoint, hover texts and messages). It is not possible to debug in a suspended project. From every suspended project it is possible to re-activate the breakpoint activation, taking back the external debugging scope and suspend the currently active project.

## Related Information

[Setting ABAP Project-Specific Debug Settings \[page 638\]](#)

### 5.3.1.10 Setting ABAP Debugging Preferences

#### Context

You can change how the ABAP debugger behaves with the [General Settings](#).

#### Procedure

1. Open the *debugger preference* page. (Choose  [Window](#)  [Preferences](#)  [ABAP Development](#) .)
2. Change debugger settings as required. The settings apply to all of your ABAP projects.

Enable debugging of system programs

Mark this checkbox to make ABAP system programs visible in the debugger. If this option is not activated, then the debugger executes code in system programs without displaying the code. The debugger also does not stop at breakpoints in system programs.

A program is a system program if its *Program Status* is set to **SYSTEM**. A system program is part of the ABAP application server infrastructure. It may, for example, work with client-independent tables or run in special clients, such as the system client or the administrative client.

---

Always create exception object

Mark this option to have the debugger force creation of an exception object even if the `INTO ref` addition is missing from the `CATCH` statement.

In the event of an exception, you can then examine the attributes of an exception object, in order to help you with the analysis of the exception. **More on this:** [Displaying Exceptions \[page 630\]](#).

---

Suspend sending of background RFC requests

Not applicable.

---

Enable debugging of automatic construction of shared object areas	Mark this option to have the debugger start on entering the process that creates a shared object memory area.
Show message if breakpoint cannot be set	Mark this option to enable or disable warning messages, if a breakpoint that you have requested cannot be set. Breakpoints cannot, for example, be set on some ABAP statements for source code modularization, such as CLASS or PUBLIC SECTION. If this option is marked, then you will be warned that you cannot set a breakpoint at such a statement. Otherwise, the warning is suppressed.
Show message if breakpoint activation conflict occurs	Mark this option to get a notification popup if a different person (IDE / project) is using the same debug scope as you do currently. <b>More on this:</b> <a href="#">Managing Technical Conflicts During Debugging [page 636]</a>

## Related Information

[Setting ABAP Project-Specific Debug Settings \[page 638\]](#)

### 5.3.1.11 Setting ABAP Project-Specific Debug Settings

#### Context

You can use project-specific debug settings to change the user for which ABAP breakpoints are in effect.

##### i Note

Debugging SAP public services: HTTP services that are defined under `sap/public` in transaction `SICF` are handled without an explicit logon. (They are processed under a special system user). This means that the breakpoints that you set in ADT do not work for debugging of `sap/public` services; a request does not run under your user, and you cannot set an external breakpoint to catch processes of the special system user. One workaround for debugging such services is to make a copy of your service under a different `SICF` path, `sap/bc`, for example. Then requests to the service run under your user, and you can stop in the debugger in your ABAP service handler.

#### Procedure

1. Open the *debugger* preference page. (Choose     .)

2. Choose **Configure Project Specific Settings...** and choose the ABAP Project for which you wish to change the customizing.
3. [Optionally:] Specify the validity scope of ABAP breakpoints that have been created in ADT.

Option	Description
<b>This project only</b>	Enables debugging of requests from within this ABAP project only. With this option, you can only debug requests that are started from within this certain project but no external requests (RFC, HTTP). <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> <b>i Note</b> <p>If this option is chosen you will not experience any issues that are reported in the topic: <a href="#">Managing Technical Conflicts During Debugging [page 636]</a>.</p> </div>
<b>Logon user</b>	Enables debugging of any request for the logon user (default option). This means that for server-requests that have been processed on the ABAP server, only the ones running under your logon user (SY-UNAME) are able to start a debug session - if they reach a breakpoint. Requests that are executed under a foreign user will not start a debug session and just ignore the breakpoint. Therefore you will not be able to stop/debug the requests of your colleagues with this option.
<b>User</b>	Enables debugging of any request for the specified user. If you want to debug requests of a foreign user you can choose this option and enter the respective user name. This allows you to debug also external requests like RFC and HTTP, as long as they are processed under the specified user on the ABAP server.
<b>Terminal ID</b>	Enables debugging of any request with a Terminal ID.

4. [Optionally:] Use [Advanced Settings](#) and mark **Allow debugging of tool requests** if you are developing ABAP components for ABAP Development Tools itself.
5. [In case of AMDP debugging:] You have the option to view or change the settings for AMDP debugging.  
[More on this:](#)

## Related Information

[Setting ABAP Debugging Preferences \[page 637\]](#)

[Breakpoints in the Debugger - Characteristics \[page 75\]](#)

## 5.3.2 Profiling ABAP Code

ABAP Profiler lets you analyze the runtime of ABAP programs.

### Usage

The ABAP Profiler allows you to examine the performance of transactions, ABAP programs, function modules, global ABAP classes, and Web Dynpro applications.

You can use the ABAP trace (the results of the ABAP Profiler) to identify runtime-intensive statements and to follow the hierarchy of program calls.

#### → Tip

You will find a compact overview and introduction to ABAP Profiler in the *SAP Community Network*. To read the related blogs, click [here...](#)  and [here...](#) 

Starting from the ABAP trace, you can:

- Identify performance hot spots where you might want to concentrate your re-factoring effort.
- Identify excessive or unnecessary use of modularization units.
- Identify CPU-intensive program functions
- Analyze the flow of your application.

#### → Tip

If you need to analyze or compare program flow, the ABAP Profiler can be a faster and more effective tool to use than the ABAP Debugger.

### Related Information

[Starting the ABAP Profiler \[page 640\]](#)

[Displaying the Trace Overview \[page 649\]](#)

[Using Analysis Tools \[page 650\]](#)

[Setting Trace Preferences \[page 678\]](#)

[Glossary \[page 690\]](#)

## 5.3.2.1 Starting the ABAP Profiler

### Before You Start Profiling...

- Run your program before profiling it: It's a good idea to run your program at least once before you launch the ABAP Profiler. That way, any side-effects of compilation, initialization of caches, or other time-users, are eliminated before you actually create an ABAP trace (result of the ABAP Profiler).

## You Can Start Profiling in Several Ways...

- By starting the ABAP Profiler for an individual ABAP program **directly** from the source editor or from the *Project Explorer* view.  
**More on this:** [Launching Profiler for an ABAP Development Object \[page 641\]](#)
- By launching the **profiler wizard tool** - independently of any context, you can select an arbitrary development object to be profiled.  
**More on this:** [Creating an ABAP Trace Using a Wizard \[page 642\]](#)
- By **creating a trace request**. Certain programs or program units cannot be started directly as ABAP applications. As an alternative to profiling such a program in its ABAP Unit tests, you can create a trace request and profile a method or function module whenever its execution is triggered in the back-end system by a transaction, HTTP request, RFC call, or within a background process  
**More on this:** [Creating an ABAP Trace Request \[page 646\]](#)
- By using a predefined **launch configuration** for the ABAP Profiler  
**More on this:** [Working with Profile Configurations \[page 649\]](#)
- Other ways** to start the ABAP Profiler for a program that you cannot start directly:
  - By using the debugger to start or stop the ABAP Profiler.  
**More on this:** [Starting and Stopping the ABAP Profiler in the Debugger \[page 644\]](#)
  - By embedding code that includes the method or other object that you want to profile in an ABAP PROGRAM. You can then use the standard option  **Profile As**  to profile the method.

### 5.3.2.1.1 Launching Profiler for an ABAP Development Object

#### Context

You want to start the ABAP Profiler for an individual ABAP program directly from the ABAP source code editor or from the *Project Explorer* view.

#### Procedure

- From the ABAP source editor or from the *Project Explorer*, choose **or**  **Profile As**  **ABAP Application (Console)**  **or**  **Profile As**  **ABAP Unit Test**.

### i Note

Cloud icon You can execute ABAP Unit tests to profile ABAP classes.

To do so, use .

2. [Optional-]When your program has completed, switch to the  perspective.
3. Open the  view. The new ABAP trace should already be shown; if it is not, simply refresh the list of traces.
4. Double-click on a trace to open it in the **Overview** page. You can also start an appropriate analysis tool, such as for example, the **Trace Hit List** or the **Call Sequence** (for non-aggregated traces) from the context menu of a trace entry.

## Related Information

[Finding Hot Spots with the Hit List \[page 657\]](#)

[Determining Runtime with the Hit List \[page 659\]](#)

[Analyzing Program Flow with the Call Sequence \[page 661\]](#)

[Understanding Parameters in the \(Condensed\) Hit List \[page 654\]](#)

### 5.3.2.1.2 Creating an ABAP Trace Using a Wizard

#### Context

Independently of any context you can start profiling development objects using the corresponding wizard tool.

To do this, you launch the profiling dialog using a shortcut. Then you specify the object (transaction, program ...) that you wish to profile in the selected system.

You can only start the profile function for a subset of development objects that are executable in the ABAP Workbench.

More precisely, the dialog enables you to start profiling for the following object types:

- Transactions
- ABAP programs
- Function modules
  - runs the test environment for function modules
- Global ABAP classes
  - runs test environment for the selected class
- Web Dynpro applications

### i Note

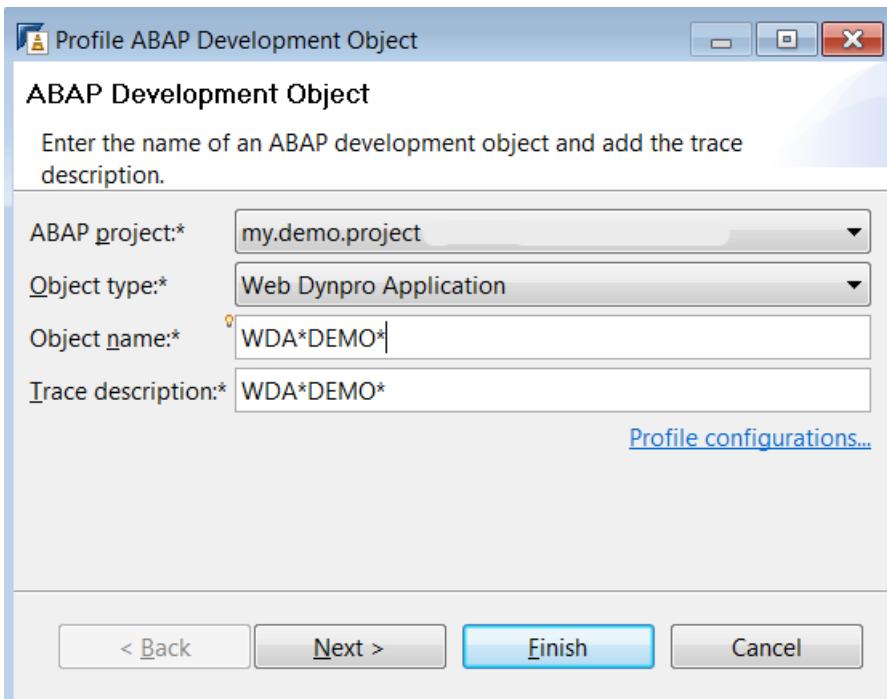
The range of object types for which you can start tracing depends on (the release of) the backend system you are connected to.

## Procedure

1. Call the profiling dialog using the menu **Run > Profile ABAP Development Object...** (shortcut **Alt + F9**).
2. Select the relevant ABAP project – if you have not already done so.
3. Select the Object Type and enter the (part of the) name of the development object or transaction for which you want to start profiling.

### → Tip

When editing the name, you can avail of the content assistant functionality by pressing **Ctrl + Space** in the Object name field.



Profiling dialog

4. [Optional:] Choose **Next** to adjust the [Understanding ABAP Profiler Settings \[page 680\]](#) to your needs.
5. Choose **Finish**.

## Results

ABAP Development Tools runs the selected development object or opens the corresponding test environment within the integrated SAP GUI.

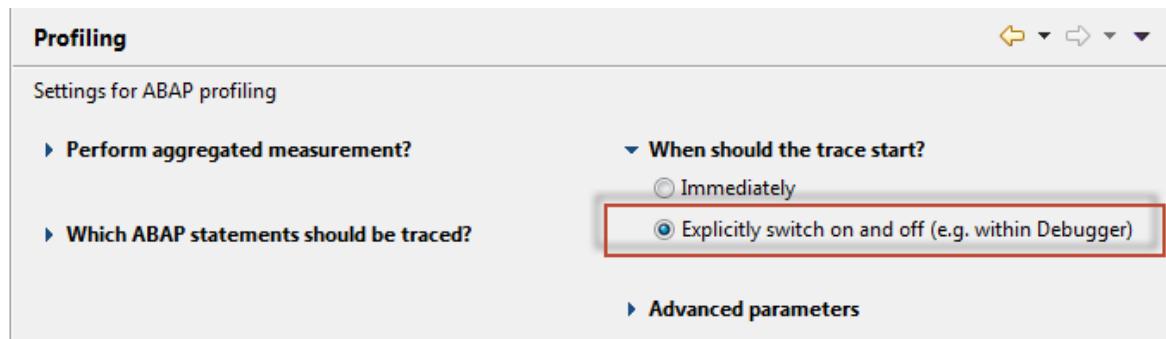
### 5.3.2.1.3 Starting and Stopping the ABAP Profiler in the Debugger

#### Context

Let's assume you want to create an ABAP trace for a source code with a specified start and end position. You can now avail of the option to explicitly switch the measurement function on and off. In particular, you can switch the ABAP Profiler on/off at defined breakpoints when debugging your program. The ABAP Profiler then captures only those source code sections for which you want to analyze the runtime, and not for the entire program execution.

#### Procedure

1. Before you launch the debugger, set a breakpoint at that line of ABAP source code that you want to analyze in detail.  
**More on this:** [Setting Breakpoints at a Line in Code \[page 614\]](#)
2. Start the Profiler wizard using the menu  **Run**  **Profile ABAP Development Object...**  (shortcut: **Alt** + **F9**).
3. In the wizard page that appears, select the relevant *ABAP project* and the *Object type*, and then enter the name of the object for which you want to start the Profiler. In addition, you can specify a specific *Trace title* for the ABAP trace to be created. Then choose **Next**.  
On the next page, you can adjust the ABAP Profiler settings to your needs. **More on this:** [Understanding ABAP Profiler Settings \[page 680\]](#)
4. Under the heading *When should the trace start?*, choose the option *Explicitly switch on and off*.

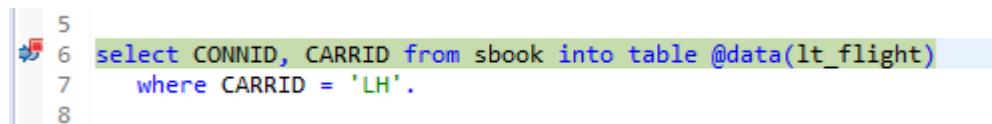


Configuration option for starting ABAP Profiler within the Debugger

5. Close the Profiler wizard with **Finish**.

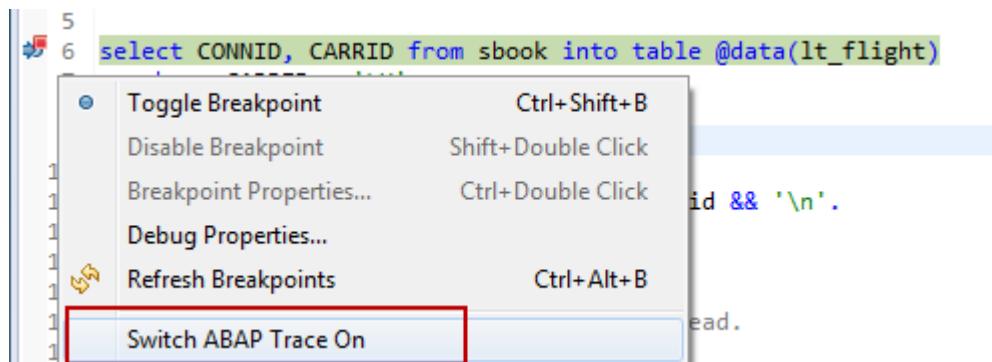
6. Switch to the Debug perspective - should this be required.

The execution stops at the specified breakpoint. The breakpoint's decorator within the ruler indicates that the ABAP Profiler is currently switched off.



```
5
6 select CONNID, CARRID from sbook into table @data(lt_flight)
7   where CARRID = 'LH'.
8
```

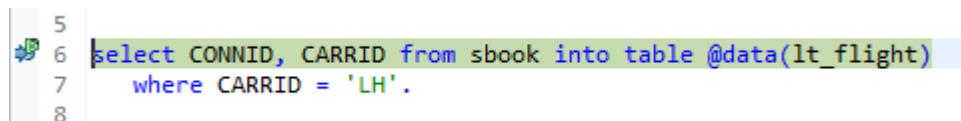
7. On the ruler, open the context menu and choose *Switch ABAP Trace On*.



```
5
6 select CONNID, CARRID from sbook into table @data(lt_flight)
7   where CARRID = 'LH'.
8
```

Switch ABAP Trace On

The decorator of the breakpoint now indicates that the ABAP Profiler is switched on.



```
5
6 select CONNID, CARRID from sbook into table @data(lt_flight)
7   where CARRID = 'LH'.
8
```

8. When the debugger stops again after stopping at another breakpoint, you have the option to explicitly switch off the ABAP Profiler. To do so, open the context menu on the ruler and choose *Switch ABAP Trace Off*.

**i Note**

The ABAP Profiler will be switched off automatically when the application is terminated.

## Results

As a result of this procedure, the Profiler creates an ABAP trace for the selected development object. You will find it in the *ABAP Traces* view under the *ABAP project* and *Trace title* - as you specified in step 3.

## Related Information

[Displaying the Trace Overview \[page 649\]](#)

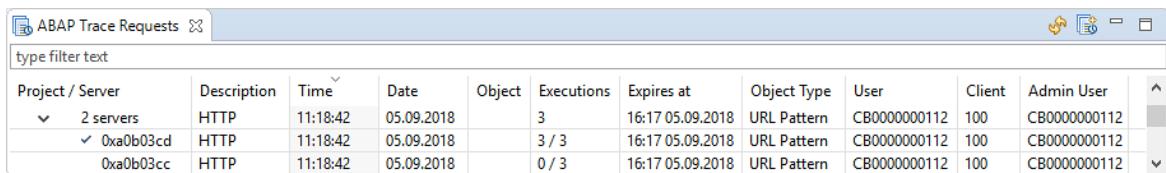
## 5.3.2.1.4 Creating an ABAP Trace Request

### Context

You can trace the execution of an ABAP request even if you cannot start it directly yourself. That is, you can create a trace request: Tell the ABAP Profiler to wait until a request starts and then automatically profile its execution.

### Procedure

1. [Optional-]Switch to the **ABAP Profiling** perspective.
2. Open the **ABAP Trace Requests** view.
3. Then click on the **Create Trace Request** button  (**Ctrl** + **N**) or choose the corresponding function from the context menu of a line in the view.
4. In the *Trace Schedule* wizard that appears, you can specify what you want to profile. Look at [Filling Out the ABAP Trace Request Dialog \[page 647\]](#) for more information on the fields in the wizard page.
5. Click on **Finish** to schedule your traces.
6. Open the new entry in the **ABAP Trace Requests** view to see the status of your asynchronous trace request.



ABAP Trace Requests											
type filter text											
Project / Server	Description	Time	Date	Object	Executions	Expires at	Object Type	User	Client	Admin User	
2 servers	HTTP	11:18:42	05.09.2018	3	3	16:17 05.09.2018	URL Pattern	CB0000000112	100	CB0000000112	
✓ 0xa0b03cd	HTTP	11:18:42	05.09.2018	3 / 3	3 / 3	16:17 05.09.2018	URL Pattern	CB0000000112	100	CB0000000112	
0xa0b03cc	HTTP	11:18:42	05.09.2018	0 / 3	0 / 3	16:17 05.09.2018	URL Pattern	CB0000000112	100	CB0000000112	

Any request that meets your scheduling specifications (type of activity, user, client, and so on) will be traced. It does not matter whether the program was started from the ABAP Development Tools client or from the back-end system.

7. Use the **Refresh** (**F5**) function in the context menu to monitor your trace request. When the **Executions** column in **ABAP Trace Requests** shows that at least one trace has been made, you can switch to the **ABAP Traces** view to examine the trace. To do so, mark the trace request entry and choose **Show in ABAP Traces**.
- The cursor will be positioned on the trace entry in the **ABAP Traces** view.
8. Double-click on a trace to open it in the **Overview** view. You can then use the analysis tools to examine the results in detail.

### Related Information

[Displaying the Trace Overview \[page 649\]](#)

[Using Analysis Tools \[page 650\]](#)

## 5.3.2.1.4.1 Filling Out the ABAP Trace Request Dialog

### Context

The table below explains how to fill out the fields on the [ABAP Trace Request](#) dialog. The dialog lets you schedule asynchronous traces of ABAP program executions.

#### • Example

For example, you can trace HTTP requests or RFC calls that are processed by the back-end system.

Field	Description
<i>Which requests do you want to trace?</i>	<p>Here you tell the ABAP Trace what sort of activity you want to trace. The available request types are:</p> <ul style="list-style-type: none"><li>• <a href="#">Web Requests (HTTP)</a> tells the system to trace an HTTP request that arrives at the ABAP back-end system.</li><li>• <a href="#">Any</a> lets you capture any type of activity. Use it if you do not know exactly which kind of requests are involved. Or, alternatively, use it if you want to specify when the activity you want to trace takes place (so that you know when it will run next).</li></ul> <p>Other options have the following meanings:</p> <ul style="list-style-type: none"><li>• <a href="#">Remote Function Modules (RFC)</a> traces an RFC call to a specified function module.</li><li>• <a href="#">Background Jobs</a> traces a specified background job in the back-end ABAP System.</li><li>• <a href="#">Shared Objects Area</a> traces the area constructor of an ABAP Shared Objects shared memory area. (Tracing is performed only if the <a href="#">Automatic Area Building</a> property is set for the shared memory area. An area constructor must be specified for automatic area building.)</li></ul>

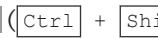
Field	Description
<i>Limit to</i>	<p>Depending on the previous selection, the following fields are available to specify what you want to trace:</p> <div data-bbox="822 428 1102 462" style="background-color: #e0e0e0; padding: 5px;"><b>→ Recommendation</b></div> <p>Choosing additional filters is optional, but it allows you to limit the number of trace files created. Therefore, SAP recommends that you set the relevant filters as precisely as possible.</p> <p><i>URI Pattern</i>: This selector enables you to focus tracing for a specific string pattern.</p> <div data-bbox="822 720 906 754" style="background-color: #e0e0e0; padding: 5px;"><b>i Note</b></div> <p>This field is only available when you specify the object type <i>Web Requests (HTTP)</i>.</p> <p><i>Object type</i>: This selector appears to specify the object to be traced depending on the previously selected request type.</p> <p><i>Object Name</i>: The legend of this field changes to reflect the type of object you want to trace. In this field, you specify the name of the object trace. This can be an ABAP program name or transaction.</p> <p><i>User and client</i>: The activity is traced in your current client. It is often best to leave <i>User</i>: empty. Enter a user name if you are sure that the development object you want to trace runs under a particular user.</p> <p><i>Server</i>: You can schedule a trace on all ABAP application servers of your back-end system or only on the current server.</p> <div data-bbox="822 1432 1102 1466" style="background-color: #e0e0e0; padding: 5px;"><b>→ Recommendation</b></div> <p>SAP recommends that you schedule a trace on all servers. The ABAP load-balancing features make it hard to predict the server on which a program will run.</p>
<i>Which restrictions apply?</i>	<p>Here you can specify how many traces are performed. Schedule more than one trace if you need to be sure of capturing the right instance of a program run. Once the requested number of traces is reached, no further traces are run on that ABAP application server.</p> <p>You can also change the expiration time of a trace request. By default, the request is canceled after one hour.</p>
<i>Title of the trace file</i>	<p>The <i>Title</i> is the trace title shown to you in the <i>ABAP Traces</i> view. The ABAP Trace generates a description for you, or you can enter your own description.</p>

## 5.3.2.1.5 Working with Profile Configurations

### Context

You want to save profile configurations, such as the name of an ABAP development object and ABAP Profiler settings, so that you can quickly and easily re-run the ABAP Profiler with predefined settings.

### Procedure

1. Start the configuration dialog from the editor or from the *Project Explorer* or by choosing  *Profile As* > *Profile Configurations*  ([**Ctrl**] + [**Shift**] + [**F9**]).
2. ABAP Development Tools presents the *Profile Configurations* dialog. In the *Project Explorer*, you can create a new run configuration or edit an existing run configuration.
3. On the **Main** tab, specify the ABAP development object that is to be traced and the ABAP project (system and client) in which the trace is to be made.
4. On the *Tracing* tab, you can set *Profiler* options.

See [Understanding ABAP Profiler Settings \[page 680\]](#) for a description of the *Profiler* options.

### Results

You have saved a configuration for the ABAP Profiler. To run the ABAP Profiler, choose **Profile** from the toolbar of the *Configurations* dialog.

## 5.3.2.2 Displaying the Trace Overview

Double-click on an entry in the *ABAP Traces* view to display the trace overview. The *Overview* is the starting point for further analysis and provides you with the following information:

### General Information

<b>Title</b>	The title given to the ABAP trace by the user or generated by the system.
<b>ID</b>	The name of the trace file, composed of the name of the instance - <code>host_sid_number</code> - on which the trace was made and the file name - <code>AT...</code>
<b>Date</b>	Date and time at which the trace was started.

<b>Server</b>	Server on which the trace was recorded.
<b>User</b>	User under which the trace was made. In scheduled traces, this may be a service user, a different user than scheduler of the trace.
<b>Aggregation Mode</b>	Indicates which aggregation mode (None, Call Stack or Call Position) has been used during the trace execution.
<b>Size</b>	Size of the trace file

## Runtime Distribution

Provides you with an information of how the traced runtime was distributed across Application Server ABAP components.

<b>Overall</b>	The total run time recorded in the trace.
<b>ABAP</b>	The run time spent processing ABAP code, not including ABAP system programs.
<b>Database</b>	The run time spent in the database, performing database operations.
<b>System</b>	The run time spent processing ABAP code in system programs. An ABAP program is a system program if it is defined with the status set to <code>System Program</code> .

## Accessing the Integrated Analysis Tools

More on this: [Using Analysis Tools \[page 650\]](#)

### 5.3.2.3 Using Analysis Tools

#### Integrated Analysis Tools

As an integral part of the ABAP Profiler, various analysis tools provide you with different views on the ABAP trace:

Analysis Tool	Allows you to ...
Condensed Hit List	... find top consumers with regard to procedures such as methods, function modules, or forms.
More on this: <a href="#">Finding Top Consumers Using the Condensed Hit List [page 651]</a>	

Analysis Tool	Allows you to ...
Hit List	<p>... find top consumers in your application by call position.</p> <p><b>More on this:</b> <a href="#">Finding Hot Spots with the Hit List [page 657]</a> and <a href="#">Determining Runtime with the Hit List [page 659]</a></p>
Aggregated Call Tree	<p>... analyze trace events that are aggregated by call stacks and displayed within a call tree.</p> <p><b>More on this:</b> <a href="#">Analyzing Trace Events in the Aggregated Call Tree [page 666]</a></p>
Call Sequence	<p>... analyze the flow of program execution.</p> <p><b>More on this:</b> <a href="#">Analyzing Program Flow with the Call Sequence [page 661]</a></p>
Call Timeline	<p>... visualize the call sequence in the form of a diagram.</p> <p><b>More on this:</b> <a href="#">Analyzing Call Sequence Using the Call Timeline Tool [page 671]</a></p>
Database Accesses	<p>... check which portion of runtime is used by database accesses and identify the top consumers among database accesses.</p> <p><b>More on this:</b> <a href="#">Understanding Parameters in the (Condensed) Hit List [page 654]</a></p>
SQL Trace	<p>... view SQL trace events and analyze how the system handles database requests.</p> <p><b>More on this:</b> <a href="#">Analyzing SQL Statements [page 692]</a></p>
Call Stack	<p>... analyze the call stack of trace events.</p> <p><b>More on this:</b> <a href="#">Displaying the Call Stack [page 674]</a></p>

### 5.3.2.3.1 Finding Top Consumers Using the Condensed Hit List

You can use the Condensed Hit List to find top consumers with regard to procedure calls, such as methods, function modules, subroutines, or other kind of calls (for example: CALL TRANSACTION, CALL SCREEN).

#### Prerequisites

- You have created an ABAP trace of your program. The ABAP trace has the status *Finished*.
- The ABAP trace is not aggregated by *Call Position*. **More on this:** [Aggregation Options in ABAP Traces \[page 682\]](#)

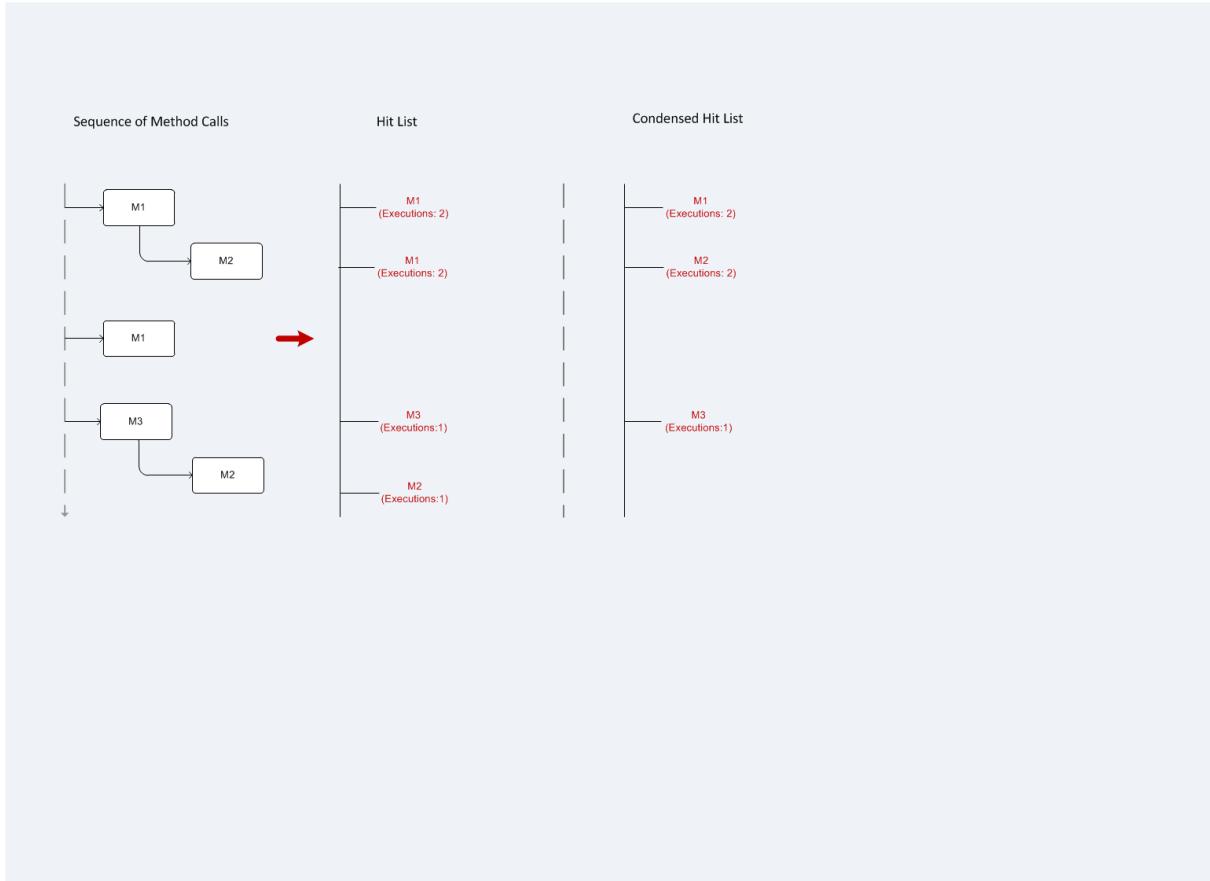
#### Context

Remember, in the *Hit List* tool, trace events are aggregated by their call position so that no stack information is available, whereas in the *Condensed Hit List*, aggregation is independent from the call position and depends only on the called unit. **See also:** [Aggregation Options in ABAP Traces \[page 682\]](#)

Aggregation means that the runtime used in those aggregated trace events that have been called in the same way, is added up to one single trace event entry. The *Executions* column displays how many trace events have been aggregated.

## EXAMPLE

Imagine a class with methods M1, M2, and M3. Now imagine a program PROG that calls M1 twice at the same call position. The first call of M1 implies also the call of method M2. Finally, M2 is called through the method M3, as shown in the figure below. In the *Condensed Hit List*, the ABAP Profiler creates a single trace entry for each method M1, M2, and M3. The *Executions* column has the value 2 for M1 and M2.



Trace events in the Hit List and in the Condensed Hit List

Use the *Condensed Hit List* to do the following:

- Analyze the runtime consumption of individual procedure calls in your ABAP application - how long does it take for a method, function module, or subroutine call to run?
- Find top consumers among the procedure calls.
- Analyze how the runtime consumption of an individual procedure call is distributed to intrinsic (unspecified) ABAP statements, to external calls, and to specified ABAP statements (for example: SELECT statements that require database calls).

Here is how to use the *Module Hit List* to evaluate potential top consumers under the procedure calls in an ABAP program.

## Procedure

1. To open the *Condensed Hit List*, select the relevant trace in the  *ABAP Traces* view and choose *Show Condensed Hit List* from the context menu.

### Note

Alternatively, double-click the trace in the *ABAP Traces* view to open the *Overview* page of the trace and then choose the *Condensed Hit List* tab.

The *Condensed Hit List* provides you with a list of individual procedure calls within an ABAP trace. Each trace event is aggregated to a method, function module, subroutine, or other kind of procedure call.

2. Identify those trace events (procedure calls) that cost disproportionate amounts of runtime and perhaps could be optimized. You can pay special attention to the *Executions*, *Own Time* (...), and *Total Time* columns. **See also:** [Understanding Parameters in the \(Condensed\) Hit List \[page 654\]](#)

Condensed Hit List							
Called Unit	Exec...	Stacks	C...	Own Tim...	Own Ti...	Total Tim...	%
Method CL_WB_REGISTRY->GET_ACCESS_TOOL	528	9	1	33,915	33,915	33,915	
Method CL_GUI_SPLITTER_CONTAINER->SET_ROW_HEIC	26	8	7	31,706	31,706	31,706	
Form GET_REQUESTED_TABLE_PROPS	5	3	1	19,880	7,468	20,609	
Method CL_GUI.Docking_CONTAINER->SET_VISIBLE	9	2	1	18,820	18,820	18,820	
Method CL_GUI_OBJECT->SET_PROPERTY	24	24	15	18,743	18,743	18,743	
Function BDS_CONNECTION_GET	5	3	1	18,311	5,316	21,033	
Form GET_SPECIAL_FROM_RELATIONS	5	3	1	16,314	5,005	17,502	
Method CL_TREE_CONTROL_BASE->CONSTRUCTOR	1	1	1	15,033	15,033	15,033	
Function DP_CREATE_URL	7	5	2	14,804	14,804	14,804	
Function DP_SEND_TABLE	2	2	1	14,338	14,338	14,696	
Form PHIO_EXISTENCE_CHECK	3	1	1	14,088	4,450	15,950	
Method CL_WB_LAYOUT_MANAGER->GET_DEFAULT_TC	36	6	2	12,827	6,079	12,827	

Trace Parameters in Condensed Hit List

Here you can focus your view on specific aspects.

3. To drill down to the relevant data, select the called unit of your interest, open the context menu, and select the desired option:

Choose ...

Option	Description
<a href="#">Show Called Unit in Source Code</a>	... to navigate to the source code position where the procedure is implemented.
<a href="#">Show All Executions</a>	... to display all executions of the selected trace event (procedure call). The results are displayed in the Search view and allow you to detect the runtime portion for each individual execution of the trace event.
 Note	This option is only enabled for aggregated trace events (number of <i>Executions</i> > 1)
<a href="#">Show in Call Sequence</a>	... to analyze the execution path of the procedure call.

Option	Description
	<p><b>i Note</b></p> <p>This option is only enabled for non-aggregated trace events (number of <i>Executions</i> = 1). The runtimes that you see for the trace event reflect a single execution of the procedure call that has been recorded in the trace entry.</p>
<a href="#">Show Call Stack</a>	... to analyze the distribution of executions over the different call stacks. This view on data allows you to follow from where the current trace event is called.
<a href="#">Show in Aggregated Call Tree</a>	... to analyze aggregated data in the call tree.
<a href="#">Show in Hit List</a>	... to analyze in detail the different call positions of the called unit.
<a href="#">System Events</a>	... if you want that system events (for example: methods of system programs) are displayed.

### 5.3.2.3.1.1 Understanding Parameters in the (Condensed) Hit List

Here is some information on the trace parameters that are displayed in the **Condensed Hit List** and the **Hit List**.

Parameter	Description
<i>Trace Event</i> (in Hit List) or <i>Called Unit</i> (in Condensed Hit List)	A trace event is an ABAP statement or other operation (a database operation, for example) for which the ABAP Profiler records discrete performance data. Each individual trace event that has been recorded allows you to analyze the runtime and memory consumption of the underlying ABAP statement or other operation.
	<p><b>i Note</b></p> <p>In the <i>Condensed Hit List</i>, a called unit in particular represents procedure calls such as methods, function modules, or subroutines.</p>
	<p>The ABAP Trace does not record all ABAP statements and operations as trace events, but only those that are potentially "expensive" in terms of system resources. A <i>CALL METHOD</i> statement is recorded as a trace event. An <i>IF</i> statement is not.</p> <p>Though the ABAP Profiler does not record all ABAP statements and events, a trace nevertheless gives you a rich and detailed view of program performance and execution flow, especially since all procedure call statements are recorded.</p> <p>The trace events that you see in a particular ABAP trace also depend on the Profiler settings that you selected before creating the ABAP trace. <b>More on this:</b> <a href="#">Understanding ABAP Profiler Settings [page 680]</a></p>
	<p><b>Aggregation of Trace Events</b></p> <p>Aggregation means that the runtime used in the aggregated trace events that have been called in the same manner is added up to a single total trace event entry. The <i>Executions</i> column is set to show how many trace events were aggregated. You can therefore calculate average values for single instances of the aggregated event.</p> <p><b>Example:</b> Imagine a class with methods M1 and M2. Now imagine a <i>PROGRAM</i> that calls M1 once and M2 twice:</p> <ul style="list-style-type: none"> <li>For M1, the ABAP trace creates a single trace entry. The <i>Executions</i> column has the value 1.</li> <li>Method M2 occurs as one entry in the <i>Condensed Hit List</i> with <i>Executions</i> = 2. If the call position of the first call of M2 is the same as for the second call, there will also be one entry in the <i>Hit List</i> with <i>Executions</i> = 2. However, if both call positions are different, there will be two entries in the <i>Hit List</i>, both with <i>Executions</i> = 1.</li> </ul>
<i>Executions</i>	<p>The number of executions of a trace event indicates whether this trace event shows aggregated runtimes or not. If you see the value 1 in <i>Executions</i>, the trace event is not aggregated. The runtimes that you see then reflect a single execution of the statement or operation recorded in the trace entry.</p> <p>If you see a <i>Executions</i> of greater than 1, that number of runs of the traced ABAP statement or operation are aggregated in the runtimes that you see. <i>Executions</i> 3 with an <i>Own Time</i> of nine seconds means that each instance of the trace event ran for an average of three seconds.</p>

Parameter	Description
<i>Callers</i>	<p>Number of direct callers of a trace event. Using this parameter you can detect how many different call positions contribute to the runtime of the called unit. The number of callers of a called unit in the <i>Condensed Hit List</i> tells you how many different entries of the <i>Hit List</i> have been aggregated to this entry.</p> <p><b>i Note</b></p> <p>This parameter is available in the <i>Condensed Hit List</i>, but not in the <i>Hit List</i>.</p>
<i>Stacks</i>	<p>Number of different call stacks associated with the trace event. The number tells you how many entries of the <i>Aggregated Call Tree</i> have been aggregated to this entry.</p>
<i>Own Time (Specified and Unspecified ABAP Statements) [μs]</i>	<p>The amount of runtime in microseconds spent running a particular trace event or set of aggregated trace events, including specified ABAP statements, such as SELECT statements that require database calls.</p> <p><b>i Note</b></p> <p>This parameter is available in the <i>Condensed Hit List</i>, but not in the <i>Hit List</i>.</p> <p>This runtime includes intrinsic (unspecified) statements and specified statements, but excludes all time spent in external calls that were started from the present trace event.</p> <p><b>Example:</b> Imagine that method M1 has a total runtime of six seconds. One tenth of a second is accounted for by creating an object that offers method M2 (specified statement). Nine tenths of a second are spent doing other intrinsic processing in M1 (unspecified statements). Five seconds more are spent in method M2 called from M1.</p> <p>In the <i>Condensed Hit List</i>, the <i>Own Time (Specified and Unspecified ABAP Statements)</i> of method M1 is one second (one tenth + nine tenths of a second) because the rest of the runtime of six seconds was spent in method M2 (five seconds).</p>
<i>Own Time [μs]</i>	<p>The amount of runtime in microseconds spent running a particular trace event or set of aggregated trace events. The <i>Own Time</i> excludes all time spent in other trace events that were started from the present trace event, such as external calls or specified statements (for example: SELECT statements).</p> <p><b>Example:</b> Imagine that method M1 has a total runtime of six seconds. One tenth of a second is accounted for by creating an object that offers method M2 (specified statement). Nine tenths of a second are spent doing other processing in M1 (unspecified statements). Five seconds more are spent waiting for calls from M1 to method M2 to complete.</p> <p>In the <i>Condensed Hit List</i> and in the <i>Hit List</i>, the <i>Own Time</i> of method M1 is nine tenths of a second because the rest of the runtime of six seconds was accounted for by creating an object (100 milliseconds) and waiting for method M2 (five seconds). Each of these events has its own trace entry. Their runtimes are therefore excluded from the <i>Own Time</i> of M1.</p>

Parameter	Description
<i>Total Time [μs]</i>	<p>The amount of runtime in microseconds from the start to the finish of a particular trace event or set of aggregated trace events. The <i>Total Time</i> includes time spent in other trace events started from the first trace event. In other words: it includes the runtime of external calls as well as unspecified (intrinsic) and specified statements (for example: SELECT statement that requires database calls) within the present trace event. <i>Total Time</i> lets you see the total contribution of the processing done within such tracing events as procedure calls.</p> <p><b>Example:</b> A method M1 includes unspecified statements, calls a second method M2, and also calls the database using SELECT statements. Three seconds runtime are spent for method M2 to complete, one second is spent for performing database calls, and two seconds are spent for the intrinsic, unspecified statements.</p> <p>The <i>Total Time</i> of method M1 is six seconds, including the runtime of method M2 and database calls.</p>
<i>% Own Time (Specified and Unspecified ABAP Statements)</i>	<p>This column shows the <i>Own Time (Specified and Unspecified ABAP Statements)</i> of the trace event as a percentage of total ABAP trace runtime.</p> <p>This column (as well as <i>% Own Time</i> and <i>% Total Time</i>) let you evaluate the importance of a particular trace event in the total runtime of a measured application. A trace event can have a long runtime. But how much of the own (total) run time is accounted for by the trace event? For example, an event with an <i>Own Time</i> contribution to a runtime of a couple of percent does not merit the refactoring attention that you would pay to an event with a 33% share of runtime.</p>
<i>% Own Time</i>	This column shows the <i>Own Time</i> of the trace event as a percentage of the total trace runtime.
<i>% Total Time</i>	This column shows the <i>Total Time</i> of the trace event as a percentage of the total trace runtime.
<i>Calling Program</i>	<i>Calling Program</i> identifies the ABAP program that initiated the trace event.
<i>Called Program</i>	<p><i>Called Program</i> identifies the ABAP program in which the trace event is located.</p> <p>These two parameters <i>Calling Program</i> and <i>Called Program</i> let you differentiate between code in your own application and the infrastructure code that originated in the ABAP Application Server.</p>
<p><b>→ Tip</b></p> <p>Use the filter field to show, for example, only trace events that belong to your own application.</p>	

### 5.3.2.3.2 Finding Hot Spots with the Hit List

#### Prerequisites

You have created an ABAP trace of your ABAP program.

## Context

Use the [Hit List](#) to do the following:

- Analyze the runtime of your ABAP application - how long does it take to run?
- Find performance hot spots – operations that cost disproportionate amounts of runtime and which perhaps could be optimized.

The [Hit List](#) aggregates trace events (potentially costly ABAP statements and events). This means that you can easily see runtime accounted for by repetitive events in your program, such as repeated calls to a particular method. And you can evaluate the impact of aggregated operations on the total runtime of the application.

Here is how to use the [Hit List](#) to evaluate potential hot spots in an ABAP program.

## Procedure

Open the [Hit List](#) by selecting a trace in the **ABAP Traces** view and choosing [Show Hit List](#) from the context menu.

The [Hit List](#) opens in a new view of its own. It starts with the list of trace events sorted by [Own Time](#). That is the run time in microseconds that was spent in the trace event itself, excluding other trace events, such as procedure calls, made during the trace event. A method may have a large total time but a small own time if it does nothing but call another method.

### Example

Here's how to read this excerpt from a hitlist, sorted by [Own Time](#). To evaluate the importance of the top entry in the [Hit List](#), `SELECT SINGLE TLDOC_ITFPARAMCONV`. This trace event accounts for fully 25 percent of the runtime of the traced program – that qualifies it as a 'hot spot' – a target for potential optimization. We'll pay special attention to the [Own Time](#) and [Executions](#) columns. Here is [Understanding Parameters in the \(Condensed\) Hit List \[page 654\]](#) for understanding all of the columns.

Here's the share of the event in total runtime							
Trace Event	Own ...	Total Time ...	Own Tim...	Total Tim...	Hits	Calling Program	Called Program
Select Single TLDOC_ITFPARAMCONV	36,955	36,955	25	25	31	CL_DOCU_ADT_ITF_HTML_CONVERTERCP	CL_DOCU_ADT_ITF
Fetch ICFSERVICE	18,954	18,954	13	13	31	SAPLHTTPTREE	SAPLHTTPTREE
Fetch ICFSERVICE	14,625	14,625	10	10	1	SAPLHTTPTREE	SAPLHTTPTREE

- Here's what happened...
- Here's the share of the event in total runtime
- Here's how much time it cost. Important is own time - time spent in the event itself and not in other called events.
- The [Trace Event](#) column shows what ABAP operation was recorded. Here, the topmost, most expensive trace event – `Select Single TLDOC_ITFPARAMCONV` – belongs directly to our own program, the program that we want to analyze. Since the trace event belongs to our code, we can potentially optimize it. The next two lines – `Fetch ICFSERVICE` – pertain to the ABAP HTTP service infrastructure. They are not directly part of our program, and we can ignore these trace events. The [Called Program](#) and [Calling Program](#) columns help you to recognize events that belong to your code and non-relevant trace events that belong to other frameworks or infrastructures.
  - [Own Time](#) shows the accumulated time spent doing the traced operation. [Own Time](#) is important because it is the time spent doing the operation by itself, without counting any time the operation may have spent waiting for other traced operations that it called. Here, our program spent 37 milliseconds

reading single entries from a database table called `TLDOC_ITFPARAConv`. The *Hits* column shows how many individual trace events are aggregated in this trace entry. In our example, there were 31 `SELECT SINGLE` operations on `TLDOC_ITFPARAConv`.

- *Own Time (%)* lets you assess the impact of this aggregated operation on the total traced runtime. 37 milliseconds is not a lot of runtime, but, on the other hand, the `SELECT SINGLE` operations account for 25% of the total runtime of the traced program. So optimizing this database operation could have a big impact on the performance of the application as a whole.

## Results

Armed with this information, you can now decide whether this top entry in the *Hit List* deserves optimizing attention or not. On the one hand, 37 milliseconds is not a lot of time. On the other hand, it is a full quarter of the total runtime of the traced program.

Since the operation is a database operation, there are several low-cost optimization strategies available.

- We could check whether caching is switched on for `TLDOC_ITFPARAConv`, and if so, whether the caching strategy is correct for the table and its use.
- Are the same records being fetched over and over again? If so, then perhaps the program could fetch each record only once.
- Are all of the fields of the table records being used? Perhaps only a couple of fields need to be fetched.
- If the table is small and most or all of the records in the table are being read, then perhaps the program would be more efficient if it did a single `SELECT` of the table contents into an internal table.

Should you wish to take a closer look at the code or set a break point in the code, just double-click on an entry. The ABAP editor opens in a separate window, positioned where the aggregated trace event occurred.

### → Tip

You can also use these context menu functions to navigate from *Hit List* entries:

- To find position of a trace event in the *Aggregated Call Tree*  
See also: [Analyzing Trace Events in the Aggregated Call Tree \[page 666\]](#)
- To show a trace event in the *Call Sequence* tool  
See also: [Analyzing Program Flow with the Call Sequence \[page 661\]](#)
- To show the call stack that led to the trace event  
See also: [Displaying the Call Stack \[page 674\]](#)

### 5.3.2.3.3 Determining Runtime with the Hit List

#### Prerequisites

You have created an ABAP trace of your ABAP program. The ABAP trace has the state *Finished*.

## Context

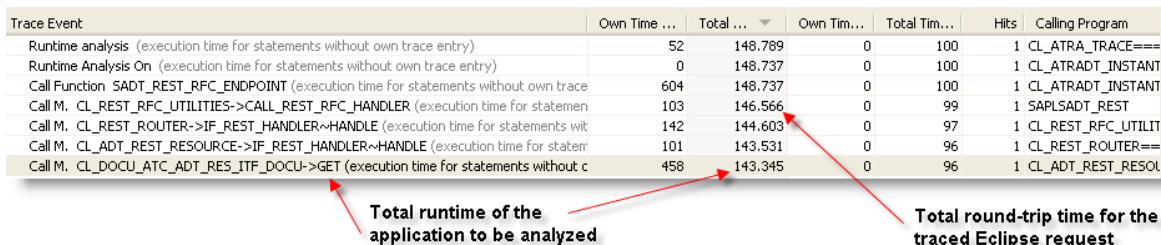
Here is how to find out the total runtime of a traced program.

## Procedure

1. Open the **Hit List**, for example, by selecting a trace in the **ABAP Traces** view and selecting **Show Hit List** from the context menu.  
The *Hit List* opens in a new view of its own.
2. The *Hit List* starts with the list of trace events sorted by *Own Time*. Re-sort the list by clicking on the **Total Time [μs]** column. This sorts the trace events by total runtime spent in a particular trace event.

Total runtime includes time spent waiting while other trace events were called and processed. For that reason, *Own Time* (time spent processing the trace event itself) is often less than *Total Time*.

The display looks something like this:



Trace Event	Own Time ...	Total ...	Own Tim...	Total Tim...	Hits	Calling Program
Runtime analysis (execution time for statements without own trace entry)	52	148.789	0	100	1	CL_ATRA_TRACE==
Runtime Analysis On (execution time for statements without own trace entry)	0	148.737	0	100	1	CL_ATRADT_INSTANT
Call Function SADT_REST RFC_ENDPOINT (execution time for statements without own trace)	604	148.737	0	100	1	CL_ATRADT_INSTANT
Call M. CL_REST RFC UTILITIES->CALL REST RFC HANDLER (execution time for statements without own trace entry)	103	146.566	0	99	1	SAPLSADT_REST
Call M. CL_REST ROUTER->IF REST HANDLER->HANDLE (execution time for statements without own trace entry)	142	144.603	0	97	1	CL_REST RFC UTILIT
Call M. CL_ADT REST RESOURCE->IF REST HANDLER->HANDLE (execution time for statements without own trace entry)	101	143.531	0	96	1	CL_REST_ROUTER==
Call M. CL_DOCU_ATC_ADT_RES_ITF_DOCU->GET (execution time for statements without own trace entry)	458	143.345	0	96	1	CL_ADT_REST_RESOL

**Total runtime of the application to be analyzed**      **Total round-trip time for the traced Eclipse request**

3. Search for the start of the trace records of your program.

As you can see in the screen shot above, the first six trace entries have to do with the ABAP Trace itself or with the communications infrastructure of the ADT. The third line, for example, shows that the total round-trip time of the ADT request was around 149 milliseconds. The total run time of our traced application (`CL_DOCU_ATC_ADT_RES_ITF_DOCU->GET`) appears in the last line shown in the screen shot. As you can see, the run time of the `GET` request was about 143 milliseconds.

(By the way, the ABAP Trace uses kernel technology and is very fast. A traced application may slow down if the ABAP Trace needs to write a lot of trace records, but even in this case, the traced runtimes reflect only the activity of the traced program. The ABAP Trace automatically excludes its own runtime for making measurements and writing trace entries from the runtimes of the traced program and not the tracing).

Usually, you must read down the *Hit List* to find the start of the runtime that pertains to your code; the first entries pertain to the ADT or the communication infrastructure. You can identify the start of your code by looking at the *Called Program* column. You can also use the *Filter* column to restrict the display to your own code or to find particular procedures or other trace events.

4. Evaluate the run time.

Is the total runtime acceptable? In this case, the runtime of 143 milliseconds for fetching and displaying a document is probably acceptable. The response time for the user is likely to be far under 0,5 seconds. If the `GET` method had shown a higher runtime, close to 500 milliseconds, for example, then that would have been good cause to start analyzing the code to find out where the runtime is being consumed. A major refactoring and optimization might be required.

It is useful to repeat your trace on different systems and on production systems under more realistic load conditions than you would find on a development or integration system. Screening your programs on a development or integration system remains, however, a good idea. An unacceptable runtime on your development system provides an early signal that optimization is needed.

#### 5. Analyze the code.

Should you need to start a closer analysis of the code or set a break point in the code, just double-click on an entry in the *Hit List*. The ABAP editor opens in a separate view, positioned where the trace event occurred. If the trace event was aggregated, then the *Search* view opens to let you open the editor from one of the aggregated instances of the trace event.

You can also use these context menu functions to navigate from hit list entries:

- **Show in Call Sequence:** Show a trace event in position in the [Analyzing Program Flow with the Call Sequence \[page 661\]](#).
- **Show Call Stack:** [Displaying the Call Stack \[page 674\]](#) that led to the trace event.

How much time does my program need to run? Does it run as quickly as I expect it to? Is the runtime acceptable? You can answer these important questions with the **Hit List**.

### 5.3.2.3.4 Analyzing Program Flow with the Call Sequence

#### Prerequisites

- You have created an ABAP trace of your ABAP program. The ABAP trace has the state *Finished*.
- The trace in question has aggregation set to *No*, as in this sample from the *ABAP Trace Requests* view.

Time	Date	Title	Traced Object	Aggregated
13:15:33	18.07.2011	DEFAULT	SE80	No

**From the ABAP Traces view...** **Full trace with aggregation off** 

If a trace is aggregated, then the trace does not record individual trace events. The *Call Sequence* function is not offered in the context menu if a trace was recorded with aggregation active.

#### Context

Use the *Call Sequence* view to analyze the execution path of a program. You can use the *Call Sequence* to do the following, for example:

- Analyze the flow of a program as part of the analysis of an error. You can see which ABAP programs are involved in the error and easily find useful places to set breakpoints.
- Understand how an unfamiliar program works and which ABAP programs play a role in its execution.

Here is how to use the *Call Sequence* to analyze the flow of program execution.

## Procedure

1. Open the **Call Sequence** by selecting a trace in the **ABAP Traces** view and choosing *Show in Call Sequence* from the context menu or from the *Overview* page.

The *Call Sequence* opens in a new view of its own. It starts with a display of the top-most branches of the call hierarchy. The main runtime consumers are highlighted with boldface. The **Show/Hide** toggle link lets you display or hide non-procedural trace events in each procedure, such as database activity or use of internal tables.

Trace Event	Stat...	Total Time ...	Own Time ...
<b>Runtime analysis</b>		<b>6.593.156</b>	<b>810</b>
All statements within Runtime analysis		810	
<b>Runtime Analysis On</b>	<a href="#">Show</a>	<b>6.592.346</b>	<b>132.024</b>
All statements within Runtime Analysis On		675.058	
Call M. CL_TREE_MODEL->CHECK_IF_VIEW_IS_UP_TO_DATE		12	12
Call M. CL_TREE_MODEL->GET_LONG_KEY_BY_VIEW_KEY	<a href="#">Show</a>	196	196
All statements within Call M. CL_TREE_MODEL->GET_LONG_KEY_BY_VIEW_KEY		196	
Call M. LCL_LOCAL_CTXMNUPROXY->POST_SHELL_CTXMNU		11.818	334
<b>Module PAI_USER_COMMAND_0100</b>		<b>5.884.994</b>	<b>75</b>
All statements within Module PAI_USER_COMMAND_0100		75	
<b>Perform FCODE_PROCESS</b>		<b>5.884.919</b>	<b>206</b>

**Main run time consumers**      **Show non-procedural trace events in this method**      **Total run time in method / Run time directly in method**

2. Drill down to the section of the trace that particularly interests you. There you can see which programs participated in processing and analyze the flow of processing at the level of procedure calls.
3. Use the information resources of the *Call Sequence* to understand the program flow and analyze the runtime behavior of the program.

In this sample, we want to understand the most expensive call to the method

`GET_DOCUMENT_FROM_DOCU_SYSTEM`, which had a runtime of around 17 milliseconds (*Total Time*). In this first view, the structure of the method is clear:

- It calls a function module from `SAPLSDOC` to fetch a document.
- It then calls a series of methods that perform operations on the document, such as resolving text symbols.

<code>Call M. CL_DOCU_ADT_ITF_DOCU_READER-&gt;GET_DOCUMENT_FROM_DOCU_SYSTEM</code>	<a href="#">Show</a>	17.324	59	<code>CL_DOCU_ADT_ITF_DOCU_READER==CP</code>	<code>CL_DOCU_ADT_ITF_DOCU_READER==</code>
All statements within <code>Call M. CL_DOCU_ADT_ITF_DOCU_READER-&gt;GET_DOCUMENT_FROM_DOCU_SYSTEM</code>		59		<code>CL_DOCU_ADT_ITF_DOCU_READER==CP</code>	
<b>Call Function DOCU_GET</b>	<a href="#">Show</a>	4.272	3.907	<code>CL_DOCU_ADT_ITF_DOCU_READER====CP</code>	<code>SAPLSDOC</code>
Call M. CL_DOCU_ADT_ITF_DOCU_READER->RESOLVE_INCLUDES		3.572	5	<code>CL_DOCU_ADT_ITF_DOCU_READER====CP</code>	<code>CL_DOCU_ADT_ITF_DOCU_READER</code>
Call M. CL_DOCU_ADT_ITF_DOCU_READER->DELETE_UNUSED_DEFAULT_SECTION		3.380	5	<code>CL_DOCU_ADT_ITF_DOCU_READER====CP</code>	<code>CL_DOCU_ADT_ITF_DOCU_READER</code>
Call M. CL_DOCU_ADT_ITF_DOCU_READER->RESOLVE_TEXT_SYMBOLS		5.857	43	<code>CL_DOCU_ADT_ITF_DOCU_READER====CP</code>	<code>CL_DOCU_ADT_ITF_DOCU_READER</code>
Call M. CL_DOCU_ADT_ITF_DOCU_READER->REPLACE_SAPSCRIPT_CONTROLS		184	1	<code>CL_DOCU_ADT_ITF_DOCU_READER====CP</code>	<code>CL_DOCU_ADT_ITF_DOCU_READER</code>

**Method GET\_DOCUMENT\_FROM\_DOCU\_SYSTEM consists of this sequence of procedure calls.**      **Total and net runtimes of the procedures**      **GET\_DOCU is called from function group SDOC**

As you take a closer look at the trace excerpt, there are some interesting features to look at. First, the `DOCU_GET` function module uses almost as much 'own' (net) runtime as total runtime. What is it doing? Click **Show** to see what used the time in `DOCU_GET`. In fact, the Own runtime is accounted for by database operations on the three ABAP documentation tables.

Call Function DOCU_GET	Hide	4.272	3.907
Unspecified statements within Call Function DOCU_GET		439	
Call Function DOCU_INIT	Show	305	305
Select Single DOKHL		118	118
Open Cursor DOKTL		2.364	2.364
Fetch DOKTL	<b>DOCU_GET spends its time fetching a document from these tables</b>		959
Read Table IT_29		5	5

Then, the calls to the native methods of class CL\_DOCU\_ADT\_ITF\_DOCU\_READER have very low *Own Time* values. What are these methods calling? Expanding the methods shows that they call function modules from various function groups. A check of these function groups shows that they all belong to the ABAP documentation system. So, class CL\_DOCU\_ADT\_ITF\_DOCU\_READER essentially encapsulates an older function-module-based API.

Call M. CL_DOCU_ADT_ITF_DOCU_READER->RESOLVE_INCLUDES	3.572	5 CL_DOCU_ADT_ITF_DOCU_READER==CP	CL_DOCU_A
All statements within Call M. CL_DOCU_ADT_ITF_DOCU_READER->RESOLVE_INCLUDES	5	CL_DOCU_ADT_ITF_DOCU_READER==CP	
⊕ Call Function TEXT_INCLUDE_REPLACE	3.567	93 CL_DOCU_ADT_ITF_DOCU_READER==CP	SAPLSTXE
Call M. CL_DOCU_ADT_ITF_DOCU_READER->DELETE_UNUSED_DEFAULT_SECTIONS	3.380	5 CL_DOCU_ADT_ITF_DOCU_READER==CP	CL_DOCU_A
All statements within Call M. CL_DOCU_ADT_ITF_DOCU_READER->DELETE_UNUSED_DEFAULT_SECTIONS	5	CL_DOCU_ADT_ITF_DOCU_READER==CP	
⊕ Call Function DOCU_UNUSED_TEMPLATE_DELETE	3.375	42 CL_DOCU_ADT_ITF_DOCU_READER==CP	SAPLSDCA
Call M. CL_DOCU_ADT_ITF_DOCU_READER->RESOLVE_TEXT_SYMBOLS	5.857	43 CL_DOCU_ADT_ITF_DOCU_READER==CP	CL_DOCU_A
All statements within Call M. CL_DOCU_ADT_ITF_DOCU_READER->RESOLVE_TEXT_SYMBOLS	8	CL_DOCU_ADT_ITF_DOCU_READER==CP	
⊕ Call Function INIT_TEXTSYMBOL	2.497	69 CL_DOCU_ADT_ITF_DOCU_READER==CP	SAPLSTXV
⊕ Loop At IT_31	35	CL_DOCU_ADT_ITF_DOCU_READER==CP	CL_DOCU_A
⊕ Call Function TEXT_SYMBOL_REPLACE	3.317	1 CL_DOCU_ADT_ITF_DOCU_READER==CP	SAPLSTXE

Class CL\_DOCU\_ADT\_ITF\_DOCU\_READER encapsulates functions borrowed from the older function-module-based APIs of the ABAP online documentation system

## Results

We now understand basically how the class works and what runtime resources it consumes.

### → Tip

- Should you wish to take a closer look at the code or set a break point in the code, just double-click on an entry. The ABAP editor opens in a separate window, positioned where the aggregated trace event occurred.
- To take a look at the aggregated impact of the trace event on runtime, choose **Show in Hit List** from the context menu of an entry. You can see how the aggregated trace event ranks among the runtime consumers in the trace.
- To [Displaying the Call Stack \[page 674\]](#) that lead to a trace event, choose **Show Call Stack** in the context menu.

## Related Information

[Finding Hot Spots with the Hit List \[page 657\]](#)

[Determining Runtime with the Hit List \[page 659\]](#)

[Understanding Parameters in the \(Condensed\) Hit List \[page 654\]](#)

## 5.3.2.3.4.1 Understanding Parameters in Aggregated Call Tree and Call Sequence

Here you will find information on the trace parameters that are displayed in the **Aggregated Call Tree** and the **Call Sequence**:

### i Note

With one exception, the parameters are identical in the *Aggregated Call Tree* and *Call Sequence* views:

- The *Executions* column appears only in the *Aggregated Call Tree*.

Parameter	Description
Trace Event	<p>A trace event is an ABAP statement or other operation (a database operation, for example) for which the <i>ABAP Profiler</i> records discrete performance data. Each individual trace event that has been recorded allows you to analyze the runtime consumption of the underlying ABAP statement or other operation.</p> <p>The ABAP Trace does not record all ABAP statements and operations as trace events, but only those that are potentially expensive in terms of resources. A <i>CALL METHOD</i> statement is recorded as a trace event. An <i>IF</i> statement is not.</p> <p>Though the <i>ABAP Profiler</i> does not record all ABAP statements and events, a trace nevertheless gives you a rich and detailed view of program performance and execution flow, especially since all procedure call statements are recorded.</p> <p>The trace events you see in a particular ABAP trace also depend on the Profiler settings that you selected before creating the ABAP trace. <b>More on this:</b> <a href="#">Understanding ABAP Profiler Settings [page 680]</a></p>
	<p><b>Aggregation Options</b></p> <p>In the <i>Aggregated Call Tree</i>, the trace events are always aggregated for display according to their call stacks. The <i>Aggregated Call Tree</i> tool is available for non-aggregated trace files and for trace files that are aggregated by <i>Call Stack</i>. The <i>Call Sequence</i> tool is only available for non-aggregated trace files <b>More on this:</b> <a href="#">Aggregation Options in ABAP Traces [page 682]</a>.</p>
Statement Filter	<p>The <i>Aggregated Call Tree</i> and the <i>Call Sequence</i> view filter the trace display to highlight procedure calls such as <i>CALL METHOD</i> and <i>CALL FUNCTION</i>. Clicking on <i>Show</i> link in this field displays specified statements within the procedure call, such as data object creation, database calls, or use of an internal table, that were also recorded within the current procedure. These trace events are shown in contrasting colors.</p>
	<p><b>→ Remember</b></p> <p>The <i>Hit List</i> view for example, shows by all trace events, including the unspecified statements that the <i>Aggregated Call Tree</i> and the <i>Call Sequence</i> hide by default.</p>

Parameter	Description
<i>Executions</i>	<p>The number of executions of a trace event indicates whether this trace event shows aggregated runtimes or not. If you see the value <i>1</i> in <i>Executions</i>, then the trace event is not aggregated. The runtimes that you see then reflect a single execution of the statement or operation recorded in the trace entry.</p> <p>If you see a <i>Executions</i> of greater than <i>1</i>, then that number of runs of the traced ABAP statement or operation are aggregated in the runtimes that you see. <i>Executions</i> with an <i>Own Time</i>, for example, of nine seconds means that each instance of the trace event ran for an average of three seconds.</p>
	<p><b>i Note</b></p> <p>This column is available in <i>Aggregated Call Tree</i> only! In the <i>Call Sequence</i> view, the <i>Executions</i> counts the value <i>1</i> for each trace event recorded).</p>
<i>Total Time [μs]</i>	<p>This represents the amount of runtime in microseconds from the start to the finish of a particular trace event or set of aggregated trace events. The <i>Total Time</i> includes time spent waiting for other trace events started from the first trace event. In other words: it includes the runtime of external calls, as well as unspecified (intrinsic) and specified statements (for example: a SELECT statement that requires database calls) within the present trace event. <i>Total Time</i> lets you see the total contribution of the processing done within such tracing events.</p> <p><b>Example:</b></p> <p>A method M1 has a total runtime of six seconds. Five seconds of that time are spent waiting for method M2 to complete. Method M2 is called from method M1. The <i>Total Time</i> of method M1 is six seconds, including the run time of method M2.</p>
<i>Own Time [μs]</i>	<p>This represents the amount of runtime in microseconds spent running a particular trace event or set of aggregated trace events. The <i>Own Time</i> excludes all time spent waiting for other trace events that were started from the present trace event, such as external calls or specified statements (for example: SELECT statements). <i>Own Time</i> lets you see how intrinsically expensive a traced statement or operation was, in and of itself.</p> <p><b>Example:</b> Imagine that method M1 has a total runtime of six seconds. One tenth of a second is accounted for by creating an object that offers method M2 (specified statement). Nine tenths of a second is spent doing other processing in M1 (unspecified statements). Five seconds more are spent waiting for calls from M1 to method M2 to complete.</p> <p>In the <i>Aggregated Call Tree</i> and in the <i>Call Sequence</i>, the <i>Own Time</i> of method M1 is nine tenths of a second because the rest of the runtime of six seconds was accounted for creating an object (100 milliseconds) and waiting for by method M2 (five seconds). Each of these events has its own trace entry. Their runtimes are therefore excluded from the <i>Own Time</i> of M1.</p>
<i>Call Level</i>	<p>This parameter counts the depth within the call hierarchy, starting from entry point of the call tree (<i>Call Level</i> = 0) downwards to the level of the selected trace event.</p>
<i>% Total Time</i>	<p>This column shows the <i>Total Time</i> of the trace event as a percentage of the total trace runtime.</p>
<i>% Own Time</i>	<p>This column shows the <i>Own Time</i> of the trace event as a percentage of the total trace runtime.</p>

Parameter	Description
Calling Program	<i>Calling Program</i> identifies the ABAP program that initiated the trace event.
Called Program	<i>Called Program</i> identifies the ABAP program in which the trace event is located.
These two parameters <i>Calling Program</i> and <i>Called Program</i> let you differentiate between code in your own application and the infrastructure code that originated in the ABAP Application Server.	
<p>→ <b>Tip</b></p> <p>Use the filter field, for example, to show only trace events that belong to your own application.</p>	

### 5.3.2.3.5 Analyzing Trace Events in the Aggregated Call Tree

You can use the [Aggregated Call Tree](#) to analyze trace events that are aggregated to call stacks and displayed within a call tree.

#### Prerequisites

- You have created an ABAP trace of your program. The ABAP trace has the status *Finished*.
- The ABAP trace is not aggregated by **Call Position**. **More on this:** [Aggregation Options in ABAP Traces](#) [page 682]

#### Context

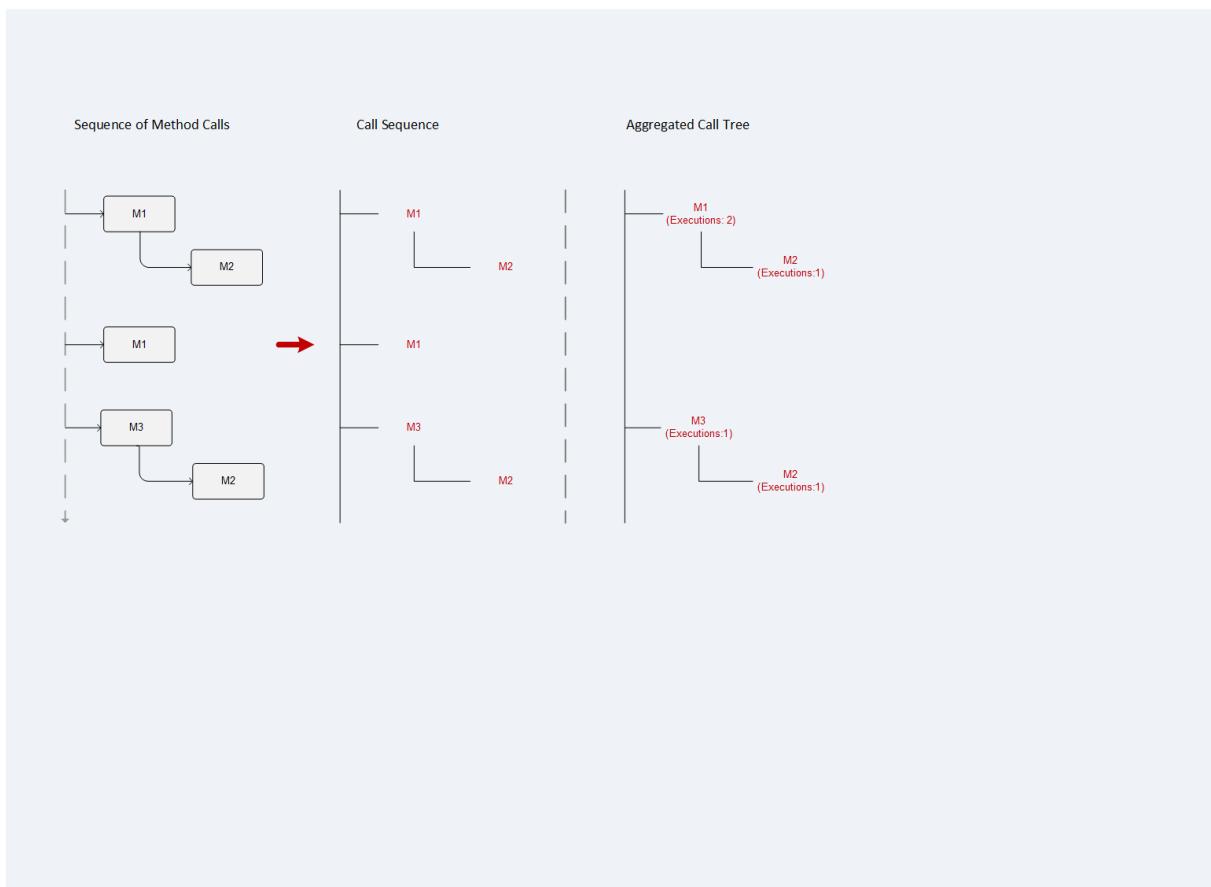
Remember, the Call Sequence tool provides you with a time sequence of trace events as a hierarchical tree. By contrast, the Aggregated Call Tree shows the trace events in an aggregated form. In the latter case, the trace events are aggregated By Call Stack and displayed in a tree hierarchy.

Aggregation **By Call Stack** means that the runtime data of the multiple executions that occurred via the same call path (the same caller) is added up to a single trace event entry. In the results display of the [Aggregated Call Tree](#), the *Executions* column displays how many trace events were aggregated in a single trace event entry.

#### EXAMPLE

Imagine a class with methods M1, M2, and M3. Now imagine a program PROG that calls M1 twice from the same call position. The first call of M1 implies also the call of method M2. Finally, M2 is called through the method M3, as shown in the figure below.

In the [Aggregated Call Tree](#), the ABAP Profiler creates a single trace entry for M1. The *Executions* column has the value 2. Due to the fact that both calls to M1 occurred via the same call path, the ABAP Profiler aggregates the two calls to M1. The number of *Executions* indicates that two executions of method M1 have been added up.



#### Trace events in the Call Sequence and in the Aggregated Call Tree

Use the [Aggregated Call Tree](#) to perform tasks such as the following:

- Analyze all usages of a procedure (such as a method) - how many executions of a method occur via a particular call path?
- Analyze how many executions of specific events occur with the same call stack.
- Analyze which specific statements are executed and how often within a procedure, and which call stacks occurred for outgoing calls of the procedure.

## Procedure

1. To open the [Aggregated Call Tree](#), select the relevant trace in the [ABAP Traces](#) view and choose [Show Aggregated Call Tree](#) from the context menu.

### i Note

Alternatively, double-click the trace in the [ABAP Traces](#) view to open the [Overview](#) page of the trace and then choose the [Aggregated Call Tree](#) tab.

2. Expand the call tree and identify those trace events within the tree display that you want to analyze in more detail. You can pay special attention to the [Executions](#) and [Total Time](#) columns. **See also:** [Understanding Parameters in Aggregated Call Tree and Call Sequence](#) [page 664]

Aggregated Call Tree					
type filter text		State...	Exec...	Total Time...	Own Tim
Trace Event					
▷ Call Function WB_GET_BROWSER_REQUESTS	Show	1	9,139		
▷ Call M. CL_WB_REQUEST->CONSTRUCTOR	Show	1	43		
△ Call M. CL_WB_STARTUP->START	Show	1	3,569,675		
All statements within Call M. CL_WB_STARTUP->START			54		
△ Call Function WB_MANAGER_START		1	3,569,621		
All statements within Call Function WB_MANAGER_START			26		
△ Call Screen 0200		1	3,569,595		
All statements within Call Screen 0200			405		
▷ Dynpro Entry	Show	1	2,142		
△ Module PAI_MANAGER_START		1	3,567,048		
All statements within Module PAI_MANAGER_START			14		
△ Call M. CL_WB_STARTUP->START_INTERNAL	Show	1	3,567,034		
All statements within Call M. CL_WB_STARTUP->START_INTERNAL			149		
▷ Call M. CL_WB_STARTUP->GET_SINGLETON_COMPONENTS	Show	1	16,071		
Call M. CL_WB_WORKSPACE->ADD_WB_REQUEST		9	69		
△ Call M. CL_WB_MANAGER->IF_WB_MANAGER->SET_WORKSPACE		1	3,550,745		
All statements within Call M. CL_WB_MANAGER->IF_WB_MANAGER->SET_WORKSPACE			75		
Call M. CL_WB_WORKSPACE->START_ITERATION		1	0		

Trace Parameters in the Aggregated Call Tree

## Related Information

[Displaying the Call Stack \[page 674\]](#)

[Analyzing Program Flow with the Call Sequence \[page 661\]](#)

[Finding Hot Spots with the Hit List \[page 657\]](#)

### 5.3.2.3.6 Analyzing Aggregated Call Trees Using the Aggregated Timeline

You use the [Aggregated Timeline](#) as an optional tool to visualize the aggregated trace events and time consumed in the form of a diagram.

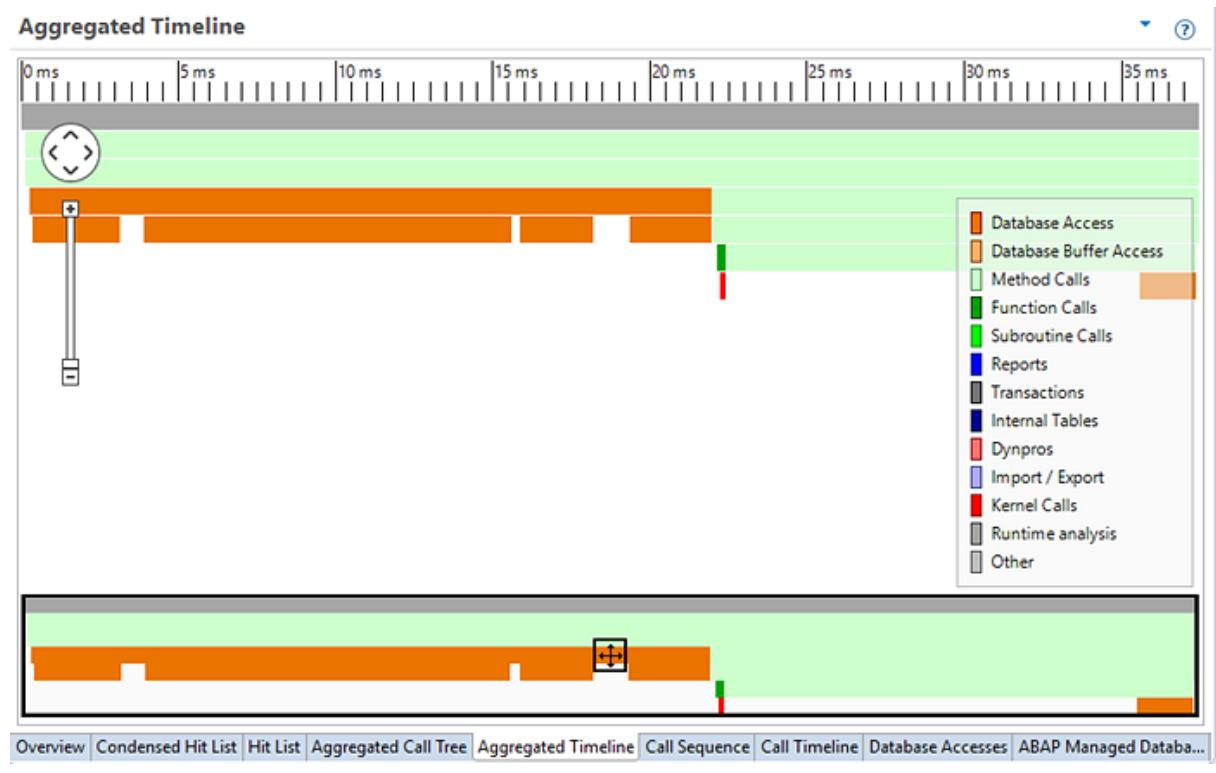
## Prerequisites

- You have created an ABAP trace for your program. The trace has the status *Finished*.
- The ABAP trace is not aggregated. **More on this:** [Aggregation Options in ABAP Traces \[page 682\]](#)

## Context

In general, the appearance of the aggregated trace events in this graphical tool corresponds to the display in the [Aggregated Call Tree](#). The horizontal axis of the diagram displays the time used by each of the aggregated traces events measured, while the vertical axis represents the call depth within a call hierarchy.

In contrast to the [Aggregated Call Tree](#) tool, however, the trace events are represented as a diagram and not as discrete tree nodes. The advantage of the new tool is the graphical representation and the quick detection of eye-catching patterns.



## Procedure

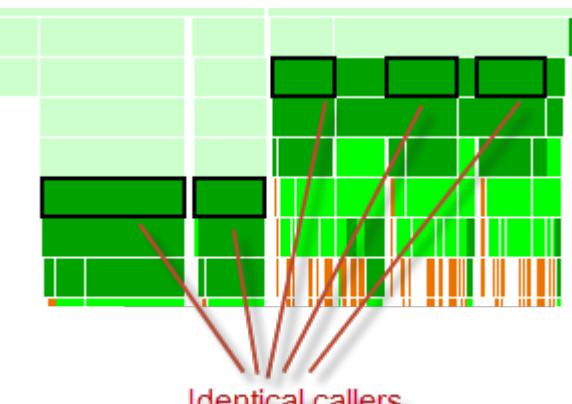
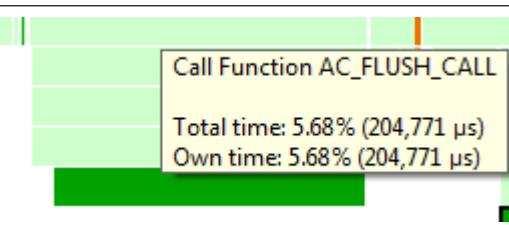
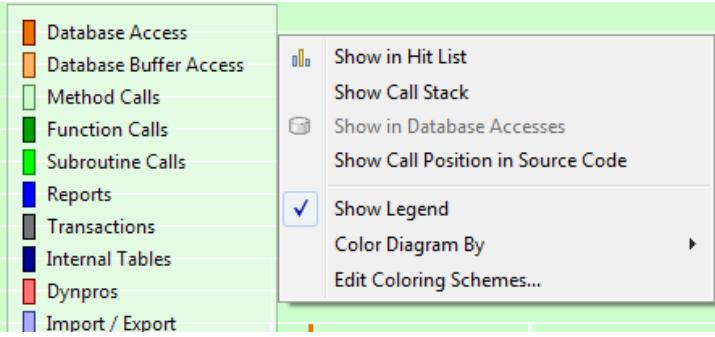
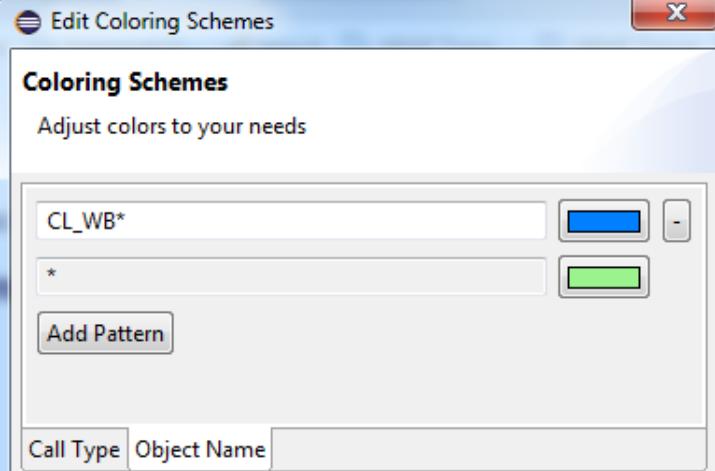
1. To open the [Aggregated Timeline](#) tool, select the relevant trace in the [ABAP Traces](#) view and choose [Show Aggregated Timeline](#) from the context menu.

### i Note

Alternatively, double-click the trace in the [ABAP Traces](#) view to open the [Overview](#) page of the trace and then choose the [Aggregated Timeline](#) tab.

2. Select the appropriate sequence using the thumbnail view of the diagram.

You can benefit from various functions that are available within this graphical tool:

Option	Description
Highlight identical callers	 Identical callers
Show measurement data of a trace event	
Display the legend	 Database Access Database Buffer Access Method Calls Function Calls Subroutine Calls Reports Transactions Internal Tables Dynpros Import / Export
Edit coloring schemas	 Edit Coloring Schemes Coloring Schemes Adjust colors to your needs CL_WB* * Add Pattern Call Type Object Name

Option	Description
Personalize tool settings	

### 5.3.2.3.7 Analyzing Call Sequence Using the Call Timeline Tool

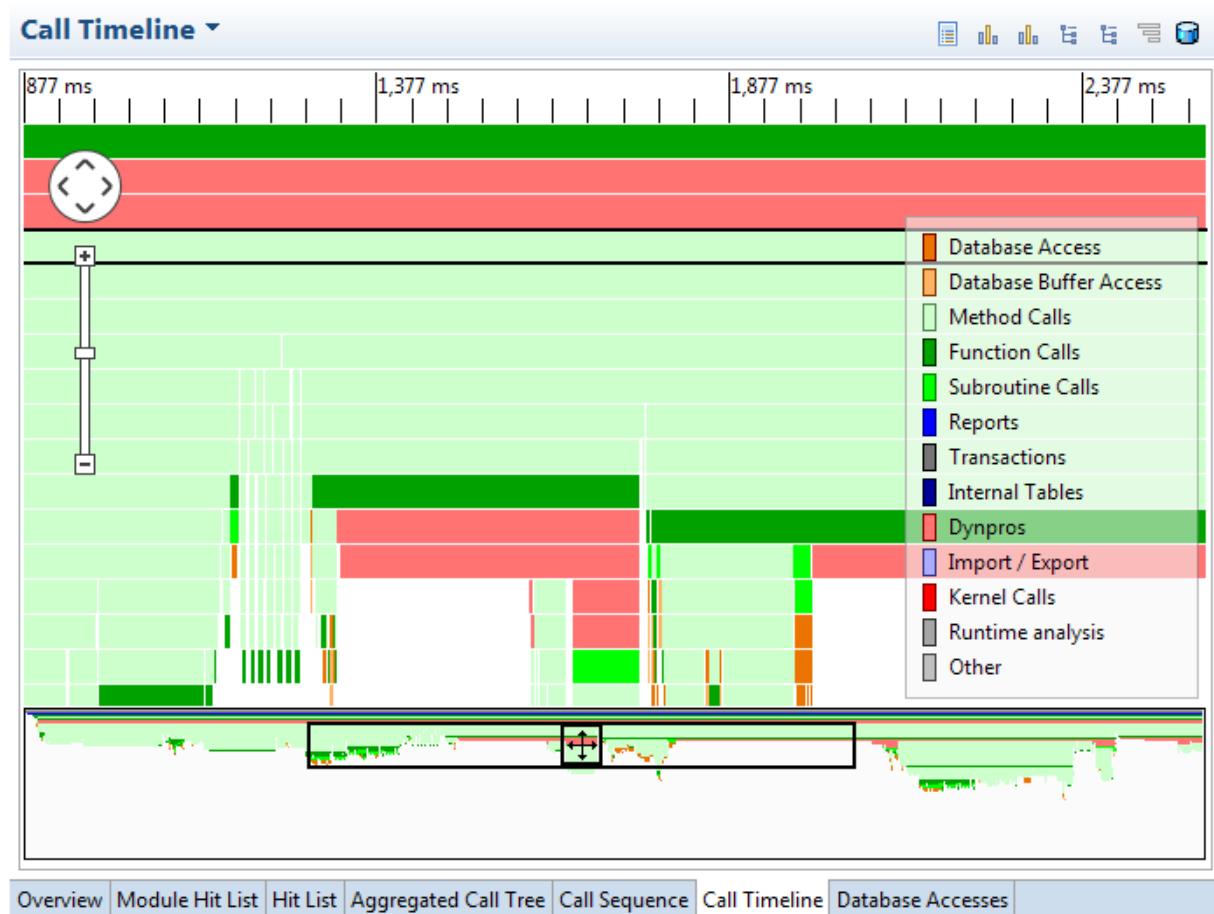
#### Prerequisites

- You have created an ABAP trace of your program. The trace has the status *Finished*.
- The ABAP trace is not aggregated. **More on this:** [Aggregation Options in ABAP Traces \[page 682\]](#)

#### Context

The *ABAP Profiler* provides you with a *Call Timeline* as an optional tool, which visualizes the trace events and time consumed in form of a diagram. In general, the appearance of the trace events in this graphical tool corresponds to the display in the *Call Sequence* tool. The horizontal line of the diagram displays the temporal sequence of each of the traces events measured, while the vertical line represents the call depth within a call hierarchy.

In contrast to the *Call Sequence* tool, however, the trace events are represented not as discrete tree nodes, but as a continuous sequence. The advantage of the new tool is the graphical representation and the quick detection of eye-catching patterns.



Trace events in the graphical Call Timeline tool

## Procedure

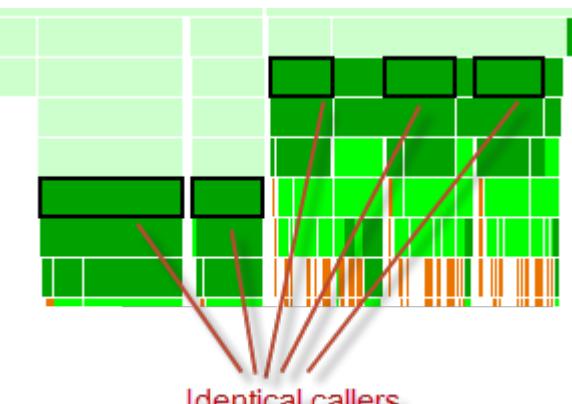
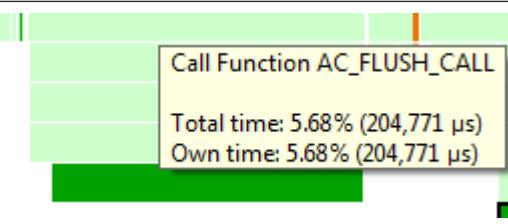
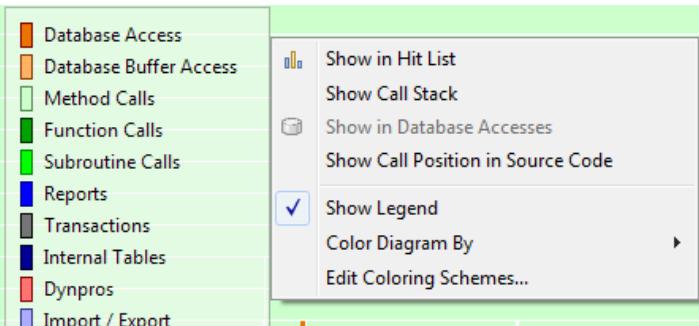
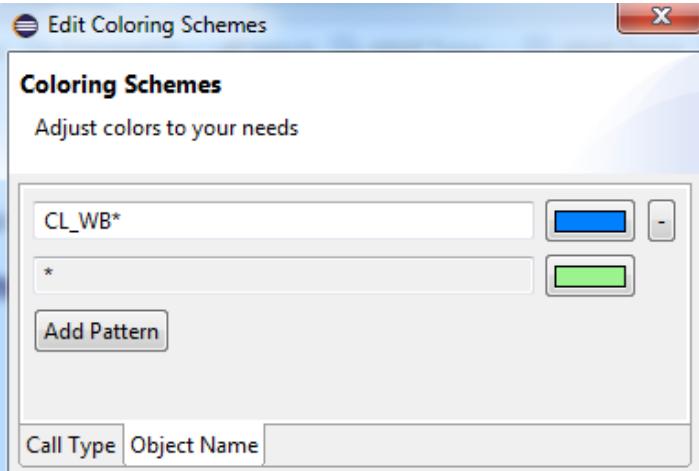
1. To open the *Call Timeline* tool, select the relevant trace in the *ABAP Traces* view and choose *Show Call Timeline* from the context menu.

### Note

Alternatively, double-click the trace in the *ABAP Traces* view to open the *Overview* page of the trace and then choose the *Call Timeline* tab.

2. Select the appropriate sequence using the thumbnail view of the diagram.

You can benefit from various functions that are available within this graphical tool:

Option	Description
Highlighting identical callers	 Identical callers
Showing measurement data of trace event	 Call Function AC_FLUSH_CALL Total time: 5.68% (204,771 µs) Own time: 5.68% (204,771 µs)
Displaying the legend	 Legend: Database Access Database Buffer Access Method Calls Function Calls Subroutine Calls Reports Transactions Internal Tables Dynpros Import / Export Context menu: Show in Hit List Show Call Stack Show in Database Accesses Show Call Position in Source Code <input checked="" type="checkbox"/> Show Legend Color Diagram By Edit Coloring Schemes...
Editing coloring schemas	 Edit Coloring Schemes Coloring Schemes Adjust colors to your needs CL_WB* <input type="color" value="blue"/> <input type="button" value="-"/> * <input type="color" value="green"/> <input type="button" value="Add Pattern"/> Call Type Object Name

Option	Description
Personalizing tool settings	

### 5.3.2.3.8 Displaying the Call Stack

#### Prerequisites

The ABAP trace must be aggregated by **Call Position**. More on this: [Aggregation Options in ABAP Traces \[page 682\]](#)

#### Context

You can display the call stack that leads to a particular trace event in the [Aggregated Call Tree](#), the [Call Sequence](#), or in another analysis tool.

#### Procedure

1. Put the cursor on an entry in the [Aggregated Call Tree](#), or the [Call Sequence](#).

→ Tip

You can also display a call stack from the [Condensed Hit List](#). If the number of stacks for hit list entry is > 1, the ABAP Trace first shows you the individual trace events. From this display, in the [Search](#) view, you can display the call stack of each trace entry.

2. Choose [Show Call Stack](#) from the context menu of the trace entry.

## Results

The ABAP Trace displays the call stack in a separate *Properties* view. From this view, you can jump to the position of a call stack entry in the *Call Sequence* or *Aggregated Call Tree*, or you can jump into the source code editor in order to analyze it or to set a breakpoint.

## Related Information

[Analyzing Trace Events in the Aggregated Call Tree \[page 666\]](#)

[Analyzing Program Flow with the Call Sequence \[page 661\]](#)

### 5.3.2.3.9 Analyzing Database Accesses

Whenever the database portion of a runtime distribution is (unexpected) high, you may wish to find out, which database accesses in your application have caused this to happen.

## Prerequisites

You have created an ABAP trace of your ABAP program. The ABAP trace has the state *Finished*.

## Context

The *Database Accesses* tool allows you to **identify and analyze** the **top consumers** during database accesses. It provides you with a list of all database tables and table views that are used during the execution of the ABAP program in question. In addition, detailed information for each database access, such as access type, buffer settings, or duration of table accesses is provided.

## Procedure

1. Open the *ABAP Traces* view.
2. In the view, expand the relevant project tree and double-click the trace entry. The *Overview* page appears for the selected trace.
3. Choose the *Database Accesses* tab.

14:59:15, | 03.06.2013 | DEMO\_ALV\_REPORTING ✎

**Database Accesses [Total Time: 879 ms, Database Time: 109.781 µs (**

type filter text

Table Name	SQL State...	Access ...	E...	B...	...	Total...	Buffer Settings
<DB Time of System Events>			0	0	0	70.132	
<DB Access from Kernel>			28	0	0	26.626	
SCARR	select	OpenSQL	1	0	1	5.509	
DDFTX	select	OpenSQL	1	1	1	4.834	Generically buffered
SALV_CSQ_PARAMS	select	OpenSQL	6	6	1	1.264	Generically buffered
V_EXT_ACT	select	OpenSQL	2	2	1	260	Entirely buffered
SXS_ATTR	select single	OpenSQL	4	4	2	253	Single Entries buffer

Overview | Hit List | Call Tree | Call Timeline | **Database Accesses**

The Database Accesses view provides you with details for each database access in your ABAP program

## Results

Using the *Databases Accesses* view, you can...

- Identify top consumers during the database accesses
- Check which portion of runtime is used by database accesses from ABAP kernel or system events
- Get a complete list of tables (views) that are used during the execution of your ABAP program
- Check, where your program repeats the same database access
- Analyze further details of each database access and check, for example, whether and how the tables are buffered
- Navigate to the table (view) definition in the ABAP Dictionary.

### → Tip

To restrict the data in the view, you can enter the appropriate filter text or configure the view by using the context menu in the table header.

**Database Accesses [Total Time: 879 ms, Database Time: 109.781 µs (1**

type filter text

Table Name	SQL Statement	Access T	Executions	Buffer
<DB Time of Sys	Table Name			
<DB Access from	SQL Statement			
SCARR	Database access type used for this statement			
DDFTX	Total execution number for this statement			
SALV_CSQ_PARA	Number of buffered accesses			
V_EXT_ACT	Number of source code positions where this statement is used			
SXS_ATTR	Total time spent in this event in microseconds			
AQXINT	Rate [%] of total time of this event to total time of the trace file			
AQXINTT	Buffer Settings			
TRSTI	Table Type			
TRSTIO	Description of the table			
SRAL_RECORD	Package			
AQXINTA	select	OpenSQL	1	

Configuring the view details using the context menu in the table header

## Related Information

[Trace Parameters in Database Accesses View \[page 677\]](#)

### 5.3.2.3.9.1 Trace Parameters in Database Accesses View

Here is some information on the trace parameters that are displayed in the **Database Accesses** tool:

Field	Description
<i>Table Name</i>	Name of database table, database view, or generated view that is used when the ABAP program in question is run. In addition, you will also find the corresponding entries for DB accesses from ABAP kernel and system events.
<i>SQL Statement</i>	Kind of SQL statement from which the database access results.
<i>Access Type</i>	Specifies whether access to the database table takes place using the <b>ABAP SQL</b> or the <b>Native SQL</b> database interface.
<i>Executions</i>	Total number of executions of the SQL statement.
<i>Buffered Accesses</i>	Total number of buffered accesses. Since non-buffered accesses to buffered tables and views may occur, this number is always <= the number of <i>Executions</i> .

Field	Description
<i>Positions</i>	Number of source code positions where the related SQL statement is used to access the corresponding table (view).
<i>Total Time [μs]</i>	Total time spent for all executions of the SQL statement in microseconds.
<i>% Total Time</i>	Total time of this trace event as a percentage of the total time of the trace file.
<i>Buffer Settings</i>	<p>These settings specify whether or not and how the relevant table is buffered - corresponding to the technical settings of the table. If the table is buffered, the type of buffering is specified by one of the following options:</p> <ul style="list-style-type: none"> <li>• <b>Entirely buffered</b> - All records of the table are loaded into the buffer when one record of the table is accessed.</li> <li>• <b>Single entries buffered</b> - Only the records of a table that are really accessed are loaded into the buffer.</li> <li>• <b>Generically buffered</b> - When a record of the table is accessed, all the records that have the same generic key as this record are loaded into the buffer.</li> </ul>
<i>Table Type</i>	Specifies whether the table entry is a transparent table, a (standard) view, or a generated view in the ABAP Dictionary.
<i>Short Text</i>	Short text that has been entered for the table (view) and is displayed in the <b>Default Language</b> of the ABAP project.
<i>Package</i>	ABAP package to which the Dictionary table (view) belongs.

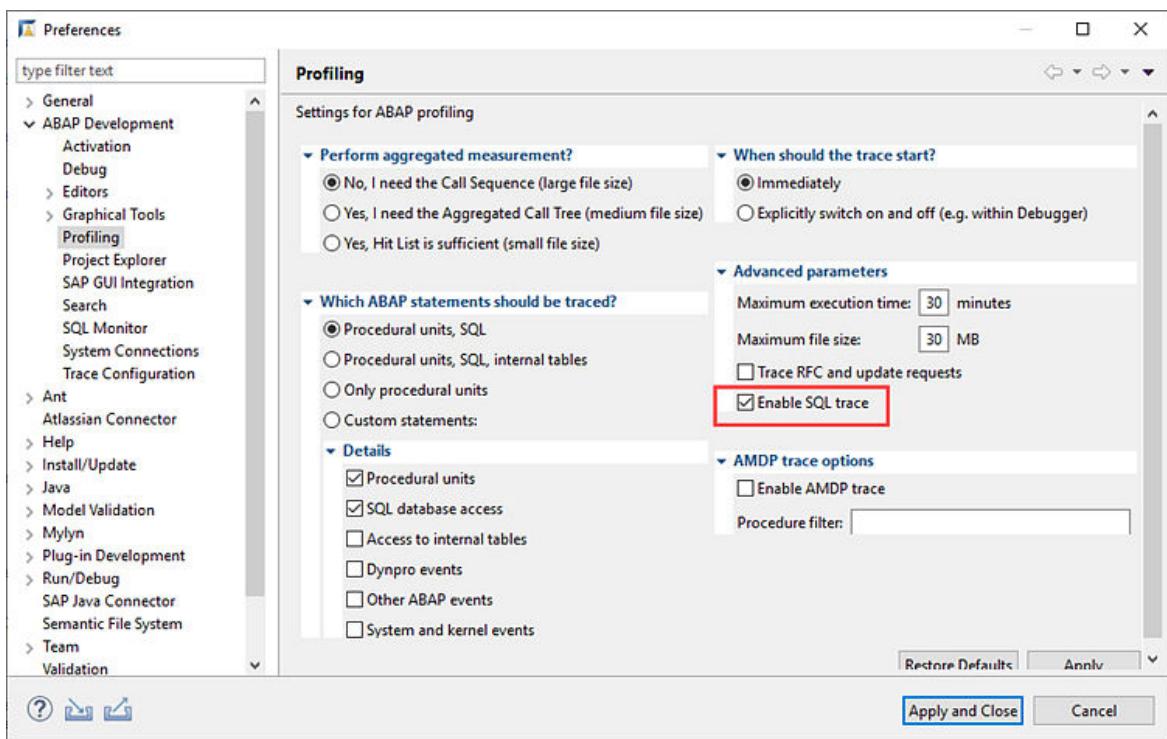
### 5.3.2.4 Setting Trace Preferences

#### Context

You can customize how the ABAP Trace records ABAP program runs. Here is how to set the ABAP Trace preferences.

#### Procedure

1. From the menu, open the    *ABAP Development*  *Profiling*  *Preferences* page.
2. In the *Advanced parameters* section, set options that apply to all traces.



The option **Enable SQL trace** lets you switch on the SQL trace in addition to the ABAP trace

See more in [Understanding ABAP Profiler Settings \[page 680\]](#).

## Results

ABAP traces that you start with **Profile As > ABAP Application** or via a trace request will use the preferences that you have set.

## Related Information

[Working with Profile Configurations \[page 649\]](#)

## 5.3.2.4.1 Understanding ABAP Profiler Settings

### Configuration Options

Here is detailed information on the configuration options you have for ABAP traces:

**Perform aggregated measurements?** Here you specify the aggregation mode for the ABAP trace to be created. The following values are available for the aggregation mode: **NONE**, **By Call Stack**, **By Call Position**.

Choose **No, I need the Call Sequence (large file size)** if you want to create a non-aggregated trace in which each trace event is recorded in a separate trace entry. This level of detail makes a trace file larger, but it is required by the *Call Sequence* analysis tool.

Choose **Yes, I need the Aggregated Call Tree (medium file size)** if you need an aggregated ABAP trace. For this option, the trace events are aggregated according to their call stack (aggregation mode: **By Call Stack**). That means, the trace events that have exactly the same call stack are accumulated in a single trace entry.

Choose **Yes, Hit List is sufficient (small file size)** if you only need the **Hit List** trace display. This is an aggregated ABAP trace where trace events are aggregated by call position (aggregation mode: **By Call Position**). That means, in identical traces, events that are called identically (such as method calls in a loop) are accumulated in a single trace entry.

Remember, the aggregation mode has a major impact on the file size created for the ABAP trace and availability of tools you can use to analyze the trace. More on this: [Aggregation Options in ABAP Traces \[page 682\]](#)

---

## Which ABAP statements should be traced?

The ABAP Trace can record all of the ABAP statements and events that are potentially performance-consuming. But this can result in a large amount of trace data, and you may not need to see all of the available trace entries. Here, therefore, you can specify which trace events you want to see.

- **Procedural units, SQL:** A trace records all of the ABAP modularization statements, such as CALL METHOD and CALL FUNCTION as well as any database activity, such as SELECT or UPDATE operations.
- **Procedural units, SQL, internal tables:** As above, but operations on internal tables are also included in a trace.
- **Only procedural units:** A trace includes only ABAP modularization statements, like CALL METHOD or CALL FUNCTION.
- **Custom statements:** Opens the checkboxes under **Details**. Here you can switch on or off tracing of the various categories of ABAP statements and events that the ABAP Trace is able to capture.  
Options not described in the rows above are as follows:
  - **Dynpro events** traces events in the special processing for traditional ABAP Dynpro screens.
  - **Other ABAP Events** switches on tracing of all traceable ABAP statements and events other than those specified above. This includes, for example, data transfer statements (READ DATASET, TRANSFER) and other statements, such as ASSIGN, MESSAGE, and IMPORT.
  - **System and kernel events** switches on tracing of ABAP infrastructure code (programs of the ABAP category *system programs*) as well as calls from ABAP to kernel functions.

The ABAP Trace measures the complete runtime of a program, no matter which statements are traced. What changes with your choices is the level of detail in the trace. The full runtime of a method is always shown in a trace; the number of different trace records within the method changes with your statement tracing choices.

---

## When should the trace start?

Immediately starts tracing when you choose .

If you specify *Explicitly switch on and off (for example, within Debugger)*, you can trace dedicated code sequences of your request. The trace for a code sequence can be switched on and off using the ABAP statement SET RUN TIME ANALYZER {ON|OFF} or the ABAP Debugger.

**More on this:** [Starting and Stopping the ABAP Profiler in the Debugger \[page 644\]](#)

---

#### Advanced Parameters

- **Maximum execution time:** This option prevents a trace from running forever (or until the file system is full). Traces end when the program that is being traced ends, so an endlessly running program can, in theory, produce a very large trace. You can leave this option set to its default value.
- **Trace RFC and update requests:** This option lets you follow RFC calls made from the program that you are tracing. If an RFC function module is called, the trace is switched on in the target server or system in which the RFC is running. You can find RFC trace entries by looking for the keyword RFC. The trace file of the request that triggered the RFC call does not contain processing details of the called function module. For these you must display the traces performed in the target system. In addition, when you choose this option, all function modules that are called from the traced coding with option IN UPDATE TASK will be traced. These traces are contained in separate trace files for each invocation of the update task by COMMIT WORK.
- **Enable SQL trace:** This option lets you switch on the SQL trace in addition to the ABAP trace. The SQL trace function records the database activity of the executing program. It allows you to see how the ABAP SQL statements that are used in ABAP code are converted to standard SQL operations and which parameters of the SQL statements are passed to the database system.

### 5.3.2.4.1.1 Aggregation Options in ABAP Traces

The aggregation option specifies whether or how the ABAP trace data is aggregated in the trace file. The aggregation option has a major impact on file size and the availability of tools for analyzing the ABAP trace:

#### Aggregation Options

Aggregation			
Option	Meaning	File Size	Tool Availability
NONE	Each trace event is stored individually in one trace entry.	Huge trace file, linear increase with request run-time	Module Hit List Hit List Aggregated Call Tree Call Sequence Call Timeline

Aggregation			
Option	Meaning	File Size	Tool Availability
<b>By Call Stack</b>	Trace events are aggregated according to their call stack.	Acceptable file size	Module Hit List Hit List  Aggregated Call Tree
<b>By Call Position</b>	Trace events are aggregated by their call position. No stack information available.	Small file size, most compact format	Hit List

### 5.3.2.5 Working with the ABAP Cross Trace

As an ABAP developer of SAP Fiori apps, you can use the ABAP Cross Trace to analyze errors.

You have the following possibilities to work with the ABAP Cross Trace:

- [Creating an ABAP Cross Trace Configuration \[page 684\]](#) to define what and how to trace, as well as the start point of the trace.
- [Stopping Tracing \[page 686\]](#) to end tracing.
- [Displaying the ABAP Cross Trace Overview \[page 687\]](#) to list the trace results.
- [Analyzing an ABAP Cross Trace Result \[page 688\]](#) to search the root cause of an error.

### Related Information

[ABAP Profiling \[page 48\]](#)

[Profiling ABAP Code \[page 640\]](#)

## 5.3.2.5.1 Creating an ABAP Cross Trace Configuration

The configuration of the ABAP Cross Trace determines what is to be traced.

### Prerequisites

You can **only** trace SAP Fiori apps that are built using the ABAP RESTful Application Programming Model (in short RAP).

Your user needs the authorization object `S_XTRACE`.

### Context

You want to investigate, for example, an error resulting from an SAP Fiori app which you work on.

#### → Recommendation

SAP recommends to stop the ABAP Cross Trace right after you have performed the request to be traced. For further information, see [Stopping Tracing \[page 686\]](#).

### Procedure

1. To open the *ABAP Cross Trace* view, choose  *Window*  *Show View*  *Other...*  *ABAP Cross Trace*  from the menu bar.  
The *Trace Configuration* tab in *ABAP Cross Trace* view is opened at the bottom.
2. Select the relevant ABAP system where you want to trace and choose *Create Configuration... (Ctrl + N)* from the context menu.  
The *Create ABAP Cross Trace Configuration* dialog is opened.
3. [Optional:] To consider all kinds of data that is processed in your SAP Fiori app in the trace result, choose the *Include potentially sensitive data* checkbox in the *Properties* section.
4. [Optional:] To define the date and time when the trace configuration should be deleted, define the *Deletion At* conditions using the relevant buttons.

#### Note

The deletion date and time is used to clean the system. The trace configuration will then be deleted without any further information.

5. [Optional:] To provide a short text, enter the relevant information in the *Description* text field.
6. [Optional:] To only trace the executions of a specific user, enter or browse for the relevant *User*.

### **i Note**

You can also use wildcards, such as asterisk "\*". You then trace all users. Note that this has impact on performance and memory consumption.

7. [Optional:] To specify the request types that you want to trace, select one of the following entries from the *Request Entry Type* drop-down list.

### **i Note**

You can leave this field blank.

8. [Optional:] To specify the requests that you want to trace, enter the name of the relevant *Request Entry Name*.

You can enter, for example, the name of the OData service.

### **i Note**

You can also use wildcards, such as asterisk "\*".

9. [Optional:] To limit the number of components to be traced, search and deselect the relevant ones in the *Components* section.

### **i Note**

To define the granularity of the data to be traced, choose the relevant entry from the drop-down list box in the *Trace Level* column.

10. To confirm, choose *OK*.

### **i Note**

If you are not authorized for tracing the user, sensitive data, or components, an error will be displayed when you try to save the configuration.

## **Results**

The trace configuration is stored in the ABAP system. In the *Trace Configurations* tab, a new configuration will be added to your project.

If the trace configuration is active, the system will now trace those requests that match the configuration. This means, a new trace result will be created for each request.

You can now, for example, run the SAP Fiori app to be traced. The trace results can then be displayed and analyzed in the *Trace Results* tab.

## **Related Information**

[Stopping Tracing \[page 686\]](#)

[Displaying the ABAP Cross Trace Overview \[page 687\]](#)

[Analyzing an ABAP Cross Trace Result \[page 688\]](#)

## 5.3.2.5.2 Stopping Tracing

After you have finished tracing, you should stop the ABAP Cross Trace.

### Context

You have configured an ABAP Cross Trace and have run an SAP Fiori app. You now want to stop tracing manually.

### Procedure

In the *Trace Configurations* tab of the *ABAP Cross Trace* view, select the relevant trace configuration and choose *Deactivate Tracing* from the context menu.

### Results

Tracing will be stopped and the trace configuration will be deactivated.

You can now display the trace results in the *Trace Results* tab.

### Related Information

[Displaying the ABAP Cross Trace Overview \[page 687\]](#)

### 5.3.2.5.3 Displaying the ABAP Cross Trace Overview

The *Trace Results* tab displays the trace results. In accordance to your permissions, you can also display the trace results of other users.

#### Context

You want to get an overview of the available trace results.

#### Procedure

1. Open the  *Window*  *Show View*  *Other...*  *ABAP Cross Trace*  view from the menu bar.  
In the *ABAP Cross Trace* view, the *Trace Configurations* tab is opened.
2. Navigate to the *Trace Results* tab.  
Here, the following information is displayed:
  - *Trace Properties* are key-value pairs. The properties provide keywords that enable you to get a first impression of the trace result.
  - *Request User* displays the name of the user who run the ABAP program to be traced.
  - *Request Entry Type* displays the request entry type of the traced request, for example, OData V4.
  - *Request Entry Name* displays the name of the request entry, for example, the name of an OData service.
  - *Description* provides a short text.
  - *Deletion At* displays the date when the trace configuration will automatically be deleted.
  - *"EPP" fields* displays the extended passport (EPP).  
For more information, see [Extended Passport \(EPP\)](#).
3. [Optional:] To display the trace results from another user, select the project and choose *Change Filter...* from the context menu and browse for the relevant user.
4. [Optional:] Refresh the view.
5. Select and double-click the relevant trace result. Alternatively, you can choose *Open Trace Records* from the context menu.

#### Results

The trace result is opened in the editor. If no records are shown, or if you miss important records, check the filters and adjust them in accordance.

You can now start analyzing the trace results.

## Related Information

[Creating an ABAP Cross Trace Configuration \[page 684\]](#)

[Stopping Tracing \[page 686\]](#)

[Analyzing an ABAP Cross Trace Result \[page 688\]](#)

### 5.3.2.5.4 Analyzing an ABAP Cross Trace Result

#### Context

You want to find the root cause of an error that relates to your SAP Fiori app.

#### Procedure

1. [Display \[page 687\]](#) the trace results.

The trace result is opened in the editor.

You can find the following information in the columns:

- *Procedure* specifies the location where the trace record originates. This can be, for example, a method name or a descriptive location name.
- *Object* specifies the location where the trace record originates. This can be, for example, an ABAP class name or a descriptive location name.
- *Message* provides the main information of the trace record.
- *Content Size* describes the size of the content in bytes. The content itself is displayed in the *Properties* view.
- *Component* that wrote this trace record.
- *Record Properties* are key-value pairs. The properties provide keywords that enable filtering. The record properties are also displayed in the *Properties* view.
- *Call Stack Depth* displays the depth of the traced call stack which is typically shortened. The call stack itself is displayed in the *Properties* view.
- *Created At* provides the time stamp of when the trace record was created.
- *Offset (ms)* describes the offset to the time stamp of the first trace record in milliseconds.
- *Record Number* provides an ascending, unique number for each trace record.
- *Trace Level* displays the trace level of the trace record, for example, *Essential Information* or *Expert Information*.

#### Note

By default, ABAP Development Tools only displays records with trace level *Essential Information*.

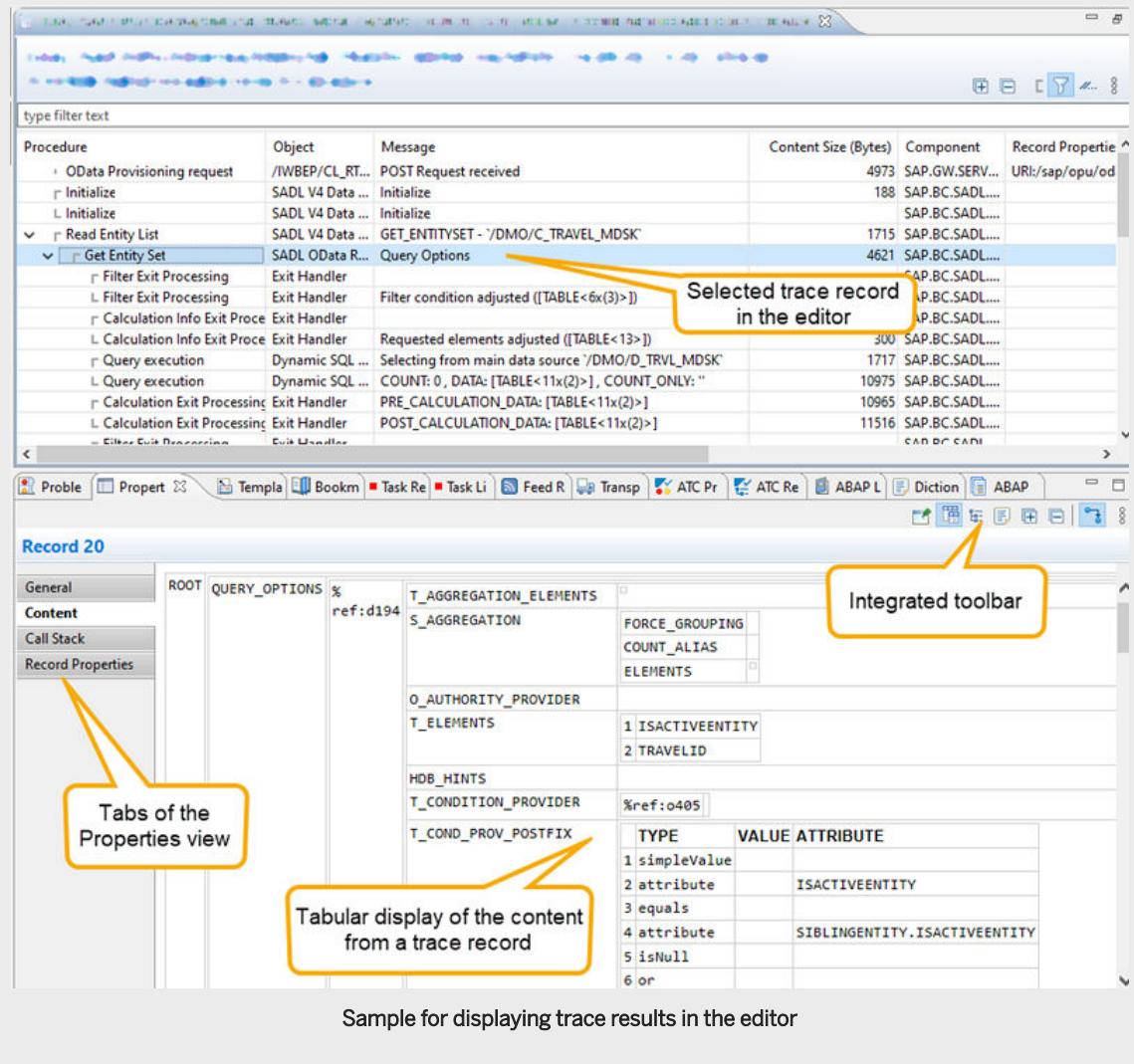
Note that only this level is relevant for SAP Fiori application developers.

To display more detailed information, select the relevant item using the  filter icon from the integrated toolbar.

2. In the editor, double-click the trace record for which you want to get more information.

The *Properties* view is opened and displays the relevant information in a tabular view in the *Content* section.

### Example



The screenshot illustrates the SAP ABAP Development User Guide interface for displaying trace results. At the top, a trace editor window shows a list of trace records. One record, 'Get Entity Set' under 'Read Entity List', is selected and highlighted with a yellow box. A callout points to this record with the text 'Selected trace record in the editor'. Below the editor, the SAP toolbar is visible. The main content area shows the 'Record 20' properties view. The 'Content' tab is selected, displaying a tabular view of the trace record's content. Annotations highlight the 'Integrated toolbar' (top right of the editor), the 'Tabs of the Properties view' (General, Content, Call Stack, Record Properties), and the 'Tabular display of the content from a trace record' (Content tab). A callout points to the content table with the text 'Tabular display of the content from a trace record'.

Sample for displaying trace results in the editor

From here you can continue your analysis in the following tabs:

- o *General* displays basic attributes of the trace record.
- o *Content* displays the content of the trace record.

To modify how to display the trace result, choose one of the following icons from the integrated toolbar:

- o  to display the content in a table.

#### Note

If the content contains references to data or ABAP objects, those will be located in a separate heap section. To resolve references, use the  icon.

-  to display the content in a tree structure.

#### Note

If the content contains references to data or ABAP objects, those will be located in a separate heap section. To resolve references, use the  icon.

-  to display the content as text in JSON format.
  - *Call Stack* displays the location in the ABAP source code where the trace record was written. Note that you can navigate to the source code by choosing *Navigate to Source* from the context menu or by double-clicking the relevant call stack entry. The development object is then opened and the cursor is positioned at the relevant occurrence.
  - *Record Properties* displays the properties of the trace record.
3. [Optional:] To inform, for example, another user about a trace result, choose the *Share Link...* icon from the integrated toolbar in the editor.

A dialog is then opened. From here, you can open your default email client that contains the corresponding ADT link **or** you can copy the ADT link to the clipboard and paste it, for example, into a document.

## Results

You can now analyze the trace result to get a better understanding of your SAP Fiori app and then take over the results of your analysis into your development activities.

## Related Information

[Creating an ABAP Cross Trace Configuration \[page 684\]](#)

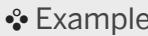
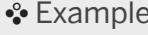
[Stopping Tracing \[page 686\]](#)

[Displaying the ABAP Cross Trace Overview \[page 687\]](#)

## 5.3.2.6 Glossary

Term	Meaning
 ABAP Cross Trace	An ABAP framework and tool used to analyze the runtime of ABAP programs

Term	Meaning
 ABAP Cross Trace Configuration	Defines the request parameters (for example, user name, OData requests, and so on) to be traced.
ABAP Profiler	Name of the complete toolset  ABAP Profiler lets you analyze the runtime of ABAP programs.
(ABAP) Profiler Wizard	Component of the ABAP Profiler  Using this wizard, you can launch the ABAP Profiler for various object types. Independently of any context, you can select an arbitrary development object to be profiled.
ABAP Profiling Perspective	Eclipse perspective that mainly includes tools that focus on profiling activities, such as <i>ABAP Traces</i> and <i>ABAP Trace Request</i> views.
ABAP Trace	Results of measurement when using the ABAP Profiler  Each ABAP trace corresponds to an ABAP trace file generated by the ABAP runtime. ABAP traces are listed in the <i>ABAP Traces</i> view.
(ABAP) Trace Request	A trace request enables profiling of an ABAP request such as a program, transaction, HTTP request, RFC call, or a background process, whenever its execution is triggered in the backend system.
ABAP Trace Request View	One of the main tools within the ABAP Profiling perspective  It enables the user to create asynchronous trace requests.
ABAP Traces View	View that is used to access individual ABAP Traces for each ABAP project that is opened in the user's workspace.
Aggregation	General property of an ABAP trace  The user can specify the aggregation option for each ABAP trace to be created. The following values are available for the aggregation: <i>NONE</i> , <i>By Call Stack</i> , <i>By Call Position</i> .
Analysis Tools	Part of ABAP Profiler toolset  Analysis tools provide the user with different views on the ABAP Trace. It includes the following tools: <i>Condensed Hit List</i> , <i>Hit List</i> , <i>Aggregated Call Tree</i> , <i>Call Sequence</i> , <i>Call Timeline</i> , <i>Database Accesses</i> , and <i>SQL Trace</i> .
Profile Configuration	Launch configuration for the ABAP Profiler  The profile configuration allows the user to re-run the ABAP Profiler with predefined settings
Requester (of an ABAP Trace)	User who launches the ABAP Profiler in order to create an ABAP Trace.
SQL Trace	(Optional) Part of the ABAP Trace  The SQL Trace can be switched on, in addition to the ABAP Trace, to record the database activities of the executing program.

Term	Meaning
to create ABAP Trace	<p>User action: allows the user to create an ABAP Trace (for example, by means of the ABAP Profiler wizard)</p> <p>Synonym: to profile</p>
to create ABAP Trace Request	User action: enables the user to tell the ABAP Profiler to wait until a program runs and then automatically profile its execution.
to profile	Synonym: to create ABAP Trace
<p> <b>Example</b></p> <p>Profile an ABAP development object</p>	
Trace Event	<p>An ABAP statement or other operation (a database operation, for example) for which the ABAP Profiler records discrete performance data</p> <p>Each individual trace event that has been recorded allows you to analyze the runtime of the underlying ABAP statement or other operation.</p> <p>Examples: method calls, call function XYZ, Prepare DB</p>
Trace Parameters	<p>Parameters that are used to analyze ABAP traces in analysis tools</p> <p>Each analysis tool displays a specific set of trace parameters.</p> <p> <b>Example</b></p> <p><i>Own Time, Total Time, Executions, Calling Program.</i></p>
User (Application User)	User who starts the execution of the object or application

### 5.3.3 Analyzing SQL Statements

In ABAP Development Tools (ADT), the SQL trace allows you to see how ABAP SQL statements that you use in ABAP source code are converted to standard SQL statements, and the parameters with which the embedded SQL statements are passed to the database system.

The SQL trace can be helpful to analyze performance issues or functional errors.

You can create and display the SQL trace in the following ways:

- [Using the SQL Trace and the Technical Monitoring Cockpit \[page 693\]](#) to create an SQL trace in ADT and to analyze it in an SAP Fiori-based UI in your default Internet browser

### 5.3.3.1 Using the SQL Trace and the Technical Monitoring Cockpit

You can create SQL trace and display it in the *Technical Monitoring Cockpit*. This enables you to analyze the performance of SQL statements.

#### Prerequisites

Before you can investigate an issue using the SQL trace tool, you need to activate the creation of a SQL trace. To do this, see step 3 below.

#### Context

You want to investigate performances issues that relate to SQL code.

#### Procedure

1. In the *Project Explorer*, select an ABAP project.
2. Choose *SQL Trace* from the context menu.

The *SQL Trace...* dialog is opened.

Here, you can find the following information:

- *SQL trace state*: Status information about SQL trace
  - *Last change by user*: Name of the user who last changed the SQL trace state
  - *Last change on*: Date and time when SQL trace has been last activated
3. You can now choose the following options:
    - **Activate/Deactivate** to start or stop SQL trace.
    - **View Trace Directory** to open the SQL trace in the *Technical Monitoring Cockpit* in your default Internet browser.
  4. Run the development object that contains the SQL code to be investigated.
  5. Deactivate SQL trace using the *SQL Trace...* dialog.
  6. Open SQL trace in the *Technical Monitoring Cockpit*.

#### Results

The *Technical Monitoring Cockpit* is opened in your default Internet browser. It displays the content of the SQL trace. You can start your analysis from here.

### → Recommendation

SAP recommends to deactivate SQL trace after your analysis. Otherwise, data is collected all the time.

## Related Information

 [Technical Monitoring Cockpit](#)

 [SQL Trace Analysis](#)

## 5.3.4 Using Dynamic Logpoints

### Prerequisites

In order to use the logpoint functionality, you need in the relevant system:

- As a minimum, the display authorization for object `S_DEVELOP` with activity: 03 to be able to launch the ABAP source editor for displaying the ABAP source code.
- The authorization for object `S_DYNLGPTS` for using the basic logpoint functionality with the value `BASIC` of the authorization field `OBJNAME` and the values `LOGPTS` and `RESULT` for the field `OBJTYPE`. This *basic privilege* enables you to...
  - Create logpoints
  - Change, activate, deactivate, or delete logpoints
  - Access log entries (results).
- An additional authorization for object `S_DYNLGPTS` that provides you with advanced logging functionality, including...
  - Switching on / off traces (value `TRACING` for field `OBJNAME`)
  - Analyzing ABAP variables (value `VARIABLES` for field `OBJNAME`)

**More:** Read the documentation for .

### Developer-Relevant Activities

- [Setting Logpoints in the Source Editor \[page 695\]](#)
- [Displaying and Managing Logpoints \[page 701\]](#)
- [Viewing Logs \[page 706\]](#)

## Related Information

Related blog on SAP community page 

### 5.3.4.1 Setting Logpoints in the Source Editor

#### Prerequisites

You can set dynamic logpoints...

- In the active version of an ABAP program.
- For any executable ABAP statement in the program.

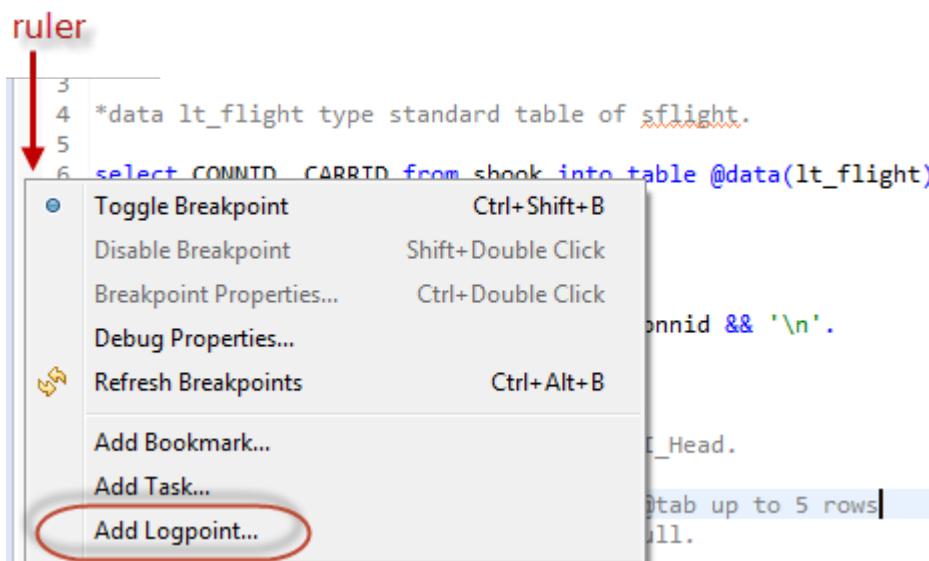
#### Context

In general, you can set dynamic logpoints for each source code line that contains an executable ABAP statement. When the logpoint is reached during program execution, the logpoint is evaluated. The logpoint evaluation first checks the condition. If the condition is fulfilled, a log entry is written according to the specified logpoint parameters.

#### Procedure

Setting Logpoints from the Ruler in the ABAP Source Code Editor

1. Position the cursor within the ruler (left bar) in the source editor at the line that contains the executable ABAP statement of your interest.
2. Open the context menu and choose *Add Logpoint*.



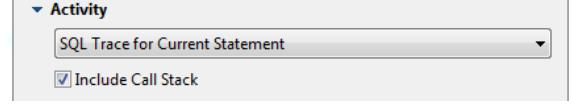
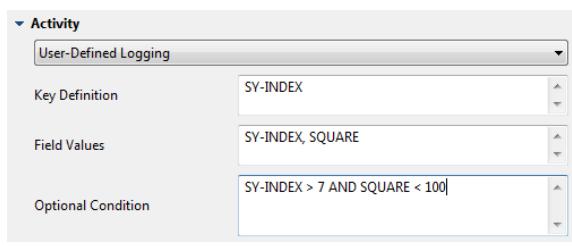
Setting logpoints from the ruler in the ABAP source code editor

3. In the wizard page that appears, specify the parameters for the logpoint to be created.

### Activity

So that you can use logpoints in the context of various use cases, the wizard provides you with a set of predefined activity options. You can select one of the predefined activity options or apply the logpoint syntax for a user-defined logpoint:

Activity	Meaning
Log Variable Values	<p><b>Activity</b></p> <p>Log Variable Values</p> <p>Variable(s) ls_flight, price</p> <p>You can enter one or multiple ABAP variables, separated by a SPACE or a comma. All variables define the key of the logpoint to be created. <b>More on this:</b> <a href="#">Key Definition in Logpoints [page 700]</a></p> <p>Whenever the logpoint is reached during program execution, the values of these variables will be added to the log entries as the key.</p> <p><b>Note</b></p> <p>Only variables with elementary data types (for example: variables that can be used within a <code>WRITE</code> statement) are valid variables here.</p>
Log Call Stacks	<p>With this option, you can set the logpoint for a program line where you are interested to know which call units (methods, functions, forms, or events) are the callers. So, in particular, you want to know which different call stacks lead to the program line for which the logpoint is defined.</p>

Activity	Meaning
<p><i>SQL Trace for Current Statement</i></p>	<p></p> <p>Using this activity option, you can switch on the SQL trace for an individual SQL statement in order to analyze performance issues in detail.</p> <p>To get the call stacks recorded in the SQL trace, in addition check <i>Include Call Stack</i>.</p>
<p><i>Table Buffer Trace for Current Statement</i></p>	<p>Using this activity, you have the option to switch on the table buffer trace. This option is valid if the selected code line comprises an ABAP SQL statement that refers to a buffered database table.</p> <p>To get the call stacks recorded in the buffer trace, check <i>Include Call Stack</i>.</p>
<p><i>User-Defined Logging</i></p>	<p>This option allows you to specify the logpoint using the logpoint syntax for the following <i>Parameters</i>:</p> <ul style="list-style-type: none"> <li>○ <i>Key Definition</i> More on this: <a href="#">Key Definition in Logpoints [page 700]</a></li> <li>○ <i>Field Values</i> More on this: <a href="#">Field Values in Logpoints [page 700]</a></li> <li>○ <i>Optional Condition</i> More on this: <a href="#">Conditions in Logpoints [page 699]</a></li> </ul> <p>Example:</p> <p></p>

## Description

In the *Description* field, you have the option to provide a text that describes the meaning or the usage context of the logpoint to be created.

### → Tip

Writing a description might especially be necessary if you want to set multiple logpoints in a single program, or even multiple logpoints for one and the same program line.

## Activation

You can specify the following parameters for the activation of the logpoint:

Parameter	Meaning
<i>Status</i>	<p>Active logpoints trigger log events during program execution.</p> <p><i>Inactive</i> logpoints are ignored during program execution.</p>
<i>Active for User</i>	<p>You have the option to restrict logpoint activation for a specific user in your system. Using this restriction you can ensure that log entries will only be evaluated when the specific user runs the corresponding program.</p> <p><b>Default Value:</b> * (activation for all users)</p>
<i>Active on Server</i>	<p>You have the option to activate the logpoint</p> <ul style="list-style-type: none"> <li>○ Globally, for all servers</li> <li>○ For specific server instances in the system</li> </ul> <p><b>Default Value:</b> All Servers</p>
<i>Active Until</i>	<p>Point of time when the logpoint will be deactivated automatically (activation expiry)</p> <p><b>Default Value:</b> 24 h</p> <div style="border: 1px solid #ccc; padding: 10px; background-color: #f9f9f9;"> <p><b>i Note</b></p> <p>In addition to this implicit deactivation, you have the option to deactivate the logpoint explicitly at any time: See also: <a href="#">Deactivating Logpoints</a></p> </div>
<i>Delete Logpoint After Deactivation [days]</i>	<p>For an active logpoint: number of days after activation expiry when the logpoint will be deleted automatically.</p> <p>For an inactive logpoint: the period starts with the day of the last logpoint change.</p> <p><b>Default Value:</b> 7 days</p> <div style="border: 1px solid #ccc; padding: 10px; background-color: #f9f9f9;"> <p><b>i Note</b></p> <p>In addition to this implicit deletion, you have the option to delete the logpoint explicitly: See also: <a href="#">Deleting Logpoints</a></p> </div>
<i>Max. Log Events for Internal Session</i>	<p>To ensure stable system performance, you can limit the maximum number of evaluations for the particular dynamic logpoint within the current internal session (roll area). Evaluation then stops after the maximum number of log events.</p> <p><b>Default Value:</b> depends on the activity selected</p>

4. Confirm your entries with *Finish*.

## Results

You have set a dynamic logpoint that is active for the specified code line. After the logpoint is created, a dedicated marker is visible in the ruler of the source editor.

```
5
▶ 6 select CONNID, CARRID from sbook into table @data(lt_flight)
7   where CARRID = 'LH'.
8
```

Marker for the active logpoint(s) is displayed in the ruler of the source editor

In addition, the *Logpoints* view will be opened where you can display and manage dynamic logpoints defined for each of your ABAP projects.

→ Remember

This logpoint applies whenever you start the corresponding ABAP program.

### 5.3.4.1.1 Conditions in Logpoints

You have the option to specify a condition for a dynamic logpoint.

The condition is evaluated at runtime, whenever the source code position of the dynamic logpoint is reached. If the condition is fulfilled (evaluates to true), the dynamic logpoint delivers results according to the content of the **KEY** and **FIELDS** entries of the specific dynamic logpoint. Otherwise, if the condition is not fulfilled, the logpoint provides no results at all.

 Note

An empty condition always evaluates to true.

The condition has an ABAP-like syntax, where you can use ABAP data objects that are valid at the source position at runtime and also the following ABAP operators:

- Standard comparison operators: **EQ,=,NE,<>,><,<LT,>,GT,<=,=<,LE,>=,=<,GE**
- Operators for string analysis: **CO,CN,CA,NA,CS,NS,CP,NP**
- Special unary operators: **IS INITIAL, IS BOUND, IS ASSIGNED**

Similar to ABAP syntax, you can also use the operators **NOT, AND, OR** and brackets to combine elementary conditions.

## Examples

```
sy-subrc <> 0
oref is bound and oref->attr = 'abc'
```

```
( itab[1]-string is initial or not itab[1]-string co '0123456789' ) and  
( itab[2]-string is initial or not itab[2]-string co '0123456789' )
```

### 5.3.4.1.2 Key Definition in Logpoints

Similar to the SUBKEY addition of the (static) LOG-POINT statement in ABAP, the key definition in dynamic logpoints is used for aggregation of the result set. While in static logpoints the SUBKEY has to be specified using a single character-like ABAP data object, in dynamic logpoints you can specify a list of data objects. For the key definition of the dynamic logpoint, you can enter a list of ABAP data objects (including literals), which are valid at the current source code position at runtime. The particular data objects in the list have to be separated by a SPACE or comma. In addition, you can specify special built-in functions, which may be called to execute a specific functionality (for example: trace). Each built-in function has a specific return value that is added to the key definition.

A syntactical validation of the given key definition is performed, when the dynamic logpoint is created.

Logpoint events with identical KEY values are aggregated into one and the same log entry (result). A new log entry will only be created after the key value has changed.

**More on this:**

#### ! Restriction

**Do not use** references, internal tables, or deep structures in the key definition as they cannot be converted into a STRING representation. The resulting key of a dynamic logpoint is built by concatenating the STRING representation of all specified data objects in the key definition. The STRING representation of each data object is determined by following the ABAP conversion semantics.

#### Examples:

```
sy-uname sy-host  
name,street, city zip_code  
stack_hash( ), sy-uname
```

### 5.3.4.1.3 Field Values in Logpoints

Using this logpoint parameter, you can specify a list of ABAP data objects, except literals, that are valid at the current source code position at runtime. The particular ABAP data objects in the list have to be separated by a SPACE or comma.

According to the FIELDS addition of the (static) LOG-POINT statement in ABAP, the *Field Values* in dynamic logpoints are used in log results. One or multiple *Field Values* in dynamic logpoints are written for each aggregated log entry.

A log entry is only updated for a logpoint event with identical *Key Definition*. Otherwise, if the *Key Definition* of the log event differs from the previous one, a new log entry is created.

**More on this:**

In contrast to the *Key Definition*, you can also specify internal tables and deep structures as data objects for *Field Values*. In addition, you can add special built-in functions to the results list. Each built-in function has a specific return value that is added to the log results.

**! Restriction**

**You cannot use** literals or any kind of references (TYPE REF TO <object> or TYPE REF TO DATA) as *Field Values* for logpoints.

**→ Remember**

In case of aggregated log entries the log result contains the values of the last log event.

**Examples:**

```
sy-uname sy-host
name, street, city zip_code
stack( ), itab
```

## 5.3.4.2 Displaying and Managing Logpoints

You can use the *Logpoints* view to display and manage dynamic logpoints defined for each project.

The *Logpoints* view provides you with an overview of the dynamic logpoints that have been created by a predefined user (default: the user currently logged on) of all ABAP projects in the workspace. For each ABAP project, a list of all dynamic logpoints that have been defined by a user is displayed. For each logpoint, you can view the details of the ABAP program and the location affected, as well as the status information.

Project / Master Program / Location	Activ...	Last ...	C...	C...	De...	Log Ev...	Expires
sys_y_sysnr_user_en [user filter: ]							
sys_x_sysnr_users_en [user filter: ]							
P Z_DLP_DEMO_KEYS_AND_FIELDS							
P Z_DLP_DEMO_CALL_STACK							
P Z_DEMO_CALL_STACK							
Z_DEMO_CALL_STACK (Program) / Line 19	24.11.1...	24.11....	BA...	B...		7	25.11.14
P AB_TEST_MESSAGE_TRUNCATED							
AB_TEST_MESSAGE_TRUNCATED (Program) / Li	21.11.1...	20.11....	BA...	B...	me...	1	21.11.14
AB_TEST_MESSAGE_TRUNCATED (Program) / Li	21.11.1...	20.11....	BA...	B...	call...	3	21.11.14

Logpoints view

## Activities on project nodes:

For each ABAP project, you can perform the following actions using the context menu in the view:

- Displaying logpoints from another user  
Choose the context menu option [Change User Filter...](#) and enter a desired user name to get a list of all logpoints that have been created or changed by the user in the system.
- Collecting log results  
By choosing the context menu [Collect Logs from All Servers](#), you have the option to explicitly trigger the transferring of the logging data from all servers of the system. The log results will be automatically updated on the [Logpoints](#) view.

## Activities on nodes for logpoints:

You can perform the following actions using the context menu:

- Collecting Log Results
- [Changing Logpoints \[page 703\]](#)
- [Copying Logpoints \[page 704\]](#)
- [Deactivating Logpoints \[page 704\]](#)
- [Activating Logpoints \[page 705\]](#)
- [Deleting Logpoints \[page 706\]](#)

### 5.3.4.2.1 Changing Logpoints

You can use the properties dialog to view or change properties of existing logpoints.

#### Context

When creating a dynamic logpoint, you specify the following details:

- The **logpoint definition** (what is the logpoint condition, whether and how results should be aggregated, and so on)
- The **activation status** of the logpoint.

Sometimes you might want to change the logpoint at a later point in time however.

##### Remember

Starting with the ADT version 2.83, you have the option of changing the logpoint definition as well. Note however that doing this deletes all previously collected log entries.

#### Procedure

To change the logpoint definition:

1. Select the node for the relevant logpoint in the *Logpoints* view.
- ##### Tip
- Alternatively, you can access the logpoint definition from the ruler in the source code editor.
2. Open the context menu and choose *Change Logpoint...*
  3. In the dialog screen that appears, change the required settings and press *Finish* to save your changes.

#### Results

All previously collected log entries of the changed logpoint are deleted. You can reproduce the deletion in the *Logpoints* view, since the number of *Log Events* is reset (after refresh) to 0 for the related logpoint.

#### Related Information

[Displaying and Managing Logpoints \[page 701\]](#)

[Viewing Logs \[page 706\]](#)

## 5.3.4.2.2 Copying Logpoints

### Context

Starting with the ADT version 2.83, you have the option of using copy, cut and paste functions on logpoints.

### Procedure

To create a logpoint using copy and paste:

1. Select the node for an existing logpoint - in the *Logpoints* view or in the ruler of the source code editor.
2. Open the context menu and choose *Copy Logpoint*.

#### Tip

You can also copy the definition and the activation status of an existing logpoint into the logpoint clipboard for later reuse as a kind of template. This can be particularly useful if you want to create multiple logpoints that are similar.

3. Set the cursor at the target line in the source code editor.
4. Open the context menu and choose *Paste Logpoint...*.
5. In the dialog screen that appears, enter the logpoint properties and choose *Finish*.

## 5.3.4.2.3 Deactivating Logpoints

### Context

Whenever you wish to suppress logging during program execution, but do not yet want to delete the associated logpoint, you have the option to disable the logpoint. An inactive logpoint is ignored during program execution.

### Procedure

To deactivate an active logpoint from the *Logpoints* view...

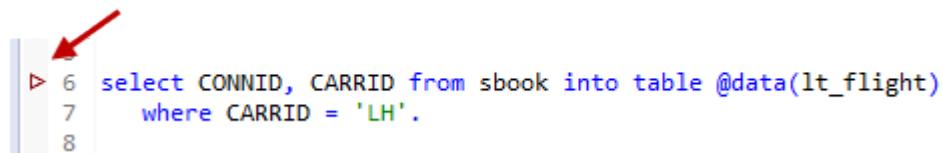
1. Select the node for the relevant active logpoint in the *Logpoints* view.
2. Open the context menu and choose *Deactivate Logpoint*

### → Tip

Alternatively, you can deactivate a logpoint from the ruler in the source code editor.

## Results

The new status is visualized with a marker at the logpoint position in the ruler of the source editor.



```
6  select CONNID, CARRID from sbook into table @data(lt_flight)
7  where CARRID = 'LH'.
8
```

Marker for inactive logpoint(s)

## Related Information

[Activating Logpoints \[page 705\]](#)

### 5.3.4.2.4 Activating Logpoints

## Context

Only active logpoints are evaluated at runtime, whereas inactive logpoints are ignored during program execution.

## Procedure

To activate an inactive logpoint in the *Logpoints* view...

1. Select the node for the relevant logpoint in the *Logpoints* view.
2. Open the context menu and choose *Change Logpoint Activation...*
3. In the properties dialog that appears, switch the *Status for Activation* to *Active*.
4. Confirm your change with *Finish*.

## Related Information

[Changing Logpoints \[page 703\]](#)

### 5.3.4.2.5 Deleting Logpoints

#### Context

In addition to implicit deletion, you have the option to delete logpoints explicitly.

#### Procedure

To delete a logpoint from the *Logpoints* view...

1. Select the node for the relevant logpoint in the *Logpoints* view.
-  Tip
- Alternatively, you can delete a logpoint from the ruler in the source code editor.
2. Open the context menu and choose *Delete Logpoint*
  3. In the confirmation dialog that appears, choose *Yes*.

#### Results

As a result of this operation, the dynamic logpoint is removed from the ABAP program and the associated log entries are deleted in the current system.

### 5.3.4.3 Viewing Logs

Logpoints are commonly used for writing a log entry when the relevant ABAP program reaches a certain point.

 Remember

The flow of a typical logging process includes the following steps:

1. In the source code of the program, position the cursor at the code line you wish to analyze in detail.
2. Set the logpoint at the selected source code line and specify the logpoint parameter.

3. Run the relevant ABAP program if this has not yet already been done.
4. Collect the currently available logs from all servers of the system.
5. View the relevant log entries.

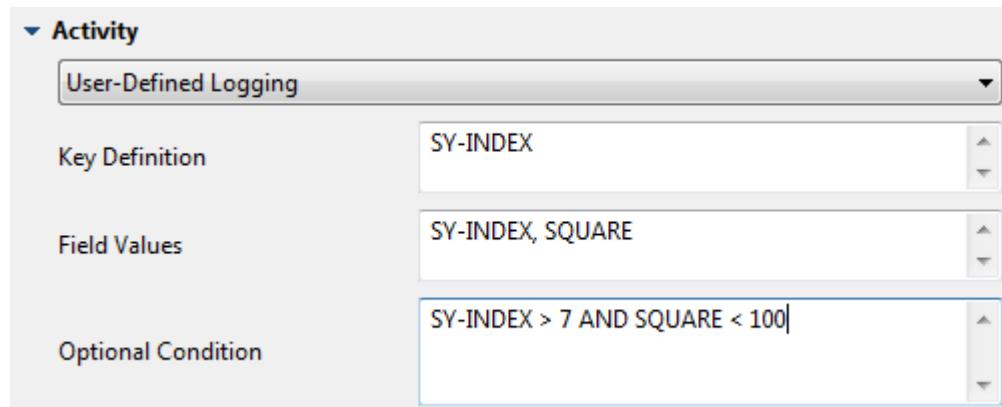
The two simple examples below show how the resulting log entries are to be interpreted.

## Example 1: User-Defined Logpoint Properties

Let's assume that the position to be examined in the program is a DO statement with which squares of numbers are to be issued.

```
REPORT Z_DEMO_SIMPLE_LP.
DATA square TYPE i.
...
DO 10 TIMES.
  square = ipow( base = sy-index exp = 2 ).
  cl_demo_output->write( |{ sy-index } { square }| ).
ENDDO.
...
```

As Activity, we select the option *User-Defined Logging*, and specify the *Key Definition*, *Field Values* and *Optional Condition* as follows:



User-defined logpoint

Due to the condition defined in the example, the result of the log is limited to two entries, which are represented by the key values [8] and [9]. The results display contains as uppermost node the *Generation Time* to which the values of the key are assigned (through the variable SY-INDEX). For each key value you have, in turn, the defined Field Values (SY-INDEX and SQUARE), whose results are issued in the *Field Value* column.

Resulting log entries

Generation Time / [SY-INDEX] / Field Name	Field Value	Log Events	Last Up...
20.11.14 16:21:23 [8] SY-INDEX	8	1	21.11.14...
20.11.14 16:21:23 [8] SQUARE	64		
20.11.14 16:21:23 [9] SQUARE	81	1	21.11.14...
20.11.14 16:21:23 [9] SY-INDEX	9		

## Example 2: Using Logpoints for Call Stack Analysis

The following program demonstrates the call of one and the same routine over two different call paths. With this simple example, you learn how these different call paths are mirrored in the log results and how you can use the log information for the call stack analysis.

```

REPORT Z_DEMO_CALL_STACK.
DO 10 TIMES.
CASE sy-index.
  WHEN 1.
    PERFORM delegate_call.
  WHEN OTHERS.
    PERFORM do_something.
ENDCASE.
ENDDO.
FORM do_something.
  MESSAGE 'Subroutine sucessfully executed' TYPE 'S'.
ENDFORM.
FORM delegate_call.
  PERFORM do_something.
ENDFORM.

```

As Activity we choose the option *Log Call Stacks* and add the *Optional Condition*, as follows, in order to restrict the number of result entries:

Activity

Log Call Stacks

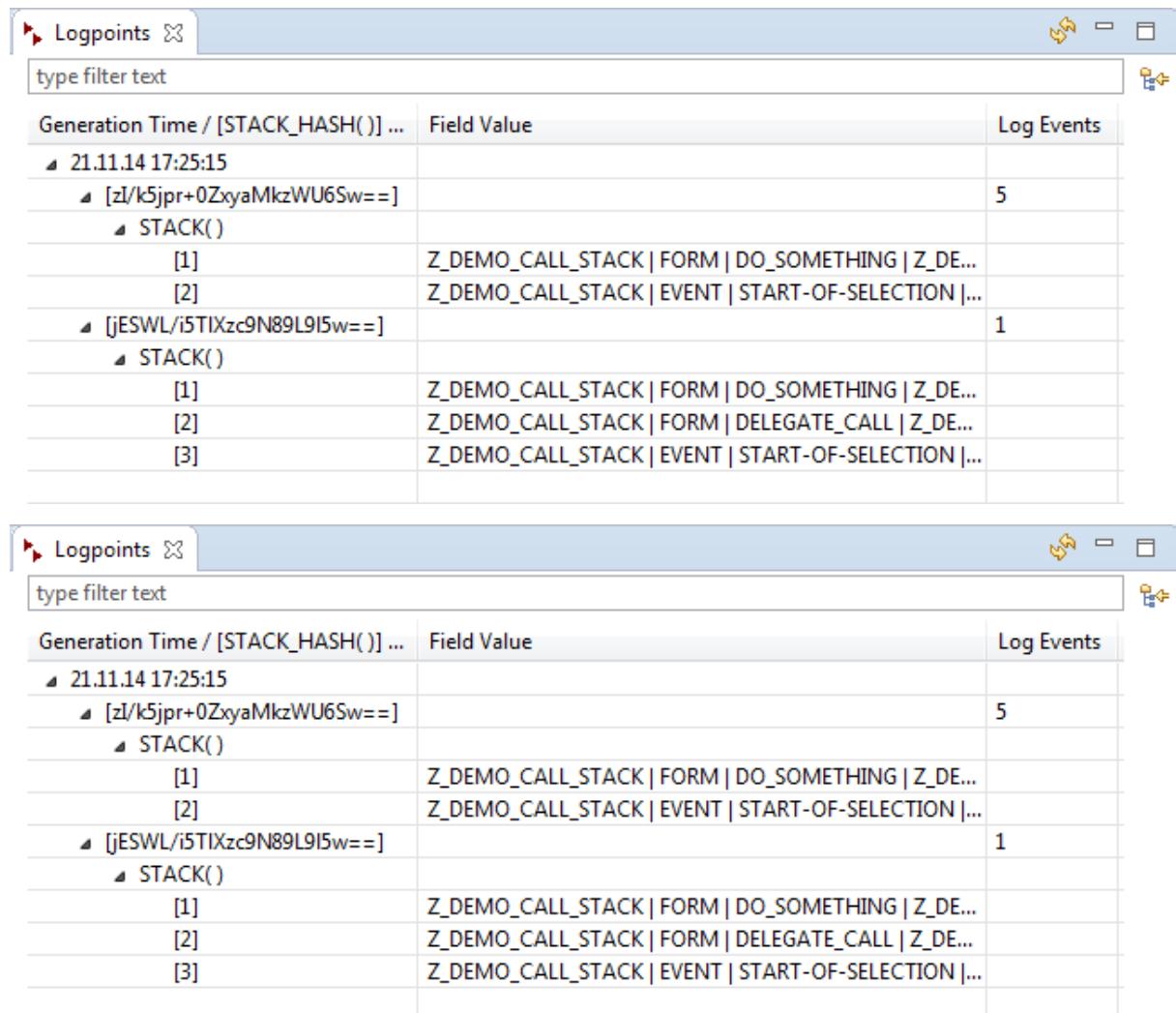
Optional Condition

SY-INDEX < 7

Log Call Stacks

As our result we get two different stacks, which represent the different call paths. In the first case (SY-INDEX = 1) 3 stack entries are counted, while for the other values (SY-INDEX > 1) 2 entries are counted. This corresponds exactly to the two call chains for the routine called in the example routine. As we limit the log

events to 6 by using the condition `SY-INDEX < 7`, we get a maximum number of 6 log events per program execution.



The screenshot shows two separate log entries from the SAP Logpoints viewer. Each entry is a row in a table with three columns: 'Generation Time / [STACK\_HASH()] ...', 'Field Value', and 'Log Events'.

**Logentry 1:**

Generation Time / [STACK_HASH()] ...	Field Value	Log Events
21.11.14 17:25:15		
↳ [z1/k5jpr+0ZxyaMkzWU6Sw==]		5
↳ STACK()		
[1]	Z_DEMO_CALL_STACK   FORM   DO_SOMETHING   Z_DE...	
[2]	Z_DEMO_CALL_STACK   EVENT   START-OF-SELECTION  ...	
↳ [jESWL/i5TIXzc9N89L9I5w==]		1
↳ STACK()		
[1]	Z_DEMO_CALL_STACK   FORM   DO_SOMETHING   Z_DE...	
[2]	Z_DEMO_CALL_STACK   FORM   DELEGATE_CALL   Z_DE...	
[3]	Z_DEMO_CALL_STACK   EVENT   START-OF-SELECTION  ...	

**Logentry 2:**

Generation Time / [STACK_HASH()] ...	Field Value	Log Events
21.11.14 17:25:15		
↳ [z1/k5jpr+0ZxyaMkzWU6Sw==]		5
↳ STACK()		
[1]	Z_DEMO_CALL_STACK   FORM   DO_SOMETHING   Z_DE...	
[2]	Z_DEMO_CALL_STACK   EVENT   START-OF-SELECTION  ...	
↳ [jESWL/i5TIXzc9N89L9I5w==]		1
↳ STACK()		
[1]	Z_DEMO_CALL_STACK   FORM   DO_SOMETHING   Z_DE...	
[2]	Z_DEMO_CALL_STACK   FORM   DELEGATE_CALL   Z_DE...	
[3]	Z_DEMO_CALL_STACK   EVENT   START-OF-SELECTION  ...	

Log entries with 2 different call stacks

### i Note

The prefined activity *Log Call Stacks* implicitly uses the built-in function as a key value. As `STACK_HASH()` delivers a different hash values for all different call stacks, only log events with different call stacks are aggregated.

### ! Restriction

Much like for static logpoints, the size of each data object stored in one single trace event of dynamic logpoints is limited by the **profile parameter** `abap/aab_log_field_size_limit`. The value of the profile parameter specifies the size in bytes. The default value is 1.024 bytes. When a log entry is generated, the content of each data object is truncated when this limit is reached. This restriction affects [Field Values in Logpoints \[page 700\]](#) of internal tables, large structures, long fields of type X/C, and long strings.

## 5.3.4.4 Resetting Logs for a Logpoint

### Context

You have the option of deleting the accumulated log entries of a dynamic logpoints without deleting the logpoint.

### Procedure

To delete the logs of a logpoint:

1. Select the node for the relevant logpoint in the Logpoints view.

 Tip

Alternatively, you can access the logpoint definition from the ruler in the ABAP source code editor.

2. Open the context menu and choose *Reset Logpoint Logs*.
3. In the dialog screen that appears, choose *Yes* to confirm that you want to delete the log entries.

### Results

All previously collected log entries are of the related logpoint are deleted. You can reproduce the deletion in the *Logpoints* view, since the number of *Log Events* is reset (after refresh) to 0 for the related logpoint.

### Related Information

[Displaying and Managing Logpoints \[page 701\]](#)

[Viewing Logs \[page 706\]](#)

## 5.3.5 Analyzing ABAP Runtime Errors

### Prerequisites

The program that you are running has just terminated because of a runtime error and has written a short dump.

The debugger is enabled in ADT. To check whether the debugger is enabled, right-click the relevant project in Project Explorer    .

The icon  in the main toolbar is disabled.

## Context

This section shows you how to display the information about the short dump and use it to analyze the error.

If an ABAP program fails with a short dump, the system responds by displaying the following dialog:



Runtime error dialog

From this dialog, you can do the following:

## Procedure

- [Display the runtime error \[page 711\]](#) by clicking *Show*.
- [Analyze the runtime error with the debugger \[page 714\]](#) by clicking *Debug*.

The system notifies you of a short dump even if the short dump occurs during an *ABAP Unit* run. [\[page 569\]](#). You can navigate to the short dump from the *ABAP Unit* view in the *Failure Trace* by clicking the link *Show Runtime Error*.

## Related Information

[Displaying ABAP Runtime Errors \[page 711\]](#)

[Analyzing ABAP Runtime Errors with Debugger \[page 714\]](#)

[Selecting ABAP Runtime Errors for a Feed \[page 714\]](#)

[Subscribing to Feeds from ABAP Repository \[page 488\]](#)

### 5.3.5.1 Displaying ABAP Runtime Errors

As of AS ABAP 7.53 runtime errors can be displayed in *ABAP Runtime Error Viewer*.

You can open *ABAP Runtime Error Viewer* using different entry points:

- If a runtime error occurs when executing ABAP programs, a dialog appears in the bottom right corner. Click *Show* to display the error.
- In the *Feed Reader* view, double-click the runtime error entry.
- If a runtime error occurred during an ABAP Unit test run, click the link *Show Runtime Error* in the *Failure Trace* section of the *ABAP Unit* view.

## ABAP Runtime Error Viewer

**[UIA] Runtime Error: TYPELOAD\_NEW\_NTAB\_VERSION 17.05.2018 12:29:14 A2ETEST**

**Header Information**

Short Text Data type "T100A" was changed at runtime.  
 Runtime Error TYPELOAD\_NEW\_NTAB\_VERSION  
 Program CL\_ADT\_MESSAGE\_CLASS\_API=====CP  
 Date/Time 17.05.2018 12:29:14 (System)  
 User A2ETEST  
 Client 000  
 Host Idai4uia\_UIA\_00

**Error analysis**

Data type "T100A" was used at least twice while the program was running. Between these two usages, data type "T100A" was changed in such a way that there are structural differences between the two versions of the type. Since the first version cannot be deleted from the roll area, the new version cannot be used.

**Information on where terminated**

The termination occurred in ABAP program or include "CL\_ADT\_MESSAGE\_CLASS\_API=====CP", in "CREATE". The main program was "SAPMSSY1".

In the source code, the termination point is in line 96 of include "CL\_ADT\_MESSAGE\_CLASS\_API=====CM001".

**Source Code Extract**

```

82  IF NOT iv_transport_request IS INITIAL.
83    lv_success = transport(
84      iv_name = iv_name
85      iv_package = iv_package
86      iv_transport_request = iv_transport_request
87      iv_scenario = cl_adt_message_class_api->gv_scenario_msg_class ).
88  IF lv_success = abap_false.
89    RETURN.
90  ENDIF.
91 ENDIF.
92
93
94  cl_adt_message_class_api->gv_access_2_t100_running = abap_true.
95  MODIFY ('T100') FROM TABLE lt_t100.
96  >> MODIFY ('T100A') FROM ls_t100a.
97  MODIFY ('T100T') FROM ls_t100t.
98  MODIFY ('T100U') FROM TABLE lt_t100u.
99  cl_adt_message_class_api->gv_access_2_t100_running = abap_false.
100
101 COMMIT WORK.
102
103 rv_success = abap_true.
104
105 ENDMETHOD. "create

```

**Active Calls/Events**

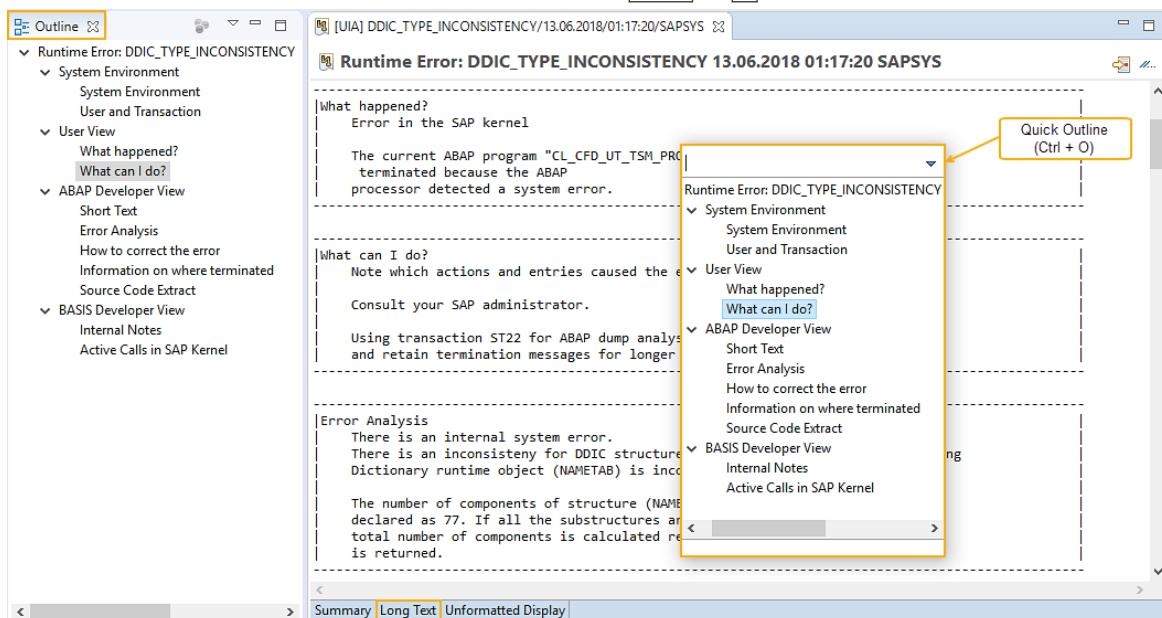
No.	Event	Program	Include	Line
10	CREATE	CL_ADT_MESSAGE_CLASS_API=====CP	CL_ADT_MESSAGE_CLASS_API=====CM001	96
9	DO_CREATE_CHILD	CL_ADT_MC_RES_CONTROLLER=====CP	CL_ADT_MC_RES_CONTROLLER=====CM005	59
8	POST	CL_NB_ADT_REST_RESOURCE=====CP	CL_NB_ADT_REST_RESOURCE=====CM001	53
7	IF_REST_HANDLER~HANDLE	CL_ADT_REST_RESOURCE=====CP	CL_ADT_REST_RESOURCE=====CM004	28
6	IF_REST_HANDLER~HANDLE	CL_REST_ROUTER=====CP	CL_REST_ROUTER=====CM003	60

**Summary tab in ABAP Runtime Error Viewer**

From the toolbar, you can navigate to the source code position where the dump occurred using the  icon and share the link to the dump using the  icon.

There are three tabs on the bottom of the editor:

- *Summary* provides the following information:
  - Header information
  - Error analysis
  - Information on where the program terminated
  - Source code extract  
The line in which the error occurred is highlighted. Click the line to navigate to the error in the source code.
  - Active calls/events
- The *Long Text* tab provides the full dump information in the same format as is familiar from transaction ST22. It supports the features *Outline* and *Quick Outline* (**Ctrl** + **O**).



#### Long Text Tab in Runtime Error Viewer

- The *Unformatted Display* tab provides the dump information in a technical format that is only necessary for special analysis situations.

## Related Information

[Analyzing ABAP Runtime Errors with Debugger \[page 714\]](#)

[Selecting ABAP Runtime Errors for a Feed \[page 714\]](#)

## 5.3.5.2 Analyzing ABAP Runtime Errors with Debugger

You can use the debugger to inspect a dump only if you have started a program (executable program, class, and so on) from the ABAP Development Tools, and it has terminated with a short dump.

While the dialog is shown, the program context still exists. Click *Debug* to start an analysis with the post-mortem debugger.

Since the program has been terminated, you cannot use the stepping function in the post-mortem debugger. However, all of the variables and other runtime information in the program context are available.

You can quit the debugger by clicking the icon  in the main tool bar.

### Related Information

[Displaying ABAP Runtime Errors \[page 711\]](#)

[Selecting ABAP Runtime Errors for a Feed \[page 714\]](#)

[Debugging ABAP Code \[page 613\]](#)

## 5.3.5.3 Selecting ABAP Runtime Errors for a Feed

### Context

When you subscribe to feeds ([Subscribing to Feeds from ABAP Repository \[page 488\]](#)), you can define rules for displaying ABAP runtime errors.

### Procedure

1. In the *Feed Reader* view in ADT, select the relevant project.
2. From the context menu, select *Add feed query....*

The window for a new feed query is opened.

#### Info

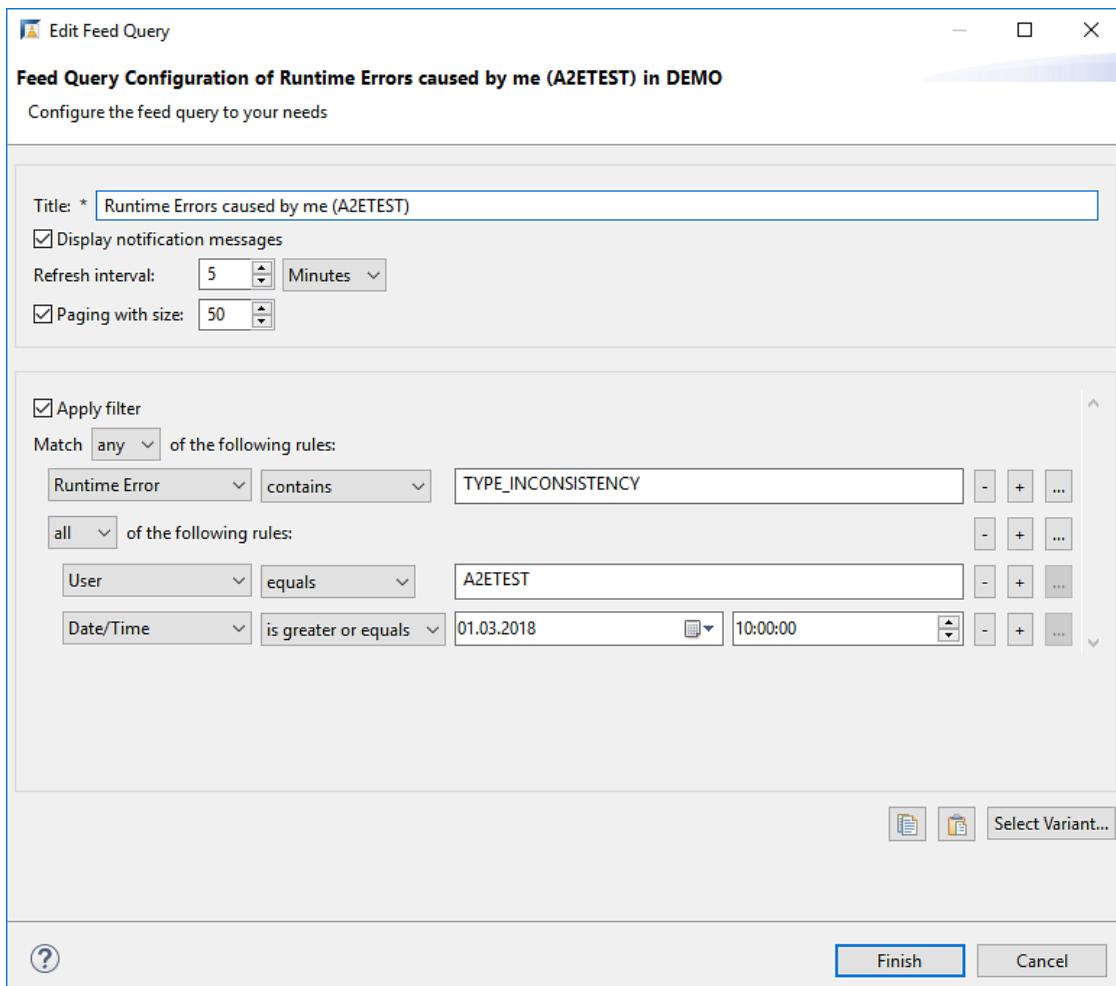
The project that you selected in the *Feed Reader* view is automatically selected. You can change it if needed.

3. Choose *ABAP Runtime Errors* and select *Next*.

The Feed Query Configuration dialog is opened.

4. Add criteria for choosing the runtime errors for the feed. Apply the filter and specify the rules for the match. Consider the following:

- Select **Any** if you want to join rules by OR logic. The feed reports all short dumps that meet any of the rules.
- Select **All** if all rules must apply and you want to join them by AND logic.
- To add a new condition to the same rule, use the  icon.
- To add a new rule, use the  icon.
- To delete the rule or condition, use the  icon.



An example of a feed query: Report any runtime errors that a) contain TYPE\_INCONSISTENCY in their names or b) occurred in the session of the A2ETEST user on 01.03.2018 at 10:00 or later.

## Results

If you do not define a filter, the feed reports all runtime errors that occur in the ABAP project.

### → Tip

You can quickly define identical feeds for runtime errors in your ABAP Projects. To do so, first set up one feed. Then choose the [Copy query selection to clipboard](#) icon . Now create a new runtime error feed or

choose [Edit feed...](#) for an existing feed from the context menu. On the Feed Query Configuration dialog, choose the *Paste query selection from clipboard* icon . The ADT applies the configuration that you copied to the new feed.

## Related Information

[Analyzing ABAP Runtime Errors with Debugger \[page 714\]](#)

[Displaying ABAP Runtime Errors \[page 711\]](#)

## 5.4 Enabling Accessibility Features in ADT

Accessibility features help ABAP developers with disabilities to work with and to use the benefits of ABAP Development Tools (ADT).

### Context

The central idea is to provide the functionalities of ADT to all ABAP developers, independently of possible disabilities. So, SAP has overhauled the application UI in order to enable tools to read out text from the user interface. In addition, Eclipse and SAP provide specific configurations and preferences to adjust ADT to your needs.

In the current client version, you have following possibilities to utilize accessibility features:

- [Setting Configurations and Preferences \[page 717\]](#)
- [Using Screen Readers \[page 720\]](#)

## Related Information

[Accessibility Features in ADT \[page 113\]](#)

## 5.4.1 Setting Configurations and Preferences

You can define preferences in order to adjust ABAP Development Tools (ADT) to the required usage.

### Context

#### i Note

The following preference paths are created on the basis of Eclipse 4.4 (Luna). They might be different in later versions.

### Procedure

To open the *Preferences* page, choose    from the menu bar.

## 5.4.1.1 Resizing Fonts

You can resize fonts for displaying text in bigger letters to improve readability.

### Context

#### → Recommendation

SAP recommends that you use the standard Eclipse themes and the Windows display settings to resize fonts in Eclipse.

### Procedure

1. In the    *Preferences* page, choose one of the following themes from the list box:
  - *Default*
  - *Classic*
  - *Windows 7*

#### i Note

Use one of the themes from the list above. Other themes might not work for all controls.

2. Open the MS Windows display settings.
3. Enlarge the font size to 150%, for example.
4. Restart your Eclipse client.

## Results

In ADT, the larger font size should now be available on all text-based controls.

## Related Information

[Setting Configurations and Preferences \[page 717\]](#)

### 5.4.1.2 Setting Colors

You can adjust the color of the displayed text to suit your own needs.

## Context

Eclipse provides options for changing the colors - for example, for the following features:

- Syntax Coloring (ABAP keywords in general and at keyword level)
- ABAP Profiler
- ABAP Code Coverage

If required, you can adopt the provided preferences as follows:

## Procedure

1. To configure syntax coloring in general, for example, for ABAP source code and CDS/DDL objects, open the  [General](#)  [Appearance](#)  [Colors and Fonts](#)  [Preferences](#) page.
  - a. Choose the theme from the *Theme* listbox.
  - b. Then choose the corresponding tree element and define the color.
2. To configure colors at ABAP keyword level or the sequence of several ABAP keywords used within a statement, open the  [ABAP Development](#)  [Editors](#)  [Source Code Editors](#)  [ABAP Keyword Colors](#)  [Preferences](#) page.
  - a. Change the corresponding color.

3. To configure the colors of annotation types, open the  [General > Editors > Text Editors > Annotations > Preferences](#) page.
  - a. Change the color of the corresponding annotation type.

## Related Information

[Changing the Font Color of Texts \[page 310\]](#)

[Changing The Color of Individual ABAP Keywords \[page 311\]](#)

[Changing Colors of Annotation Types \[page 313\]](#)

### 5.4.1.3 Setting Higher Contrast

In Eclipse, you can work with the higher contrast mode of your operating system. This enables you to improve readability, for example.

## Context

When you are using the High Contrast Mode of your operating system, SAP recommends that you adapt the preferences, as follows, to suit your needs:

## Procedure

1. To open the [Preferences](#) page, choose  [Windows > Preferences](#) from the menu bar.
2. Open the  [General > Editors > Text Editors > Preferences](#) page and deselect the *Highlight current line* checkbox.
3. To adapt the appearance of the ABAP occurrence annotations (ABAP Occurrences, ABAP Indirect Occurrences, ABAP Tentative Occurrences and ABAP Write Occurrences), open the  [General > Editors > Text Editors > Annotations > Preferences](#) page.

By default, the ABAP occurrence annotations are highlighted with a grey background color.

- a. To enable these annotations - for example, for the high contrast mode, choose *Underlined* in the *Text as* dropdown box.
- b. To contrast annotations from the background, define the underline color by choosing the corresponding color in the *Color* field.

## 5.4.1.4 Enabling Screen Readers

A screen reader is an additional application that transforms displayed text into an electronic voice. As a result, the user can follow this voice.

To activate the corresponding accessibility feature, select the following checkboxes:

Accessibility Feature	Preferences Path	Preferences Setting
Avoids appearance of the progress indication dialogs, which require some time to get read by the screen reader	▶ <a href="#">General</a> ▶	<i>Always run in background</i> checkbox
Enables the screen reader to read more information in table controls, and so on	▶ <a href="#">ABAP Development</a> ▶ <a href="#">Additional Accessibility Information</a> ▶	<i>Enable additional accessibility information</i> checkbox
Opens status messages in a dialog	▶ <a href="#">ABAP Development</a> ▶ <a href="#">Notifications</a> ▶	<i>Show error status messages also in a dialog box</i> and <i>Show information status messages also in a dialog box</i> checkboxes

To reduce the number of entries in the main menu and toolbar, you can enable or disable entries in the [Customize Perspective](#) dialog. Open it through ▶ [Window](#) ▶ [Customize Perspective...](#) ▶ from the menu bar. Then, select the relevant checkboxes in the tabs *Tool Bar Visibility*, *Menu Visibility*, and *Shortcuts*.

## 5.4.2 Using Screen Readers

A screen reader is a software application that identifies and interprets textual content on the UI. A screen reader reads this content to the user through an audio function.

### Context

ABAP Development Tools (ADT) provides features for working efficiently with a screen reader. Eclipse also provides shortcuts for many operations.

To display the list of available shortcuts, perform the following steps:

### Procedure

1. To open a window showing a list of all shortcuts that are currently active, choose `Ctrl` + `Shift` + `L`.

#### i Note

Usually this list is too long to read through all of the shortcuts.

2. To open the *Keys Preference* page, choose the same shortcut again.

Here the cursor is automatically positioned on a filter field.

3. Search for the command you are looking for.

**i Note**

If you do not find a shortcut for a particular command, you can define your own shortcut in this *Preference* page. To do this, select the command in the *Command* list. In the *Binding* entry field, position the cursor and choose the desired shortcut on the keyboard. Choose *Ok* to apply the new shortcut.

## Related Information

[Working with Editors and Views Side by Side \[page 721\]](#)

[Working with Maximized Editor Area and Views in Background \[page 724\]](#)

[Creating Development Objects \[page 725\]](#)

[Opening Development Objects \[page 726\]](#)

[Reading in ABAP Source Code Editors \[page 727\]](#)

[Editing in ABAP Source Code Editors \[page 733\]](#)

[Unit Testing and Coverage Analyzer \[page 736\]](#)

### 5.4.2.1 Managing Editors and Views in Eclipse

Eclipse is a development environment that provides a large number of views and editors side by side. This kind of display requires knowledge of the keyboard controls in order to focus the required view.

#### 5.4.2.1.1 Working with Editors and Views Side by Side

Eclipse offers commands to work efficiently with the keyboard when many editors and views are opened side by side.

## Context

By default, Eclipse opens editors stacked in the *Editor* area. Around this *Editor* area, views are arranged in *View* areas that are grouped on the left, bottom, and right. Views might be stacked in these *View* areas.

## Procedure

Use the  [Window](#)  [Navigation](#)  menu to find functions to manage editors and views.

### Note

Some of these functions will be explained in this section.

## 5.4.2.1.1.1 Switching Editors

You can navigate between several opened editors.

## Procedure

### 1. Using the *Switch to Editor* dialog:

- a. To open the *Switch to Editor* dialog, choose the `Shift` + `Ctrl` + `E` shortcut.
- b. Select the editor you want using the arrow keys.
- c. To switch to the selected editor, choose `Enter`.

### Note

You can also use this dialog to manage your open editors. You can do this by proceeding as follows:

1. Select the editors you want to close.
2. Choose the *Close selected editors* button.

### 2. Using the *Next Editor* command:

- a. To switch to the next editor(s), choose `Ctrl` + `F6`.
- b. To open an editor switcher window with a list of opened documents, choose `Ctrl` + `F6` again.
- c. To move onto the selection of the next list item, hold the `Ctrl` key and choose `F6`.  
The editor title will be announced.
- d. As soon as you have found the corresponding list item, release `Ctrl`

## Results

The selected editor will receive focus.

## 5.4.2.1.1.2 Switching Views

You can navigate between several opened views.

### Context

You can switch between views using the following shortcuts:

### Procedure

1. To open a view switcher window with a list of views given in the last used order, choose `Ctrl` + `F7`.

#### i Note

As long as you choose and hold the `Ctrl` key and choose again `F7` key. The selection will go to the next view until you release the key.

2. To set the selected view in focus, release the `Ctrl` key.
3. To return the focus immediately from any view to the last used editor, choose `F12`.

#### i Note

If this editor is invisible, it will automatically become visible after choosing `F12`.

The focus will move to the last used editor.

4. To be able to take a look at properties or variables while you are in debugging mode, assign your own key combinations for these purposes.

#### i Note

Eclipse does not provide default key combinations to move focus into the properties view or variables view.

#### → Recommendation

SAP recommends adding customer-specific shortcuts for frequently used views to the corresponding *Show View* commands.

5. To open the *Show Views* dialog, choose `Alt` + `Shift` + `Q`.

Each view has a character assigned in this dialog. Example: *Show View (Breakpoints)*: B

To open the corresponding view directly without searching through the list, choose `Alt` + `Shift` + `Q`. As soon as the dialog is opened, choose this character.

6. To search and open a view from the *Show View* dialog, choose `Enter`.

### 5.4.2.1.1.3 Switching Perspectives

#### Context

You can switch between perspectives using the following shortcut:

#### Procedure

To switch to the next or previous perspective, choose the `Ctrl` + `F8` and `Shift` + `Ctrl` + `F8` shortcuts, respectively.

## 5.4.2.1.2 Working with Maximized Editor Area and Views in Background

#### Context

##### i Note

All editors are stacked in the editor area by default. If you put one of the editors into full screen, the entire editor area is put into full screen.

#### Procedure

1. To toggle to full screen, choose the `Ctrl` + `M` shortcut for each editor or view.
2. To restore the windows to the previous state, choose the same shortcut again.
3. To switch to another editor, open the development object using the `Ctrl` + `Shift` + `A` shortcut.
4. To open it also in full screen mode or to navigate to it, choose the `Ctrl F6` shortcut  
If you now open a view, for example, using `Ctrl` + `F7`, the view is opened and focused. The view now behaves much like a modal dialog.
5. To close a view, choose `Esc`.

#### Results

The focus returns to the editor area.

## 5.4.2.2 Creating Development Objects

### Context

You have, for example, the following possibilities to create new objects:

- From the *Project Explorer* tree using the context menu
- Through the menu bar
- By using the shortcut `Ctrl` + `N`

#### i Note

In the creation wizards, an asterisk \* indicates a mandatory input field.

### Procedure

1. In the input fields referencing another object , you can choose `Ctrl` + `Space` to use the content assist functionality.  
Then, the list of possible entries is opened.
2. To focus the list, choose the `Tab` key.
3. To select the relevant entry, choose `Enter`.

### Results

The selected entry is added to the input field. Continue choosing the relevant input entries.

### Related Information

[Creating Development Objects \[page 156\]](#)

### 5.4.2.3 Opening Development Objects

To edit an existing development object, you need to open it from the database.

#### Context

You can open development objects, for example, from the *Project Explorer* view or through the *Open ABAP Development Objects* dialog.

In the *Project Explorer* view, you will find your ABAP projects with a tree of ABAP packages and objects.

#### i Note

The content of the project explorer tree is loaded asynchronously. Occasionally, the screen reader announces a *Loading repository tree...* node. This node will be replaced by the real content as soon as it is available.

#### Procedure

1. To open a development object through the *Project Explorer* view, navigate to the corresponding node in the tree and choose *Enter*.
2. As an alternative to the *Project Explorer*, you can use the *Open ABAP Development Objects* dialog. For this, choose the `Ctrl` + `Shift` + `A` shortcut.

#### Results

The selected development object(s) is (are) opened in the corresponding editor.

#### Related Information

[Opening Development Objects \[page 159\]](#)

[Enabling Screen Readers \[page 720\]](#)

## 5.4.2.4 Reading in ABAP Source Code Editors

### Related Information

[ABAP Source Code Editor \[page 57\]](#)

### 5.4.2.4.1 Navigating with the Keyboard

You can use the keyboard to navigate, for example, through source code elements of an opened development object.

### Context

The following shortcuts help you to navigate efficiently in source code:

### Procedure

1. To navigate to a method when the cursor is placed on a method call, choose **[F3]**.
2. To navigate back to the method call, choose **[Alt] + [Left]**.

#### i Note

In general, **[Alt] + [Left / Right]** allows you to navigate through the navigation stack.

3. To navigate from the beginning statement (like `method`, `if`, or `loop`) to the corresponding end statement (`endmethod`, `endif`, or `endloop`), use **[F3]**.
4. To navigate through annotations, choose the *Next Annotation (Ctrl .)* and *Previous Annotation (Ctrl ,)* commands.

#### i Note

An annotation can be, for example:

- A syntax error
- The beginning and end statement of a block (like `method / endmethod` or `loop / endloop`)
- The occurrence of ABAP variables

There are many different annotation types. You can find them on the **General** **Editors** **Text Editors** **Annotations** **Preferences** page. For each annotation you can select whether it should be included in the *Next / Previous Annotation* navigation.

#### → Recommendation

SAP recommends using these shortcuts, especially for the ABAP annotations (like block highlighting, occurrences, and coverage).

5. To navigate to the last edited location, choose **Ctrl** + **Q**.

### 5.4.2.4.2 Using the Ruler

Each source editor in Eclipse has a ruler column next to the source code. It displays information about the corresponding line. Currently, it is not possible to focus the ruler.

#### Context

The ruler displays the following information:

- Line number
- Possible breakpoint
- Bookmark
- Task
- Indicator for syntax errors or warnings
- Indicator for ABAP enhancements

The following methods can be used to access and change the ruler content:

#### Procedure

1. To find out the current line number, choose **Ctrl** + **L**.  
The [Go To Line](#) dialog is opened and shows the current line number in the focused input field.
2. To open the ruler tooltip, add a customer-specific shortcut, such as **Alt** + **Ctrl** + **F10**, to the [Show Ruler Annotation](#) command in the preferences.  
Use this shortcut to get information about breakpoints, bookmarks, or tasks on a line number where the focus is currently located at.
3. Alternatively, you can open and navigate to the [Bookmarks](#), [Breakpoints](#), or [Tasks](#) views.  
There you will find a list of these options for navigating to the related code line.
4. Use **Ctrl** + **F10** to open the ruler context menu from the editor.  
The context menu provides menu items for adding breakpoints, bookmarks, and tasks.

### 5.4.2.4.3 Using the ABAP Element Info View

The element info displays, for example, the type of a variable or the parameters of a method.

#### Procedure

To open the *ABAP Element Info* view, position the cursor on an identifier and choose `F2`.

#### Results

This will open and focus a popup showing the information available for this element. The screen reader starts reading the information. The information is shown as HTML.

##### i Note

When the cursor is located inside a method, you can retrieve the element info for this method through `Alt` + `F2`. A popup is then opened and focused, in this case showing the parameters of the method again.

#### Related Information

[Element Info \[page 33\]](#)

[Displaying Element Info in Source Code Editors \[page 328\]](#)

## 5.4.2.4.4 Reading Text from Error and Information Status Messages

A screen reader can read status messages for errors and other information that is displayed in dialogs.

### Context

To make such information accessible for the screen reader, you need to display it in a dialog. To do this, proceed as follows:

### Procedure

1. Open the *Preferences* page, choose   *Preferences* from the menu bar.
2. In the  *ABAP Development* page, navigate to the *Notifications* area.
3. Choose one or both of the following checkboxes:
  - **Show error status messages also in a dialog**
  - **Show information status messages also in a dialog**
4. Confirm with *OK*.

### Results

If there is a specific occurrence where a status or an error message needs to be displayed, its content is opened in a dialog. From here, the screen has access to the text on the user interface and can read it.

### Related Information

[Working with Messages and Message Classes \[page 275\]](#)

## 5.4.2.4.5 Navigating and Researching with the Outline

The outline displays the structure of the currently selected or edited object – for example, all its attributes, methods, and other members of a class.

### Context

The outline can be shown directly in the editor or in the *Outline* view. In this view, you can:

- Use filters
- Sort options from the toolbar
- Link with the navigation tools in the editor.

### Procedure

1. To access the outline in the editor, choose `Ctrl` + `O`.  
The outline is shown in a popup and the focus is directly set on the filter field.
2. To switch to the *Result* tree, choose the `Tab` key.  
The result is linked to the editor selection. The outline focuses on the member at the cursor position.
3. To display inherited members also, choose `Ctrl` + `O` again.
4. To navigate to a member, focus it in the list and choose `Enter`.

### Related Information

[Outline View \[page 80\]](#)

[Viewing the Outline \[page 332\]](#)

## 5.4.2.4.6 Using Syntax Highlighting

If you are not sure whether the word at the current cursor position is a keyword, identifier, comment or literal, you can use the shortcut of your screen reader for reading colors.

### Context

The Eclipse color settings might not be accessible through the screen reader.

The default color settings for ABAP source code are:

Color	Type
Blue	Keyword
Black	Identifier
Limegreen	Literal
Dodger blue	Number literal
Grey	Comment
Red	Error in syntax highlighting
Alice blue	Background of current line
Grey 83	Background of the ABAP occurrence marker

## Procedure

To read out the color, use the shortcut of your screen reader for reading colors. For example, use [Insert 5](#) for JAWS.

## Results

The screen reader reads the name of the foreground color and the background color.

### 5.4.2.4.7 Displaying and Changing Properties of an Object

You can find and edit the properties of an object (for example, description, author, created at, package, and so on) in the [Properties](#) view.

## Context

The [Properties](#) view is always linked to the editor that is currently open. When you open the [Properties](#) view, the focus is in the [General](#) tab on the [Package](#) field.

## Procedure

To access the tab control and switch to the [Specific](#) tab, use the [Shift](#) + [Tab](#) shortcut.

## 5.4.2.4.8 Comparing Source-Based Development Objects

You can compare a development object, for example, between two ABAP systems or with older versions of the object. This enables you to investigate differences between both versions.

### Context

You have the following options for comparing development objects:

- Use the *Compare with* context menu in the editor.  
Here you can select the comparison target (such as an ABAP project, revision history, or local history). A new editor that shows the two sources next to each other is then opened.
- To navigate to the next or previous difference, choose the `Ctrl` + `.` and `Ctrl` + `,` shortcuts.  
The cursor will be positioned on the line where the difference is located.
- To switch to the second source at the corresponding position, choose `Ctrl` + `Tab`.
- To return to the first source, choose `Ctrl` + `Shift` + `Tab`.
- To read the current line number, choose `Ctrl` + `L`.

You can edit directly in the compare editor. Just type into the editor or use the copy actions in the compare editor toolbar.

#### Note

No shortcuts are defined for these copy actions by default, but you can define your own in the *Preferences*.

### Related Information

[Comparing Source Code \[page 207\]](#)

[Using Screen Readers \[page 720\]](#)

## 5.4.2.5 Editing in ABAP Source Code Editors

You can adopt the specific functions and utilities for efficiently creating, editing, and navigating in ABAP source code in accordance with your needs.

### Related Information

[Editing ABAP Source Code \[page 294\]](#)

## 5.4.2.5.1 General Behavior

The following section describes the editing features that are the same in all editors:

### Procedure

1. An editor is locked immediately when you start editing. To unlock the editor again, choose `Ctrl` + `U`. You will be asked whether you want to discard your changes.
2. The syntax check is executed automatically in Eclipse. When you stop typing, a syntax check is triggered, and in normally less than a second the syntax check result is available in the *Problems* view.
3. To activate an object, choose `Ctrl` + `F3` or the context menu in the editor.
4. In Eclipse, the name of the pretty printer is *Source formatter*. You can execute the source formatter by choosing  *Source*   from the menu or the context menu of an editor, or by choosing the `Shift F1` shortcut.

### Related Information

[Formatting ABAP Source Code \[page 307\]](#)

## 5.4.2.5.2 Using Code Completion

In the *ABAP source code editor*, the code completion proposes valid ABAP keywords and identifiers to you that can be inserted within the source code at your current position.

### Procedure

1. To trigger code completion in the editor, choose `Ctrl` + `Space`.  
A popup with a list of possible entries at the current source code position is opened.
2. Focus the popup by choosing `Tab`.  
The screen reader announces the entries and you can access additional information for each entry by choosing `Tab` again. Now another popup with additional information in HTML is focused.
3. To get back to the code completion list, choose `Esc`.
4. To select one entry, choose `Enter`.
5. To select a method, for example, you can insert the whole method signature by choosing `Space` + `Enter`.

## Related Information

[Getting Support from the Content Assist \[page 295\]](#)

### 5.4.2.5.3 Using Quick Assists

Quick assists, like renaming an attribute or defining a local variable, help you to change ABAP source code in a semi-automated way.

#### Context

#### Procedure

1. To open the Quick Assist popup, choose `Ctrl` + `1`.
2. Then choose `Tab` to focus the popup.
3. Navigate through the list.

Like with code completion, there is an additional popup that shows more information for each quick assist. Currently, screen readers might not be able to access this popup in Eclipse. Therefore, ADT provides the [Quick Assist](#) view as an alternative. This view can be linked with the current source code position to show the available quick assists and the additional information at this position.

## Related Information

[Quick Assists \[page 85\]](#)

[Applying Quick Assists \[page 342\]](#)

## 5.4.2.6 Debugging ABAP Source Code

ABAP Development Tools (ADT) supports ABAP debugging. However, fewer operations than in the SAP GUI might trigger prolongation of the debug session timeout.

### Context

Eclipse does not interact with the debugger when you scroll through ABAP source code and navigate between different locations. If you do not interact with the debugger for some time, the common timeout settings of the system will terminate the debugged process.

#### i Note

The timeout is configured as the system profile parameter (`rdisp/max_wprun_time`). The value is set by default between 10 and 20 minutes.

### Procedure

To avoid losing the debugged process, interact with the debugger or trigger a dummy interaction such as the synchronization of breakpoints. To do this, choose `Ctrl` + `Alt` + `B` within any editor of the corresponding project.

### Related Information

[Debugging ABAP Code \[page 613\]](#)

## 5.4.2.7 Unit Testing and Coverage Analyzer

ABAP Unit is a unit-testing framework that is integrated into the ABAP language.

### Procedure

1. For unit testing:
  - a. Start unit tests through the *Run as* menu in the context menu of an editor or the *Project Explorer*. The focus will be set to the *Unit Test* view, where the results will be displayed in a tree as soon as they are available. Tests that have errors are marked with a specific icon.

2. For coverage measurement:
  - a. Start unit tests with coverage measurement through the *Coverage as* menu in the context menu of an editor or the *Project Explorer*.  
The coverage results are displayed in the *ABAP Coverage* view. The source code is colored with green and red background color and specific coverage annotations are added to the code.
  - b. If these annotations are included in the *Next annotation* navigation, choose `ctrl` + `.` to navigate to lines that have been covered only partially or not at all.

## Related Information

[Unit Testing in ABAP \[page 71\]](#)

[Evaluating ABAP Unit Code Coverage Results \[page 571\]](#)

# 6 Reference

## 6.1 Released Object Types

The following sections list the types of development objects that are supported in the context of ABAP Environment:

### Full Access

In ABAP Development Tools (ADT), you can **use**, **create**, **update**, and **delete** the following development objects:

Use Case	Development Object	O b j e c t N a m e
Core development	ABAP class	C L A S
	ABAP function group	F U G R
	ABAP function module	F U N C
	Interface	I N T F

Use Case	Development Object	
	Message class	M S A G
	Simple transformation (ST)	X S L T
Other ABAP Repository objects	ABAP packages	D E V C
	Whitelisting of Objects	A P I S
ABAP Dictionary	Domain	D O M A
	Data element	D T E L
	Database table	T A B L
	Table type	T T Y P

Use Case	Development Object	Object Name
	Lock object	E N Q U
	Structure	S T R C
Core Data Services (CDS)	DCL source	D C L S
	DDL source	D D L S
Connectivity	HTTP service	H T T P
Business services	Service consumption model	S R V C
	Service definition	S R V D
	Service binding	S R V B

Use Case	Development Object	
Business objects	Behavior definition	B D E F
	Behavior implementation	C L A S
Identity and access management (IAM)	Authorization check field	A U T H
	Authorization default values	S U S H
	Authorization object	S U S O
	Business catalog	S I A 1
	IAM app	S I A 6
	Restriction field	S I A 5

Use Case	Development Object	
	Restriction type	S I A 2
Service integration and communication	Communication scenario	S C O 1
	Inbound service	S C O 2

## Read-Only Access

In ABAP Development Tools (ADT), you can only **read** the following development objects:

Use Case	Development Object	Object Name
Core development	ABAP programs	PROG
ABAP Dictionary	ABAP type group	
Core Data Services (CDS)	CDS annotation definition*	DDLA
	Metadata extensions	DDLX

### i Note

\* This object type can only be created and edited by SAP.

## 6.2 Comparison of the ADT Features in the Context of ABAP Environment vs. ABAP Platform

The following tables contrast the feature set of ABAP Development Tools (ADT) that is provided in the relevant context:

- General Information [\[page 743\]](#)
- Defining Authorization and Access [\[page 743\]](#)
- Managing ABAP Projects [\[page 744\]](#)
- Developing ABAP Applications [\[page 745\]](#)
- Working with ABAP Tools [\[page 746\]](#)

### General Information

 Note that the following applies in the context of ABAP environment:

- The embedded SAP GUI is not supported, which means you create, update, edit, and delete development objects in their native integrated editors within ADT.
- A subset of the ABAP language version is supported. This helps you to focus using features and [released \[page 738\]](#) object types which are relevant for the SAP products within SAP Cloud Platform.

 Note that in the context of ABAP Platform, the ADT features mentioned below always relate to the latest back-end version.

### Defining Authorization and Access

This chapter compares the features and tool set for SAP administrators to set up an ABAP environment and to access the relevant ABAP back-end system:

Feature	SAP Cloud Platform ABAP Environment	ABAP Platform (On-Premise)
Identity & Access Management (IAM)	Using the administration apps on the SAP Fiori launch pad to configure the ABAP environment and to provide the business users with the relevant business roles and business catalogs	General administration tools, for example transaction PFCG or SU01, for accessing and securing the ABAP development process on the AS ABAP.

Feature	SAP Cloud Platform ABAP Environment	ABAP Platform (On-Premise)
Development authorizations	<p>In accordance with the developer role defined by SAP, an ABAP developer can, for example, do the following:</p> <ul style="list-style-type: none"> <li>• Create and release transport requests</li> <li>• Debug in read-only mode</li> <li>• Develop in the customer namespace</li> <li>• No administrative tasks are enabled within ADT</li> </ul>	<p>In accordance with the provided permissions defined by the authorization administrator of your company, an ABAP developer can, for example, do the following:</p> <ul style="list-style-type: none"> <li>• Create and release transport requests</li> <li>• Debug with change authorization</li> <li>• Develop in the customer namespace</li> <li>• Administrative tasks, such as terminating tasks using SM50, are enabled using the integrated SAP GUI within ADT</li> </ul>
Developing IAM artifacts	Development of authorization fields and authorizations objects is not enabled	Development of authorization fields and authorization objects is enabled using the integrated SAP GUI
	Aggregating authorizations in business catalog for usage in business roles	Aggregating authorizations in roles using transaction PFCG
	Using native integrated editors for creating access controls (DCL) to restrict access to business data in the context of Core Data Services (CDS)	

## Managing ABAP Projects

This chapter compares the features that are relevant when preparing, managing, and organizing development activities:

Feature	SAP Cloud Platform ABAP Environment	ABAP Platform (On-Premise)
Client-server communication	<ul style="list-style-type: none"> <li>• Communication is based on HTTP using an ABAP cloud project</li> <li>• Using HTTP</li> <li>• Authentication is browser-based and checked on the basis of the identity provider</li> <li>• Single-Sign-On (SSO) is only supported through an external browser</li> </ul>	<ul style="list-style-type: none"> <li>• Communication is enabled through Remote Function Call (RFC) using an ABAP project</li> <li>• Using Remote Function Calls (RFC) as the interface for communication with the ABAP system</li> <li>• Authentication is checked through the AS ABAP</li> <li>• Single-Sign-On (SSO) is supported</li> </ul>

Feature	SAP Cloud Platform ABAP Environment	ABAP Platform (On-Premise)
Creation of support tickets within the ADT client	Built-in option for creating customer incidents	No built-in option available
Project explorer	Customizable using ABAP Repository trees which display the following by default: <ul style="list-style-type: none"> <li>• <i>Favorite Packages</i> <ul style="list-style-type: none"> <li>◦ ZLOCAL</li> </ul> </li> <li>• <i>Released Objects</i> containing released APIs and development objects</li> </ul>	Customizable using ABAP Repository trees which display the following by default: <ul style="list-style-type: none"> <li>• <i>Local Objects (\$TMP)</i></li> <li>• <i>Favorite Packages</i></li> <li>• <i>System Library</i></li> </ul>

## Developing ABAP Applications

This chapter compares the elementary features which support ABAP developers when building ABAP applications:

Feature	SAP Cloud Platform ABAP Environment	ABAP Platform (On-Premise)
ABAP package interface	Not supported	Supported
Supported ABAP programming models	ABAP RESTful Programming Model	ABAP Programming Model for SAP Fiori
ABAP language	Optimized ABAP for SAP Cloud Platform language can be used	All ABAP keyword capabilities can be used
ABAP for SAP HANA development	Supported for ABAP-Managed Database Procedures (AMDP)	
SAP objects	Only released objects can be used Modification of SAP objects is not supported	All SAP objects can be used Modification of SAP objects is supported
Development object types	Only <a href="#">released [page 738]</a> object types from the are supported	All object types can be opened in the integrated editors or SAP GUI
Embedded SAP GUI	Not supported	Supported
Supported UI technologies	SAP Fiori	<ul style="list-style-type: none"> <li>• SAP Web Dynpro</li> <li>• SAP Dynpro</li> <li>• Floorplan manager tools</li> <li>• SAP Fiori</li> <li>• And others</li> </ul>

Feature	SAP Cloud Platform ABAP Environment	ABAP Platform (On-Premise)
Http service	Using <a href="#">native tooling [page 286]</a>	Using transaction <code>SICF</code> in the embedded SAP GUI
OData service	Using the	Using Core Data Services (CDS) with the CDS annotation <code>or</code> transaction <code>SEGW</code> in the embedded SAP GUI

## Working with ABAP Tools

This chapter compares the tools which an ABAP developer can use to build ABAP applications:

Feature	SAP Cloud Platform ABAP Environment	ABAP Platform (On-Premise)
Source-based and form-based editors for ABAP Dictionary objects	The same features can be used.	
Transport Organizer	The same features can be used.	
Versioning	Versioning managed in Git	Versioning using ABAP in the back end
Refactoring	The same features can be used.	
Quick assists	The same features can be used.	
Supported search functions	<ul style="list-style-type: none"> <li>ABAP object search</li> <li>Where-used function</li> </ul>	<ul style="list-style-type: none"> <li>ABAP object search</li> <li>ABAP source search</li> <li>Where-used function</li> </ul>
CDS toolset	<ul style="list-style-type: none"> <li>Creating, activating, and editing data models</li> <li>Comprehensive feature set provided for analyzing data models</li> <li>Previewing data records and following associations</li> <li>Extending data models using extend views</li> <li>Tools and object types to define and control access to data sources</li> </ul>	<ul style="list-style-type: none"> <li>Creating, activating, and editing data models</li> <li>Comprehensive feature set provided for analyzing data models</li> <li>Previewing data records and following associations</li> <li>Extending data models using metadata extensions and extend views</li> <li>Tools and object types for defining access to data sources</li> </ul>
ABAP unit testing	The same features can be used.	
ABAP Test Cockpit	The same features can be used.	

Feature	SAP Cloud Platform ABAP Environment	ABAP Platform (On-Premise)
Troubleshooting tools	Debugging is supported for read-only. Therefore, for example, the application of variables is not enabled while checking an ABAP application	Debugging also supports the application of, for example, variables while checking an ABAP application
Documentation capabilities with ABAP Doc	The same features can be used.	

## 6.3 Keyboard Shortcuts for ABAP Development

The following shortcuts are highly frequently used for working with development objects in ABAP Development Tools.

### i Note

The availability of the shortcuts depends on the user interface area you are currently working.

Select `Ctrl` + `Shift` + `L` to open a list that displays the area-specific keyboard shortcuts.

You can display a list of all available keyboard shortcuts from the menu bar ( [Windows](#)  [Preferences](#)  [General](#)  [Keys](#) ).

The subsequent lists are designed for using the operating systems Microsoft Windows<sup>®</sup> or Linux<sup>®</sup>. If you are working with Apple Mac<sup>®</sup>, you might need to choose the `Command` tab instead of `Ctrl`.

### 6.3.1 Edit Actions

Function	Shortcut
Activate inactive development object	<code>Ctrl</code> + <code>F3</code>
Activate all inactive development objects	<code>Ctrl</code> + <code>Shift</code> + <code>F3</code>
Check consistency and syntax	<code>Ctrl</code> + <code>F2</code>
Close	<code>Ctrl</code> + <code>W</code>
Close all	<code>Ctrl</code> + <code>Shift</code> + <code>W</code>
Code completion / Content Assist	<code>Ctrl</code> + <code>Space</code>
Delete line	<code>Ctrl</code> + <code>D</code>
Delete next word	<code>Ctrl</code> + <code>Delete</code>

Function	Shortcut
Delete previous word	<code>Ctrl</code> + <code>Backspace</code>
Find Next	<code>Ctrl</code> + <code>K</code>
Find Previous	<code>Ctrl</code> + <code>Shift</code> + <code>K</code>
Format source code (a.k.a. Pretty Printer)	<code>Shift</code> + <code>F1</code>
Format source block (a.k.a. Pretty Printer)	<code>Ctrl</code> + <code>Shift</code> + <code>F1</code>
Keyword completion	<code>Tab</code>
Mark word	Double-click
Mark whole line	Triple-click
New ABAP development object	<code>Ctrl</code> + <code>N</code>
Open quick fix / quick assist dialog	<code>Ctrl</code> + <code>1</code>
Rename	<code>Alt</code> + <code>Shift</code> + <code>R</code>
Save	<code>Ctrl</code> + <code>S</code>
Save all	<code>Ctrl</code> + <code>Shift</code> + <code>S</code>
Selection to upper case	<code>Ctrl</code> + <code>Shift</code> + <code>X</code>
Selection to lower case	<code>Ctrl</code> + <code>Shift</code> + <code>Y</code>

## 6.3.2 Displaying Actions

Function	Shortcut
Open a development object in other ABAP project(s)	<code>Alt</code> + <code>Ctrl</code> + <code>P</code>
Show bookmarks, breakpoints, and tasks of the focused editor row	<code>Alt</code> + <code>Ctrl</code> + <code>P</code>
Show the properties of the currently focused object or file	<code>Shift</code> + <code>Alt</code> + <code>P</code>
Show bookmarks, breakpoints, and tasks of the focused editor row	<code>Shift</code> + <code>Alt</code> + <code>P</code>
Show the <i>Variable</i> view in the ABAP debugger and to set the focus into the view.	<code>Shift</code> + <code>Alt</code> + <code>Q/V</code>

### i Note

Switch back to the source by using `F12`.

Function	Shortcut
Set the <i>View</i> menu, for text editors the ruler context menu is shown, or add bookmarks, breakpoints, or tasks	[Ctrl] + [F10]
Show bookmarks, breakpoints, and tasks of the focused editor row	[Alt] + [Ctrl] + [F10]
Adopt the size of the current editor, view, and so on to the maximum screen size	[Ctrl] + [M]

### 6.3.3 Navigation Actions

Function	Shortcut
Backward navigation to the previous opened tab	[Alt] + [Left]
Forward navigation to the next opened tab	[Alt] + [Right]
Select the next open editor you want to navigate to	[Ctrl] + [F6]
Switch between perspectives	[Ctrl] + [F8]
Select the previous open editor you want to navigate to	[Ctrl] + [Shift] + [F6]
Navigate to last edited location	[Ctrl] + [Q]
Open development object	[Ctrl] + [Shift] + [A]
Open Quick Outline	[Ctrl] + [O]
Open Quick Type Hierarchy	[Ctrl] + [T]
Switch to next view	[Ctrl] + [F7]
Switch to next perspective	[Ctrl] + [F8]
Navigate to ABAP source code	[F3] or [Ctrl] + [Click]
Move the keyboard focus into the top level editor or the active editor tab	[F12]
Show context menu	[Shift] + [F10]

### 6.3.4 Moving Actions

Function	Shortcut
Backward one word	[Ctrl] + [Left]
Forward one word	[Ctrl] + [Right]
Jump to line	[Ctrl] + [L]
Move one line down	[Alt] + [Down]

Function	Shortcut
Move one line up	[Alt] + [Up]

### 6.3.5 Commenting Actions

Function	Shortcut
Add comments	[Ctrl] + [<]
Remove comments	[Ctrl] + [>]
Add block comment	[Ctrl] + [Shift] + [/]
Toggle comments	[Ctrl] + [7]
Hide comments	[Ctrl] + [Alt] + [7]

### 6.3.6 Search and Help

Function	Shortcut
Open <i>Search</i> dialog	[Ctrl] + [H]
Show ABAP Keyword Documentation of an ABAP keyword	[F1]
Show ABAP element info	[F2]
Where-used list	[Ctrl] + [Shift] + [G]

### 6.3.7 Windows™-based Shortcuts

Function	Shortcut
Find / Replace text	[Ctrl] + [F]
Copy selection	[Ctrl] + [C]
Close	[Ctrl] + [F4]
Close all	[Ctrl] + [Shift] + [F4]
Cut selection	[Ctrl] + [X]
Paste selection	[Ctrl] + [V]
Redo typing	[Ctrl] + [Y]

Function	Shortcut
Select all	[Ctrl] + [A]
Undo typing	[Ctrl] + [Z]

## 6.4 Syntax of ABAP Dictionary Objects

This documentation describes the syntax that is used for editing classic ABAP Dictionary objects within ABAP Development Tools (ADT).

- [Structures \[page 751\]](#)
- [Database Tables \[page 772\]](#)

### i Note

Detailed concept information about classic objects in ABAP Dictionary can be found in the ABAP Keyword Documentation.

## Related Information

 [Classic Objects in ABAP Dictionary \(ABAP Keyword Documentation\)](#)

[ABAP Dictionary Editors \[page 63\]](#)

### 6.4.1 Structures

#### Syntax Form

```
structure_annotations [page 752]
DEFINE STRUCTURE structure {
    components [page 756]
}
```

#### Definition

A structure is a data type that consists of components. Each component can be an elementary type (either through a data element or the specification of the data type and length in the structure definition), another structure, or a table type. A structure can be nested to any depth.

## Notes

In ABAP Development Tools (ADT), you can create and edit structures in the ABAP source code editor.

## Use

### Example

The source code below defines a structure named `employee` that

- has a single component for the name of the employee.
- has the short text "Employee of a company" and cannot be enhanced. This restriction is defined by its structure annotation.

```
@EndUserText.label: 'Employee of a company'  
@AbapCatalog.enhancementCategory: #NOT_EXTENSIBLE  
DEFINE STRUCTURE employee {  
    name : employee_name;  
}
```

## Related Information

 [Structures \(ABAP Keyword Documentation\)](#)

### 6.4.1.1 Structure Annotations

Annotations enable you to add metadata to a structure.

In ABAP Development Tools (ADT), you can add the following annotations to structures:

- [Short Description \[page 754\]](#)
- [Enhancement Category \[page 753\]](#)

#### Note

Both annotations are mandatory when you are creating or editing structures in ADT.

## 6.4.1.1.1 Enhancement Category

### Syntax Form

```
@AbapCatalog.enhancementCategory : #NOT_EXTENSIBLE |
#NOT_CLASSIFIED |
#EXTENSIBLE_CHARACTER |
#EXTENSIBLE_CHARACTER_NUMERIC |
#EXTENSIBLE_ANY
```

### Definition

SAP's structures and database tables can be enhanced by customers using customizing includes or append structures.

### Use

You use enhancement categories to predefine warnings to be displayed if your changes result in problems.

### Values

You can use the following values to define enhancement categories in the source code:

List of Supported Enhancement Categories

Annotation Value	Description
#NOT_CLASSIFIED	The structure/database table has no enhancement category. <b>i Note</b> This value is only provided for structures that already exist. Therefore, it should not be used when creating new structures or changing existing ones.
#NOT_EXTENSIBLE	The structure/database table might be enhanced. <b>i Note</b> This is the default value for creating structures in ADT.
#EXTENSIBLE_CHARACTER	All structure/database table components and their enhancements must be character-like and flat.
CTER_NUMERIC	All structure/database table components and their enhancements must be flat.

Annotation Value	Description
#EXTENSIBLE_ANY	All structure/database table components and their enhancements can have any kind of data type.

## Related Information

 [Enhancement Category of Structures \(ABAP Keyword Documentation\)](#)

### 6.4.1.1.2 Short Description

#### Syntax Form

```
@EndUserText.label: 'short_description'
```

#### Definition

The short text is used as an explanatory text for the entire structure/database table.

#### Notes

These texts are always displayed in the original language. If you change any text, the text is saved in the original language as well.

If the logon language differs from the original language of the structure, a warning is displayed in the editor.

## Related Information

 [Semantic Attributes of Structures \(ABAP Keyword Documentation\)](#)

### 6.4.1.1.3 Output Style

#### Syntax Form

```
@AbapCatalog.decfloat.outputStyle : #NORMAL |  
  #SIGN_RIGHT |  
  #SCALE_PRESERVING |  
  #SCIENTIFIC |  
  #SCIENTIFIC_WITH.LEADING_ZERO |  
  #SCALE_PRESERVING_SIENTIFIC |  
  #ENGINEERING
```

#### Definition

This value determines the output style of a decimal floating point number.

#### Values

You can use the following values to define the output style of a decimal floating point number in the source code:

List of Supported Output Styles

Annotation Value	Description
#NORMAL	Mathematical or scientific in accordance to the available space
	<b>i Note</b> This is the default value.
#SIGN_RIGHT	Leading sign on the right
#SCALE_PRESERVING	Preservative scaling
#SCIENTIFIC	Scientific
#SCIENTIFIC_WITH.LEADING_ZERO	Scientific with leading zero
#SCALE_PRESERVING_SIENTIFIC	Scientific preservative scaling
#ENGINEERING	Engineering

## 6.4.1.2 Structure Components

A component is, for example, a subunit of a structure. The components of a structure can be elementary (any data type), a structure itself, an internal table, or a reference type.

Components can be created from [scratch \[page 756\]](#) or by including them from a [structure \[page 757\]](#) that already exists.

### Syntax Form

#### 1. Creating Components from Scratch

```
[component_annotations [page 758]]  
[foreign_key_annotations [page 765]]  
[KEY] component : [REFERENCE TO] user_defined_type [page 763] |  
predefined_type [page 762]  
[NOT NULL] [foreign_key_addition [page 768]]  
[value_help_addition [page 764]];
```

#### 2. Including Components from Existing Structures

```
[group :] INCLUDE structure | database table | view [WITH SUFFIX suffix]  
[component_extensions [page 770]];
```

### 6.4.1.2.1 Creating Components from Scratch

#### Syntax Form

```
[component_annotations [page 758]]  
[foreign_key_annotations [page 765]]  
[KEY] component : [REFERENCE TO] user_defined_type [page 763] |  
predefined_type [page 762]  
[NOT NULL] [foreign_key_addition [page 768]]  
[value_help_addition [page 764]];
```

#### Definition

A component always consists of a name and its type information, which can be followed optionally by a foreign key or a value help assignment.

## Notes

You can use the optional keywords `REFERENCE TO` for defining reference-like components directly in the structure. Data references can be defined by a reference to any data type in ABAP Dictionary or to the generic type `DATA`. Object references can be defined by a reference to classes or interfaces in the class library or to the generic type `OBJECT`.

You can use the optional keyword `KEY` in front of a component definition to define it as a key field of the structure. This is relevant for structures that are bound into database tables. Also, components of structures used as lock parameters in a lock object must be defined as `KEY` fields.

You can use the optional `NOT NULL` keywords at the end of a component definition to define that the field needs to have initial values and cannot be null in a database table. Therefore this is relevant for structures that are bound into database tables only.

## Example

The `employee` structure defines the following components:

- `name` is typed with the `employee_name` data element
- `date_of_birth` is typed to the predefined dictionary type `abap.dats` in order to contain a date value
- `address` is defined as a reference to the `if_address_object` interface

```
DEFINE STRUCTURE employee {
  name          : employee_name;
  date_of_birth : abap.dats;
  address       : REFERENCE TO if_address_object;
  ...
}
```

### 6.4.1.2.2 Including Components from Existing Structures

#### Syntax Form

```
[group :] INCLUDE structure | database table | view
          [WITH SUFFIX suffix] [component_extensions [page 770]];
```

#### Definition

Components of a structure can be defined by including the components of other structures in ABAP Dictionary, including database tables or views.

The name of the include structure thus follows after the `INCLUDE` keyword.

## Use

Optionally, you can add:

- a group to address the whole include structure like a component inside of an ABAP program
- a suffix to resolve name conflicts with other components of the structure by using the `WITH SUFFIX` keywords, followed by a suffix of up to three characters that is added to the included components
- component extensions to add/overrule value help or foreign key assignments to/of components from the include structure

## Example

The `employee` structure includes all components of the following structures:

- `org_data`
- address as the group `office_address` with the suffix `off`
- address as the group `private_address` with the suffix `pri`

```
...
INCLUDE org_data;
office_address : INCLUDE address WITH SUFFIX off;
private_address : INCLUDE address WITH SUFFIX pri;
...
```

### i Note

The following special characters are allowed in the field names `$%/#@!~?<>.{}_`. If a field name contains one of these characters, it must be written in double quotes. The same applies to group names and include suffixes.

## Related Information

 [Included Structures \(ABAP Keyword Documentation\)](#)

### 6.4.1.2.3 Component/Field Annotations

Component/field annotations enable you to add metadata to a component/field.

In ABAP Development Tools (ADT), you can add the following annotations to a component/field:

- [Short Description \[page 759\]](#) if the component/field is typed with a predefined type
- Reference to a [Currency Field \[page 759\]](#) if the component/field is an amount
- Reference to a [Unit Field \[page 760\]](#) if the component/field is a quantity

- Reference to a [Geographic Data Type \[page 761\]](#) if the component/field is a geographical location

### 6.4.1.2.3.1 Short Description

#### Syntax Form

```
@EndUserText.label: 'short_description'
```

#### Definition

The short text is used as an explanatory text for a component/field that is typed with a predefined ABAP Dictionary type. For components/fields that are defined with a user-defined ABAP Dictionary type, the corresponding short text is taken.

#### Editor-Specific Information

These texts are always displayed in the original language. If you change any text, the text is saved in the original language as well.

If the logon language differs from the original language of the structure, a warning is displayed in the editor.

#### Related Information

 [Semantic Attributes of Structures \(ABAP Keyword Documentation\)](#)

### 6.4.1.2.3.2 Currency Field

#### Syntax Form

```
@Semantics.amount.currencyCode : 'structure.field_containing_currency_key'
```

## Definition

If a component/field is based on the `abap.curr` predefined type, it has to be linked with a currency code field based on the `abap.cuky` predefined type.

## Example

The `salary` structure defines the unit component with the predefined data type `abap.curr`. Therefore, the reference to the `currency_key` component needs to be set. This is set through the given annotation.

```
DEFINE STRUCTURE salary {
  @Semantics.amount.currencyCode : 'salary.currency_key'
  amount : abap.curr(10,2);
  currency_key : abap.cuky;
}
```

## Related Information

 [Currency Fields \(ABAP Keyword Documentation\)](#)

### 6.4.1.2.3.3 Unit Field

#### Syntax Form

```
@Semantics.quantity.unitOfMeasure: 'structure.field_containing_unit_key'
```

## Definition

If a component/field is based on the `abap.quan` predefined data type, it has to be linked with a unit field based on the `abap.unit` predefined type.

## Example

The `order` structure defines the `item_quantity` component with the `abap.quan` predefined data type. Therefore, the reference to the `item_unit` component needs to be set. This is set through the given annotation.

```
DEFINE STRUCTURE order {
  @Semantics.quantity.unitOfMeasure: 'order.item_unit'
  item_quantity : abap.quan(4);
  item_unit      : abap.unit(2);
  ...
}
```

## Related Information

[Quantity Fields \(ABAP Keyword Documentation\)](#)

### 6.4.1.2.3.4 Geographic Data Type

## Syntax Form

```
@AbapCatalog.geo.spatialRefSystem : 'id'
```

## Definition

This value determines the underlying spatial reference system of a geographic element that is typed with the predefined ABAP data type `abap.geom_ewkb` as Extended Well Known Binary (EWKB).

## Notes

If the annotation is not explicitly given when defining an element of the `abap.geom_ewkb` data type, the annotation will be added with the default value '0' when saving the containing ABAP Dictionary object automatically. If an element already exists in a structure, the annotation is always shown with the given value.

The underlying spatial reference system can no more be changed after activation.

## Values

You can use, for example, the following values for defining the underlying geographic reference system in the source code:

List of Supported System Values

Annotation Value	Description
0	Default spatial reference system for undefined Spatial Reference Identifiers (SRID)
4326	Standard spatial reference system for spherical surfaces of the Earth
1000004326	Spatial reference system for the WGS84 planar standard for spherical surfaces of the Earth
2147483646	For unbounded planar spatial reference system

**i Note**

Only relevant for SAP-internal use.

## Related Information

[Spatial Reference Systems \(SRS\) and Spatial Reference Identifiers \(SRID\)](#)

### 6.4.1.2.4 Predefined ABAP Types

#### Syntax Form

```
... abap.name [ (n) | (n,m) ]
```

#### Definition

Elementary components/fields can be typed with a user-defined data element or directly with a predefined data type. In the latter case, you have to add the leading prefix `abap.`, followed by the name of the predefined dictionary type in a component/field definition. If the chosen type requires additional information on length and decimal places, you will need to add this information in brackets.

## Notes

You will find a list of the available predefined ABAP Dictionary types and their technical attributes in the ABAP Keyword Documentation that is linked below.

## Example

Both components are assigned to the predefined data types that begin with `abap.` as leading prefix. After the value, the length and decimals places follow in brackets.

```
...
  gender      : abap.char(1);
  cost_factor : abap.dec(2,1);
...
```

## Related Information

[Cloud icon](#) [Predefined Data Types in ABAP Dictionary \(ABAP Keyword Documentation\)](#)

### 6.4.1.2.5 User-Defined ABAP Types

#### Syntax Form

```
...
  data element | structure | database table | view | table type | class | interface
  ...
...
```

#### Definition

Components can be typed with any user-defined dictionary type or with classes or interfaces from the class library.

The following categories of user-defined types are provided:

- Data element
- Structure

#### Note

The optional keyword `BOXED` can be added after the structure name. This component is then defined as a boxed component within the surrounding structure.

- Database table
- View
- Table type
- ABAP class or interface: can only be used with the leading `REFERENCE TO` addition.

## Example

The following components are typed:

- `employee_id` as a data element
- `address` as a boxed component with a surrounding `struct` structure
- `accounts` as a table type
- `department` is referenced to an ABAP class

```
...
  employee_id : dtel_employee_id;
  address      : struct_address BOXED;
  accounts     : itab_accounts;
  department   : REFERENCE TO cl_department
...

```

## Related Information

[Cloud Data Types \(ABAP Keyword Documentation\)](#)

[Cloud Boxed Components \(ABAP Keyword Documentation\)](#)

### 6.4.1.2.6 Value Help Assignments

#### Syntax Form

```
...
  WITH VALUE HELP value_help
  WHERE value_help_parameter1 = structure.component1
         [AND value_help_parameter2 = structure.component2 ...];
...

```

#### Definition

You can add a value help to an elementary component/field. For this, you add the `WITH VALUE HELP` keywords, followed by the name of the value help and the assignments, to the value help parameters as a `WHERE` condition.

## Notes

In the WHERE condition, you must provide all parameters of the value help.

Single assignments are connected through the AND keyword.

## Editor-Specific Information

You can only use value helps of the ABAP Dictionary. Value helps that are implemented with check tables or similar technologies are maintained through a foreign key relationship.

## Example

The employee\_number component is typed with the s\_employee\_id data element. For this component, the z\_department value help is defined. The id and depnumber value help parameters are supplied with components of the surrounding structure. The language value help parameter receives a constant value.

```
...  
    employee_number : s_employee_id WITH VALUE HELP z_department  
    WHERE id = structure.id  
        AND depnumber = structure.dep_number  
        AND language = 'EN';  
...
```

## Related Information

 [Semantic Attributes of Structures \(ABAP Keyword Documentation\)](#)

### 6.4.1.2.7 Foreign Key Annotations

A foreign key represents one or more columns of a database table (foreign check table) that contains primary keys of another database table (check table). Both tables have a foreign key dependency.

If the component/field has a [foreign key \[page 768\]](#) relationship, the following foreign key annotations can be added:

- [Foreign Key Short Description \[page 794\]](#)
- [Foreign Key Field Type \[page 766\]](#)
- [Input Value Check on Dynpro \[page 767\]](#)
- [Message Class and Message Number \[page 768\]](#)

## i Note

You will find examples of all foreign key annotations in the [Foreign Keys \[page 768\]](#) chapter.

## Related Information

[Foreign Key \(ABAP Keyword Documentation\)](#)

[Check Table \(ABAP Keyword Documentation\)](#)

### 6.4.1.2.7.1 Foreign Key Field Type

#### Syntax Form

```
@AbapCatalog.foreignKey.keyType : #KEY |  
    #NON_KEY |  
    #TEXT_KEY
```

#### Definition

The foreign key field type of a foreign key describes the meaning of the foreign key fields in the foreign key table. In structures, this information is used for documentation purposes only.

#### Notes

You can assign the following values of foreign key types to a component:

List of Available Foreign Key Field Values

Values in ADT	Meaning	Description
#NON_KEY	No key fields / key candidates	The foreign key fields are not primary key fields of the foreign key table and do not uniquely identify a record of the foreign key table.
#KEY	Key fields / key candidates	The foreign key fields are either primary key fields of the foreign key table or they uniquely identify a record of the foreign key table as a key candidate.

Values in ADT	Meaning	Description
#TEXT_KEY	Key fields of a text table	If this type of foreign key field is defined, the foreign key table is handled as a text table to the check table. The primary key of the foreign key table must match the check table, plus a language key field with the LANG type. There can be only one text table for each check table. Otherwise, a warning occurs at activation.

## Related Information

 [Foreign Key](#)

### 6.4.1.2.7.2 Input Value Check on Dynpro

#### Syntax Form

```
@AbapCatalog.foreignKey.screenCheck : true | false
```

#### Definition

The input value check on dynpros and Web Dynpros tests whether the check table contains a record with the key given by the values in the foreign key fields.

#### Notes

You can define the values true (for activation) and false (for deactivation).

 Dynpro and WebDynpro are not relevant in the context SAP Cloud Platform ABAP Environment. Therefore, the ABAP annotation from above is not relevant in this context. Nevertheless, this annotation might be displayed in the source code of SAP's own ABAP Dictionary objects.

## Related Information

 [Dynpro \(ABAP Keyword Documentation\)](#)

### 6.4.1.2.7.3 Message Class and Message Number

#### Syntax Form

```
@AbapCatalog.foreignKey.messageClass : 'message_class_name'  
@AbapCatalog.foreignKey.messageNumber : 'message_number'
```

#### Definition

If an input value check on a dynpro or Web Dynpro fails, a standard message is displayed. This standard message can be replaced with any message in the definition of the foreign key. To define this, you can set a message class and message number through its annotation.

#### Notes

You should only add a message class if the `screenCheck` annotation is set on `true`.

Cloud WebDynpro is not relevant in the context SAP Cloud Platform ABAP Environment. Therefore, the CDS annotations from above are not relevant in this context. Nevertheless, these annotations might be displayed in the source code of SAP's own ABAP Dictionary objects.

#### Related Information

[Cloud Messages \(ABAP Keyword Documentation\)](#)

### 6.4.1.2.8 Foreign Keys

#### Syntax Form

```
foreign_key_annotations [page 765]  
...  
  WITH FOREIGN KEY [[1,m]] check_table  
    WHERE check_field1 = structure.component1  
      [AND check_field2 = structure.component2 ...];  
...
```

## Definition

You can add a foreign key relationship to an elementary component. To do this, you add the `WITH FOREIGN KEY` keywords, followed by the name of the check table and the assignments, to its check fields as a `WHERE` condition. Single assignments are connected with the `AND` keyword.

## Notes

If you define a foreign key for a component or on a component extension, you should add foreign key annotations.

You can define an optional `[n:m]` cardinality for each foreign key. In structures, cardinalities serve for documentation purposes only. In the source code editor,

- `n` relates to records typed with the surrounding structure
- `m` relates to entries in the check table

### Possible Values for `n`

- `[1]`: Precisely one record must exist for each row in the check table.
- `[0..1]`: There can be no more than one record for each row in the check table.
- `[1..*]`: There must be at least one record for each row in the check table.
- `[0..*]`: There can be any number of records for each row in the check table.

### Possible Values for `m`

- `[1]`: Precisely one row in the check table must exist for each record.
- `[0..1]`: No assigned rows must exist in the check table for a record.

## Example

The elementary component `department_id` has a foreign key relationship with the `sdepartment` check table. In this relationship, the `mandt` component for entries in the check table must match the `sy-mandt` field and the `id` component of the check table must match the `department_id` field of any record based on this structure.

The cardinality defines that for each record an entry should exist in the check table, but not vice versa.

If the foreign key check fails, the message with the number 001 of the `employee_messages` message class is displayed.

```
...
@AbapCatalog.foreignKey.label : 'Departments'
@AbapCatalog.foreignKey.keyType : #KEY
@AbapCatalog.foreignKey.screenCheck : true
@AbapCatalog.foreignKey.messageClass : 'employee_messages'
@AbapCatalog.foreignKey.messageNumber : '001'
  department_id : dtel_department WITH FOREIGN KEY [0..*,1] sdepartment
  WHERE mandt = syst.mandt
```

```
AND id = employee.department_id;  
...
```

## Related Information

[Cloud Foreign Key \(ABAP Keyword Documentation\)](#)  
[Cloud Check Table \(ABAP Keyword Documentation\)](#)

### 6.4.1.2.9 Component Extensions

#### Syntax Form

```
[foreign_key_annotations [page 765]]  
EXTEND field : foreign_key_addition [page 768] | value_help_addition [page 764];
```

#### Definition

Component extensions can be used:

- to add/overrule value help or foreign key assignments to/of components from an include structure
- in append structures to add value help or foreign key assignments to components of the original structure

A component extension is introduced with the EXTEND keyword, followed by the name of the component, and a foreign key or value help addition, or both.

#### Use

A component extension is used to add, overrule, or remove one or more components from another or the same structure.

#### Notes

If a structure is included, you can remove existing value helps or foreign keys using the REMOVE VALUE HELP or REMOVE FOREIGN KEY keywords.

## Example

The `zip_code` and `street` components of the `address` include structure are extended:

- For the `zip_code` component, a foreign key relationship is added.
- For the `street` component, the value help originally defined in the included structure is removed for usage in the `employee` structure.

```
...
office_address : INCLUDE address
  EXTEND zip_code: WITH FOREIGN KEY subsideries
    WHERE plz = employee.zip_code
  EXTEND street:
    REMOVE VALUE HELP;
...
```

### i Note

To find more information about how to use component extensions in an append structure, see an example in the chapter [Append Structures \[page 771\]](#).

## Related Information

[Including Components from Existing Structures \[page 757\]](#)

### 6.4.1.3 Append Structures

#### Syntax Form

```
<structure_annotations> [page 752]
EXTEND TYPE struct WITH append_struct {
  [components [page 756]]
  [component_extensions [page 770]]
}
```

#### Definition

Append structures are used to add components or component extensions to an existing structure without modifying the latter. They are independent development objects that are edited separately.

An append structure is introduced with the `EXTEND TYPE` keywords, followed by the name of the original structure, followed by the `WITH` keyword and the name of the append structure.

For append structures you use the same annotations as for regular structures.

## Notes

You can only add a

- value help if no value help exists for the original component
- foreign key if no foreign key exists for the original component

## Example

The `employee_with_address` append structure adds the `street` and `city` components.

In addition, the existing component `department` is extended by an assignment to the `departments` value help.

```
@EndUserText.label: 'Employee with additional fields'  
@AbapCatalog.enhancementCategory: #EXTENSIBLE_CHARACTER  
EXTEND TYPE employee WITH employee_with_address {  
    street : abap.char(40);  
    city   : abap.char(30);  
    EXTEND department : WITH VALUE HELP departments  
        WHERE department_id = employee_with_address.department;  
    ...  
}
```

## Related Information

[Append Structures \(ABAP Keyword Documentation\)](#)

[Creating Append Structures \[page 226\]](#)

## 6.4.2 Database Tables

### Syntax Form

This statement defines a database table in ABAP Dictionary.

```
database_table_annotations [page 774]  
DEFINE TABLE database_table {  
    components [page 784]  
}
```

## Definition

A database table is the set of data from a relational database. Each database table has technical and semantic properties that are defined in ABAP Dictionary.

## Use

A database table can be used to create, edit, or display database tables in a source-based editor.

All database tables defined in ABAP Dictionary can be referenced in ABAP programs as a structured data type.

You can access database tables defined in ABAP Dictionary directly in ABAP programs using ABAP SQL.

## Notes

Database tables defined in ABAP Dictionary are instantiated as database objects in ABAP database schema of the underlying database of the current AS ABAP.

In this reference documentation, you will find the ADT-specific content for source-based database tables. To get more concept information about database tables, see the ABAP Keyword Documentation below.

## Example

The source code below defines a database table named `employees`.

It consists of ...

- the following annotations from definition properties of the entire table:
  - `@EndUserText.label` determines the "All employees of a company" description
  - `@AbapCatalog.enhancementCategory` determines the `NOT_EXTENSIBLE` enhancement category and cannot be enhanced
  - `@AbapCatalog.tableCategory` determines the `TRANSPARENT` table category
  - `@AbapCatalog.deliveryClass` determines the `A` application type for application table and stores master data and transaction data
  - `@AbapCatalog.dataMaintenance` determines the `LIMITED` maintenance category for limited write access
- several table fields that are defined as follows:
  - `key <employee_id>` determines the key field of the database table using the `abap.char` predefined ABAP type that allows up to 20 characters
  - `name` determines the name of the employee using the `abap.char` predefined ABAP type that allows up to 40 characters. Here, the text label "Employee Name" is added using the `@EndUserText.label` annotation. It determines a semantic property of the subsequent table field.

- include `<zabc_struct_empl>` structure includes components of the `zabc_struct_empl` structure
- `<wage>` determines the currency of the employee wages that is specified in the `z_wage` data element
- The `@Semantics.amount.currencyCode` annotation refers to the `currency` field of the `demo_cds_currco` classical view. The latter defines the currency of the table field itself.

```

@EndUserText.label : 'All employees of a company'
@AbapCatalog.enhancementCategory : #NOT_EXTENSIBLE
@AbapCatalog.tableCategory : #TRANSPARENT
@AbapCatalog.deliveryClass : #A
@AbapCatalog.dataMaintenance : #LIMITED
define table employees {
  key employee_id : abap.char(20) not null;
  @EndUserText.label : 'Employee Name'
  name : abap.char(40);
  include zabc_struct_empl;
  @Semantics.amount.currencyCode : 'demo_cds_currco.currency'
  wage : z_wage not null;
}

```

## Related Information

 [Database Table \(ABAP Keyword Documentation\)](#)

 [Database Tables \(ABAP Keyword Documentation\)](#)

### 6.4.2.1 Database Table Annotations

Annotations enable you to add metadata to a database.

In ABAP Development Tools (ADT), you can use the following annotations for source-based database tables:

- Technical attributes:
  - [Table Category \[page 775\]\\*](#)
  - [Enhancement Category \[page 753\]\\*](#)
  - [Activation Type \[page 777\]](#)
- Semantic attributes:
  - [Short Description \[page 754\]\\*](#)
  - [Data Maintenance \[page 779\]\\*](#)
  - [Delivery Class \[page 781\]\\*](#)
- [Replacement Object \[page 783\]](#)

#### Note

\*These annotations are mandatory when creating or editing database tables in ADT.

When the source code of a database table is activated, annotations and comments that cannot be mapped to the database are removed.

## 6.4.2.1.1 Technical Attributes

Technical attributes describe information about the technical specification and implementation of a database table. The technical attributes of a database table are the attributes of their structure and the database table-specific properties.

The technical settings (data class, size category, buffering, and storage type) constitute a standalone object and can be activated and transported separately from the table.

### i Note

To edit these technical settings or to create/edit indexes, open the integrated SAP GUI.

### 6.4.2.1.1.1 Table Category

#### Syntax Form

```
@AbapCatalog.tableCategory: #TRANSPARENT |  
#GLOBAL_TEMPORARY
```

#### Definition

The table category specifies the storage method and primary access method to the underlying database.

### i Note

ABAP Dictionary supports transforming database tables from one category to another.

#### Values

You can use the following values for defining the table category in the source code:

List of Supported Table Categories

Annotation	Value	Description	Note
	#TRANSPARE NT	The database object of a transparent table has the identically named characteristic with the same columns as the definition in ABAP Dictionary.	<p><b>i Note</b></p> <p>This is the default value for creating database tables in ADT.</p>

Annotation Value	Description	Note
#GLOBAL_TEMPORARY	A global temporary table is a special transparent table exclusively used for repository of temporary data.	

## Related Information

- [Table Category of Database Tables \(ABAP Keyword Documentation\)](#)
- [Global Temporary Tables \(ABAP Keyword Documentation\)](#)
- [Storage Type of Database Tables \(ABAP Keyword Documentation\)](#)

## 6.4.2.1.1.2 Enhancement Category

### Syntax Form

```
@AbapCatalog.enhancementCategory : #NOT_EXTENSIBLE |
    #NOT_CLASSIFIED |
    #EXTENSIBLE_CHARACTER |
    #EXTENSIBLE_CHARACTER_NUMERIC |
    #EXTENSIBLE_ANY
```

### Definition

SAP's structures and database tables can be enhanced by customers using customizing includes or append structures.

### Use

You use enhancement categories to predefine warnings to be displayed if your changes result in problems.

## Values

You can use the following values to define enhancement categories in the source code:

List of Supported Enhancement Categories

Annotation Value	Description
#NOT_CLASSIFIED	The structure/database table has no enhancement category.  <b>i Note</b> This value is only provided for structures that already exist. Therefore, it should not be used when creating new structures or changing existing ones.
#NOT_EXTENSIBLE	The structure/database table might be enhanced.  <b>i Note</b> This is the default value for creating structures in ADT.
#EXTENSIBLE_CHARACTERIC	All structure/database table components and their enhancements must be character-like and flat.
#EXTENSIBLE_CHARACTERNUMERIC	All structure/database table components and their enhancements must be flat.
#EXTENSIBLE_ANY	All structure/database table components and their enhancements can have any kind of data type.

## Related Information

 [Enhancement Category of Structures \(ABAP Keyword Documentation\)](#)

### 6.4.2.1.1.3 Activation Type

#### Syntax Form

```
@AbapCatalog.activationType : #NOT_CLASSIFIED |
    #NAMETAB_GENERATION_OFFLINE |
    #ADAPT_C_STRUCTURES |
    #INITIAL_TABLE_REQUIRED
```

#### Definition

The activation type of a database table describes if a database table can be activated using the relevant tools in ABAP Dictionary or if activation needs to be triggered by the ABAP runtime.

## Use

You, as a customer, can only define the default value.

## Values

You can use the following values for defining activation types in the source code:

List of Supported Activation Types

Annotation Value	Description	Note
#NOT_CLASSIFIED	The database table can be activated using with an ABAP Dictionary tool interactively.	This is the default value for creating database tables in ADT.
	<p>→ Recommendation</p> <p>SAP recommends that database tables for application development should use this default value.</p>	
#NAMETAB_GENERATION_OFFLINE	The runtime object of the database table must be created using an ABAP kernel program before activating in ABAP Dictionary. This makes it impossible to modify and activate important system tables directly.	This value is used for database tables that are needed from the ABAP runtime. Therefore, it is only relevant for SAP-internal development.
#ADAPT_C_STRUCTURES	The database table is used from the ABAP kernel. Activation and therefore creation of its runtime objects is despite possible in ABAP Dictionary.	
	<p>Check the programs in use from the kernel for possible adoptions.</p>	
#INITIAL_TABLE_REQUIRED	The database table is required before other objects are modified as part of a transport.	

## Related Information

[Activation Type of Database Tables \(ABAP Keyword Documentation\)](#)

[Activating ABAP Dictionary Types \(ABAP Keyword Documentation\)](#)

## 6.4.2.1.2 Semantic Attributes

Semantic attributes provide additional information and specify the behavior of a database table. The semantic attributes of a database table are the properties of their structure and the database table-specific properties.

### 6.4.2.1.2.1 Short Description

#### Syntax Form

```
@EndUserText.label: 'short_description'
```

#### Definition

The short text is used as an explanatory text for the entire structure/database table.

#### Notes

These texts are always displayed in the original language. If you change any text, the text is saved in the original language as well.

If the logon language differs from the original language of the structure, a warning is displayed in the editor.

#### Related Information

 [Semantic Attributes of Structures \(ABAP Keyword Documentation\)](#)

### 6.4.2.1.2.2 Data Maintenance

#### Syntax Form

```
@AbapCatalog.dataMaintenance : #LIMITED |  
    #ALLOWED |  
    #NOT_ALLOWED
```

## Definition

Data maintenance describes the possibilities if displaying and/or editing the content of a database table is allowed **or** not.

## Use

You can use the

- [Data Browser](#) (transaction SE16) and [Table View Maintenance](#) (transactions SM30 and SM31) to maintain database tables.
- [Data Preview](#) to access and display the content stored in a database table.

### i Note

In the current client version of ABAP Development Tools, the technical settings and maintenance features can only be edited using the embedded SAP GUI.

## Values

You can use the following values for defining the data maintenance possibilities in the source code:

List of Supported Data Maintenance Values

Annotation Value	Description
#LIMITED	<p>⌚ The content which is stored in a database table can be displayed, but not be edited.</p> <p><b>i Note</b></p> <p>This is the default value for creating database tables in ADT.</p> <p>In the current ADT version, the #LIMITED and #ALLOWED values result in the same effect.</p>
#ALLOWED	<p>⌚ The content which is stored in a database table can be displayed and edited.</p> <p><b>i Note</b></p> <p>This value is not relevant in the context of ABAP environment, because maintaining the content of a database table is not supported in the <a href="#">Data Preview</a> in the current version of ADT.</p> <p>In the current ADT version, the #LIMITED and #ALLOWED values result in the same effect.</p>

Annotation Value	Description
#NOT_ALLOWED	<p>⌚ The content which is stored in a database table <b>cannot</b> be displayed and edited.</p> <p><b>i Note</b></p> <p>This value is not relevant in the context of ABAP environment.</p>

## Related Information

[Displaying and Editing Database Tables \(ABAP Keyword Documentation\)](#)

### 6.4.2.1.2.3 Delivery Class

#### Syntax Form

```
@AbapCatalog.deliveryClass : #A |
  #C |
  #L |
  #G |
  #E |
  #S |
  #W
```

#### Definition

The delivery class of a database table controls

- the transport of table data from installations, upgrades, or client copies, and
- the transports between customer systems.

It is also considered in extended table maintenance.

## Values

You can use the following values for the defining delivery class in the source code:

List of Supported Delivery Classes

Annotation Value	Description
#A	Application table for master data and transaction data. The data is written by application programs.
	<b>i Note</b> This is the default value for creating database tables in ADT.
#C	Customer table for data entered by the customer only
#I	Table used as a repository of temporary data
	<b>i Note</b> SAP delivers tables in the delivery class I as empty tables.
#G	Customer table where SAP can add data but not modify or delete
	<b>i Note</b> A customer table of this delivery class must be assigned a customer namespace in the database table TRESC using program RDKOR54.
#E	System table in which customers can make entries
	<b>i Note</b> A customer table of this delivery class must be assigned a customer namespace in the database table TRESC using program RDKOR54.
#S	System table delivered by SAP with predefined data as part of the system, such as ABAPDOCU_TREE
	<b>i Note</b> Data of the tables of this delivery class is transported and handled like repository objects. This means that they have an original system. Any changes in other systems are handled as modifications or repairs.
#W	System table for administration data of a system, such as TFDIR as a directory of all function modules
	<b>i Note</b> A system table in the delivery class W is usually delivered with entries from SAP. It can be affected by customer modifications. The content is transported using the transport objects assigned to the objects in questions.

## Related Information

[Cloud Delivery Class for Database Tables \(ABAP Keyword Documentation\)](#)

### 6.4.2.1.3 Replacement Object

#### Syntax Form

```
@AbapCatalog.replacementObject : 'cds_entity_name'
```

#### Definition

A CDS view can be defined as a replacement object for a transparent database table or a classic database view in ABAP Dictionary. In this case, read access using ABAP SQL is carried out through the replacement object.

#### Use

You can assign a transparent database table and a classic database view in ABAP Dictionary to a CDS view. To do this, you assign the name of the relevant CDS entity to the `@AbapCatalog.replacementObject` annotation as a replacement object.

This enables you to retrieve data using a substitute object instead of a database table or a classic database view.

##### → Recommendation

SAP might deliver a replacement object for database tables for aggregated data. But SAP does not recommend you create your own replacement objects.

##### i Note

You can only assign a replacement object if the structure type – defined by the CDS view – fulfills specific requirements.

You cannot define replacement objects for global temporary tables.

For further information, see the Related Information below.

## Value

Enter the name of the replacement object to be considered in quotation marks after the annotation name and the colon.

## Notes

The relevant name will not be considered for translation.

## Related Information

- ⌚ [Replacement object \(ABAP Keyword Documentation\)](#)
- ⌚ [Replacement Objects \(ABAP Keyword Documentation\)](#)

## 6.4.2.2 Database Table Fields

### Definition

A database table field (short form field) describes the data to be stored in the database.

### Use

Database table fields can be created:

- from [scratch \[page 785\]](#) or
- by including them from a [structure \[page 786\]](#) with a flat data type that already exists.

### Syntax Form

You have the following possibilities to add fields to database tables:

1. Creating Fields from Scratch

```
[component_annotations [page 758]]  
    [foreign_key_annotations [page 765]]  
    [KEY] component : user_defined_type [page 792] |  
predefined_type [page 762]  
    [NOT NULL] [foreign_key_addition [page 768]]  
    [value_help_addition [page 764]];
```

## 2. Including Fields from Existing Structures

```
[group :] [KEY] INCLUDE structure [component_extensions [page 770]] | [NOT NULL];
```

### 6.4.2.2.1 Creating Fields from Scratch

#### Syntax Form

```
[component_annotations [page 758]]  
[foreign_key_annotations [page 765]]  
[KEY] field : user_defined_type [page 763] | predefined_type [page 762]  
[NOT NULL] [foreign_key_addition [page 768]]  
[value_help_addition [page 764]];
```

#### Definition

A field always consists of a name and its type information.

In addition, a foreign key or a value help assignment can be added to a field.

#### Use

You can use the optional keyword **KEY** in front of a component definition to define it as a key field of the structure. This is relevant for structures that are bound to database tables. Also, components of structures used as lock parameters in a lock object must be defined as **KEY** fields.

You can use the optional **NOT NULL** SQL keywords at the end of a component definition to define that the field needs to have initial values and cannot be null in a database table. This is relevant for structures that are bound into database tables only.

#### Example

The `employees` table defines the following components:

- `pers_id` is the key field of the table and typed to the predefined ABAP type `abap.char(40)`
- `name` is typed with the `employee_name` data element

```
DEFINE TABLE employees {  
  key pers_id    : abap.char(40);  
  name      : employee_name;  
  ...
```

}

### i Note

The following special characters are allowed in the group names \$%/#@!\*()~?<>.{}\_ where . and ! cannot be used in the first position. If a group name contains one of these characters, it must be written in double quotes. The same applies to include suffixes.

## Related Information

[Names of Structure Components \(ABAP Keyword Documentation\)](#)

### 6.4.2.2.2 Including Fields from Existing Structures

An include structure (include) in a database table is a reference to an existing structure.

## Syntax Form

```
[group :] INCLUDE structure
      [WITH SUFFIX suffix] [component_extensions [page 770]];
```

## Definition

Fields of a database table can be defined by including the components of other structures in ABAP Dictionary.

The name of an include structure (includes) comes after the INCLUDE keyword. The data type of an include structure must be flat.

## Use

Optionally, you can add:

- a group to address the whole include structure like a component inside of an ABAP program
- a suffix to resolve name conflicts with other components of the structure by using the WITH SUFFIX keywords, followed by a suffix of up to three characters that is added to the included components
- field extensions to add/overrule value help or foreign key assignments to/of fields from an include structure

## Example

The `employee` structure includes all components of the following structure:

- `org_data`
- address as the group `office_address` with the suffix `off`
- address as the group `private_address` with the suffix `pri`

```
...
INCLUDE org_data;
office_address : INCLUDE address WITH SUFFIX off;
private_address : INCLUDE address WITH SUFFIX pri;
...
```

### i Note

The following special characters are allowed in the group names `$%&#@!*()~?<>.{}_`. If a group name contains one of these characters, it must be written in double quotes. The same applies to include suffixes.

## Related Information

 [Included Structures \(ABAP Keyword Documentation\)](#)  
[Component Extensions \[page 770\]](#)

### 6.4.2.2.3 Component/Field Annotations

Component/field annotations enable you to add metadata to a component/field.

In ABAP Development Tools (ADT), you can add the following annotations to a component/field:

- [Short Description \[page 759\]](#) if the component/field is typed with a predefined type
- Reference to a [Currency Field \[page 759\]](#) if the component/field is an amount
- Reference to a [Unit Field \[page 760\]](#) if the component/field is a quantity
- Reference to a [Geographic Data Type \[page 761\]](#) if the component/field is a geographical location

#### 6.4.2.2.3.1 Short Description

##### Syntax Form

```
@EndUserText.label: 'short_description'
```

## Definition

The short text is used as an explanatory text for a component/field that is typed with a predefined ABAP Dictionary type. For components/fields that are defined with a user-defined ABAP Dictionary type, the corresponding short text is taken.

## Editor-Specific Information

These texts are always displayed in the original language. If you change any text, the text is saved in the original language as well.

If the logon language differs from the original language of the structure, a warning is displayed in the editor.

## Related Information

 [Semantic Attributes of Structures \(ABAP Keyword Documentation\)](#)

## 6.4.2.2.3.2 Currency Field

### Syntax Form

```
@Semantics.amount.currencyCode : 'structure.field_containing_currency_key'
```

## Definition

If a component/field is based on the `abap.curr` predefined type, it has to be linked with a currency code field based on the `abap.cuky` predefined type.

## Example

The `salary` structure defines the unit component with the predefined data type `abap.curr`. Therefore, the reference to the `currency_key` component needs to be set. This is set through the given annotation.

```
DEFINE STRUCTURE salary {  
  @Semantics.amount.currencyCode : 'salary.currency_key'  
  amount : abap.curr(10,2);  
  currency_key : abap.cuky;
```

```
}
```

## Related Information

[Cloud Currency Fields \(ABAP Keyword Documentation\)](#)

### 6.4.2.2.3.3 Unit Field

#### Syntax Form

```
@Semantics.quantity.unitOfMeasure: 'structure.field_containing_unit_key'
```

#### Definition

If a component/field is based on the `abap.quan` predefined data type, it has to be linked with a unit field based on the `abap.unit` predefined type.

#### Example

The `order` structure defines the `item_quantity` component with the `abap.quan` predefined data type. Therefore, the reference to the `item_unit` component needs to be set. This is set through the given annotation.

```
DEFINE STRUCTURE order {
  @Semantics.quantity.unitOfMeasure: 'order.item_unit'
  item_quantity : abap.quan(4);
  item_unit      : abap.unit(2);
  ...
}
```

## Related Information

[Cloud Quantity Fields \(ABAP Keyword Documentation\)](#)

## 6.4.2.2.3.4 Geographic Data Type

### Syntax Form

```
@AbapCatalog.geo.spatialRefSystem : 'id'
```

### Definition

This value determines the underlying spatial reference system of a geographic element that is typed with the predefined ABAP data type `abap.geom_ewkb` as Extended Well Known Binary (EWKB).

### Notes

If the annotation is not explicitly given when defining an element of the `abap.geom_ewkb` data type, the annotation will be added with the default value '0' when saving the containing ABAP Dictionary object automatically. If an element already exists in a structure, the annotation is always shown with the given value.

The underlying spatial reference system can no more be changed after activation.

### Values

You can use, for example, the following values for defining the underlying geographic reference system in the source code:

List of Supported System Values

Annotation Value	Description
0	Default spatial reference system for undefined Spatial Reference Identifiers (SRID)
4326	Standard spatial reference system for spherical surfaces of the Earth
1000004326	Spatial reference system for the WGS84 planar standard for spherical surfaces of the Earth

Annotation Value	Description
2147483646	For unbounded planar spatial reference system
	<p><b>i Note</b> Only relevant for SAP-internal use.</p>

## Related Information

[Spatial Reference Systems \(SRS\) and Spatial Reference Identifiers \(SRID\)](#)

### 6.4.2.2.4 Predefined ABAP Types

#### Syntax Form

```
... abap.name [ (n) | (n,m) ]
```

#### Definition

Elementary components/fields can be typed with a user-defined data element or directly with a predefined data type. In the latter case, you have to add the leading prefix `abap.`, followed by the name of the predefined dictionary type in a component/field definition. If the chosen type requires additional information on length and decimal places, you will need to add this information in brackets.

#### Notes

You will find a list of the available predefined ABAP Dictionary types and their technical attributes in the ABAP Keyword Documentation that is linked below.

#### Example

Both components are assigned to the predefined data types that begin with `abap.` as leading prefix. After the value, the length and decimal places follow in brackets.

```
...
```

```
gender      : abap.char(1);
cost_factor : abap.dec(2,1);
...
```

## Related Information

[Cloud Predefined Data Types in ABAP Dictionary \(ABAP Keyword Documentation\)](#)

### 6.4.2.2.5 User-Defined ABAP Types

#### Syntax Form

```
...      data element | predefined\_abap\_type \[page 762\]
...
```

#### Definition

Fields can be typed with any user-defined ABAP Dictionary type.

The following categories of user-defined types are supported:

- Data element
- Predefined ABAP types

#### Example

The following components/fields are typed:

- `employee_id` as a data element
- `date_birth` as the predefined ABAP type for dates

```
...      employee_id  : dtel_employee_id;
          date_birth   : abap.dats;
...
```

## 6.4.2.2.6 Value Help Assignments

### Syntax Form

```
...  
  WITH VALUE HELP value_help  
  WHERE value_help_parameter1 = structure.component1  
        [AND value_help_parameter2 = structure.component2 ...];  
...
```

### Definition

You can add a value help to an elementary component/field. For this, you add the **WITH VALUE HELP** keywords, followed by the name of the value help and the assignments, to the value help parameters as a **WHERE** condition.

### Notes

In the **WHERE** condition, you must provide all parameters of the value help.

Single assignments are connected through the **AND** keyword.

### Editor-Specific Information

You can only use value helps of the ABAP Dictionary. Value helps that are implemented with check tables or similar technologies are maintained through a foreign key relationship.

### Example

The `employee_number` component is typed with the `s_employee_id` data element. For this component, the `z_department` value help is defined. The `id` and `depnumber` value help parameters are supplied with components of the surrounding structure. The `language` value help parameter receives a constant value.

```
...  
  employee_number : s_employee_id WITH VALUE HELP z_department  
  WHERE id = structure.id  
        AND depnumber = structure.dep_number  
        AND language = 'EN';  
...
```

## Related Information

 [Semantic Attributes of Structures \(ABAP Keyword Documentation\)](#)

### 6.4.2.2.7 Foreign Key Annotations

A foreign key represents one or more columns of a database table (foreign check table) that contains primary keys of another database table (check table). Both tables have a foreign key dependency.

If the component/field has a [foreign key \[page 768\]](#) relationship, the following foreign key annotations can be added:

- [Foreign Key Short Description \[page 794\]](#)
- [Foreign Key Field Type \[page 766\]](#)
- [Input Value Check on Dynpro \[page 767\]](#)
- [Message Class and Message Number \[page 768\]](#)

 Note

You will find examples of all foreign key annotations in the [Foreign Keys \[page 768\]](#) chapter.

## Related Information

 [Foreign Key \(ABAP Keyword Documentation\)](#)

 [Check Table \(ABAP Keyword Documentation\)](#)

### 6.4.2.2.7.1 Foreign Key Short Description

#### Syntax Form

```
@AbapCatalog.foreignKey.label: 'literal_string'
```

#### Definition

The short text is used as an explanatory text for a foreign key relationship.

## 6.4.2.2.7.2 Foreign Key Field Type

### Syntax Form

```
@AbapCatalog.foreignKey.keyType : #KEY |  
  #NON_KEY |  
  #TEXT_KEY
```

### Definition

The foreign key field type of a foreign key describes the meaning of the foreign key fields in the foreign key table. In structures, this information is used for documentation purposes only.

### Notes

You can assign the following values of foreign key types to a component:

List of Available Foreign Key Field Values

Values in ADT	Meaning	Description
#NON_KEY	No key fields / key candidates	The foreign key fields are not primary key fields of the foreign key table and do not uniquely identify a record of the foreign key table.
#KEY	Key fields / key candidates	The foreign key fields are either primary key fields of the foreign key table or they uniquely identify a record of the foreign key table as a key candidate.
#TEXT_KEY	Key fields of a text table	If this type of foreign key field is defined, the foreign key table is handled as a text table to the check table. The primary key of the foreign key table must match the check table, plus a language key field with the LANG type. There can be only one text table for each check table. Otherwise, a warning occurs at activation.

### Related Information

 [Foreign Key](#)

### 6.4.2.2.7.3 Input Value Check on Dynpro

#### Syntax Form

```
@AbapCatalog.foreignKey.screenCheck : true | false
```

#### Definition

The input value check on dynpros and Web Dynpros tests whether the check table contains a record with the key given by the values in the foreign key fields.

#### Notes

You can define the values true (for activation) and false (for deactivation).

Cloud Dynpro and WebDynpro are not relevant in the context SAP Cloud Platform ABAP Environment. Therefore, the ABAP annotation from above is not relevant in this context. Nevertheless, this annotation might be displayed in the source code of SAP's own ABAP Dictionary objects.

#### Related Information

[Dynpro \(ABAP Keyword Documentation\)](#)

### 6.4.2.2.7.4 Message Class and Message Number

#### Syntax Form

```
@AbapCatalog.foreignKey.messageClass : 'message_class_name'  
@AbapCatalog.foreignKey.messageNumber : 'message_number'
```

#### Definition

If an input value check on a dynpro or Web Dynpro fails, a standard message is displayed. This standard message can be replaced with any message in the definition of the foreign key. To define this, you can set a message class and message number through its annotation.

## Notes

You should only add a message class if the `screenCheck` annotation is set on `true`.

Cloud WebDynpro is not relevant in the context SAP Cloud Platform ABAP Environment. Therefore, the CDS annotations from above are not relevant in this context. Nevertheless, these annotations might be displayed in the source code of SAP's own ABAP Dictionary objects.

## Related Information

[Cloud Messages \(ABAP Keyword Documentation\)](#)

### 6.4.2.2.8 Foreign Keys

#### Syntax Form

```
foreign_key_annotations [page 765]
...
  WITH FOREIGN KEY [[1,m]] check_table
    WHERE check_field1 = structure.component1
      AND check_field2 = structure.component2 ...];
...
```

#### Definition

You can add a foreign key relationship to an elementary component. To do this, you add the `WITH FOREIGN KEY` keywords, followed by the name of the check table and the assignments, to its check fields as a `WHERE` condition. Single assignments are connected with the `AND` keyword.

## Notes

If you define a foreign key for a component or on a component extension, you should add foreign key annotations.

You can define an optional `[n:m]` cardinality for each foreign key. In structures, cardinalities serve for documentation purposes only. In the source code editor,

- `n` relates to records typed with the surrounding structure
- `m` relates to entries in the check table

#### Possible Values for `n`

- [1]: Precisely one record must exist for each row in the check table.
- [0..1]: There can be no more than one record for each row in the check table.
- [1..\*]: There must be at least one record for each row in the check table.
- [0..\*]: There can be any number of records for each row in the check table.

#### Possible Values for `m`

- [1]: Precisely one row in the check table must exist for each record.
- [0..1]: No assigned rows must exist in the check table for a record.

## Example

The elementary component `department_id` has a foreign key relationship with the `sdepartment` check table. In this relationship, the `mandt` component for entries in the check table must match the `sy-mandt` field and the `id` component of the check table must match the `department_id` field of any record based on this structure.

The cardinality defines that for each record an entry should exist in the check table, but not vice versa.

If the foreign key check fails, the message with the number 001 of the `employee_messages` message class is displayed.

```
...
@AbapCatalog.foreignKey.label : 'Departments'
@AbapCatalog.foreignKey.keyType : #KEY
@AbapCatalog.foreignKey.screenCheck : true
@AbapCatalog.foreignKey.messageClass : 'employee_messages'
@AbapCatalog.foreignKey.messageNumber : '001'
  department_id : dtel_department WITH FOREIGN KEY [0..*,1] sdepartment
    WHERE mandt = syst.mandt
    AND id = employee.department_id;
...
...
```

## Related Information

[Foreign Key \(ABAP Keyword Documentation\)](#)

[Check Table \(ABAP Keyword Documentation\)](#)

## 6.4.2.2.9 Component Extensions

### Syntax Form

```
[foreign_key_annotations [page 765]]
EXTEND field : foreign_key_addition [page 768] | value_help_addition [page 764];
```

## Definition

Component extensions can be used:

- to add/overrule value help or foreign key assignments to/of components from an include structure
- in append structures to add value help or foreign key assignments to components of the original structure

A component extension is introduced with the `EXTEND` keyword, followed by the name of the component, and a foreign key or value help addition, or both.

## Use

A component extension is used to add, overrule, or remove one or more components from another or the same structure.

## Notes

If a structure is included, you can remove existing value helps or foreign keys using the `REMOVE VALUE HELP` or `REMOVE FOREIGN KEY` keywords.

## Example

The `zip_code` and `street` components of the `address` include structure are extended:

- For the `zip_code` component, a foreign key relationship is added.
- For the `street` component, the value help originally defined in the included structure is removed for usage in the `employee` structure.

```
...
office_address : INCLUDE address
EXTEND zip_code: WITH FOREIGN KEY subsideries
  WHERE plz = employee.zip_code
EXTEND street:
  REMOVE VALUE HELP;
...
```

### i Note

To find more information about how to use component extensions in an append structure, see an example in the chapter [Append Structures \[page 771\]](#).

## Related Information

[Including Components from Existing Structures \[page 757\]](#)

## 6.5 Syntax for Breakpoint Conditions

Breakpoint conditions have one or more operands. These can be:

- Variables (including special debugger symbols)
- Literals
- Built-in debugger functions

## Syntax

The condition syntax allowed in breakpoint conditions is basically a subset of what is allowed in ABAP logical expressions, such as an `IF` statement.

You can specify elementary logical expressions (for example, `SY-SUBRC = 0`) and combine them using the keyword `AND`, `OR`.

In addition, you can change the order of valuation by using brackets - as you are familiar with from logical expressions in ABAP.

Similar to ABAP, you can also make use of short-circuit evaluation semantics, just like in the following example:

```
oref is bound and oref->attr = 'abc'
```

The second elementary logical expression `oref->attr = 'abc'` is evaluated only if the first expression `oref is bound` has been evaluated to true.

## Examples for Valid Conditions

- `sy-index > 5`
- `sy-index = sy-tabix`
- `lines( itab ) > 0`
- `lines( itab ) <> sy-tabix`
- `lines( itab ) < lines( itab2 )`
- `strlen( s ) >= sy-index`
- `INEXACT_DF = 'X'`
- `oref is bound and oref->attr = 'abc'`

- 'DEADBEEF' cs 'BEEF' and sy-fdpos = 4
- a > b AND (c > b OR c > d)

## Related Topics

- [Variables in Breakpoint Conditions \[page 801\]](#)
- [Literals in Breakpoint Conditions \[page 802\]](#)
- [Built-in Functions in Breakpoint Conditions \[page 803\]](#)
- [Operators in Breakpoint Conditions \[page 804\]](#)

## 6.5.1 Variables in Breakpoint Conditions

In breakpoint conditions, you can use both simple variables and components of structures as variables. In addition, you can also access field symbols, class attributes and the content of internal table rows. Finally, you have the option, to specify offset and length for strings, xstrings, and character fields.

### Using Variable Symbols

In addition, to access variables, you can make use of some specific debugger notations (variable symbols) that are not available in ABAP source code.

Variable Symbol	Meaning
itab[ ]	Identifies the internal table itab  This symbol is only important for tables with a header row where itab identifies the header row of the internal table.
itab[n]	Accesses the n-th row of the internal itab
{C:my_class}	Identifies a class with the name my_class

You can use these symbols to access components as operands for breakpoint conditions, for example:

{c:myclass}-attr1 - Accesses the static attribute attr1 of the class myclass

itab[100]-comp1 - Accesses the component comp1 of the internal table itab for the 100-th row

## 6.5.2 Literals in Breakpoint Conditions

Breakpoint conditions support literals as operands. As you are already familiar with from ABAP syntax, the following types of literals are also valid in breakpoint conditions:

## Character Literals

Character literals are identified by single quotes, for example:

‘This is a type c literal’

### i Note

Character literals have the ABAP type c.

## String Literals

String literals are identified by back quotes, for example:

‘This is a string’

### i Note

String literals have the ABAP type STRING.

## Numeric Literals: Type I and Type P

Numeric literals consist only of number characters, optionally introduced by a '-' character, indicating a negative value:

- 123456789
  - -123456789

## i Note

## 6.5.3 Built-in Functions in Breakpoint Conditions

There are several built-in functions available for defining breakpoint conditions. All functions have a return value and can include up to one argument. To find out what the type of the return value and the argument is, read the detailed description of the functions below.

Note that the **syntax** of the built-in functions is **based on ABAP style**. For example, SPACES are mandatory before the argument and before the closing bracket.

### Functions Available:

#### **LINES( itab )**

<i>Semantics</i>	Returns the number of rows of an internal table <i>itab</i>
<i>Return value</i>	Number of rows
	Return value has ABAP type I.
<i>Argument</i>	Valid ABAP variable that represents an internal table  If an argument type other than ABAP internal table or an invalid symbol is specified, the program will stop when it reaches the breakpoint and a corresponding error message is issued.

#### **STRLEN( str )**

<i>Semantics</i>	Returns the current length of string <i>str</i>
<b>i Note</b> Trailing blanks in character fields with fixed lengths are not counted.	
<i>Return value</i>	Number of characters
	Return value has ABAP type I.
<i>Argument</i>	Valid ABAP variable of generic type CHARLIKE  If an argument type other than CHARLIKE or an invalid symbol is specified, the program will stop when it reaches the breakpoint and a corresponding error message is issued.

#### **XSTRLEN( str )**

<i>Semantics</i>	Returns the number of bytes
------------------	-----------------------------

<i>Return value</i>	Number of bytes
Return value has ABAP type I.	
<i>Argument</i>	Valid ABAP variable of type X or XSTRING
If an argument type other than X or XSTRING, or an invalid symbol is specified, the program will stop when it reaches the breakpoint and a corresponding error message is issued.	

## **INEXACT\_DF()**

<i>Semantics</i>	This function is used to identify inexact arithmetic operations that are executed with the numeric data type <code>decfloat16</code> or <code>decfloat34</code> .
<i>Return value</i>	'X' - if the final result of the an operation (for example, COMPUTE or MOVE) is inexact due to rounding  ' ' - otherwise (exact result).
<i>Argument</i>	No argument

## **6.5.4 Operators in Breakpoint Conditions**

The operators you can use in your logical expression along with breakpoint conditions are a subset of what you can use in ABAP source code. However, the syntax and semantics of this subset during runtime are the same:

- Standard comparison operators: `=`, `<`, `>`, `<=`, `>=`, `<>` or `EQ`, `LT`, `GT`, `LE`, `GE`, `NE`
- Logical operators: `NOT`, `AND`, `OR` and brackets to combine elementary conditions
- Operators for string analysis: `CO`, `CN`, `CA`, `NA`, `CS`, `NS`, `CP`, `NP`
- Special binary operators: `IS [NOT] INITIAL`, `IS [NOT] BOUND`, `IS [NOT] ASSIGNED`

## **Examples**

```
sy-subrc <> 0
oref is bound and oref->attr = 'abc'
( itab[1]-string is initial or not itab[1]-string co '0123456789' ) and ( itab[2]-
string is initial or not itab[2]-string co '0123456789' )
( I < J ) AND NOT oref->i GT dref->k OR X = Y[3]-COMP1 OR X = Y[2]-COMP1
```

# 7 Security Guide

## Target Audience

- System administrators
- ABAP developers

## Why Is Security Necessary?

With the increasing use of distributed systems for managing business data, the demands on security are also on the rise. When using a distributed system, you need to be sure that your data and processes support your business needs without allowing unauthorized access to critical information.

This Security Guide assists you in making the ABAP development process secure using the ABAP Development Tools (ADT).

## About This Document

The Security Guide comprises the following sections:

- [Resource Protection on Front-End Client \[page 806\]](#) explains how you can protect local resources of ABAP projects
- [Installing Plug-ins from Third Parties \[page 807\]](#) points to risks caused by installing third party plug-ins

## Additional Information

For more information about specific topics, see the Quick Links as shown in the table below.

Content	Quick Link on SAP Service Marketplace or SAP Community
SAP Community	<a href="#">Security Community</a>
Related SAP Notes	<a href="#">SAP Security Notes &amp; News</a>
Cloud SAP Cloud Platform ABAP Environment	<a href="#">SAP Cloud Platform ABAP Environment expert page</a>

## Related Information

 [Security Features at SAP Cloud Platform in the Cloud Foundry and Neo Environments](#)

 [Access Management in the Cloud Foundry Environment](#)

## 7.1 Resource Protection on Front-End Client

In ABAP Development Tools (ADT), an ABAP project provides a user-specific view of all development objects of the back-end system.

Like all other projects under Eclipse, projects, too, have a local representation of their data on the front-end and are managed in a workspace. In other words: When you have a project, local copies of development objects also exist on the front-end. This, in turn, means that both the metadata and the source code of development objects are also accessible outside the ABAP Repository at the level of the local file system.

## Risks

The potential dangers lie in the following areas:

- Metadata and sources of development objects being spied out by third parties
- ABAP source code being manipulated locally on the front-end by third parties. If external users have access to the local workspace folder, they have the chance to manipulate the development objects on the file system level. Data changed at this level could be then propagated into the ABAP Repository as "hidden changes". In this way, even "malicious" ABAP source code could find its way in to the business application system.

## Protection Measures

For the protection of local project resources, we strongly recommend the following protection measure:

Create your workspace folder for local storage of project resources in such a way that it cannot be read by third parties. Use the protection measures that are already provided at operating-system level.

### Note

Files that are located under Windows<sup>TM</sup> in the user's private folder subtree can only be accessed by the user himself/herself, and by any user who is a local administrator.

In particular, we recommend making use of the **default workspace that has been created with the IDE installation**.

## 7.2 Installing Plug-ins from Third Parties

Your installation of ABAP Development Tools can be enriched with additional plug-ins from various providers (3<sup>rd</sup> parties).

### Risks

These plug-ins can gain control over your client installation or even your complete front-end PC.

### Protection Measures

You should carefully decide on plug-ins that you are going to install. For this, only use plug-ins from trusted sources and which are using signatures.

# 8 What's New in ABAP Core Development

ABAP Development Tools (ADT) is released to customers in combination with the back end. This means, in order to use certain ADT functionalities, you need to provide the corresponding back end.

## ABAP Environment

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

- [Version 3.14 \[page 808\]](#)
- [Version 3.12 \[page 814\]](#)
- [Version 3.10 \[page 819\]](#)
- [Version 3.8 \[page 824\]](#)
- [Version 3.6 \[page 824\]](#)
- [Version 3.4 \[page 829\]](#)
- [Version 3.2 \[page 837\]](#)
- [Version 3.0 \[page 842\]](#)
- [Version 2.102 \[page 848\]](#)

## 8.1 Version 3.14

Get an overview of the most significant changes in ABAP core development that relate to the following:

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

### Note

All the features that are highlighted with a '\*' are client-specific and are therefore available for all supported ABAP systems.

In this topic, you will find release information about the following:

- [Profiling ABAP Code \[page 809\]](#)
- [Creating an ABAP Cloud Project \[page 810\]](#)
- [Finding Development Objects by their Documentation Status \[page 810\]](#)
- [Working with Business Services \[page 812\]](#)
- [Displaying Code Inspector Check Variants \[page 812\]](#)
- [Previewing Analytical Queries \[page 813\]](#)
- [Ensuring Quality of ABAP Code \[page 814\]](#)

## Cloud Profiling ABAP Code

### ABAP Cross Trace

The ABAP Cross Trace ...

- is designed for ABAP developers who build OData Services or other ABAP functionality related to the ABAP RESTful Application Programming Model (RAP).
- provides insights into the RAP runtime framework. This includes the processing of OData requests, for example, in SAP Fiori applications.
- includes functionality which is similar to the payload trace of SAP Gateway trace in SAP GUI. However, it does **not** include a replay functionality for OData requests.

This tool enables you, for example, to identify the root cause of an error.

The following sample represents the trace result of an OData request:

The screenshot shows the ABAP Cross Trace editor interface. At the top, there is a toolbar with various icons. Below the toolbar is a table titled 'type filter text' showing trace records. One record is highlighted with a yellow box and labeled 'Selected trace record in the editor'. The table columns include Procedure, Object, Message, Content Size (Bytes), Component, and Record Properties. The 'Record Properties' column shows a URL: <URI:/sap/opu/odata>. The table contains several entries related to OData requests and RAP processing. Below the table is a 'Record 20' view, which is a properties view. It has tabs on the left: General, Content, Call Stack, and Record Properties. The 'Content' tab is selected. It displays a hierarchical tree structure of trace data, with a tabular display of content at the bottom. A yellow box highlights the 'Content' tab and the tree structure, labeled 'Tabs of the Properties view'. Another yellow box highlights the tabular display, labeled 'Tabular display of the content from a trace record'. A yellow box highlights the toolbar at the top of the properties view, labeled 'Integrated toolbar'.

Sample for displaying trace results in the editor

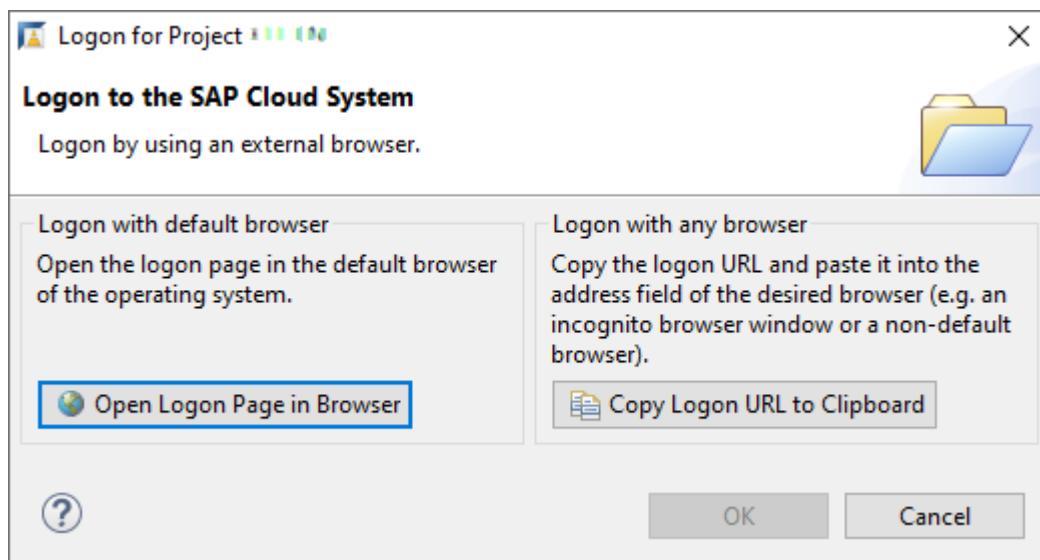
For more information, see

- ABAP Profiling [page 48]
- Working with the ABAP Cross Trace [page 683]

## Cloud Creating or Logging on to an ABAP Cloud Project

### Using an Existing Service Key

To logon to the ABAP back-end, you can use the default browser that is defined for your operating system or the browser of your choice. This approach follows state-of-the-art logon functionality.

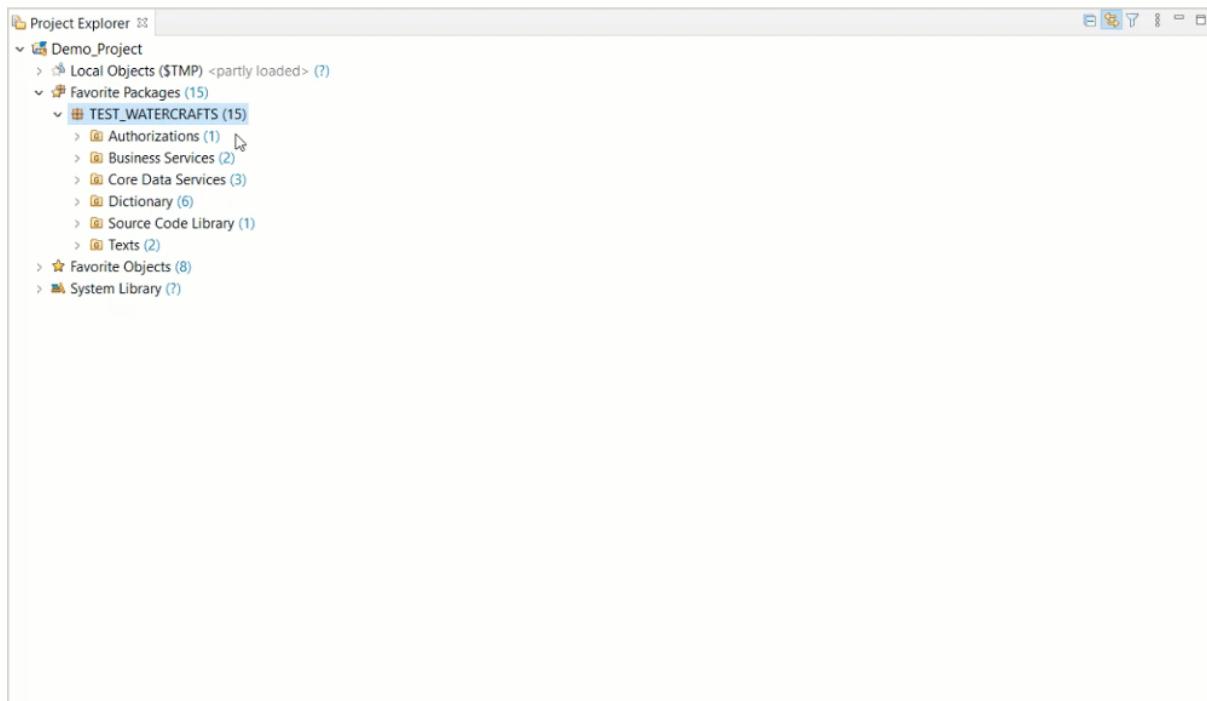


Dialog to Logon an SAP Cloud System

For more information, see [Using an Existing Service Key \[page 121\]](#)

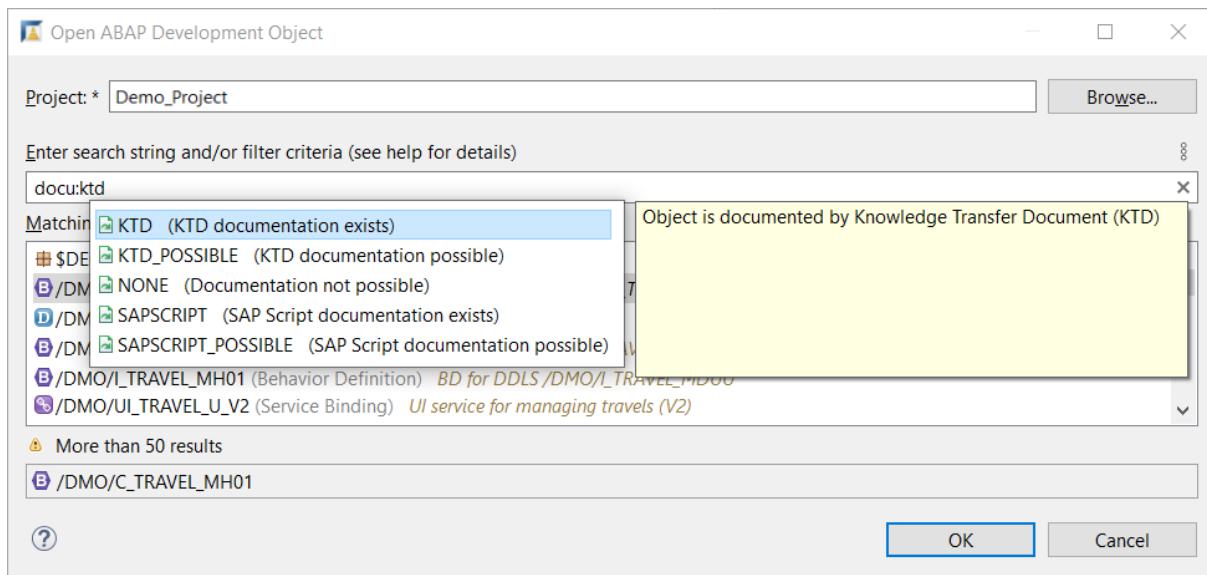
## Cloud Finding Development Objects by their Documentation Status

In the *Project Explorer*, you can now group the objects by their documentation status. Therefore, choose **► Expand Folder by... ► Documentation Status** from the context menu.



#### Finding Development Objects by their Documentation Status

Furthermore, you can now filter development objects according to their documentation status in the *Quick Search Dialog* (`Ctrl` + `Shift` + `A`).



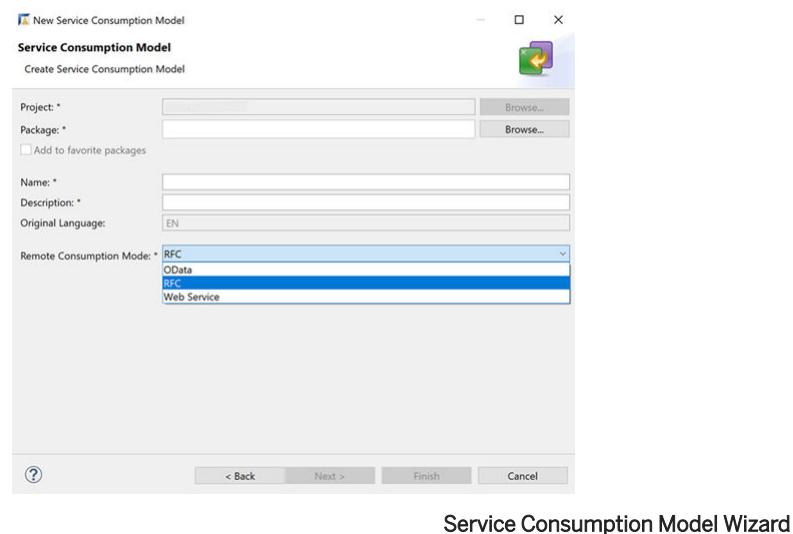
#### Filter Criteria in the Quick Search Dialog

For more information, see [Finding Development Objects by their Documentation Status](#) (video)

## Cloud Working with Business Services

### Creating Service Consumption Model for RFC

You can now create a service consumption model to generate proxies for remote function call (RFC).



**i** For more information, see [Generating Proxies for Remote Function Call \(RFC\) \[page 255\]](#)

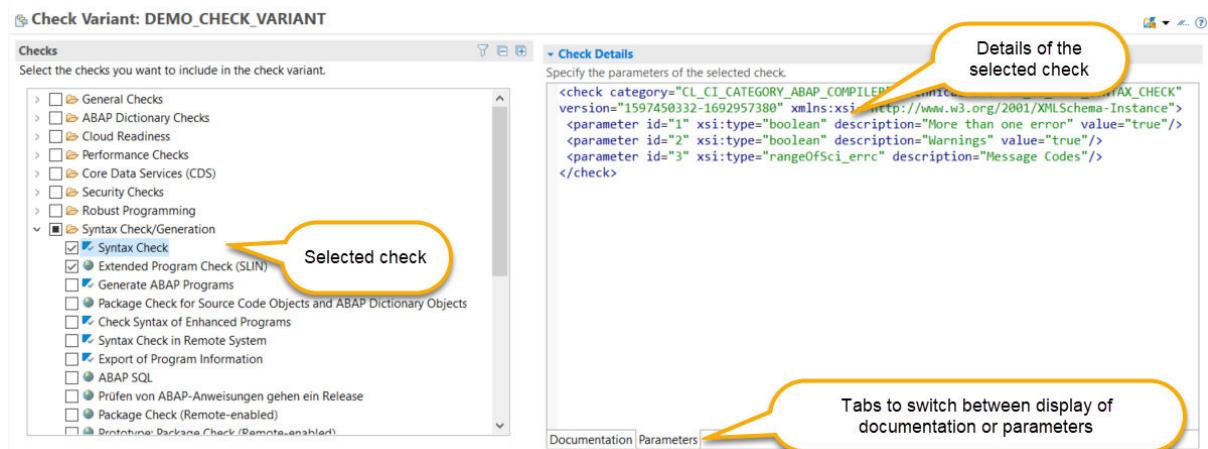
## Cloud Displaying ABAP Test Cockpit Check Variants

You can now display parameters and available documentation for ATC check variants with the check variant editor.

### **i** Note

The check variant editor is **only** available in display mode.

To start the check variant editor, select a check variant from the *Project Explorer* and open it. You can also search for available checks in your system with the search string `type:chkv`.

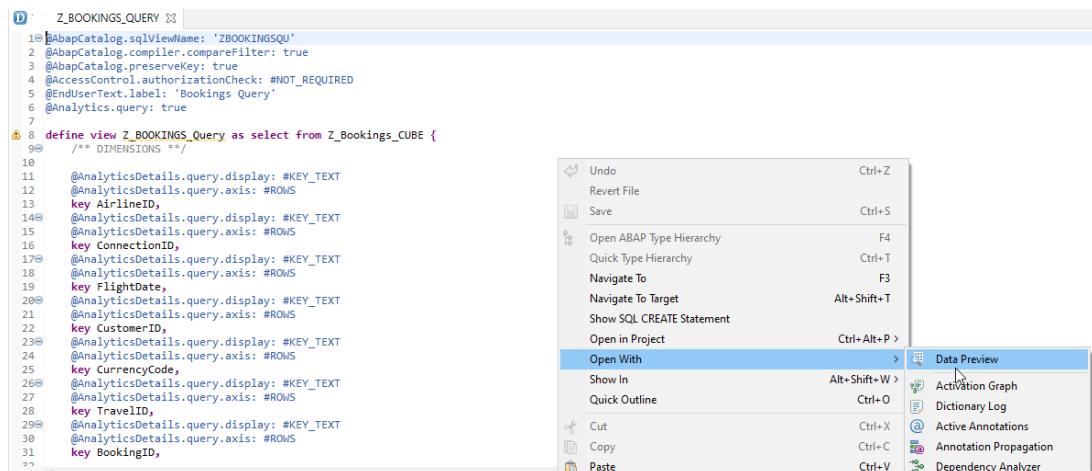


### Check Variant Editor

For more information, see [Displaying Check Variants \[page 599\]](#)

## Cloud Previewing Analytical Queries

You can now preview the data of analytical queries with the Preview Analytical Queries app.



### Open Preview Analytical Queries App

**Z\_BOOKINGS\_VE\_QUERY**(Bookings View Entity - CDS Data Model)

Travel ID	Booking Number	Booking Date	Customer ID	Airline ID	Customer Country	Customer City	Flight Number(Key)	Currency Code	Flight Date	Agency ID	Agency Name	Total of Bo	Run Query
1	1	06/10/2020	594	UA	DE	Heidelberg	UA1537	USD	06/27/2020	70041	Maxtrip	1	
1	2	06/27/2020	594	AA	DE	Heidelberg	AA322	USD	06/29/2020	70041	Maxtrip	1	
1	3	04/04/2021	594	UA	DE	Heidelberg	UA1537	USD	04/23/2021	70041	Maxtrip	1	
1	4	04/08/2021	594	AA	DE	Heidelberg	AA322	USD	04/25/2021	70041	Maxtrip	1	
2	1	06/25/2020	99	UA	ES	Sevilla	UA1537	USD	06/27/2020	70007	Hot Socks Travel	1	
2	2	06/25/2020	660	UA	FR	Grasse	UA1537	USD	06/27/2020	70007	Hot Socks Travel	1	
3	1	06/07/2020	93	UA	FR	Velizy	UA1537	USD	06/27/2020	70046	Hendrik's	1	
3	2	06/07/2020	706	UA	DE	Wiesloch	UA1537	USD	06/27/2020	70046	Hendrik's	1	
3	3	06/10/2020	93	AA	FR	Velizy	AA322	USD	06/29/2020	70046	Hendrik's	1	
3	4	06/10/2020	706	AA	DE	Wiesloch	AA322	USD	06/29/2020	70046	Hendrik's	1	
3	5	04/19/2021	93	UA	FR	Velizy	UA1537	USD	04/23/2021	70046	Hendrik's	1	
3	6	04/19/2021	706	UA	DE	Wiesloch	UA1537	USD	04/23/2021	70046	Hendrik's	1	
3	7	04/12/2021	93	AA	FR	Velizy	AA322	USD	04/25/2021	70046	Hendrik's	1	

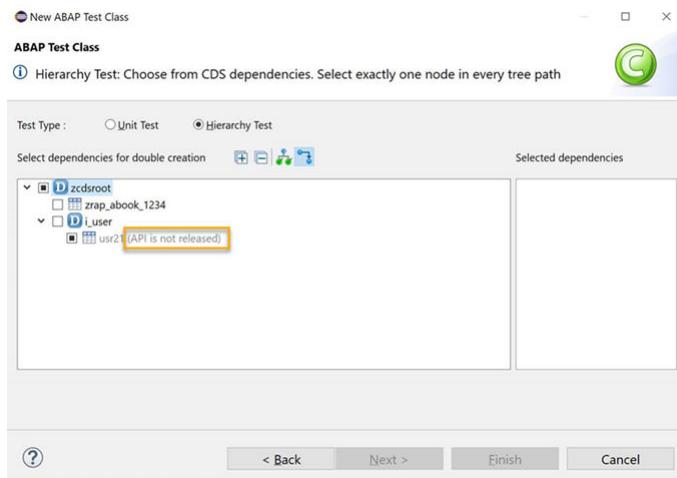
Preview Analytical Queries app

For more information, see [Previewing Analytical Queries \[page 486\]](#)

## Ensuring Quality of ABAP Code

### Displaying API Release State of Dependent Objects

While creating a new ABAP test class for a CDS entity using the wizard, the CDS dependency tree now displays the API release state of the dependent objects. You can only select the dependencies that have released APIs for creating test doubles. Until now, in the wizard, you could select the dependent objects with APIs that are not released. This was only identified later in the runtime during the test class execution.



ABAP Test Class Wizard

For more information, see [Creating a Test Class Using a Wizard \[page 519\]](#).

## 8.2 Version 3.12

Get an overview of the most significant changes in ABAP core development that relate to the following:

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

### Note

All the features that are highlighted with a '\*' are client-specific and are therefore available for all supported ABAP systems.

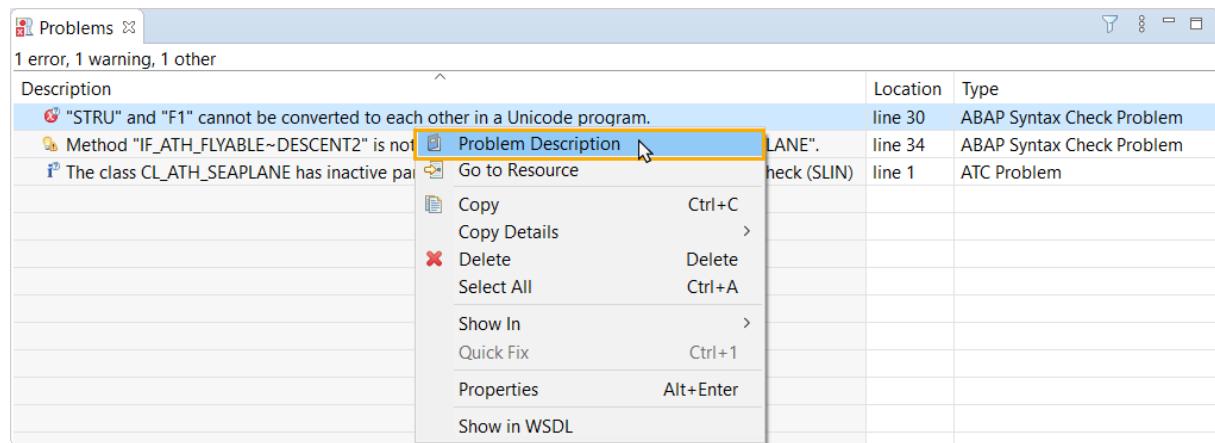
In this topic, you will find release information about the following:

- [Working with Problems View \[page 815\]](#)
- [Working with Business Add-Ins \(BAdIs\) \[page 815\]](#)
- [Working with Number Range Objects \[page 816\]](#)
- [Improved Class Creation Wizard \[page 817\]](#)
- [Working with Transport Organizer \[page 818\]](#)
- [Ensuring Quality of ABAP Code \[page 819\]](#)

## **Working with Problems View**

The *Problems View* displays problems (errors and warnings) with an icon and a message text. Sometimes an additional long text exists that can be displayed by selecting *Problem Description* from the context menu.

New: Now an additional decorator on the icon  informs you about the existence of the long text. It simplifies reading and understanding the problem messages.



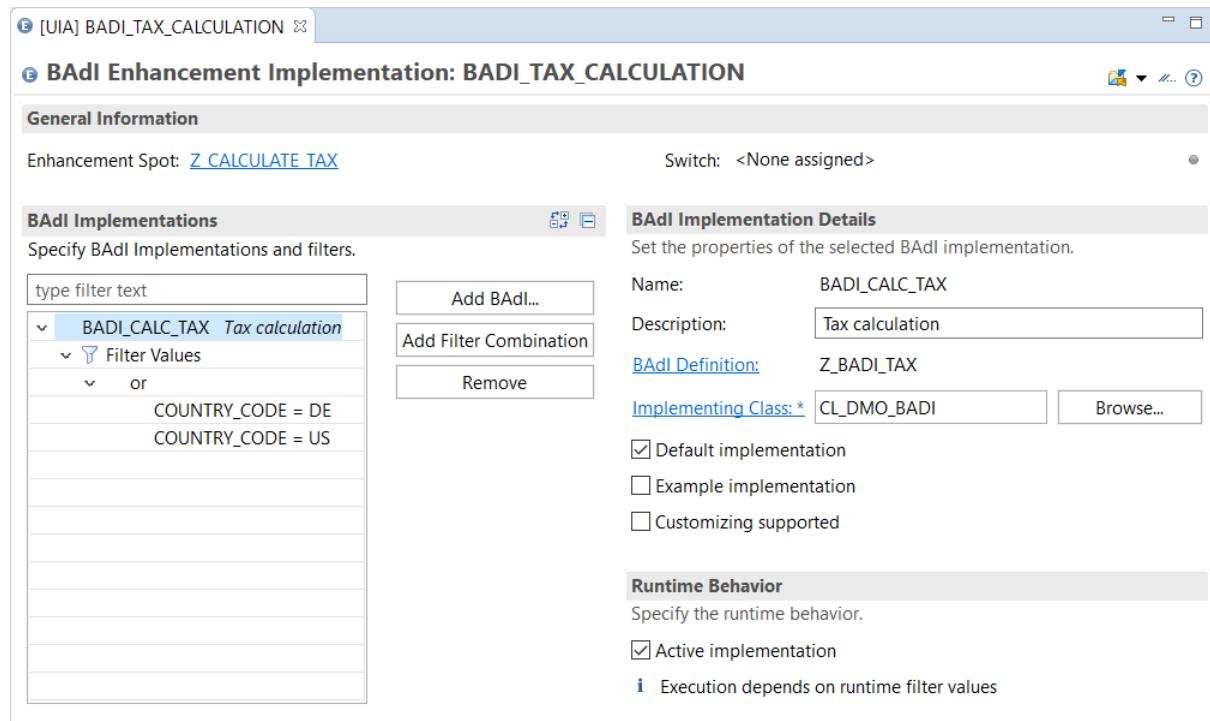
**Working with Problems View**

 For more information, see [Troubleshooting for ABAP Syntax Errors and Warnings \[page 180\]](#)

## **Working with Business Add-Ins (BAdIs)**

You can now create *BAdI Enhancement Spots* and define BAdI definitions using the *BAdI Enhancement Spot Editor*.

To implement BAdl definitions you can now create *BAdl Enhancement Implementations*, where you can add BAdl Implementations using the *BAdl Implementation Editor*.



BAdl Enhancement Implementation Editor

i For more information, see

- [Creating BAdl Enhancement Spots \[page 263\]](#)
- [Editing BAdl Enhancement Spots \[page 265\]](#)
- [Creating BAdl Implementations \[page 267\]](#)
- [Editing BAdl Implementations \[page 268\]](#)

## Cloud Working with Number Range Objects

You can now create *Number Range Objects* and edit them in the *Number Range Object Editor*.



```
1.. [YI3] MY_NROB ✎
1⑩ {
2  "header": {
3⑩   {
4    "description": "Demo number range object",
5    "masterLanguage": "EN",
6    "abapLanguageVersion": "default"
7  },
8  "interval": {
9⑩   {
10    "numberLengthDomain": "CHAR10",
11    "percentWarning": 10.0,
12    "subType": "",
13    "untilYear": false,
14    "rolling": true,
15    "prefix": false
16  },
17  "configuration": {
18⑩   {
19    "buffering": "none",
20    "bufferedNumbers": 0
21  }
22 }
```

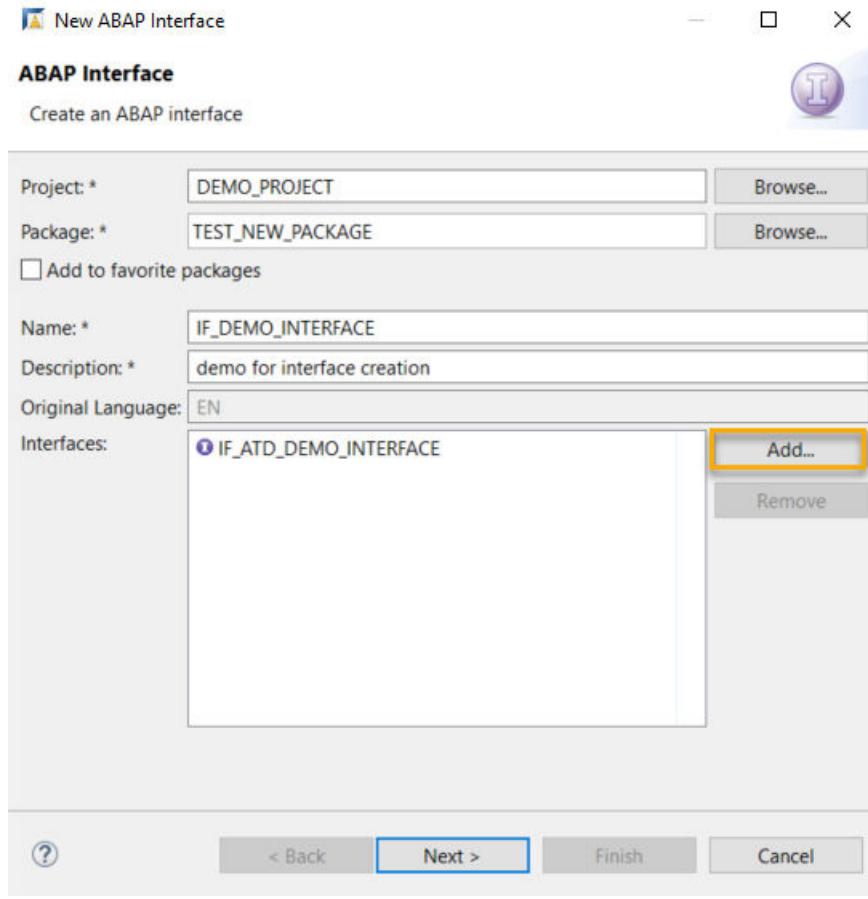
Number Range Object Editor

**i** For more information, see

- [Creating Number Range Objects \[page 245\]](#)
- [Editing Number Range Objects \[page 247\]](#)

## Improved Class Creation Wizard

When creating a class, you can now add one or more interfaces within the class creation wizard.



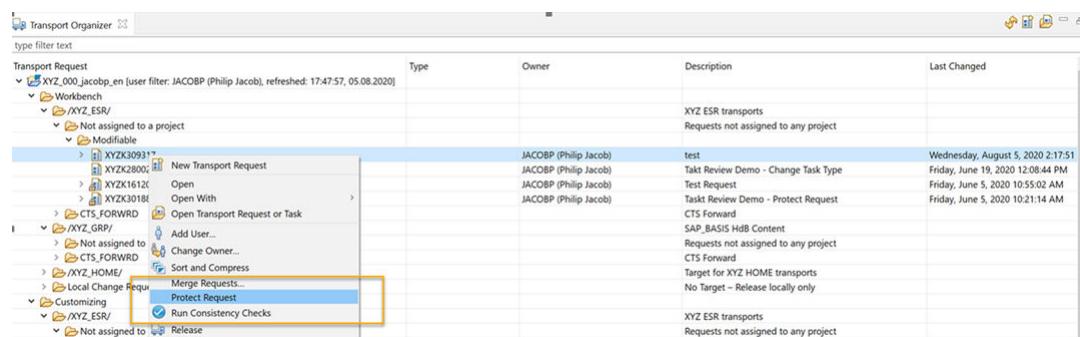
Class Creation Wizard

**i** For more information, see [Creating ABAP Classes \[page 185\]](#)

## Working with Transport Organizer

### Protecting a Transport Request

You can now ensure that only the owner of the request adds more users to it by protecting the request.



Protect Transport Request

**i** For more information, see [Protecting a Transport Request \[page 454\]](#).

## Changing a Task Type

You can now change the task type of a task that is added under a transport request.

**i** For more information, see [Changing a Task Type \[page 455\]](#).

## Cloud Ensuring Quality of ABAP Code

### Supporting CDS View Entities and Projection Views in Test Double Frameworks

The CDS test double framework and ABAP SQL test double framework now supports the CDS view entities and projection views.

**i** For more information, see

- [ABAP CDS Test Double Framework \[page 516\]](#).
- [ABAP SQL Test Double Framework \[page 539\]](#).

### Testing ABAP Authority-Check Statements using Test Helper API

Until now, there was limited support for testing role-based functionality using an AUTHORITY-CHECK statement in ABAP. Writing unit tests by configuring various users with the required roles and authorizations is complex. The ABAP Authority Check Test Helper API is a secure API-based approach that can be used in an ABAP Unit test class.

With ABAP Authority Check Test Helper API, you can now configure authorizations for single or multiple users, set combination of expectations, and get comprehensive log summary of the AUTHORITY-CHECK statements.

**i** For more information, see [Managing Dependencies on ABAP Authority Checks with ABAP Unit \[page 545\]](#).

## 8.3 Cloud Version 3.10

Get an overview of the most significant changes in ABAP core development that relate to the following:

- Client: ABAP Development Tools (ADT) **3.10**
- Back end version:  SAP Cloud Platform ABAP Environment **2005**

### **i** Note

Features that are highlighted with a '\*' are client-specific and are therefore available for all supported ABAP systems.

In this topic, you will find release information about the following:

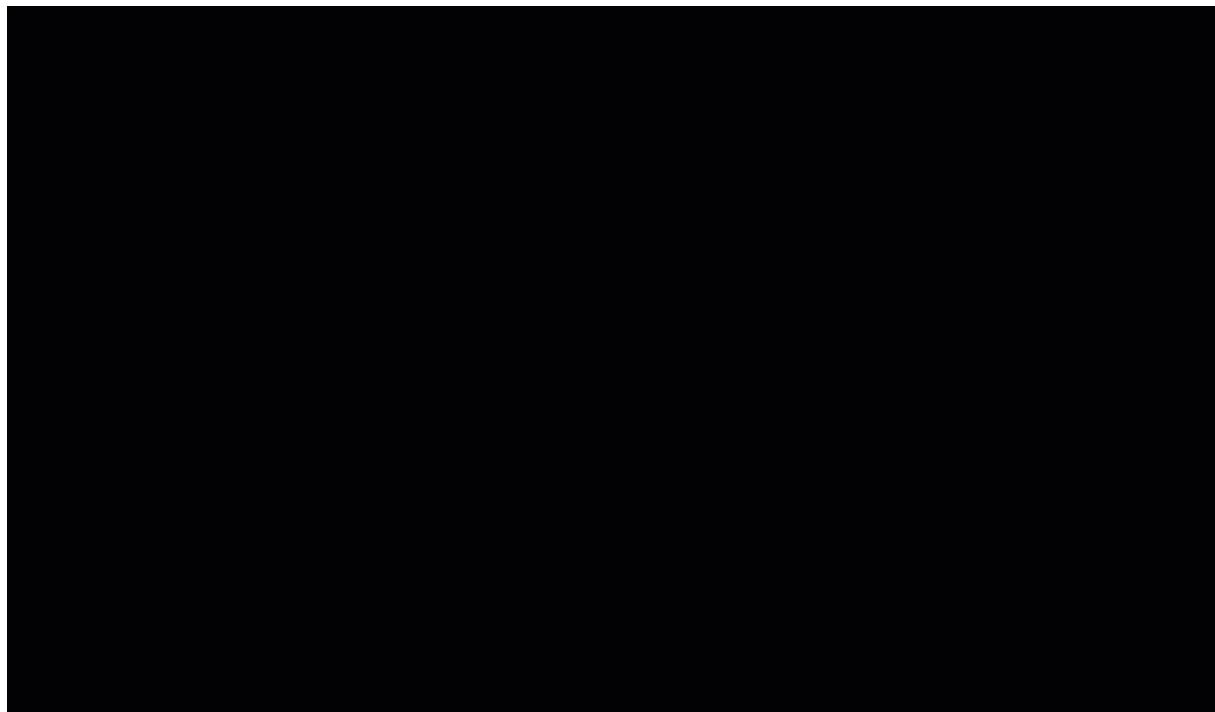
-  [Documenting Development Objects \[page 820\]](#)
-  [Working with ABAP Dictionary Objects \[page 820\]](#)
-  [Consuming Services in a UI \[page 821\]](#)
- [Working with Collaborative Tools \[page 821\]](#)

- [Comparing Source Code \[page 822\]](#)
- [Working with Business Services \[page 823\]](#)

## **Documenting Development Objects**

### **Linking to Development Objects**

In the *Knowledge Transfer Document Editor*, you can now link to development objects and their elements using content assist functionality.



Linking to development objects

 For more information, see

- [Linking to Development Objects \[page 261\]](#)
- [Knowledge Transfer Document \[page 30\]](#)

## **Working with ABAP Dictionary Objects**

### **Adjusting Database Tables**

Changing the type of a database table field can result in the need for a database adjustment, as the simple activation does not work in these cases. Until now, it was not possible to trigger these kinds of adjustments from ABAP Development Tools. You can now apply quick fixes to adjust the database.

 For more information, see [Adjusting Database Tables \[page 243\]](#).

## Cloud Consuming Services in a UI

### Creating Business Role Templates

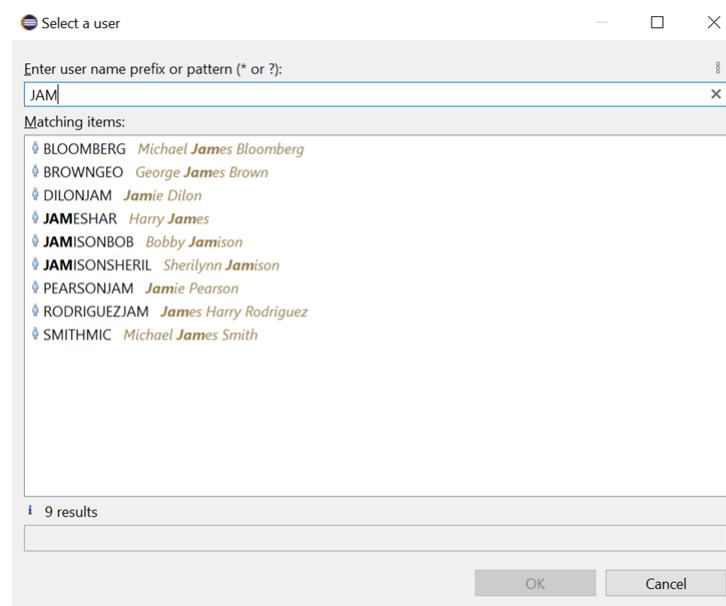
You can now create business role templates and assign business catalogs to them. These business role templates are then available in the *Business Role Templates* app and you can use them as a basis for creating new business roles.

**i** For more information, see [Creating a Business Role Template \[page 126\]](#).

## Working with Collaborative Tools

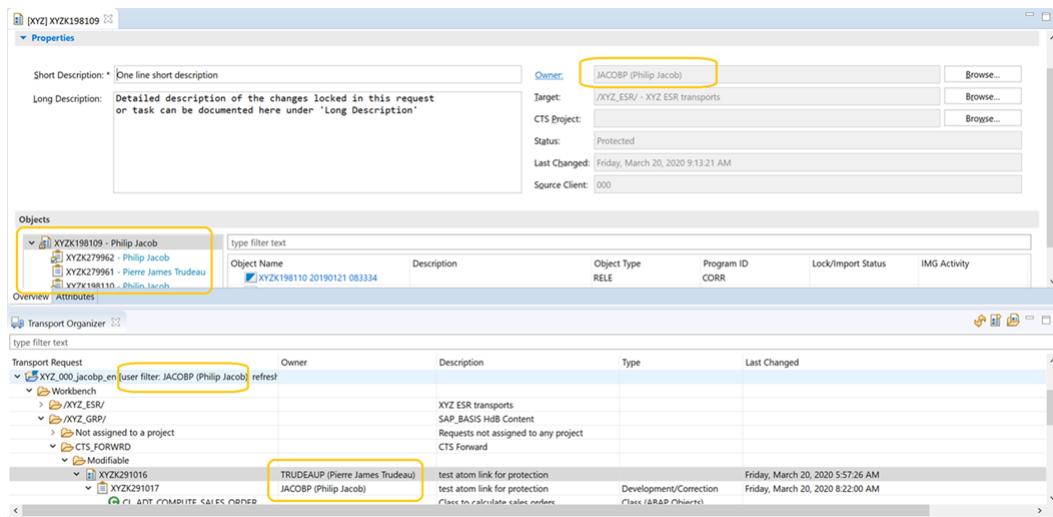
### Searching for a User

You can now search for a user in the user value help using the first name, last name or middle name. You can also search for the users with their real usernames in the user selection dialog



### Improved Usernames

Until now, it was difficult to identify users with only their technical usernames. It is essential to identify users in tools where collaboration among users is required. You can now see the business user's real username along with the technical name in the *Properties* view, *Runtime Error* viewer, *Transport Organizer* view, *Transport Request/Task* editors, and *History* view.



Displaying username in Properties view, Transport Organizer view



Displaying username in Runtime Error view

## Comparing Source Code

### Compare Viewer

The [ABAP Compare](#) viewer now shows different colors, depending on whether the text has been added, removed or modified. The default colors are green, red and grey respectively.

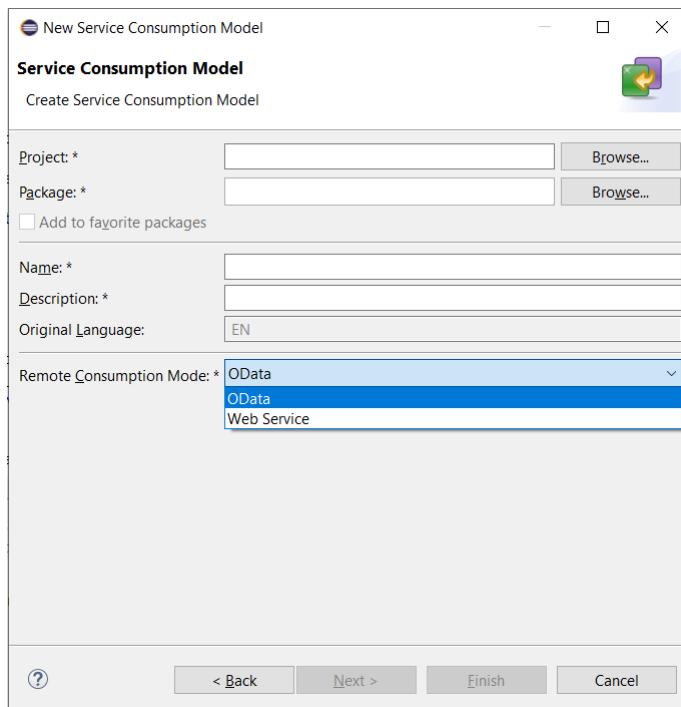
ABAP Compare

CL_DEMO_CLASS_1 (Global Class) - Baseline	CL_DEMO_CLASS_1 (Global Class) - Working Copy
<pre> 1 CLASS cl_demo_class_1 DEFINITION 2 PUBLIC 3 FINAL 4 CREATE PUBLIC 5 for testing. 6 PRIVATE SECTION. 7 CLASS-DATA: 8   environment TYPE REF TO if_cds_test_environment. 9 CLASS-METHODS: 10  class_setup 11    RAISING 12    cx_static_check, 13  class_teardown, 14  class_hello, 15  class_hello1. 16  DATA: 17    act_results_1      TYPE STANDARD TABLE OF cdsfrwk_sales_order_item WITH 18    act_results      TYPE STANDARD TABLE OF cdsfrwk_sales_order_item WITH 19    test_data      TYPE REF TO if_cds_test_data, 20    sales_order_items TYPE STANDARD TABLE OF smwd_so_i WITH EMPTY KEY. 21  METHODS: 22    setup RAISING cx_static_check, 23    amount_100_tax_0_ok FOR TESTING RAISING cx_static_check, 24    amount_100_tax_025_ok FOR TESTING RAISING cx_static_check, 25    amount_333_tax_111_ok FOR TESTING RAISING cx_static_check, 26    amount_15_tax_7_ok FOR TESTING RAISING cx_static_check, 27    amount_20_tax_10_ok FOR TESTING RAISING cx_static_check, 28    amount_30_tax_20_ok FOR TESTING RAISING cx_static_check, 29    amount_40_tax_30_ok FOR TESTING RAISING cx_static_check, 30    amount_50_tax_40_ok FOR TESTING RAISING cx_static_check. 31 32 ENDCLASS. 33 </pre>	<pre> 1 CLASS cl_demo_class_1 DEFINITION 2 PUBLIC 3 FINAL 4 CREATE PUBLIC 5 for testing. 6 PRIVATE SECTION. 7 CLASS-DATA: 8   environment TYPE REF TO if_cds_test_environment. 9 CLASS-METHODS: 10  class_teardown, 11  class_hello1. 12  DATA: 13    act_results_1      TYPE STANDARD TABLE OF cdsfrwk_sales_order_item WITH 14    act_results      TYPE STANDARD TABLE OF cdsfrwk_sales_order_item WITH 15    test_data      TYPE REF TO if_cds_test_data, 16    test_data_1      TYPE REF TO i, 17    test_data_2      TYPE i, 18    test_data_3      TYPE i, 19    test_data_4      TYPE c, 20    sales_order_items TYPE STANDARD TABLE OF smwd_so_i WITH EMPTY KEY. 21  METHODS: 22    setup RAISING cx_static_check, 23    amount_20_tax_10_ok FOR TESTING RAISING cx_static_check, 24    amount_30_tax_20_ok FOR TESTING RAISING cx_static_check, 25    amount_40_tax_30_ok FOR TESTING RAISING cx_static_check, 26    amount_60_tax_50_ok FOR TESTING RAISING cx_static_check, 27    amount_70_tax_60_ok FOR TESTING RAISING cx_static_check, 28    amount_80_tax_70_ok FOR TESTING RAISING cx_static_check, 29    amount_90_tax_80_ok FOR TESTING RAISING cx_static_check, 30    amount_50_tax_40_ok FOR TESTING RAISING cx_static_check. 31 32 ENDCLASS. 33 </pre>

## Working with Business Services

### Creating Service Consumption Model

You can now create a service consumption model to generate proxies for a remote Web service also.



For more information, see [Creating Service Consumption Model \[page 248\]](#).

## 8.4 Version 3.8

Get an overview of the most significant changes in ABAP core development that relate to the following:

- Client: ABAP Development Tools (ADT) **3.8**
- Back end version:  SAP Cloud Platform ABAP Environment **2002**

### Note

All the features that are highlighted with a '\*' are client-specific and are therefore available for all supported ABAP systems.

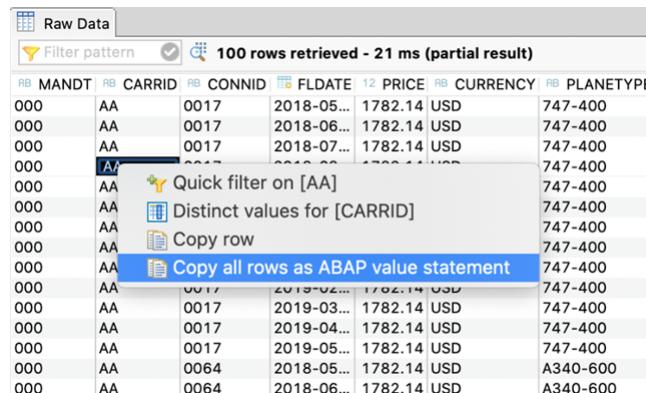
In this topic, you will find release information about the following:

- [Working with Data Preview \[page 824\]](#)

## Working with Data Preview

### Copying Rows as ABAP Value Statement

You can now copy all the rows to the clipboard as an ABAP value statement in [Data Preview](#). This enables you to copy the rows as ABAP value statement and use it in the ABAP code to work with internal tables.



Copy all rows as ABAP value statement

MANDT	CARRID	CONNID	FLDATE	PRICE	CURRENCY	PLANETYPE
000	AA	0017	2018-05...	1782.14	USD	747-400
000	AA	0017	2018-06...	1782.14	USD	747-400
000	AA	0017	2018-07...	1782.14	USD	747-400
000	AA	0017	2018-08...	1782.14	USD	747-400
000	AA	0017	2018-09...	1782.14	USD	747-400
000	AA	0017	2018-10...	1782.14	USD	747-400
000	AA	0017	2018-11...	1782.14	USD	747-400
000	AA	0017	2018-12...	1782.14	USD	747-400
000	AA	0017	2019-01...	1782.14	USD	747-400
000	AA	0017	2019-02...	1782.14	USD	747-400
000	AA	0017	2019-03...	1782.14	USD	747-400
000	AA	0017	2019-04...	1782.14	USD	747-400
000	AA	0017	2019-05...	1782.14	USD	747-400
000	AA	0064	2018-05...	1782.14	USD	A340-600
000	AA	0064	2018-06...	1782.14	USD	A340-600

 For more information, see [Copying Rows as ABAP Value Statement \[page 484\]](#).

## 8.5 Version 3.6

Get an overview of the most significant changes in ABAP core development that relate to the following:

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

## i Note

All the features that are highlighted with a '\*' are client-specific and are therefore available for all supported ABAP systems.

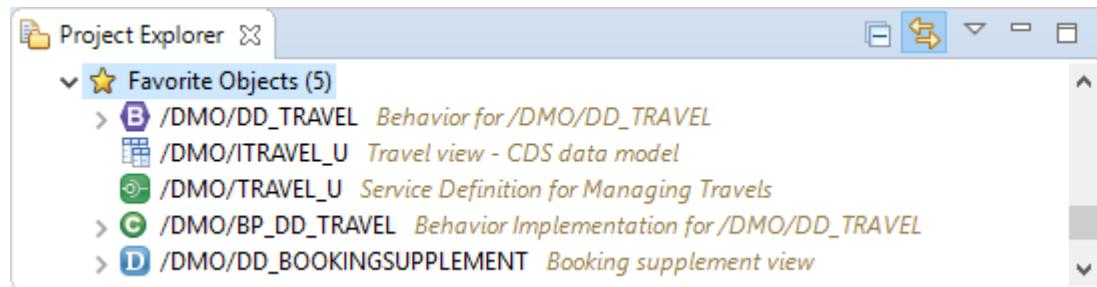
In this topic, you will find release information about the following:

- [Working with ABAP Repository Trees \[page 825\]](#)
- [Working with Transport Organizer \[page 825\]](#)
- [Ensuring Quality of ABAP Code with ATC \[page 827\]](#)
- [Working with Data Preview \[page 827\]](#)
- [Working with Development Objects \[page 828\]](#)
- [Working with ABAP Cloud Projects \[page 828\]](#)
- [Installing ADT on the Open Eclipse 2019-09 \(4.13\) Platform\\* \[page 829\]](#)

## Working with ABAP Repository Trees

### Working with Favorite Objects

You can now add objects that you work with frequently to the *Favorite Objects* tree for quick access.



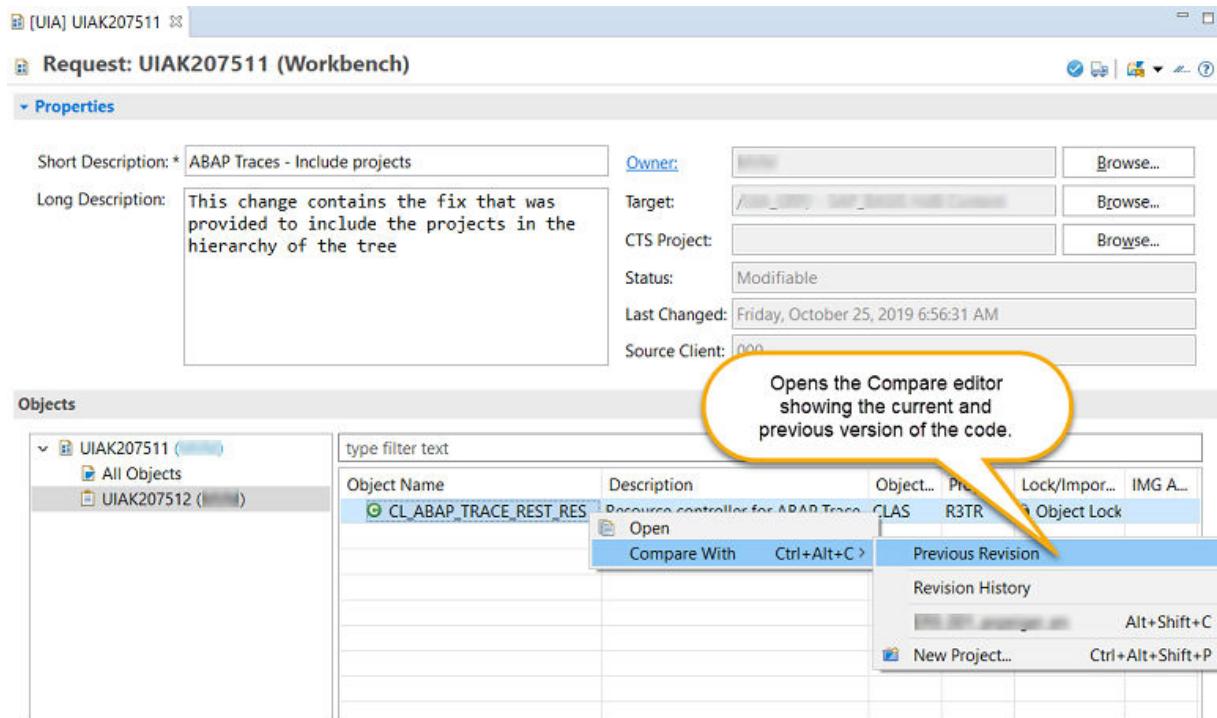
Favorite objects tree

i For more information, see [Working with Favorite Objects \[page 154\]](#).

## Working with Transport Organizer

### Comparing with Previous Version

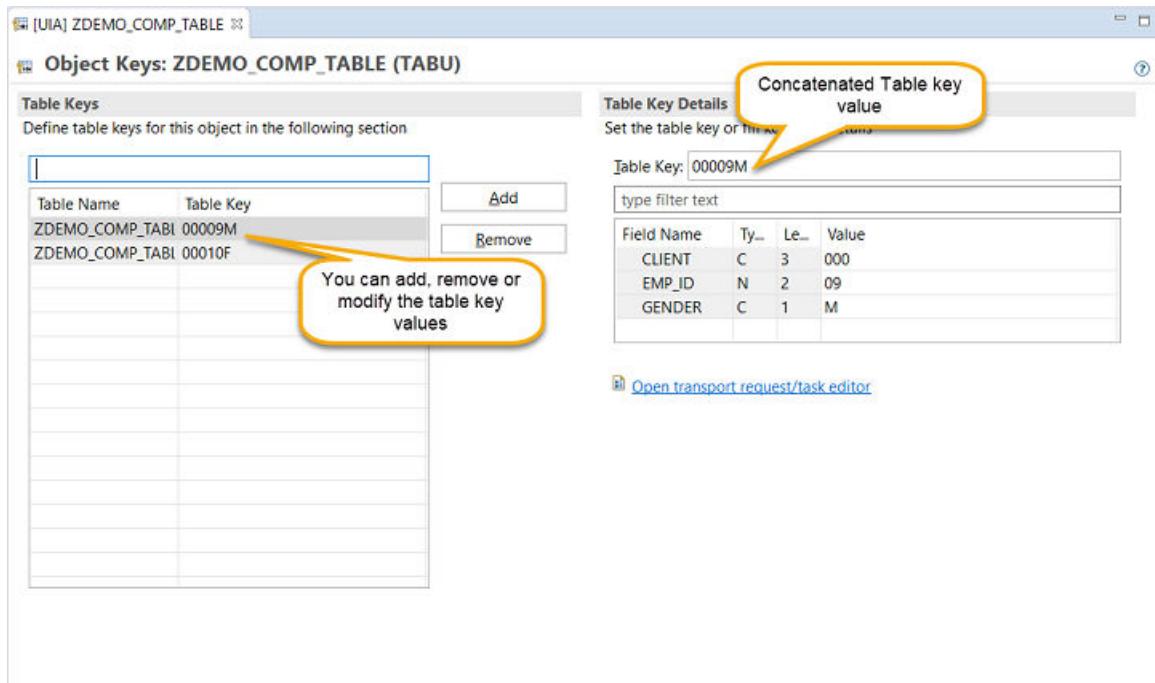
The *Transport Request* editor now provides a feature to compare the changes made to the source code with the last revised version that was transported. This option is also available in the *ABAP Source Code editor* and the *Project Explorer* view.



For more information, see [Comparing with Previous Revision \[page 459\]](#).

## Specifying Table Keys

The *Transport Request* editor and *Transport Organizer* view now allow you to edit one or more entries of the table locked in a transport request.

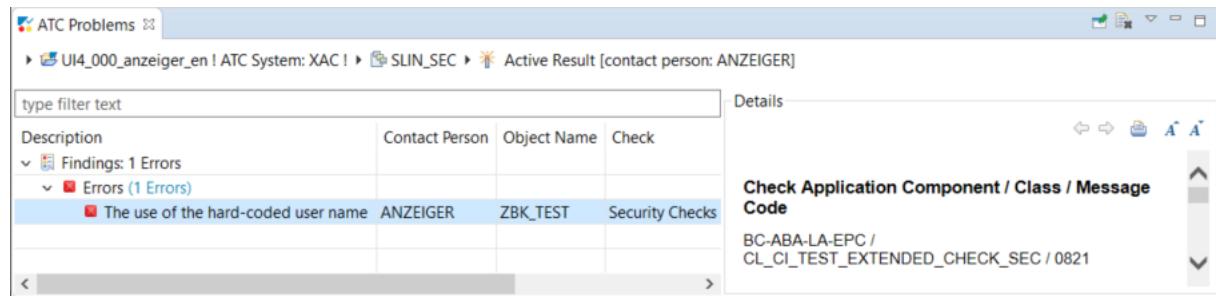


For more information, see [Specifying Table Keys \[page 459\]](#).

## Cloud Ensuring Quality of ABAP Code with ATC

### Displaying Local Active Results in the ATC Problems View

You can now display local active results, that are assigned to you, in the ATC Problems view.



Description	Contact Person	Object Name	Check
Findings: 1 Errors			
Errors (1 Errors)	ANZEIGER	ZBK_TEST	Security Checks
<b>The use of the hard-coded user name</b>	<b>ANZEIGER</b>	<b>ZBK_TEST</b>	<b>Security Checks</b>

Details

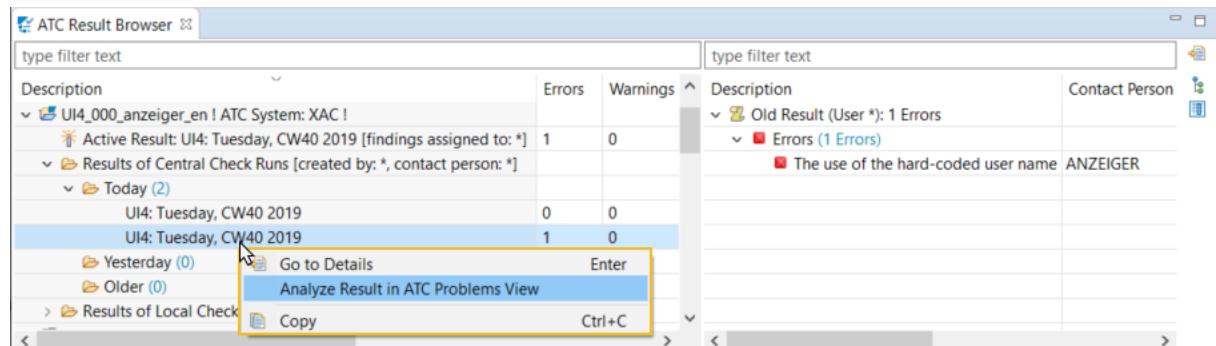
Check Application Component / Class / Message Code  
BC-ABA-LA-EPC / CL\_CI\_TEST\_EXTENDED\_CHECK\_SEC / 0821

Local active results in ATC Problems view

For more information, see [Displaying Local Active Results in the ATC Problems View \[page 589\]](#).

### Analyzing Local ATC Results in the ATC Problems View

You can now analyze and fix ATC issues that are displayed in the ATC Result Browser, for example, by applying quick fixes.



Description	Errors	Warnings
UI4: Tuesday, CW40 2019	1	0
UI4: Tuesday, CW40 2019	0	0
UI4: Tuesday, CW40 2019	1	0

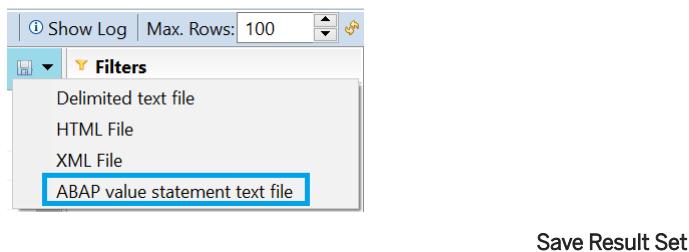
For more information, see [Analyzing Local ATC Results from the ATC Result Browser \[page 607\]](#).

## Cloud Working with Data Preview

### Saving Result Set

You can now save data in a text file as an ABAP value statement in the *Data Preview*. This enables you to copy the ABAP value statement from the file and use it in the ABAP code to work with internal tables.

To save data in a text file, choose *ABAP value statement text file* from the save drop-down list.



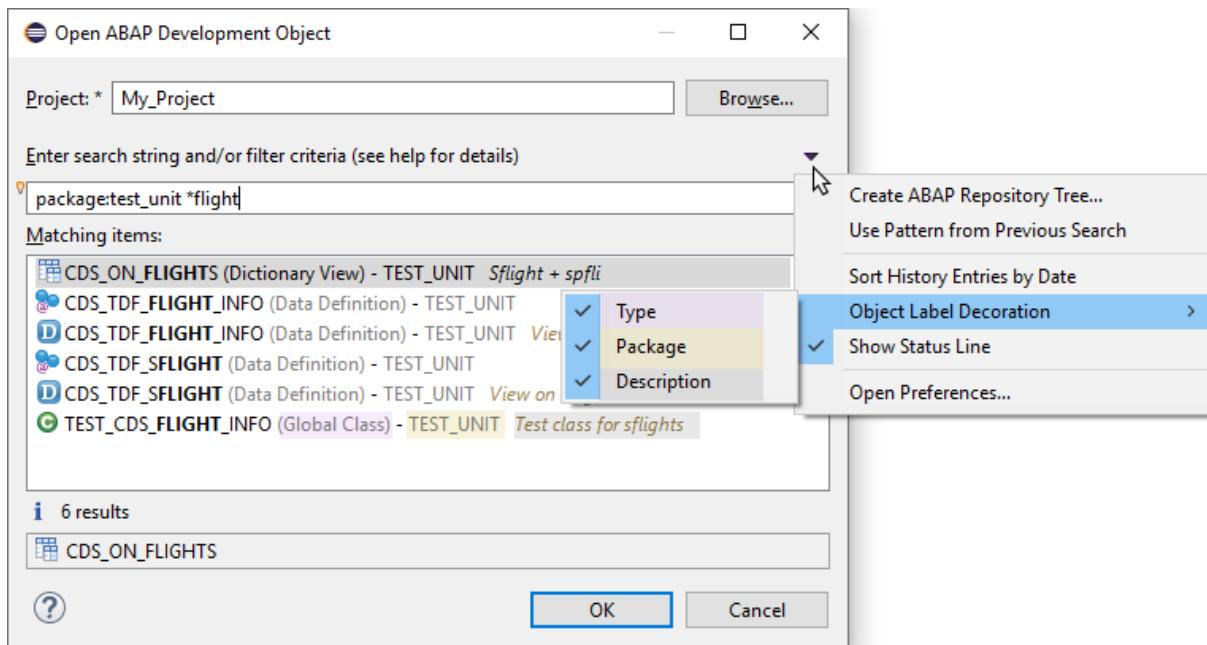
**i** For more information, see [Saving Result Set \[page 482\]](#).

## Working with Development Objects

### Open ABAP Development Object Dialog

There have been several improvements on the Open ABAP Development Object dialog. You can now configure object label decoration and define if type, package, or description of the objects should be displayed along with the objects.

The matches are now highlighted according to the search query.



Open ABAP Development Object Dialog

**i** For more information, see ['Open ABAP Development Object' Dialog \[page 160\]](#).

## Working with ABAP Cloud Projects

You can now define necessary authorizations by means of a communication scenario.

**i** For more information, see [Consuming Services in the Context of API with Technical Users \[page 127\]](#).

## Installing ADT on the Open Eclipse 2019-09 (4.13) Platform\*

You can now install the ADT client on the latest Eclipse 2019-09 (4.13) platform.

This ADT client version supports the following Eclipse platforms:

- 2019-09 (4.13)
- 2019-06 (4.12)

**i** For more information, see

- [Installing ABAP Development Tools](#)
- [Eclipse 4.13 - New and Noteworthy](#).

## 8.6 Version 3.4

Get an overview of the most significant changes in ABAP core development that relate to the following:

- Client: ABAP Development Tools (ADT) **3.4**
- Back end version:  SAP Cloud Platform ABAP Environment **1908**

### **i** Note

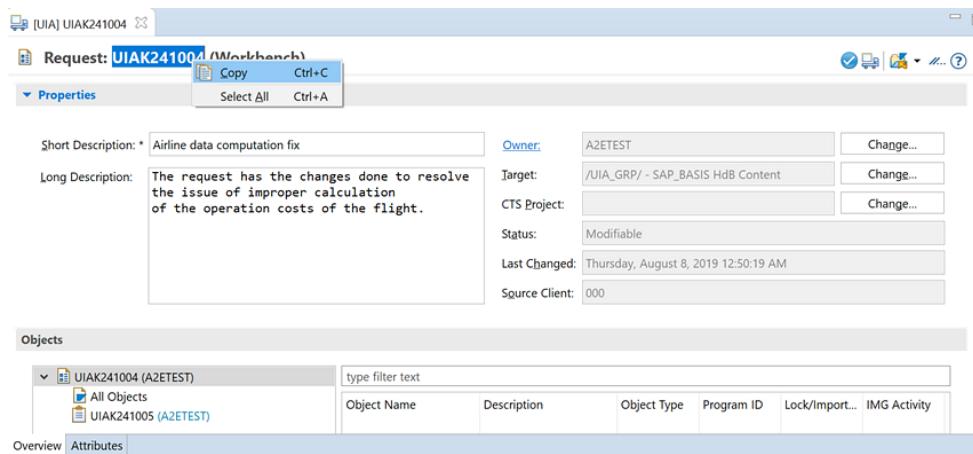
All the features that are highlighted with a '\*' are client-specific and are therefore available for all supported ABAP systems.

In this topic, you will find the release information about the following areas:

- [ABAP Form-Based Editor \[page 829\]](#)
- [Working with Relation Explorer \[page 830\]](#)
- [Working with Transport Organizer \[page 831\]](#)
- [Ensuring Quality of ABAP Code \[page 833\]](#)
- [Using Troubleshooting Tools \[page 834\]](#)
-  [Defining Restriction Fields and Restriction Types \[page 835\]](#)
-  [Working with the HTTP Service Editor \[page 835\]](#)
-  [Adaptation of ABAP Source Code for SAP Cloud Platform \[page 836\]](#)
- [Installing ADT on the Open Eclipse 2019-06 \(4.12\) Platform \[page 837\]](#)

## ABAP Form-Based Editor

You can now copy the title from the form-based editor to your clipboard.



### i Note

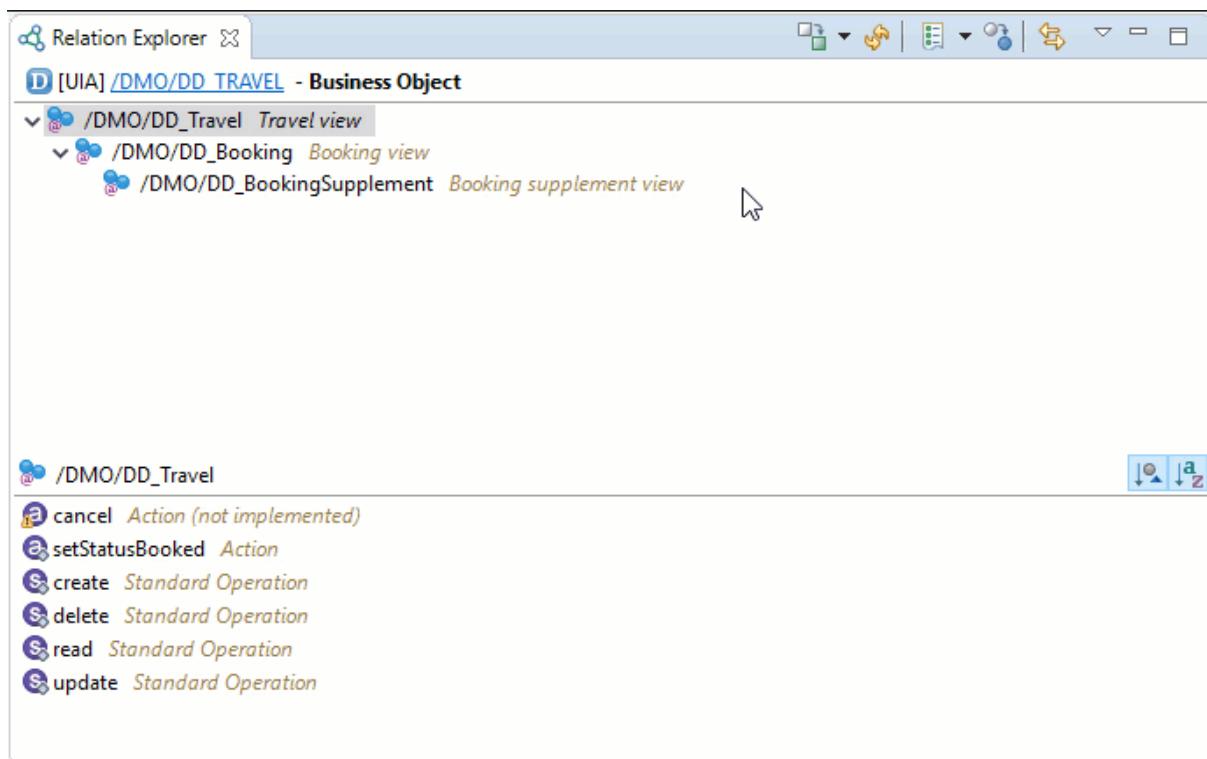
This feature is available in an ADT client installed on the latest Eclipse 2019-06 platform.

## Working with Relation Explorer

### Exploring Business Objects

A business object consists of hierarchical connected entities. The behavior for each entity is defined in the behavior definition object and implemented in the behavior classes. In the *Relation Explorer*, you can see structure and behavior of a certain business object independent of the technical location. You can navigate to all the entities and the corresponding behavior (definition and implementation).

Sometimes you might be interested in more CDS-specific aspects and want to see access control lists or test classes. You can achieve this by switching the context from **Business Object** to **Core Data Services** context, as you can see in the following animation.



Switching the context in the Relation Explorer

For more information, see

- [Relation Explorer \[page 36\]](#)
- 

## Working with Transport Organizer

### Configure the transport organizer tree

Once the transport organizer tree is configured in an ADT instance, transport organizer view displays the content based on the same configuration across ADT instances in any machine for the same ABAP project, client and login user.

For more information, see [Configuring Tree \[page 440\]](#)

### Compare objects of a request/task with other revisions

You can now navigate to the [History](#) view of an ABAP object directly from the transport request/task editor to view all the previous revisions of that ABAP object. From the history view you can compare with the required revision.

Object Name	Description	Object Type	Program ID
CDS_VIEW_CONSUME_CUSTOM	Consuming custom entity	DDLS	R3TR
COMPANY_NAME_FIELD	Data element for company name - INT	DTEL	R3TR
CL_ADT_COMPUTE_SALES_ORDER====CCAU		CINC	LIMU
CL_ADT_READ_FLIGHT_DETAILS		CLSD	LIMU
CL_ADT_READ_FLIGHT_DETAILS		CPUB	LIMU
CL_ADT_SFLIGHT_OPERATION=====CCAU		CINC	LIMU

You can also compare the changes in an object with other systems available in your ADT directly from the request/task editor itself.

For more information, [Comparing the Change in Objects \[page 457\]](#)

## Working with Development Objects

### Using ADT Links in All Supported Operating Systems\*

You can now click ADT links in **all** supported operating systems.

This means, when clicking an ADT link, the relevant development object is opened in ADT directly.

Before you can click ADT links, select the `adt` schema on the new [General > Link Handlers](#) preference page in advance.

Link handlers enable you to handle custom URL schemas, such as the `adt` schema.

**i** For more information, see

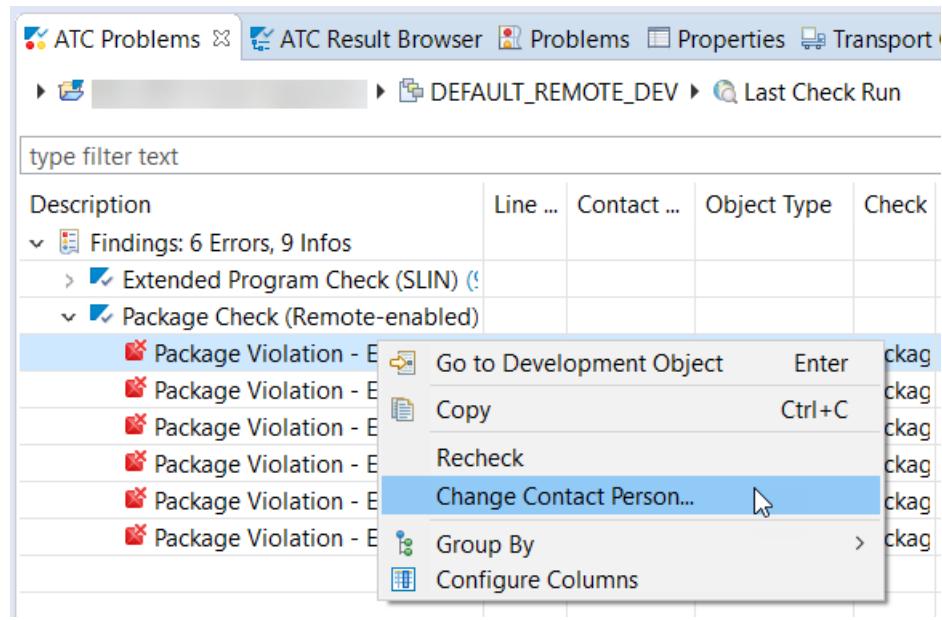
- [Sharing and Opening a Link to a Development Object \[page 166\]](#)
- [Link Handlers](#)
- [Added support for custom URL schemes](#)

## Ensuring Quality of ABAP Code

### Changing the Contact Person of ATC Findings

You can now change the contact person of ATC findings, if you want to assign the ATC findings to a certain developer.

To do so, select the ATC findings in question in the *ATC Problems* view and choose *Change Contact Person* in the context menu.

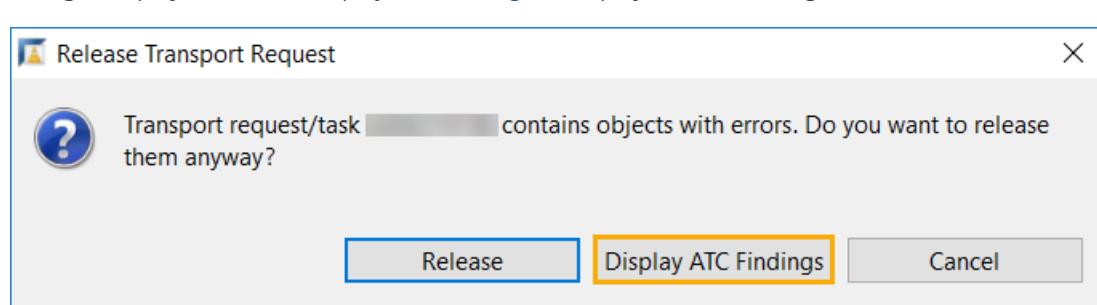


For more information, see [Changing the Contact Person of ATC Findings \[page 587\]](#)

### Displaying ATC Findings When Releasing Transport Requests

You can now display ATC findings when releasing a transport request.

When releasing a transport request in the *Transport Organizer* view, an ATC check run is launched automatically in the background. If the ATC check run reports ATC findings, the *Release Transport Request* dialog is displayed. Choose *Display ATC Findings* to display the ATC findings in the *ATC Problems* view.



For more information, see [Launching ATC Check Implicitly \[page 601\]](#)

## Displaying Results from the Reference Check System

You can now display central check results from the reference check system directly in *ATC Result Browser*.

Description	Errors	Warnings	Infos	Check Variant	Executed On
Local central check result	42	2	15	SLIN_SEC	10/07/19 14:48
Central check result from the reference check system	2720	913	200	SLIN_SEC	10/07/19 14:47
Active Result: Remote Central Result 2: Wednesday, CW28 2019 [findings a	2720	913	200	SLIN_SEC	10/07/19 14:47
Results of Central Check Runs [created by: *, contact person: *]					
Today (5)					
Local Central Result 3: Wednesday, CW28 2019	42	2	15	SLIN_SEC	10/07/19 14:48
Remote Central Result 2: Wednesday, CW28 2019	2720	913	200	SLIN_SEC	10/07/19 14:47
Local Central Result 2: Wednesday, CW28 2019	2693	98	518	SLIN_SEC	10/07/19 14:46
Local Central Result: Wednesday, CW28 2019	4	0	0	SLIN_SEC	10/07/19 14:39
Remote Central Result: Wednesday, CW28 2019	42	2	15	SLIN_SEC	10/07/19 14:01
Yesterday (0)					
Older (5)					
Results of Local Check Runs [created by: *, contact person: *]					
Today (169)					
Yesterday (270)					
Older (1439)					

Displaying central check results from the reference check system

**i** For more information, see [Browsing ATC Results](#).

## Supported Test Scenarios in ABAP SQL Test Double Framework

OSQL test double framework now supports creation of test doubles for the CDS projection views.

You can now pass the CDS projection view name as part of the `i_dependency_list` parameter

**i** For more information, see [ABAP SQL Test Double Framework \[page 539\]](#)

## ABAP Test Double Framework

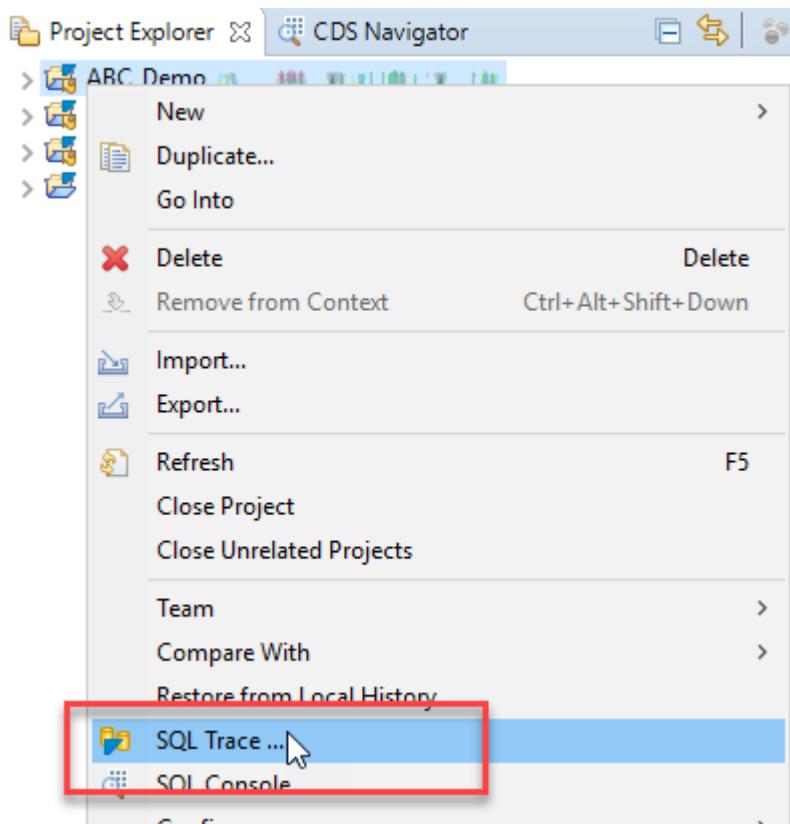
You can now configure a call to a static method in an ABAP interface using ABAP Test Double Framework.

## Using Troubleshooting Tools

### Analyzing SQL Statements Using the SQL Trace and the Technical Monitoring Cockpit

You can now create an SQL Trace and display it in the *Technical Monitoring Cockpit*. This enables you to analyze the performance of SQL statements.

To open the *SQL Trace* dialog, open the context menu of an ABAP project and choose *SQL Trace*....



Context menu from an ABAP project to open the SQL trace

From the dialog that will then be opened, you can

- start and stop the SQL trace and
- navigate to the *Technical Monitoring Cockpit* in order to analyze the SQL trace results.

**i** For more information, see [Using the SQL Trace and the Technical Monitoring Cockpit \[page 693\]](#)

## Cloud icon Defining Restriction Fields and Restriction Types

You can now expose and maintain authorization fields and authorization objects in a business role. To do so, you have to define corresponding restriction fields and restriction types and assign the restriction fields to the required restriction types.

For more information, see [Defining Restriction Fields \[page 496\]](#) and [Defining Restriction Types \[page 497\]](#).

## Cloud icon Working with the HTTP Service Editor

### Defining a Custom Handler Class Name

When using the HTTP service wizard, you can now define a custom handler class name for your HTTP service.

**i** For more information, see [Working with the HTTP Service Editor](#).

## Cloud Adaptation of ABAP Source Code for SAP Cloud Platform

To adapt your custom code in the context of SAP Cloud Platform using the ABAP Test Cockpit (ATC), you can now apply specific quick fixes for ATC findings.

To perform the SAP Cloud Platform readiness checks which provide these quick fixes, you can use, for example, the ATC check variant SAP\_CLOUD\_PLATFORM\_READINESS.

Recommended Quick Fixes

Select Quick Fixes and Post Processing Options

19 of 25 selected findings offer recommended quick fixes. Confirm the pre-selected recommended quick fixes or select an alternative quick fix from the dropdown list in the Quick Fix column.

Quick Fix Selection

Finding	Quick Fix
READ - BINARY SEARCH for result of statement at ... line ... (1 Errors)	Insert a SORT statement after the SELECT statement
ZCL_FLIGHT_EVALUATION / GET_EVALUATIONS_BY_FLIGHT_DATA (Method Implementation)	Insert a SORT statement after the SELECT statement
Syntax error in restricted language scope (8 Errors)	Replace MOVE with an assignment.
ZCL_FLIGHT_EVALUATION / ZIF_FLIGHT_EVALUATION-SET_MEAL_RATING (Method Implementation)	Replace MOVE with an assignment.
ZCL_FLIGHT_EVALUATION (Local Types) / line 26	Replace 'REFRESH' with CLEAR.
ZCL_FLIGHT_EVALUATION (Local Types) / line 101	Replace MOVE with an assignment.
ZCL_FLIGHT_EVALUATION (Local Types) / line 91	Replace MOVE with an assignment.
ZCL_FLIGHT_EVALUATION (Local Types) / line 32	Replace 'DESCRIBE TABLE' with the built-in function LINES( ).
ZCL_FLIGHT_EVALUATION (Local Types) / line 86	Replace MOVE with an assignment.
ZCL_FLIGHT_EVALUATION (Local Types) / line 27	Replace 'FREE' with CLEAR.
ZCL_FLIGHT_EVALUATION (Local Types) / line 96	Replace MOVE with an assignment.
Syntax warning in restricted language scope (4 Warnings)	Replace 'DIVIDE' with a calculation assignment.
ZCL_FLIGHT_EVALUATION (Local Types) / line 102	Replace 'DIVIDE' with a calculation assignment.
ZCL_FLIGHT_EVALUATION (Local Types) / line 87	Replace 'ADD' with a calculation assignment.
ZCL_FLIGHT_EVALUATION (Local Types) / line 92	Replace 'SUBTRACT' with a calculation assignment.
ZCL_FLIGHT_EVALUATION (Local Types) / line 97	Replace 'MULTIPLY' with a calculation assignment.
Usage of deprecated Basis API (6 Warnings)	Replace deprecated object DTEL_BOOLE_D with released object DTEL_ABAP_BOOLEAN
ZIF_FLIGHT_EVALUATION (Interface) / line 27	Replace deprecated object DTEL_BOOLE_D with released object DTEL_ABAP_BOOLEAN
ZCL_FLIGHT_EVALUATION (Test Classes) / line 15	Replace deprecated object CLAS_CL_AUNIT_ASSERT with released object CLAS_CL_ABAP_UNIT_ASSERT
ZCL_FLIGHT_EVALUATION / CREATE_FLIGHT_EVALUATION (Method Implementation) / line 5	Replace deprecated object DTEL_BOOLE_D with released object DTEL_ABAP_BOOLEAN
ZCL_FLIGHT_EVALUATION / ZCL_FLIGHT_EVALUATION (Private Section) / line 11	Replace deprecated object DTEL_BOOLE_D with released object DTEL_ABAP_BOOLEAN
ZCL_FLIGHT_EVALUATION / GET_SYSTEM_INFO (Method Implementation) / line 2	Replace deprecated object DTEL_UNAME with released object DTEL_SYNAME
ZCL_FLIGHT_EVALUATION / GET_SYSTEM_INFO (Method Implementation) / line 6	Replace deprecated object DTEL_LANGU with released object DTEL_SPRAS

Select All | Deselect All | Group by Object

Post Processing

Activate changed objects

Recheck

?

< Back | Next > | Finish | Cancel

Dialog that lists the possible quick fixes

Preview of the custom code adaptations that will be performed by the ATC quick fixes

**i** For more information, see [Applying Quick Fixes for ATC Findings \[page 591\]](#)

## Installing ADT on the Open Eclipse 2019-06 (4.12) Platform

You can now install the ADT client also on the latest Eclipse 2019-06 (4.12) platform.

This ADT client version supports the following Eclipse platforms:

- 2019-06 (4.12)
- 2019-03 (4.11)

**i** For more information, see

- [Installing ABAP Development Tools](#)
- [Eclipse 4.12 - New and Noteworthy](#)

## 8.7 Version 3.2

Get an overview of the most significant changes in ABAP core development that relate to the following:

- Client: ABAP Development Tools (ADT) 3.2

- Back end version:  SAP Cloud Platform ABAP Environment **1905**.

### Note

All the 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:

-  [Working with Released APIs \[page 838\]](#)
-  [Working with Transport Organizer \[page 838\]](#)
-  [ABAP CDS Test Double Framework \[page 840\]](#)
-  [Working with Authorization Objects and Fields \[page 840\]](#)
-  [Working with Development Objects \[page 840\]](#)
-  [Using Troubleshooting Tools \[page 841\]](#)
-  [Eclipse IDE Package \[page 842\]](#)

## **Working with Released APIs**

### **Deprecating Development Objects**

You can now mark a development object as deprecated, for example if you have defined an improved successor that is to be used instead.

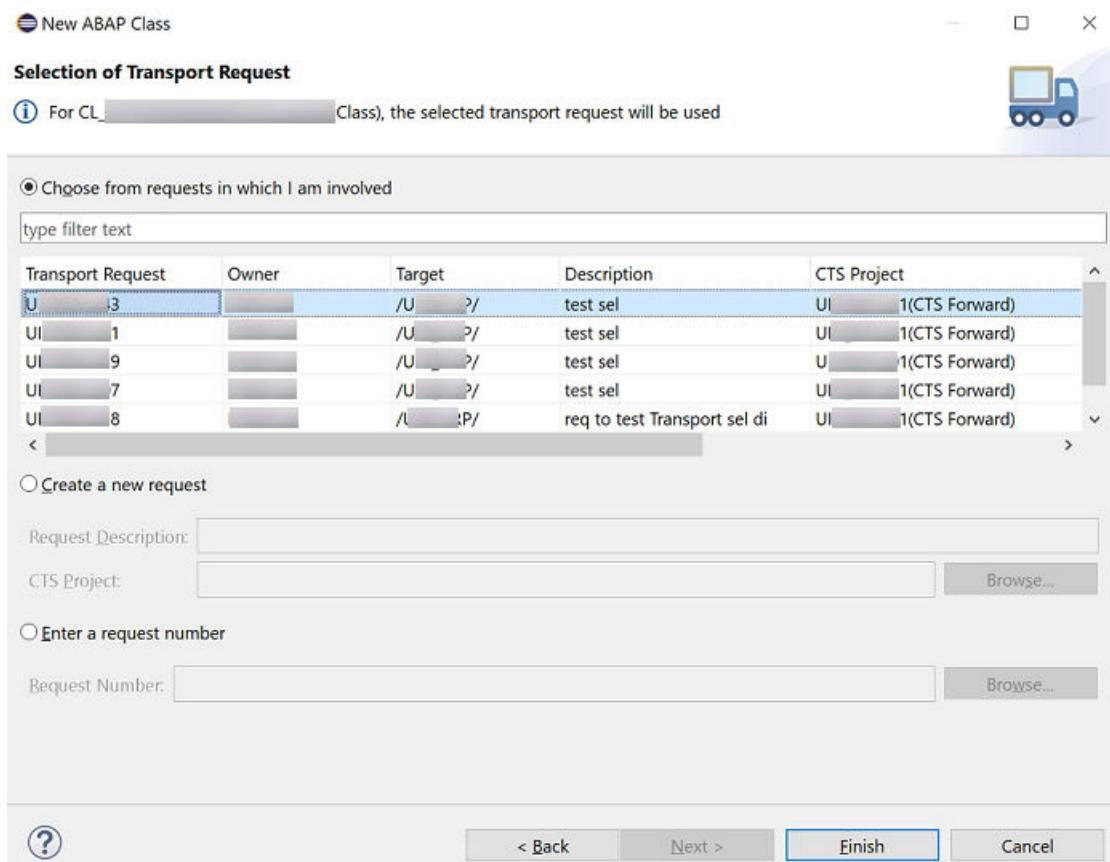
 For more information, see

- [Deprecation \[page 103\]](#)
- [Deprecating Development Objects \[page 466\]](#)

## **Working with Transport Organizer**

### **Support for Display of CTS Projects**

The *Transport Organizer* view and the dialog now support the display of CTS projects. Selection of Transport Request



**i** For more information, see

- Selection Transport Organizer [page 104]
  - Transport Request [page 108]

## ABAP CDS Test Double Framework

### Supported Test Scenarios

CDS Test Double framework now also supports creation of test doubles for the following dependent-on components:

- Shared tables (For all share types like R, W, T, and S)
- Session Variables

**i** For more information, see [ABAP CDS Test Double Framework \[page 516\]](#)

## Working with Authorization Objects and Fields

### Defining Authorization Objects and Fields

You can now define authorization fields and bundle them in authorization objects to enable authorization checks and to protect certain activities in your business processes.

When using the authorization objects in your implementation, you can define default authorizations on service level.

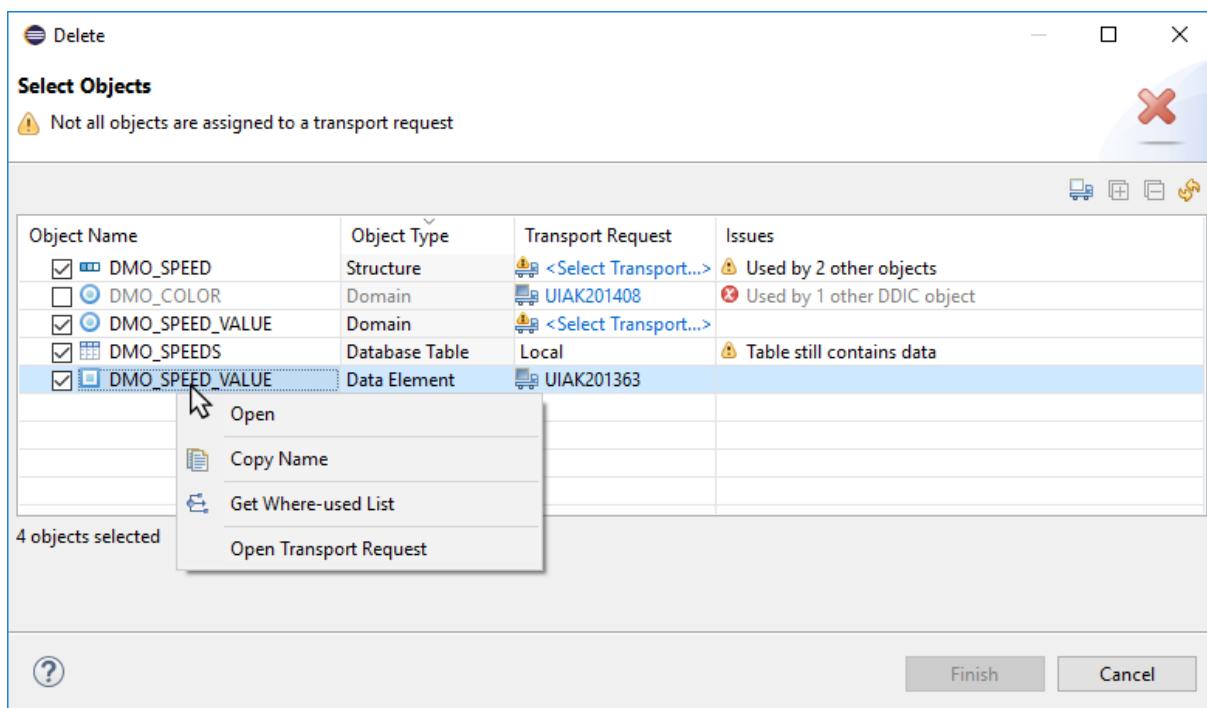
You can adjust these values if required when you create an app based on these services.

**i** For more information, see [Working with Authorization Objects and Fields \[page 493\]](#).

## Working with Development Objects

### Deleting Development Objects

You can now delete multiple development objects simultaneously after selecting them in the Project Explorer. This feature allows you to delete two data definitions referencing each other. The deletion dialog opens and shows if an individual object can be deleted or not. Details and number of usages are displayed in the *Issues* column. From the context menu on an individual object, you can get a where-used list.



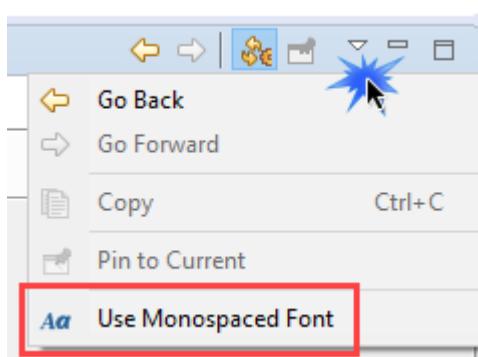
**i** For more information, see [Deleting Development Objects \[page 183\]](#).

## Using Troubleshooting Tools

### Using Monospaced Fonts to Analyze Internal Tables

You can now display the content of an internal table in monospaced fonts for a precise string analysis.

To do so, choose [Use Monospaced Font](#) from the toolbar menu in the *ABAP Internal Table (Debugger)* view.



Opening the toolbar menu to display the content in monospaced font

**i** For more information, see [Analyzing Internal Tables \[page 625\]](#).

## Eclipse IDE Package\*

### Supported Eclipse Platforms

You can now install the ADT client on the latest Eclipse 2019-03 (4.11) platform.

This ADT client version supports the following Eclipse platforms:

- 2018-09 (4.9)
- 2018-12 (4.10)
- 2019-03 (4.11)

#### i Note

Contrary to SAP's support and maintenance strategy, this ADT client version currently supports three Eclipse platform versions temporary.

i For more information, see

- [Installing ABAP Development Tools](#)
- [Eclipse 4.11 - New and Noteworthy](#)
- Support Strategy of ABAP Development Tools (SAP Note [1856565](#))

### Preparation of a Distributable Eclipse IDE Package

The ADT Installation Guide now contains information on how to prepare an Eclipse IDE package, including ABAP Development Tools, that can be used for central distribution. This includes a description on how to automate the preparation.

i For more information, see [Installing ABAP Development Tools](#) (6.3 Recommendations for the System Administrator).

## 8.8 Version 3.0

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

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

#### 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 will find the release information about the following areas:

-  [Working with Classic Objects in ABAP Dictionary \[page 843\]](#)
-  [Editing ABAP Source Code \[page 844\]](#)
- [Ensuring Quality of ABAP Code \[page 844\]](#)

- [Working with the Transport Organizer \[page 845\]](#)
- [Using Troubleshooting Tools \[page 847\]](#)
- [Supported Eclipse Platforms \[page 847\]](#)

## Working with Classic Objects in ABAP Dictionary

### Working with Table Types

You can now edit table types using the table type editor.

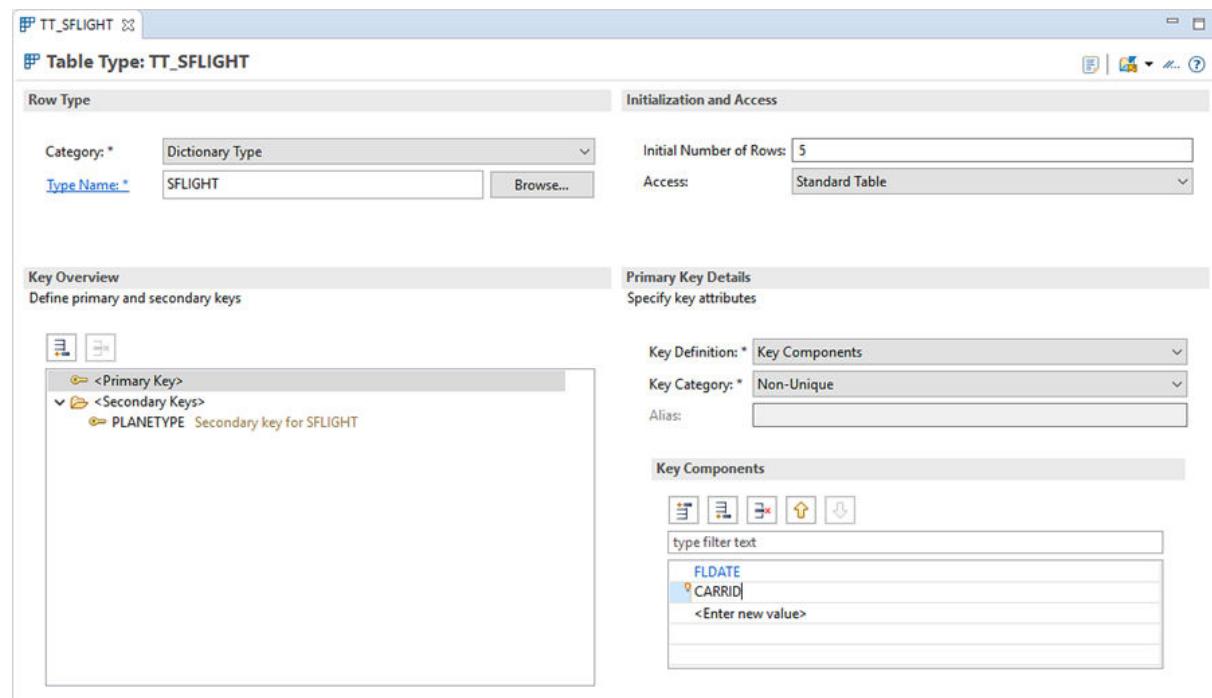


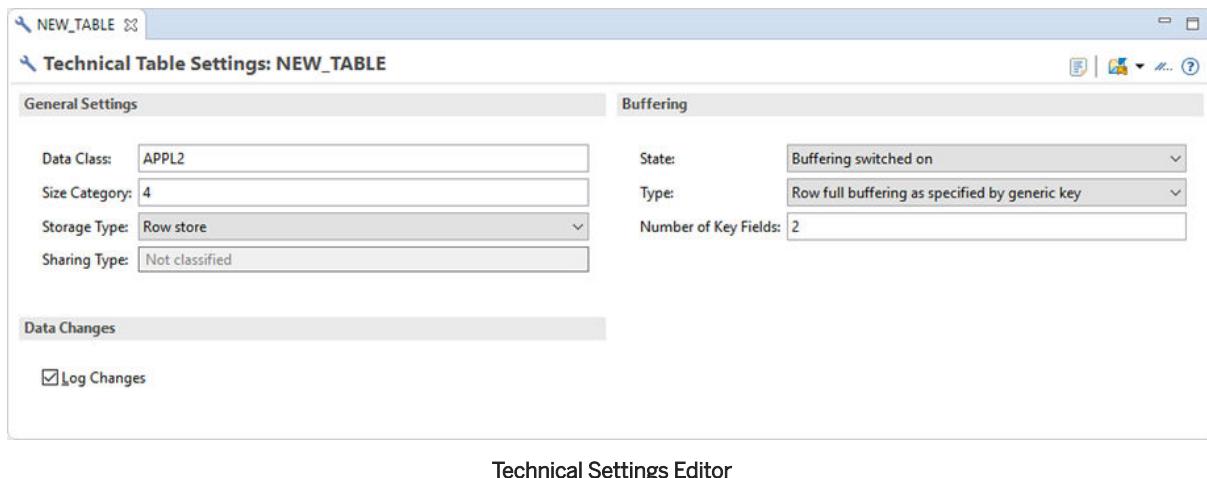
Table Type Editor

For more information, see

- [Editing Table Types \[page 238\]](#)
- [Released Object Types \[page 738\]](#)

### Editing Technical Table Settings

You can now edit technical settings of the database table using the technical settings editor.



Technical Settings Editor

**i** For more information, see [Editing Technical Table Settings \[page 235\]](#)

## Cloud icon Editing ABAP Source Code

### Using Code Completion

Code completion now only suggests released objects and your own customer objects. Objects that cannot be used are hidden. This improves efficiency when inserting objects to your source code.

#### **i** Note

This feature is not supported in the integrated source-based editors of ABAP CDS objects and ABAP Dictionary objects.

**i** For more information, see

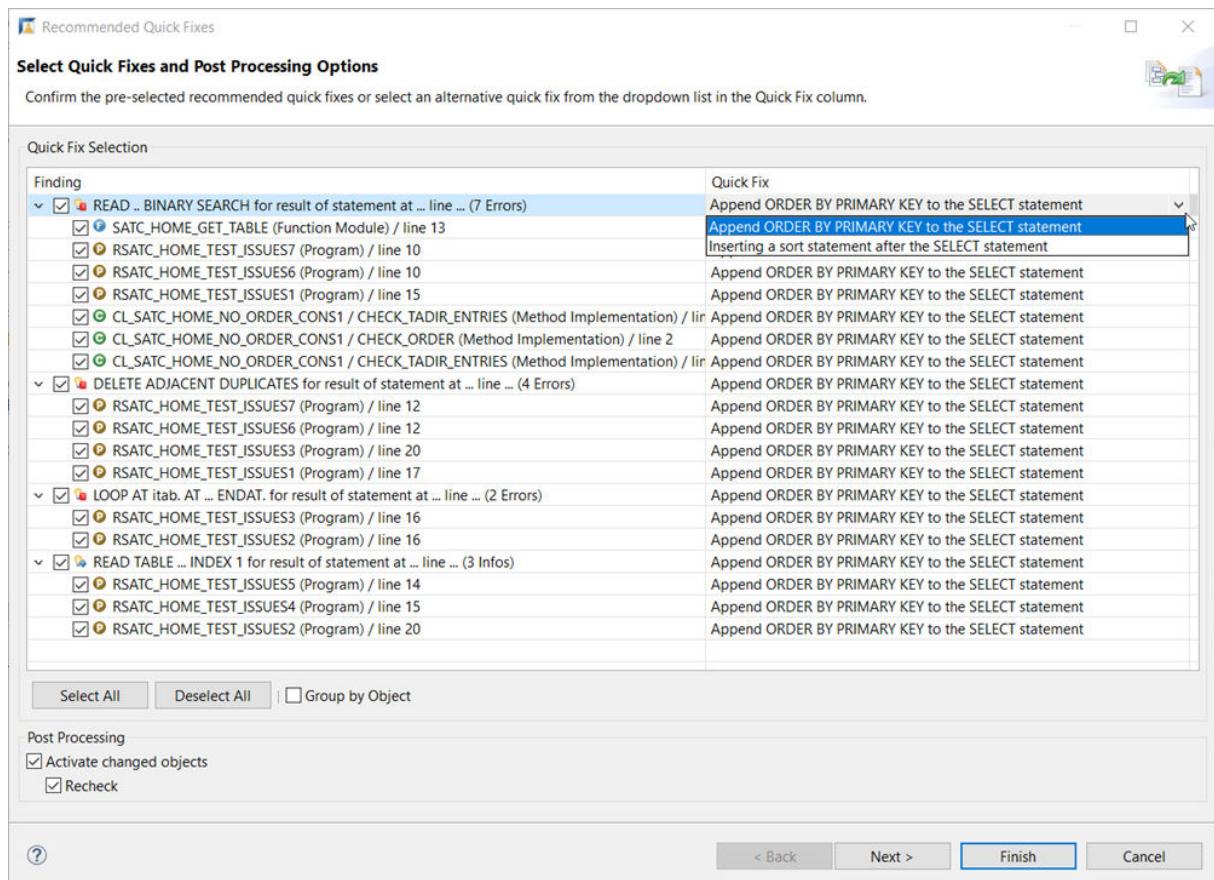
- [Code Completion \[page 298\]](#)
- [Released Object Types \[page 738\]](#)

## Cloud icon Ensuring Quality of ABAP Code

### Recommended Quick Fixes Wizard for ATC Findings

You can now fix multiple ATC findings at once with the *Recommended Quick Fixes* wizard. This wizard suggests recommended Quick Fixes for all selected ATC findings.

Quick Fixes provide functions that enable you to resolve errors and warnings without adapting your source code manually.



Apply Recommended Quick Fixes for ATC Findings in the Recommended Quick Fixes Wizard

**i** For more information, see: [Applying Recommended Quick Fixes for Multiple ATC Findings \[page 594\]](#)

## Working with Transport Organizer

### Editing the Attributes of a Transport Request

You can now modify the attributes of a transport request.

**i** For more information, see:

## View the Transport Logs of a Transport Request

You can now view the logs of a transport request that is in *Released* status.

The screenshot shows the SAP Workbench interface. The top navigation bar has 'Request: (Workbench)' and a search bar. The left sidebar shows a tree view of logs, with a callout pointing to the 'Ctrl+Click to view the logs' option for the 'Checks at Operating System Level' node. The main area displays a table of logs with columns for date, time, status, and severity. The bottom part shows the ABAP Log viewer with a message about buffer writing and a table of log entries.

Date	Time	Status
04.02.2019	10:07:19	(0) Completed
04.02.2019	10:07:22	(0) Completed
04.02.2019	10:07:28	(0) Completed
	10:07:26	(0) Completed
	10:07:27	(0) Completed
04.02.2019	10:07:27	(0) Completed
04.02.2019	10:07:27	(0) Completed
05.02.2019	07:31:04	(4) Ended with Warning
05.02.2019	07:52:41	(0) Completed

Message	Severity
##### Checking whether buffer can be written to	information
Transport request :	information
System :	information
tp path : tp	information
Version and release: 381.138.02 776	information
Checking buffer of target system: /usr/sap/trans73/buffer/7IT	information
Buffer /usr/sap/trans73/buffer/7IT can be written to	information
Checking buffer of target system: /usr/sap/trans73/buffer/8NW	information
Buffer /usr/sap/trans73/buffer/8NW can be written to	information

**i** For more information, see:

## Using Troubleshooting Tools

### Profiling ABAP Code

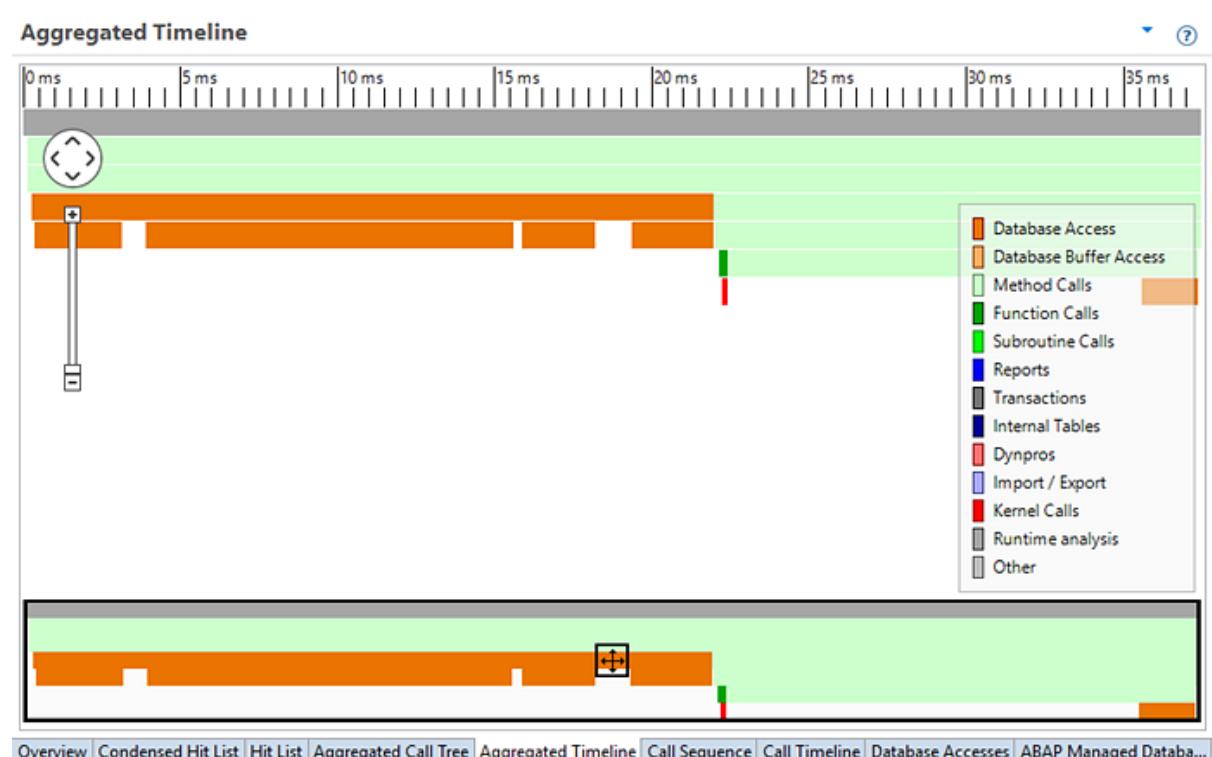
When creating a trace request, you can now set the *URI Pattern* filter. This enables you to focus tracing for specific HTTP requests.

For more information, see [Filling Out the ABAP Trace Request Dialog \[page 647\]](#)

### Analyzing ABAP Traces with Call Stack Aggregation

You can now display the data of the aggregated call tree as a diagram in the *Aggregated Timeline* tool.

This enables you to easily identify eye-catching patterns.



Aggregated trace events in the Aggregated Timeline tool

For more information, see [Analyzing Aggregated Call Trees Using the Aggregated Timeline \[page 668\]](#)

### Supported Eclipse Platforms

You can now install the ADT client also on the latest Eclipse 2018-12 (4.10) platform.

This ADT client version supports the following Eclipse platforms:

- 4.8 (Photon)
- 2018-09 (4.9)
- 2018-12 (4.10)

### i Note

Contrary to SAP's support and maintenance strategy, this ADT client versions supports 3 Eclipse platform versions temporary.

As of ADT 3.0, on Microsoft Windows<sup>®</sup> operating systems only the 64-bit Eclipse version is supported.

i For more information, see

- [Installing ABAP Development Tools](#)
- [Eclipse 4.10 - New and Noteworthy](#)
- [Support Strategy of ABAP Development Tools \(SAP Note 1856565\)](#)

## 8.9 Version 2.102

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

- Client: ABAP Development Tools (ADT) **2.102**
- Back end version:  SAP Cloud Platform ABAP Environment **1811**.

### 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 will find the release information about the following areas:

-  [Working with Classic Objects in ABAP Dictionary \[page 848\]](#)
- [Working with Transport Organizer \[page 848\]](#)
- [Installing ADT on the Open Eclipse 4.8 \(Photon\) Platform \[page 849\]](#)

## Working with Classic Objects in ABAP Dictionary

### Working with Table Types

You can now use table types.

i For more information, see [Released Object Types \[page 738\]](#)

## Working with Transport Organizer

### Merge Requests

You can now move all the objects and tasks under one request to another request.

For more information, see [Merge Requests \[page 460\]](#)

### **Including Objects in a request Manually**

You can now include objects of your choice to a request or assign objects from one request to another request.

For more information, see [Including Objects in a Request Manually \[page 456\]](#)

## **Installing ADT on the Open Eclipse 4.8 (Photon) Platform**

You can now install the ADT client also on the latest Eclipse 4.8 (Photon) platform.

This ADT client version supports the following Eclipse platforms:

- 4.7 (Oxygen)
- 4.8 (Photon)

**i** For more information, see

- [Installing ABAP Development Tools](#)
- [Eclipse Photon \(pre-release milestones\) - New and Noteworthy](#) 

# 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.



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.