HANDBOOK

ABAP Development Tools (ADT) in Eclipse

As of March 2023

Table of Contents

Table of Contents **2**
Table of Figures **6**
Index of Tables **12**
Introduction **13**
Why a DSAG guide on ABAP Development Tools (ADT) in Eclipse? 13
Motivation for and topicality of the guide 13
1 Introduction Eclipse and ADT **15**
1.1 Eclipse … Why? 15
1.2 A brief look at the history of its creation 15
1.3 The concept of extensions 15
1.4 Difference between perspective and view 16
1.5 The power of keyboard shortcuts 16
1.6 Which Eclipse feature is available in which version? 16
2 Motivation for ADT **18**
2.1 You want to apply and implement modern ABAP. 18
2.2 You want to use one development environment for everything. 18
2.2.1 Technological aspect 18
2.2.2 Development on multiple system lines 19
2.3 You want to break free from the technical limitations of the SAP GUI. 19
2.3.1 Six sessions in one system are simply not enough 19
2.3.2 You don't want to lose code due to network interruptions. 20
2.4 You like to improve your ABAP code through refactoring. 20
2.5 You like to adapt the look and feel of your development environment to your wishes. 21
2.6 Would you like even more tool-based support? 21
2.7 You already know Eclipse from other programming languages? Great! 22
2.8 You are interested in new things. 22
2.9 The benefits for the organization 23
2.10 You are missing the form-based editor 23
2.11 Why ABAP Development Tools 24
3 Working with ADT **26**
3.1 Introduction: Basics of working with ADT 26
3.1.1 Getting Started with ABAP Development Tools 26
3.1.2 The shift from form-based to text-oriented code creation 27

3.1.3 Context of the exercise 27
3.1.4 Connecting the Development System – New Project 28
3.1.5 Creating a class in text mode 30
3.1.6 Definition of a method in the class 33
3.1.7 Automatic completion and formatting of the code 35
3.1.8 Implementation of the method using Quick Fix 36
3.1.9 Renaming of Parameters – Refactoring 39
3.2 Features of ADT 43
3.2.1 Overarching features 43
3.2.2 Search and Navigation 51
3.2.3 ABAP Editor 55
3.2.4 Other object types 61
3.2.5 ABAP Views 62
3.2.6 Refactoring Code with ADT 88
3.2.7 Versioning and Comparison 90
3.2.8 Documentation with ABAP Doc 92
3.2.9 Executing Source Code 94
3.2.10 Data Preview 97
3.2.11 Core Data Services 100
4 Troubleshooting Tools in Eclipse 107
4.1 Troubleshooting with the ABAP Development Tools 107
4.2 The Debugger in ABAP Development Tools 108
4.2.1 Breakpoints and Soft-Breakpoints 108
4.2.2 Debugging Perspective 108
4.2.3 Special Behavior in the Debugger 109
4.2.4 Additional Information 109
4.3 Checkpoint IDs and dynamic logpoints 110
4.4 Performance Analysis 113
4.4.1 Note on HANA Studio and SAP HANA Tools 123
4.5 Feed Reader 123
4.6 Documentation Links 126
5 Installation, Distribution und Update Strategies 127
5.1 Delimitations 127
5.1.1 Installation Guide of SAP 127

5.2 Preparations 127
5.2.1 Java Development Kit and Java Runtime Environment 127
5.2.2 Backend 128
5.2.3 SAP GUI Installation 129
5.2.4 Visual Studio Redistributable 129
5.3 Technical Structure of an Eclipse Installation 129
5.4 Plug-ins 130
5.4.1 Eclipse Marketplace 130
5.4.2 Update Site 132
5.5 Installation and Distribution Strategies 134
5.5.1 Overview and Comparison 134
5.5.2 Manual Installation 136
5.5.3 Pre-configured Initial Installation 136
5.5.4 Eclipse Installer 137
5.6 Error Situations 154
5.6.1 Errors during update or upgrade 154
5.6.2 Single-sign-on Libraries 155
5.6.3 “No repository found containing” 155
5.6.4 PKIX - Certificate Error 155
5.6.5 macOS aarch64 support & SAP GUI for Java 156
5.6.6 Offline-Installation – Download ADT-Dependencies 156
6 Best Practices Eclipse Configuration 157
6.1 Settings in Eclipse 157
6.1.1 Globale Settings 157
6.1.2 Project-specific Settings 163
6.2 Views and Perspectives 165
6.2.1 Views 165
6.2.2 Perspectives 168
6.3 Recommended additional views 170
6.4 Suggestions for distribution 171
7 Plug-ins 172
7.1 Introduction 172
7.2 Useful Open-Source Plug-ins 172
7.2.1 Open Editors 172

7.2.2 AnyEdit Tools ... 173
7.2.3 PDE Tools ... 175
7.3 Useful Open-Source ADT Plug-ins ... 176
7.3.1 ABAP Favorites ... 176
7.3.2 ABAP Continuous Integration ... 178
7.3.3 ABAP ADT Extensions ... 179
7.3.4 ADT Classic Outline ... 182
7.3.5 ABAP Quick Fix ... 183
7.3.6 ABAPQuickFixS4Conversion ... 184
7.3.7 ABAP Tags ... 186
7.3.8 ABAP Search and Analysis Tools ... 188
7.3.9 ABAP Code Search ... 189
7.3.10 abapGit Eclipse Plug-in ... 191
7.4 Develop own ADT Plug-ins ... 191
7.4.1 Prerequisites ... 191
7.4.2 Setting up the development environment ... 192
7.4.3 Key Concepts/Artifacts ... 194
7.4.4 Creation of a Plug-in Project ... 194
7.4.5 Creation of a Feature Project ... 198
7.4.6 Creation of an Update Site ... 200
7.4.7 Extension of the ADT Backends with ABAP Code ... 203
7.4.8 Java Code Snippets for recurring tasks in ADT ... 205
7.4.9 Project Set-up with Maven ... 209
Authors ... 210
Imprint ... 214

Table of Figures

Figure 1 Creating an ABAP Project in Eclipse 28
Figure 2 The Project Explorer 29
Figure 3 Adding Packages to Favorites 30
Figure 4 Creating a New ABAP Class in Project Explorer 31
Figure 5 Properties Dialog: Creating an ABAP Class 31
Figure 6 Transport Request Dialog 32
Figure 7 Display of the new class in ADT 33
Figure 8 Editing the Class 34
Figure 9 Example Code Completion for the Import Parameter 35
Figure 10 Using the Quick Fix for Method Implementation 37
Figure 11 Component Selection Using Code Completion 38
Figure 12 Renaming Method Parameters 40
Figure 13 Query Dialog for saving the Code 40
Figure 14 Entering a new Parameter Name 41
Figure 15 Transport Selection and Options 41
Figure 16 Renaming Preview 42
Figure 17 Query of the Workspace Directory 43
Figure 18 Switching the Workspace 45
Figure 19 The Workspace Dialog 46
Figure 20 Adding Packages to Favorites 47
Figure 21 Project Explorer Detail Screen with Button Bar 48
Figure 22 Working Sets Settings 49
Figure 23 Creation and Processing of the Working Sets 49
Figure 24 Assignment of Project to Working Set 50
Figure 25 Project Explorer Display Setting 50
Figure 26 Illustration of Projects in Working Sets 51
Figure 27 Object Search Dialog 52
Figure 28 Results of the Search 53
Figure 29 Display of additional Search Options 53
Figure 30 Result with Object and Type Filter 53
Figure 31 Navigations Icons 54

Figure 32 Actions for the Project Explorer 55
Figure 33 ABAP Editor - Main Window 56
Figure 34 Element Info for a Method 56
Figure 35 Element Info for a Data Element 57
Figure 36 Display of the Element Info after Selecting the Object 58
Figure 37 Formatting Context Menu 59
Figure 38 Settings for the ABAP Formatter 60
Figure 39 Display of Refactoring Options 61
Figure 40 Comparison of Function Groups in the Project Explorer of ADT and in the SE80 61
Figure 41 Display of Properties in the Outlines 62
Figure 42 Displaying the Messages in the Problems View 63
Figure 43 Display of the Options of the View 63
Figure 44 Configuration of the Displayed Points 64
Figure 45 Properties View 65
Figure 46 History of Transports 65
Figure 47 Transport View 66
Figure 48 Template View Browser 67
Figure 49 Selection of Templates in Content Assist 68
Figure 50 Result of the Proof of Use/Where-used List 69
Figure 51 Filter for Where-Used-Search 70
Figure 52 Menu for Creating the Bookmark 71
Figure 53 Entering a Name (Bookmark) 71
Figure 54 Display of a Bookmark in the Source Code 72
Figure 55 Bookmarks View 72
Figure 56 Sharing the Source Code as a Link (Context Menu) 73
Figure 57 Link Sharing Dialog 74
Figure 58 Opening the ABAP Type Hierarchy 75
Figure 59 Display of the Type Hierarchy in the View 75
Figure 60 Transport Organizer View 76
Figure 61 Representation of a Run-Time Error 76
Figure 62 Example of a System Message 77
Figure 63 Executing the ABAP Unit Test via the Context Menu 77

Figure 64 Display of the Results of the ABAP Unit Test 78
Figure 65 Restart the Execution 78
Figure 66 Dialog for Setting the Execution of ABAP Unit Tests 79
Figure 67 Carrying out the Coverage Measurement 79
Figure 68 Color Highlighting of Source Code after Unit Test 80
Figure 69 Display of the Results of the ATC Run 81
Figure 70 Requesting Exemptions via the ATC View 81
Figure 71 Dialog for Classifying the Exception 82
Figure 72 Calling the ABAP Language Help from the Context Menu 83
Figure 73 ABAP Language Help View 84
Figure 74 Button Bar of the View 84
Figure 75 Navigation to Help Content 85
Figure 76 Overview of Available Help and Documentation 86
Figure 77 Search in Help 87
Figure 78 Further Help and Documentation 87
Figure 79 Display of Navigation 88
Figure 80 Context Menu for Comparing Versions 90
Figure 81 Comparison View – Comparison of two Versions 92
Figure 82 Context Menu for Completely Transferring a Version from Local Version Management 92
Figure 83 ABAP Doc Documentation of the Method 93
Figure 84 Executing a class in SAP GUI 95
Figure 85 Result of Execution 95
Figure 86 Selection of the Project 96
Figure 87 Output to the Console 97
Figure 88 Start the Data Preview from the Table 98
Figure 89 Display of the Data Preview 98
Figure 90 Navigation via Associations 99
Figure 91 SQL Console 99
Figure 92 Object Types of the CDS in Navigation 100
Figure 93 Different Files Define the Properties of a CDS View Entity 101
Figure 94 Overview of a CDS View Entity Using Element Info 102
Figure 95 Calling the Dependency Analyzer via the Context Menu 103

Figure 96 SQL Dependency Tree 103
Figure 97 SQL Dependency Graph 104
Figure 98 Complexity Metrics 105
Figure 99 Active Annotations of a View 106
Figure 100 Debugging Perspective in Eclipse 108
Figure 101 Values of the Variables in the Debugging Perspective 109
Figure 102 Creation of a Log Point via the Context Menu 111
Figure 103 Attributes when Creating a Log Point 112
Figure 104 Log Points View in the Debugging Perspective 113
Figure 105 ABAP Trace Requests in the Debugging Perspective 114
Figure 106 Context Menu of a Trace 114
Figure 107 Overview of the Properties of a Trace 115
Figure 108 Aggregated Overview of a Trace History 115
Figure 109 Jump to the SQL Trace 116
Figure 110 Getting started with managing User Parameters 117
Figure 111 Setting the User Parameter HDB_OPEN_STUDIO 117
Figure 112 Selection of eclipse.exe as a New Way to Display *.plv Files 118
Figure 113 Selection of a Specific Selection 118
Figure 114 Download the *.plv File 119
Figure 115 Query Execution Plan Display 119
Figure 116 Creation of ABAP Cross Traces 121
Figure 117 Detailed View of Operations 122
Figure 118 Subscribing to Popular RSS Feeds 124
Figure 119 Multiple Runtime Errors within a Feed 124
Figure 120 View of an SAP Gateway Error from the Error Log 125
Figure 121 Getting Started with the Eclipse Marketplace 131
Figure 122 Sample Search for Plug-ins in the Eclipse Marketplace 132
Figure 123 Installation of New Software via the Context Menu 133
Figure 124 Entering the Update Site 134
Figure 125 Switching between Indices 140
Figure 126 Adding and Setting Properties 141
Figure 127 Components of a "minimal" ADT Installation 142
Figure 128 Components of a Minimal ADT Installation 144

Figure 129 Components of the SAP Update Site 146
Figure 130 Ability to View the Settings that have Already Been Saved 147
Figure 131 Settings that Have Already Been Saved 147
Figure 132 Switch to "Advanced Mode" 149
Figure 133 Switching between Indices 150
Figure 134 Selection of the Product Version 151
Figure 135 Selection of the Stream 152
Figure 136 Querying other Variables 153
Figure 137 Getting Started with Global Settings 157
Figure 138 Setting for the Dark Theme 158
Figure 139 Source Code Indentation Setting 158
Figure 140 Example of Setting 159
Figure 141 Result Image in Source Code 159
Figure 142 Color Settings to Highlight the Keywords in the Source Code 161
Figure 143 Administration of ABAP Templates in Settings 162
Figure 144 Inserting the Template into the Source Code 162
Figure 145 Getting Started with Project-specific Settings 164
Figure 146 Possible Settings for the Pretty Printer / ABAP Formatter 165
Figure 147 Move the View via the Label/Tab 165
Figure 148 The Markings Indicate the Placeability of the Window 166
Figure 149 Display of Stacked Views 166
Figure 150 Minimizing View Groups 166
Figure 151 Restoring the View Groups 167
Figure 152 Closing a View 167
Figure 153 Displaying a View 168
Figure 154 Switching Between Different Perspectives 168
Figure 155 Reset a Perspective 169
Figure 156 Save a Perspective 169
Figure 157 Naming the new Perspective 170
Figure 158 New Perspective with Name 170
Figure 159 Possible Setting of the ABAP Perspective 171
Figure 160 Possible Setting of the Debugger Perspective 171
Figure 161 Dialog for Adjusting the Sort Order of Open Editors 173

Figure 162 Examples of Available Operations in the Context Menu 174
Figure 163 Clipboard History (Keyboard Shortcut Ctrl+Shift+V) 175
Figure 164 ABAP Favorites View 177
Figure 165 Context Menu of a Folder in ABAP Favorites View 178
Figure 166 Colored Highlighting of the Status Bar per Project + Test Status 179
Figure 167 Management of Packages for which Unit Tests and/or ATC Test Runs are Scheduled 179
Figure 168 View for Managing the Stored Access Aata of ABAP Systems 180
Figure 169 Context Menu on Project in Project Explorer 181
Figure 170 Status Bar in the Eclipse Window 181
Figure 171 Classic Outline View 182
Figure 172 ABAP Code Before Quick-Fix Execution 183
Figure 173 ABAP Code After Quick-Fix Execution 183
Figure 174 Example of Quick Fix Availability on a SELECT Statement 185
Figure 175 SELECT Statement after Applying the Quick Fix 185
Figure 176 View “Tag Manager” 186
Figure 177 Search-Dialog with “ABAP Object Search” 187
Figure 178 Search Dialog on Page "ABAP Object Search" 188
Figure 179 View “CDS Analyzer” - Top-Down-Analysis 189
Figure 180 Search-Dialog with “ABAP Code Search” 190
Figure 181 Eclipse Bundle “Eclipse IDE for RCP and RAP Developers” 192
Figure 182 Plug-in Project Wizard – Entrance 195
Figure 183 Plug-in Project Wizard – Content 196
Figure 184 Plug-in-Projekt in the Project Explorer View 197
Figure 185 Feature Project Wizard - Entrance 198
Figure 186 Feature Project Wizard - Plug-in Selection 199
Figure 187 Update Site Wizard 201
Figure 188 Compiler Settings in the Eclipse Settings Dialog 202
Figure 189 Dialog for Adding an Update Site 202
Figure 190 GitHub Repository Settings for GitHub Pages 203
Figure 191 Example of a Simple Transformation for the Transformation of ABAP <-> XML 204
Figure 192 Sample EMF Model for Serializing XML <-> Java Object 205

Index of Tables

Table 1 Comparison of Different Installation Options 136
Table 2 Terms in Oomph..... 138
Table 3 Key Elements 142
Table 4 Most Important Elements of a Project 143
Table 5 Features of Eclipse Ini Task 148
Table 6 Authors……...213

Introduction

Why a DSAG guide on ABAP Development Tools (ADT) in Eclipse?

Maybe you're like some of us and you're just getting started with Eclipse? Or you are already a professional and already very familiar with ADT but need basic information and a handout to make this development environment palatable to your colleagues. Or, you belong to the group of regular Eclipse users for a programming language other than ABAP and would like to know what works differently in ABAP than in JAVA, for example.

From a lot of feedback at AK Development events, it quickly became clear that a guideline created by DSAG members in the "tradition" of earlier guidelines (e.g. programming guidelines or ABAP Test Cockpit) would meet with great interest. Fortunately, in response to the AK Development call in February 2022 by Sebastian Freilinger-Huber, many volunteers came forward to create a supporting handbook, so that the work could be started while spring.

A look at the chapter list shows that we have tried to cover as many aspects of development with ADT as possible and that the individual chapters are also aimed at different target groups. Pick the topics that are most interesting to you! In appropriate places, we refer to further sources so as not to overload this guide, but still give you the opportunity to easily access detailed and further information.

We hope that you will find the information which is helpful to you in the guide, regardless of your previous experience with Eclipse and ADT, and hope you enjoy reading the following chapters!

Motivation for and topicality of the guide

Knowledge advantage, influence, and network – these are the three most important aspects from the point of view of the German-speaking SAP User Group e.V. (DSAG).

This guideline was initiated by members of the DSAG Development Working Group and addresses the first aspect: knowledge advantage for users and partners.

In recent years, extensive knowledge and experience has been compiled in numerous DSAG member companies about the framework of ADT with Eclipse. Our goal is to make these findings available to the general public in an appealing form.

Since ADT are continuously developed by SAP (and also via plug-ins from the community), this guide cannot represent a completed state. Rather, it should be understood as a "living document". New findings are constantly integrated, and knowledge already gained is updated. For this reason, in addition to a published

version 1.0 (PDF), a continuously updated Git repository is provided for managing the content. You can use the already established mechanisms of Git (Issues) to place your feedback on the guide. The current version of the guide is continuously available via the website, the PDF version is updated at regular intervals. However, due to the effort to maintain the repository in a continuous way, the feedback mechanism is only supported in German language.

1 Introduction Eclipse and ADT

1.1 Eclipse … Why?

Why is Eclipse the right platform for ADT and not VSCode, NetBeans or any other development environment? It will not be possible to give a fully comprehensive answer here. The fact is, however, that SAP has strategically decided to use Eclipse as the basis for the future ABAP development environment (as the successor to the classic ABAP Workbench) and plans to continue to pursue this approach. This can also be seen in the roadmaps of the ABAP platform and can be read there.

However, it must be mentioned that the connection to Eclipse has created an open interface (between the development environment and the SAP backend) that can be used by other environments. For VSCode, for example, there are already extensions based on these interfaces. However, the completeness and usability are not discussed in the guide. If you need more information about this, you will quickly find suitable results via the usual search engines.

1.2 A brief look at the history of its creation

The origin of Eclipse is at IBM and became independent in 2004 with the Eclipse Foundation. Eclipse is implemented in the Java programming language and is openly available under an open-source license.

Initially, the new Eclipse versions were named after objects from space (including Jupiter's moons). For the sake of simplicity, however, in 2018, the Eclipse Foundation opted for a comprehensible naming convention for the individual versions: < (year)-(month) >. At the same time, the company switched to significantly faster (three-month) release cycles.

1.3 The concept of extensions

The basic installation of Eclipse is already delivered with a basic set of functions and extensions. The scope of these functions and extensions depends largely on which version you select for download on the website. From a technical point of view, the core of the application is always identical. All additional functionalities are delivered by means of extensions – this also applies to the ABAP development tools. This means that you can always add or remove all extensions afterwards from your installation.

Typically, you obtain the extensions via the Eclipse Marketplace, which you can also access directly from the tool.

1.4 Difference between perspective and view

An important feature of Eclipse is its customizability. This will be discussed in detail in the following chapters. It is essential to understand the basic terminology.

A view is an independent part of the tool that is either already included in the basic installation or was added later via the Eclipse extension. This view can be arranged by the user at different locations within the development environment.

A perspective contains a specific layout of Eclipse, i.e., all displayed views and their position. Perspectives thus give you the focus for an activity with all the views you need for it. For details, see Chapter 3 - Working with ADT.

Knowledge of the terminology is essential to be able to work efficiently with Eclipse. Documentation of all kinds uses this terminology to explain functions. This also applies to this ADT guide.

1.5 The power of keyboard shortcuts

A distinctive feature of Eclipse is the use of keyboard shortcuts. In the beginning, you will spend a lot of time trying to find and memorize the functionalities you are used to from the classic ABAP Workbench (transaction: SE80). In contrast to SE80, however, Eclipse is optimized to be operated with keyboard shortcuts. Keep that in mind and take advantage of the opportunities to internalize the keyboard shortcuts. For more information, see Chapter 3 - Working with ADT.

1.6 Which Eclipse feature is available in which version?

Since the functionality of the ABAP Development Tools is continuously being expanded by SAP, it is important to check which functions are available in the ADT version you are using. It should be noted that some of the functions provided also depend on the release of the SAP system used. To find out the current status of the latest updates of ADT, the official sources of SAP are the means of choice, as they are continuously maintained.

For Cloud:

https://help.sap.com/docs/BTP/5371047f1273405bb46725a417f95433/ab03dcd9072f4a2d85c945d05929d3fb.html

For On-Premise:

https://help.sap.com/doc/2e9cf4a457d84c7a81f33d8c3fdd9694/Cloud/en-US/inst_guide_abap_development_tools.pdf

2 Motivation for ADT

In this chapter, we will introduce you to several reasons for using the ABAP Development Tools (ADT) as a development environment. At the end, we also go into what is sometimes the biggest "sticking point" for long-time ABAP developers. We give hints, why the aspects, which are often considered as disadvantages, are actually advantages.

We would be pleased if we can convince you of the benefits of ADT with this chapter or at least arouse your curiosity to study this guide.

2.1 You want to apply and implement modern ABAP.

The first and most obvious reason for a comprehensive use of the ABAP development tools in Eclipse is the strategic orientation on the part of SAP. The ABAP Development Tools were introduced in 2012 and have developed into a stable development environment with a wide range of functions over these ten years.

In contrast, the SAP GUI-bound development tools with their most prominent example, the ABAP Workbench, are in maintenance mode. This means that bug fixes will continue to take place here, but no new features will be delivered.

This may not be the most motivating reason in the short term, but it will one day catch up with every developer in the SAP environment. Therefore, we recommend that you start and switch to ABAP development tools today rather than tomorrow.

For this reason, SAP recommends the use of ADT as the standard environment for ABAP development to benefit from new functions and corrections with each release. The actual range of functions depends on the release status of the ABAP stack of the SAP systems used. A rough overview and further information can be found in the section: Chapter 1 - Introduction to Eclipse and ADT.

2.2 You want to use one development environment for everything.

2.2.1 Technological aspect

With the ABAP Development Tools (ADT), you can develop not only for on-premise systems such as SAP ERP or S/4HANA. ADT are also the only way to perform developments for cloud systems such as the Business Technology Platform (BTP).

This circumstance is particularly important for ABAP developers who develop in the SAP Cloud ABAP Environment ("Steampunk"). In this context, classic, SAP GUI-

oriented development tools cannot be used, and developments can only be carried out with ADT.

Closely related to this is also the work with some new development artifacts. The creation or maintenance of CDS views is only possible with ADT. And SAP's new programming model, the "RESTful Application Programming Model" (RAP), can only be used with ADT.

2.2.2 Development on multiple system lines

One development environment for everything applies to ADT but also if you are allowed or must work on several development systems. In the Eclipse environment, various systems are listed as projects, and you can access them very conveniently and clearly. By means of the working sets (see chapter 3 - "Working with ADT"), the systems can be grouped into umbrella terms and even color-coded using additional plug-ins (see chapter 7). There is no need to open the systems from the SAP Logon Pad, log on to the system and open the SE80. This results in numerous other synergies in the work, which are described in this guide.

2.3 You want to break free from the technical limitations of the SAP GUI.

2.3.1 Six sessions in one system are simply not enough

SAP GUI-based development tools such as the ABAP Workbench are subject to various limitations due to their execution in the SAP GUI. These are not relevant when you work with ADT.

This includes, for example, the dependence on the maximum number of simultaneous SAP GUI sessions (modes, cf. parameter "rdisp/max_alt_modes"). This value is specified individually for each system called via SAP GUI. By default, there are at most six simultaneous SAP GUI sessions per user in the same system. This restriction does not apply to ADT.

If you are a developer working with different systems at the same time, you benefit from another advantage: Each connection to a system is maintained as an "ABAP Project" for on-premise systems or "ABAP Cloud Project" for cloud systems within ADT. The connections to these systems can be used simultaneously. This allows you to copy source code from a source system to the clipboard and paste it into a target system that is open at the same time, all within the same Eclipse application window.

In addition, a simple code comparison between different systems, even between systems without an existing RFC connection, is also possible in this way. Displaying multiple SAP GUI windows as needed previously is no longer necessary.

2.3.2 You don't want to lose code due to network interruptions.

Due to the technical conditions of the SAP GUI, it is necessary that the network connection is stable and uninterrupted. If there are network interruptions while you are writing code in a GUI window, it may happen that the work of the last few minutes was in vain because SAP GUI has lost the connection to the server and closes the window.

With ADT network interruption is no longer a problem. The Eclipse window remains open even if the connection is lost. Once the connection is restored, the code can be saved in the SAP system.

If major problems occur in the network, the code can simply be cached in its entirety as text using copy-and-paste in an alternative text editor until the SAP system is available again.

2.4 You like to improve your ABAP code through refactoring.

The maintainability of development artifacts is a central challenge of software development. To achieve good maintainability, the development procedure is often based on clean code principles (see Clean ABAP). To comply with the clean code principles, repeated revision of development artifacts, also known as refactoring, is essential.

The ADT support typical refactoring tasks with the help of the Quick Assists functions, which also include the Quick Fixes. The Quick Assists are context sensitive. For example, you can outsource complex source code sections to smaller and independent methods, which significantly increases comprehensibility and thus maintainability. Without the support of Quick Assists, such an iterative approach is much more labor-intensive and error prone.

Further and more detailed information on refactoring and the tools provided to you in ADT for this purpose can be found in Chapter 3 - Working with ADT.

2.5 You like to adapt the look and feel of your development environment to your wishes.

ADT are based on Eclipse, a widely used open-source development environment. Strengths of this development environment are, among other things, the wide range of customization options such as in the display (cf. Views and Perspectives in Chapter 3 - Working with ADT) or in the shortcuts (keyboardshortcuts) and the support of helpful functions such as e.g. a comprehensive search.

As a user of Eclipse, you can therefore customize and use the development environment in a variety of ways, which allows each user to better support their preferred way of working.

Since ADT allow access to different SAP systems at the same time, you can work consistently across all systems. This is in contrast to working with SAP GUI-based development tools such as the ABAP Workbench, which must be individualized separately in each system.

With the so-called workspaces, you have the option of saving different configurations of the working environment. For example, different projects, different favorite packages, which views and objects are open and even the place in the code where the last work was done are stored in the respective working set. In addition, different workspaces can be opened in several Eclipse instances running in parallel. This gives you the opportunity to put together the most efficient environment depending on the project, customer, or task. See Chapter 3 - Working with ADT and Chapter 6 - Best Practices Eclipse Configuration for information on how to use workspaces.

2.6 Would you like even more tool-based support?

If the range of functions provided by SAP is not sufficient for you, you can extend it with additional plug-ins that are available as a supplement to ADT.

A selection of these plug-ins is presented in Chapter 7 - Plug-ins . With the right know-how, you can also create your own plug-ins and make them available to the SAP community.

In the past, various ABAP developers have already made use of this option and developed their own plug-ins, which they make available to the SAP community free of charge. An example of this is the ABAP Quick Fix. This is an extension of the quick fixes offered in the standard ADT to support the automatic conversion of the classic ABAP syntax into the respective modern ABAP syntax counterpart.

2.7 You already know Eclipse from other programming languages? Great!

Eclipse as the basis of ADT is already in use in various companies. This is because it is one of the most widely used development environments for JAVA, for example. JAVA, on the other hand, is a frequently used programming language (cf. TIOBE Index) and therefore plays a role in many development projects. In addition, Eclipse can also be used for other development purposes, e.g. for development in the Python programming language or working with data in the Extensible Markup Language (XML) format.

So, there is a certain probability that developers and companies already have previous knowledge of using Eclipse as a development environment. This makes it easier and faster to extend the use of ADT than to use a completely new, possibly completely unknown development tool.

2.8 You are interested in new things.

Finally, we would like to point out the interest of many people in new things and the associated attractiveness as a motivating factor for the change. If the reasons for switching in the previous sections were more factual aspects, this one is about emotions and values deeply rooted in people: the interest in something new and the attractiveness that something new has. Or in short: about progress and change.

For some people, the standstill with occasional small changes means a certain security in everyday working life. However, this security is deceptive, as the (IT) world is changing rapidly, and existing technologies and methods do not always offer the right answers to current and future challenges.

Especially in the ABAP world, a lot has changed in the last ten years. After numerous new language constructs have enriched ABAP, new artifacts such as CDS views and new programming models such as the ABAP RESTful Application Programming Model (RAP) have been added. And technological change such as the path to the cloud cannot be dismissed out of hand. Consequently, this also requires the further development of the tools used in development.

In addition, a technological standstill can also lead to the exclusion of subsequent generations of developers, as they often look at technologies with a completely different perspective, compare their previous experience with other development environments and programming languages and make their evaluation on this basis.

2.9 The benefits for the organization

With this guide, we want to motivate not only the ABAP developer to use ADT. There are also numerous advantages for the organization through the extensive use of ADT in the SAP development area.

In addition to the higher efficiency of the developers, among other things through better tool support for creating and improving the code, future viability and uniform tooling are again the most important reasons for the organization to promote and demand the use of ADT in the SAP development teams.

To enable smooth use for the individual developer and to avoid obstacles to a changeover right from the start, and thus to achieve a broad use of ADT, the following topics must be clarified centrally and made available to developers in the form of generally available documentation:

- Framework conditions and general information on ADT
- Installing Eclipse
- Access from Eclipse to resources on the Internet (updates and plug-in installations)
- Developer permissions for ADT objects in the backend (S_RFC for ADT*)

Again, this is an initial investment, but it pays off as soon as the development teams work uniformly with ADT and the advantages of the development environment can be fully exploited in their daily work.

You can find a lot of information, assistance, and best practices in Chapter 5 - Installation, Distribution and Update Strategies.

2.10 You are missing the form-based editor

ADT use a text-oriented representation of development artifacts in many places, which you are familiar with from other development environments such as Microsoft Visual Studio Code (VSCode). This eliminates the previous, form-based presentation that you are familiar with from the SAP GUI-oriented development tools.

This is certainly one of the biggest hurdles for many ABAP developers who want or should switch from GUI-based tools to Eclipse. To overcome this hurdle, you will find a detailed introduction to working with ADT in Chapter 3 - Working with ADT.

This change in presentation is also accompanied by a certain change in the way of working. There is no need to frequently switch between different SAP GUI interfaces that are in a specific context. Instead, there is often "only" text – i.e., instructions that

are entered or read. This leads to a strong focus on the actual instructions, their effects, and relationships (syntax and semantics).

An example of this changed representation is the signature of a function module. With the SAP GUI-based development tools, the signature of a function module consisting of IMPORT, EXPORT, CHANGING, TABLES and EXCEPTIONS is displayed as five separate registers. In ADT, the signature is displayed and maintained as text. There is no need to switch between different context-bound registers.

After the initial adjustment, you will certainly quickly recognize the advantages of the text-oriented way of working, which result from the omission of the navigation steps through the GUI and from the numerous supports such as code completion and quick fixes.

2.11 Why ABAP Development Tools

If you have not yet been convinced by the advantages of ADT described so far, we would like to motivate you to put yourself in the shoes of ADT users with quotes from the authors of the guide. Perhaps you will find the impetus here to take up the topic after all.

Michael Keller: "*Clean ABAP without ADT and thus the support of the quick fixes is unthinkable for me - after all, they save the developer a lot of time and work.*"

Florian Henninger: "*Refactoring without ADT is a bit like trying to eat soup with a fork - it can work, but no one does.*"

Bärbel Winkler: "*While helping to create this guide, I learned many good reasons to work with ADT more often from now on than I did before.*"

Jens Zähringer: "Although the ABAP *development tools have been available for over 10 years, I recently made the switch from ABAP Workbench to ADT for myself. The transition wasn't without its challenges, but in the end, it was definitely worth it!"*

Peter Luz: *"Using the where-used list, you can quickly determine where a method is used and how it is called there. Then rename this method that is used multiple times. Then extract a piece of code from it into your own method. Finally, compare the code on the central development system with the version on the Q machine of a system line. In ADT, by using a few keyboard shortcuts it is done in seconds. This makes creating and revising software fun and helps to significantly improve the quality of the software by using the tools described here in the guide. For me, creating ABAP software without ADT is now unthinkable."*

Michael Biber: *"Yes, SE80 is now quite good. However, I see this like the switch to object orientation: At the beginning you ask yourself 'Why?' and see existing stumbling*

blocks. However, once you have experienced the other side (object orientation, ADT ...), you don't want to go back. For me, the advantages of better clarity (element info), theoretically infinitely open sources in parallel, live syntax checking and many more outweigh all the small media breaks (→ older SAP releases) and other approaches."

Björn Schulz: *"Without ADT, I would be much slower to get information out of the system and probably wouldn't be able to cope with only six modes."*

Dr. Wolfgang Röckelein: "*Is there ABAP development without ADT?" "No clean code without refactoring, no refactoring without ADT.*"

Uwe Fetzer: "*There is no faster, more convenient, and safer way to develop ABAP Clean Code*."

Sebastian Freilinger-Huber: *"It's better to leave the 'SE80 comfort zone' today than tomorrow - it's worth it. If you still have doubts, you will find numerous arguments for the switch in the following chapters."*

3 Working with ADT

The ABAP development tools offer many functions, which also offer correspondingly numerous possibilities for use. This can be confusing for developers who are new to ADT.

The first step is always the hardest. At the beginning of this chapter, we would like to make it easier for beginners to get started in the ADT world using the example of creating a class. The procedure shown here can then be transferred to other development objects, and the entry is done.

The other functionalities are described in the following section, and hints are given, and best practices are explained on how the numerous tools and aids can be used in daily work and offer added value.

In addition to ADT beginners, developers experienced in ADT will also find many useful hints and perhaps also new things for their daily work with the ABAP development tools.

With the SAP ABAP Development User Guide, hereinafter referred to as the User Guide, SAP provides the official documentation for ADT. For further information on individual functions, the links to the corresponding section in the user guide are noted.

3.1 Introduction: Basics of working with ADT

3.1.1 Getting Started with ABAP Development Tools

This section is intended for developers who have not yet worked with ADT and want to get started. In the user guide, in the "Getting Started" section, there is a section, where all functions are explained in detail. In this guide, we would like to make it as easy as possible to get started by means of a step-by-step description of how to create an ABAP class and show the advantages of using the ABAP development tools.

The following steps are described step by step:

- Setting up the project
- Setting up the Favorite Packages
- Creating a class and a method
- Working on code and refactoring

For the purpose of general traceability, the examples shown here are displayed in an instance of the BTP Trial Account, but they can be used in common on-premise systems without any problems.

Developers who have already gotten started and want to get an overview of the individual functions can skip this section.

3.1.2 The shift from form-based to text-oriented code creation

In contrast to SAP GUI-based transactions such as SE80 or SE24, ADT does not have a form-based editor. The creation of classes (and also function modules, etc.) is purely text-based in ADT. For development tools that are not yet available in Eclipse depending on ADT and backend version, these transactions can be accessed from Eclipse in an integrated manner.

It also takes some time to get used to swapping the **F2** and **F3** keys. While the F3 key is used as the back key in the SAP GUI, the F3 key in ADT is used for forward navigation, the F2 key for context-sensitive help.

This is sometimes the biggest hurdle for getting started, as long-established practice and familiar work processes change with the switch to ADT. And if you want to go fast, you like to fall back on familiar and well-known ways of working.

The introduction and the changeover therefore require time and effort of practice. But the initial additional effort pays off after a short time. Because after a little acclimatization and practice, the points described are no longer a problem. The numerous functions offered by the ABAP development tools make it easier to write and revise ABAP code and thus increase the efficiency of development. Therefore, the switch from SE80 & Co. on ADT in Eclipse can be seen as a personal investment in an efficient and future-proof way of working.

3.1.3 Context of the exercise

The example shown here is deliberately simple, as the basic functions and working methods of ADT are primarily to be presented.

We would like to create a small class that offers the following functions:

1. Determination of flights from the table ***/DMO/FLIGHT*** as entered.
2. Calculation of the available seats of the flight.
3. Calculation of the fare based on an additional percentage fee.

The class serves as an internal service class and does not provide a UI or output data.

It shows the basic working methods and most frequently used functions that lead to efficiency gains during code creation and modification.

3.1.4 Connecting the Development System – New Project

A development system is represented in ADT in the form of a project. Therefore, to link a development system to ADT, we need to create a new project.

In a new installation of ADT, a new project is created via

File → New → ABAP Project

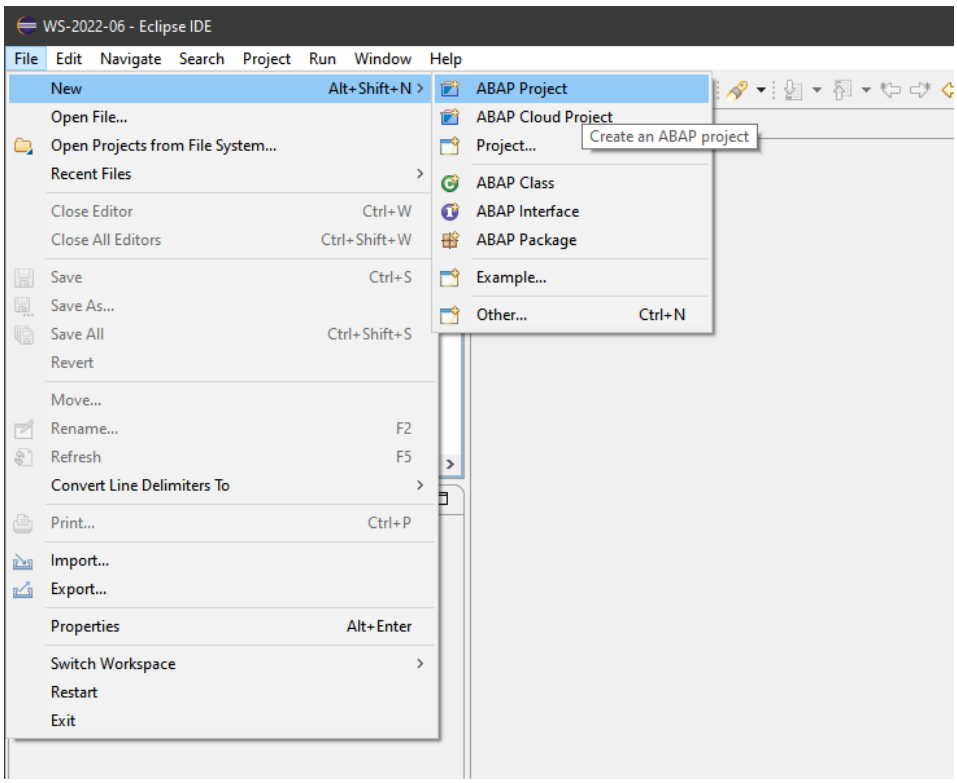


Figure 1 Creating an ABAP Project in Eclipse

When you create an ABAP project for on-premise systems, the list of systems linked in the SAP logon is displayed. The login data must be stored if no SSO is used and in the last step the project can be given a descriptive name. The language English is selected as the default.

The newly created project and the associated development system is displayed in the so-called Project Explorer.

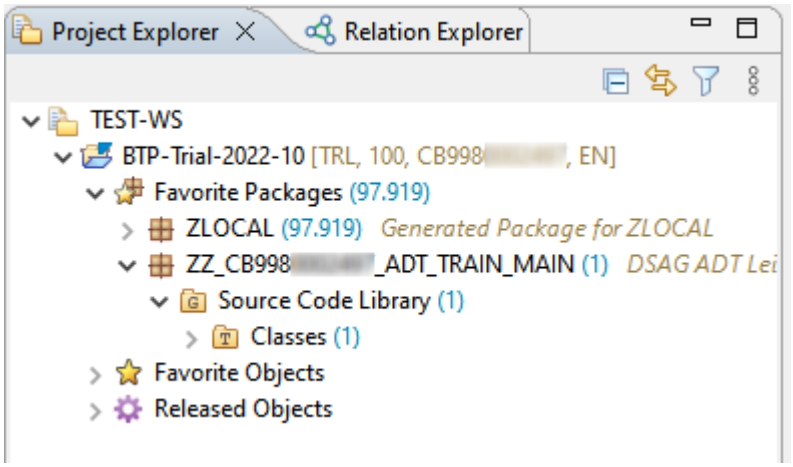


Figure 2 The Project Explorer

The Project Explorer is the central entry point and object catalog after the corresponding development system has been opened. The objects are displayed in hierarchical form based on the packages, as already known from the SE80. In daily work, the objects to be edited are opened from this.

The main workspace is the Favorite Packages node. To add the package to the Favorite Packages into which the class is to be inserted, execute the "Add Package" command using the context menu.

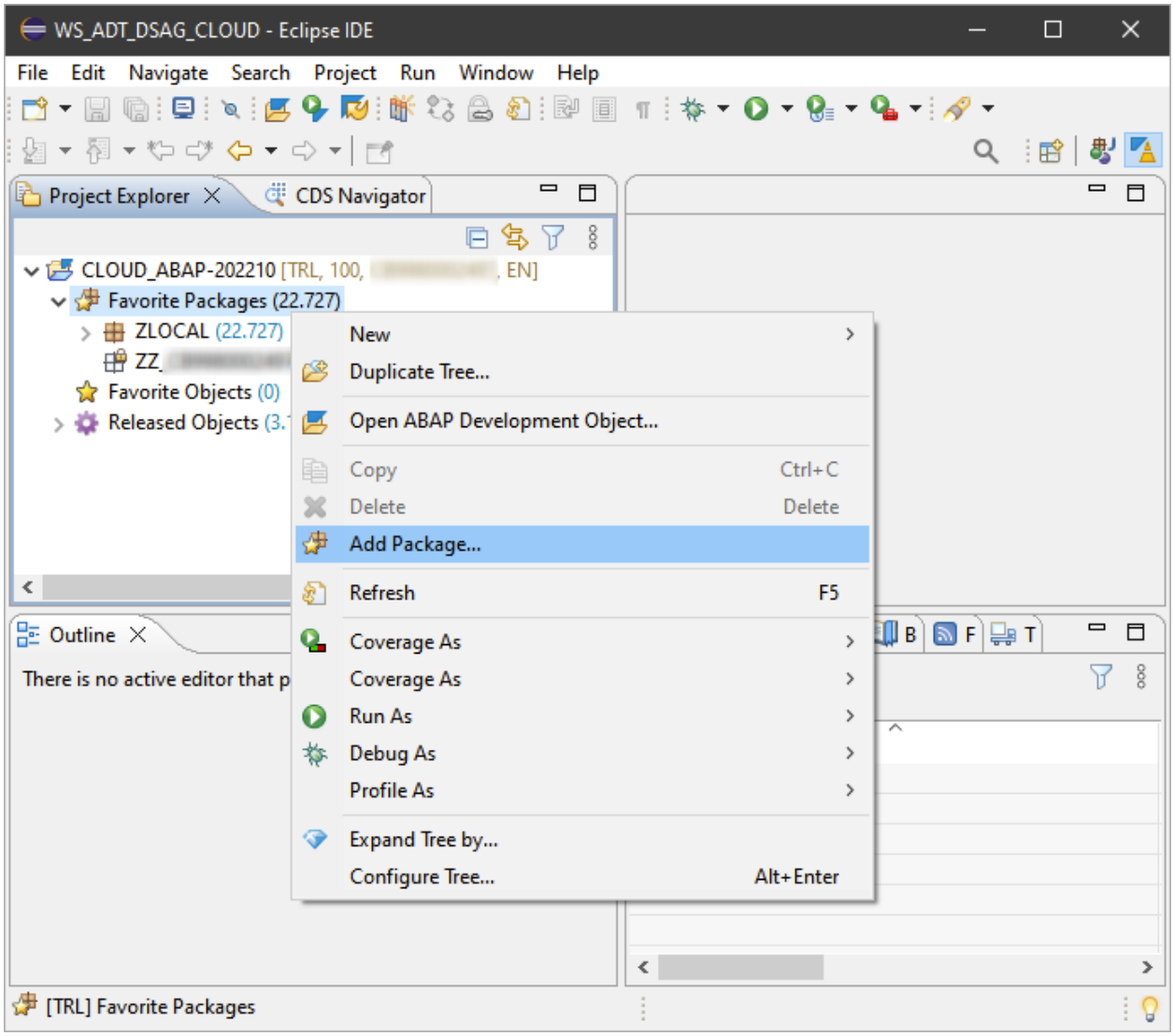


Figure 3 Adding Packages to Favorites

3.1.5 Creating a class in text mode

To create a new ABAP class, navigate to the desired package in the Project Explorer, right-click on the context menu, and find the command

New → ABAP Class

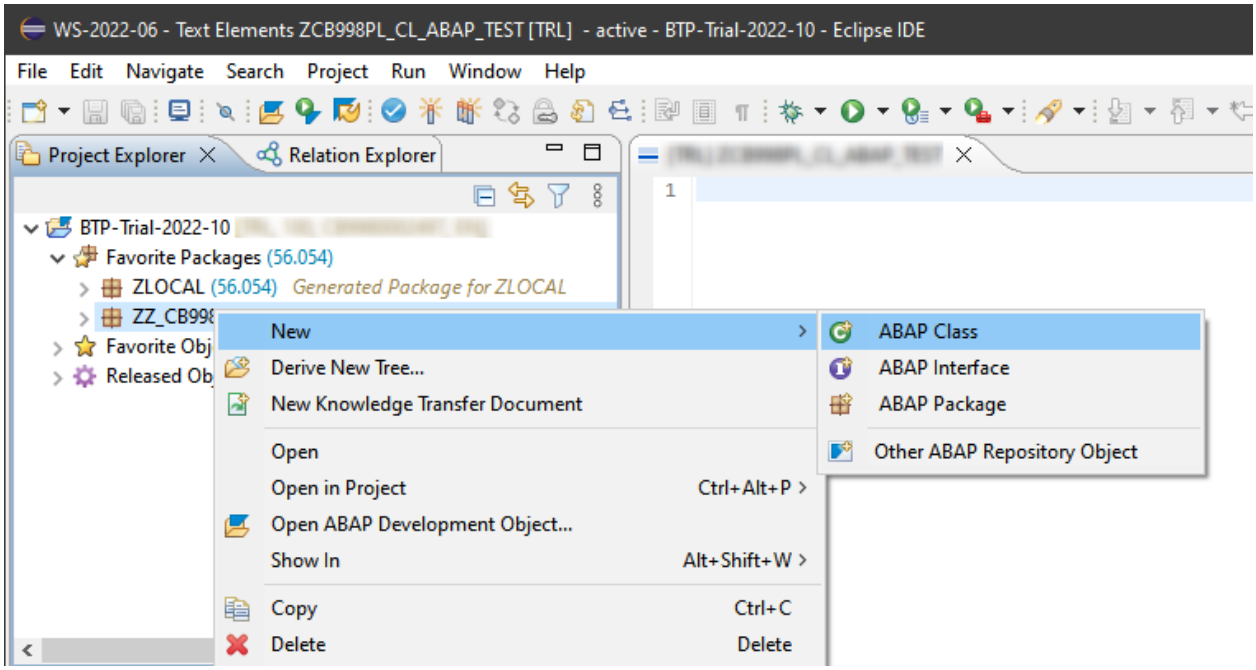


Figure 4 Creating a New ABAP Class in Project Explorer

A window opens in which the data of the class can/must be specified.

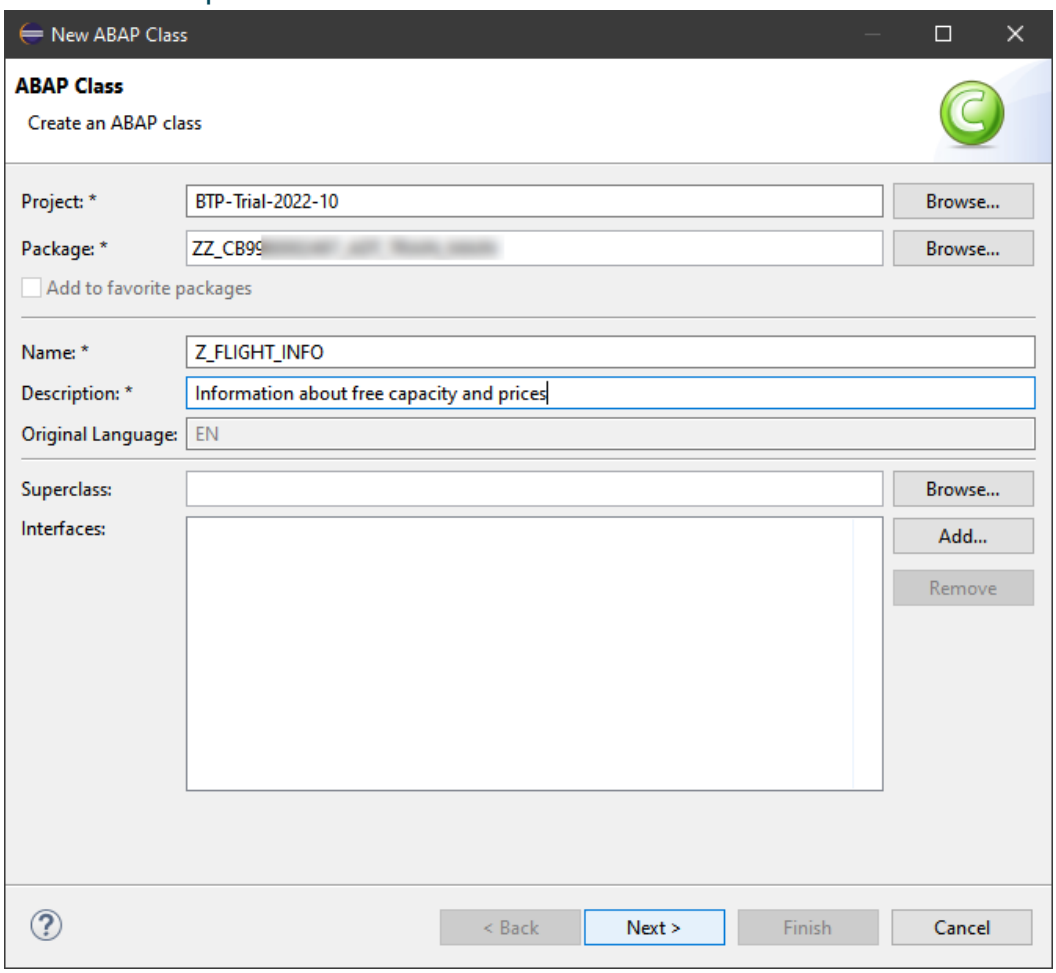


Figure 5 Properties Dialog: Creating an ABAP Class

If required, the super class and interfaces to be referenced can already be specified here. However, this can also be done later text-based directly in the source code. After clicking on "Next", the window for selecting or creating the transport request opens.

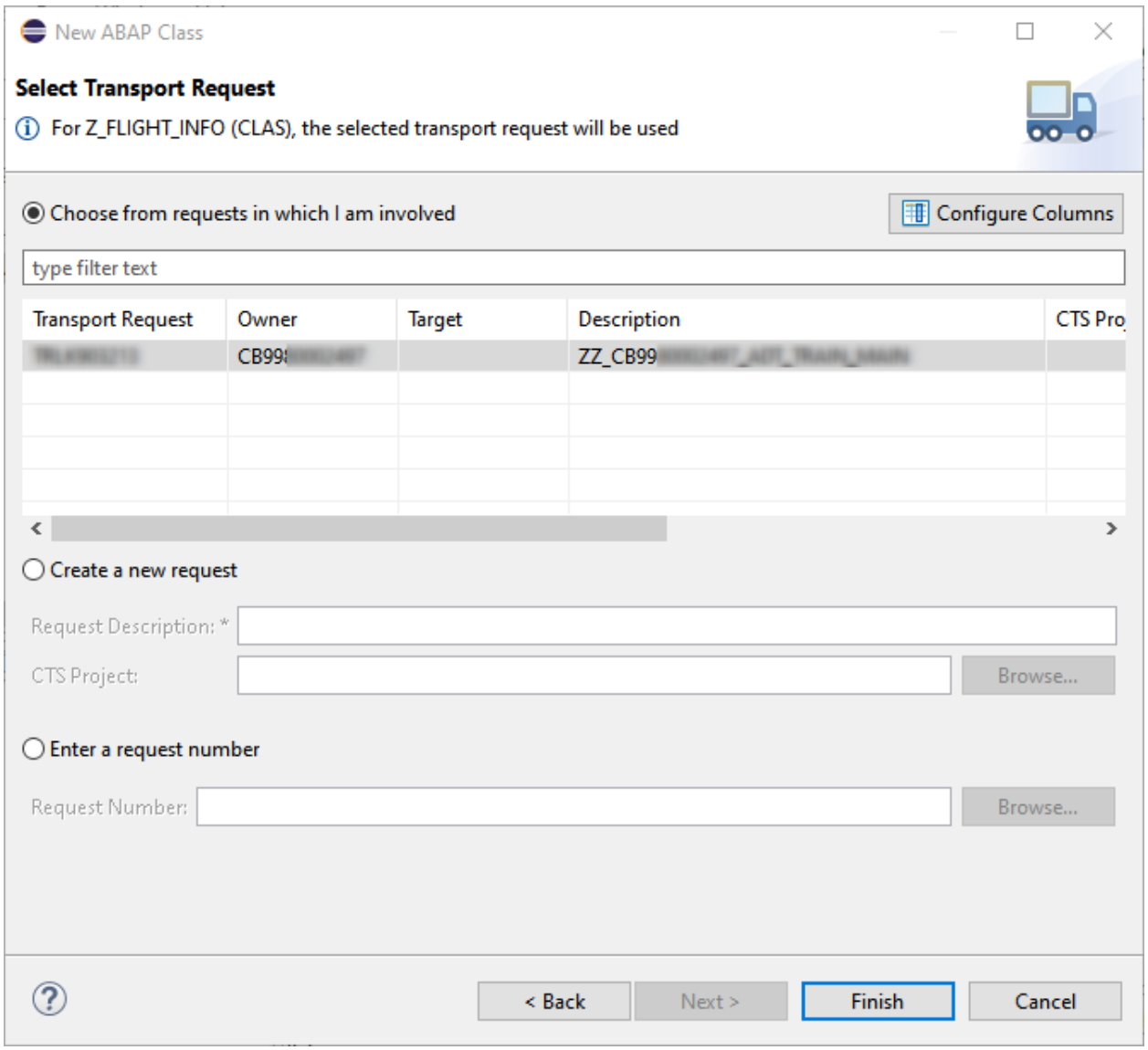


Figure 6 Transport Request Dialog

After clicking on "Finish", the class is created, and it can be found in the object tree of the Project Explorer and in the source code editor on the right side of ADT.

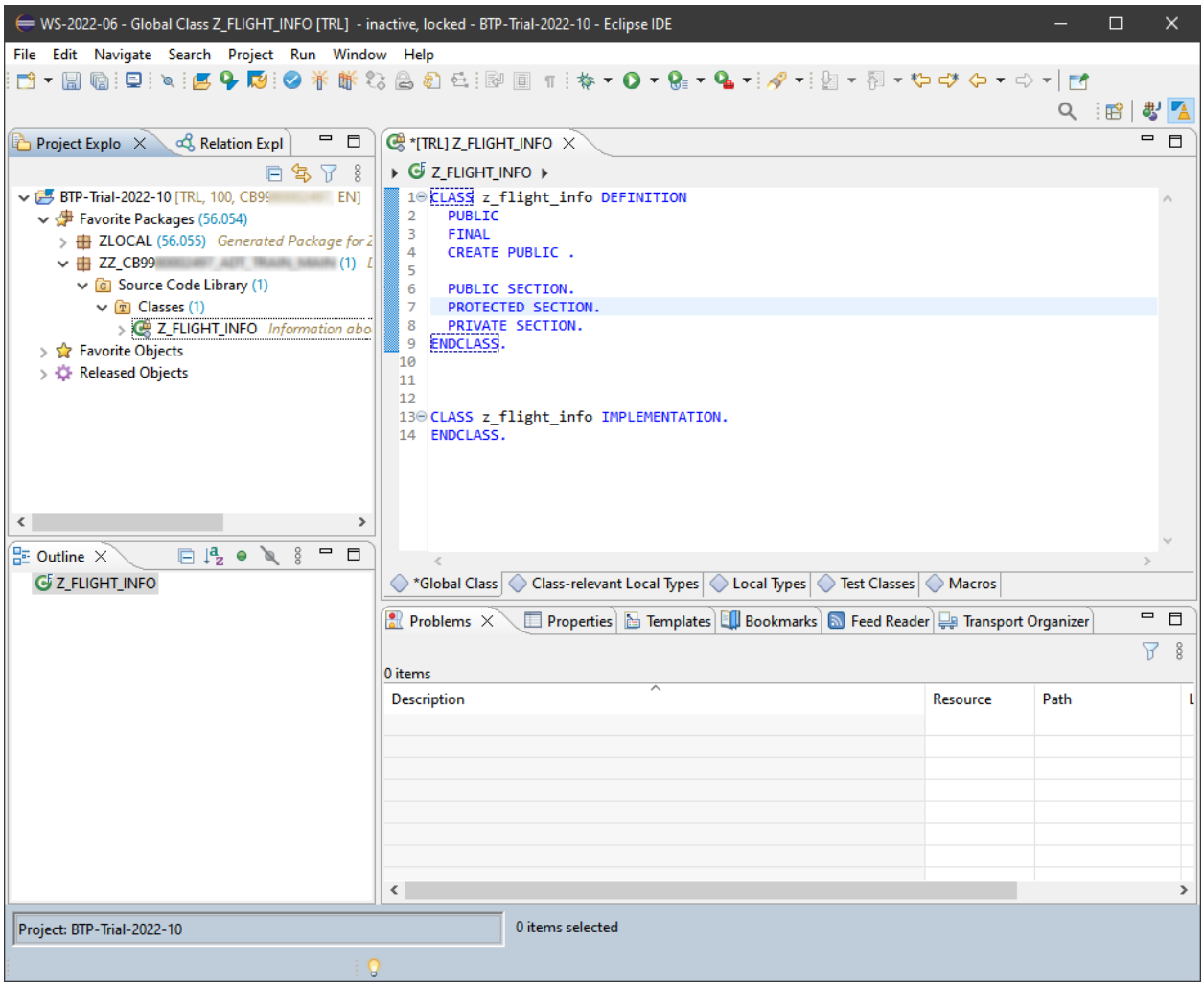


Figure 7 Display of the new class in ADT

All further operations on the class are now performed in the source code, i.e., all types, data definitions and methods are written text based as source code in the class. This seems complicated and unusual at first, but ADT offer some functions that make the elaboration of the class very efficient.

3.1.6 Definition of a method in the class

Now we want to create the first method of the class, which reads data from the /DMO/FLIGHTS table and outputs the number of free seats for a defined flight.

We limit ourselves here only to the core functionalities and will not provide any additional functions for output, etc.

In ADT, an ABAP class is divided into the main areas "Definition" and "Implementation". Accordingly, for our first method, we will first define the method with its parameters in the "Definition" section and then perform the implementation with the source code. The so-called "Quick Fix" function will save us typing work.

To create a method, navigate to the "Global Class" tab in the

```
CLASS <classname> DEFINITION
```

and places the cursor in the visibility area of the class in which the method is available. In our case, the method should be visible to other users and should therefore be defined in the PUBLIC area. The definition of the method is introduced with the METHODS keyword.

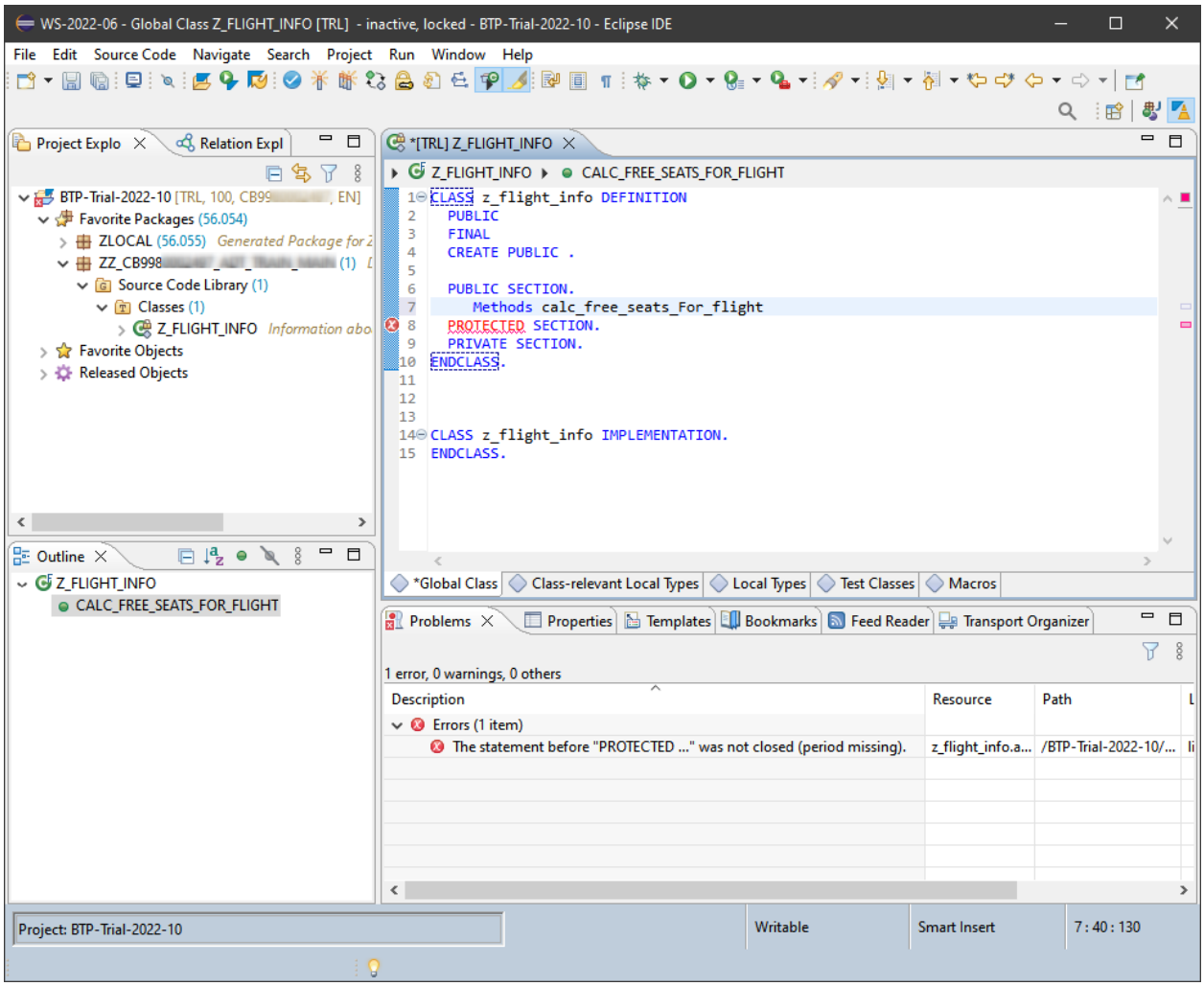


Figure 8 Editing the Class

Already at this point, one of the main advantages of ADT over the SE80 becomes apparent. As soon as code has been entered, the syntax check runs automatically and indicates whether the code is syntactically correct.

Since the completion point does not exist, ADT displays a hint (as a hover message) in the left bar as well as the syntax error in the view problem. There is no need for an extra syntax check. Just a small thing that means a significant increase in efficiency in daily work. At the latest when you make a change in the GUI-based tools for various reasons, you will miss this function.

3.1.7 Automatic completion and formatting of the code

We complete the method definition with the creation of the parameters and the completion point. The use of code completion makes our work much easier. To do this, we only enter the first two to three letters of the desired keyword. The keyboard shortcut **`CTRL+SPACEBAR`** shows us the appropriate keywords. These can then be selected from the suggestion list using **`TAB+ARROWKEYS`** For further automation of code creation, the use of templates is recommended (see section Code Template).

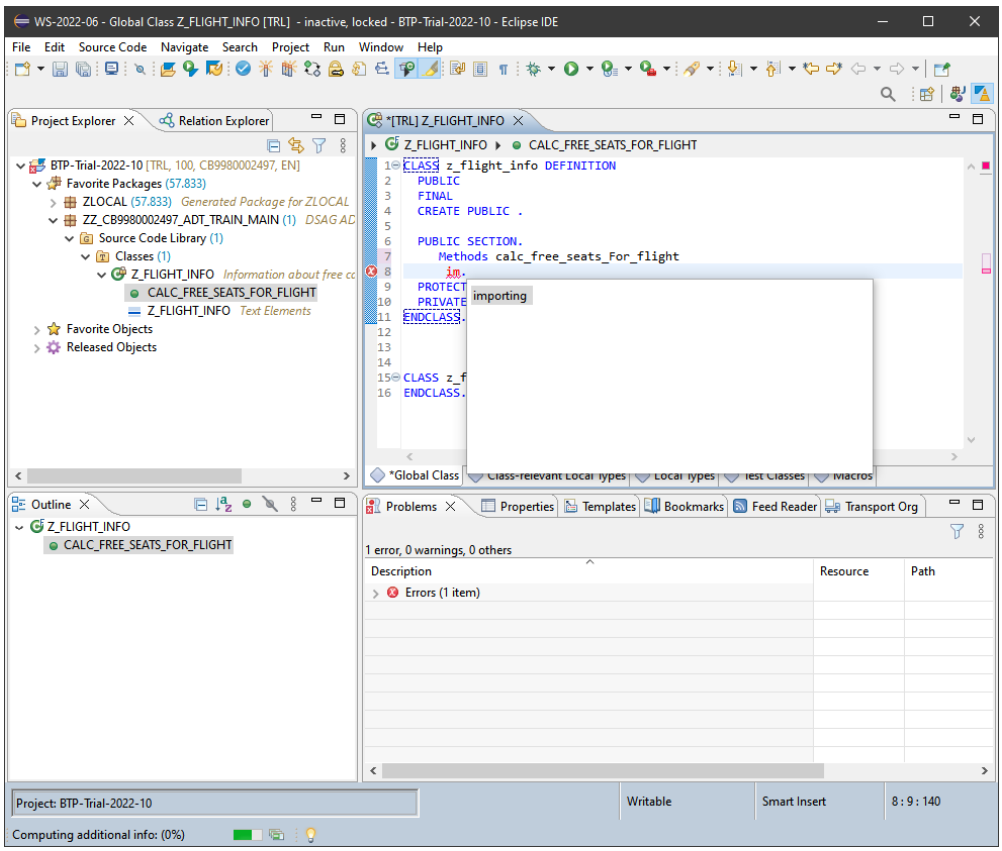


Figure 9 Example Code Completion for the Import Parameter

After the import and return parameters have been defined, the cursor is positioned in the text area of the code and the formatting of the code is executed using the right-click → Source Code → Format context menu or the keyboard shortcut **`Shift+F1.`**

This corresponds to the Pretty Printer in the GUI transactions. The code is then saved using

CTRL+S.

If there is a syntax error, the problem view and the color marking in the code will show the error. If the code is syntactically correct, the artifact can then be activated using **CTRL+F3**.

After practicing the new, mainly keyboard-based, way of working, an automatism is created after a short time, which, after writing a few lines of code and subsequent sequence of the above-mentioned key combinations, results in a formatted, checked, and saved code. This saves time-consuming surprises in the form of syntax errors when activating, as can happen in the GUI tools.

3.1.8 Implementation of the method using Quick Fix

The method is now defined, due to the missing implementation, ADT displays the following error in the problems view:

”Implementation missing for method "CALC_FREE_SEATS_FOR_FLIGHT".

This "problem" can be solved very efficiently with the help of the Quick Fixes.

To implement the method, use the Quick-Fix-Function, which can be called up via the context menu or the key combination **CTRL+1**.

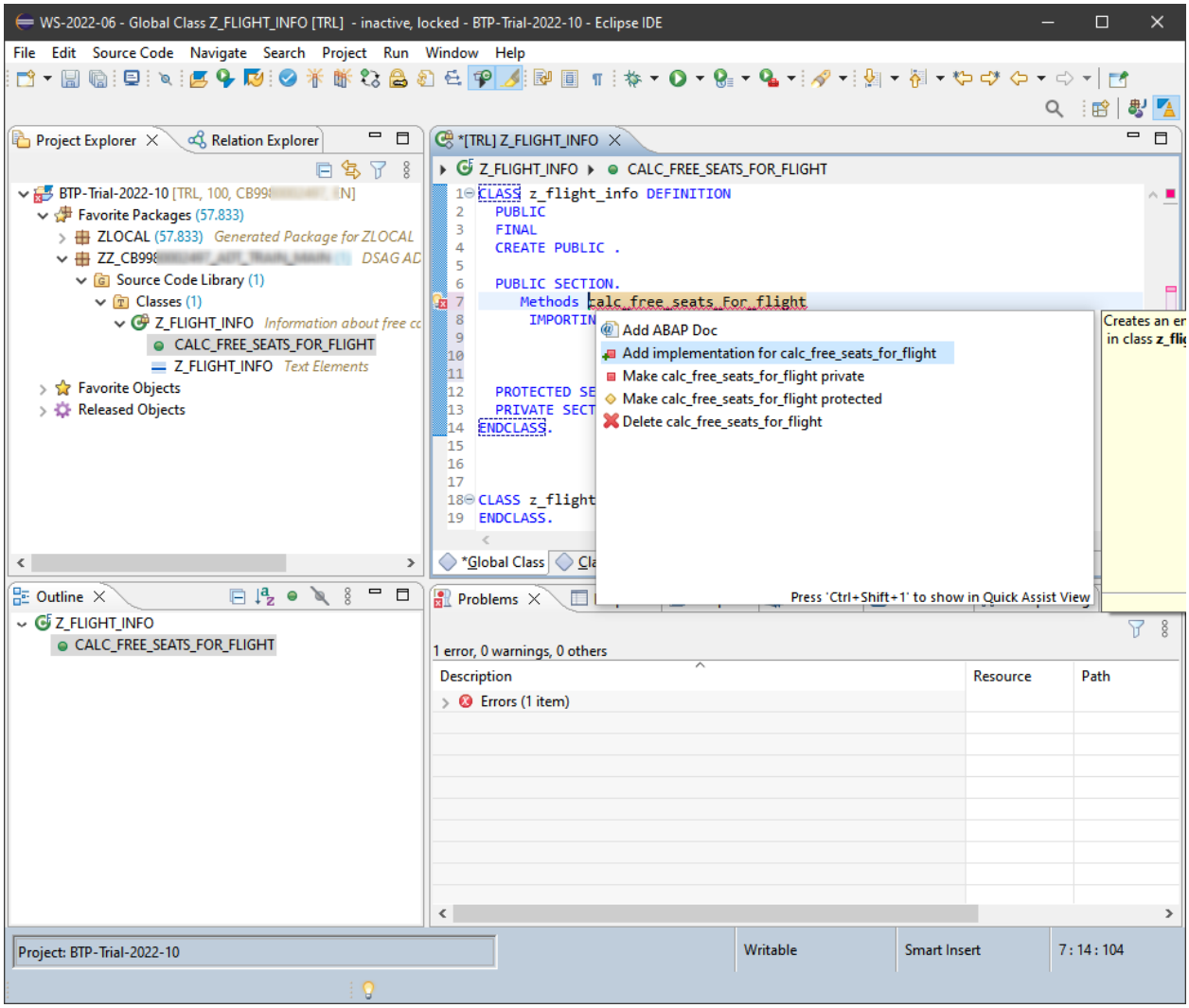


Figure 10 Using the Quick Fix for Method Implementation

You select "`Add Implementation for ...`" and confirm with `ENTER`. To run the quick-fix commands, the code should be saved and error-free, as some quick fixes (automatic code corrections/refactorings) require this.

The method implementation here means that the section

```
METHOD

ENDMETHOD
```

in the area of the "Class Implementation" is created by ADT and thus an empty method implementation exists. The development of the method logic within this area is then your task as a developer.

You can jump back and forth between definition and implementation simply by pressing the **`F3`**-key.

By placing the cursor on the method name and pressing **F2** you can display the parameters of the methods. This makes it easier to write the code and saves cumbersome navigation into the definition area. Alternatively, the ABAP Element Info is available for this purpose.

In our tutorial, we use the import parameters to read the desired record using a SELECT command. Subsequently, the number of free seats is calculated and returned to the user as a returning parameter.

Again, using Code Completion, which is invoked with the keyboard shortcut **CTRL+SPACEBAR**, helps to create the code efficiently and free of typos.

To avoid writing the whole returning parameter "r_f_free_seats" and to get type information, we write the structure and the component separator "-" and use the key combination **CTRL+SPACEBAR** to display the components, which can then be selected and inserted into the code.

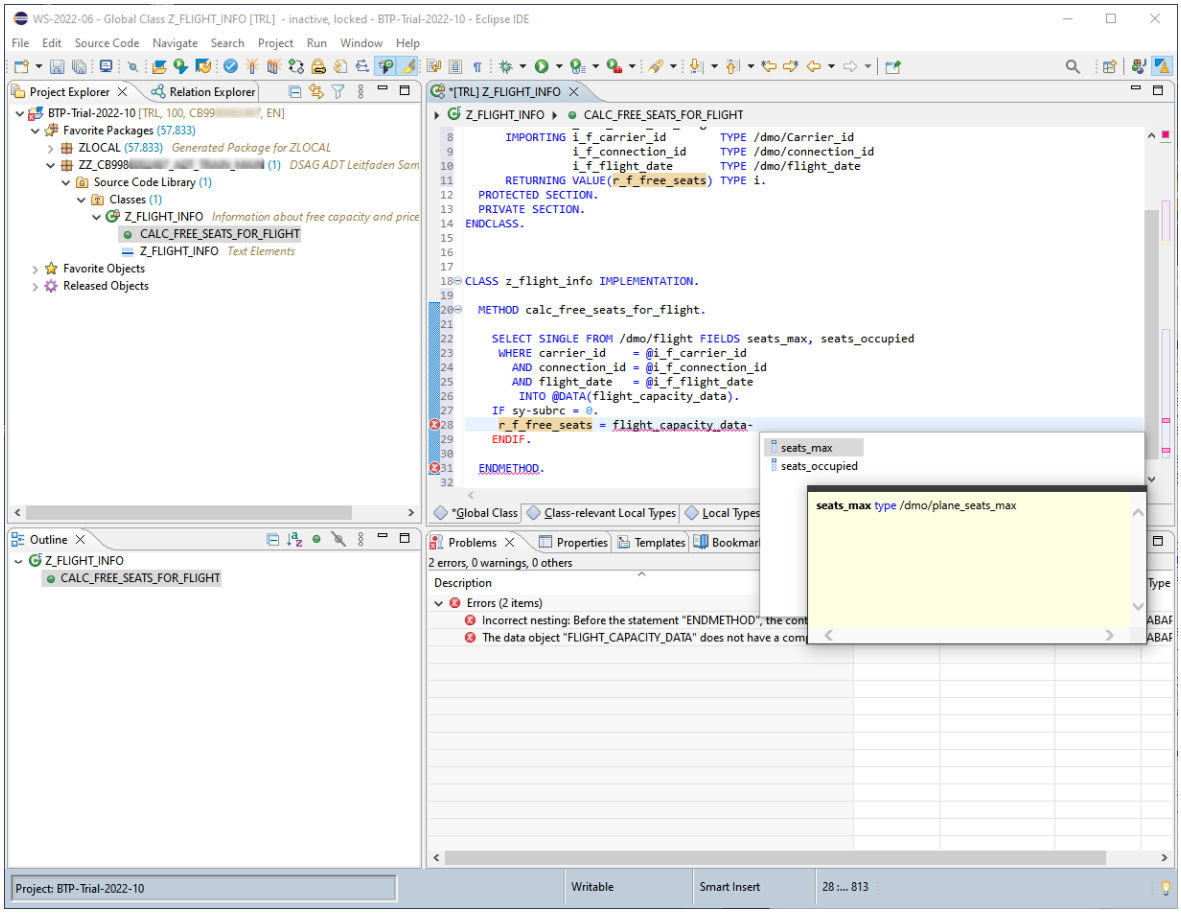


Figure 11 Component Selection Using Code Completion

These options for forward navigation and inline display of further element information are not available in this form in SE80 and are a further building block for increasing developer efficiency in ADT.

We have thus defined and implemented the class with the first method in ADT.

3.1.9 Renaming of Parameters – Refactoring

The ongoing optimization of existing code is an important task of every developer. This is supported by ADT in the best possible way. The different ways to refactor are explained in detail in the section: Refactoring Code with ADT. Further information can also be found in the User Guide.

We would like to present the general procedure of refactoring in detail by means of a renaming.

As usual, we used the Hungarian notation and now want to rename the parameters of the method to remove the prefixes (see ABAP Clean Code - https://github.com/SAP/styleguides/blob/main/clean-abap/CleanABAP.md).

While such an action can be very time-consuming and error-prone in the SAP GUI environment, the rename function in ADT offers a very convenient way to rename variables, parameters, and method names across all uses. This means that in all development objects that call the method, the parameters are automatically renamed. There is no need for a time-consuming search for users via the where-used list. Of course, this only works if no dynamic method calls are used. ADT cannot detect such cases.

Thus, code cleaning and refactoring with ADT can be carried out very efficiently, and the risk of resulting errors is significantly reduced compared to the manual method.

To rename the parameters, the Rename function from the context menu via right mouse button → source code → Rename or the keyboard shortcut

`ALT+SHIFT+R` or via Quickfix selection via

`CTRL+1`

executed.

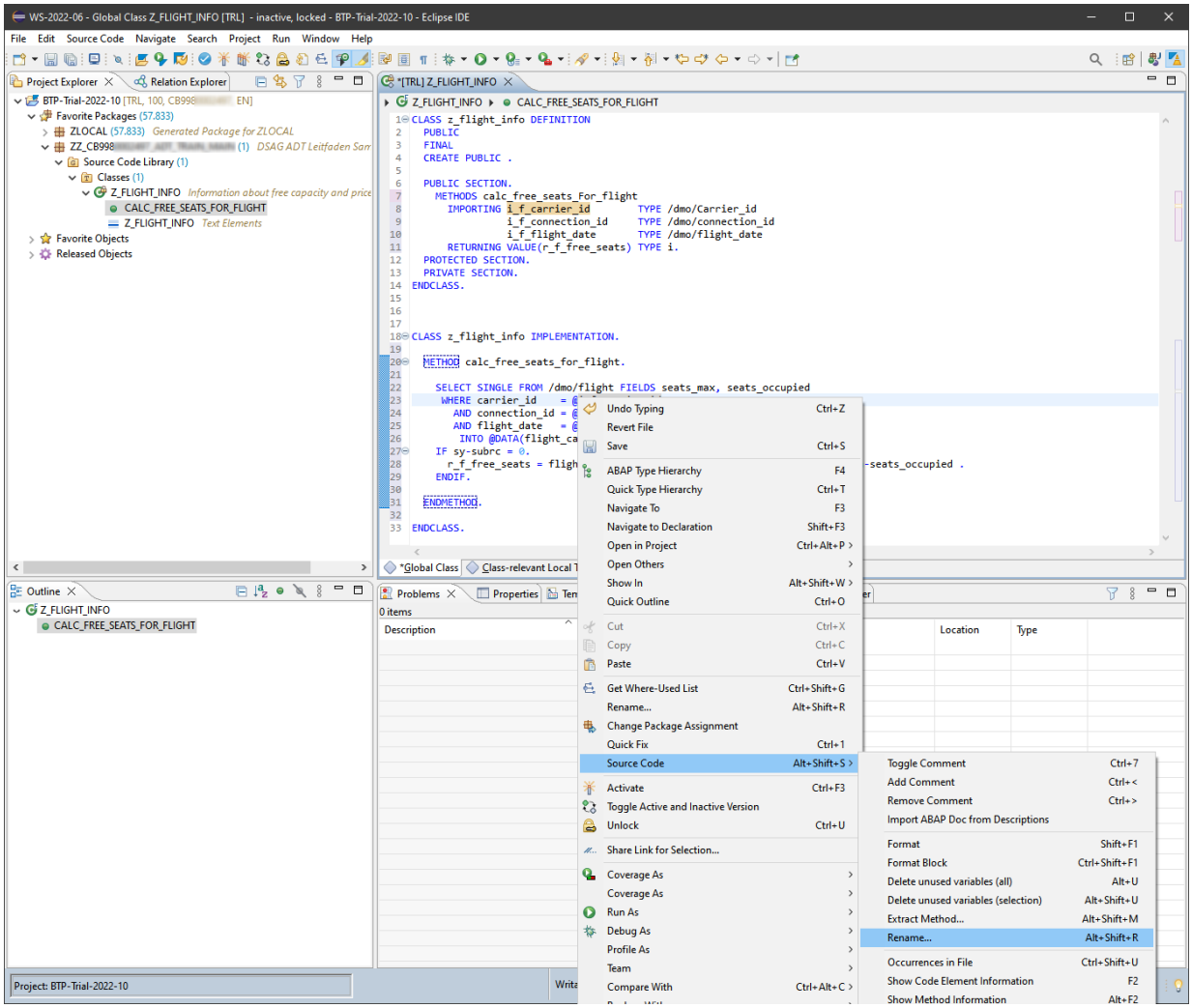


Figure 12 Renaming Method Parameters

First, the code must be saved. If this has not been done, a save prompt will appear, which must be confirmed.

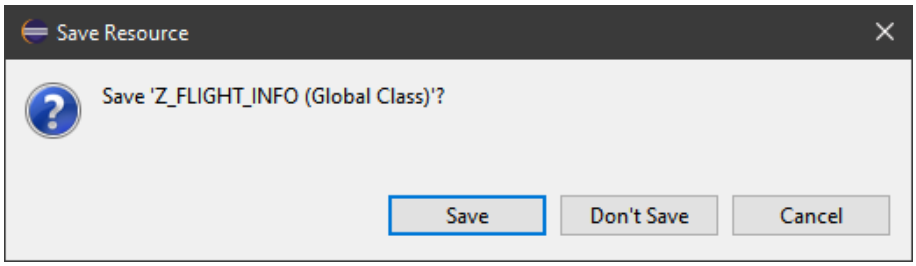


Figure 13 Query Dialog for saving the Code

A dialog box appears for entering the parameter name.

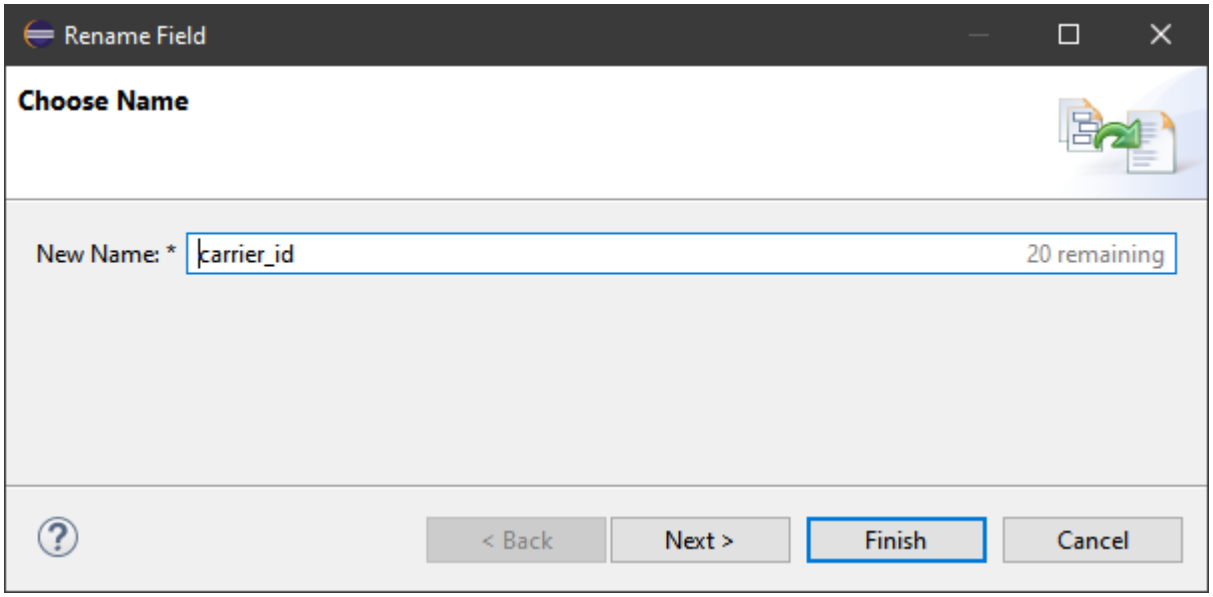


Figure 14 Entering a new Parameter Name

If the object is already assigned to a transport, the renaming can be carried out directly with "Finish".

With "Next" further optional settings, such as the transport to be used and the activation option, can be made.

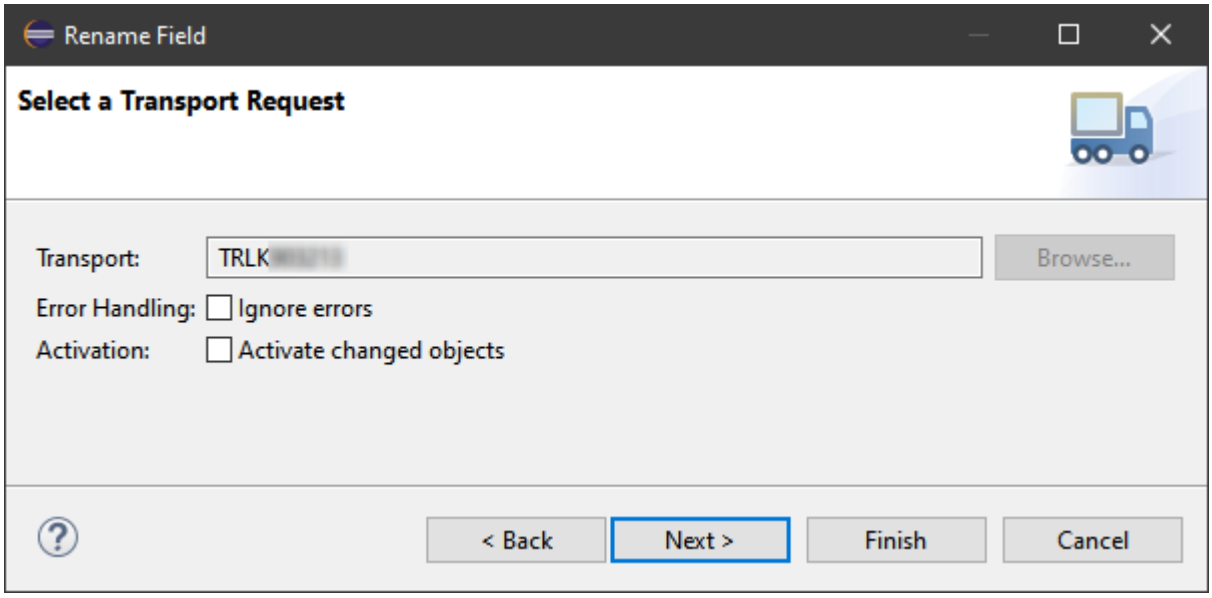


Figure 15 Transport Selection and Options

Before the final execution, a preview of the change can be displayed.

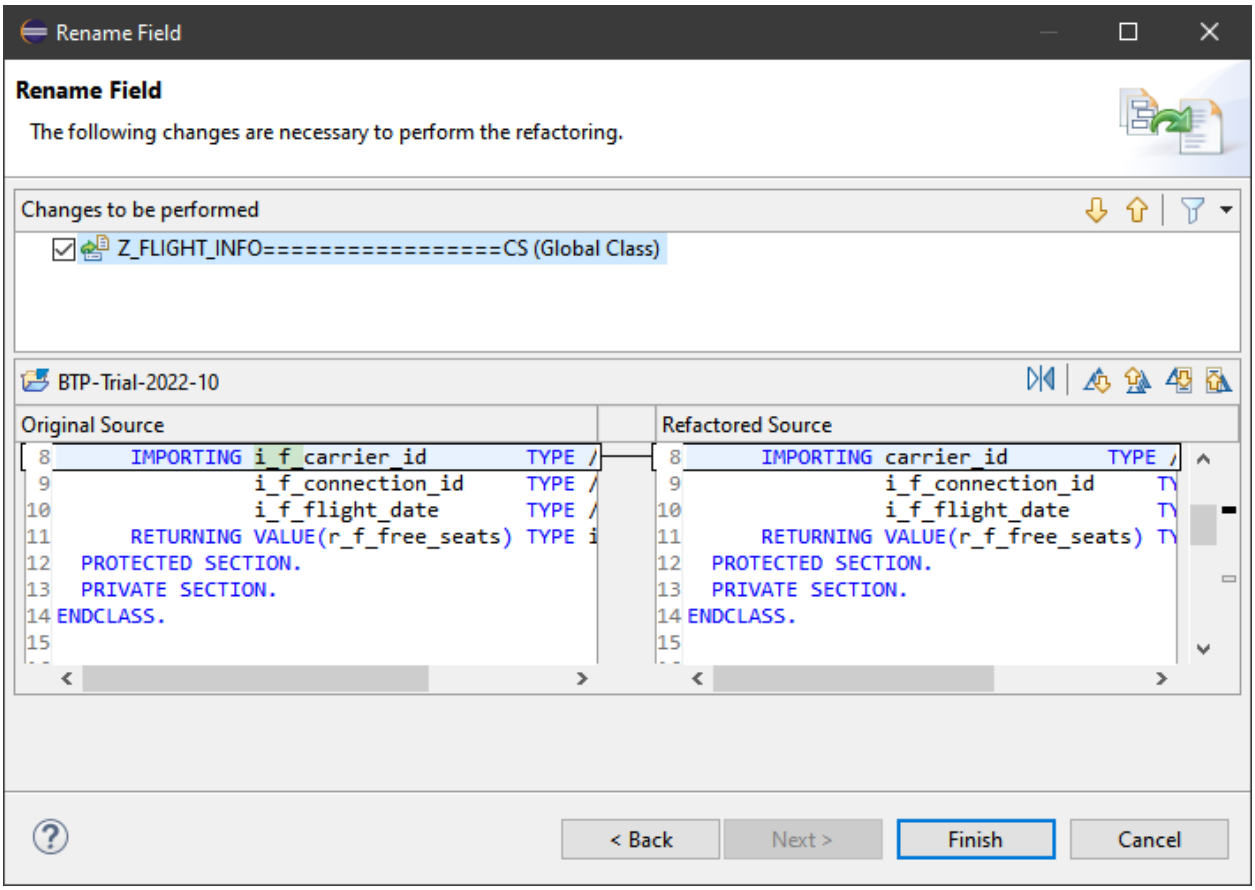


Figure 16 Renaming Preview

After clicking on "Finish", both the parameters in the definition, the usage in the method and the parameter names are changed appropriately at the points where the method is used. The renaming shown here works analogously in the same way for variables, methods, and even class names.

This shows a very strong advantage over the SAP GUI-based development tools, in which a cross-object renaming is not possible automatically.

This is the first step, the basic functions and working methods in ADT are known and can now be applied. The first hurdle has been cleared and the basis for the application of the numerous functions of the ABAP Development Tools, which are explained in detail in the following sections, has been created.

3.2 Features of ADT

The main purpose of the previous section is to get you started and develop with the ABAP development tools in Eclipse. The Functions section is primarily dedicated to describing the numerous features and teaching best practices in dealing with ADT in daily work.

After ADT has been successfully set up and a project, i.e., a connection to a Netweaver On-Premise or an ABAP Cloud Environment in SAP BTP, has been established, development can begin. In the following chapters, we offer an overview of the possibilities that ADT offers in the development of new objects and in the extension/revision ("refactoring") of existing objects.

3.2.1 Overarching features

3.2.1.1 Workspaces

The so-called workspaces serve as the main level of work structuring and storage of the Eclipse and ADT configuration. When you start Eclipse for the first time, you will be asked in which directory the workspaces should be stored.

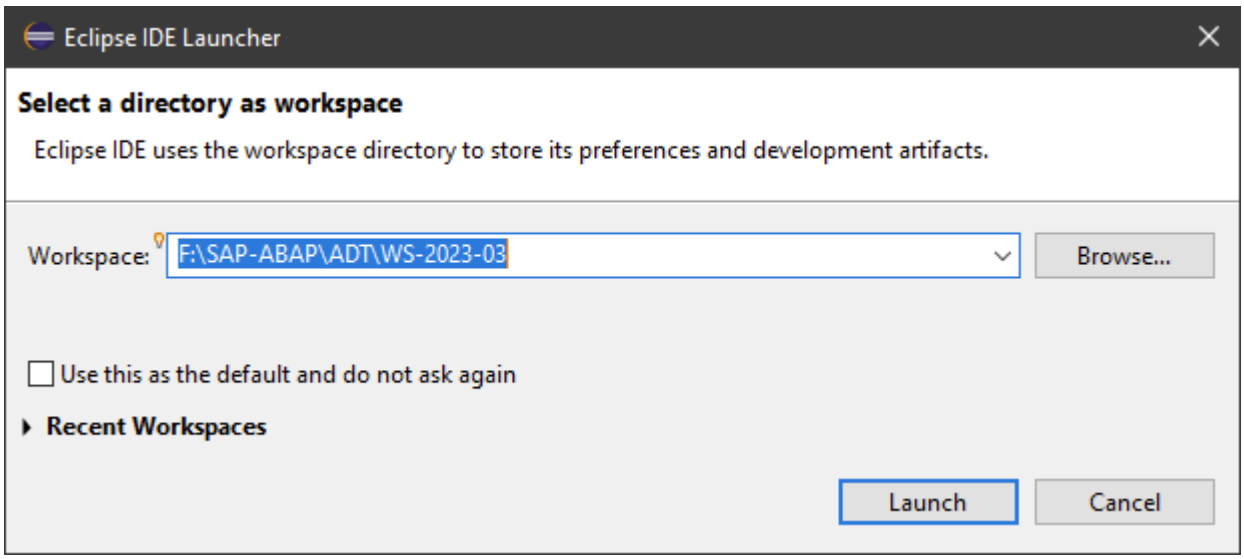


Figure 17 Query of the Workspace Directory

One way in which the directory structure can be designed can be found in Chapter 6 - Best Practices Eclipse Configuration.

Numerous configuration settings are stored in this directory. These include:

- the projects and thus accessible SAP systems,
- which favorite packages are used in the projects,
- which perspectives are used,
- which views and objects are open.

If you can get by with a workspace, you can check the box "Use this as default ..." so that this workspace can be used in the future without being asked. This setting can be changed at any time in the settings.

For most cases, one workspace is sufficient. If you work in several projects with different system lines or with different customers, the workspaces can help to keep the system environment used clear and to have the most efficient configuration available for every situation.

If there is a need to create or change a new workspace,

File → Switch Workspace

either select a previously open workspace from the list or use

File → Switch Workspace → Other

the workspace dialogue.

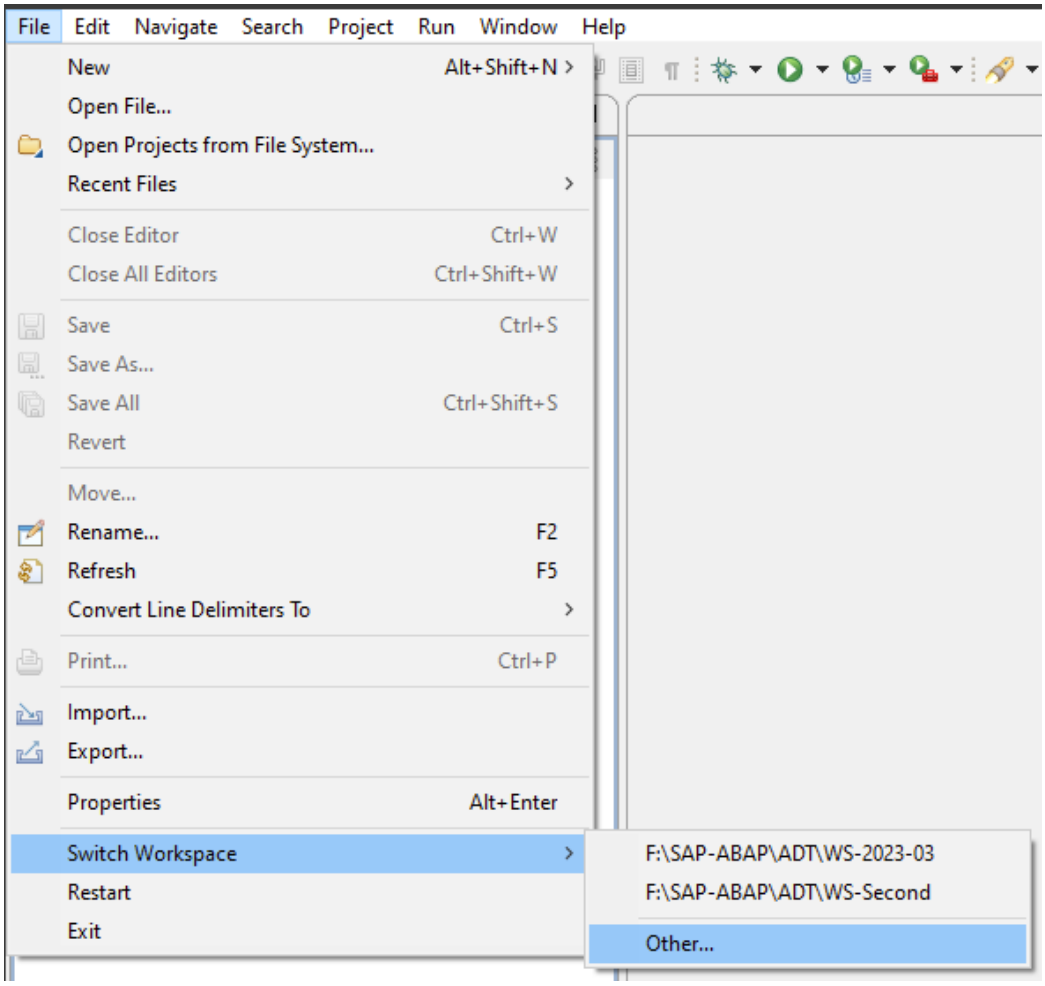


Figure 18 Switching the Workspace

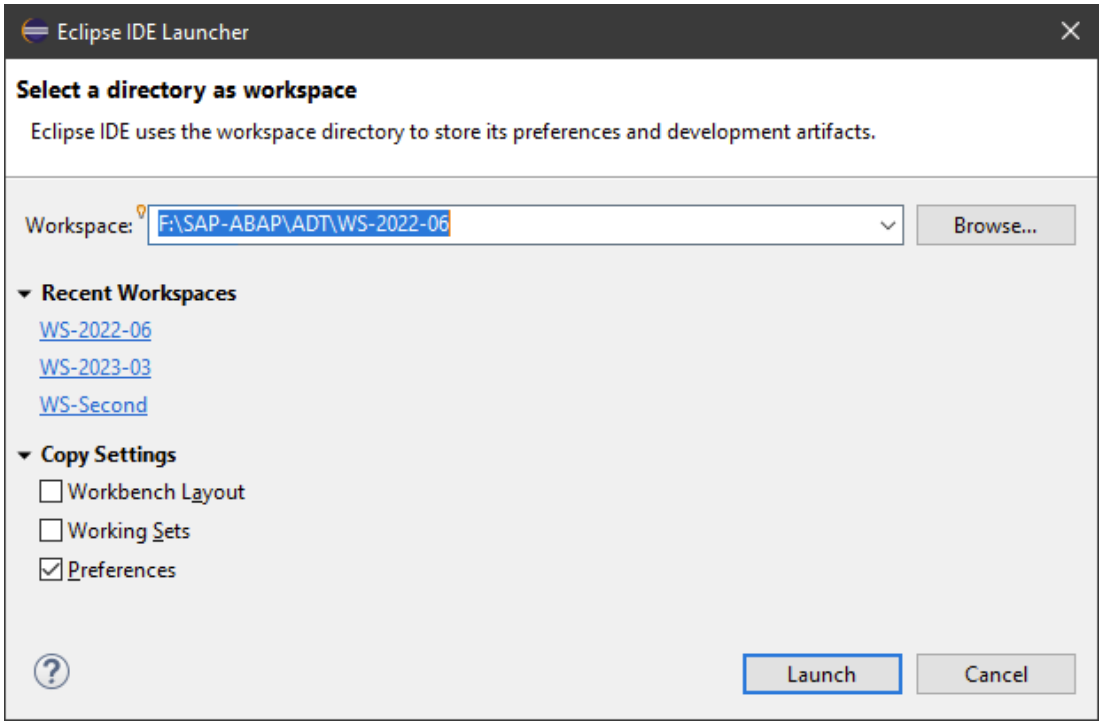


Figure 19 The Workspace Dialog

If you want to create a new workspace, enter a new name in the Workspace field. This will create a new workspace with the current settings after clicking on Launch. You define which settings are to be transferred from the source workspace using the Copy Settings. Alternatively, you can access one of the displayed workspaces under Recent Workspaces directly by clicking on the blue links.

3.2.1.2 Project Explorer

The Project Explorer is a central component for navigation in the integrated systems. As soon as you integrate a new system as an "ABAP Project", it appears in the list. If you log in to a system and expand it, you will receive further information about shared objects, favorites, inactive objects, etc., depending on the system. These repository trees can be freely defined and customized. At the package level, the view behaves like the SE80 and maps object hierarchies through which you can navigate.

3.2.1.3 Favorite Packages

For day-to-day work, it is advisable to add packages as favorites in which you work regularly, or which fall within your personal responsibility. This gives you a good overview and allows you to quickly find "your" objects.

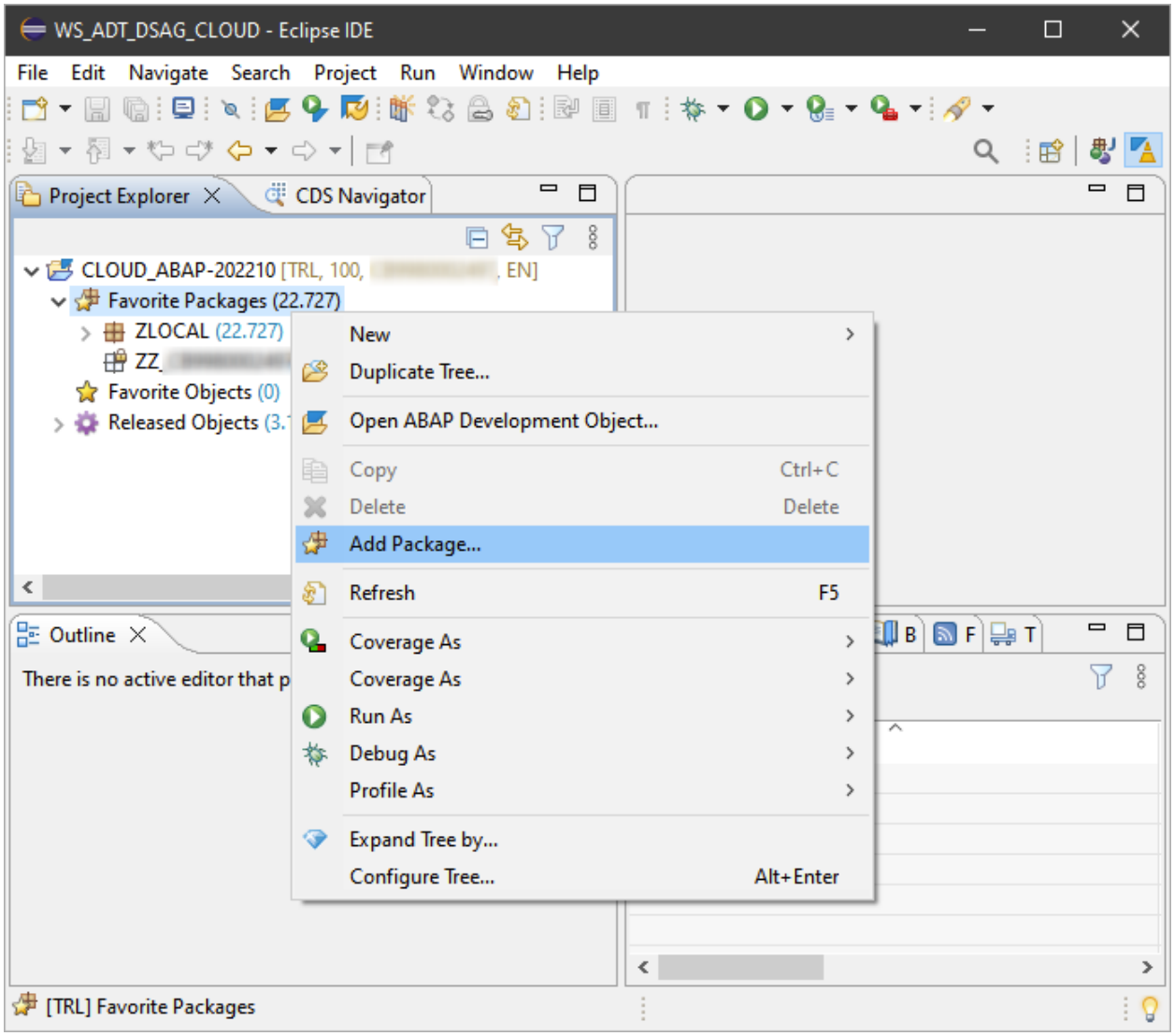


Figure 20 Adding Packages to Favorites

For storing and organizing favorites at object level, the "ABAP Favorites" plug-in can be recommended, which can be installed in Eclipse via the plug-in installation mechanism, see Chapter 7 - Plug-ins.

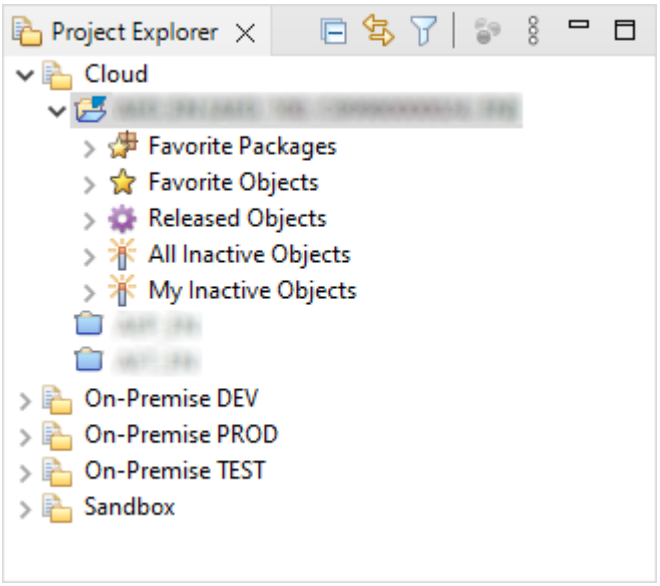


Figure 21 Project Explorer Detail Screen with Button Bar

In the button bar there are still some useful functions with which you can make settings on the view:

- Double arrow (Link with Editor) - Objects that are in focus in the editor are displayed in the Project Explorer by loading the hierarchy.
- Three dots (View Menu) - Further settings of the views, for example to create working sets. These are folders that can be used to group systems (see screenshot above).

3.2.1.4 Working Sets

If you are a developer working with multiple system lines, we recommend using the Working Sets. These make it possible to group projects in Eclipse and thus clearly display several systems.

The three-dot icon in the upper right corner of the Project Explorer offers various setting options. Among other things, the working sets can be created and configured here.

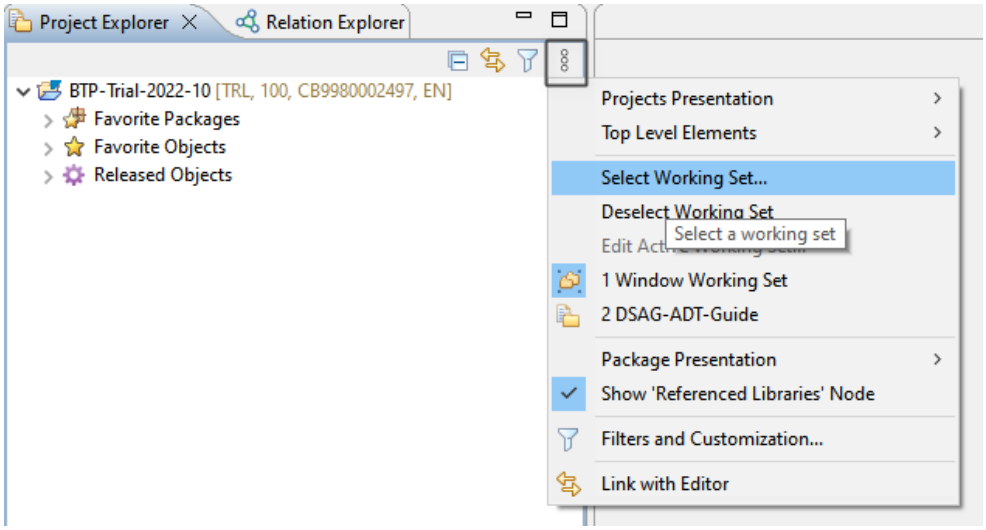


Figure 22 Working Sets Settings

The dialogue can now be used to create working sets (New) and make assignments (Edit).

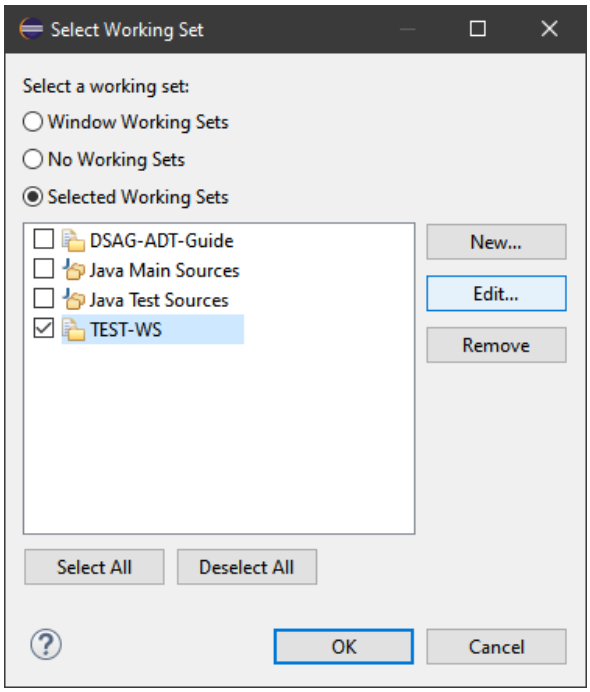


Figure 23 Creation and Processing of the Working Sets

The edit button can be used to assign the desired projects to the working set in the subsequent dialogue.

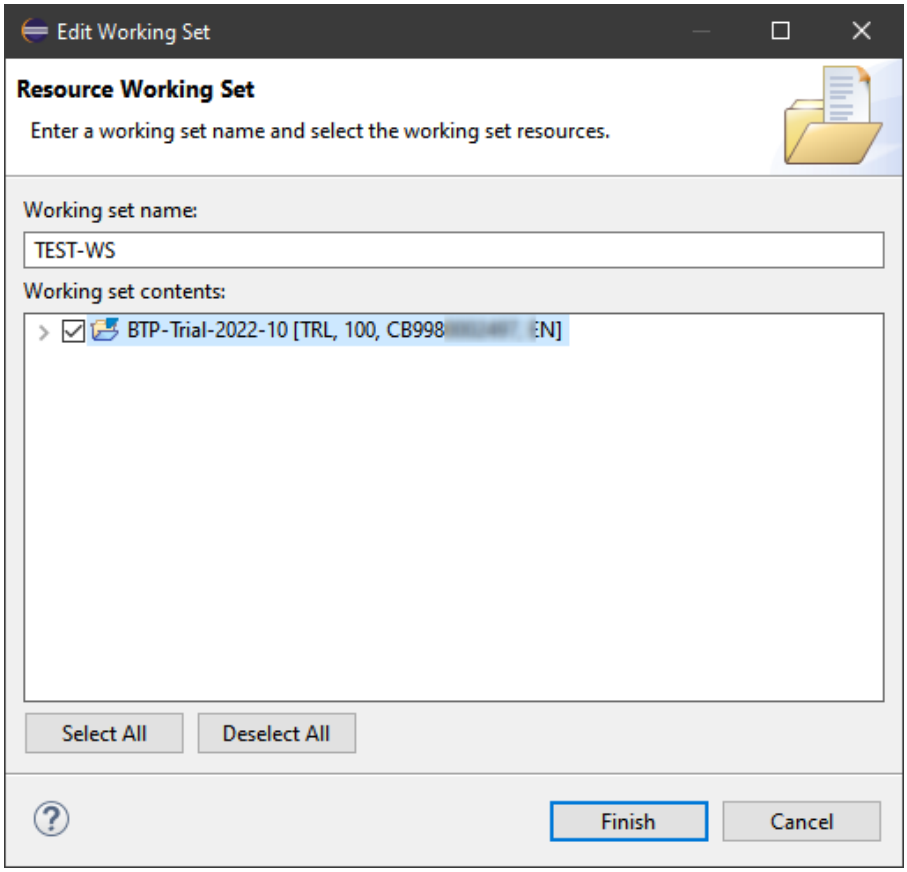


Figure 24 Assignment of Project to Working Set

This function enables a clear structuring of the systems according to system landscape or, if applicable, according to project or customer. Finally, the display of the Top-Level Elements must be set to Working Sets.

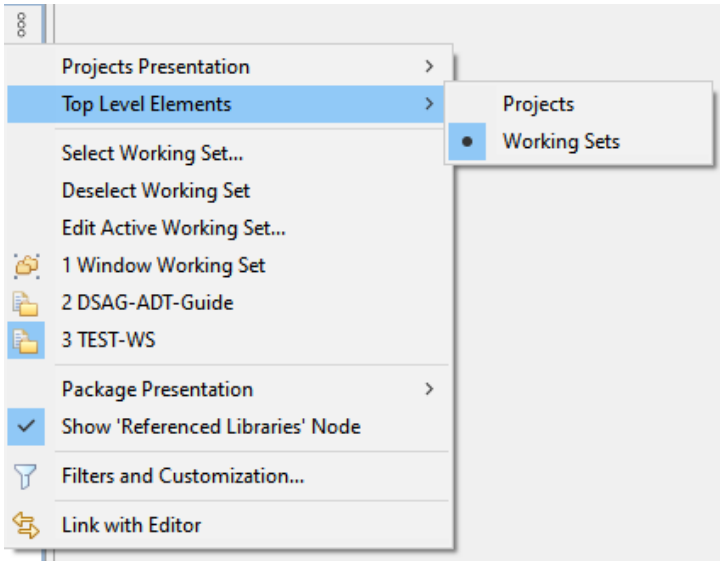


Figure 25 Project Explorer Display Setting

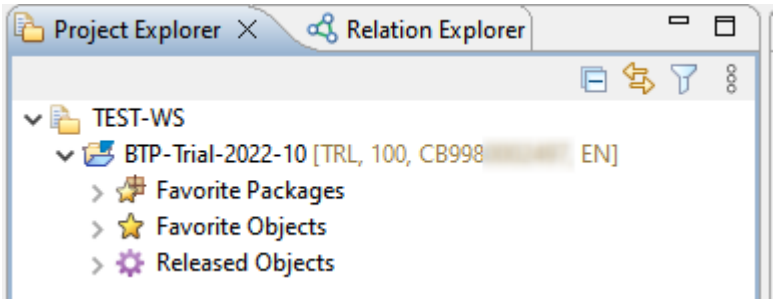


Figure 26 Illustration of Projects in Working Sets

3.2.2 Search and Navigation

The search for objects in Eclipse is a central part of the daily work, as is the navigation between the objects or the forward navigation. In this section, you will learn more about searching and navigating between ABAP objects.

3.2.2.1 Searching for objects

To search for or open an object in the system, you can use the "Open ABAP Development Object" dialogue (accessible via the key combination **`CTRL+SHIFT+A`**).

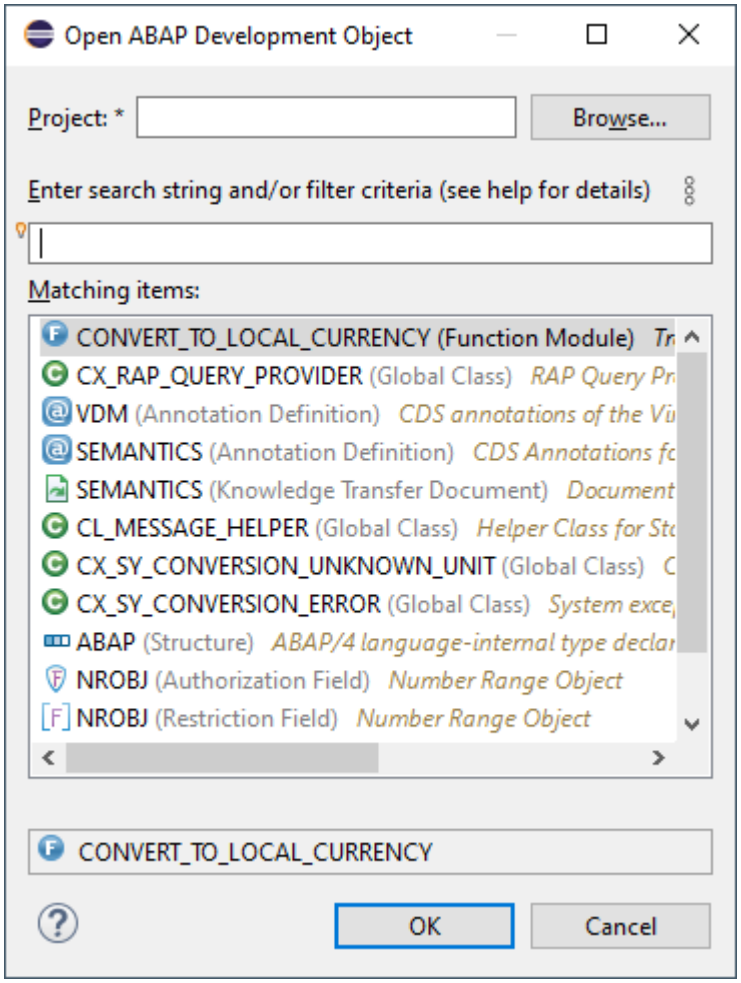


Figure 27 Object Search Dialog

In the dialogue, you have the option of changing the ABAP project in the upper part and thus, deciding on which system you want to search for the object. If nothing has been entered in the search field, you will receive a history of the most recently opened objects. Use the *question mark* at the bottom to get more information, tips, and tricks on how to use the search. Detailed information on the object search can be found in the User Guide.

3.2.2.2 Filter Objects

In the object search, you now have the option of working with search strings and patterns to further narrow down the result set. The field supports "Content Assist" (**CTRL+SPACEBAR**) to use further restrictions and filters. A simple search might look like this:

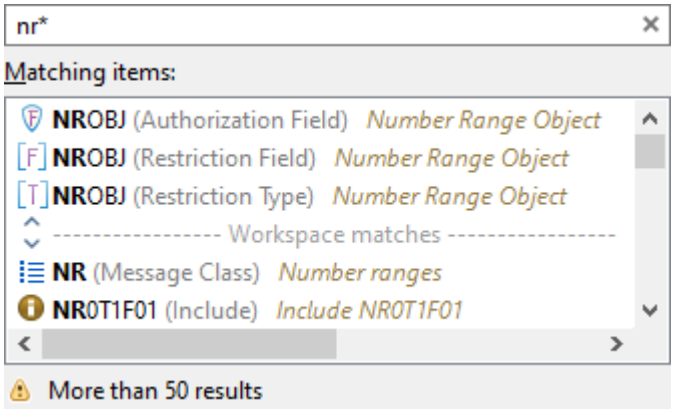


Figure 28 Results of the Search

However, more than 50 results are displayed (the default) and most likely the desired result is not included in the result set. In this case, you can go to the "Content Assist" to get more options for filtering.

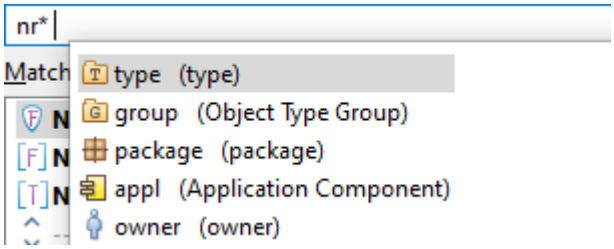


Figure 29 Display of additional Search Options

For example, if you want to restrict to table types, then you would restrict further by type (TYPE). The "Content Assist" also suggests the different types of objects, so that you can also find the table type (TTYP).

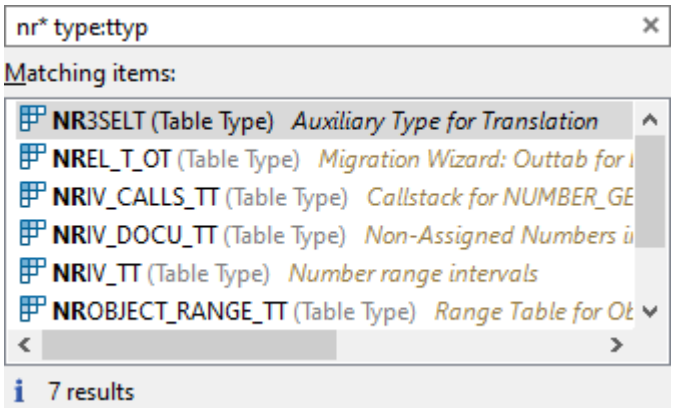


Figure 30 Result with Object and Type Filter

The other filters and types can be freely combined with each other to find the desired target quantity or object. By double-clicking on the desired entry, the object is then displayed in the editor.

To speed up the search, it is recommended to specify the type of the desired object, otherwise the search – compared to the usual speed in the SE80/SE11/etc. – takes a very long time.

3.2.2.3 Navigation

In the ABAP Workbench, you can navigate to the next object by double-clicking the corresponding expression in the source code. In Eclipse, only the source code is selected here. There are three ways to trigger forward navigation:

- Place the cursor on the object, press F3
- Hold down the **CTRL** key and click on the object
- A clickable link is offered in the interface (e.g. data element → domain)

The object opens in a new tab within the editor, the source object remains open, and you can easily navigate between the most recently modified objects using the keyboard shortcuts:

- **ALT+Right Arrow**: forward
- **ALT+Left Arrow**: backward

This can also be done analogously with the arrow keys in the area of the pushbutton bar. In the area of the pushbutton bar, there are also various options to get to the last tab used (**ALT+arrow left**).

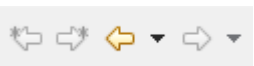


Figure 31 Navigations Icons

3.2.2.4 Displaying the ABAP Repository Tree

After you have found an object, in many cases you will want to continue working or researching in this package. To do this, you can load the object tree by activating the double arrow ("Link with Editor“) in the Project Explorer.

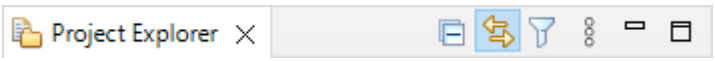


Figure 32 Actions for the Project Explorer

In this case, the package hierarchy is loaded to the object focused on in the editor. You can then navigate to the other objects and structures in the Project Explorer.

The work with the repository tree is described in detail in the user guide.

3.2.3 ABAP Editor

The ABAP Editor is a simple text editor that enables the purely textual creation of ABAP artifacts. The context function can be used to call up the most important functions such as quick fixes, refactoring functions, and formatting functions. Getting started with the ABAP Editor is described in more detail in the section Creating a Class in Text Mode.

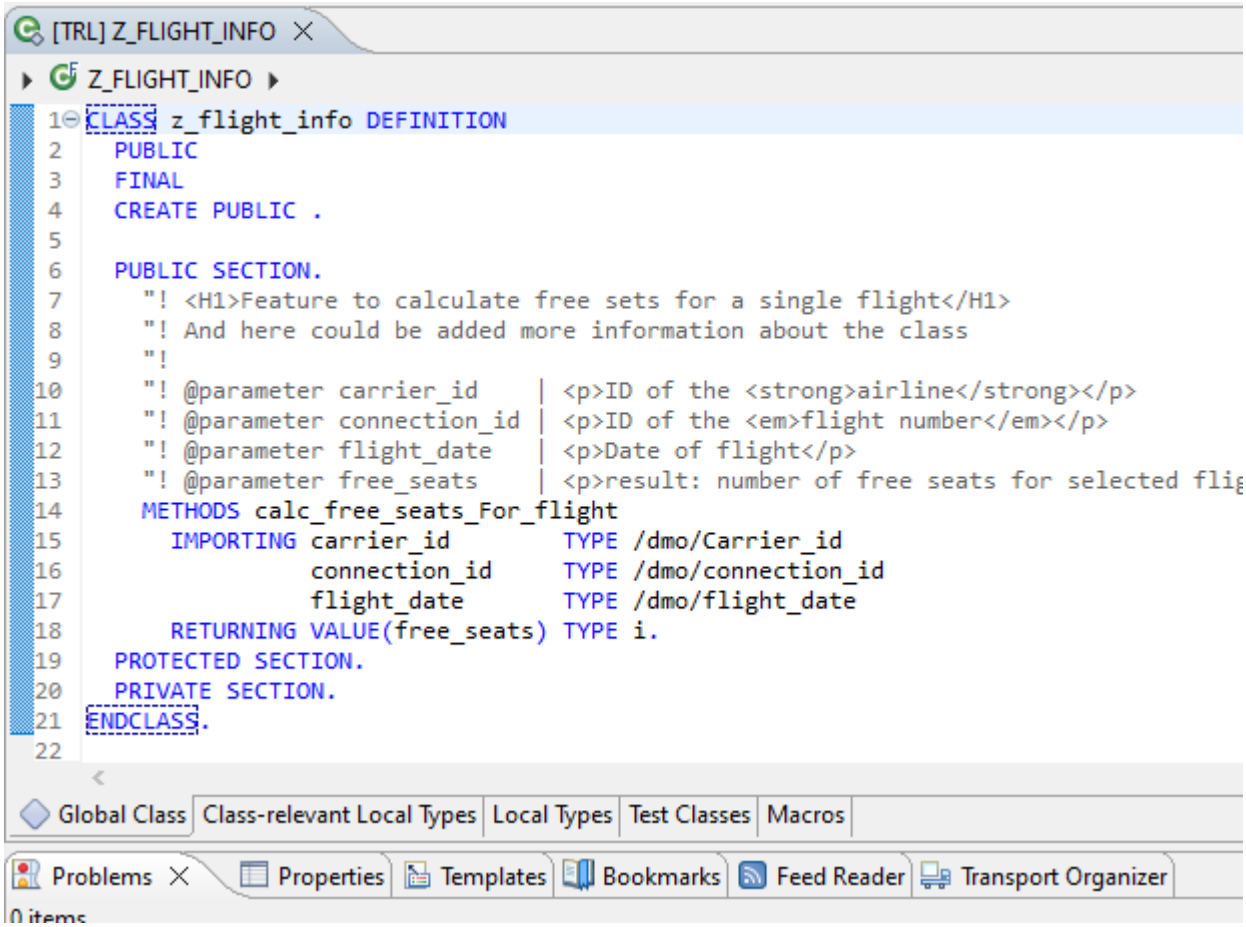


Figure 33 ABAP Editor - Main Window

3.2.3.1 Element Info

By positioning the cursor on an object and the shortcut **F2**, a pop-up with additional information appears. Here is an example of a method and a data element:

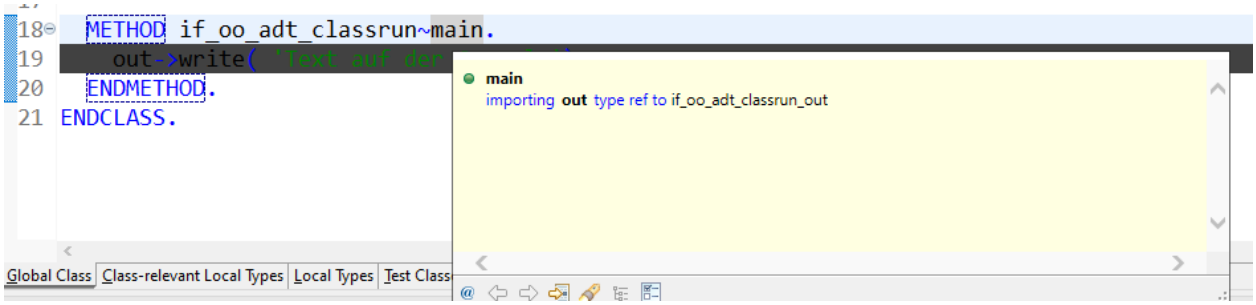


Figure 34 Element Info for a Method

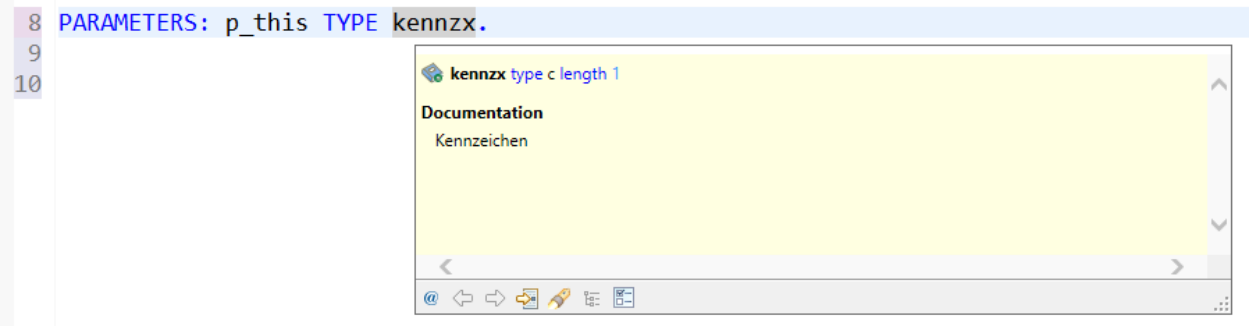


Figure 35 Element Info for a Data Element

In the Element Info , details are thus directly visible for which you still had to navigate the system in the old world. In addition, you can navigate further in the Element Info to view details of deeper levels, e.g. the stored domain, etc.

The ABAP Element Info also exists as a stand-alone view. This can be activated via the Windows → Show View menu → ABAP Element Info. After that, "Link with Selection" automatically displays the Info element and the documentation from ABAP Doc each time a relevant development object is clicked:

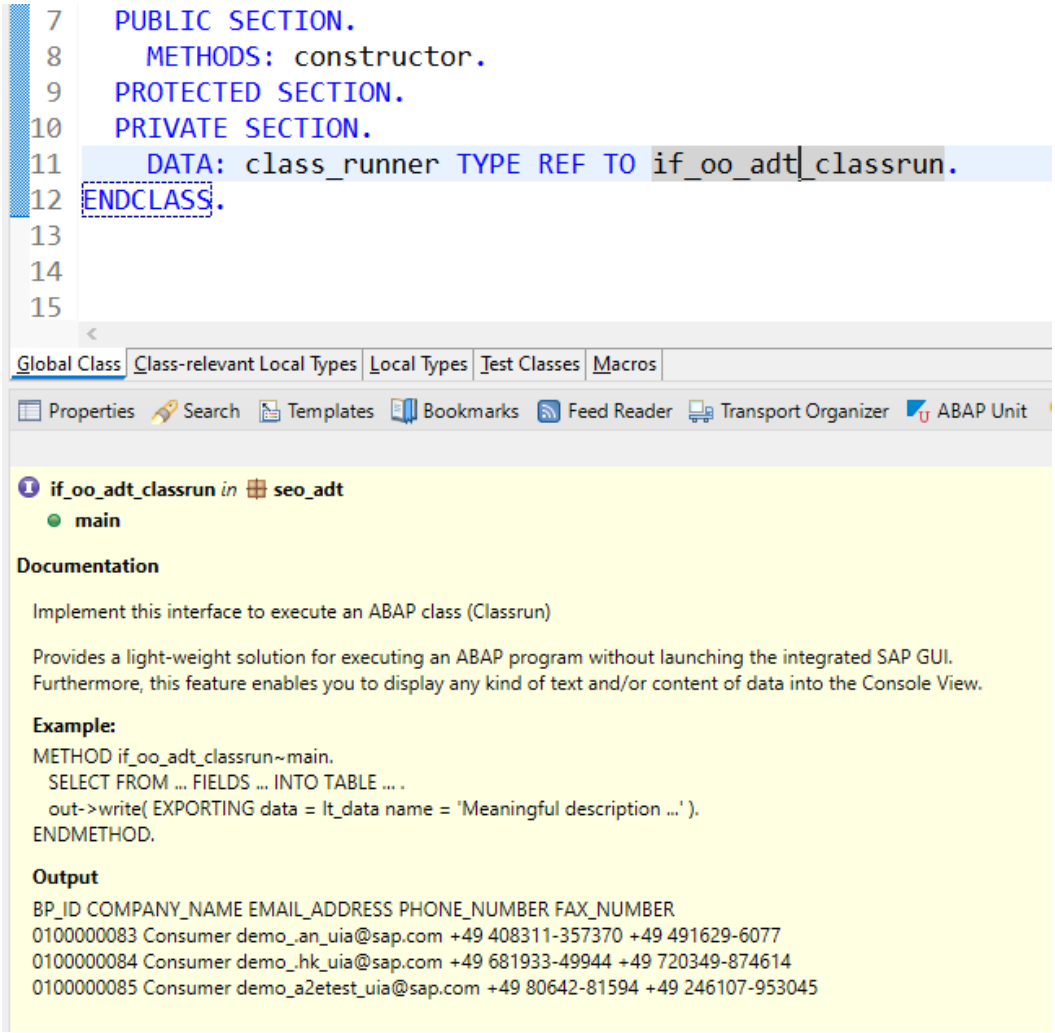


Figure 36 Display of the Element Info after Selecting the Object

Via "Pin this view“ the information is displayed permanently, even if another element is clicked or the element info is opened by **pressing F2** for another development object.

3.2.3.2 Source Code Formatting with the ABAP Formatter

In the SAP GUI, the tool for formatting the source code is called Pretty Printer. The counterpart in ADT is the ABAP Formatter. It can be accessed either via the keyboard shortcut

SHIFT+F1

or via the context menu in the source code.

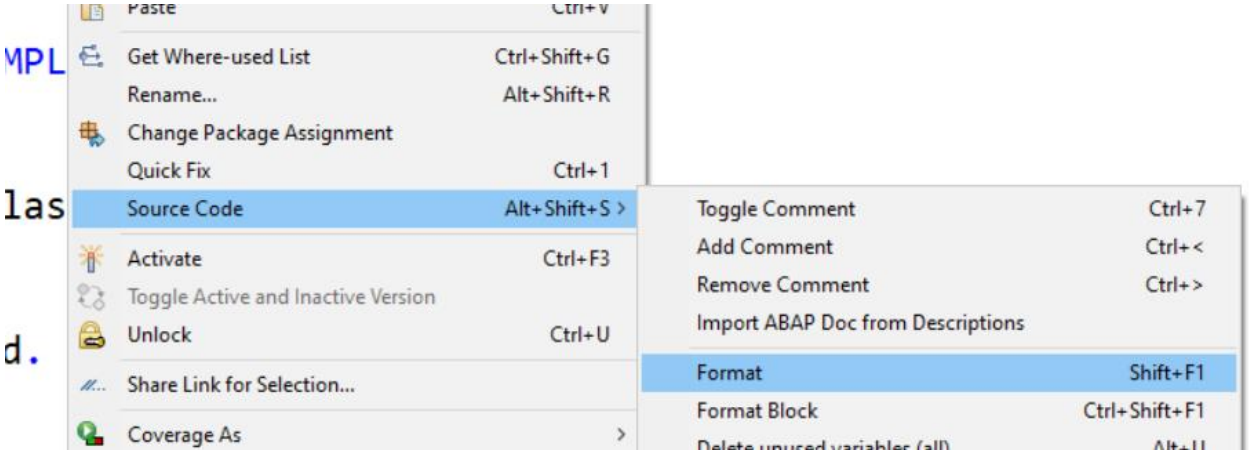


Figure 37 Formatting Context Menu

For the ABAP Formatter to be able to do its job, it must be configured in advance. Analogous to the Pretty Printer settings in the SAP GUI, you specify whether indentations are to be made and how the upper and lower case is to be formatted. This is done separately for each ABAP project.

3.2.3.2.1 Setting up the ABAP Formatter

If the settings have not yet been made, a pop-up window will appear with the message that this must be done first. There is also a link to the settings included in this pop-up window. Alternatively, you can also use them directly via the context menu entry *Properties* of the corresponding project. In the image below, you can see the location in the settings.

The settings correspond to those in the SAP GUI. If you try out the individual options, you will see the respective result in the preview window. A new feature is the ability to retain camel case identifiers. This is particularly practical in connection with the CDS views, as they are consistently used in the virtual data model of SAP (VDM).

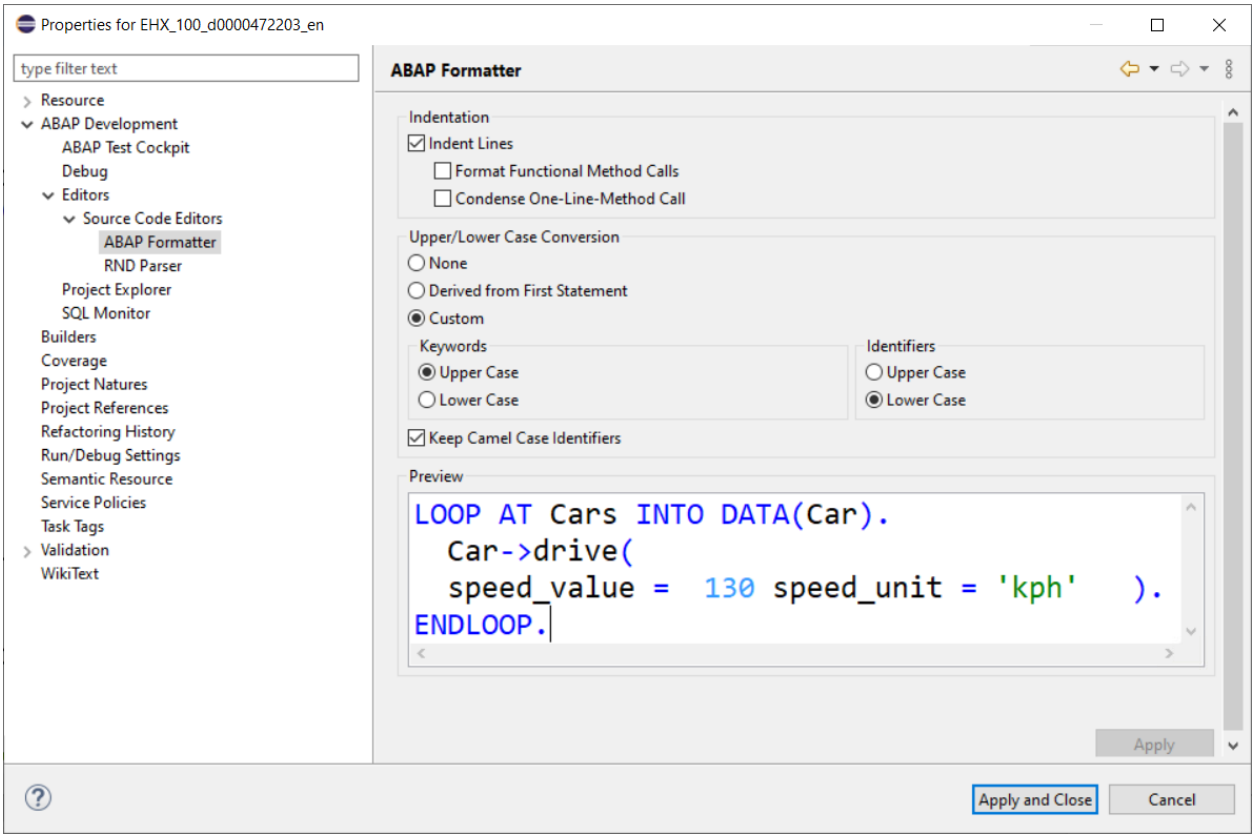


Figure 38 Upper and Lower Case in ABAP Formatter

3.2.3.3 Quick Fixes

Quick fixes are automated solutions for common problems in the context of software development with ABAP. Quick fixes are offered natively by ADT, but can also be extended by various plug-ins. The use due to the large number of available Quick Fixes makes working with ADT much more efficient than with the SE80. In addition, the risk of errors due to manual adjustments is reduced. ADT make the adjustments automatically and always identically.

Quick fixes provide functionality for two areas:

- Automatic creation of non-existent objects (e.g. method implementations)
- Automatic modification of existing objects without changing the functionality (so-called refactoring, e.g. "extract method")

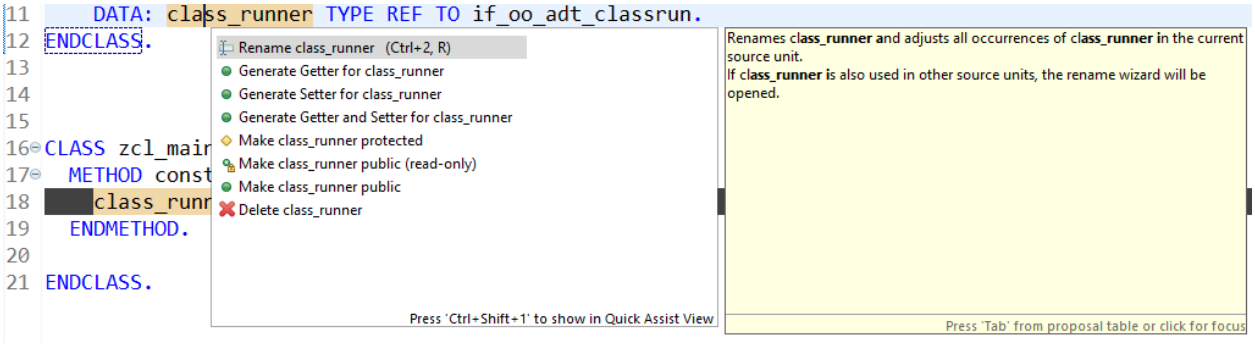


Figure 39 Display of Refactoring Options

Due to the large number of quick fixes and the constant changes in this area, the individual quick fixes are not described here. An overview can be found in the documentation.

3.2.4 Other object types

3.2.4.1 Programs and function groups

Programs and function groups are displayed in the *Source Code Library* folder in the navigation of the Project Explorer. Under the corresponding object types, all components are displayed according to the SAP GUI.

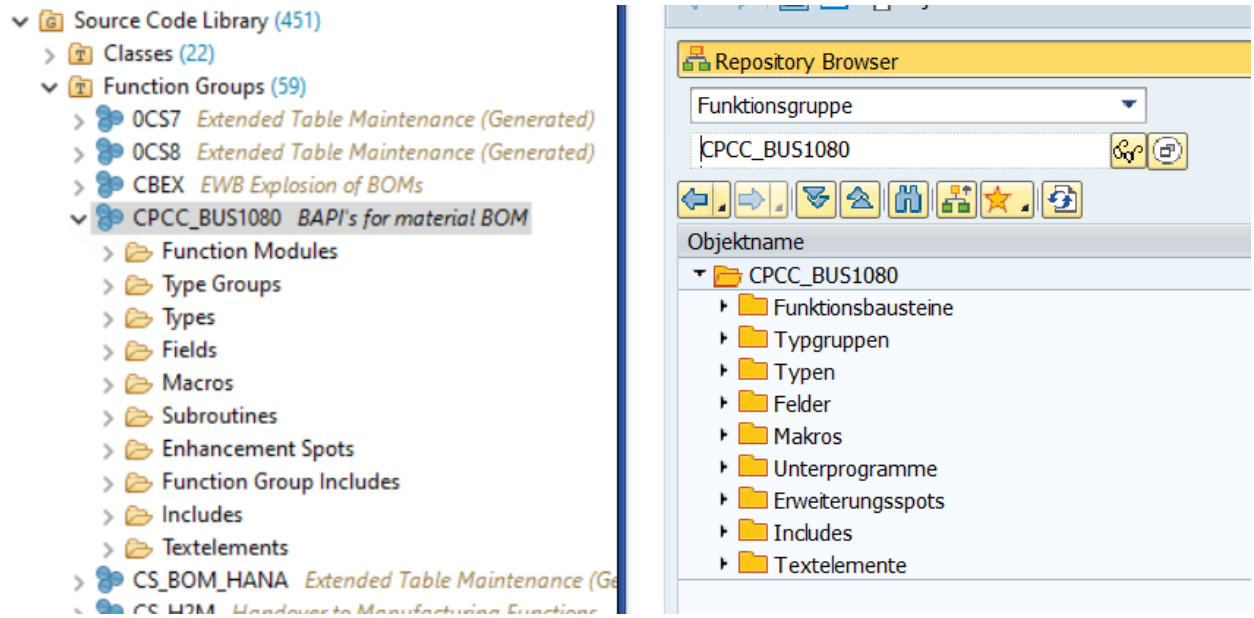


Figure 40 Comparison of Function Groups in the Project Explorer of ADT and in the SE80

The same source code editor is used as for ABAP classes. This means that all features from ABAP Formatter to Language Help are mostly identical.

3.2.5 ABAP Views

3.2.5.1 Outline

The Outline View provides information about the currently focused development object and resolves variables, local classes, types, etc. The view can be compared with the SE80, but only shows you the current context of the object. In the screenshot you can see a class, below with the corresponding methods and two private attributes of the class. By clicking on an entry, you navigate to the corresponding location in the source code.

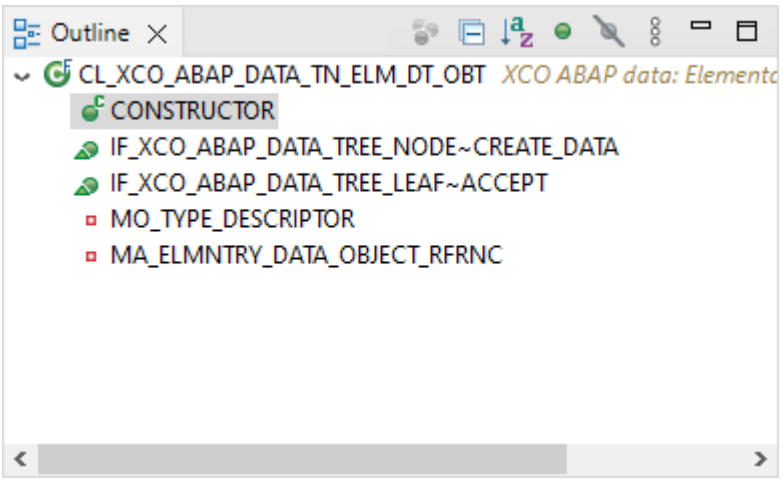


Figure 41 Display of Properties in the Outlines

In the button bar there are other different functions for the view:

- "Sort" – sorting of entries by alphabet or by type and alphabet
- "Hide Non-Public Members" – Only attributes and methods that can be used from the outside are displayed (topic visibility)

Note: In Chapter 7 - Plug-ins you will find more information about the plug-in "Classic Outlines", which extends the outlines.

3.2.5.2 Problems

The Problems View is probably one of the most important views. It shows information about errors within development objects in the form of a list. The view updates automatically when new errors are added, or existing ones are corrected. A continuous "checking" of the source code is not necessary.

In the standard configuration, all errors in the user's own open objects, not just the object currently in progress, are displayed across all systems.

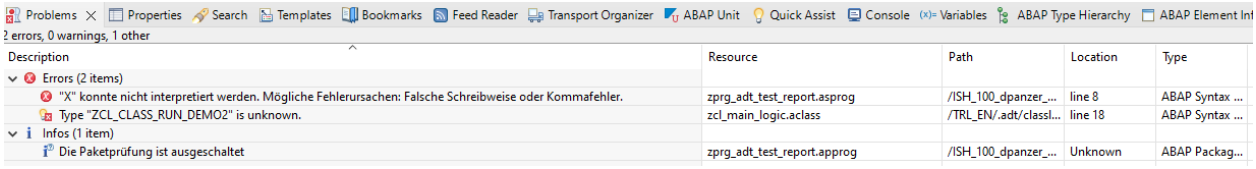


Figure 42 Displaying the Messages in the Problems View

By double-clicking you can navigate to the corresponding place in the source code.

The button can be used to further configure the view:

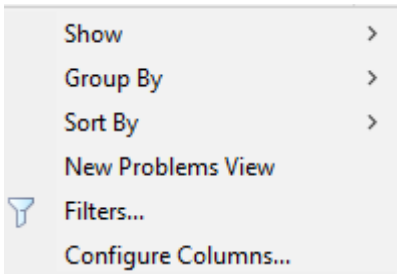


Figure 43 Display of the Options of the View

- Under "Show" you can configure which errors/warnings are displayed in the Problem View, e.g. only those of the currently in progress development object or all of them.
- "Group by" allows grouping according to various criteria, usually grouped by "Severity", i.e. error/warning/info.
- "Sort By" can be used to change the order of the display.
- "New Problems View" duplicates the view.

- "Filters" allows you to filter the list of results in detail down to the development object type.
- "Configure Columns" allows you to show and hide columns and change the order of the columns.

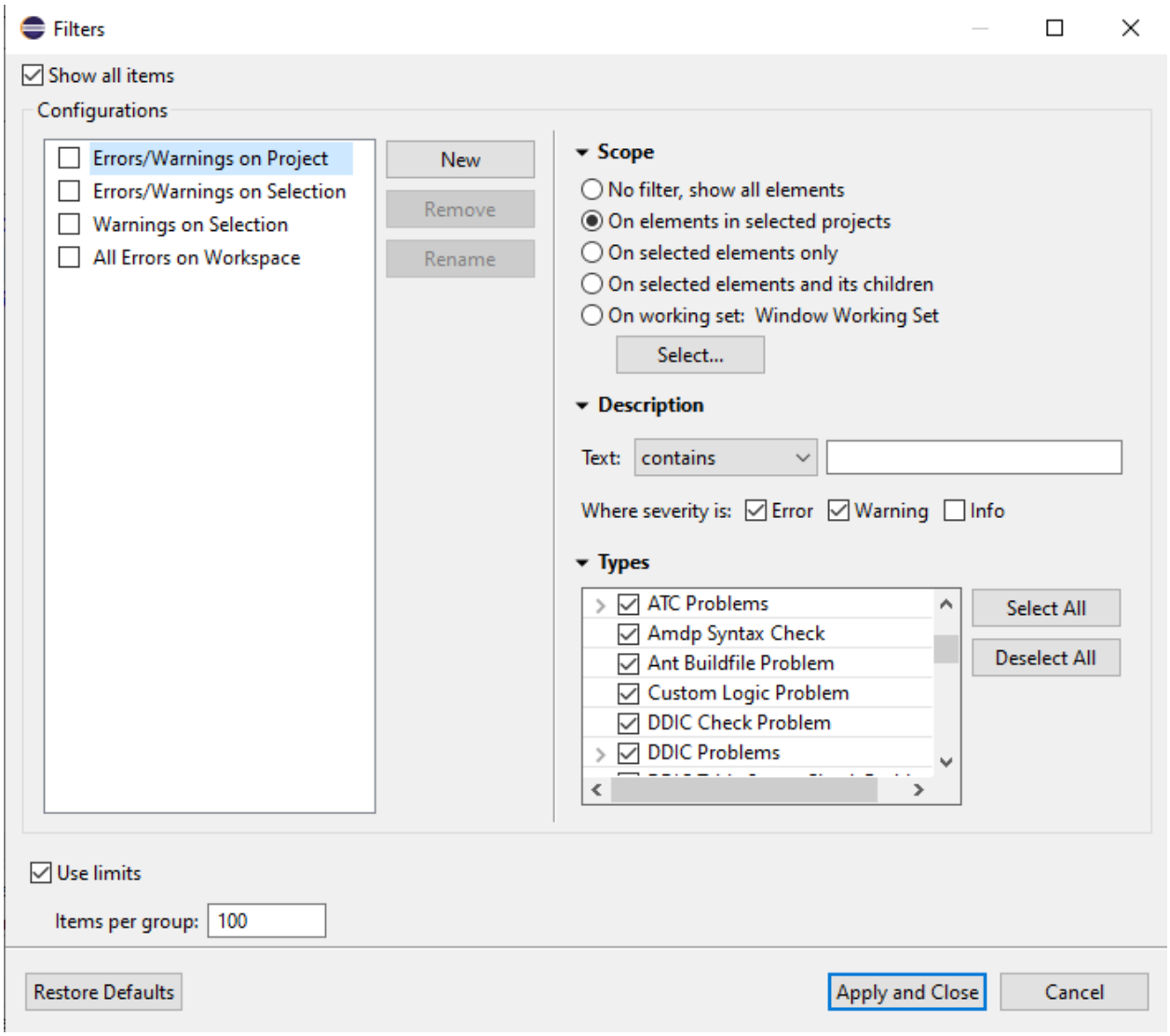


Figure 44 Configuration of the Displayed Points

3.2.5.3 Properties

The Properties View collects information that is displayed in the classic SAP GUI under "Properties". These include, for example:

- State development object (active/inactive)
- Creation and modification information

- Package Mapping

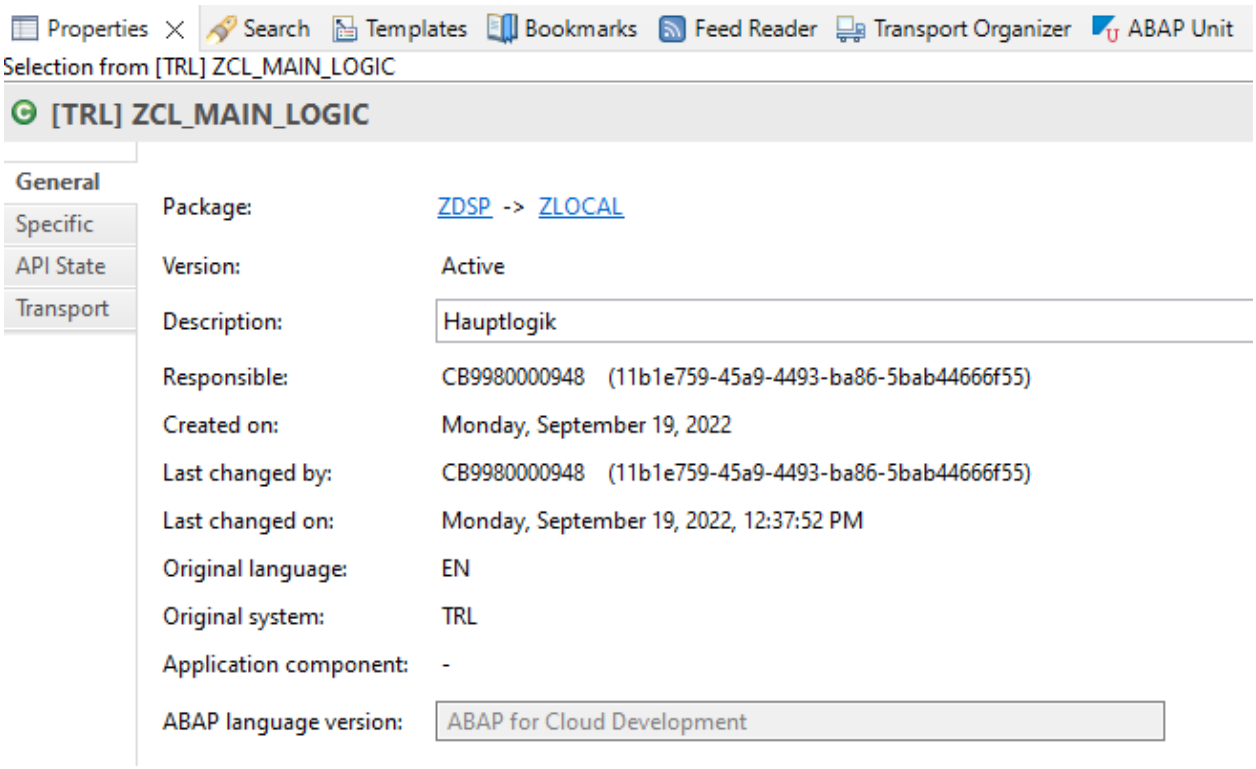


Figure 45 Properties View

In the "Specific" area, object-specific properties are displayed and can be partially changed there. For classes, for example, the fixed-point arithmetic can be set here.

In the Transport section, a history of the transport orders in which the object is contained is listed.

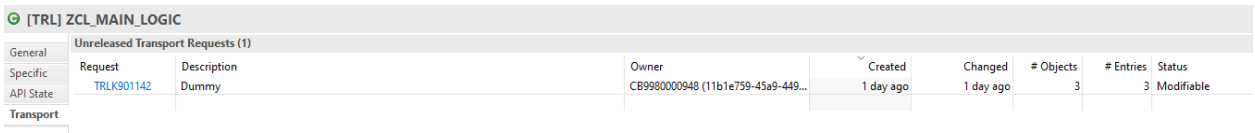


Figure 46 History of Transports

The context menu can be used to branch to the transport request for further analysis.

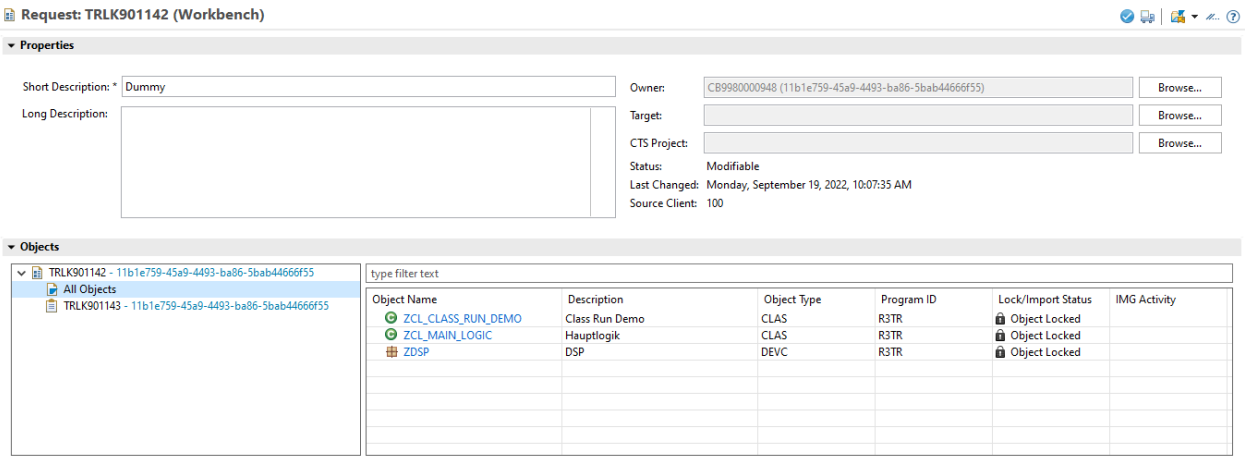


Figure 47 Transport View

Depending on the type of open development object, there are additional areas that contain specific information about the respective object.

In order to view several objects in parallel, several properties views can also be created via the button.

The Properties View has several advantages compared to classic SAP GUI development:

- It is (usually) constantly displayed and available, so that there is no need to navigate time-consumingly.
- It aggregates information that was previously only identifiable via multiple transactions/tabs/tabs.

3.2.5.4 Templates

Code templates are ready-made samples of source code that can be implemented in any way in an application. These patterns reflect static source code and have dynamic elements in the form of variables. In the standard delivery of ADT, some templates are delivered.

TEMPLATE VIEW

Templates are made available via a separate view (Window → Show View → Templates) and can also be adjusted via the settings (General → ABAP Development → Editors → Source Code Editors → ABAP Templates).

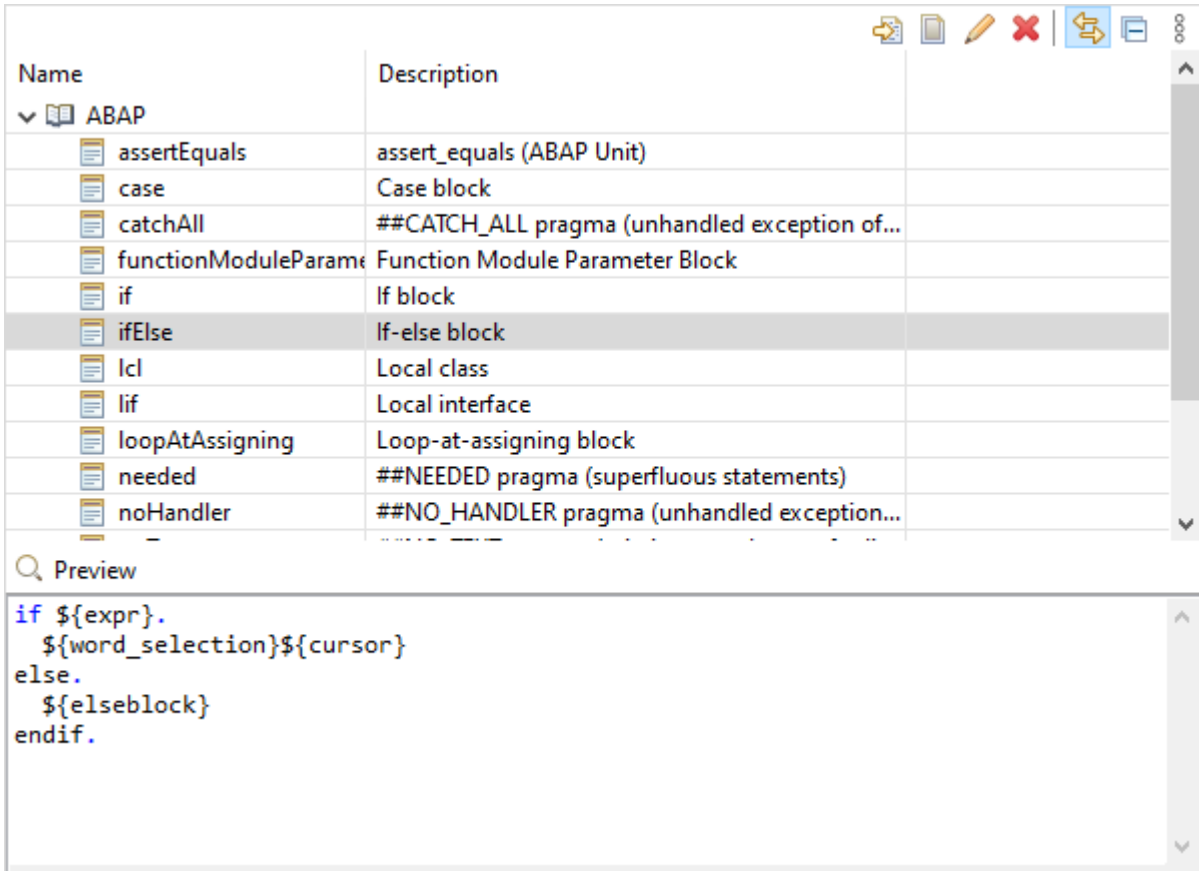


Figure 48 Template View Browser

The view consists of a button bar in the upper area, a list of code templates and a preview of the template in the lower part.

Use Template

The template can be used directly in the source code. Start typing the name and select the appropriate template with the help of the "Content Assist" (here are the first two entries).

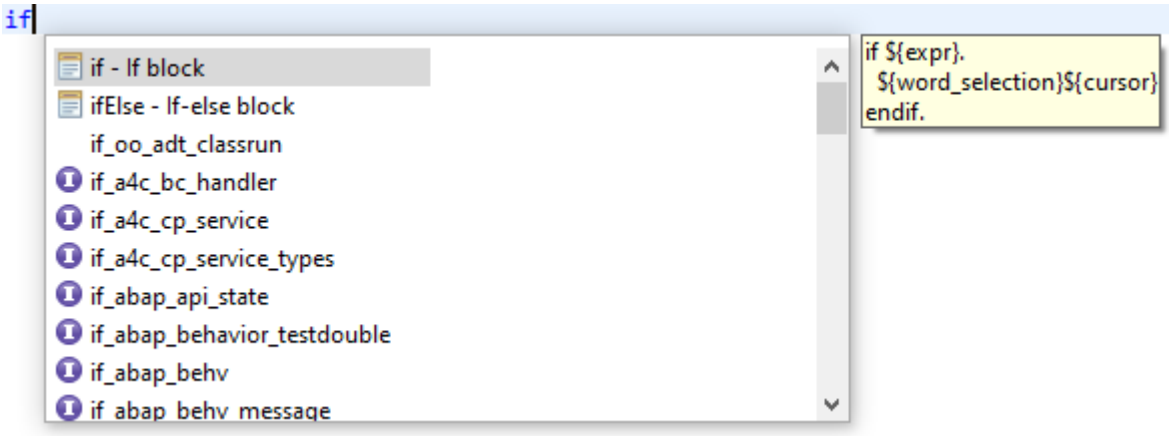


Figure 49 Selection of Templates in Content Assist

The entire template is inserted, and you can start filling in the placeholders (variables). Use the tab to jump back and forth between the individual placeholders.

Frequently used templates for use are, for example:

- **lcl** - plant of a local class
- **testClass** - Creation of a test class
- **functionModuleParameter** - Example interface for function modules

The creation of your own templates is very well suited for saving development effort for repetitive tasks or similar code sections. Furthermore, they can be helpful in training if you want to insert larger sections of code without copy-and-paste.

Create Template

In principle, a template can be defined with all language commands used in ABAP (source code, comments). Variable parts of the template can be tagged with placeholders ("${placeholder}"). Variables are also available for deriving context-specific information. These are available for the following scenarios:

- Name of the object
- Name of the package
- ID des Systems
- User, date and time
- Current year
- Cursor position after insertion

Placeholders with the same name are always uniformly adjusted after insertion (e.g. the name of the class).

AVAILABILITY OF TEMPLATES

Templates are available within an Eclipse workspace, but unlike your own patterns, they are available across systems. Templates can be imported and exported via the settings in order to exchange them among colleagues/employees. It is not possible to store templates centrally for all developers.

3.2.5.5 Where-Used-List

The where-used list finds all static uses of a development object in the source code of the current project. The where-used list can be found using the key combination

STRG+SHIFT+G (Get-Where-Used-List)

attainable. The result is displayed in the "Search" tab:

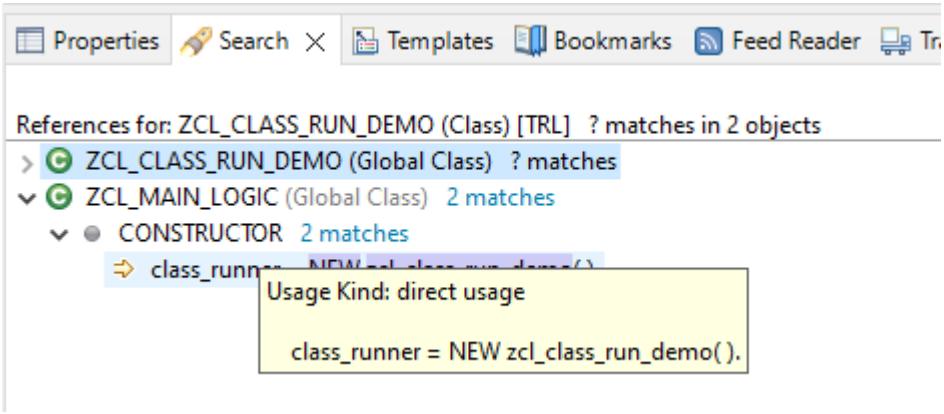


Figure 50 Result of the Where-used List

By using the filter function with the help of the filter icon, it is possible to restrict to packages, object types and users. Here, too, auto-completion can be used via **CTRL+SPACE** to find objects faster.

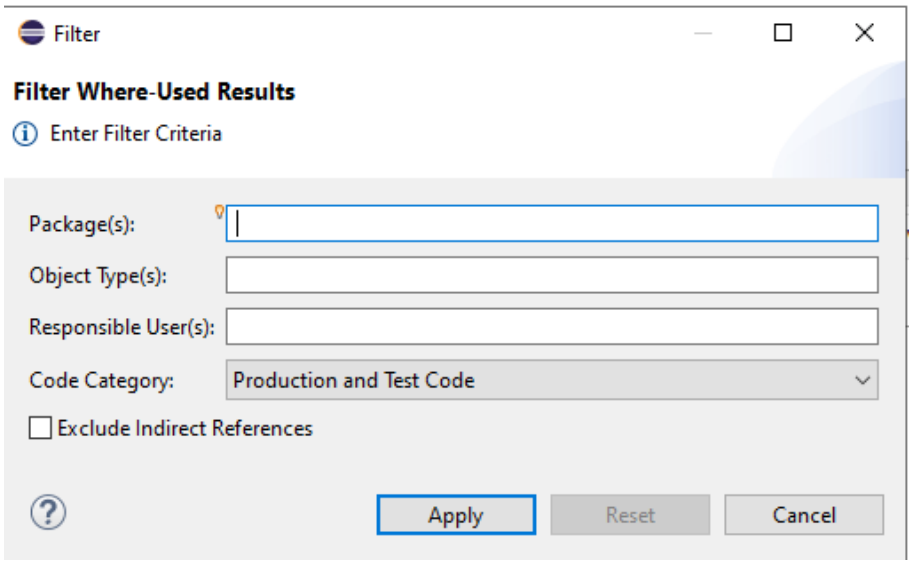


Figure 51 Filter for Where-Used-Search

Via **CTRL+.** and **CTRL+**, the citations can be conveniently browsed, i.e. the next or previous citation can be displayed.

3.2.5.6 Bookmarks

Bookmarks are "links" to development objects defined in the system. There are often key points where adjustments are often necessary. There can be various reasons for this, for example:

- A large historically grown include, where extensions take place again and again.
- A class with complex logic that turns out to be error-prone.
- Objects that are regularly worked on.

Bookmarks can be created by right-clicking on the list next to the source code:

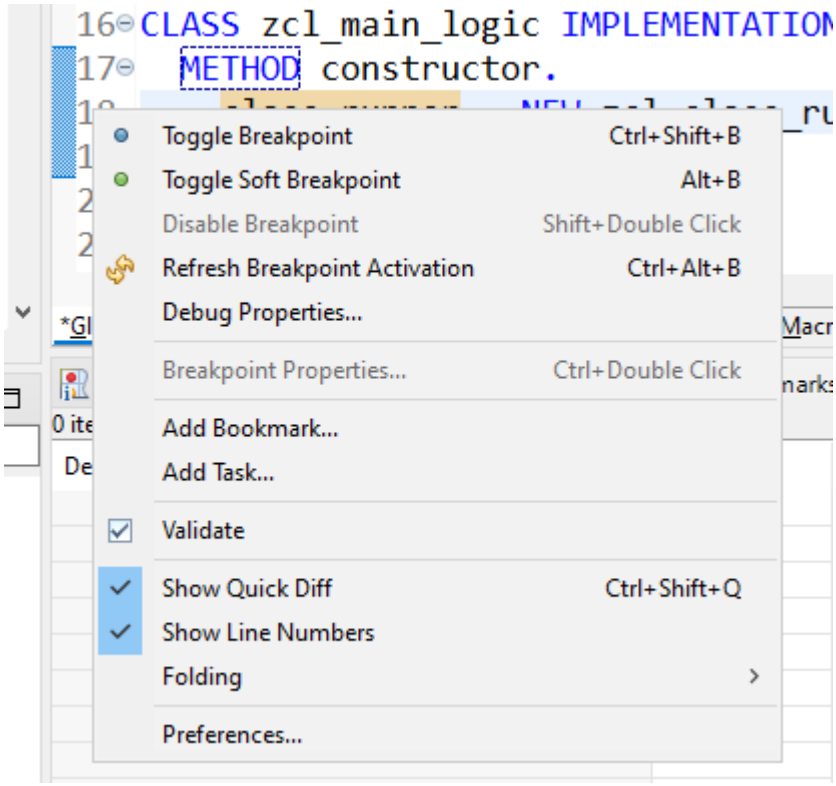


Figure 52 Menu for Creating the Bookmark

By default, the code of the selected line is specified as the name. It is a good idea to assign a descriptive and technically meaningful name, which can then be used to easily find the bookmark again.

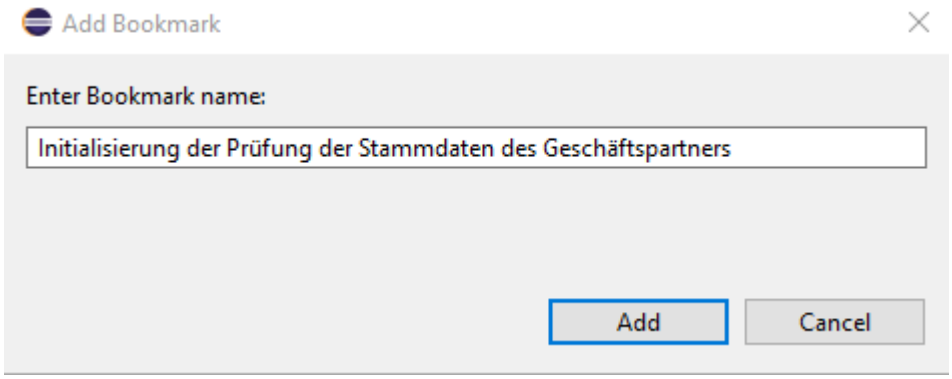


Figure 53 Entering a Name (Bookmark)

Within the source code editor, a small blue flag now appears next to the selected line number:

```
16 CLASS zcl_main_logic IMPLEMENTATION.
17   METHOD constructor.
18     class_runner = NEW zcl_class_run_demo( ).
19   ENDMETHOD.
20
21 ENDCLASS.
```

Figure 54 Display of a Bookmark in the Source Code

The bookmark is then available in the list and can be accessed by double-clicking. You can also customize the view and perform filtering via . The context menu allows you to delete the bookmark and edit the description.

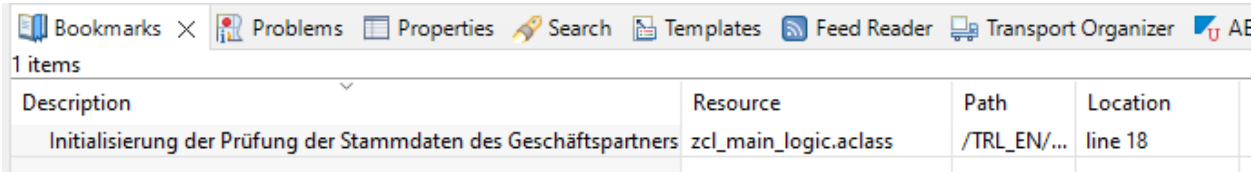


Figure 55 Bookmarks View

3.2.5.7 Sharing ADT-Links

In everyday developer life, it often happens that code has to be talked about together (e.g. during reviews) or a problem is found in a piece of coding that is the responsibility of another developer (no shared code ownership). Often it is then said "Can you please take a look at the class *XYZ method ABC line 1203* ... I think there's a bug?". The other developer has to navigate through the IDE until he finds the mentioned place.

ADT offers the possibility to send a link that leads the recipient directly to the appropriate code location when he clicks on it. To do this, an area must be selected in the source code and then selected "Share Link" in the context menu.

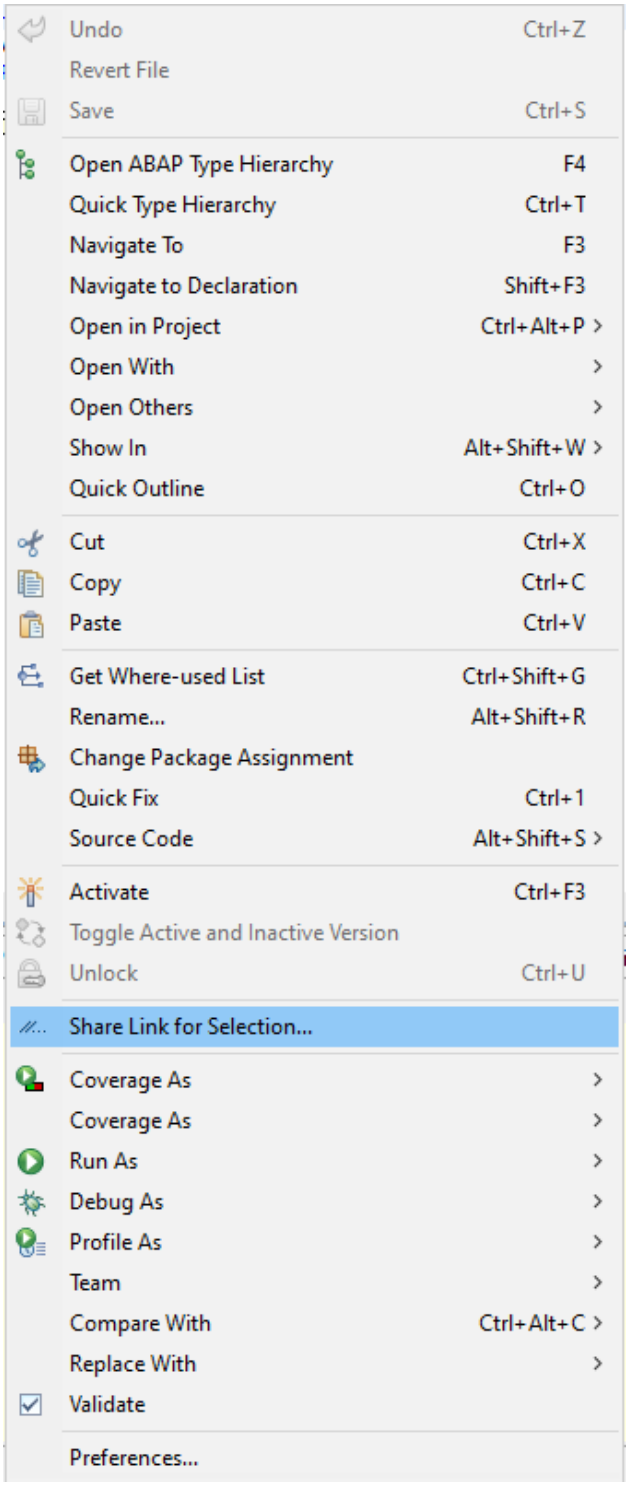


Figure 56 Sharing the Source Code as a Link (Context Menu)

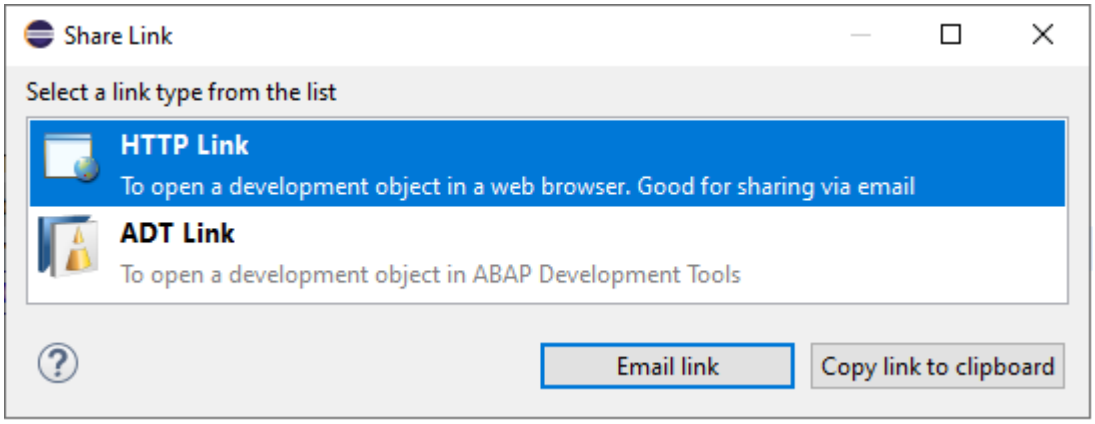


Figure 57 Link Sharing Dialog

The correspondingly generated link can then be sent to the colleague by e-mail or copied to the clipboard and sent via a chat program, for example. You have the choice between HTTP link and ADT link. HTTP links are opened directly in the browser, ADT links branch to ADT (Eclipse).

Structure of an ADT link (URI):

adt://<System>/sap/bc/adt/oo/classes/<Klasse>/source/<Methode>#start=18,0

More details can be found in the user guide .

3.2.5.8 ABAP Type Hierarchy

The View Type Hierarchy is used to represent the inheritance hierarchy of classes and interfaces. To use the view, all you have to do is place the cursor on the class or interface and press the shortcut **`F4`**. Alternatively, you can open the ABAP Type Hierarchy via the context menu.

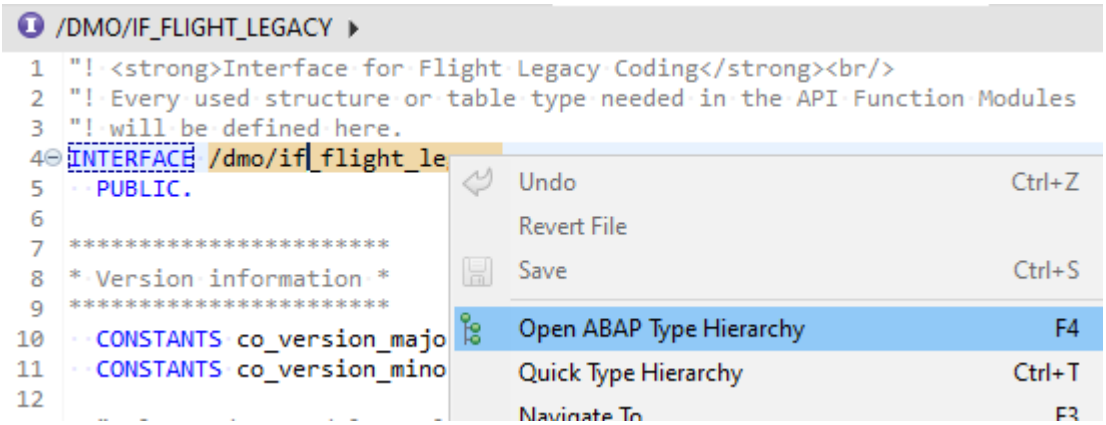


Figure 58 Opening the ABAP Type Hierarchy

The view displays the hierarchy in a tree structure.

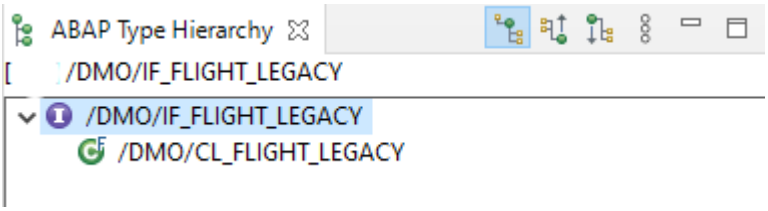


Figure 59 Display of the Type Hierarchy in the View

You can double-click to open the selected object in the ABAP Editor. The Quick Type **`Hierarchy`** can also be opened inline in the code via **`CTRL+T`**.

3.2.5.9 Transport Organizer

The Transport Organizer displays an overview of the open ABAP projects. Under the systems, you will find the current transports in the system. By default, you can see all your own transports. By right-clicking on a system and selecting "Configure Tree ..." these settings can be overridden and transports can also be viewed by other developers.

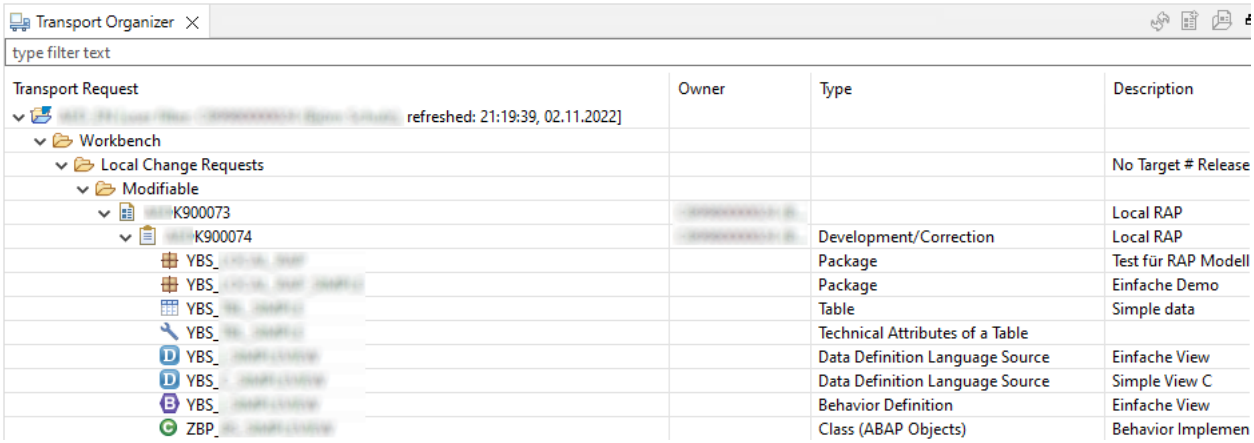


Figure 60 Transport Organizer View

All functions of the Transport Organizer (SE09/SE10) are available from the SAP GUI:

- Double-click - Show details of the job/task in your own view
- Right-click - Various functions such as: change user, consistency check, share

3.2.5.10 Feed Reader

The Feed Reader can be used in conjunction with ADT to receive specific notifications from the SAP system. By default, the following notifications are consumed for an ABAP project:

- Runtime error (dump) caused by the user's own user
- Runtime error for objects for which the user is responsible
- Systemmeldungen

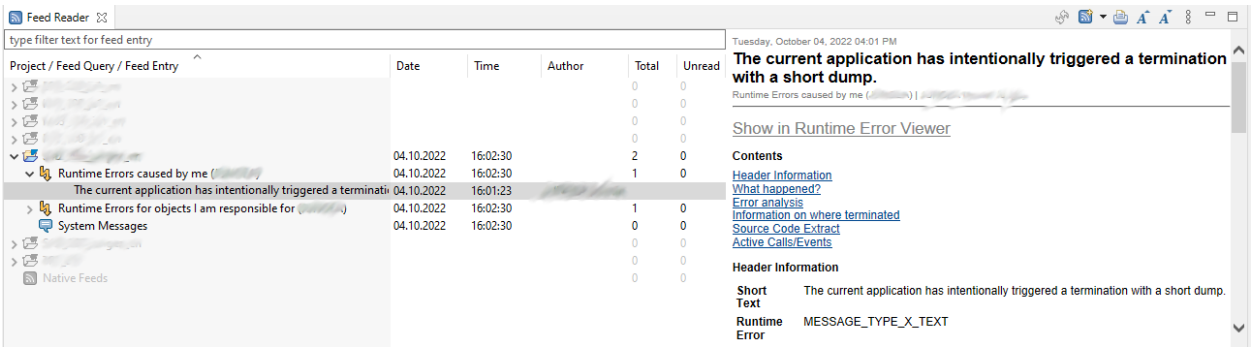


Figure 61 Representation of a Run-Time Error

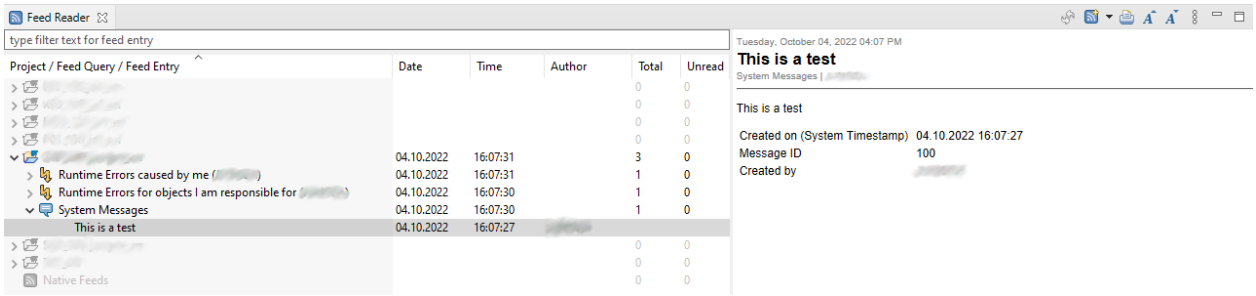


Figure 62 Example of a System Message

By clicking on the button "Add feed..." you can consume more notifications. The type of notification that is available depends on the release status of the linked SAP system. For most notifications, you can specify different filter criteria. In addition, you can also add a normal Atom or RSS feed via URL and thus consume it.

Details can be found in the user guide in the help functions of Eclipse. The article Getting Feeds is a good starting point.

3.2.5.11 ABAP Unit

After executing ABAP Unit tests, the ABAP Unit view opens and lists the executed test methods and their statuses. The ABAP unit tests can be started using the shortcut **`CTRL+SHIFT+F10`** *or the context menu in the ABAP Editor with the menu item* "Run As".

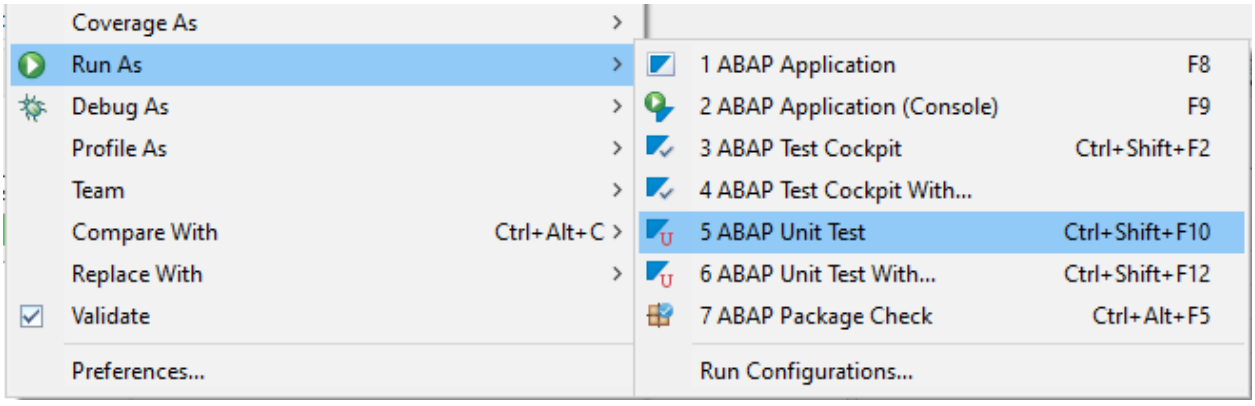


Figure 63 Executing the ABAP Unit Test via the Context Menu

Within the view, you can filter the list by status, re-execute test cases and display details about faulty runs. The latter appears by clicking on the affected test method.

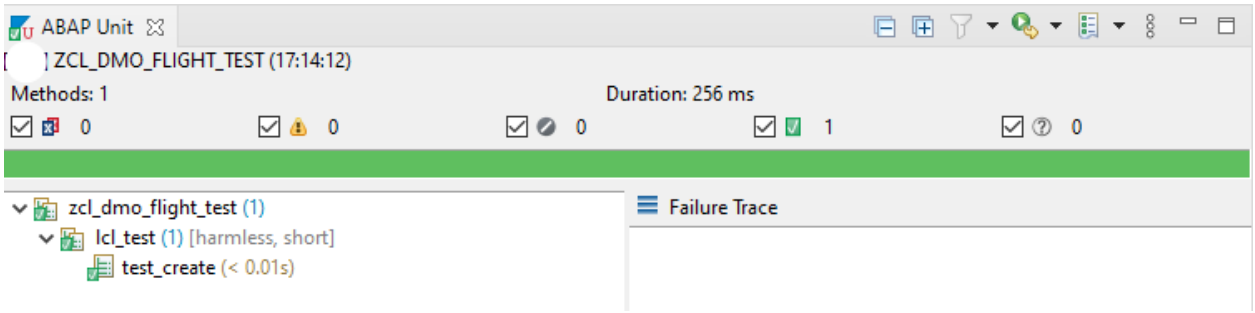


Figure 64 Display of the Results of the ABAP Unit Test

The scope of the test methods to be executed can be determined in various ways. On the one hand, it depends on the context. For example, if you focus on a single test method, only this method is executed. If the focus is on the class to be tested, then all test classes (and test methods) are executed. You can even extend the whole thing to a complete package by highlighting the package in the Project Explorer and running the unit tests. In addition, you can rerun individual or all tests in the view via the context menu - depending on which level you choose. For example, you could run all test methods of only one test class. This option is especially helpful if a test case is not successful and you need to analyze the behavior.

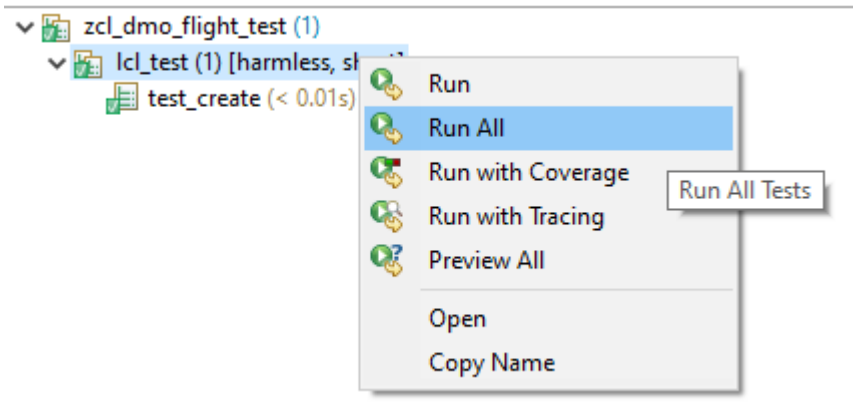


Figure 65 Restart the Execution

On the other hand, you can use "ABAP Unit Test With..." determine what kind of tests should be performed.

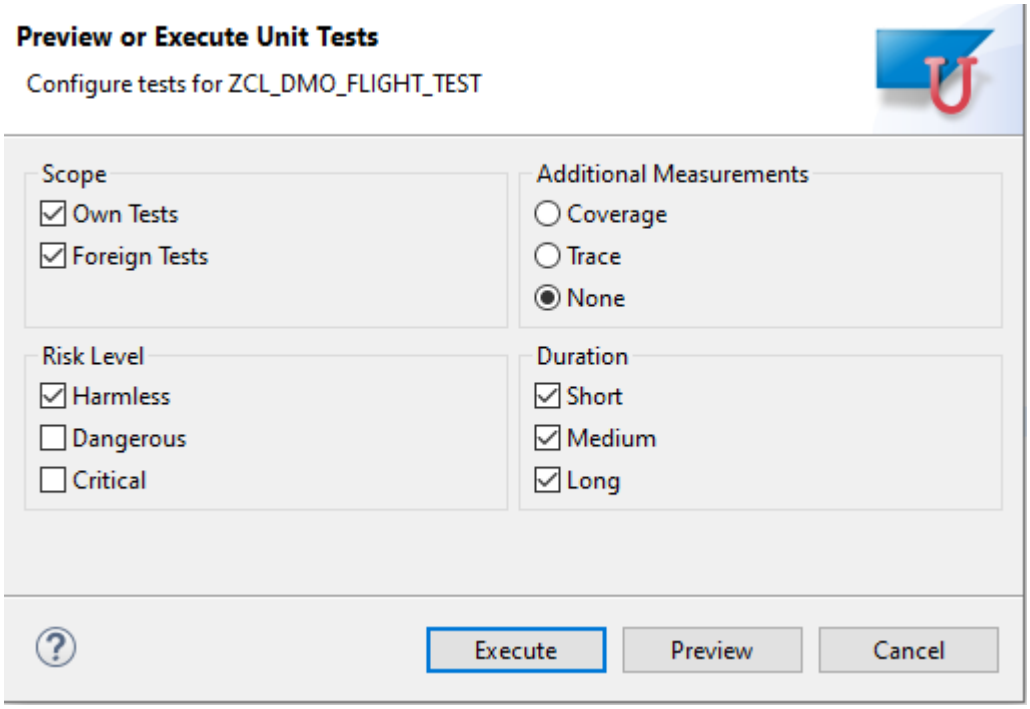


Figure 66 Dialog for Setting the Execution of ABAP Unit Tests

Thus, for example, you could only execute the test methods in a package that have the risk level "Dangerous" and the duration "Medium".

3.2.5.12 ABAP Coverage

The ABAP Coverage view appears when you run ABAP unit tests with coverage. Test coverage provides an indication of which code is not covered by automated testing. The test coverage there can be a conscious decision, since a test coverage of one hundred percent requires a lot of effort in development in the long run. Coverage can also provide clues as to where more test coverage might be needed. A blanket recommendation for test coverage cannot be given and may also depend on the criticality of the application.

This type of execution can be started via shortcut **`CTRL+Shift+F11`** or via the context menu with the menu item "Coverage As".

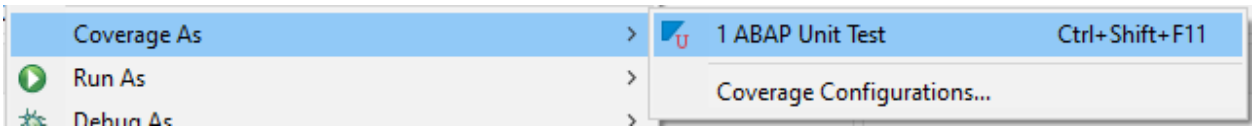


Figure 67 Carrying out the Coverage Measurement

The view displays the executed code in a tree structure and provides information about how many statements were executed absolutely and relative to the executed test methods. In addition, the ABAP Editor color-coded exactly which statements were executed (with green) and which were not (with red).

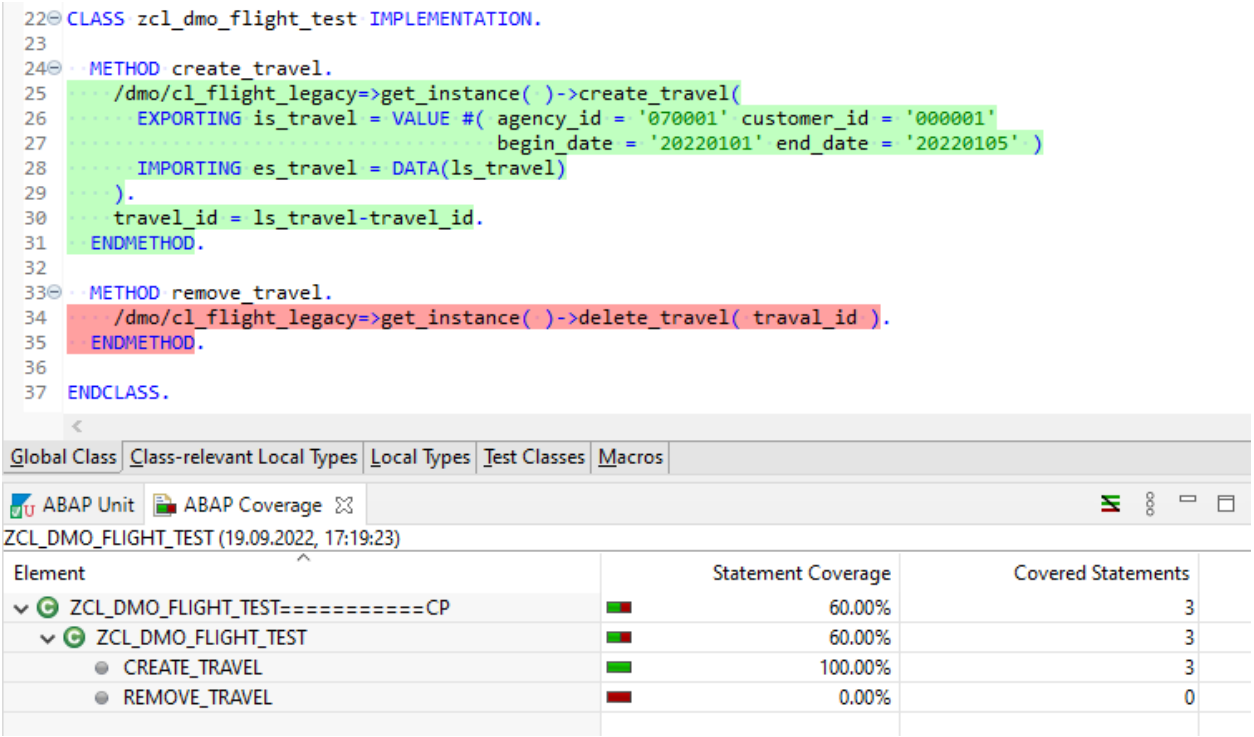


Figure 68 Color Highlighting of Source Code after Unit Test

If the colored view is not visible in the source code, it can be activated via the icon .

3.2.5.13 ATC and Exemption

The ABAP Test Cockpit can be executed in ADT as well as in the SAP GUI. There are several ways in which you can start the exam:

- Using the keyboard shortcut **CTRL+SHIFT+F2**
- Right-click in the Project Explorer under the item "Run As"
- In the ribbon at the top, below the button to start the object

After the tests have been executed, you will receive the view for the "ATC Problems", i.e. the feedback on the messages found by the set tests.

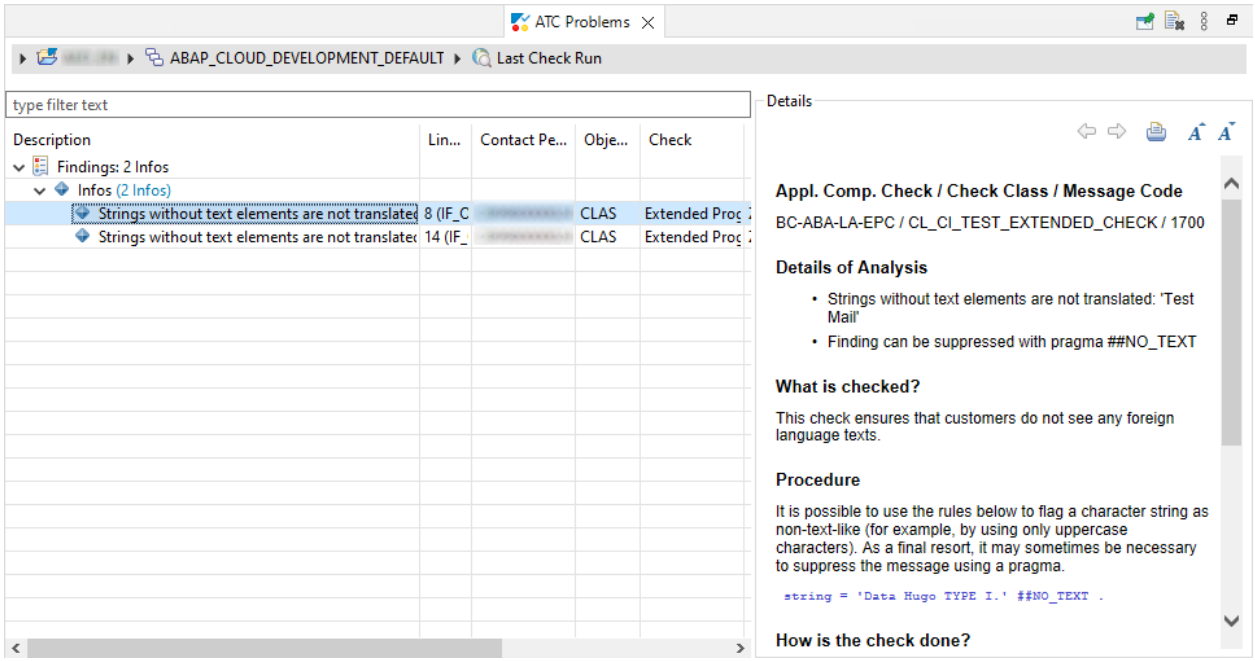


Figure 69 Display of the Results of the ATC Run

On the left, the messages are displayed sorted by the severity of the error. On the right you will find information about the selected entry. Here it is explained again for you what has been checked and what a correction can look like. In the button bar at the top of the view, the result can also be deleted again. In this way, the markers in the source code will disappear.

By right-clicking on the message, an exception can also be requested via the menu item "Request Exemption".

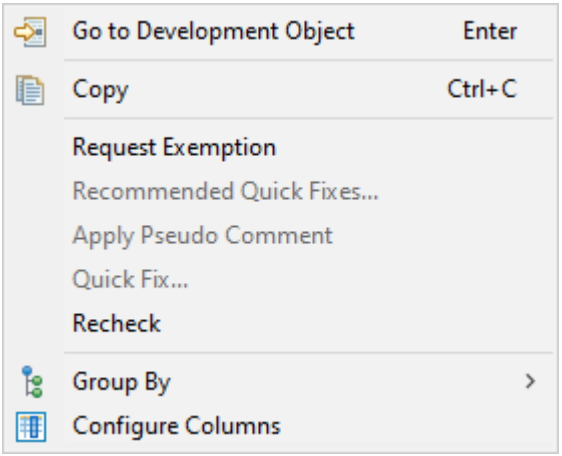


Figure 70 Requesting Exemptions via the ATC View

The information of the form corresponds to the SAP GUI and guides you through the approval process. At the end, the request can be processed as usual via the ATC Cockpit.

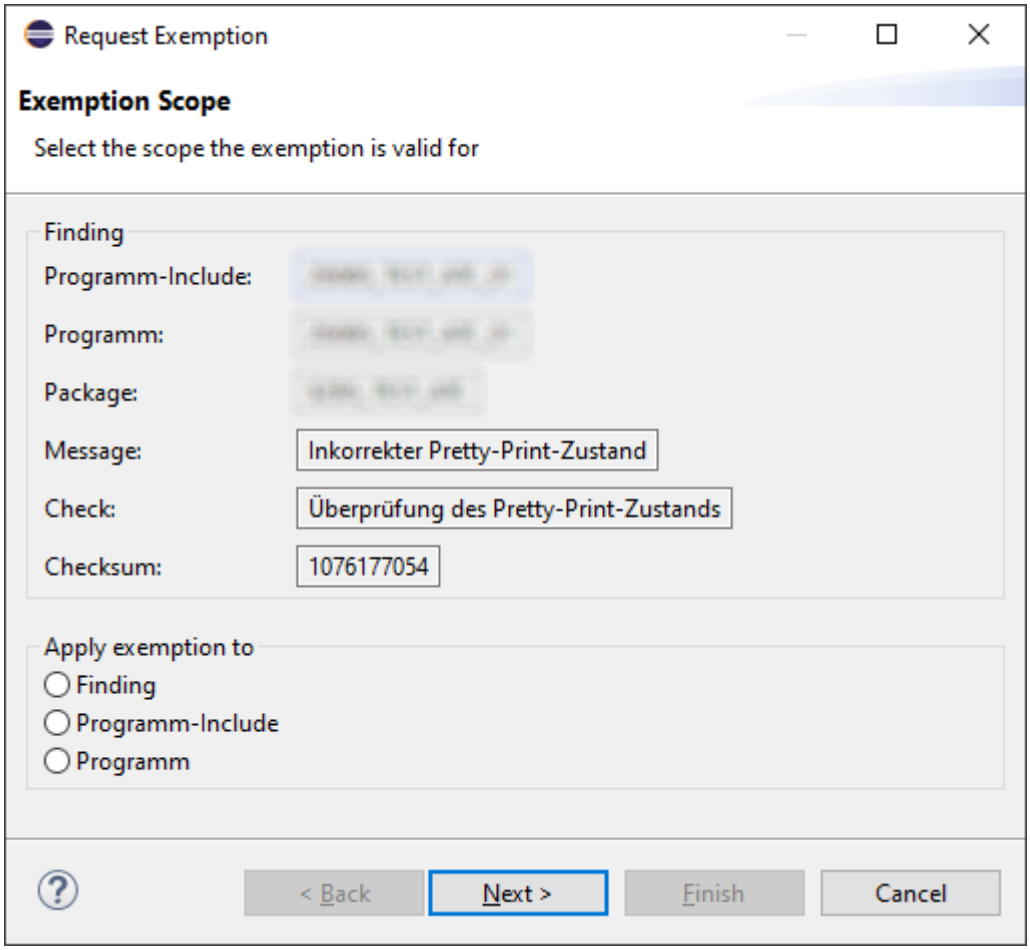


Figure 71 Dialog for Classifying the Exception

3.2.5.14 ABAP Language Help

In any source code editor, for example for ABAP, CDS, or BDL, the F1 key can be used to call the respective language help (not just ABAP!) directly for the keyword on which the cursor is located. Alternatively, you can also get it via the context menu by right-clicking on the corresponding instruction:

Source Code → Show ABAP Language Help

This allows you to get support at any time if you are not sure of the exact syntax of a statement.

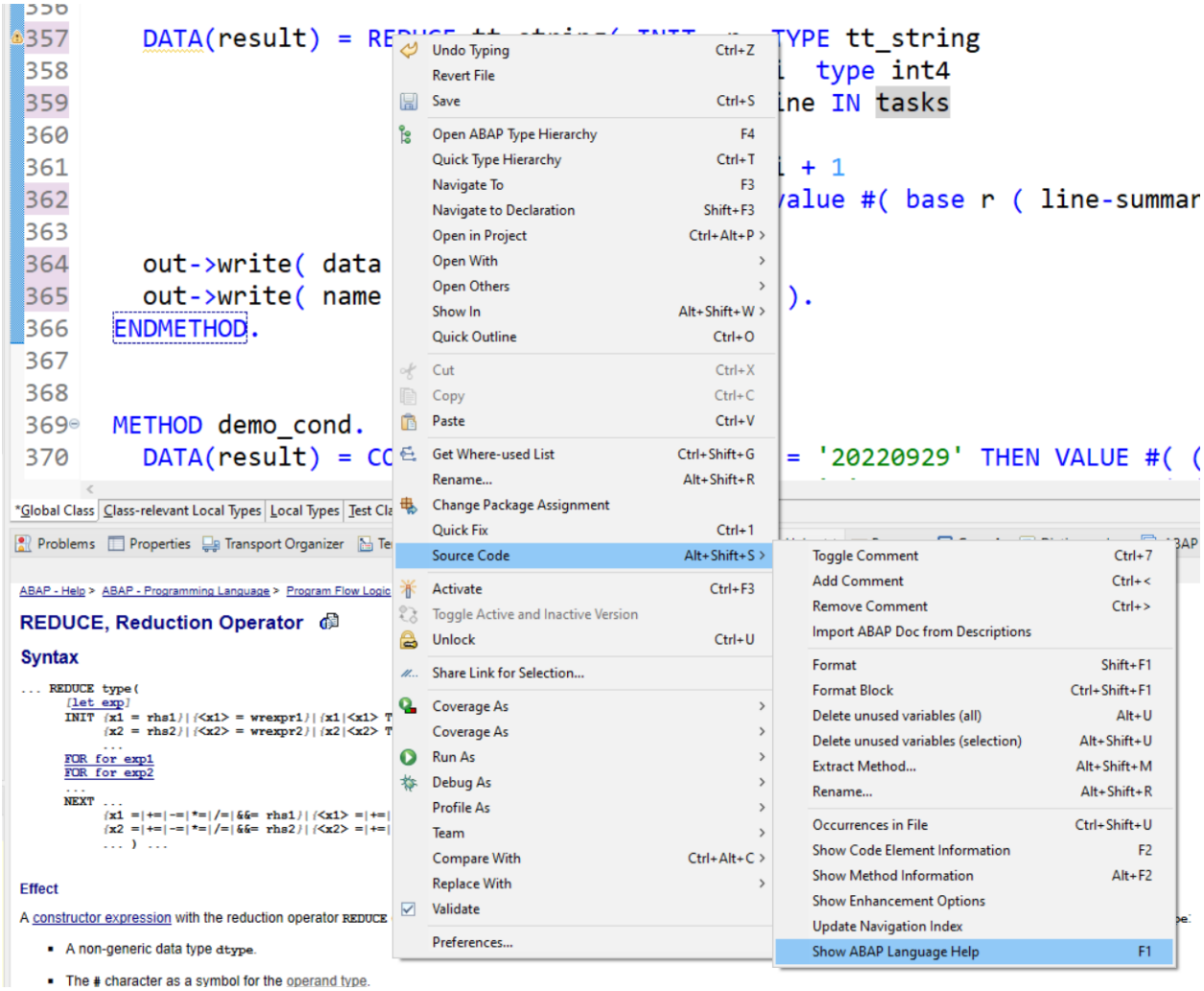


Figure 72 Calling the ABAP Language Help from the Context Menu

3.2.5.15 The ABAP Language Help View

The corresponding documentation is displayed as an HTML document in the ABAP Language Help View. This allows forward navigation via hyperlinks.

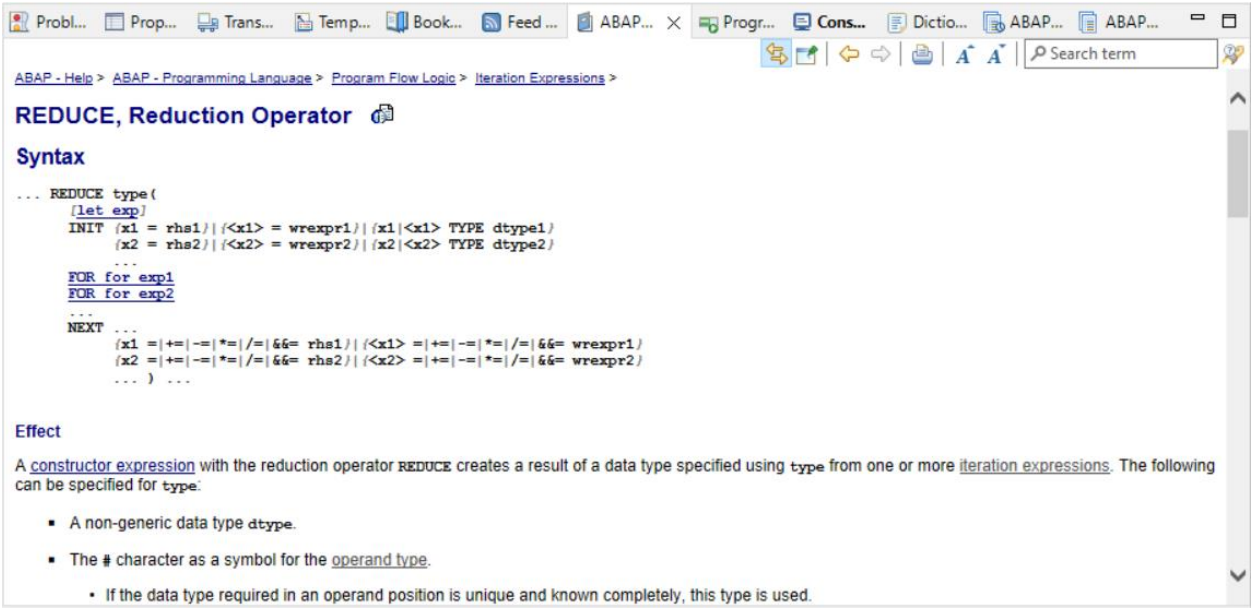


Figure 73 ABAP Language Help View

As in many views in Eclipse, there are some useful standard buttons here.

Figure 74 Button Bar of the View

- The yellow double arrow links the view to the active editor. This means that the view always displays the appropriate help for a statement on which the cursor is currently located.
- The green pin holds the content of the view. If help is requested again with F1, a new view for ABAP Language Help opens.
- The yellow arrows to the right and left are used for navigation (analogous to a web browser).
- With the help of the printer symbol, you can create a paper-like copy of ABAP Help with the appropriate hardware.
- The two symbols A with the arrows pointing up and down are responsible for changing the font size.
- The search field can be used to search the complete ABAP Help, including other languages such as CDS or BDL.

3.2.5.16 Application Help

In addition to the ABAP Keyword Documentation (or ABAP Language Help), SAP provides so-called Eclipse Help plug-ins for each development scenario. To do this, click

Help → Help Contents

in the menu bar to open the Help Browser.

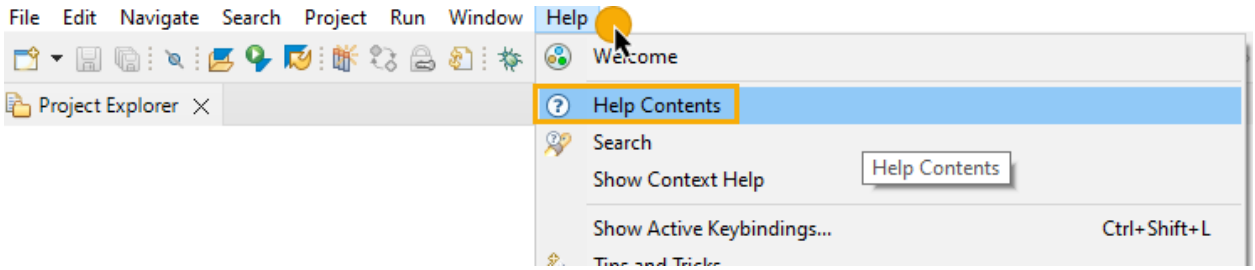


Figure 75 Navigation to Help Content

You can recognize the SAP Help plug-ins by the yellow book icon. Currently, the following Help plug-ins are available:

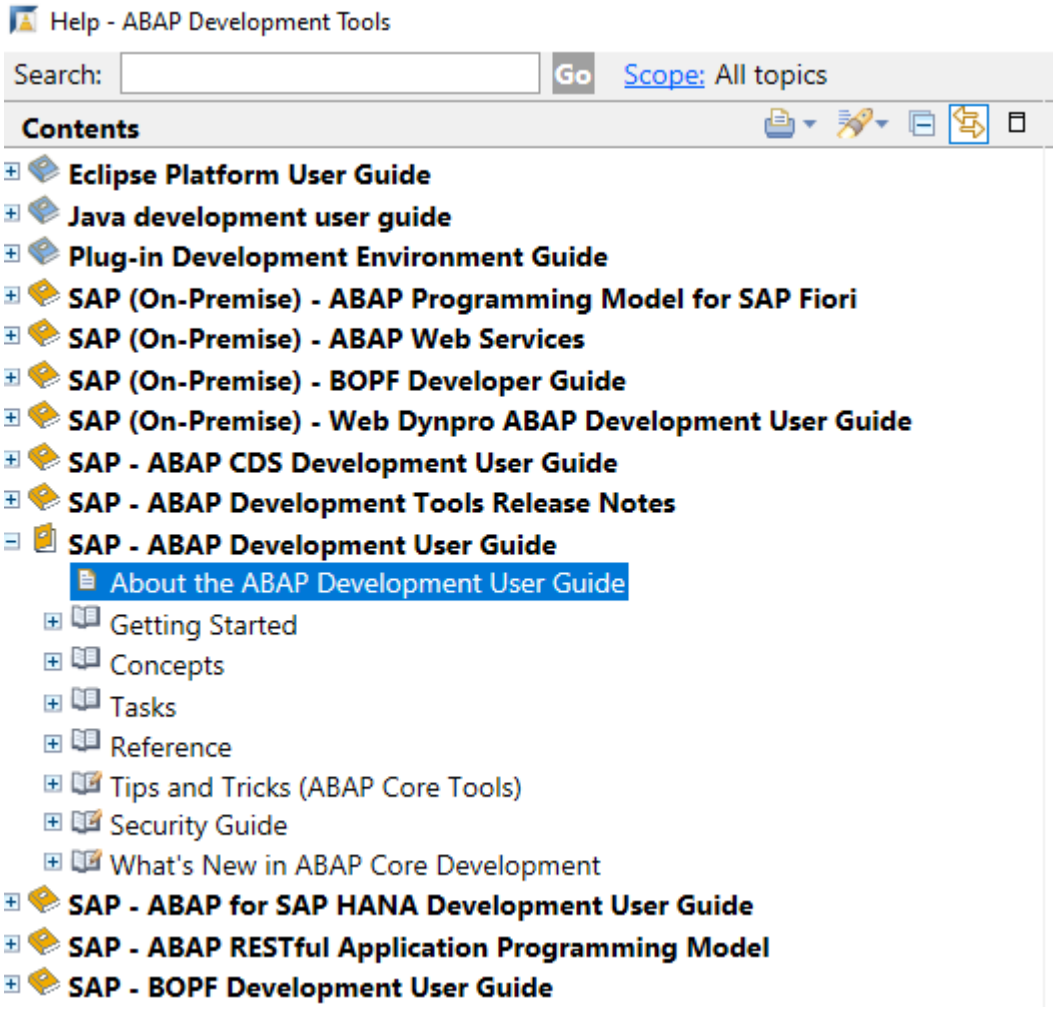


Figure 76 Overview of Available Help and Documentation

With the help of the search (Search) you can search for keywords. You can use Scope to limit the search to one or more help plug-ins.

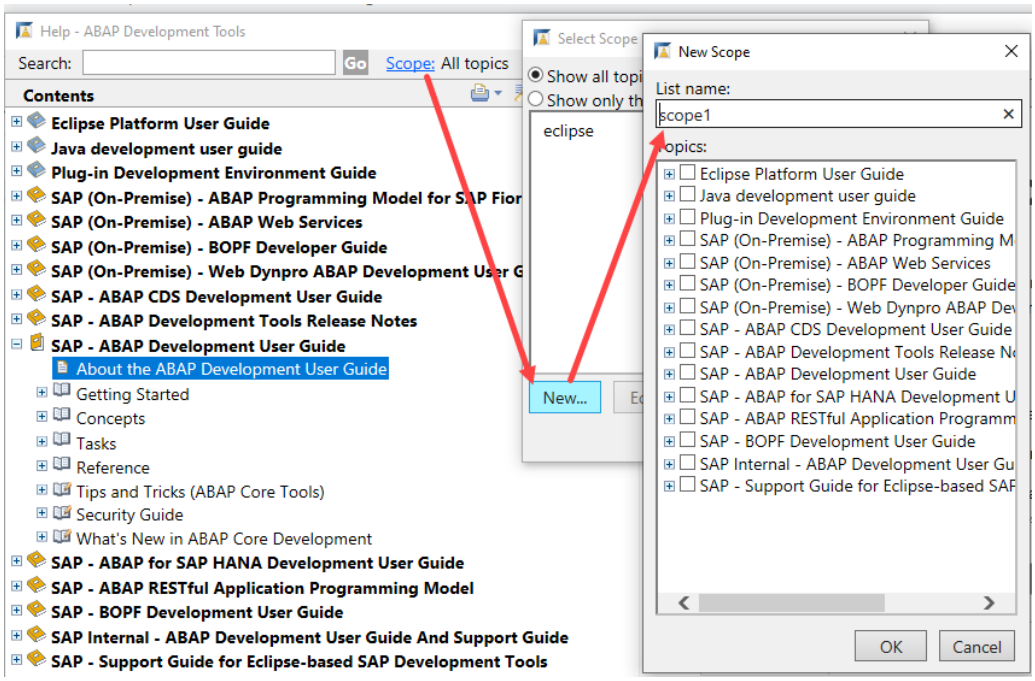


Figure 77 Search in Help

In wizards that offer the ? icon, you can open the context-sensitive help. This will take you directly to the respective help content that is available for the wizard.

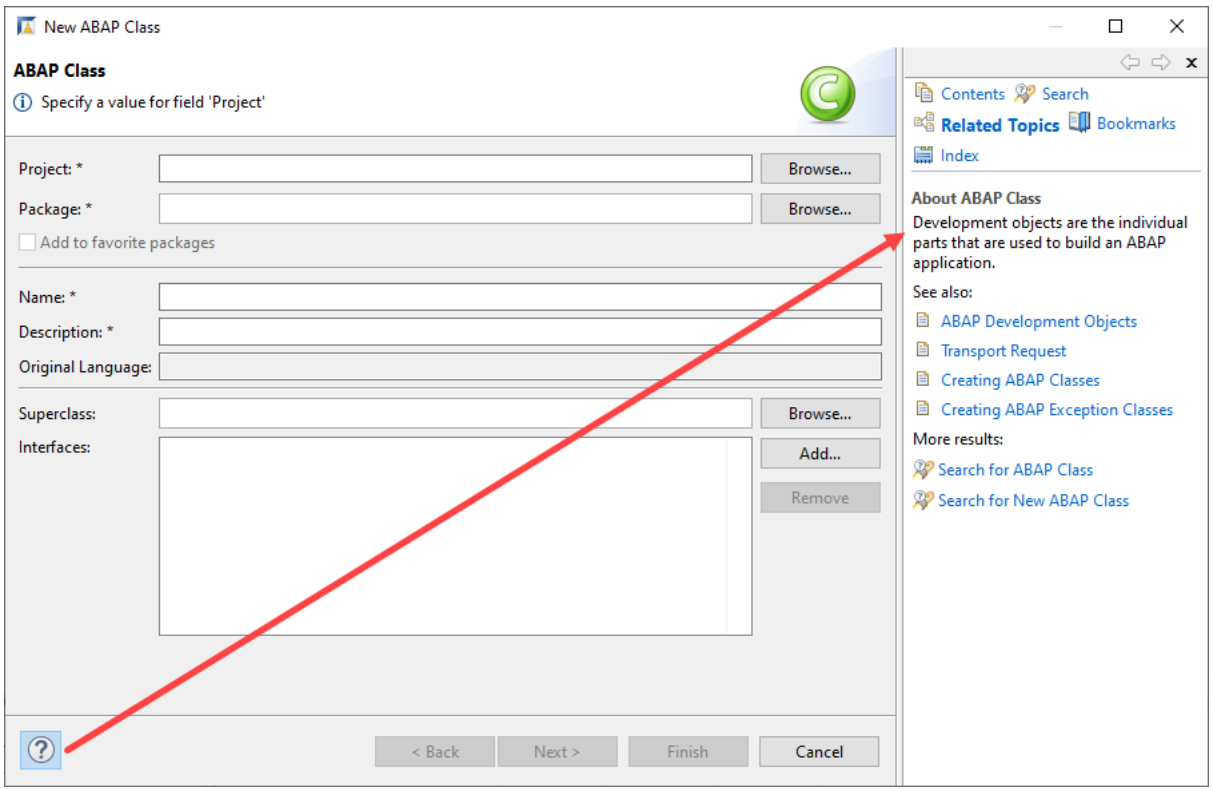


Figure 78 Further Help and Documentation

You can click so-called Active Links in a help page to open an Eclipse UI from the help. You can recognize Active Links by the green icon with the white arrow.

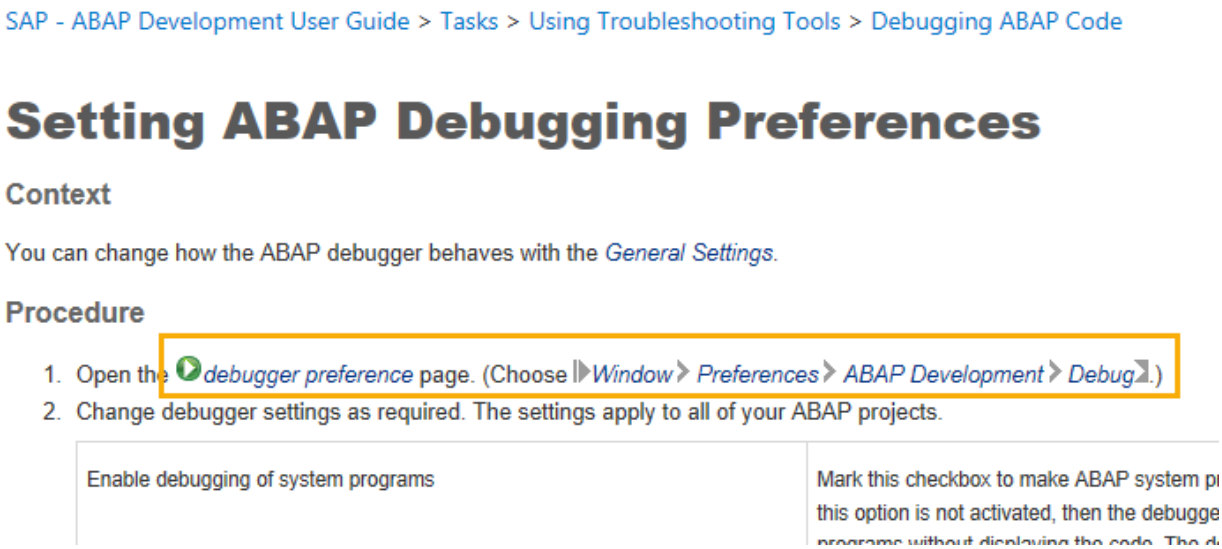


Figure 79 Display of Navigation

The help content that is identical to that which is available to you in ADT Client is also available online on the SAP Help Portal .

3.2.6 Refactoring Code with ADT

As briefly mentioned in Chapter 2 - Motivation for ADT, the numerous functions in ADT also allow code to be refactored. But what is refactoring, what advantages does it offer and what functions are provided by ADT? These questions will be answered in detail in the following section.

Refactoring refers to the modification of source code in order to improve its structure and readability without changing the functions. This means that no new functionalities are added, no functionalities are eliminated and the correctness of the solution is maintained, i.e. it continues to deliver the correct result. No new bugs will be introduced.

Maintaining correctness is certainly the most important aspect and one that is most difficult to achieve in the SAP world. The easiest way to prove correctness is through automated testing. Unfortunately, these are not very common in the SAP cosmos, have been little supported in the past, and are often difficult to implement because the structures of historical ABAP code are difficult to implement. Thus, however, the creation of automated testability is a primary goal of refactorings.

In addition, there are other reasons for refactoring:

- Increasing the comprehensibility of the code ("clean code")
- Improve the customizability of code for extensions
- Reduction of technical debt
- Updates of deprecated commands/modules

Refactoring is an integral part of software development and should be done during daily development in order to maintain a certain standard of quality. It is not advisable to carry out special "refactoring sprints" or similar, as these are often viewed skeptically by financiers or not approved at all. The authors therefore recommend observing the Boy Scout rule: "Always leave the code better than you found it."

In the past, this was usually associated with a great deal of effort with the SE80. Due to the lack of support for the IDE, the refactorings had to be carried out manually to a large extent. This high manual effort and their susceptibility to errors led to a low acceptance of this process and clean code in general.

With ADT, this situation has now changed. If there are no automated tests as a double bottom, it is still possible to carry out so-called save refactorings, which we would like to describe here. A save refactoring is characterized by the fact that it is carried out with the help of tools, i.e. by functions of the IDE or with additional plug-ins. This eliminates the risk of introducing new errors into the code due to manual changes. Automation makes it easy to carry out refactorings and thus become part of daily work.

ADT offer the following refactorings via Quick Assists (**`CTRL+1`**):

1. Rename Identifier – renaming within a code block or globally
2. Extract Method – Extract a method from source code or from an expression
3. Extract Constants – Convert text literals into constants
4. Extract Variables – Extract and convert variables
5. Move Member – Modify and move attributes of classes
6. Exception Handling – Automated creation/transformation of exception blocks

In particular, the Rename and Extract Method functions support the developer in keeping the code clean and avoiding or reducing code redundancies.

For example, since the rename function handles identifiers not only within the unit, but across all users, it is now easy to change an inappropriately chosen name to a name that better fits the overall context. There is no risk that users will be forgotten and errors will be built into the code.

The Extract function analyzes the selected code, provides assistance with parameter assignment and replaces the location of the code with the call of the newly created method. If the method to be extracted has a comment, it is used as a suggestion for naming the method.

Furthermore, the authors recommend the use of the plug-in "ABAP Quick Fix" (https://marketplace.eclipse.org/content/abap-quick-fix) by Lukasz Pegiel, which is described in Chapter 7 - Plug-ins. In general, the refactoring tools in ADT provided by means of quick fixes offer enormous help both in the creation and revision of existing code. The use of this helpful plug-in improves the currently edited code on the one hand, but also helps to apply the newer language constructs yourself when creating new code, if you are not yet experienced in them.

3.2.7 Versioning and Comparison

The context menu item "Compare with" hides some of the most important features for daily work. These work for all source code editors in ADT, not just for the development of ABAP code.

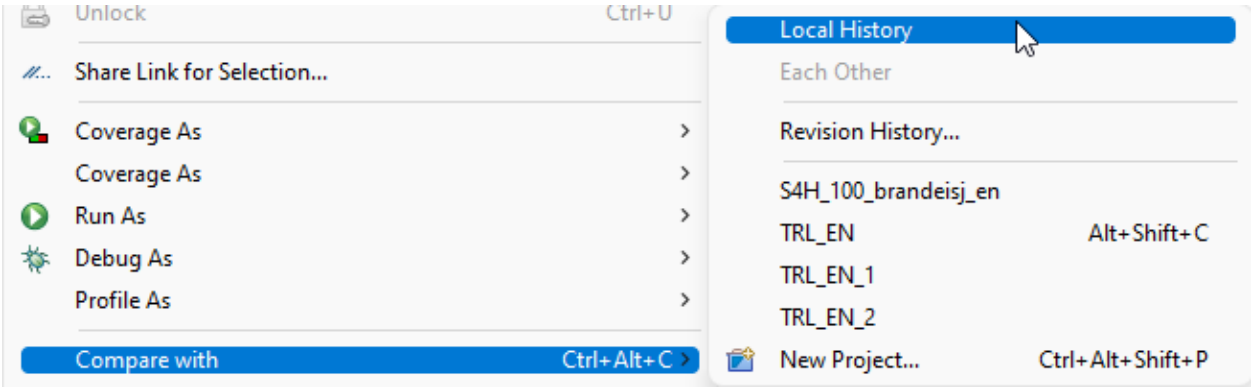


Figure 80 Context Menu for Comparing Versions

3.2.7.1 Local History – The local Visioning

The Local History provides access to older versions of the current source code document from the Eclipse Workspace that the user is currently working with. Each time the object is saved, a version is drawn. This means that you can easily track your own work over the course of hours and days and easily switch back to older versions.

Since the local version history only refers to your own Eclipse workspace, it can happen that you also have changes on another device or that a colleague has last made changes.

3.2.7.2 Revision History – Version Management of ABAP

Under the menu item Revision History, you can access the "normal" source code management of the ABAP server, which was already available in the SAP GUI. By default, versions are pulled here when a transport request is released.

The versions of the revision history are accordingly globally accessible to all users, regardless of the workspace of the Eclipse installation.

3.2.7.3 Other Project Name – Compare across systems

If you pick a project from the list, you can compare across systems. This can be a system from the same system landscape, e.g. the production system, or a completely different system. This means that code from different systems that do not have an RFC connection to each other can be compared from ADT. In complex system landscapes, this offers a great advantage over GUI-based comparison options.

3.2.7.4 Comparison View

In the Comparsion View you can see the current state of the source code on the left and the version selected for comparison on the right. The discrepancies are highlighted:

1. Green are the things that are new in the current version
2. Red are the things that have been deleted and are in
3. The changes are highlighted in grey

With the buttons, the old state can be restored by copying from right to left. However, it is also possible to make changes directly in this comparison view on the left side. After saving, the comparison is repeated.

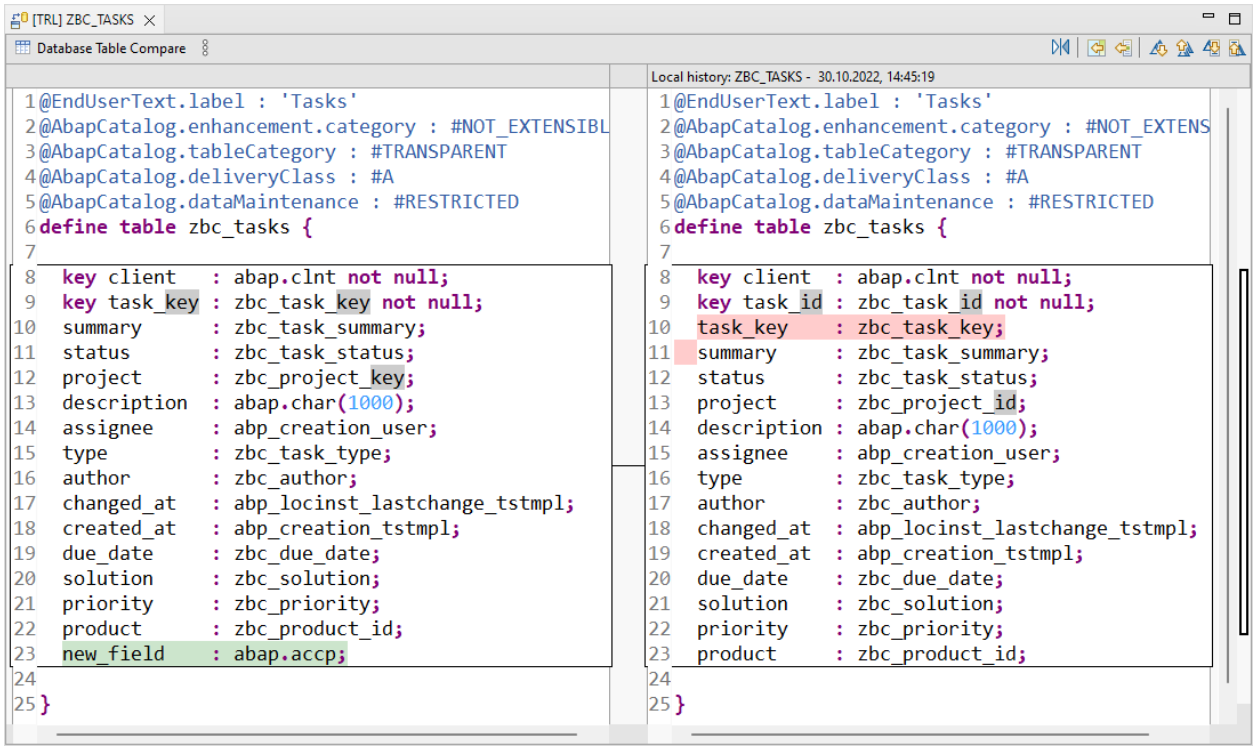


Figure 81 Comparison View – Comparison of two Versions

If you want to take over an old version completely, you can select the appropriate version directly from the context menu using Replace With → Local History.

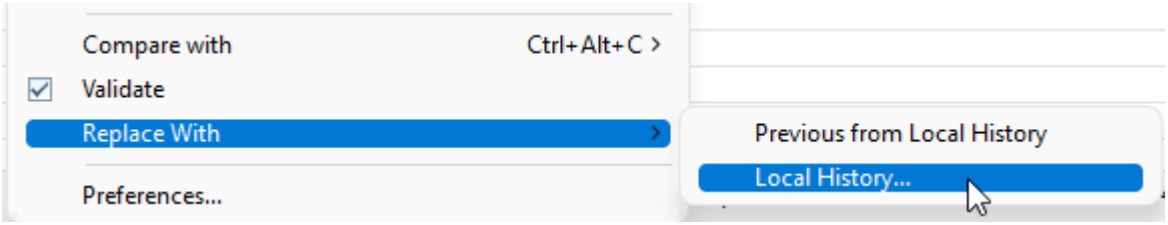


Figure 82 Context Menu for Completely Transferring a Version from Local Version Management

3.2.8 Documentation with ABAP Doc

3.2.8.1 What is ABAP Doc?

ABAP Doc enables code-based documentation, such as methods and their parameters.

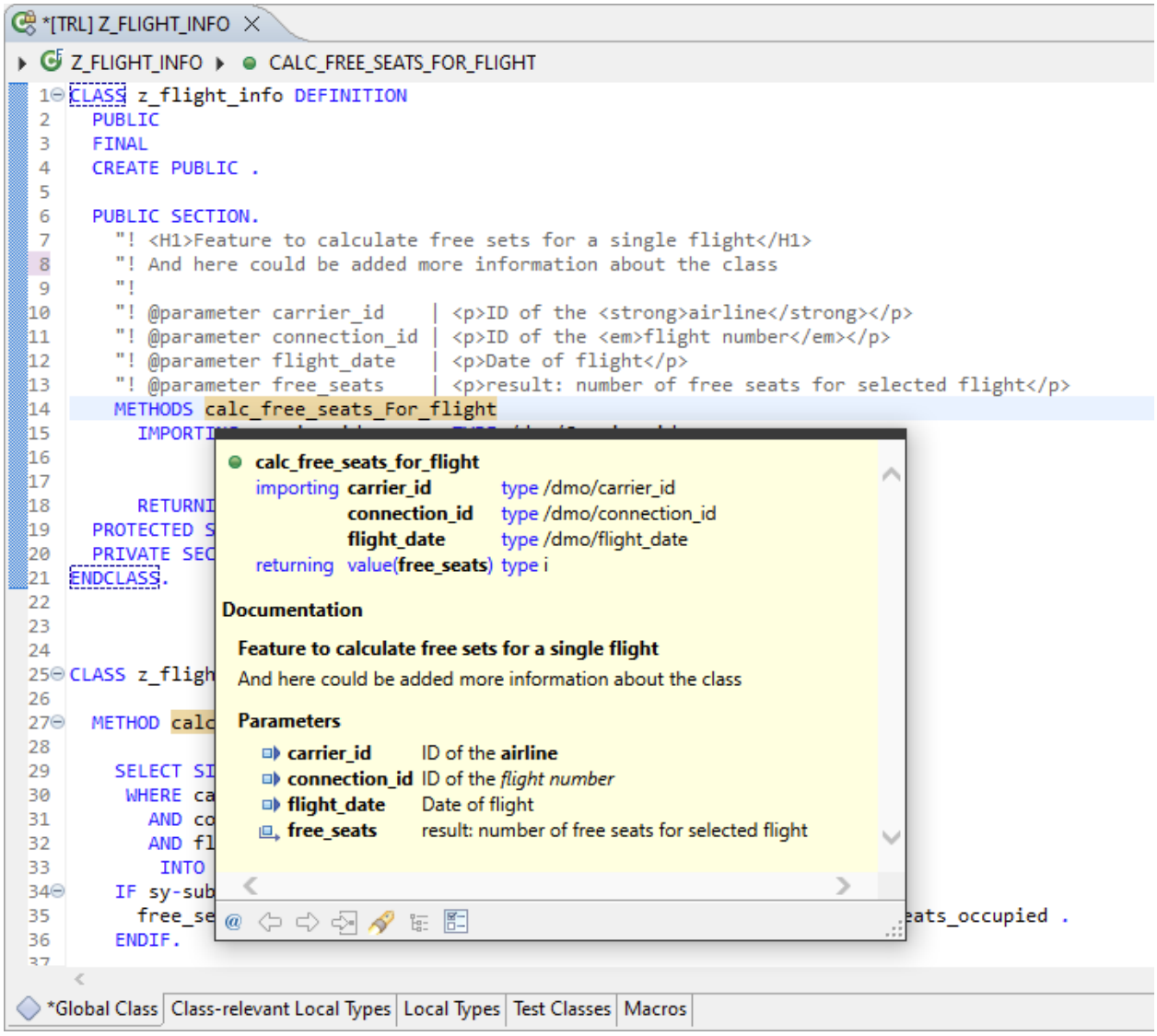


Figure 83 ABAP Doc Documentation of the Method

ABAP Doc is a feature that is only supported in ADT. Instead of the form-based editor with the option of briefly describing methods, which is no longer available in ADT, a much more powerful replacement has been established with the ABAP Docs, which is also available in a similar form in other programming languages (e.g. JavaDoc).

In the following, the use of ABAP Docs in the context of classes/methods is discussed in detail for better readability of the text. However, the ABAP Docs can also be applied to other development artifacts, such as function modules (see SAP Help).

The ABAP Doc function can be used to enter textual descriptions for classes and their methods. In addition, descriptions can also be stored in ABAP Doc for the individual parameters and exceptions.

The notes to be created with ABAP Doc are created in the definition area. However, the added value arises above all from the simple access to this documentation by the user. This is possible both at the call point and in the area of implementation of development artifacts using the F2 key. In addition, the texts created in ABAP Docs can even be formatted using HTML tags. In this way, the documentation can be enriched with headings or text formatting and thus presented in an even more appealing and structured way.

The "Synchronized" tag is used to transfer descriptions created in ABAP Doc to the short texts displayed in the SAP GUI, so that the headings are visible even when viewed using SE24/SE80.

This can be useful if objects contain enhancements that are not edited directly in ADT and therefore the modification still has to be done in the GUI-based tools.

We do not recommend a mixture of ABAP Doc and GUI short texts. The use of ABAP Doc is the method of choice to relieve the functional code of comments and to provide the user with helpful information about the development artifacts.

3.2.8.2 Use of Quick Fixes to Create ABAP Doc

Creating the ABAP Doc is easy by calling the Quick Fixes. To do this, select the method definition, call the Quick Fixes, and choose "Add ABAP Doc". If a method definition has been changed, for example, by adding a parameter, and an update of the documentation is required, the ABAP Doc can be updated by selecting the ABAP Doc area directly instead of the method signature and thus calling the Quick Fixes.

3.2.8.3 Further Information about ABAP Doc

Further information about ABAP Doc can be found in the official SAP Help (for example) under ABAP Doc - ABAP Keyword Documentation (sap.com) (7.50), in the example class CL_DEMO_ABAP_DOC and in the User Guide.

3.2.9 Executing Source Code

Even in ADT, open source code can still be executed comfortably. F8 is used to initialize an SAP GUI instance of the respective system and execute the open development object. In the case of classes, for example, this corresponds to the "Test Class X" function, and in the case of reports, the report is executed normally.

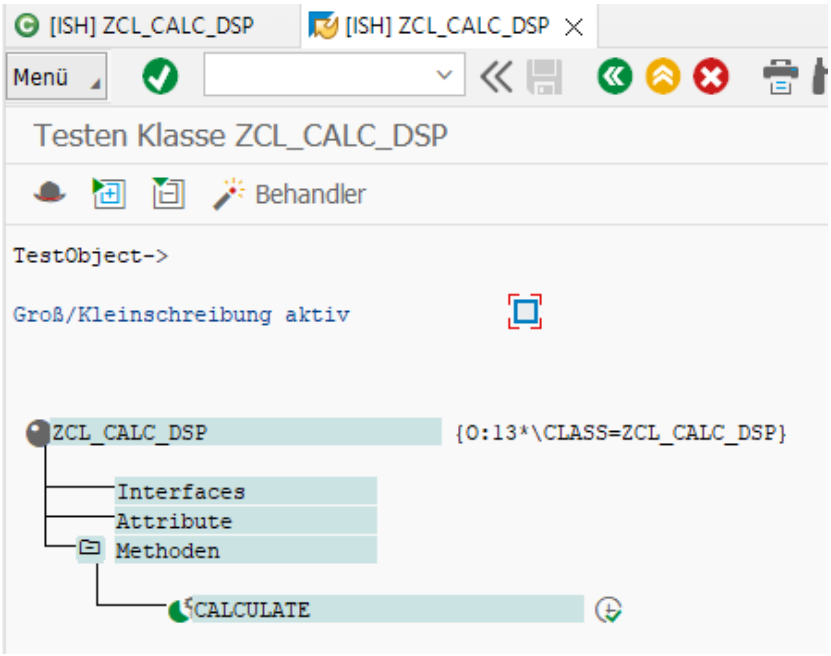


Figure 84 Executing a class in SAP GUI

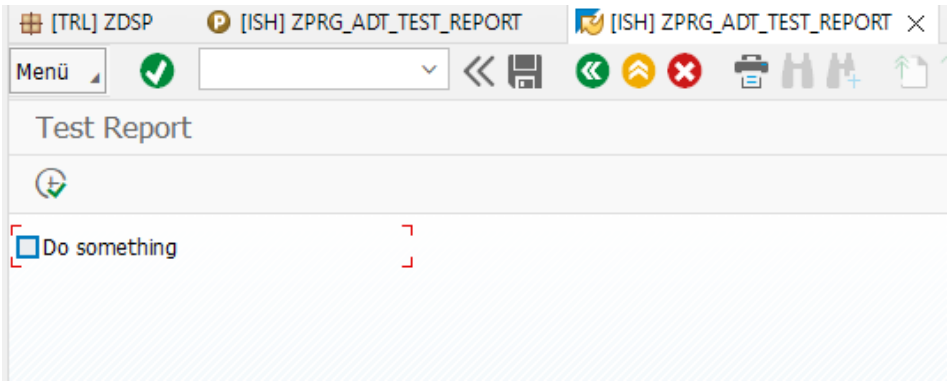


Figure 85 Result of Execution

If any development object is to be executed, **ALT+F8** can be used to search for an object via the Object Finder. Any project can be selected, i.e. an SAP system connected to ADT. This does not have to be the system in which development is currently taking place – it can also be a quality assurance system. It is important that the ADT functions are activated for this system or that the corresponding authorizations exist.

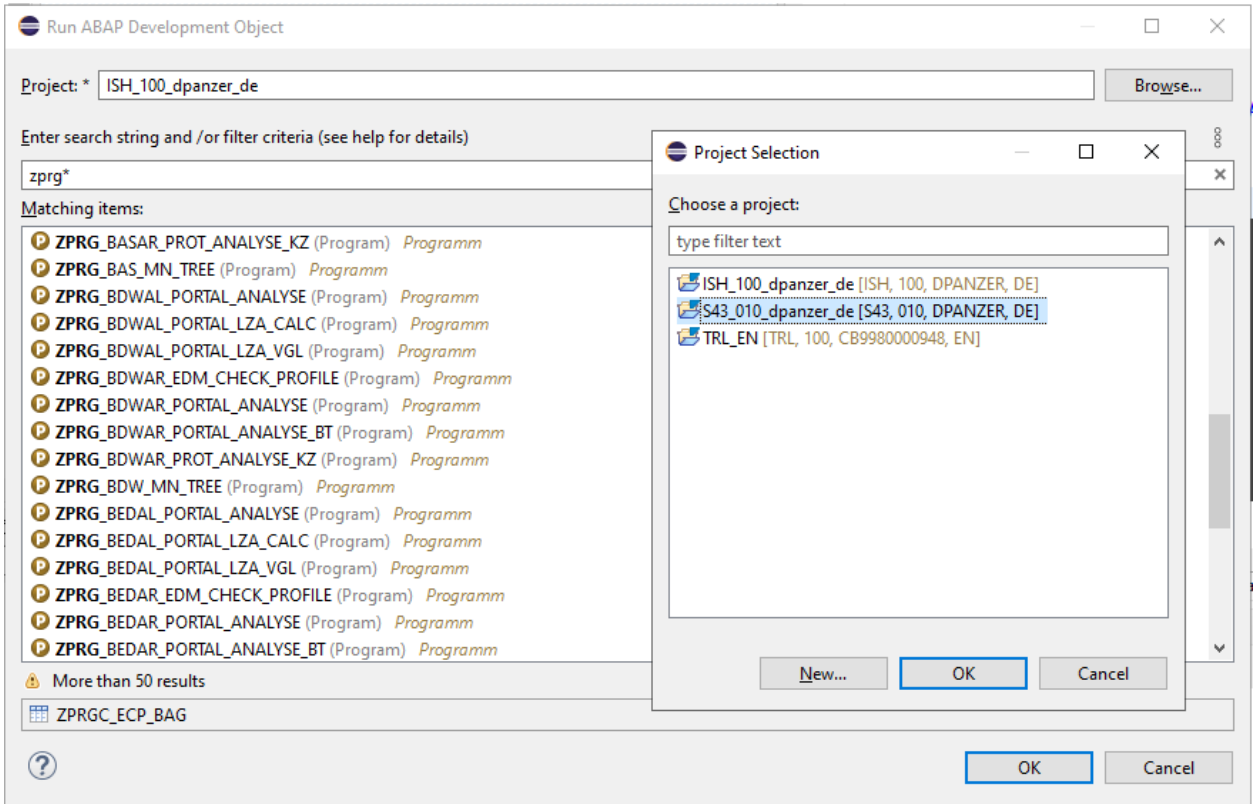


Figure 86 Selection of the Project

The menu item "Run → Run History" provides additional information about objects that have already been executed, so that their execution can be conveniently repeated.

Classes that implement the interface if_oo_adt_classrun can also be executed directly via F9 as a console application and thus generate output in the console.

Reports that generate write output can be executed via F9. The WRITE output is then redirected to the console as well.

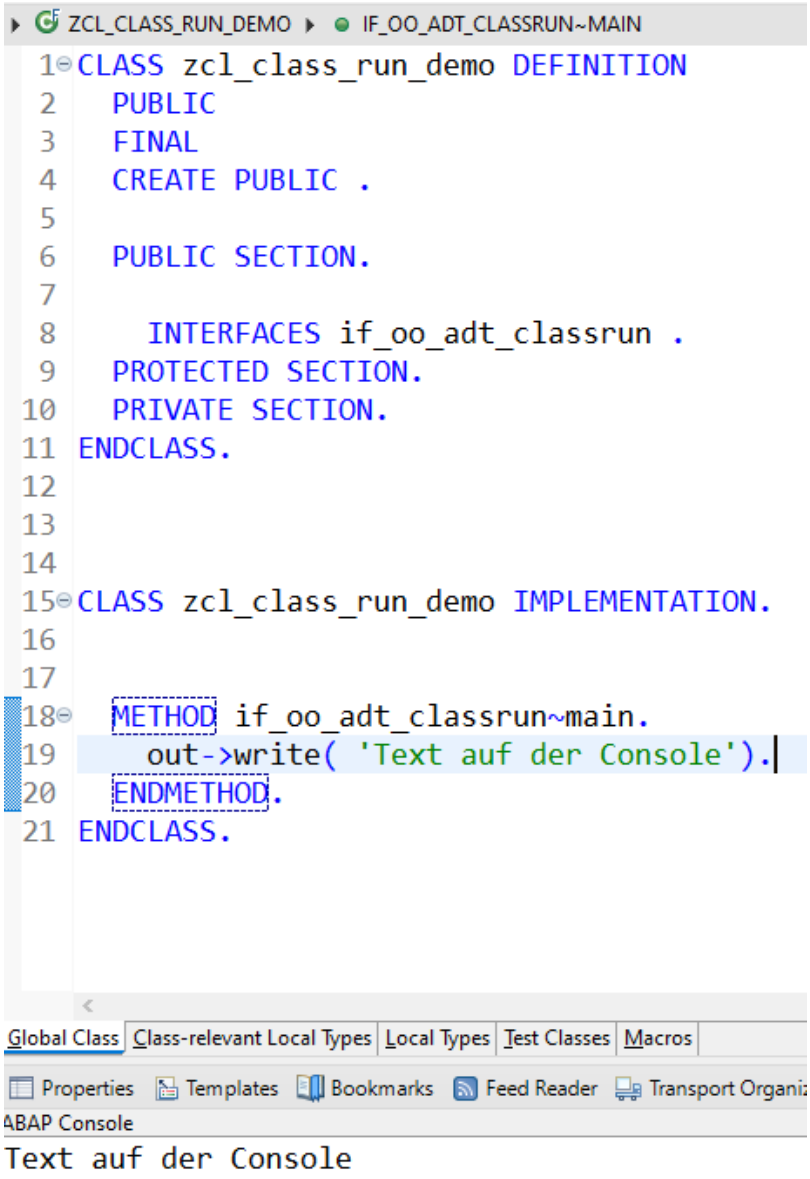


Figure 87 Output to the Console

3.2.10 Data Preview

The View Data Preview can be used to display data from database tables and (CDS) views. The view opens either by selecting a corresponding object in the Project Explorer and pressing the shortcut **F8**, or by using the context menu.

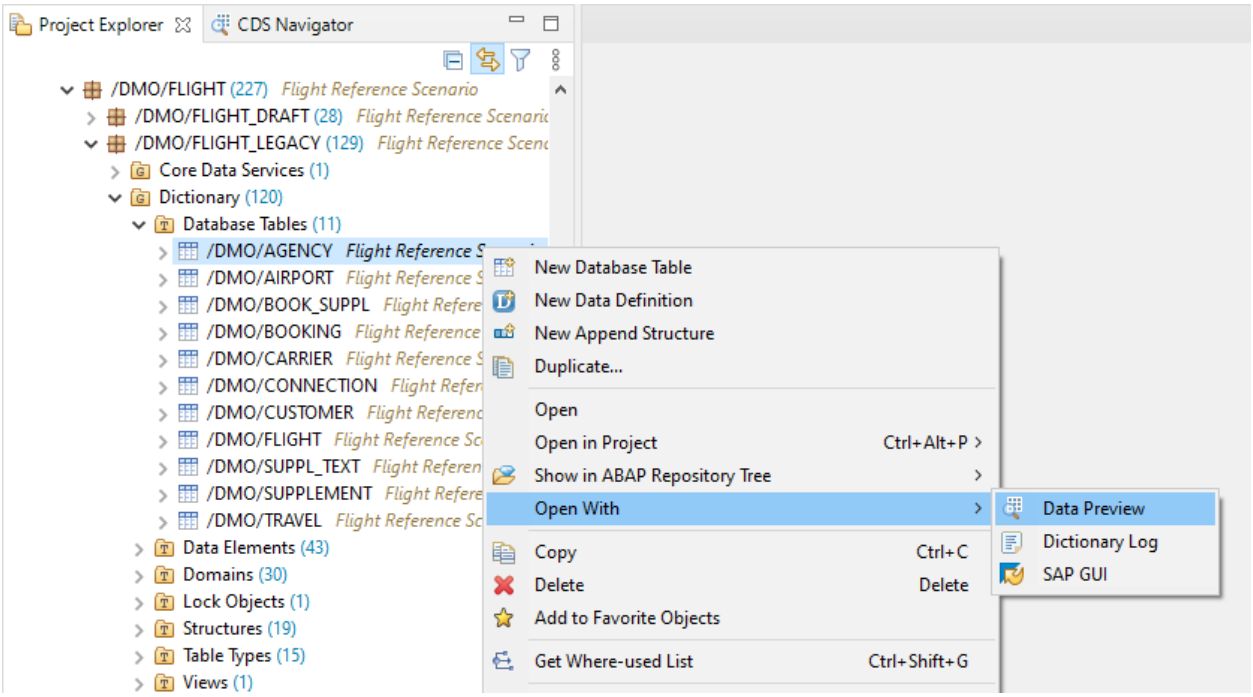
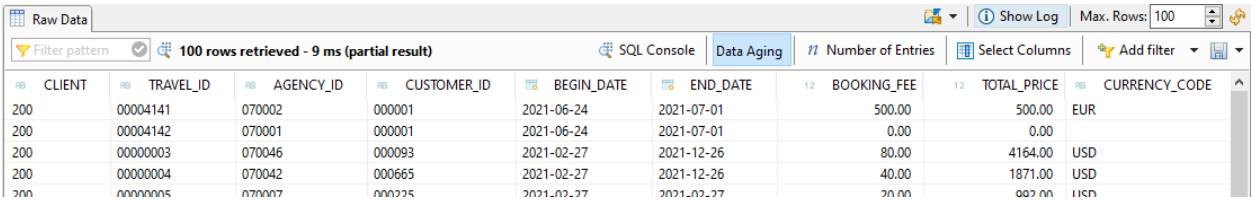


Figure 88 Start the Data Preview from the Table

The shortcut **F8** also works if you have a corresponding object open and the focus is on the object.

When opened, the view immediately executes the data selection and then lists the selected data in table form. In addition, it shows the number of selected rows and the time required for them.



CLIENT	TRAVEL_ID	AGENCY_ID	CUSTOMER_ID	BEGIN_DATE	END_DATE	BOOKING_FEE	TOTAL_PRICE	CURRENCY_CODE
200	00004141	070002	000001	2021-06-24	2021-07-01	500.00	500.00	EUR
200	00004142	070001	000001	2021-06-24	2021-07-01	0.00	0.00	
200	00000003	070046	000093	2021-02-27	2021-12-26	80.00	4164.00	USD
200	00000004	070042	000665	2021-02-27	2021-12-26	40.00	1871.00	USD

Figure 89 Display of the Data Preview

You have various options in the view to adjust the selection. These include:

- Number of selected rows
- Selected columns
- Filter
- Sorting (clicking on the column header)

In addition, you can search for a pattern in the displayed data (incl. ? and * as a joker sign). Matching data is then highlighted in color and bold. In addition, the total number of affected entries and a log of actions executed can be viewed. The Save button allows you to save the displayed values in different formats within a file. It is even possible to generate an ABAP value statement, which can be very useful for creating test data.

In the case of CDS views with associations , it is possible to follow the associations and thus display the linked data. To do this, select one of the data records and select the desired association using the arrow at the top.

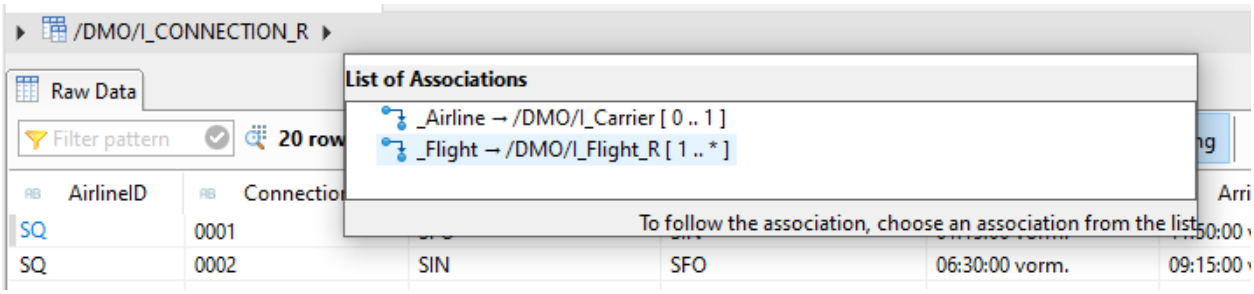


Figure 90 Navigation via Associations

A highlight of the Data Preview View is the SQL Console. Based on the selected columns, specified filters and the collation, an SQL Select Statement is generated, which is then used to select the data.

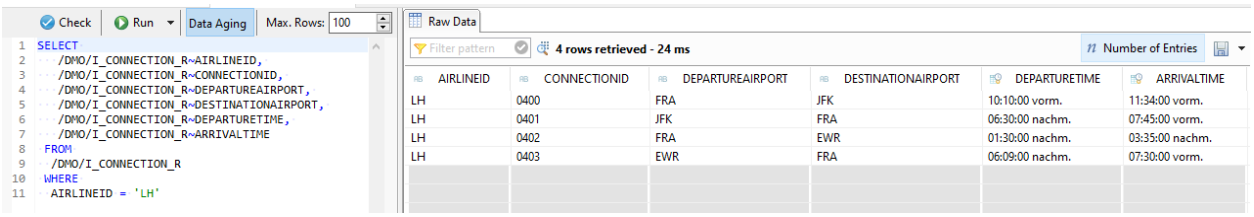


Figure 91 SQL Console

This SQL statement can be customized, checked and executed. The following rules and restrictions apply:

- Only SELECT statements according to ABAP Open SQL syntax are allowed.
- It is possible to create aggregations and complex selections, e.g. with JOIN and UNION.

- Only read-only accesses are possible (no SQL statements with data changes)
- Keywords related to internal tables cannot be used

You can also call the SQL Console directly by opening the context menu for the ABAP project in the Project Explorer and selecting the SQL Console menu item. The view displays the most recently used SQL statement and executes it immediately.

All in all, the SQL Console is a powerful tool that can be used to easily execute selections, evaluate data or test adjustments to selects.

Details about the Data Preview can be found in the Eclipse Help for ADT.

3.2.11 Core Data Services

In the area of Core Data Services (CDS), there are different file types that can be created in ADT:

- Data Definitions (DDLS) - Source code files for data models in the following variants:
  - DDIC Based CDS Views
  - CDS View Entities
  - Abstract CDS entities
  - Hierarchies
  - Extensions of the views
  - CDS Table Functions
- Access Control Files (DCLS) - Access Definitions
- Metadata Extensions (DDLX) - Outsourcing of annotations from the CDS definition
- Behavior Definitions (BDEF) - Behavior Definitions for RAP Business Objects

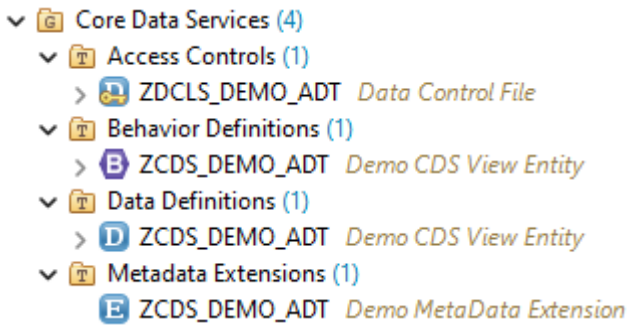


Figure 92 Object Types of the CDS in Navigation

The source code editors for the different file types of Core Data Services behave largely like the source code editor for ABAP code. Among other things, the following features are included:

- Code Completion (**`CTRL+SPACE`**) - Default values that fit in context.
- Element Info (**`F2`**) - Information about the element on which the cursor is placed.
- Pretty Printer (**`SHIFT+F1`**)

One difference is the colors used in the editor.

A basic problem with Core Data Services is that the properties of an object (e.g. a CDS View Entity) are composed of several files and the properties of the data sources. These files each declare their affiliation, and the data sources propagate their field properties (annotations). This is very practical for the expansion concept. But transparency suffers because the files are not necessarily in the same development package.

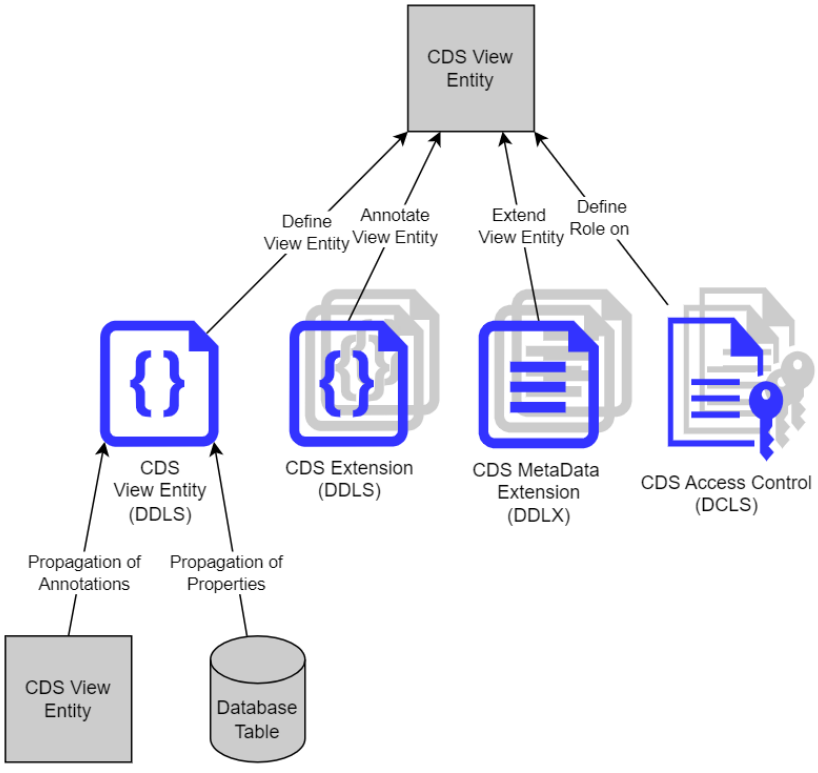


Figure 93 Different Files Define the Properties of a CDS View Entity

In order to get a complete picture of a CDS object, taking into account all files and propagations, tools are therefore necessary. These include:

- Element Info
- Dependency Analyzer
- Active Annotations

3.2.11.1 Item Info for CDS

With **F2** or the separate Element Info View, you get a good overview of the data structure and associations for a CDS View, regardless of where they were defined. It also displays all the relevant extension files.

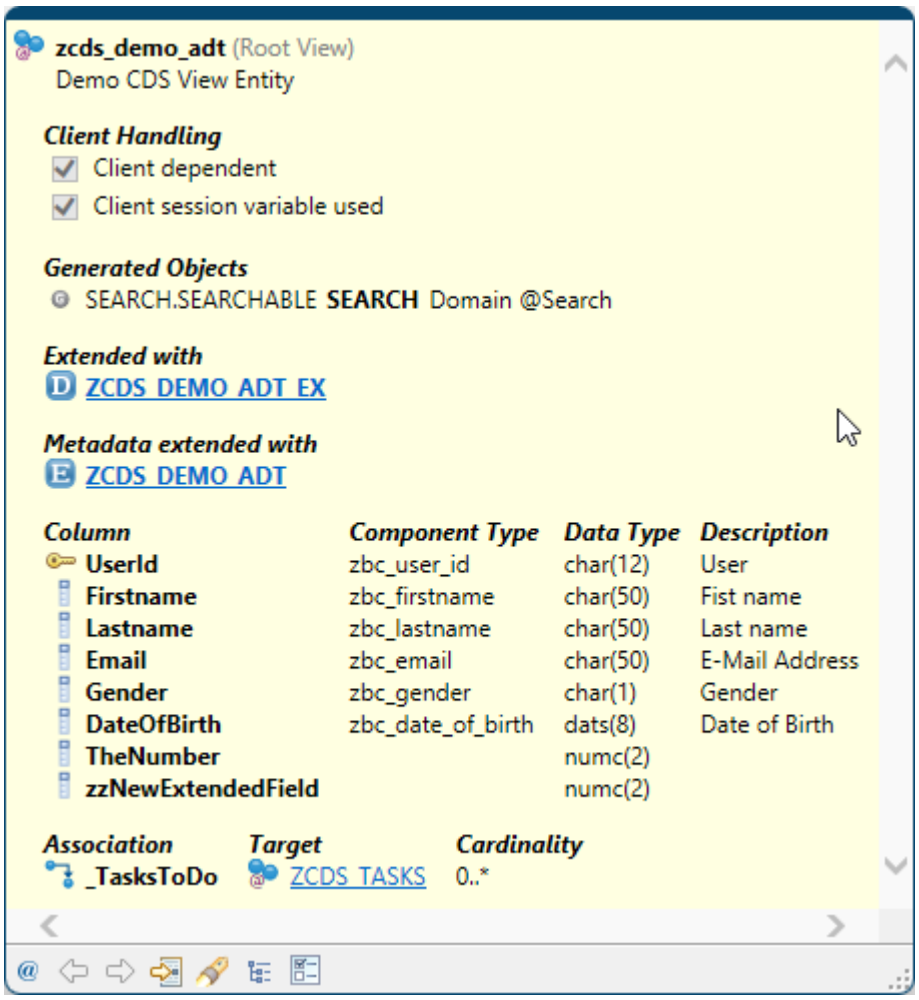


Figure 94 Overview of a CDS View Entity Using Element Info

3.2.11.2 Dependency Analyzer

The Dependency Analyzer provides a good overview of the origin of the data. It is accessed via the context menu.

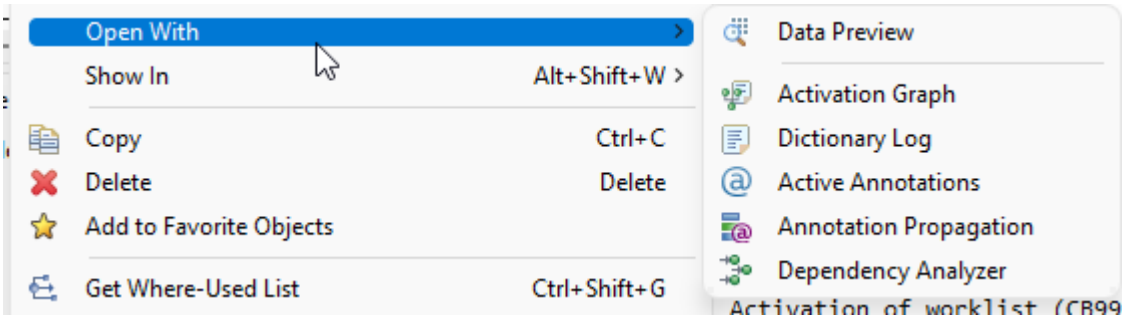


Figure 95 Calling the Dependency Analyzer via the Context Menu

The Dependency Analyzer has three tabs that display information about a view:

- The **SQL Dependency Tree** displays the hierarchical structure in tabular form

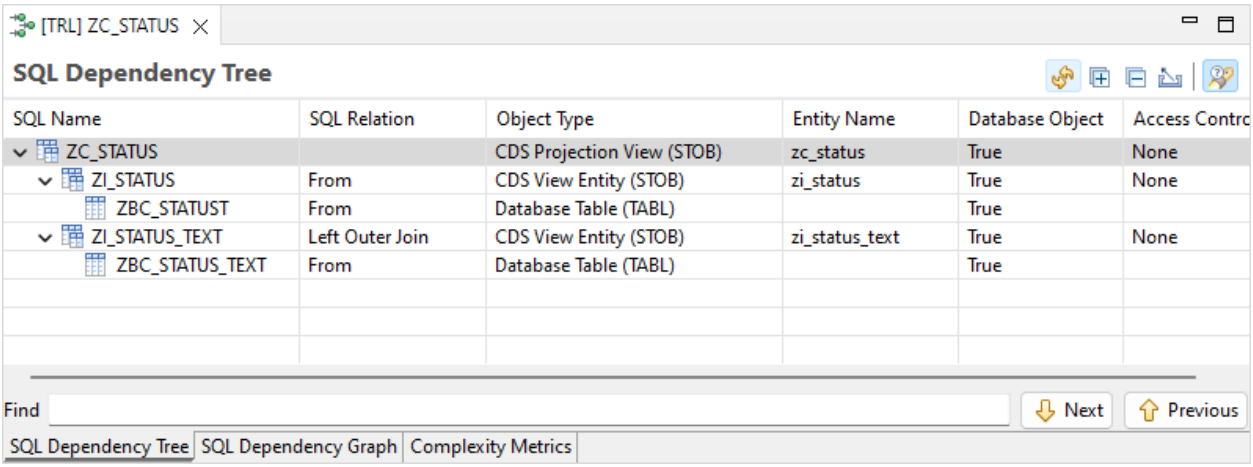


Figure 96 SQL Dependency Tree

- The **SQL Dependency Graph** displays the same information graphically.

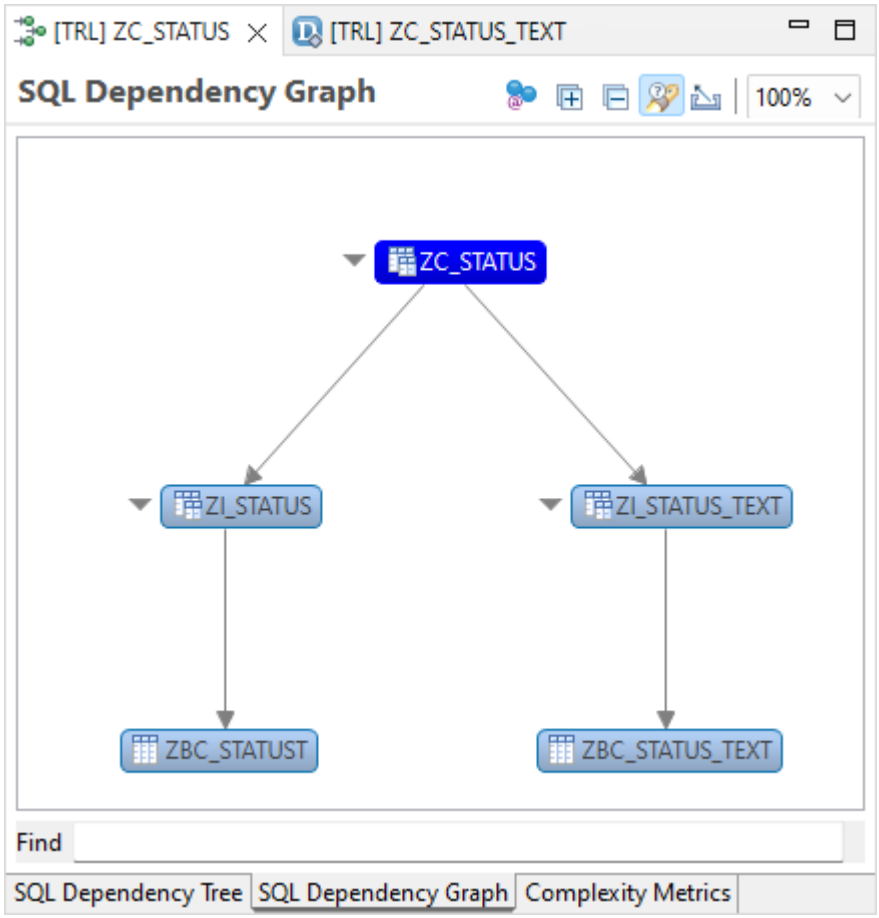


Figure 97 SQL Dependency Graph

- The **Complexity Metrics** tab displays additional information on the overall complexity of the CDS view, including all source views.

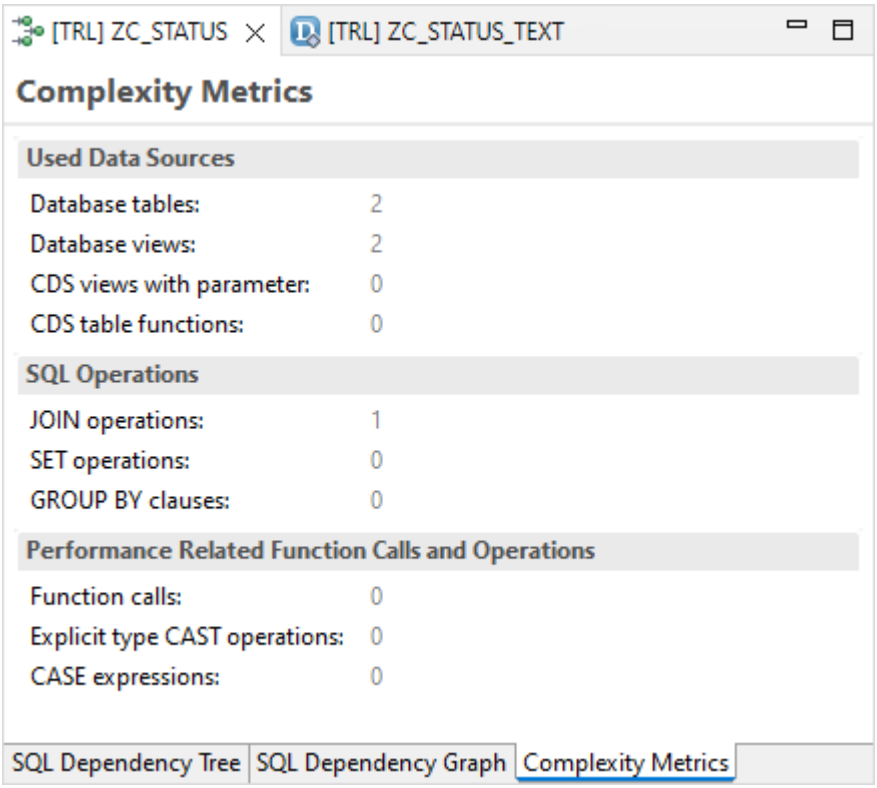


Figure 98 Complexity Metrics

3.2.11.3 Active Annotations

The Active Annotations view is also accessed via the context menu in the navigation. It displays the values of all active annotations of the views:

- At the view level **, only exactly the annotations that have been defined in the view are visible.**
- At **the field level** , all valid annotations are visible. Their origin, such as data element, metadata extension or data source, is also displayed.
- At **parameter level**

Some properties, e.g. field texts, can already be defined by the data elements and then propagated to the view. This is exemplified in the following figure:

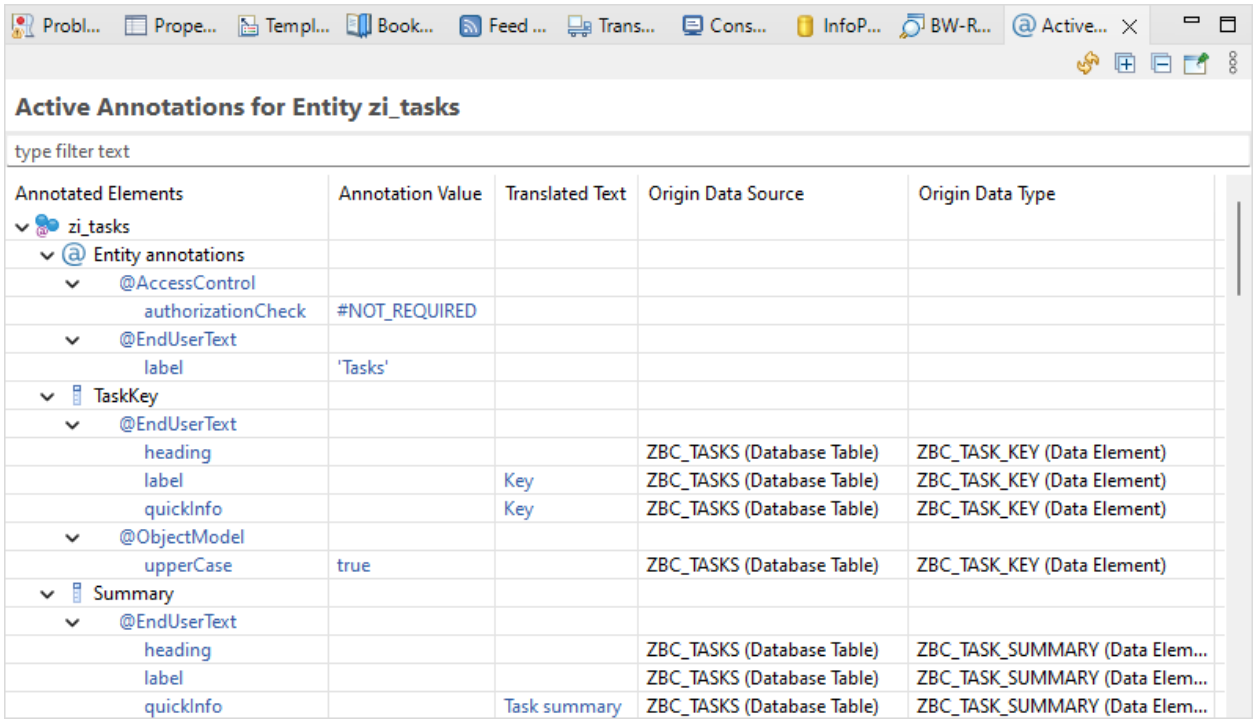


Figure 99 Active Annotations of a View

4 Troubleshooting Tools in Eclipse

This chapter provides an overview of the tools that are available for troubleshooting in the ABAP Development Tools . This includes, but is not limited to, debugging development artifacts, performance analysis, and other analysis and debugging methods.

In addition to the description of the tools, we provide tips for their sensible use. For detailed functional descriptions, we recommend the official documentation and, if available, provide helpful links.

4.1 Troubleshooting with the ABAP Development Tools

Anyone who has managed to get started with the ABAP development tools and appreciates the advantages of ADT in development is reluctant to continue developing in an SAP GUI-based environment. But the ABAP development tools are not just an environment for creating and modifying code. ADT contains numerous tools for the general analysis of code, the finding of errors and the detailed analysis of the performance of the functionalities. In summary, these activities can be referred to as troubleshooting.

But as with the entry into development with ADT, there is a steep learning curve to overcome with the troubleshooting tools, the conquest of which is rewarded with numerous advantages. As an example, it should be mentioned here that it is possible to change the faulty code directly in the debugger instead of cumbersomely opening a new mode in parallel, searching for the relevant location and then making the change. There are many advantages to discover in the troubleshooting tools, which are described here in the individual sections.

In addition to the direct advantages, the aspect of "Eclipse ADT as the same tool for everything" should not be neglected in the area of troubleshooting.

For example, if you are well versed in the SE80 debugger, it will be more difficult to make the switch, as the advantages of the ADT debugger must first be worked out. For the development process as a whole, however, it makes sense to use one tool for all use cases and to reduce or completely avoid switching back and forth between environments.

4.2 The Debugger in ABAP Development Tools

In Eclipse, a debugger is available for analysis purposes. This is started via the button . When you start it, the perspective in Eclipse automatically changes to the perspective of the debugger.

4.2.1 Breakpoints and Soft-Breakpoints

Breakpoints in Eclipse are exclusively external breakpoints. The breakpoints are drawn each time the software is run. The process is interrupted at the appropriate point.

Breakpoints can be set in all perspectives in the editor to the left of the line number. Alternatively, breakpoints can be set via the context menu. Set breakpoints are indicated by a blue dot next to the line of code.

In addition to the normal breakpoints, the ABAP development tools offer the possibility of soft breakpoints. These are set via the context menu and marked with a green dot. In contrast to the standard breakpoints, the program flow is only stopped at this point if the software is running in the debugging context. Otherwise, soft breakpoints are skipped.

4.2.2 Debugging Perspective

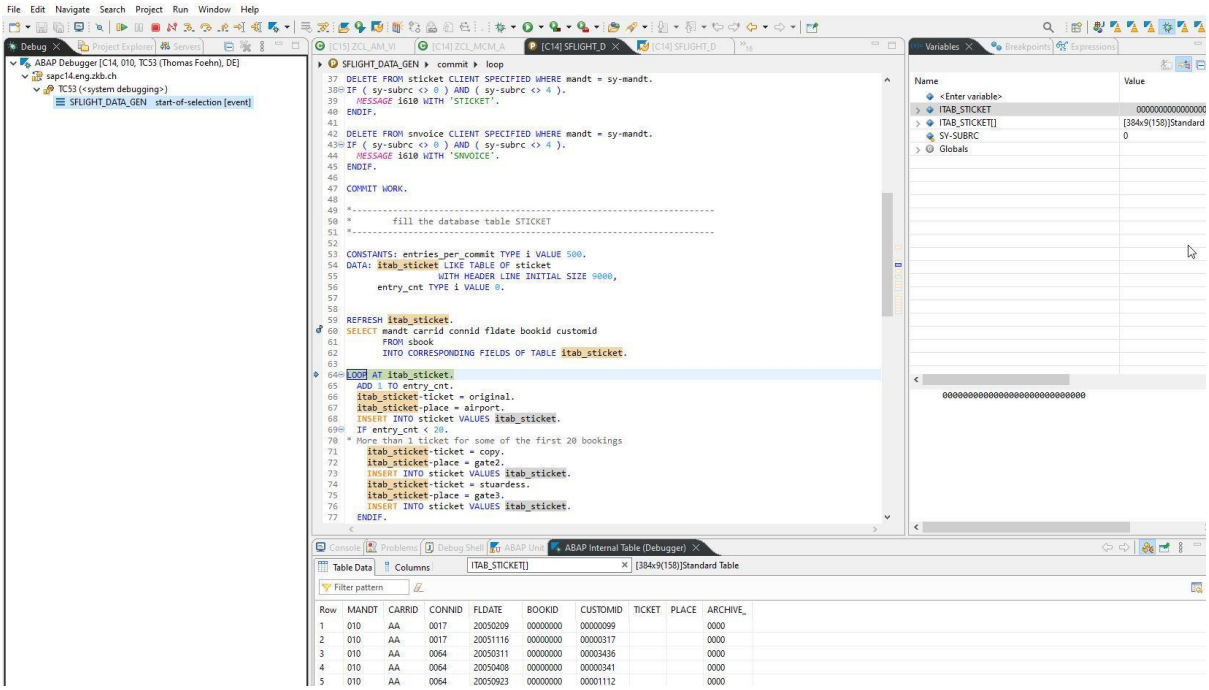


Figure 100 Debugging Perspective in Eclipse

The debugging perspective in Eclipse provides a quick overview of the program code, call stack, variable contents, and contents of internal tables. The variables and internal tables can be selected by double-clicking in the program code. They are displayed on the right side.

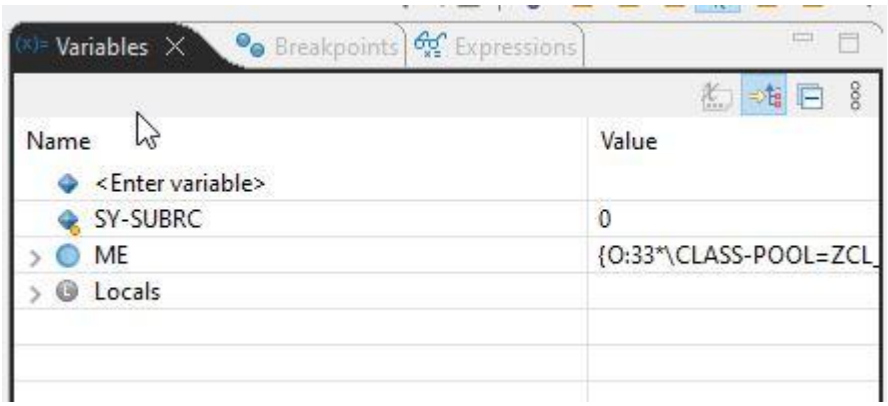


Figure 101 Values of the Variables in the Debugging Perspective

4.2.3 Special Behavior in the Debugger

In the debugger of the ABAP Development Tools, it is possible to modify and activate the code directly. However, in the current debugging context, the coding is not yet active. The program run must be restarted.

4.2.4 Additional Information

Further information about debugging with the ABAP Development Tools in Eclipse can be found in the SAP blogs. The following two should be emphasized. They describe cases and solutions that can occur when working with the debugger.

https://blogs.sap.com/2020/04/21/adt-abap-debugger-what-to-do-if-your-program-does-not-stop-at-breakpoints/

https://blogs.sap.com/2015/11/02/breakpoint-validity-scope-and-activation-conflicts-in-abap-development-tools-adt/

4.3 Checkpoint IDs and dynamic logpoints

A very helpful tool in the field of error analysis and debugging are the so-called checkpoint IDs. These can be created via transaction SAAB or in ADT under "others". These IDs are defined using the following commands:

- BREAK POINT ID [GROUP NAME]
- LOG POINT ID [GROUP NAME]
- ASSERT ID [GROUP NAME]

anchored in the code. For the detailed syntax and options of the commands, please refer to SAP Help.

As in the debugger of the SAP GUI, these dynamic breakpoints can be activated for debugging or used for logging. The main advantage here is that the developer can add breakpoints to important places in the code in advance. If the code is to be analyzed, the Checkpoint ID only needs to be activated once. When the unit is called, the debugger is called at the appropriate point if the breakpoint is active.

In order to use the checkpoints effectively, it is advisable to create appropriate templates, which can then be easily called up using quick fixes (see Chapter 3 - Working with ADT in the section on templates).

While checkpoint IDs can also be used in the GUI-based debugger, ADT for on-premise systems also offers the possibility to set dynamic log points in the debugger, which can be used to read internal program values. This option is useful if it is not possible to change the productive code or if an analysis has to be carried out very promptly on the production system.

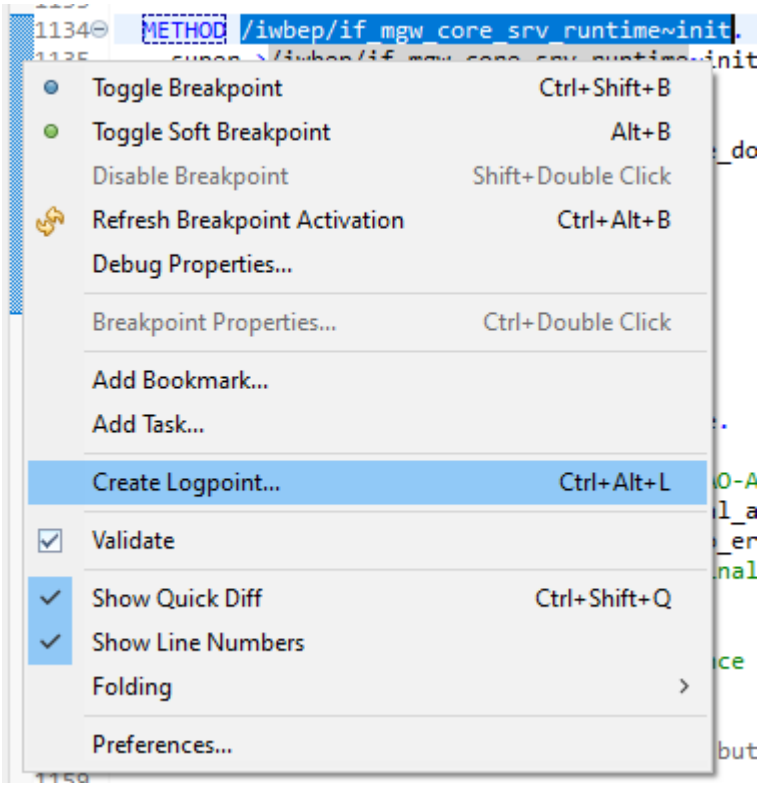


Figure 102 Creation of a Log Point via the Context Menu

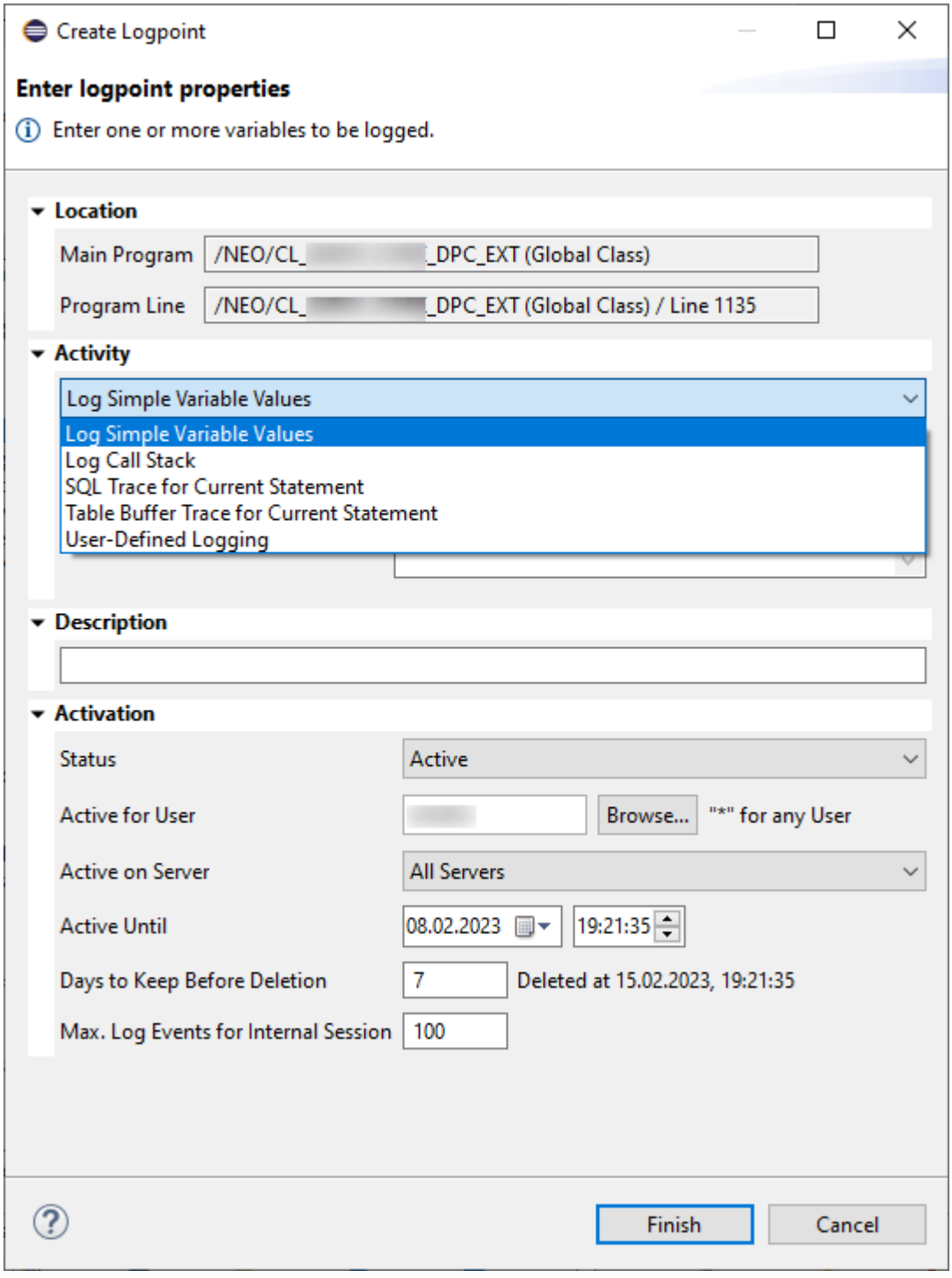


Figure 103 Attributes when Creating a Log Point

In the dialog, you can decide what should be recorded in the log, you can give the dynamic logpoint a description that will then be used in the log output, and you can specify various criteria for whether (based on a condition – hidden in the screenshot – and/or user/server) and how long the log output should take place. Created logpoints

are displayed in the editor on the left margin and on the right margin next to the vertical scroll bar and listed in the "Logpoints" view:

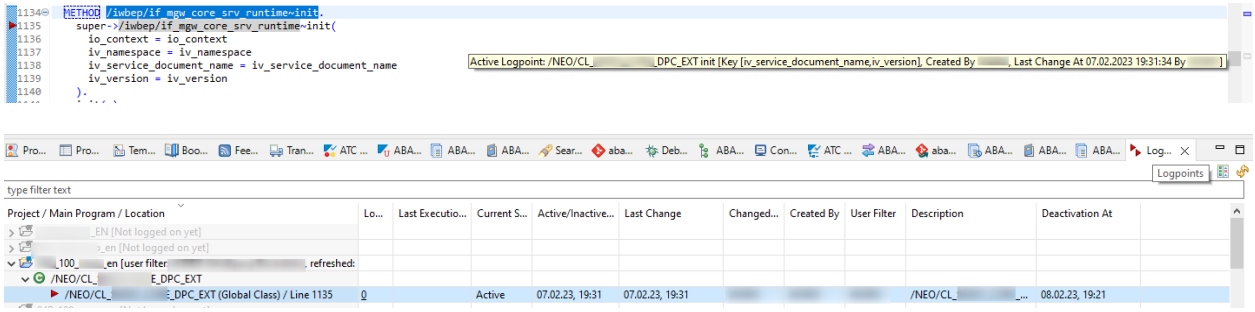


Figure 104 Log Points View in the Debugging Perspective

We recommend that you refer to the SAP documentation (On-Premise), to know details about the application. This blog entry also offers a good introduction to this: Dynamic Logpoints in ABAP | SAP-Blogs

4.4 Performance Analysis

For an integrated and graphical performance analysis, ADT offer you convenient access to the ABAP Profiler (on-premise/cloud) as a successor to transaction SAT, among others. There are several ways to start the profiler:

If you have opened an executable program (on-premise only), a console application (ABAP Cloud only), or an artifact with assigned unit tests, you can start the profiler directly from the "Profile as" context menu. Alternatively, you can also use the wizard, which can be accessed via the Run menu.

In case you need a different starting point, you can use trace requests to create requests for the start of the profiler. To do this, first display the "ABAP Trace Requests" view (which can be found in the view list below ABAP, see Views and Perspectives in Chapter 3 - Working with ADT). In this view, after selecting a system with the Create-Trace-Request-Icon, you get a wizard. This allows you to select different triggers for the start of the profiler. If the system is accessed directly with HTTP(S) accesses (note: this is usually not the case in a hub/FES configuration in the backend), a pattern can be used for the URL, e.g. the name of the OData service with a leading and trailing asterisk. Furthermore, an RFC call of a function module or the start of a background job and various other triggers can be used to start the trace. In SAP Fiori development, the function module /IWBEP/FM_MGW_HANDLE_REQUEST can be used as a trigger in a hub/FES configuration for OData accesses in the back-end system. The number of trigger activations can be limited, as well as there is a possibility for time

limitation. You can use various settings to control the scope of data collection. Trace requests can be deleted via the context menu in the list.

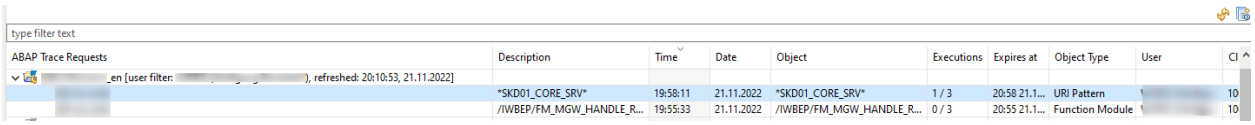


Figure 105 ABAP Trace Requests in the Debugging Perspective

In addition, there is the possibility to start the trace from the ADT debugger (on-premise/cloud).

In the case of trace requests, the number of traces already generated per request can be updated via the update icon in the trace request view. Via the context menu or by double-clicking on it, you can jump to the view with the list of traces. From there, the display of a trace can be opened. You can either use the context menu to jump to the various tabs of the analysis or to the overview page by double-clicking.

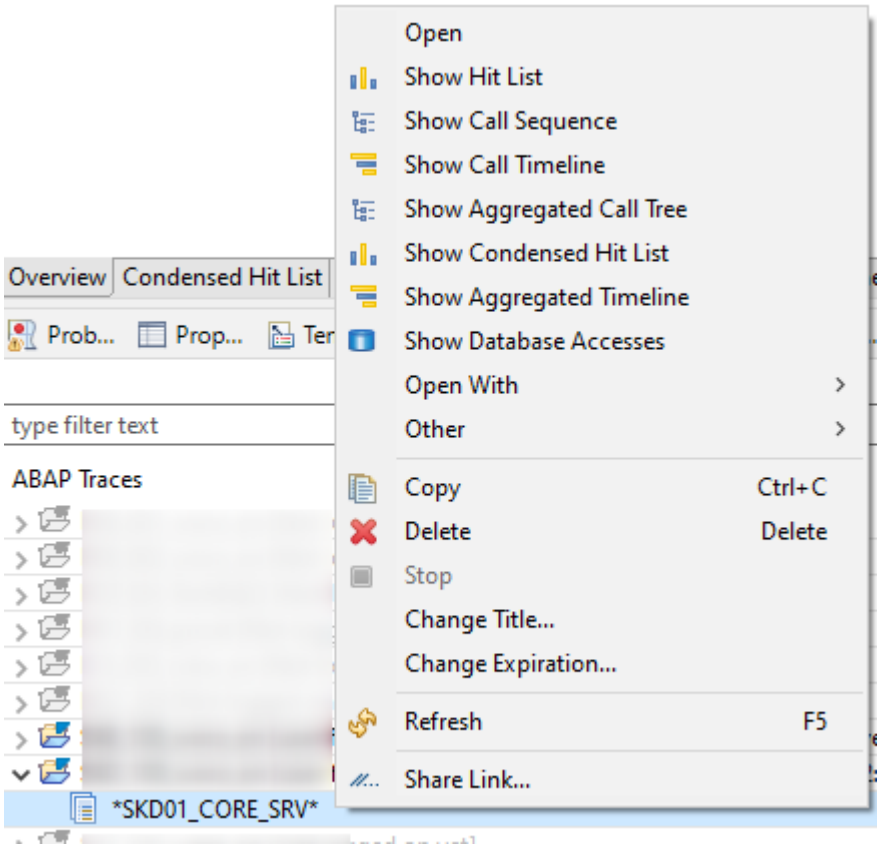


Figure 106 Context Menu of a Trace

In addition to a brief overview of the runtime, the overview page also offers direct jumping points into the various tabs.

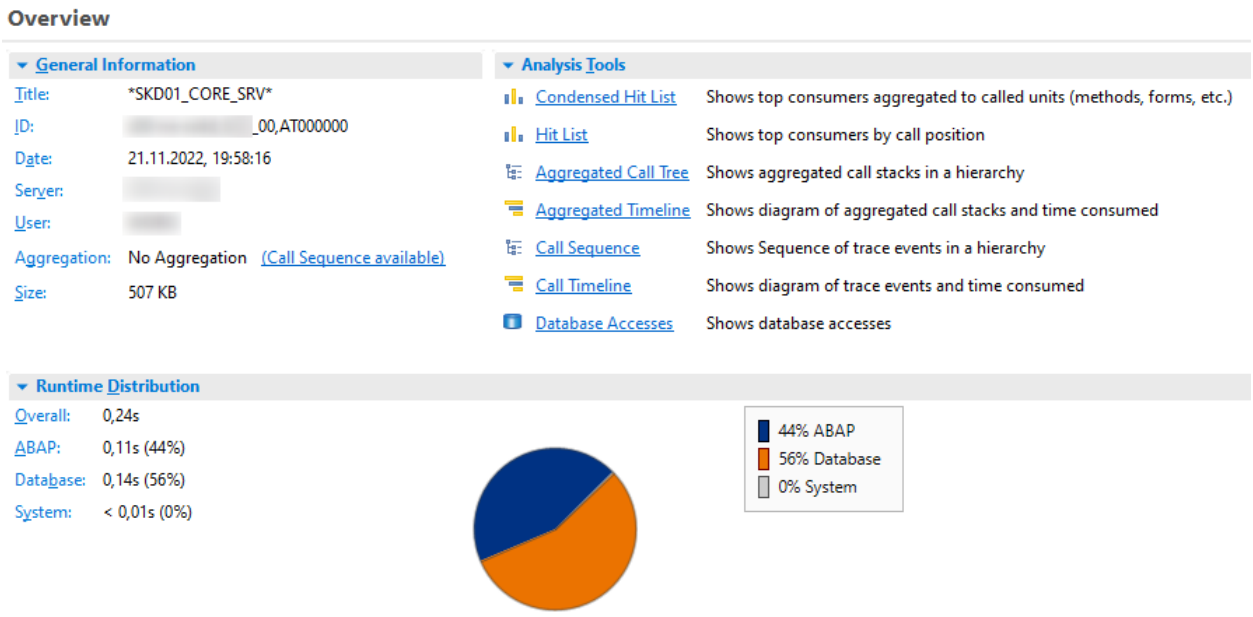


Figure 107 Overview of the Properties of a Trace

In particular, the graphical analysis of the timeline is only available in ADT and facilitates analysis. If you move the mouse pointer over the blocks, you will be shown details directly and can also navigate directly to the source code via the context menu.

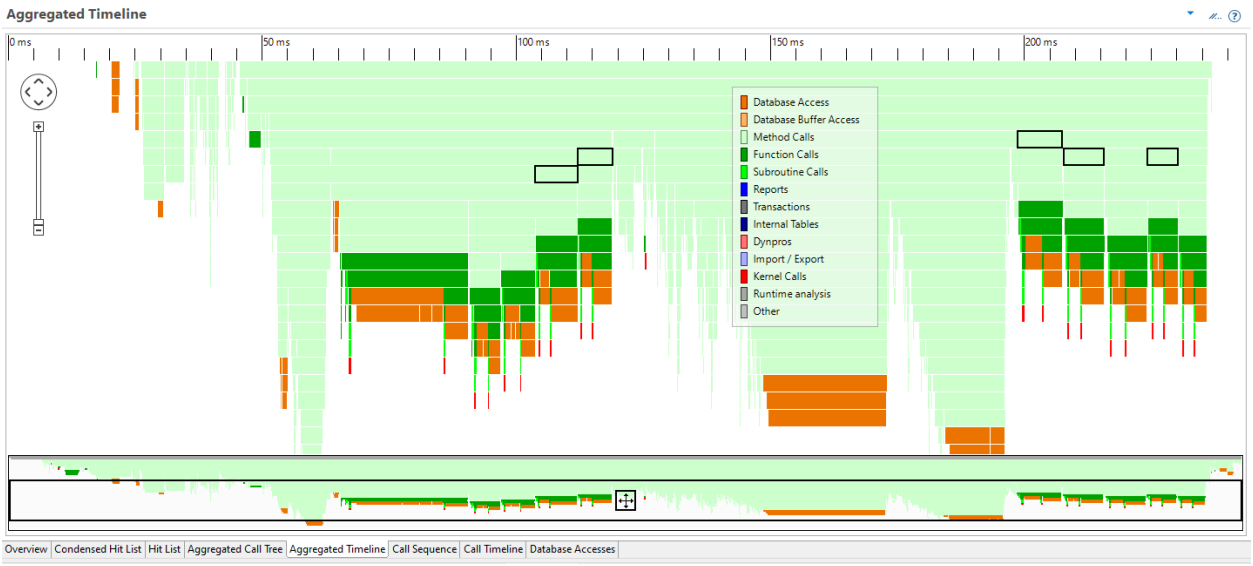


Figure 108 Aggregated Overview of a Trace History

The SQL trace, which provides the PLV files for a visual analysis of the query plans, especially in the case of HANA as a database, can be started in the context menu of a system in the Project Explorer (also works in the ABAP environment). However, the results are then displayed in a Web application outside ADT (or in transaction ST05).

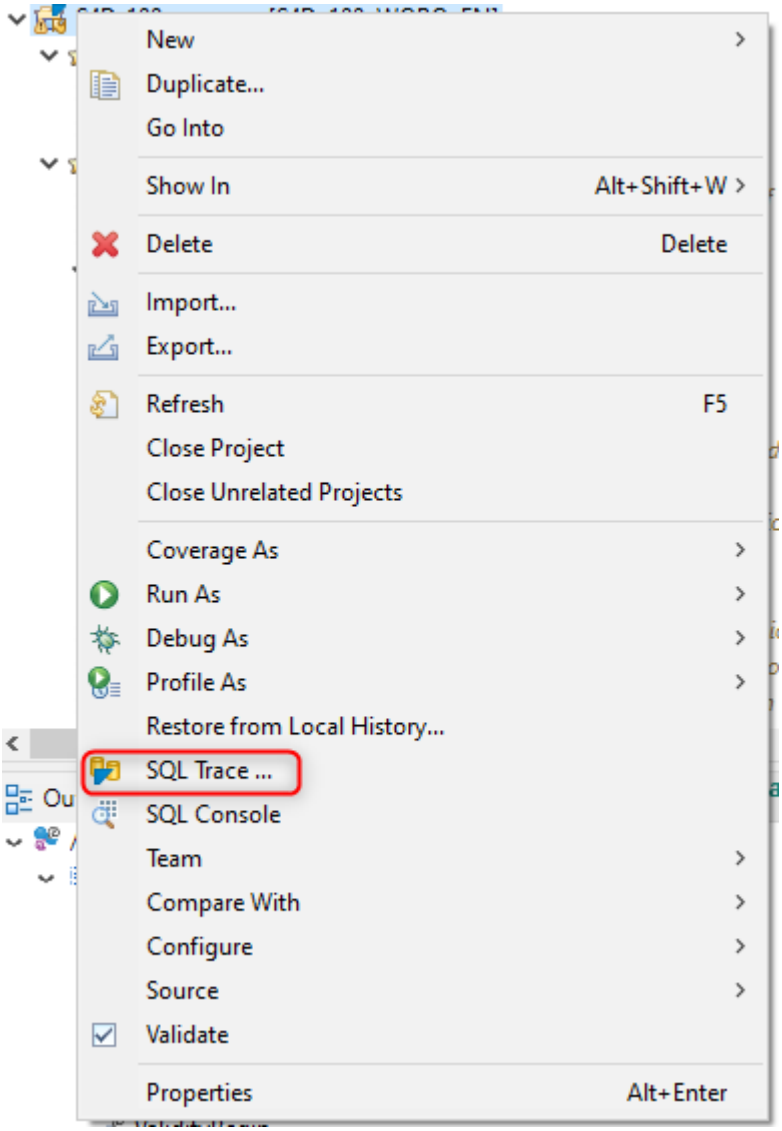


Figure 109 Jump to the SQL Trace

For visual analysis, you can currently also install the SAP HANA administration tools from the HANA Studio in ADT Eclipse (or the complete SAP HANA Studio in parallel) and thus configure the automatic start of the visual analysis from the transaction. To do this, set the user parameter HDB_OPEN_STUDIO to X

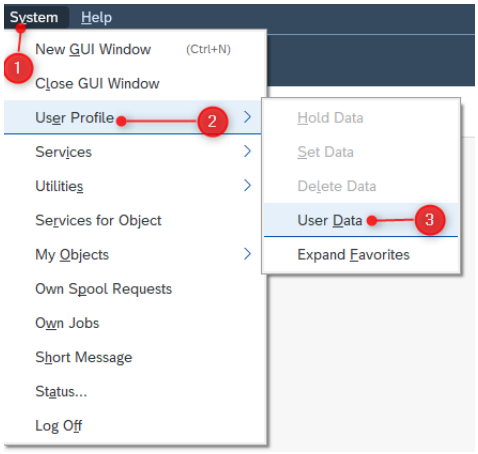


Figure 110 Getting started with managing User Parameters

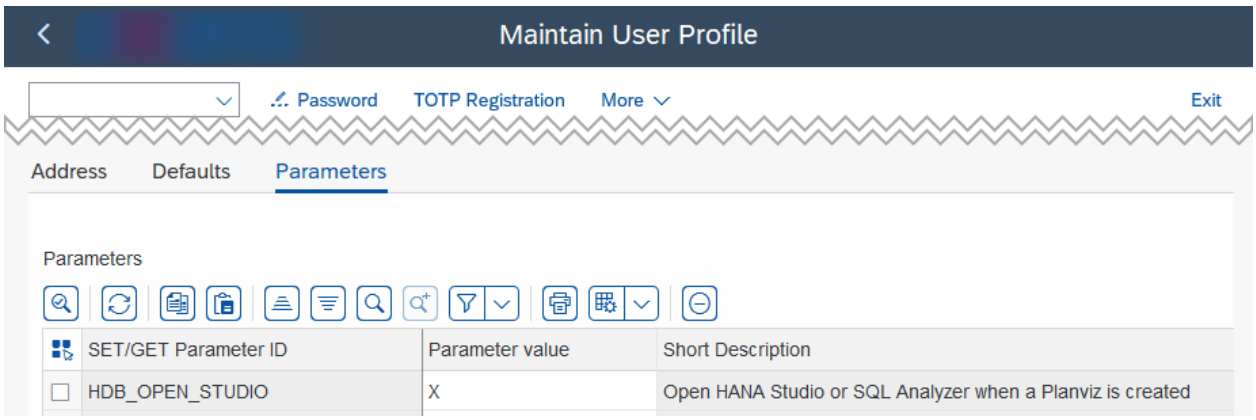


Figure 111 Setting the User Parameter HDB_OPEN_STUDIO

and associate opening *.plv files with the right eclipse .exe in the operating system. For example, on Windows, there is the option "Open with..." in the context menu of the file in File Explorer and there you will find the option "Find another app on this PC" at the bottom, after clicking you have to search for and select the "eclipse.exe" of your Eclipse/ADT installation as well as the option "Always this app ... use".

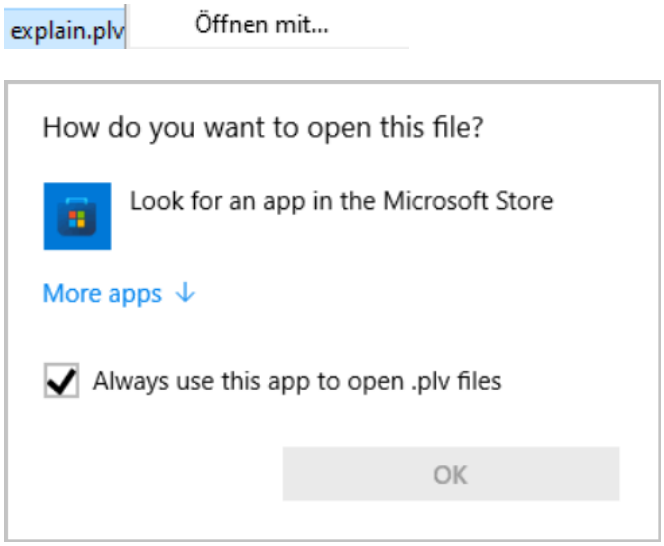
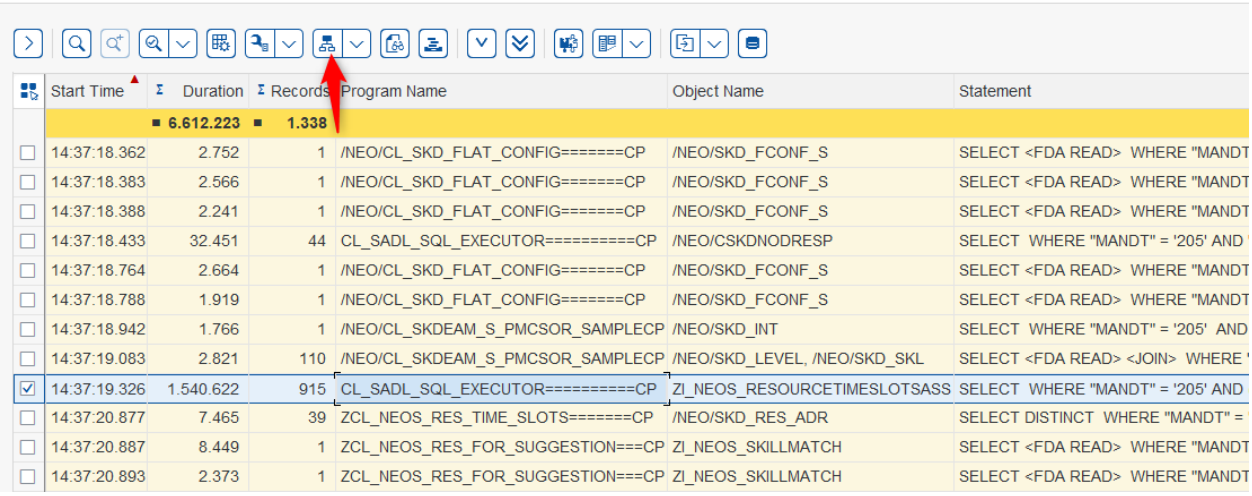


Figure 112 Selection of eclipse.exe as a New Way to Display *.plv Files

After selecting a row in the SQL trace list of the ST05 or selecting a trace record and an SQL statement in the SQL trace analysis of the Technical Monitoring Cockpit, you can request the HANA PlanViz Query Plan visualization:



Start Time	Duration	Records	Program Name	Object Name	Statement
	6.612.223	1.338			
14:37:18.362	2.752	1	/NEO/CL_SKD_FLAT_CONFIG=======CP	/NEO/SKD_FCONF_S	SELECT <FDA READ> WHERE "MANDT
14:37:18.383	2.566	1	/NEO/CL_SKD_FLAT_CONFIG=======CP	/NEO/SKD_FCONF_S	SELECT <FDA READ> WHERE "MANDT
14:37:18.388	2.241	1	/NEO/CL_SKD_FLAT_CONFIG=======CP	/NEO/SKD_FCONF_S	SELECT <FDA READ> WHERE "MANDT
14:37:18.433	32.451	44	CL_SADL_SQL_EXECUTOR==========CP	/NEO/CSKDNODRESP	SELECT WHERE "MANDT" = '205' AND
14:37:18.764	2.664	1	/NEO/CL_SKD_FLAT_CONFIG=======CP	/NEO/SKD_FCONF_S	SELECT <FDA READ> WHERE "MANDT
14:37:18.788	1.919	1	/NEO/CL_SKD_FLAT_CONFIG=======CP	/NEO/SKD_FCONF_S	SELECT <FDA READ> WHERE "MANDT
14:37:18.942	1.766	1	/NEO/CL_SKDEAM_S_PMCSOR_SAMPLECP	/NEO/SKD_INT	SELECT WHERE "MANDT" = '205' AND
14:37:19.083	2.821	110	/NEO/CL_SKDEAM_S_PMCSOR_SAMPLECP	/NEO/SKD_LEVEL, /NEO/SKD_SKL	SELECT <FDA READ> <JOIN> WHERE
14:37:19.326	1.540.622	915	CL_SADL_SQL_EXECUTOR==========CP	ZI_NEOS_RESOURCETIMESLOTSASS	SELECT WHERE "MANDT" = '205' AND
14:37:20.877	7.465	39	ZCL_NEOS_RES_TIME_SLOTS=======CP	/NEO/SKD_RES_ADR	SELECT DISTINCT WHERE "MANDT" =
14:37:20.887	8.449	1	ZCL_NEOS_RES_FOR_SUGGESTION===CP	ZI_NEOS_SKILLMATCH	SELECT <FDA READ> WHERE "MANDT
14:37:20.893	2.373	1	ZCL_NEOS_RES_FOR_SUGGESTION===CP	ZI_NEOS_SKILLMATCH	SELECT <FDA READ> WHERE "MANDT

Figure 113 Selection of a Specific Selection

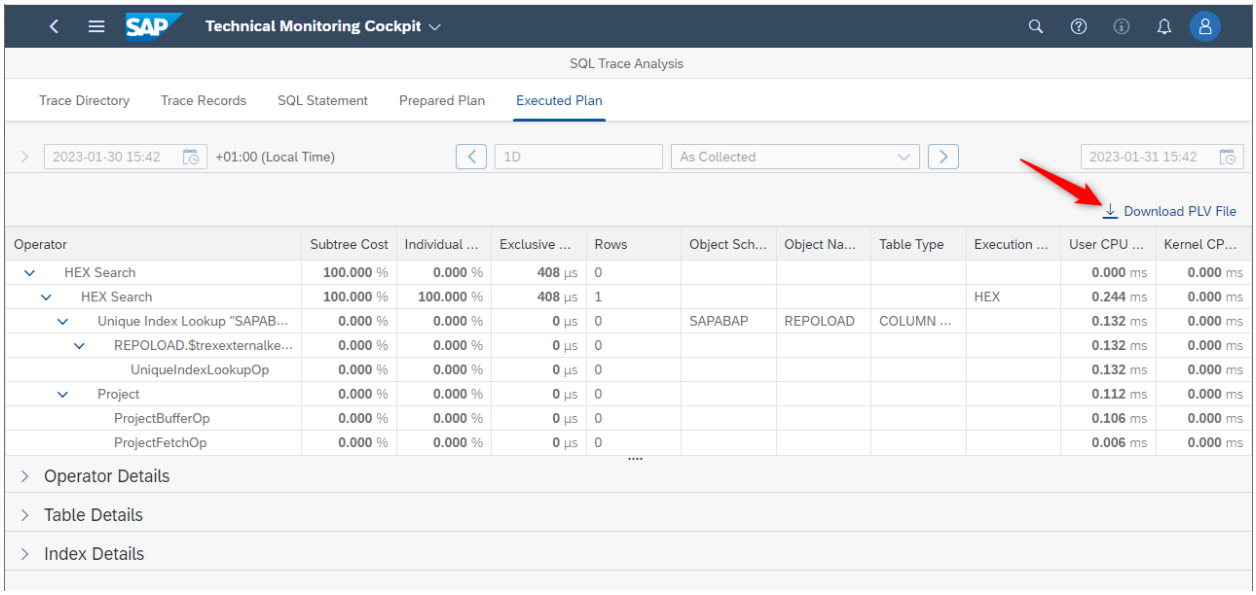


Figure 114 Download the *.plv File

Eclipse then automatically starts the correct view, and the Tab Executed Plan shows the visual analysis of the query plan.

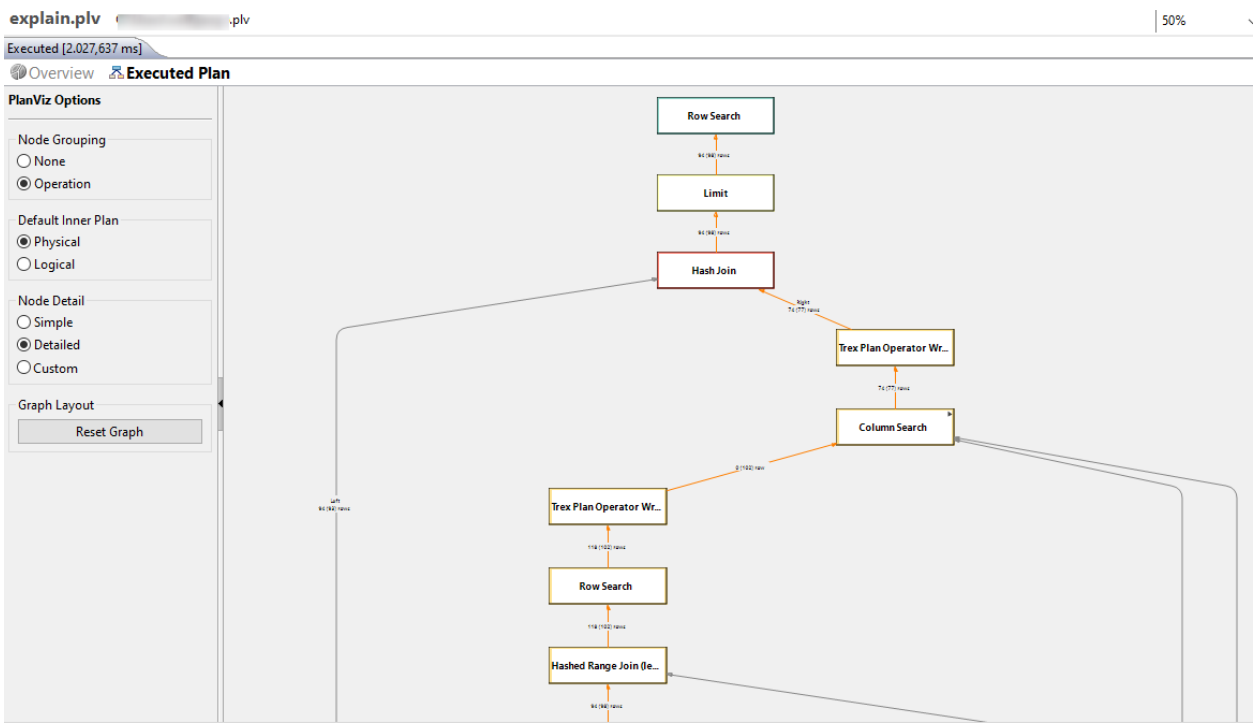


Figure 115 Query Execution Plan Display

For development with the RAP (ABAP RESTful Application Programming Model), there is a special trace tool called Cross Trace (Doku On-Premise/Cloud), which you can use to analyze requests from Fiori apps across the RAP software stack (SAP Gateway, BO Behavior, SADL, ABAP Core). A corresponding separate authorization is required for use.

To start, display the "ABAP Cross Trace" view. This view has two tabs. In the first tab, you can create a new cross-trace configuration in the context menu of a system. A cross-trace configuration can be provided with a description to distinguish it, can be active or inactive, automatic deactivation after a certain number of trace requests is possible, as well as automated deletion at a certain point in time. You can decide whether non-sensitive or sensitive data should be recorded. Optionally, you can filter by user, access type and target (e.g. only a specific OData service; * is also possible here for wildcard filtering), as well as you can specify whether and with which trace level a recording should take place for the respective cross-trace components.

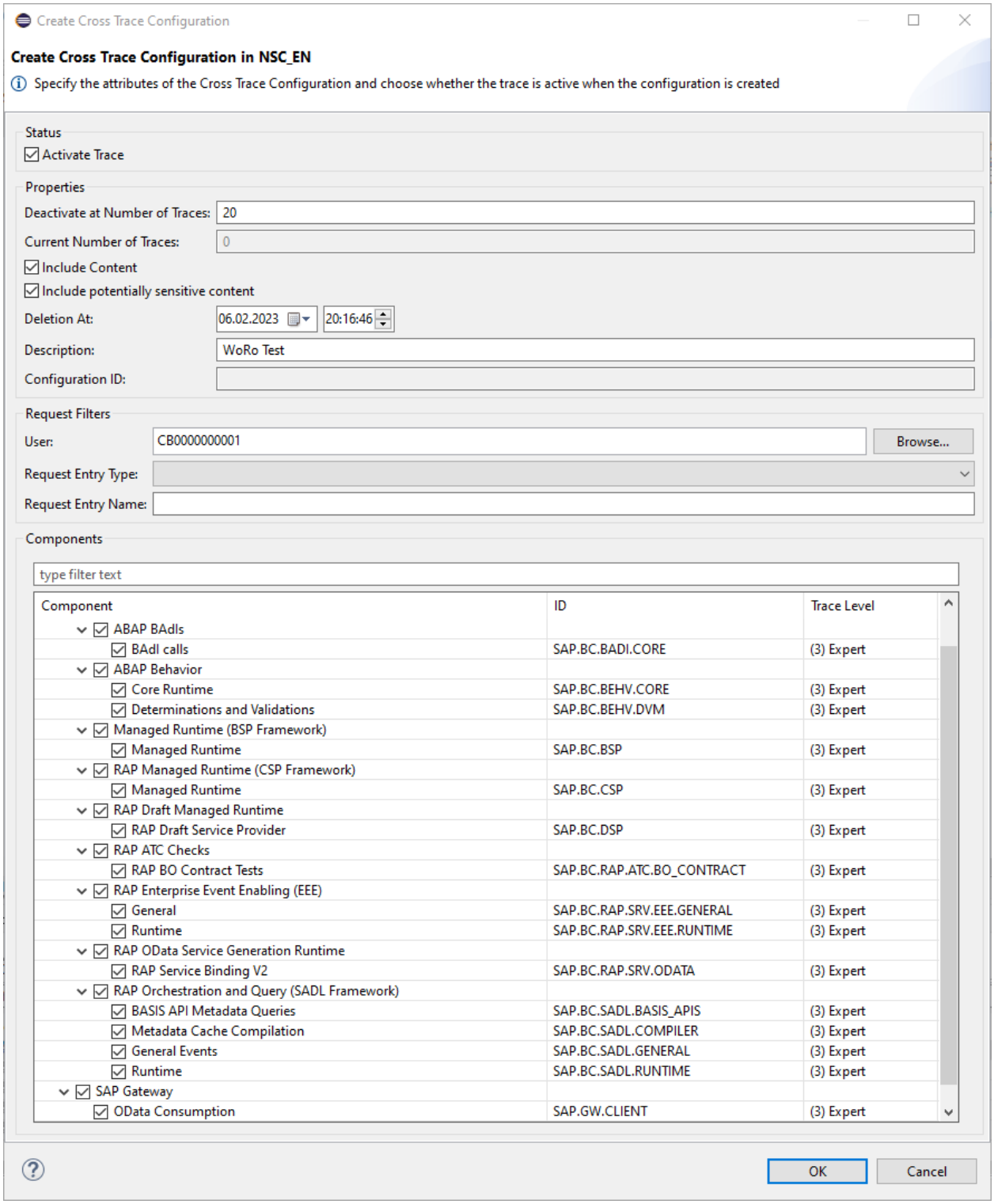


Figure 116 Creation of ABAP Cross Traces

After confirming with OK, the configuration is displayed in the view, where the current status (active/inactive), the description and the number of remaining accesses to be recorded can also be viewed. In the context menu, a configuration can be edited and activated/deactivated/deleted/updated. A global update is possible in the upper right corner of the view. To display the results, switch to the second tab of the view. Here you can see a trace for each individual access. In the context menu, you can open a trace (and delete it, etc.). The trace then opens in another view. There you can see the path of access through the individual components and filter the displayed lines or start a text search. You can use the offset column to understand the time course of access. Details are displayed for each row in the Properties View.

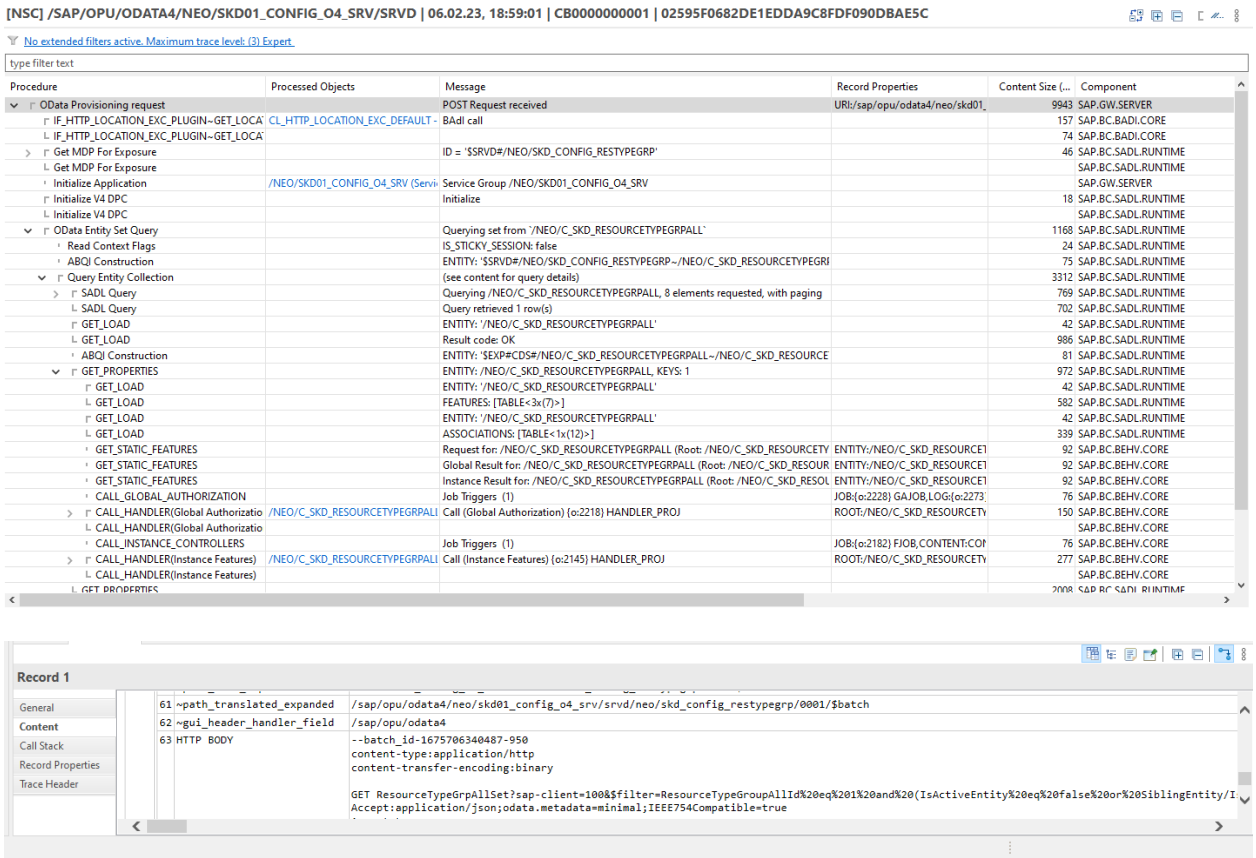


Figure 117 Detailed View of Operations

You can jump directly to the triggering line of source code, view the call hierarchy, and so on.

4.4.1 Note on HANA Studio and SAP HANA Tools

HANA Studio is only developed to a limited extent by SAP. A future solution for the visual analysis of query plans within ADT or for the ABAP Cloud is still pending. As a solution outside of Eclipse, there is a Visual Studio Code plug-in that can also open *.plv files and display them graphically, cf. SQL Analyzer Extension.

The SAP HANA tools follow a different release cycle than ADT and are therefore often not included in the current or the "latest" update site. For example, in January 2023, the latest version of SAP HANA tools was only available on the https://tools.hana.ondemand.com/2022-09 site.

4.5 Feed Reader

Feeds enable event-related notifications in ADT including access to a list of previous events. The feeds are displayed in a separate view (see Views and Perspectives in Chapter 3 - Working with ADT). These can be found in the list of views below ABAP with the title "Feed Reader". The individual sources are called feeds and for each feed you can set how often it should be updated by the source and whether notification messages should be displayed in Eclipse.

In addition to the ABAP runtime errors/dumps and system messages described in the documentation under "Getting Feeds" (On-Premise/Cloud), the following information in particular can also be displayed in the Feed Reader (depending on the release status of the source system):

- Gateway/OData errors (can be filtered in a variety of ways, including users)
- ATC results (can be filtered in a variety of ways, including users)
- Enterprise event errors (filterable by channel and user)
- BW Job Repository (can be filtered in a variety of ways, including users)
- URI Creation Error (only relevant for ADT developers)

For this purpose, the events in the pull procedure are queried in the background. However, for the background query from SAP systems to work, you must have accessed the desired systems at least once in some form after an Eclipse start (you can also trigger this by clicking on a feed) and have gone through the login procedure.

In addition, any Atom/RSS feeds can also be subscribed to, this can be, for example, the RSS feed for the last blog posts on a day on blogs.sap.com:

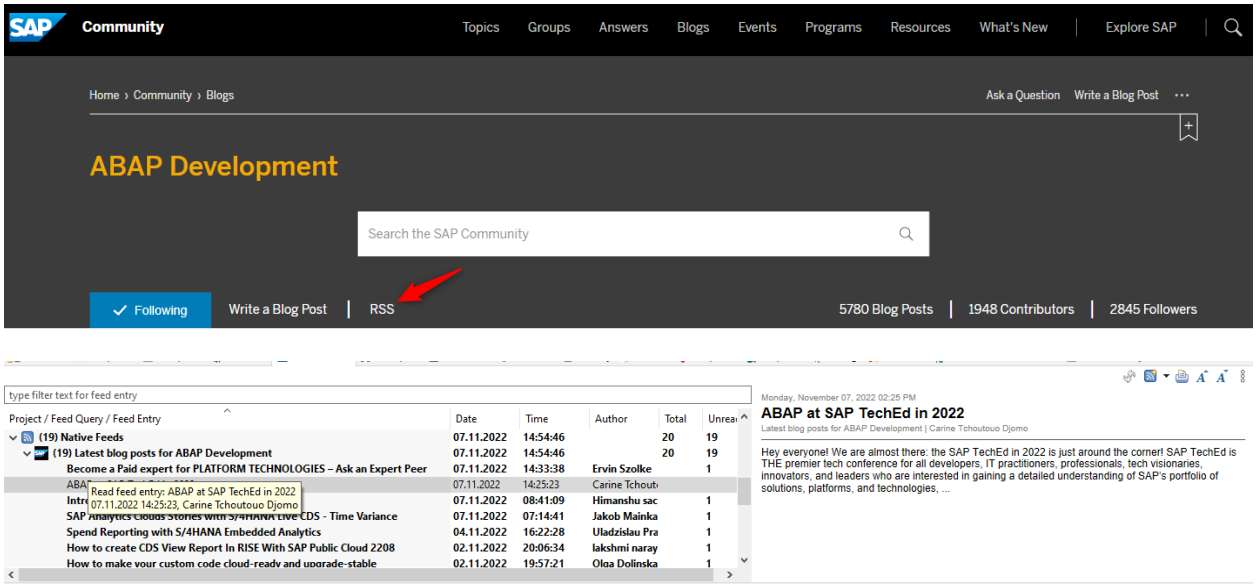


Figure 118 Subscribing to Popular RSS Feeds

With a left-click on the title in the detail display or right-click in the list and the context menu item Open you can open the blog article (for this it makes sense to set an external browser in Eclipse, because the Eclipse internal browser pretends to be IE11 for the website).

The system news feed displays the current messages from the system administrators.

You can add various filter options to feeds on runtime errors (ST22 dumps), including filtering by triggering user, responsible user, object and package user, or package. The filters can also be used with and/or

(all/any-) Links to a hierarchical filter tree can be arranged. You can open a runtime error in a separate editor view (context menu in the list, link in the details; the well-known long text format or an unformatted display is also available there). You can navigate directly to the source code lines of the triggering location and the call hierarchy directly in the ADT source code editor.

An example with multiple runtime errors in one feed:

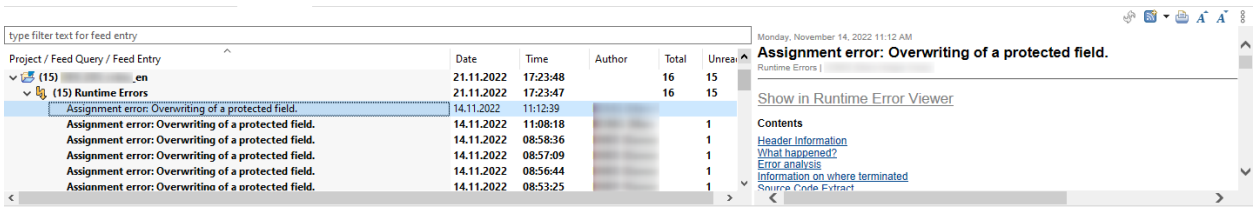


Figure 119 Multiple Runtime Errors within a Feed

Documentation on the SAP Gateway Error Log Feeds can be found in the PDF document from Note 1797736 - SAP Gateway Troubleshooting Guide and in the blog How to use the SAP Gateway Error Log in ADT.

You can add various filter options to feeds, including user, service, namespace or package. In order to cope with many entries, paging can be activated. In the detail display, you can jump directly to transaction /IWFND/GW_CLIENT for replay. In addition, you can navigate directly into the ADT source code editor analogous to the runtime errors, see the following example:

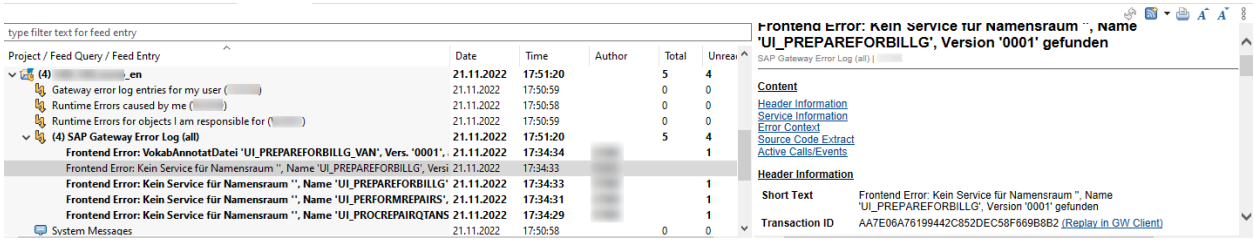


Figure 120 View of an SAP Gateway Error from the Error Log

In the context of enterprise events, you can add feeds to errors from event processing, for example, to see errors when passing the events to the event mesh. To do this, you can filter channel and user.

You can create feeds for the ATC results from tests in a central ATC check system, see Documentation (On-Premise/Cloud). Here, too, various filter options are available. In the feed list, you can navigate to the details and source code line of a result.

If you have BW/S4HANA systems, you can use the BW Job Repository Feed to check the status of various job types (e.g. DATAFLOWCOPY or DTP_LOAD) have it informed, cf. Documentary. You can then branch from the feed entries to view the job details.

Since the Feed Reader is the ideal tool for proactive monitoring of applications (e.g. during and after a go-live of a new application), it makes sense to consider the authorization for ADT not only for the development systems, but also to enable the developers to collect the feeds from test and production systems by means of authorizations.

4.6 Documentation Links

ABAP Debugger Concept:

https://help.sap.com/docs/BTP/5371047f1273405bb46725a417f95433/4ec365a66e391014adc9fffe4e204223.html

Troubleshooting Tools:

https://help.sap.com/docs/BTP/5371047f1273405bb46725a417f95433/4ecc7d3a6e391014adc9fffe4e204223.html

Syntax for Breakpoint Conditions:

https://help.sap.com/docs/BTP/5371047f1273405bb46725a417f95433/d878e676fe904eba9f4bb79193154092.html

5 Installation, Distribution und Update Strategies

ADT are based on the open source Eclipse framework, which is very well known in other programming languages. For installation and distribution, this has the advantage that the wheel does not have to be reinvented. Depending on the size of the company and the heterogeneity of a company's IT landscape, it is even possible that Eclipse-based development environments are already used and distributed in the company. In this case, it is advisable to continue using the existing infrastructure. This can also reduce any effort required for one or the other solution.

5.1 Delimitations

5.1.1 Installation Guide of SAP

SAP publishes its own installation guide (link). In addition to the manual installation method also listed here, it also lists other variants for locked-down environments with very limited Internet access. Since these are solutions for specific situations, they will not be discussed in the following sections.

5.1.1.1 Other tools with identical installation path

SAP also publishes other tools based on Eclipse, such as BW Tools (mandatory from BW/4HANA 1.0) and HANA Development Tools. However, since this guide focuses on ABAP development with ADT, we will not go into further detail about the other tools mentioned. However, most of the findings are transferable.

5.2 Preparations

For every ADT installation (sometimes also Eclipse without ADT) there are certain requirements. These are independent of the installation/distribution strategy and are described in detail in the above-mentioned ADT Installation Guide.

5.2.1 Java Development Kit and Java Runtime Environment

If only ABAP is developed with the installation (and in particular no Java, i.e. no plug-in development to complement ADT, see Chapter 7 Plug-ins), no Java Development Kit (JDK) is necessary. A Java Runtime Environment (JRE) is sufficient. If the official

installer of Eclipse is used as a basis (see Chapter 6 Installation, Distribution and Update Strategies), this comes automatically.

It should be noted that Eclipse is now installed as a 64-bit application by default (often referred to as *x64* or *x86_64*). Because the JRE must use the same architecture as the Eclipse installation, a 64-bit JRE is also installed.

5.2.2 Backend

In addition to the local ADT installation, in the case of development on an on-premise system, it must also be prepared for connection to ADT. These steps are also described in the official SAP ADT Configuration Guide . However, two essential steps are often overlooked, which is why it should be explicitly pointed out again here.

5.2.2.1 Web Services

For communication, ADT use special web services on the backend, which are delivered with the supported basic releases. The definition of the web services and the handler classes entered in them are one reason why the functional scope of ADT differs between the basic releases. These web services are inactive by default and must first be enabled. The current list can be found in the SAP Configuration Guide linked above.

At the time of this document, these are the following services:

- ABAP Docu (required)
  - *default_host → sap → public → bc → abap → docu*
  - *default_host → sap → bc → abap → docu*
- Error texts and element info (required)
  - *default_host → sap → public → bc → abap → toolsdocu*
  - *default_host → sap → bc → abap → toolsdocu*
- Sharing HTTP links (optional)
  - *default_host → sap → bc → adt*
- Web Dynpro (only necessary for WD developers)
  - *default_host → sap → bc → webdynpro → sap → wdy_aie_vd_preview*

5.2.2.2 Permissions

In order to be allowed to use the aforementioned web services, the users must be authorized. In addition, RFC blocks and transaction codes are required.

SAP provides two roles as templates for this purpose:

1. → *developer role with all features* → Entwicklerrolle mit allen Features
2. → *View permissions for all features* → Anzeigeberechtigungen für alle Features

Details regarding the built-in authorizations and their purpose can be found in the SAP ADT Configuration Guide .

5.2.3 SAP GUI Installation

If an SAP GUI transaction start is to be possible on a back-end system (not available in SAP Public Cloud and SAP BTP ABAP Environment/Steampunk), a local SAP GUI installation is required. ADT do not supply them.

5.2.4 Visual Studio Redistributable

On Windows, the Visual Studio 2013 (VC++ 12.0) redistributable package is also required in exactly this version. Often, this dependency is already present due to other software already installed on the target device.

5.3 Technical Structure of an Eclipse Installation

As mentioned in previous chapters, an Eclipse installation consists of the following components:

- Installation folders (software-only packages and eclipse.exe)
- User-Settings or Configuration-Area (plug-ins, parts of the configuration)
- Workspace (user-specific part: settings, views, system connections, etc.)

The installation of Eclipse creates the installation folder and user settings. Using advanced techniques, parts of the workspace can already be pre-assigned.

Due to various problems, it is recommended to place all components in a folder that can be written to with normal user rights (i.e. **not** C:\Program Files\).

For example, a directory structure for the Eclipse installation might look like this:

`C:\ADT\`	Complete directory for ADT
`C:\ADT\IDE`	Directory for storage of the different Eclipse versions

`C:\ADT\IDE\2022-12`	Unzipped files of Eclipse version 2022-12
`C:\ADT\IDE\2023-03`	Unzipped files of Eclipse version 2023-03
`C:\ADT\WS`	Directory for storing the various workspaces
`C:\ADT\WS\2022-12`	Directory for version 2022-12 workspaces

…

This example is based on the assumption that the Eclipse installation is done by unpacking the zip files and that the versions should be used in parallel.

Since the workspaces with ascending versions are converted to the new version, the workspaces should be copied per version so that you can continue to use the older versions if necessary.

The extent to which workspaces are used sensibly is explained in more detail in chapter 3 "Working with Eclipse".

5.4 Plug-ins

Plug-ins such as ADT can be added to an existing Eclipse platform by specifying the update site in the Help → Install New Software dialog. However, it is easier to use the Eclipse Marketplace if the plug-in

5.4.1 Eclipse Marketplace

The Eclipse Marketplace is hidden in the Help menu.

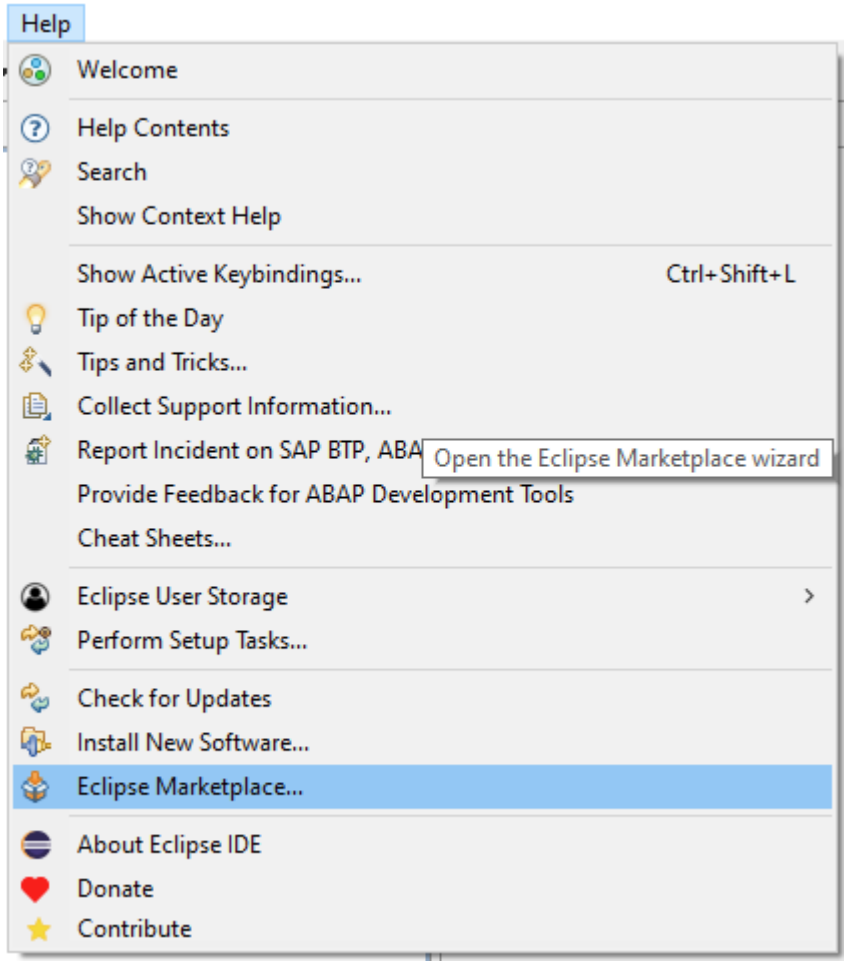


Figure 121 Getting Started with the Eclipse Marketplace

Here you can search for plug-ins. As of 2022, the search term *ABAP* returned 11 hits in this example.

Figure 122 Sample Search for Plug-ins in the Eclipse Marketplace

Each plug-in has its own install button on the right-hand side. If necessary, a license must be confirmed and certificates trusted. Finally, a restart of Eclipse is necessary.

5.4.2 Update Site

If you know the update site of a plug-in or if it is not listed on the Eclipse Marketplace, the classic installation method can also be used.

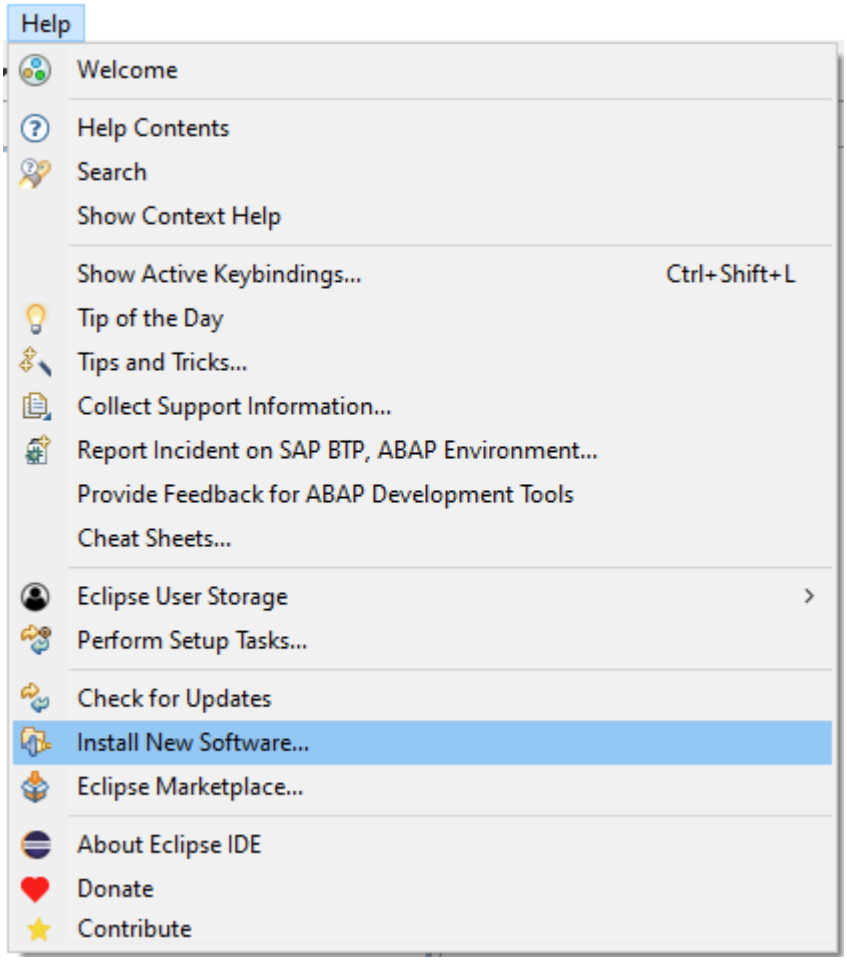


Figure 123 Installation of New Software via the Context Menu

The update site is entered in the *Work with* field. If this is valid, the plug-ins available there are displayed below. In addition to web pages, a downloaded version of the plug-in in a zip file can also be an update site. In the latter case, however, updates must be carried out manually with another download.

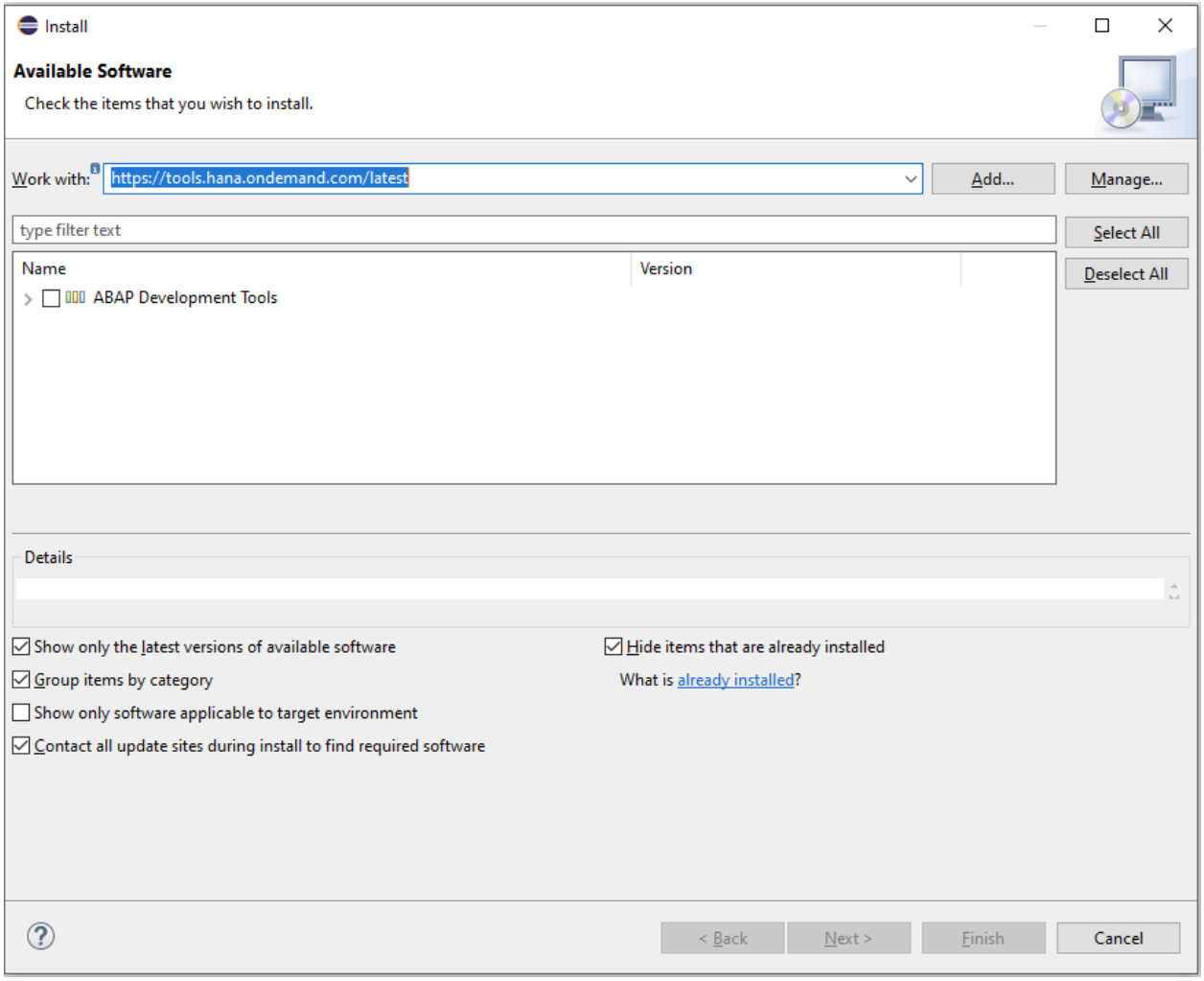


Figure 124 Entering the Update Site

Here, too, licenses may have to be accepted and certificates trusted. After a restart, the plug-in is available.

5.5 Installation and Distribution Strategies

5.5.1 Overview and Comparison

Since ADT were added to the Eclipse ecosystem comparatively late, various installation strategies for Eclipse-based development environments have already been established outside the SAP environment. The most common are:

1. Completely manual setup
2. Pre-configured initial installation
3. Eclipse Installer (Oomph)

In the following chapters, these mechanisms are discussed in detail. In addition to these, there are many other options that were either designed for rather unusual situations (e.g. lack of Internet access) or have now been replaced by convenient variants.

∘ + (good), ∘ (medium) and - (bad) are to be understood as a ranking and not as absolute values at the beginning, middle and end of the respective spectrum. The reason for this is that, for example, an effort is perceived very individually. Thus, a high effort marked with - (badly) can still be acceptable for a person. However, these symbols represent a sequence since measurable differences exist.

Criterion	Manual Installation	Pre-configured Initial Installation	Eclipse Installer
User Effort	-	∘	+
Necessary Knowledge User	-	∘	+
Effort Central Administration	+ *(none)*	∘	-
Required Central Infrastructure	+ *(none)*	∘	-
User Effort for Update/Upgrade	∘	∘	∘
Effort Central Administration for Update/Upgrade	+ *(none)*	-	∘
Automatic distribution Add-ons, Settings	- *(impossible)*	∘ *(once)*	+ *(continuously)*

Criterion	Manual Installation	Pre-configured Initial Installation	Eclipse Installer
Recommended for Company Size	*Individuals/small businesses*	*Medium-sized and large companies*	*Large companies*

Table 1 Comparison of Different Installation Options

5.5.2 Manual Installation

In this variant, a standard Eclipse installation is re-downloaded and installed with just a few clicks. Since there is no ready-made installer for ABAP, ADT are installed individually. Without plug-ins, settings and perspective adjustments, a usable installation is available after about 15 minutes.

SAP has provided illustrated instructions on its learning platform: https://developers.sap.com/tutorials/abap-install-adt.html.

Once this installation is complete, the desired settings must be made and the system connections must be added. If further plug-ins are used (see chapter 7 "Plug-ins"), they must also be installed.

5.5.3 Pre-configured Initial Installation

A preconfigured initial installation is basically a manual installation, the state of which is backed up immediately after setup (usually packaged as a zip file). This condition can then be distributed to other people in various ways.

With macOS as the target system, it is also important to note that a so-called app-zip translocation could take effect here. This creates a kind of "shadow copy" when executed by software such as Eclipse, which has just been unpacked *in the same directory*. The result behaves like missing write permissions in the installation folder. If updates are installed, errors occur.

Depending on the intended use, more or less of the three components of the installation (installation folder, user settings, workspace) can be packed in.

If you take the user settings with you, ADT and its update site, for example, can be distributed at the same time for later updates. In Citrix environments, on the other hand, this would be a hindrance, as the user settings require write permissions. If you take two separate packages for this, the installation folder could also be placed in a read-only, centrally provisioned part.

The workspace could also be packaged and distributed as a separate template. This means that initial settings can be distributed once. However, since this area changes very frequently and cannot be used without losing the current settings, layouts and open objects during an update anyway, this should only be done with caution.

5.5.4 Eclipse Installer

On the Eclipse website, a convenient installer with a very small download size can be downloaded (also called *Oomph Installer*). When you start it, you can then select a basic configuration and a few detailed settings and then have exactly this installation created.

However, there are no configurations including ADT here. However, this is not magic, but the available settings are only stored in a certain format on a server of the Eclipse Foundation. The path where the installer should search for configurations is customizable. Thus, it is possible to create company-specific configurations that anyone can install at will. This method could already be used in some companies for other development languages, which significantly reduces the maintenance effort.

Only the Eclipse installer has to be distributed, for example, via a software distribution mechanism together with the setting where the configurations can be found. In addition, this variant offers the option of presetting individual settings in the workspace and keeping them up-to-date.

The disadvantage of this variant is the comparatively high central effort. It is therefore not suitable for single/few standard installations.

The Oomph project has provided an extensive documentation in English, which is well suited as a reference work: Link.

Administrator Information

The following sections explain how to create and adjust the Oomph configurations. This is usually done by a few administrators.

Definitions

If you read through Oomph's documentary, you will first be overwhelmed with many new terms. Therefore, here are the most important terms used in the following sections.

Term	Description
Setup Model	The Oomph configurations, as in most programming languages, are file-based. A set of these files with a specific format is called a setup model.
Product	An installation-level configuration (platform with specific version + plug-ins)
Project	Project-specific settings. In the git-based world, for example, this can be the default of standard repositories. Workspace-level configuration.
Index	The library of available configurations that can be selected in the Eclipse Installer.

Table 2 Terms in Oomph

An installation using Eclipse/Oomph Installer always installs a platform, the plug-ins, Oomph Updater, Oomph Recorder and finally the project settings in the workspace.

Required software for administrators

If you want to design and manage your own configurations, you need the Oomph SDK. This is nothing more than a set of plug-ins on a standard Eclipse platform. This delivers the required views and ready-made perspectives in order to be able to edit the configuration files graphically (tree views, forms, etc.).

Step by step to a basic configuration

When creating configurations, one usually proceeds from general to specific. For all the required files mentioned below, the Oomph SDK offers Wizards (File → New → Other → Oomph → ...).

Index

This means that you first create an *index*. An index refers to available *product catalogs* and *project catalogs*. By default, the index is named *org.eclipse.setup*, but it can also *be called* myFirst.setup, for example.

The following is the structure of an index:

```
<?xml version="1.0" encoding="UTF-8"?>
<setup:Index
    xmi:version="2.0"
    xmlns:xmi="http://www.omg.org/XMI"
    xmlns:setup="http://www.eclipse.org/oomph/setup/1.0"
    name="myCompany Eclipse Setups"
    label="index">
  <productCatalog
      href="myCompany.products.setup#/"/>
  <projectCatalog
      href="myCompany.projects.setup#/"/>
</setup:Index>
```

In the extended view of the Eclipse Installer you also have the possibility to switch between several available indexes.

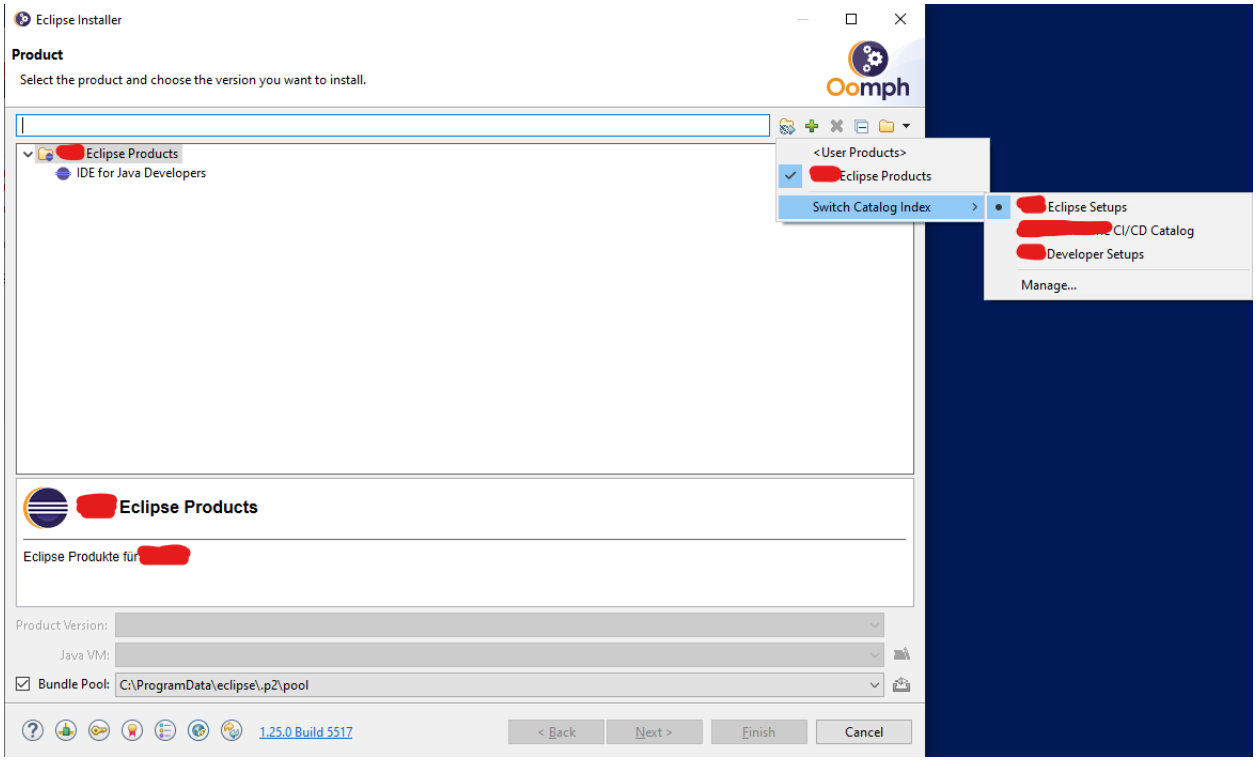


Figure 125 Switching between Indices

Product Catalog and Product

A product catalog lists various products. It also contains overarching settings, such as:

- internal redirects of update sites to local caches
- Definition of variables that can later be filled differently for each product (version) or project (for example, for version-specific update sites)
- the installation of the Oomph Client for later distribution of updates to the settings specified in the Products and Projects, Oomph Recorder (see section User Information).

The following properties can be stored at all levels (Product Catalog, Product and Product Version):

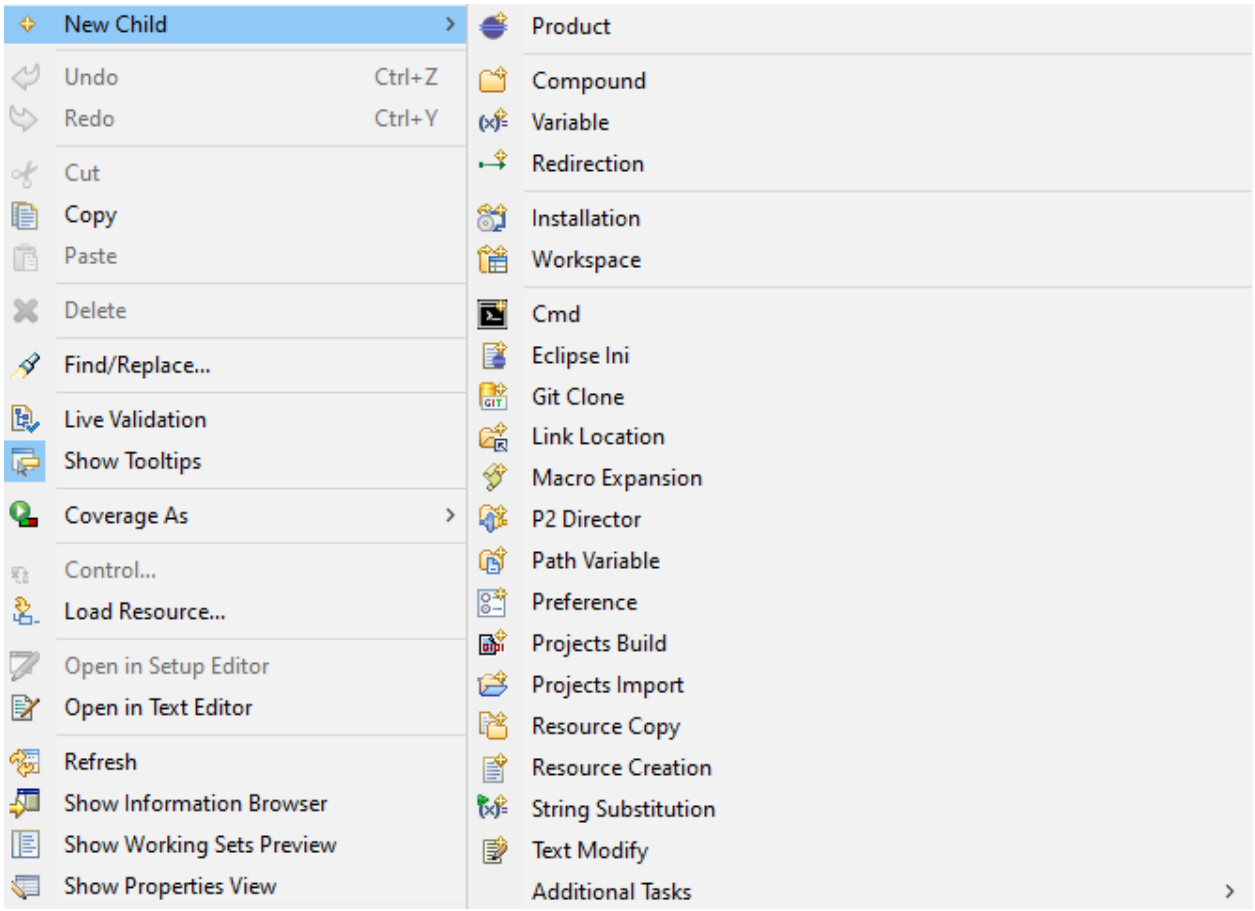


Figure 126 Adding and Setting Properties

The most needed elements are:

Product	Adds a product to the product catalog (for example, IDE for JAVA Developers). Can only be added at the Product Catalog level.
Product Version	Adds a version to a product in which version-specific settings can then be made. Can only be added at the Product level.
Compound	This is a kind of folder in which settings can be grouped later.
Eclipse ini	Adds an option to the Eclipse.ini.
Variable	Variable with value assignment. It can be referenced later.

Redirection	URL redirect. Thus, the installations can integrate the original sources, but in fact, for example, an Artifactory is accessed.
P2 Director	List of Feature Groups to install.
Repository	Provides an installation source for Feature Groups

Table 3 Key Elements

For a minimal ADT installation, the following is required::

- One Product, z. B. “SAP”
- One Product Version, z. B. “2022-03 (4.23)”
- One P2 Director Task with:
  - Eclipse Platform Packages
    - epp.package.java (Value-Range starts at desired release → 4.23)
    - org.eclipse.platform (Value-Range starts at desired release → 4.23)
    - org.eclipse.rcp (Value-Range starts at desired release → 4.23)
    - org.eclipse.buildship
    - org.eclipse.tips.feature
    - org.eclipse.epp.mpc
- Repository-URLs for Eclipse Platform Packages
  - https://download.eclipse.org/releases/2202-03/202203161000 (Link release-dependent!)
  - https://download.eclipse.org/technology/epp/packages/2022-03/202203101200 (Link release-dependent!)
  - The most recent version is available via https://download.eclipse.org/technology/epp/packages/latest/

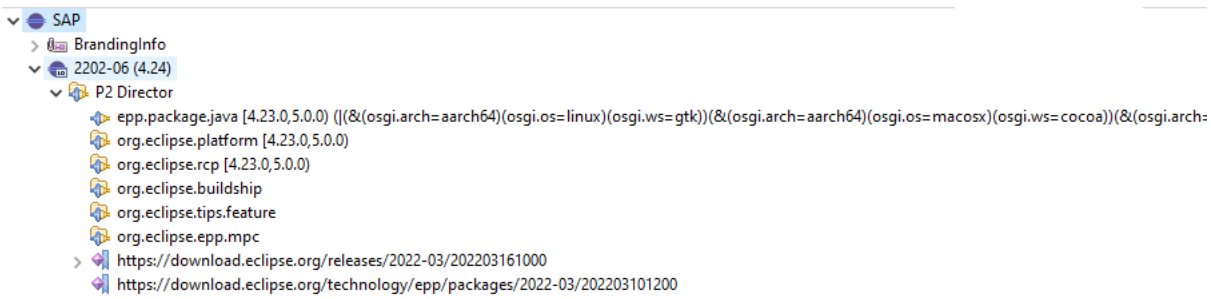


Figure 127 Components of a "minimal" ADT Installation

Project Catalog and Project

The Project Catalog lists available projects. The latter contain instructions for changing the workspace. For example, the plug-ins to be installed are also stored here.

The most important elements in a project are:

P2 Director	Grouped requirements and repositories
Requirement	Feature Group to install
Repository	Update Site URL
Stream	Obligatory object. Different configurations per stream possible. Can be present with a name, but without content.
Variable	Variable string with referencing, e.g. in repositories
Eclipse ini	Changes to the Eclipse.ini
Preference	Preset change of settings (Window → Preferences).

Table 4 Most Important Elements of a Project

For a minimal ADT installation, the following is required:

- P2 Director Nodes
  - ADT Feature Groups
    - com.sap.adt.tools.hana.devedition
    - com.sap.core.devedition
  - Repository for ADT
    - https://tools.hana .ondemand.com/latest
    - Alternativ: ${Variable} → z. B. ${sap.repository.url}
- An empty stream, "Master" by default

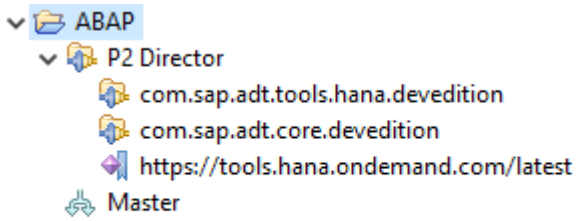


Figure 128 Components of a Minimal ADT Installation

It should be noted that a plug-in for Eclipse can consist of several feature groups. How to find out is explained in the section Installing an additional plug-in .

Distribution of the installer and configuration

The Eclipse installer is the same exe file that can be downloaded from the Eclipse home page. This must be distributed to the end devices.

The configuration is distributed by creating/customizing a file with a specified name and path:

C:\Users\<currentUser>\.eclipse\org.eclipse.oomph.setup\setups\indices.xmi

Example of an index list:

```
<?xml version="1.0" encoding="UTF-8" ?>
<base:Annotation
    xmi:version="2.0"
    xmlns:xmi="http://www.omg.org/XMI"
    xmlns:base="http://www.eclipse.org/oomph/base/1.0"
    source="IndexLocations">
  <detail key="https://pages.github.com/path/subpath/myFirst.setup">
    <value>Description of my first Oomph Catalog</value>
  </detail>
</base:Annotation>
```

The path to one or more indexes must be specified. The path can also be a web link or git repository path. The linked files will then be downloaded/updated each time the Eclipse installer is started.

Now any user in possession of the installer can select and install these configurations. Alternatively, the installer can also be distributed within the company via software distribution. Thus, a synchronization of the versions between Oomph components in the installations and the Eclipse installer used could be achieved.

Adaption Examples

So far, minimal configurations for installing Eclipse and SAP ADT have been described. However, the strengths of using the Oomph Installer only become apparent in the further pre-configuration during installation. The most common expansion requests will now be addressed.

Installing an additional plug-in

One of the advantages of Eclipse as a development platform is its openness to extensions. Thus, plug-ins can be written by third-party manufacturers or dedicated community members that extend the functionality of ADT even further.

These can be automatically pre-installed in all installations in an Oomph Project. Unfortunately, however, it is not possible to add a repository as an update site, but then not install a plug-in on it. These update sites will be discarded when Oomph is installed (as of 2022).

Now, a plug-in consists of one or more feature groups. The latter must be deposited in the Oomph Project. However, the name is usually not known. For this purpose, there is a view "Repository Explorer" in the Oomph SDK (Window → Show View → Other → Oomph → Repository Explorer). By (manually) specifying an update site, a query of the provided feature groups is performed. These can then be done by drag-and-drop or **`CTRL+C`** and **`CTRL+V`** in the P2 Director node of the Oomph Project. In addition to the feature groups, there must also be a repository node that makes this update site available (can also be done more generally, i.e. in the Project Catalog or Product (-Catalog)).

As of October 2022, the query of the SAP update site looks like this:

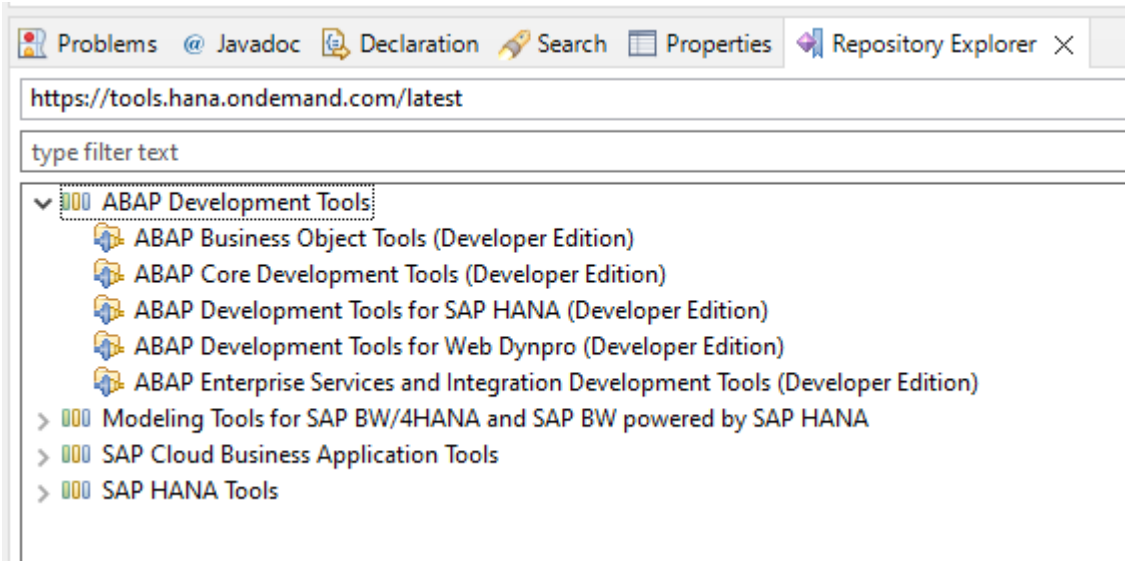


Figure 129 Components of the SAP Update Site

Specifying settings

In addition to the installation of ADT, one of the biggest advantages is the presetting of various settings for all installations.

All settings to be assigned are placed in the Compound/*User-Preferences* folder of the Oomph Project. Below that, there are individual subfolders per feature (as part of a feature group) and then the settings. However, it is up to each feature to decide in which representation it stores its settings. Some settings are stored as direct values (classic checkboxes), others, for example, as one large XML per settings page. Especially with the latter, only all settings of the page or none can be specified. An example of XML representation is the code templates of SAP ADT.

Since this non-uniform display is impractical to administer and the names of the features are usually not known, there are also auxiliary tools here: An oomph recorder is installed in the settings. For more information on how to use it as a user, see the Oomph Recorder section. The Oomph Recorder records the latest status of all changed settings at the user level (i.e. across Eclipse installations) and asks if they should be saved always/once/never after closing the settings. It also makes a button available in the button bar of Eclipse to view the settings that have already been saved.

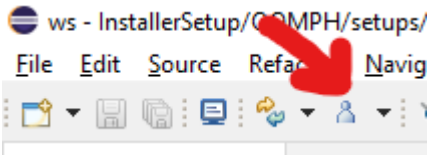


Figure 130 Ability to View the Settings that have Already Been Saved

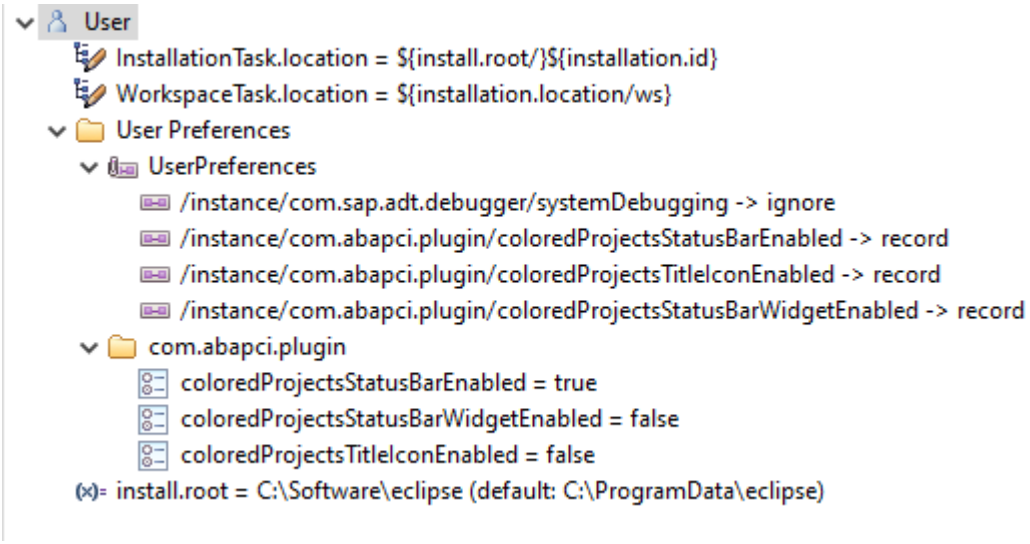


Figure 131 Settings that Have Already Been Saved

Again, these are setup files. This means that the recorded settings can be **`transferred to the Oomph Project`** with drag-and-drop or CTRL+C and **`CTRL+V`**.

The following are a few examples of generally distributed settings. Here are some of ADT-exclusive features, all of which SAP unfortunately does not enable by default.

- Code Templates
- Activation of additional code highlights in ABAP (different colors)
- Enable continuous syntax checking
- Activating the Occurrence Marker
- Uniform display of indents (tabs or spaces, width of indentation)

Increasing memory limits

With SAP ADT, there are no longer any limits to the number of open windows/objects. However, this also means that with many simultaneous tabs, the required RAM

skyrockets. Since Eclipse itself is also a Java application, VM parameters must be set. If the available memory of a Java VM is exceeded, a swapping in hard disk files takes place analogous to the RAM management of the operating system. This makes Eclipse slow and the fun of using it suffers. Minimum and maximum limits, initial reserved memory and much more can be set for VM memory. However, since this can also break a lot and the required limits are very individual, concrete recommendations are deliberately omitted.

If such parameters are specified for all users, they must be specified in the Eclipse .ini. Thus, corresponding Eclipse Ini tasks must be created in the Oomph Project.

First start with ABAP Perspective

If the SAP ADT is installed, the standard perspectives are initially only available, but not opened. Eclipse, for example, starts with the Java Perspective, which most ABAP developers will rarely need. Therefore, it is desirable to start the installation directly with the delivered ABAP Perspective. Unfortunately, it is not possible to store additional perspectives (e.g. debug, ABAP profiling, etc.) in the most recently used perspectives in the upper right corner.

Forcing a start perspective is a start option in the Eclipse .ini. Therefore, an Eclipse Ini Task with the following properties must be created:

Option	-perspective
Value	com.sap.adt.ui.AbapPerspective
VM	false

Table 5 Features of Eclipse Ini Task

5.5.4.1 User Informationen

Installation via Eclipse Installer

When the Eclipse installer is started for the first time, it may start in simple mode. However, the extended mode is required to use your own configurations. You can switch to it via the menu at the top right ("Advanced Mode").

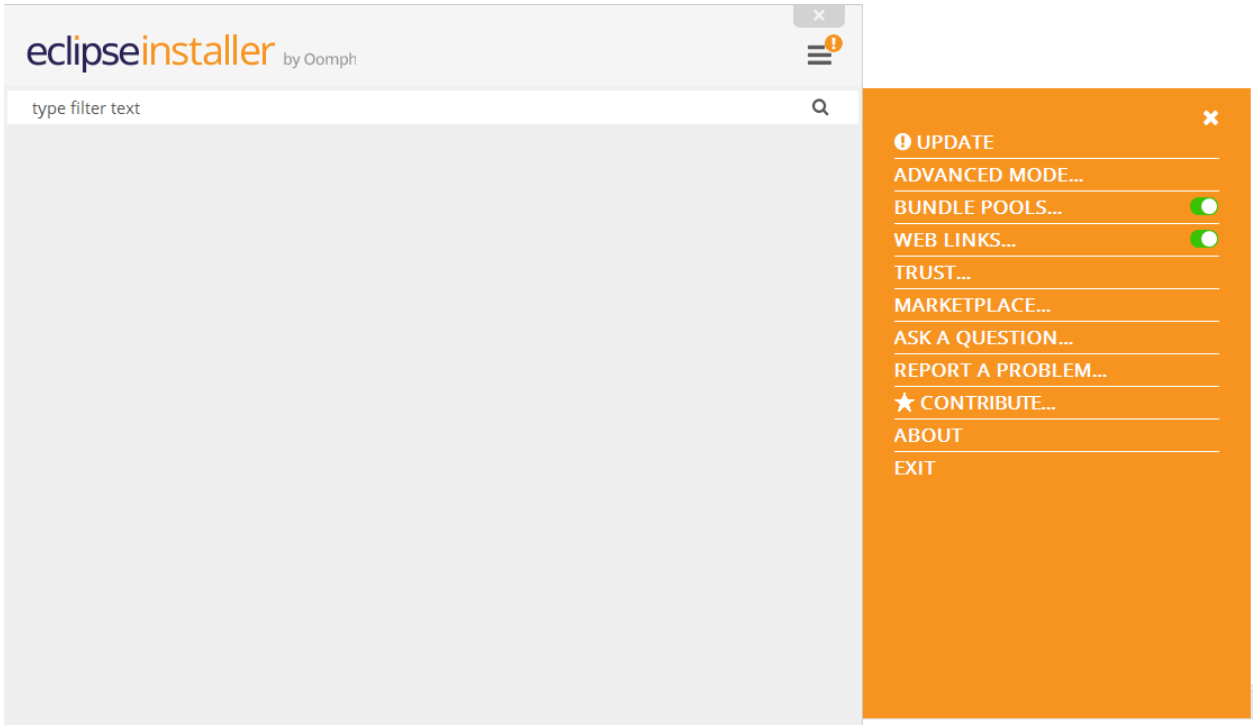


Figure 132 Switch to "Advanced Mode"

Now you can see a list of available products. This is the content of the first referenced index in the Indices.xmi. In the upper right corner you can switch between all listed indices.

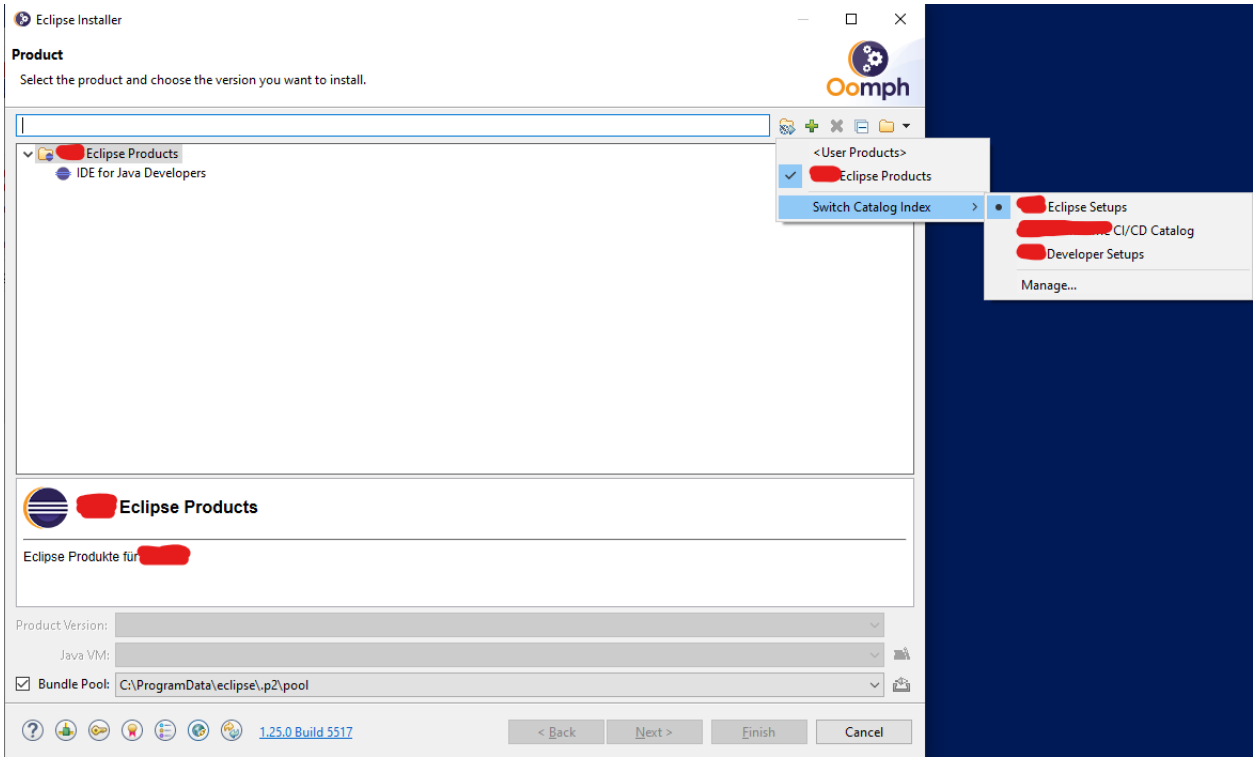


Figure 133 Switching between Indices

Here you select a suitable entry. Once this is done, the available product version will be selectable at the bottom.

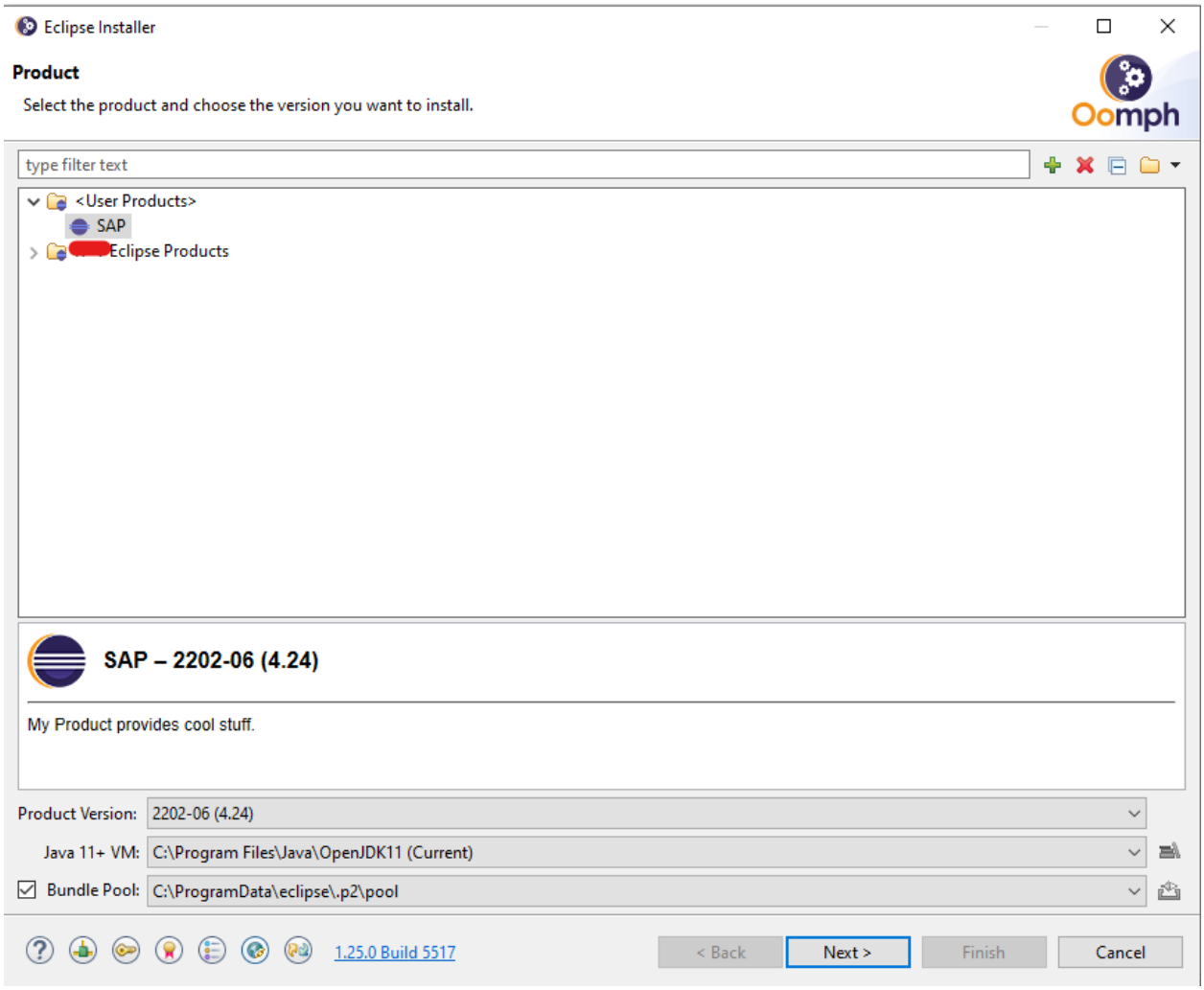


Figure 134 Selection of the Product Version

In the next picture, the projects can be selected. Theoretically, several projects can be selected for one installation. However, it can happen that these projects then make competing settings, and problems arise.

A stream must then also be selected for a project. If only one has been defined, it is already preselected.

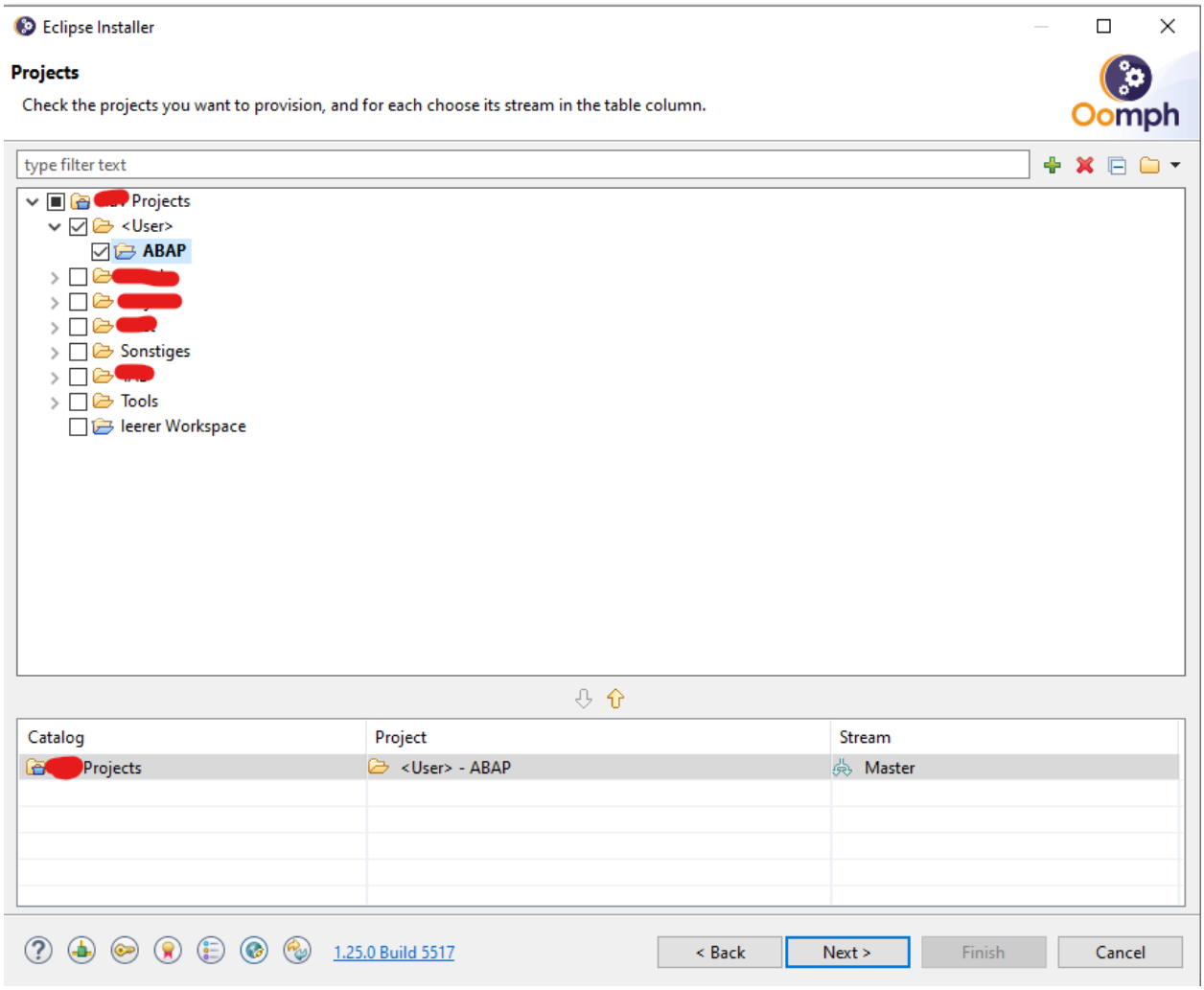


Figure 135 Selection of the Stream

Now defined and used, but not filled variables are queried. These can be, for example, the paths to the installation and workspace.

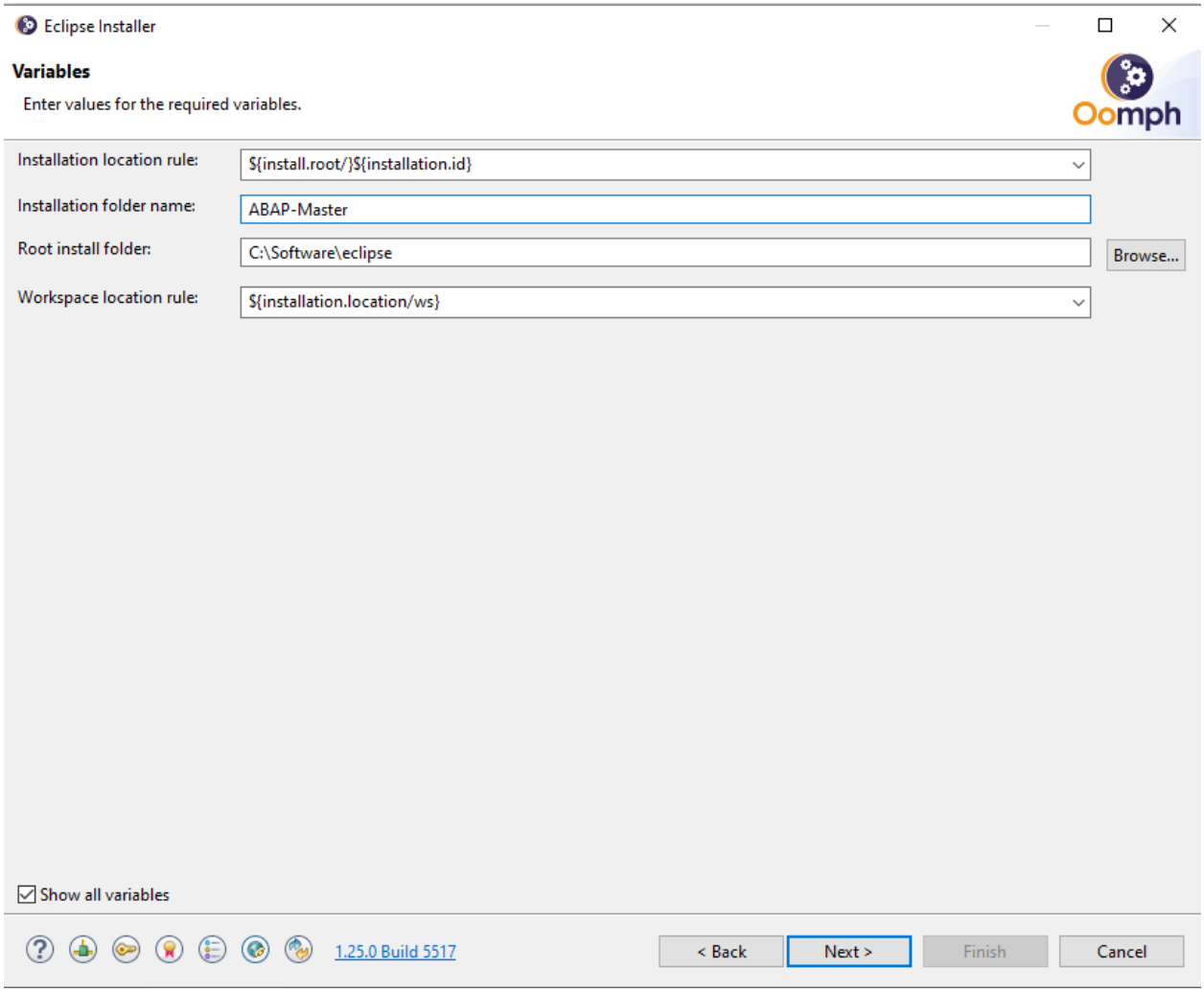


Figure 136 Querying other Variables

Finally, a confirmation page is displayed, but it usually does not contain any new information.

During installation, pop-ups may appear asking which license terms to accept and/or trust certificates.

If the check mark is set in the installer, the instance that has just been installed will be started after the installation. The settings of the workspace are then preconfigured again, since the workspace only exists now. Now the installation is complete.

Oomph Recorder

The Oomph Recorder is a tool to unify and save settings across Eclipse installations. Optionally, these settings can also be loaded to the Eclipse User Account, so that cross-device synchronization is also possible here. In most companies, however, the use of an Eclipse account is probably not possible or not welcome.

The Oomph Recorder is activated under Window → Preferences with a new recording icon. From then on, every time you close the settings, you will be asked whether the settings that have just been changed should always/once/never be saved in the last state.

These user-level settings are applied to a new Eclipse installation only after the first start and after the settings of the Oomph Project used. This means that standard settings can also be overridden on a user-specific basis or your own default settings can be defined.

5.6 Error Situations

During installation or an update, various errors may occur. The most common mistakes will be discussed here.

5.6.1 Errors during update or upgrade

When performing an update or upgrade, the installation may fail.

An installation usually takes place in the following phases:

1. Calculate target versions of all components and their packages, including consideration of dependencies
2. Download Packages
3. Installing Packages

Experience has shown that the most common cause of error in the calculation of target versions is dependencies that cannot be fulfilled.

For example, in a made-up example, an ADT upgrade also requires a newer version of Eclipse itself. However, only the current version can be found in the available update sites. Thus, the cause of the error here is an incorrect or outdated update site.

In more difficult cases, for example, in newer releases of Eclipse, a component may be removed or replaced. However, a plug-in like ADT still has a dependency on it. Only the manufacturer of the plug-in can remedy this.

Such a situation can arise when using HANA Studio as the Eclipse platform. Here, no public update sites for the Eclipse platform are set up during installation, which is why it no longer fits with current ADT versions after six months (= two quarterly releases) at the latest.

Errors occur much less frequently during the download. In the first step, the inventory lists of the update sites (artifacts.xml and contents.xml) were examined and a suitable package version was found. However, if this is not in the corresponding subfolder, a download error occurs. In this case, the update site is inconsistent. This can also only be fixed by the provider of the package.

5.6.2 Single-sign-on Libraries

Depending on the single sign-on strategy, environment variables on libraries may be necessary. If SAP GUI is only installed as a 32-bit application (up to SAP GUI 7.70 the only option), the libraries referenced in the environment variables SNC_LIB and SNC_LIB_2 are always used.

However, if Eclipse is installed as a 64-bit application (the default case), the SSO library referenced in the SNC_LIB_64 environment variable is used.

In older blog entries, the Kerberos libraries of Windows are still recommended (32 bit: gsskrb5.dll, 64 bit: gx64krb5.dll). However, these are no longer present in newer Windows installations and should no longer be used. SAP delivers an SAP CryptoLib with the SAP GUI. This is available in 32 bits and 64 bits.

5.6.3 “No repository found containing”

https://launchpad.support.sap.com/#/notes/2186770

Every now and then there seem to be problems updating ADT. Several errors appear in the log: "No repository found containing: ...". The note recommends removing the update site, restarting Eclipse, and then re-adding the update site.

5.6.4 PKIX - Certificate Error

https://launchpad.support.sap.com/#/notes/3131747

This is a certificate error. An encrypted connection (HTTPS) is established to the update site. If SSL connections are broken in the corporate network or there is no common keystore for internal and external update sites, then this error can occur. The note gives a possible solution to this. Another option is to distribute your own custom JDK.

5.6.5 macOS aarch64 support & SAP GUI for Java

https://launchpad.support.sap.com/#/notes/3251738

The architecture of the SAP GUI and Eclipse installation should always be installed with the same processor architecture. Especially with Apple M1 and subsequently, there could be deviations here.

5.6.6 Offline-Installation – Download ADT-Dependencies

https://launchpad.support.sap.com/#/notes/2369308

If ADT is to be installed offline, various dependencies must be taken into account. This note gives some remedies in this case.

6 Best Practices Eclipse Configuration

6.1 Settings in Eclipse

In Eclipse, there are numerous settings that can make life as a developer easier, but sometimes harder. In this section, you will learn more about the different options and the most important settings. Important to know: There are two levels at which you can make settings. The *global level for* Eclipse and the *project-specific level* for an SAP system.

6.1.1 Globale Settings

After opening the settings via the menu (Window → Preferences) you will see all the settings for Eclipse. In the window on the left is the tree with sub-nodes for navigation, above it a search box to search for nodes or settings. On the right side are the settings for the selected point.

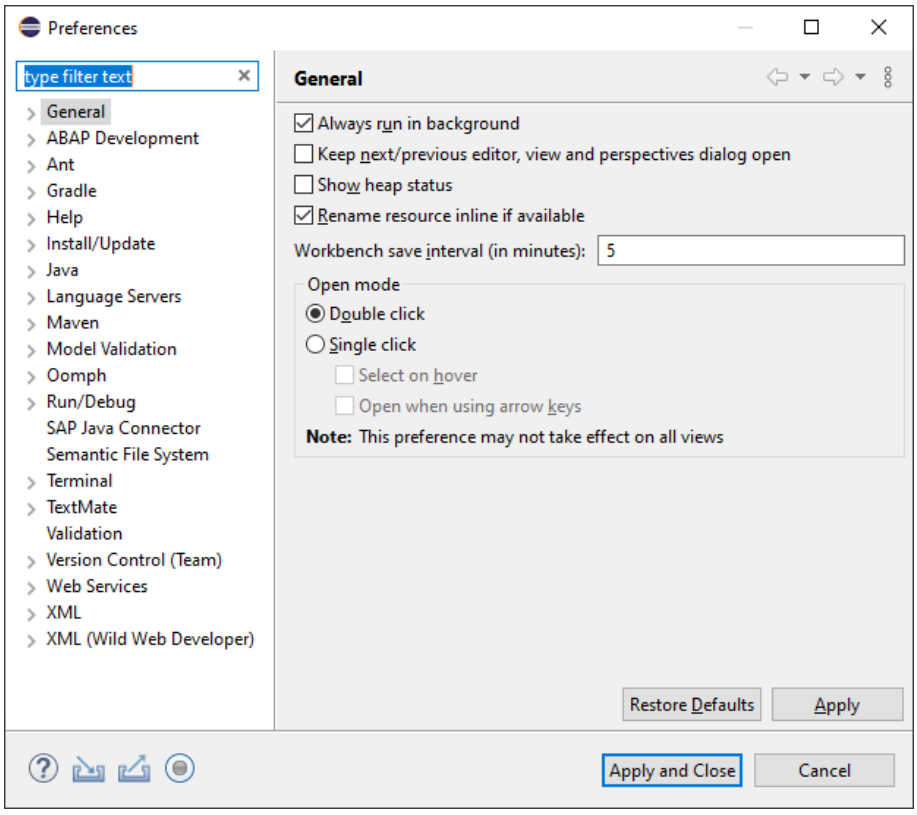


Figure 137 Getting Started with Global Settings

In the following sections, we'll show our suggestion for these settings, the path within the settings to find them, and a brief explanation of the effects.

6.1.1.1 Dark Theme

(General → Appearance)

Many development environments now offer the possibility to work with a light or dark theme to protect the eyes or simply to suit personal taste.

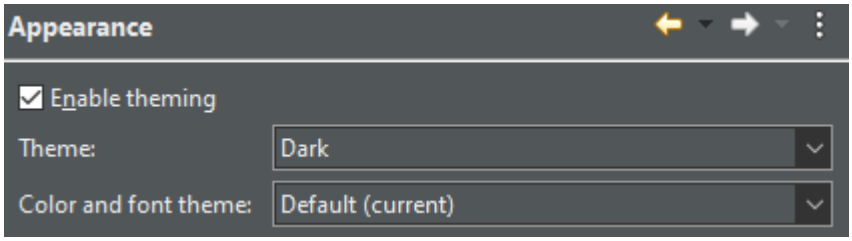


Figure 138 Setting for the Dark Theme

6.1.1.2 Indentation

(General → Editors → Text Editors)

When writing ABAP source code, a tab width of 2 spaces is often used, but by default it is set to 4 ("Displayed tab width") in Eclipse. You can also set whether spaces are inserted instead of a tab ("Insert spaces for tabs") and whether an entire tab should be removed when deleting ("Remove multiple spaces on backspace/delete").

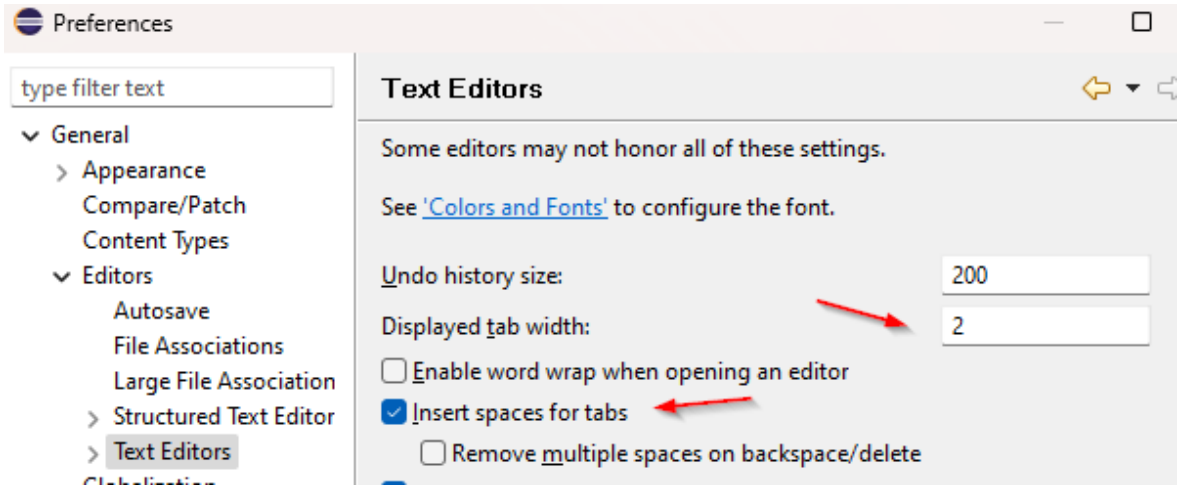


Figure 139 Source Code Indentation Setting

6.1.1.3 Error message in the Code

(General → Editors → Text Editors)

Error messages appear as icons to the left of the source code. To get the information about the error, you need to hover over the icon with the mouse. You can also use the option "Show code minings for problem annotations" to display the entire error message directly in the code. To do this, select the type of message.

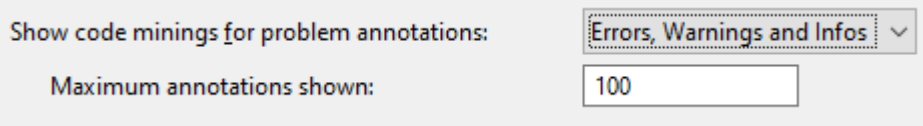


Figure 140 Example of Setting

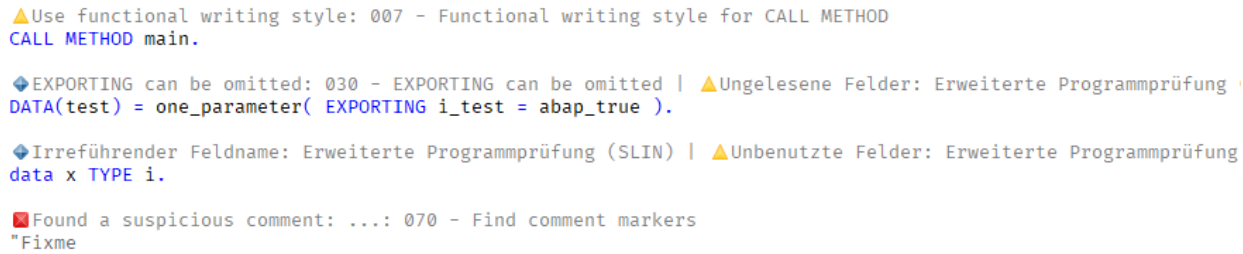


Figure 141 Result Image in Source Code

6.1.1.4 Spell checker: Comments

(General → Editors → Text Editors → Spelling)

If you don't write comments in English in Eclipse, the spell checker will give you a lot of red comments. You can deactivate this check via the settings ("Enable spell checking") or you can download the dictionary for your language.

6.1.1.5 Shortcuts

(General → Keys)

Configuration of the keyboard shortcuts in Eclipse that allow you to define your desired settings. In addition, you can display shortcuts if they have been triggered ("Through keyboard") or if there is a key combination for the executed action ("Through mouse

click"). This option is always useful if you want to give training courses, help colleagues familiarize themselves with Eclipse or familiarize yourself with the keyboard shortcuts.

See also:

Blog Post Useful Keyboard Shortcuts for ABAP in Eclipse

SAP Help Keyboard Shortcuts for ABAP Development

6.1.1.6 Debugging

(ABAP Development → Debug)

Possibility to specify the general debugger settings, but also to enable ("Enable debugging of system programs") of system debugging.

6.1.1.7 Color formatting

(ABAP Development → Editors → Source Code Editors → ABAP Keyword Colors)

To highlight important keywords in Eclipse, you can highlight them with additional color combinations. To do this, you can highlight individual keywords or select all ("Select all"). This makes it easier to identify important passages in the source code.

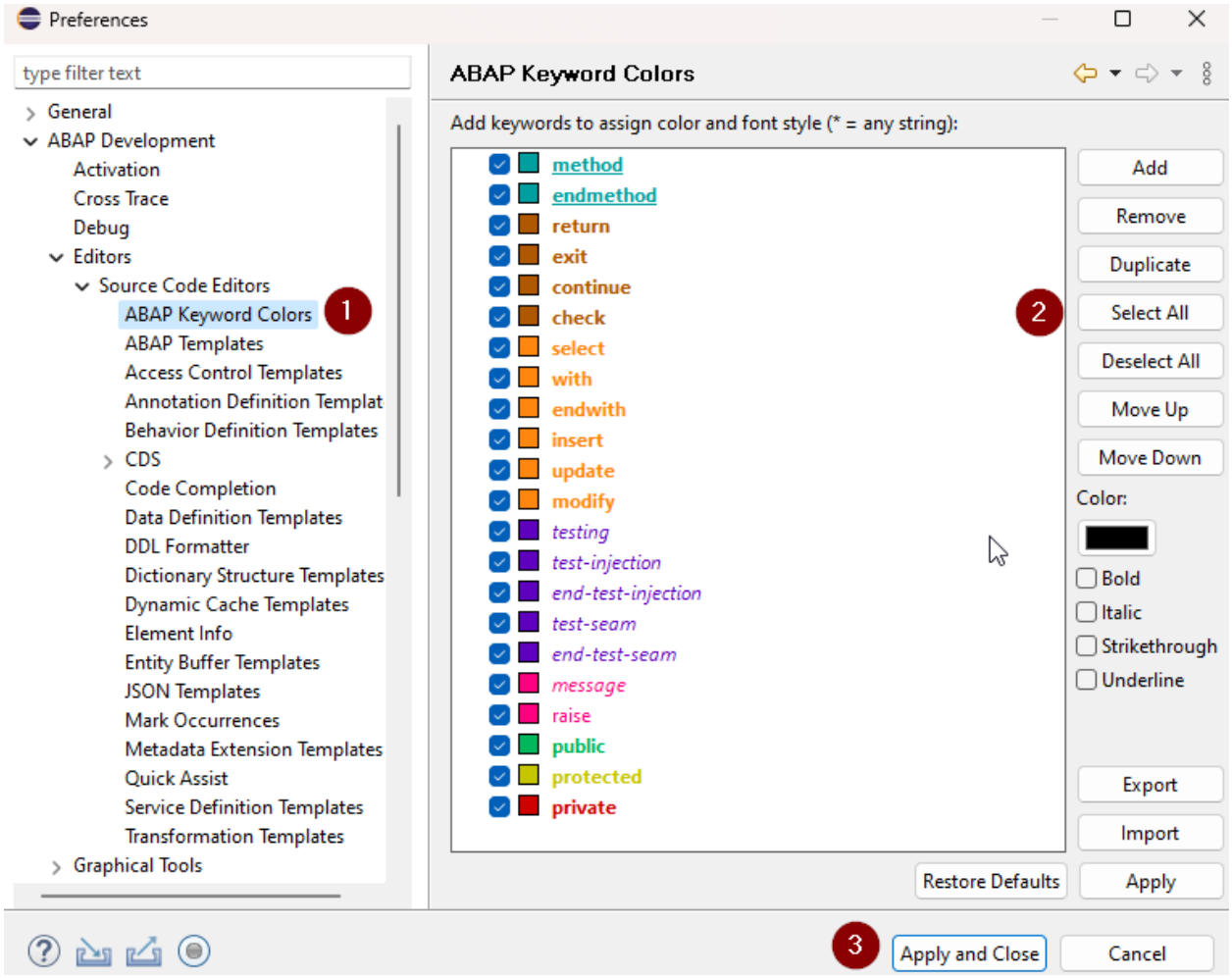


Figure 142 Color Settings to Highlight the Keywords in the Source Code

6.1.1.8 Code Templates

(ABAP Development → Editors → Source Code Editors → ABAP Templates)

For frequently used code fragments, SAP delivers templates that can be adapted to your own needs. New templates are also possible. The templates are inserted in the coding by entering the template name and autocomplete (**`CTRL+SPACE)`** .

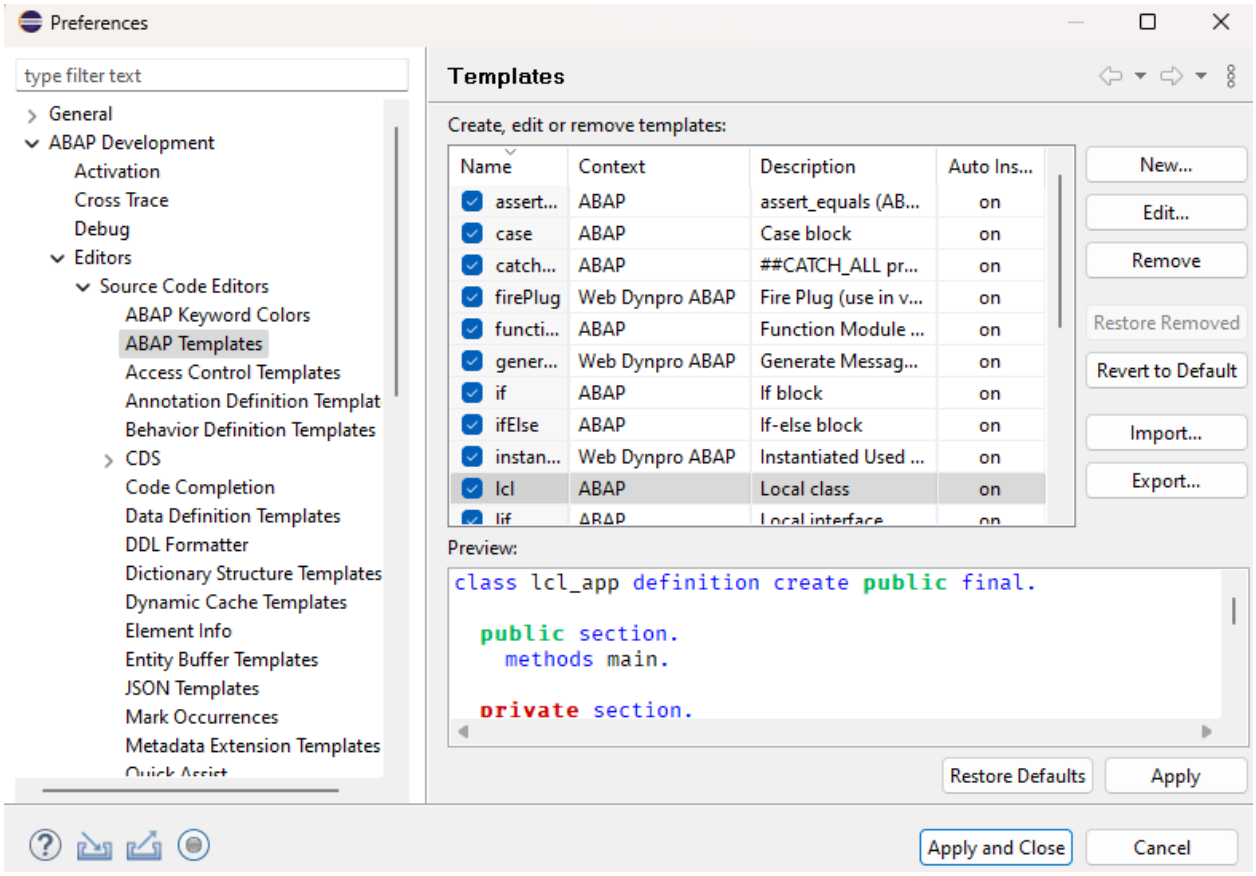


Figure 143 Administration of ABAP Templates in Settings

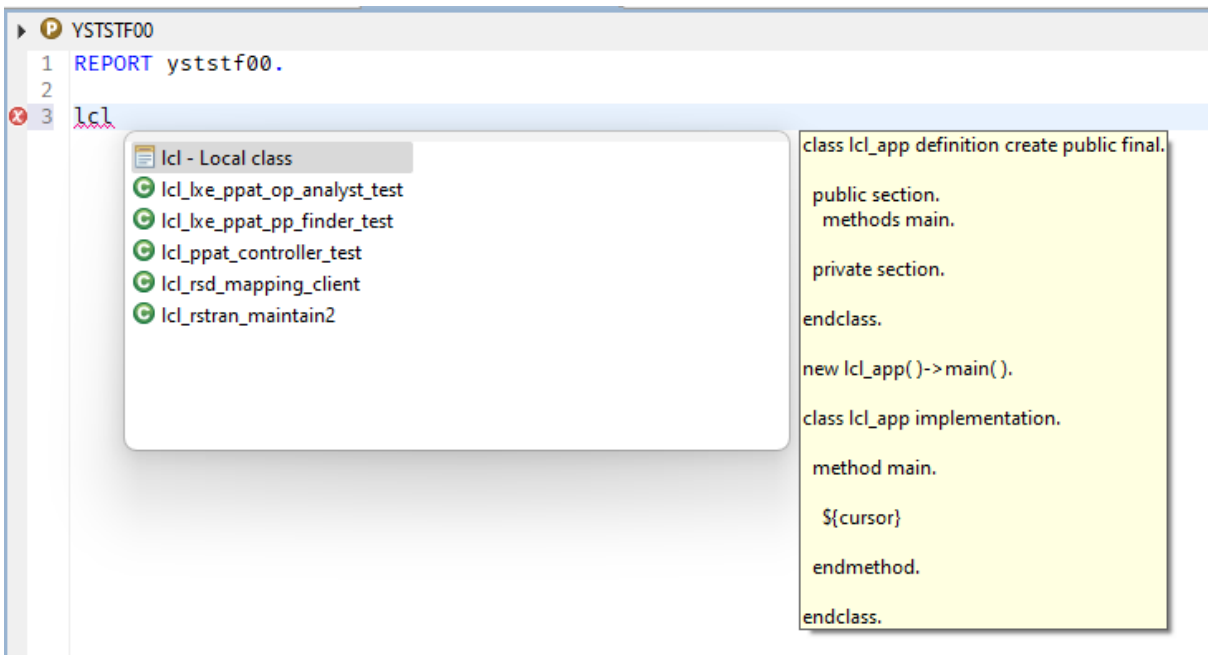


Figure 144 Inserting the Template into the Source Code

6.1.1.9 Alias for CDS Fields

(ABAP Development → Editors → Source Code Editors → CDS → Code Completion)

When creating a Core Data Service (CDS), when a table is included, the field names are provided without an underscore and in Camel Case with an alias. With the option ("Add aliases for table fields ..."), this is done by default when inserting via "Insert all elements" (default setting).

6.1.1.10 Auto-complete

(ABAP Development → Editors → Source Code Editors → Code Completion)

By default, Eclipse adds parentheses and quotation marks at the end of an expression ("Automatically close brackets and literals") and inserts spaces inside parentheses ("Add additional whitespace inside ..."). If you are bothered by these options, they can be disabled here. Furthermore, you can also have non-keywords suggested by Eclipse ("Also suggest non-keywords"), which will then also suggest variable names, for example.

6.1.1.11 Search

(ABAP Development → Search)

Here you can make settings in the search dialog (**`CTRL+SHIFT+A`**), e.g. whether the old search pattern is continued ("Use pattern from previous search") or the number of hits displayed ("Maximum number of results"). However, the type of the object ("Display object types") and in which package ("Display packages") it is located is also important.

6.1.2 Project-specific Settings

You can find the system-specific settings by right-clicking on the ABAP project under "Properties". The structure of the window is similar to that of the global settings and can be operated immediately.

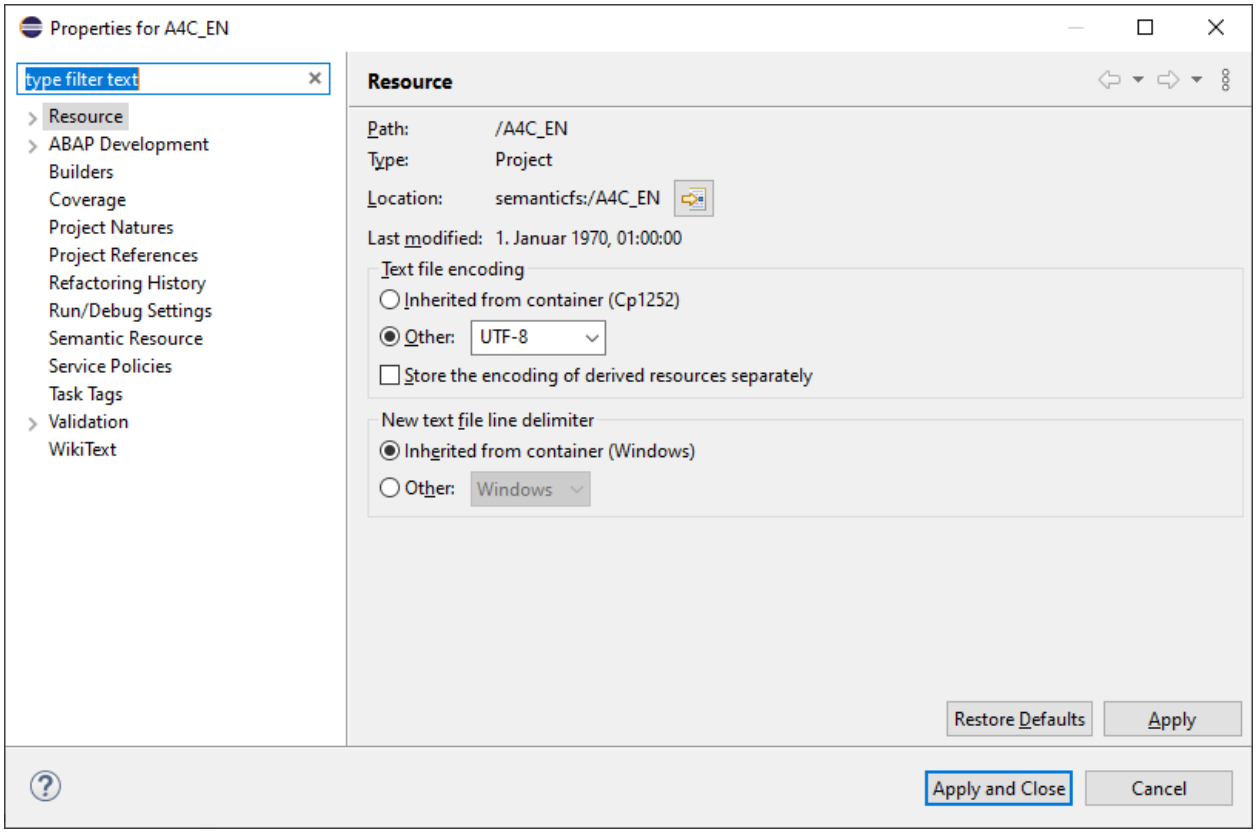


Figure 145 Getting Started with Project-specific Settings

6.1.2.1 External Debugging

(ABAP Development → Debug)

With this option you can set for which user debugging is active. This can be the current user ("Logon User") or another user ("User") if you want to perform external debugging.

Breakpoints in Eclipse are automatically active for all access types (SAP GUI, ABAP Unit, HTTP, RFC). There is no explicit "external debugging" anymore.

6.1.2.2 Pretty Printer

(ABAP Development → Editors → Source Code Editors → ABAP Formatter)

As with Pretty Printer, you make the formatting settings here when the Code Formatter (**`SHIFT+F1`**) is executed. A standard for this would be, for example:

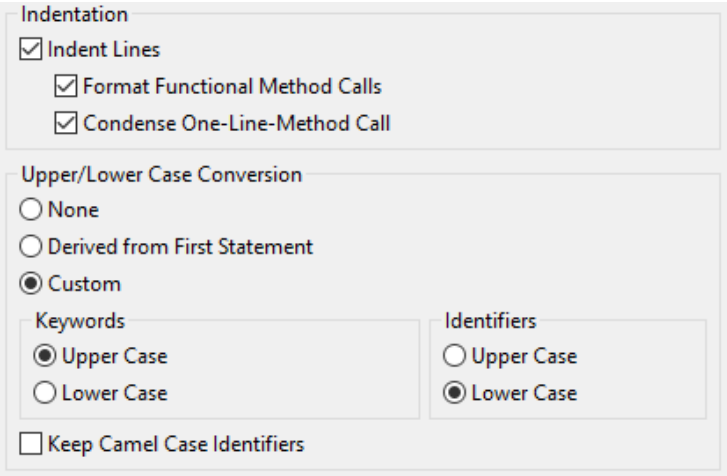


Figure 146 Possible Settings for the Pretty Printer / ABAP Formatter

(Settings depending on the backend system version, e.g. "Keep Camel Case Identifiers" is only available in S/4HANA)

6.2 Views and Perspectives

Definitions of terms: see Chapter 1

Working with the different views: see chapter 4

6.2.1 Views

All the information you see and work with is displayed in views, such as the Project Explorer or Editor. Views can be moved to the screen as desired by touching (holding the mouse button, "drag") and dragging the table tab of the view.

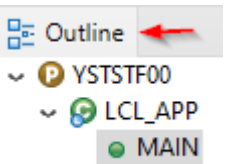


Figure 147 Move the View via the Label/Tab

As you move, you'll see a preview of the new layout.

Figure 148 The Markings Indicate the Placeability of the Window

After dropping, the view is moved to this point.

Views can also be placed outside the Eclipse screen and exist there. This is especially useful when working with multiple monitors.

If you drag the view next to another table tab, the views are stacked, i.e. combined in a view group.

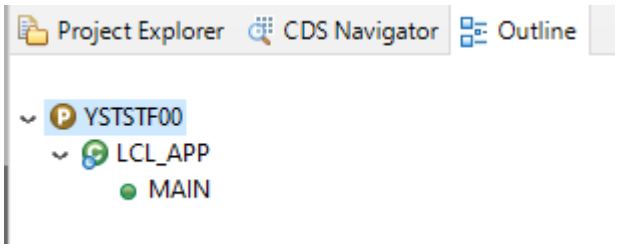


Figure 149 Display of Stacked Views

View groups can be minimized and restored together.

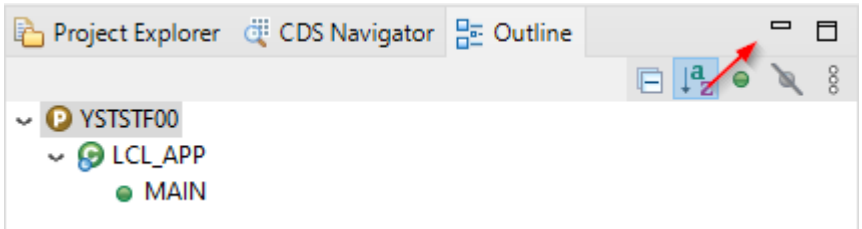


Figure 150 Minimizing View Groups

The result is that the view group is minimized at the edge of the screen. You can use the "Restore" button to restore the view group.

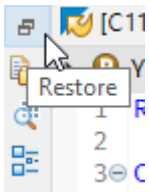


Figure 151 Restoring the View Groups

After double-clicking on a table tab, the view is displayed in full screen. This is particularly useful for editor views or large screens that are displayed in the SAP GUI View. Double-clicking on the table tab again reduces the size of the view.

Views that are no longer needed can be closed using the close icon – for example, the Feature Explorer after you have worked through the tutorial.

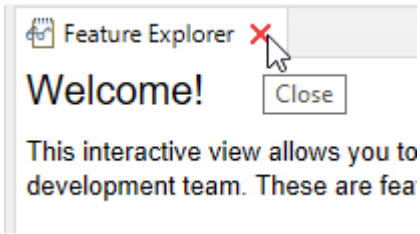


Figure 152 Closing a View

You can add new or accidentally closed views to a perspective at a later date.

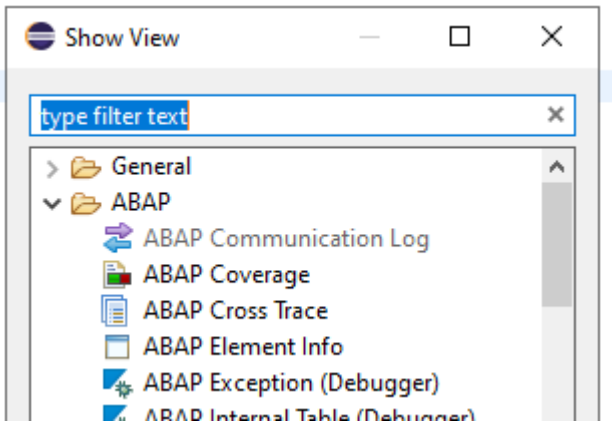


Figure 153 Displaying a View

In this way, non-ABAP views (see "Recommended Additional Views") can also be assigned to the ABAP perspective.

6.2.2 Perspectives

The arrangement of all views on the screen is stored in a perspective. For different activities, separate perspectives are delivered, which can be adapted as desired.

In the ABAP development tools, the ABAP and debugging perspectives are mainly used, between which you can switch as you wish.

Figure 154 Switching Between Different Perspectives

Tip: Especially in the first few weeks of ADT use, people tend to forget to return to the ABAP perspective after a debugging session.

If you have adjusted your perspective "too much", you can use the menu to restore the perspective to its delivery state.

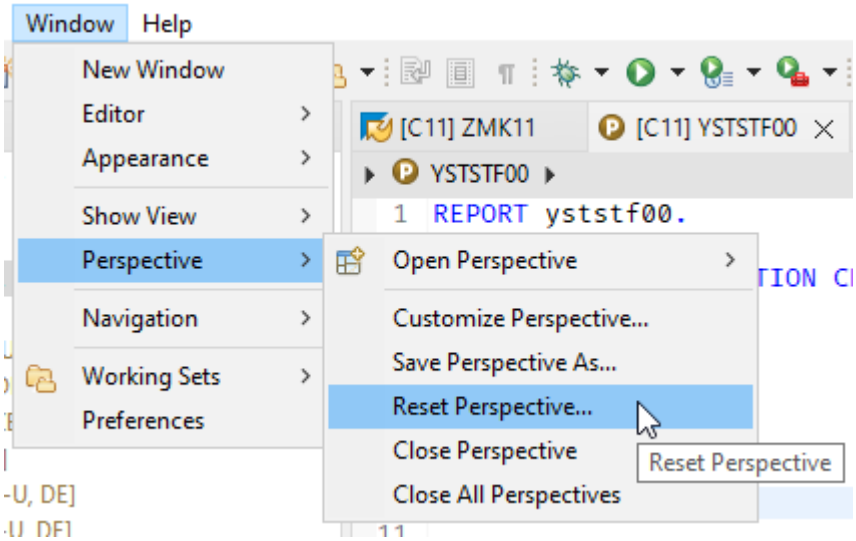


Figure 155 Reset a Perspective

6.2.2.1 Own Perspectives

You can also define your own perspectives. This is especially useful if you work with different monitor configurations (e.g. two monitors). This allows the size and arrangement of the views to be adjusted. Your own perspective can be created via "Save Perspektive As..." can be stored.

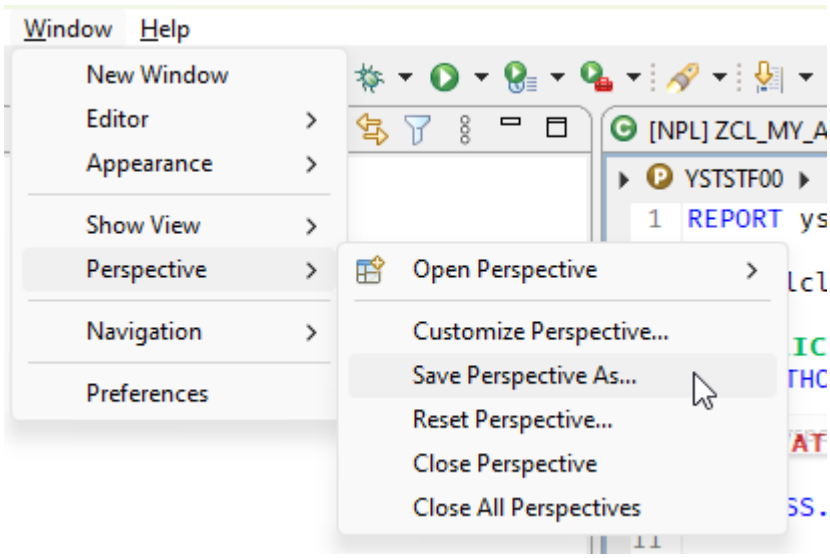


Figure 156 Save a Perspective

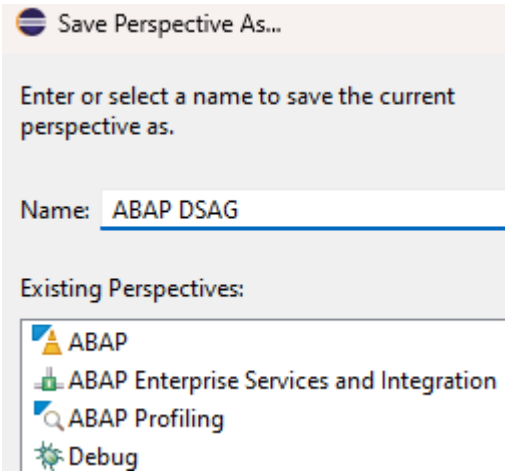


Figure 157 Naming the new Perspective

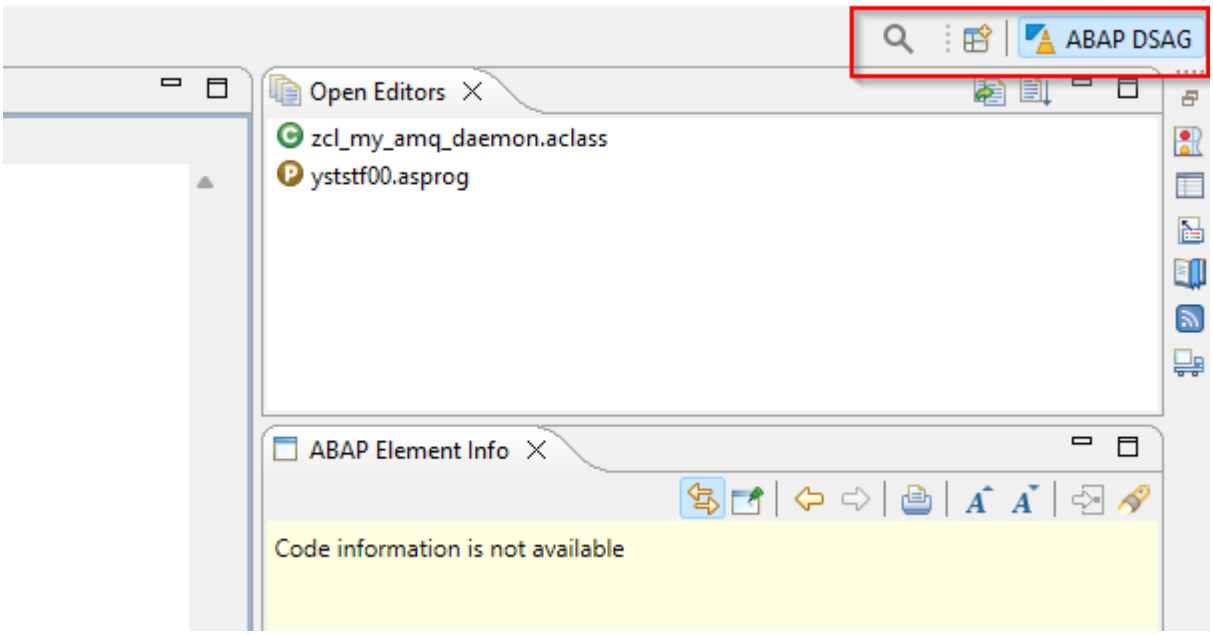


Figure 158 New Perspective with Name

6.3 Recommended additional views

Views already installed via Eclipse or ABAP Development Tools:

- ABAP Element Info
- Minimap

Views that can be installed via Eclipse Marketplace

- Open Editors

6.4 Suggestions for distribution

Setting up a virtual workplace is as individual as setting up a real one. In addition to personal preferences, it also depends, for example, on the size of the monitor, how many views can be displayed at the same time. Therefore, only very subjective suggestions can be made here.

<table>
<tr><td>Project Explorer</td><td rowspan="3">Editor</td><td>Open Editors</td><td rowspan="3"> </td></tr>
<tr><td rowspan="2"></td><td>ABAP Element Info</td></tr>
<tr><td>Outline</td><td>Minimap</td></tr>
</table>

Figure 159 Possible Setting of the ABAP Perspective

<table>
<tr><td rowspan="3">Project Explorer</td><td rowspan="2">Editor</td><td>Variablen</td></tr>
<tr><td>Callstack (“Debug”)</td></tr>
<tr><td colspan="2">Interne Tabellen</td></tr>
</table>

Figure 160 Possible Setting of the Debugger Perspective

7 Plug-ins

7.1 Introduction

Eclipse is an integrated development environment (IDE) that consists of many small units called plug-ins. If you look at the Eclipse variants Eclipse IDE for Java Developers or Eclipse IDE for C/C++ Developers as examples, then these preconfigured packages are only collections of plug-ins that have been developed for a specific purpose.

The ABAP Development Tools (ADT) are categorically exactly the same, i.e. a collection of plug-ins, and thus provide ABAP developers with a modern development tool. It is precisely this modular structure that now enables every developer to create their own plug-ins in order to further adapt Eclipse and/or ADT and thus, for example, simplify recurring tasks or provide functions that are not offered by ADT and can only be found in the ABAP Workbench.

Such plug-ins can consist of pure UI code, such as the ABAP Favorites plug-in, which brings functions of the Easy Access Menu (SAP GUI) to Eclipse. However, they can also be more extensive and require extra ABAP code on the SAP system. An example of this is the ABAP Code Search plug-in, which is comparable to the SAP GUI transaction CODE_SCANNER.

7.2 Useful Open-Source Plug-ins

7.2.1 Open Editors

Provides a new view that shows all open editors in Eclipse. This view also offers the possibility to adjust the sort order of the editors.

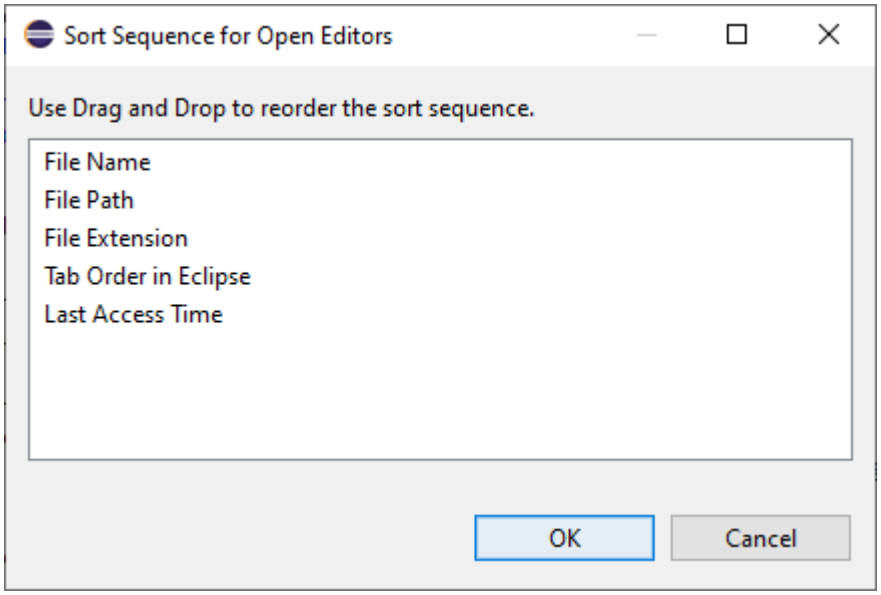


Figure 161 Dialog for Adjusting the Sort Order of Open Editors

Prerequisites Eclipse:

- Eclipse IDE for Java Developers

Links:

- Source-Code on GitHub
- Eclipse Marketplace

7.2.2 AnyEdit Tools

Offers many options for editing text/source code:

- Conversion of text to lowercase/uppercase letters
- Conversion of text from Pascal to Camel notation
- Sorting of the selected rows (alphabetically, numerically, by line length)
- ...

In addition, the plug-in also offers many options for comparing text.

- Comparing an editor to text on the clipboard
- Comparing an editor with any other editor
- Comparing an editor with an external file

All possible operations are available via the context menu of an editor.

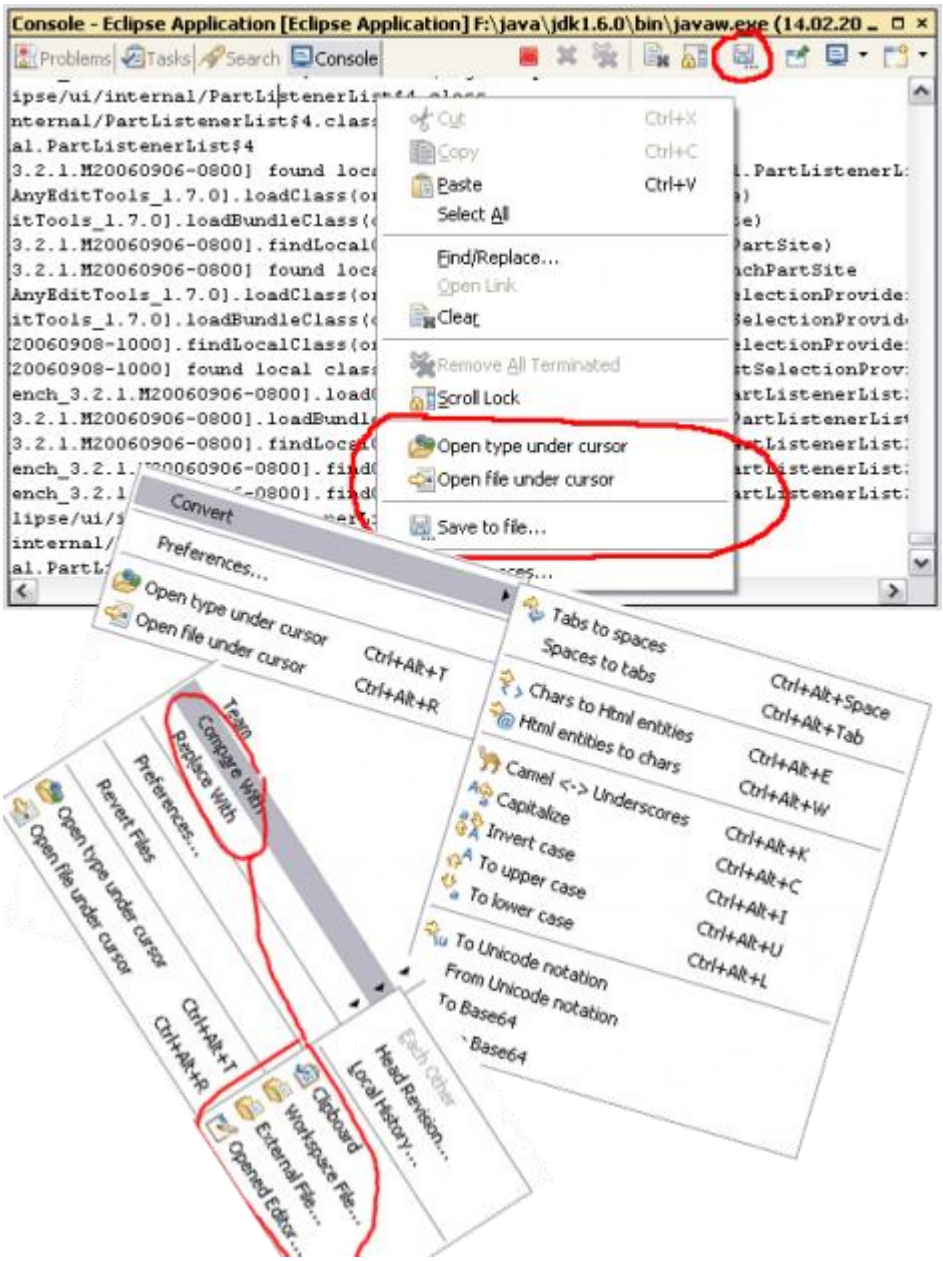


Figure 162 Examples of Available Operations in the Context Menu

Prerequisites Eclipse:

- Eclipse IDE for Java Developers

Links

- Source-Code on GitHub
- Eclipse Marketplace

7.2.3 PDE Tools

This plug-in extends Eclipse with useful tools for plug-in developers:

- Preview icon files directly in Project Explorer
- Generation of Java constants for icon folders
- Screenshot tool for UI elements in Eclipse Workbench

However, there are features that are also helpful outside of plug-in development:

- Extended clipboard history
- Direct launch of a new Eclipse window with a specific workspace

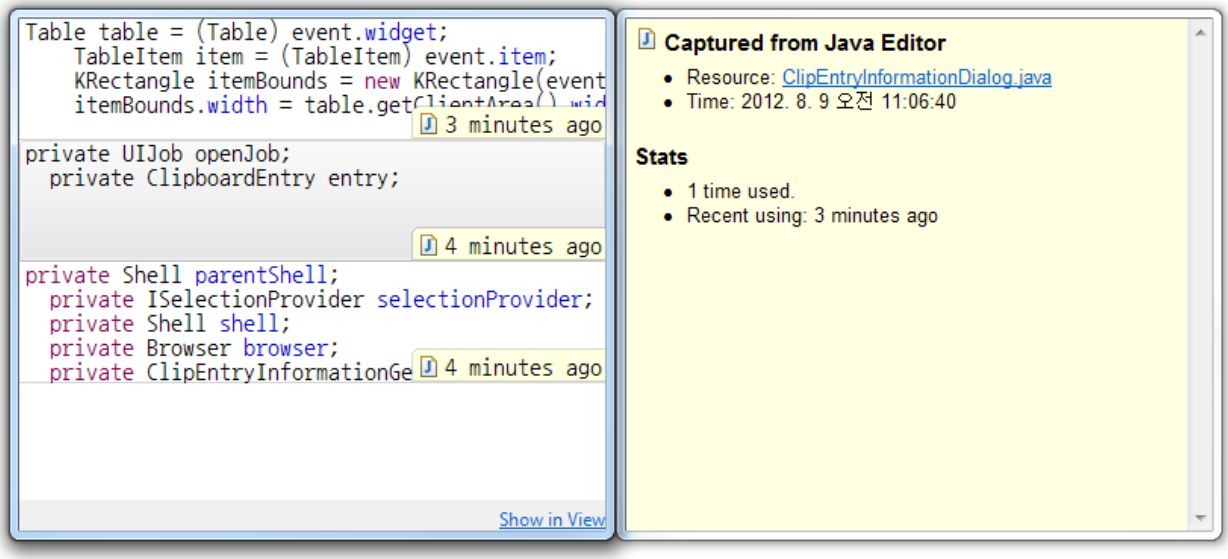


Figure 163 Clipboard History (Keyboard Shortcut Ctrl+Shift+V)

Links:

- Source-Code on GitHub
- Eclipse Marketplace

7.3 Useful Open-Source ADT Plug-ins

7.3.1 ABAP Favorites

The ABAP Favorites plug-in was developed to map the functionality of the SAP GUI User Menu. In this menu, each user can add transactions, reports or URLs to their favorites and structure them according to their personal preferences.

The plug-in installation brings two new views, Favorites and Favorite DevObjects, available via Windows → Show View → Others. Both views offer a filtered tree view in which the favorite objects can be managed.

The difference between the two views lies in the possibilities for creating the folders (containers). The Favorites View allows two types of these folders: "Standard" for transactions, reports and URL and "DevObject" for managing development objects such as classes, function modules, CDS views, etc. In the case of Favorite DevObjects, only "DevObject" folders can be created. The division of these views allows the user to choose whether he wants to mix all folders or manage them according to the separation described.

If you compare the "Standard" folder with the "DevObject" folder, apart from the selectable object types, the biggest difference is that with "Standard" the objects are executed by double-clicking. Double-clicking in a DevObject folder opens the selected object.

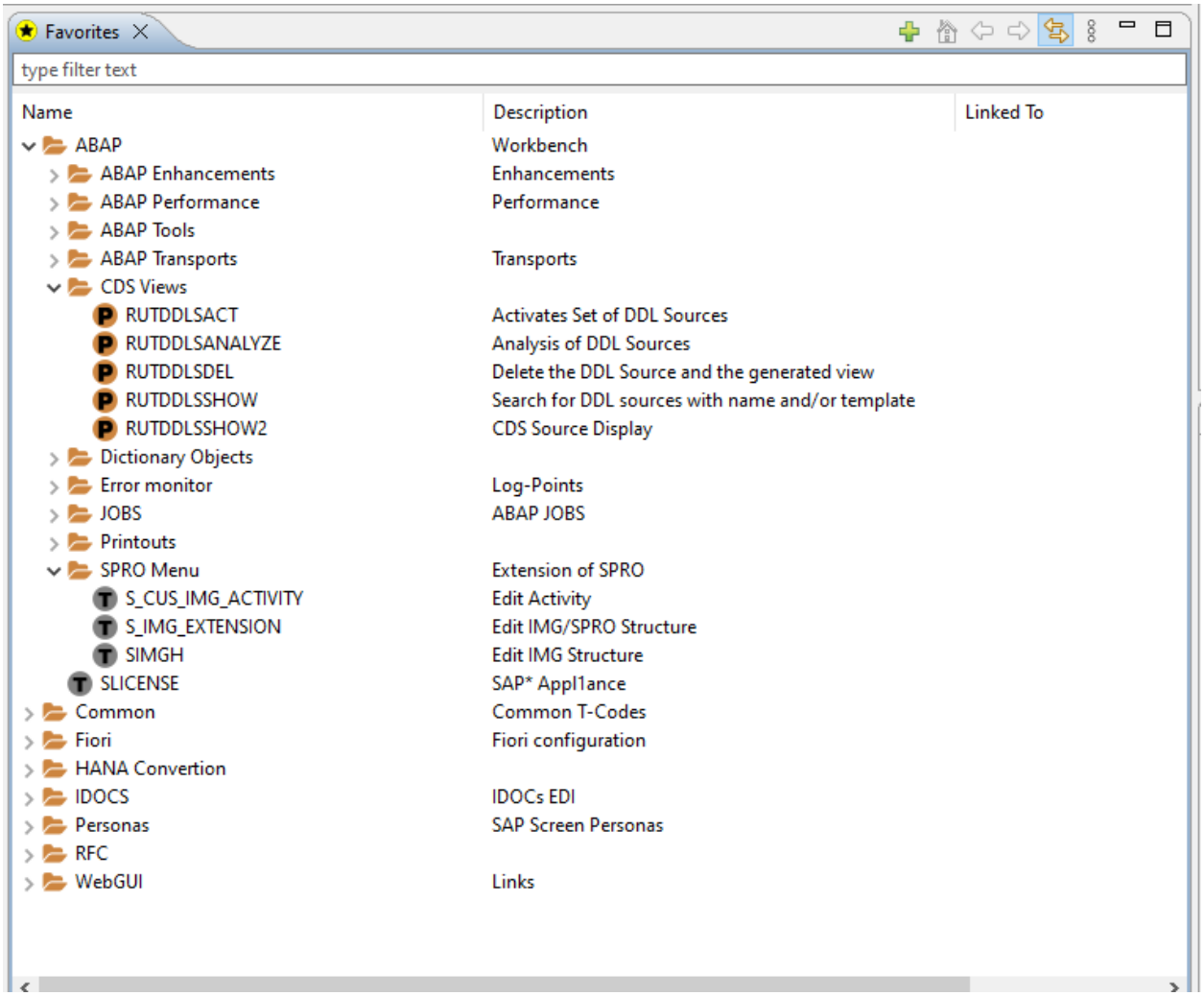


Figure 164 ABAP Favorites View

To add new objects to the favorites, you can use the context menus of the Favorites View, the ABAP Editor, or the Project Explorer.

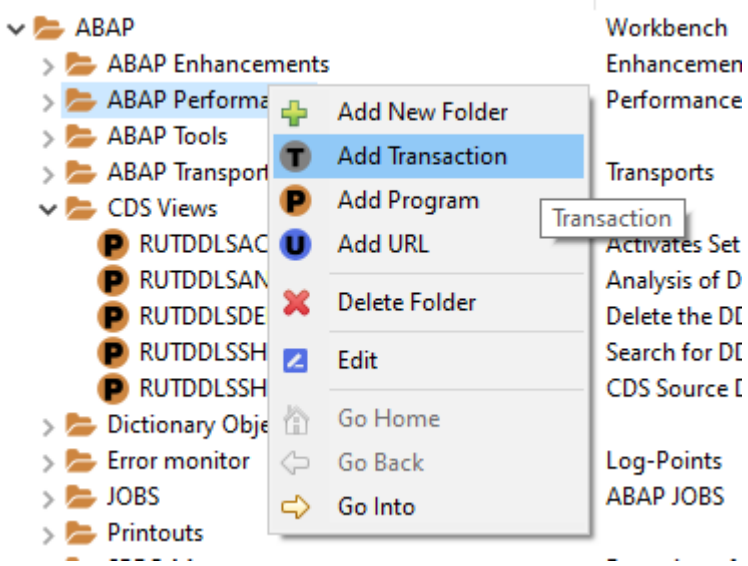


Figure 165 Context Menu of a Folder in ABAP Favorites View

Prerequisites:

- Eclipse IDE for Java Developers
- ADT

Links:

- Source-Code on GitHub
- Eclipse Marketplace

7.3.2 ABAP Continuous Integration

AbapCI is an open source Eclipse plug-in that provides various Continuous Integration (CI) tools for ABAP development with Eclipse. The plug-in is based on ADT's CI capabilities.

The plug-in provides the following functions:

- Automatic unit test runs
- Automatic ATC runs
- Visualization of the source code status on the user interface
- Different color schemes for each ABAP project
- Automatic source code formatting
- Shortcut for abapGit

- Triggering Jenkins from Eclipse (experimental)

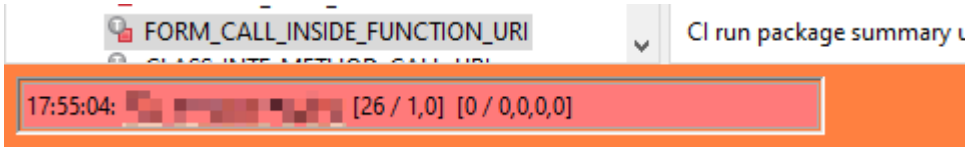


Figure 166 Colored Highlighting of the Status Bar per Project + Test Status

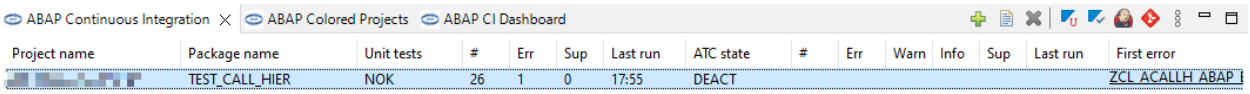


Figure 167 Management of Packages for which Unit Tests and/or ATC Test Runs are Scheduled

More information can be found in the GitHub repository.

Prerequisites:

- Eclipse IDE for Java Developers (<= 2022-06, installation with newer versions currently only possible with workaround; see issue on GitHub)
- ADT

Links:

- Source-Code on GitHub
- Eclipse Marketplace

7.3.3 ABAP ADT Extensions

This plug-in extends ADT with several additional functionalities.

7.3.3.1 Automatic login to SAP systems

The "Automatic Login" functionality allows the developer to manage their user/password combinations within Java's Secure Storage. From a security point of view, these should be encrypted.

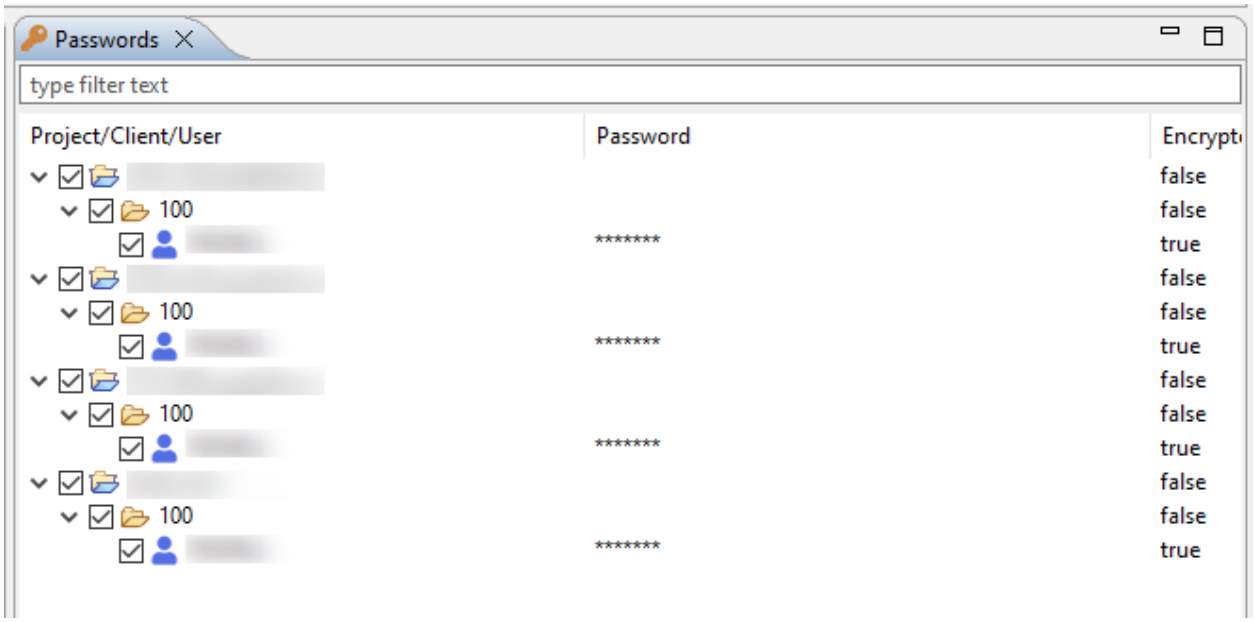


Figure 168 View for Managing the Stored Access Aata of ABAP Systems

Depending on the settings of the plug-in, you can automatically log in to any of the selected on-premise SAP systems that are mapped using ABAP/BW projects. The passwords can be maintained when the project is created or later via the password view.

7.3.3.2 Changing ABAP Project Attributes

The context menu of the Project Explorer can be used to change the project attributes (client, user, language) for ABAP/BW projects. In addition, the breakpoint users can be set.

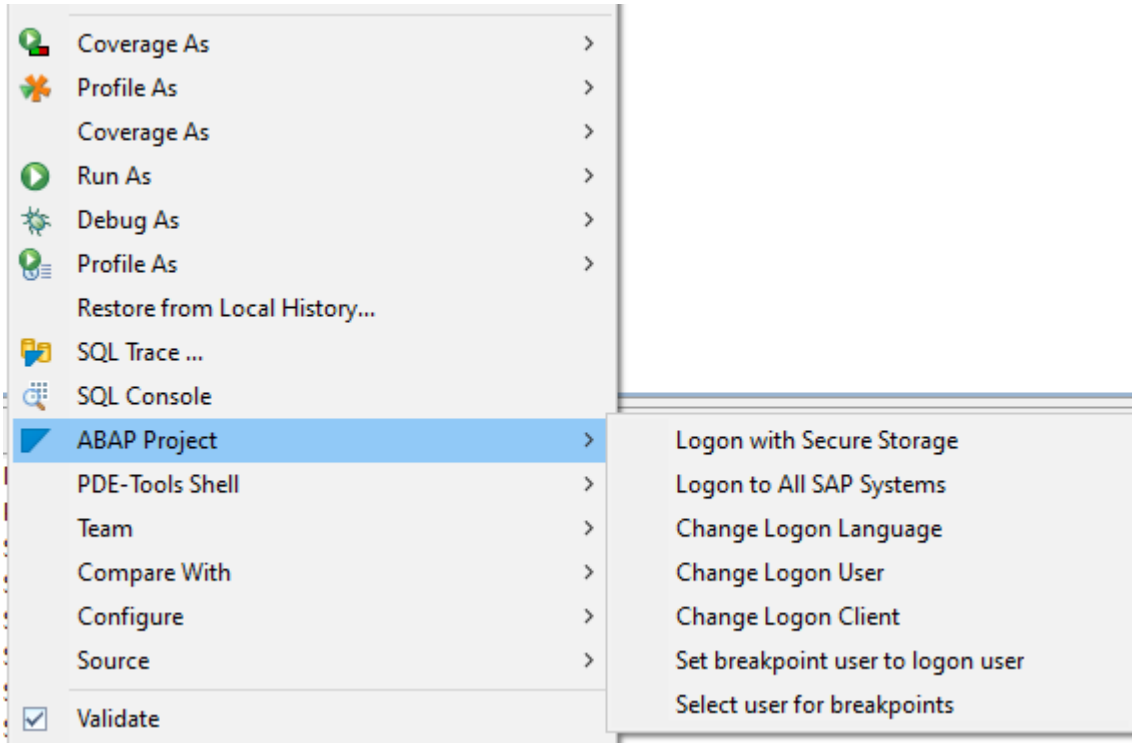


Figure 169 Context Menu on Project in Project Explorer

7.3.3.3 Input field for executing transaction codes

After installation, a toolbar appears in the lower right area of Eclipse with an input field for transaction codes. The field can be operated by mouse or via the shortcut **`Shift+F8`** . After entering the code and confirming with Enter, the transaction is executed in the currently active project.

Figure 170 Status Bar in the Eclipse Window

Prerequisites:

- Eclipse IDE for Java Developers
- ADT

Links:

- Source-Code on GitHub
- Eclipse Marketplace

7.3.4 ADT Classic Outline

This plug-in adds a new view called "Classic Outline" to your interface, which in a sense maps the SE80 object list. In most cases, this view can replace the built-in ADT outline. The displayed object list can be filtered and allows the object attributes to be viewed, similar to the built-in ADT outline. By double-clicking (or single clicking depending on the settings) you can navigate to the selected object.

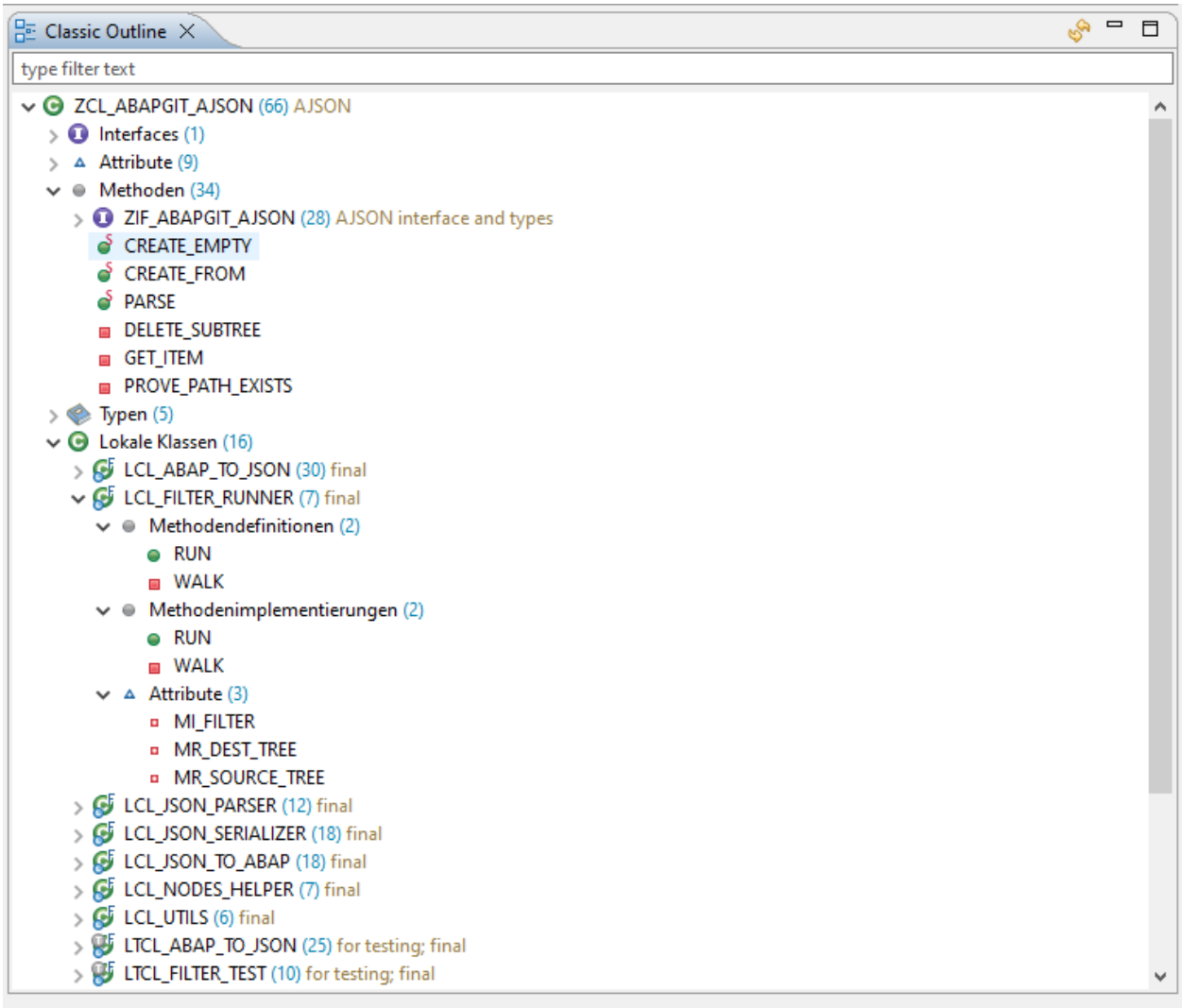


Figure 171 Classic Outline View

Prerequisites:

- Eclipse IDE for Java Developers
- ADT

Prerequisites ABAP:

- SAP NetWeaver 7.40 SP08 or newer
- abapGit repository ADT Classic Outline Backend must be installed

Links:

- Source-Code on GitHub: Frontend and Backend
- Eclipse Marketplace

7.3.5 ABAP Quick Fix

Quick fixes are part of the Eclipse IDE. In the ADT standard, they are processed in the backend system and can be used by the user if required using the shortcut **CTRL+1** . The ABAP Quick Fix plug-in provides additional quick fixes that are processed directly from the Eclipse environment.

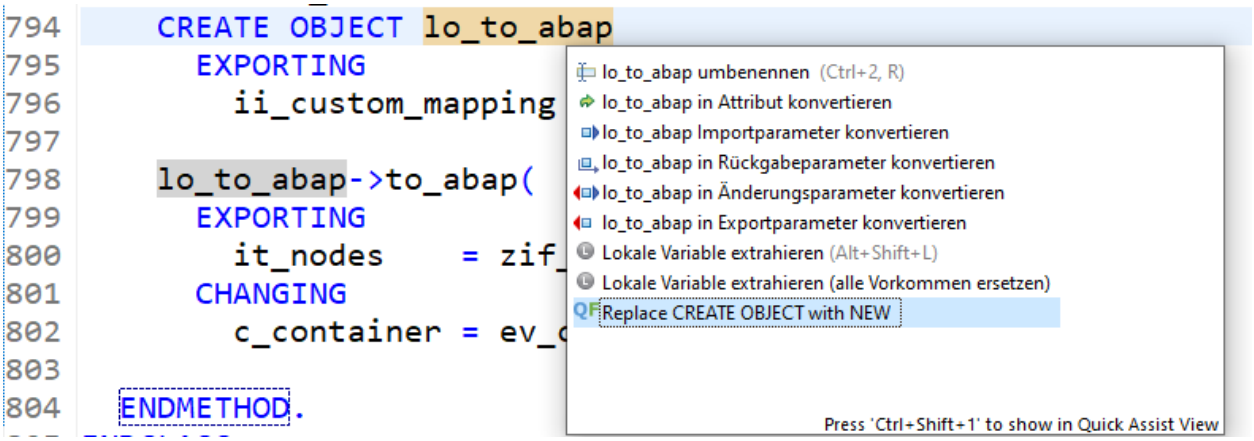


Figure 172 ABAP Code Before Quick-Fix Execution

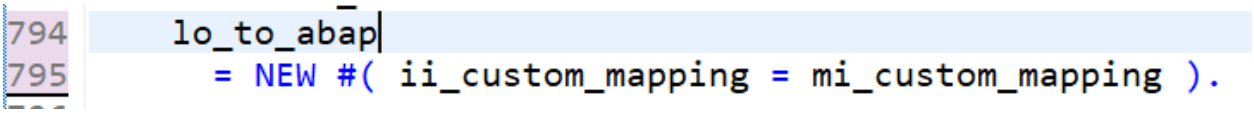


Figure 173 ABAP Code After Quick-Fix Execution

For an excerpt of the available features, see the following list:

- REplace READ TABLE with ASSIGN, REF#, Table Expression or line_exists.
- Replace CALL METHOD with the direct call.
- Replace MOVE with direct assignment.

- Change APPEND TO to APPEND VALUE#( ) TO.
- Replace CREATE OBJECT with NEW.
- Replace “full line comments” of statement.
- Omit the self-reference ME->.
- Replace the operators EQ, NE, GT, GE, LT, LE with =, <>, >, >=, <, <=
- Indent operators in the selected area accordingly.
- Indent TYPE and LIKE in the declaration block corresponding to the variable.

Prerequisites:

- Eclipse IDE for Java Developers
- ADT

Links:

- Source-Code on GitHub
- Eclipse Marketplace

7.3.6 ABAPQuickFixS4Conversion

This plug-in is a very nice example of developer collaboration within the community. ABAPQuickFixS4Conversion is an extension of the ABAP Quick Fix Plug-in from SAP and adds the following functionalities:

- Convert SELECT SINGLE to SELECT ... UP TO 1 ROWS ... ORDER BY
- Adjust the custom ORDER BY list for any table
- Change SELECT SINGLE to modern SQL style
- Convert SELECT/ENDSELECT to modern SQL style
- Transform MOVE_CORRESPONDING to CORRESPONDING #( )

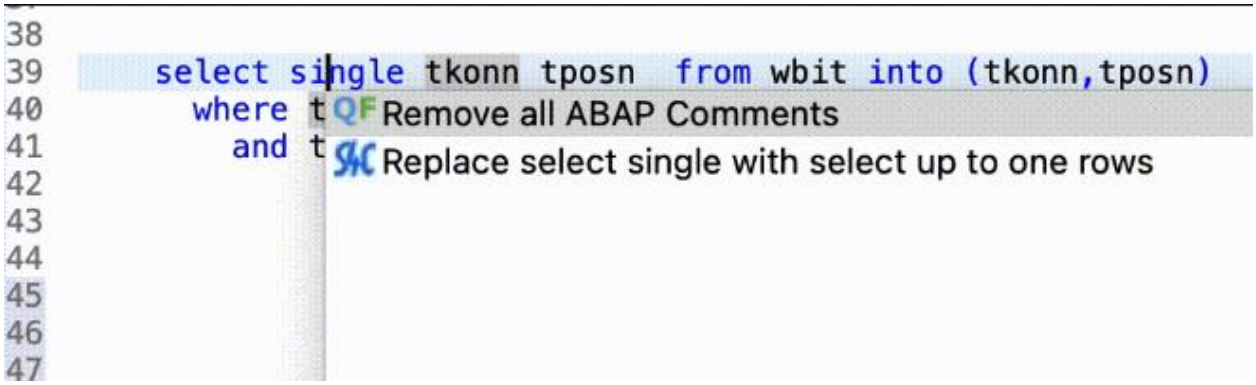


Figure 174 Example of Quick Fix Availability on a SELECT Statement

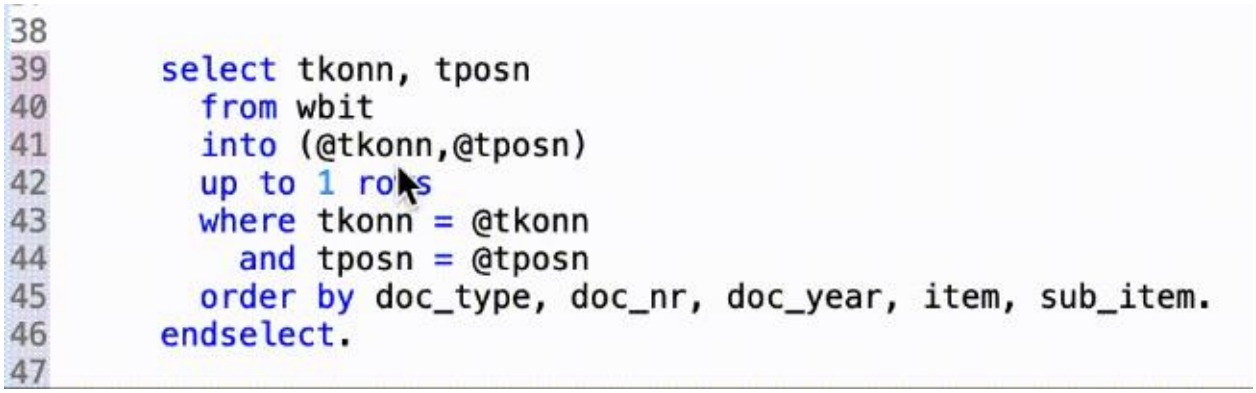


Figure 175 SELECT Statement after Applying the Quick Fix

Prerequisites:

- Eclipse IDE for Java Developers
- ADT
- ABAP Quick Fixes plugin

Links:

- Source-Code on GitHub
- Eclipse Marketplace

7.3.7 ABAP Tags

The ABAP Tags plug-in allows you to create tags that can then be added to any development object. The tags and their assigned objects are persisted on the respective ABAP system. This makes it easier for other users to access tagged objects. In general, the plug-in allows you to create tags either in the global or in the user-specific scope. User-specific tags can also be shared with others, making collaboration easier.

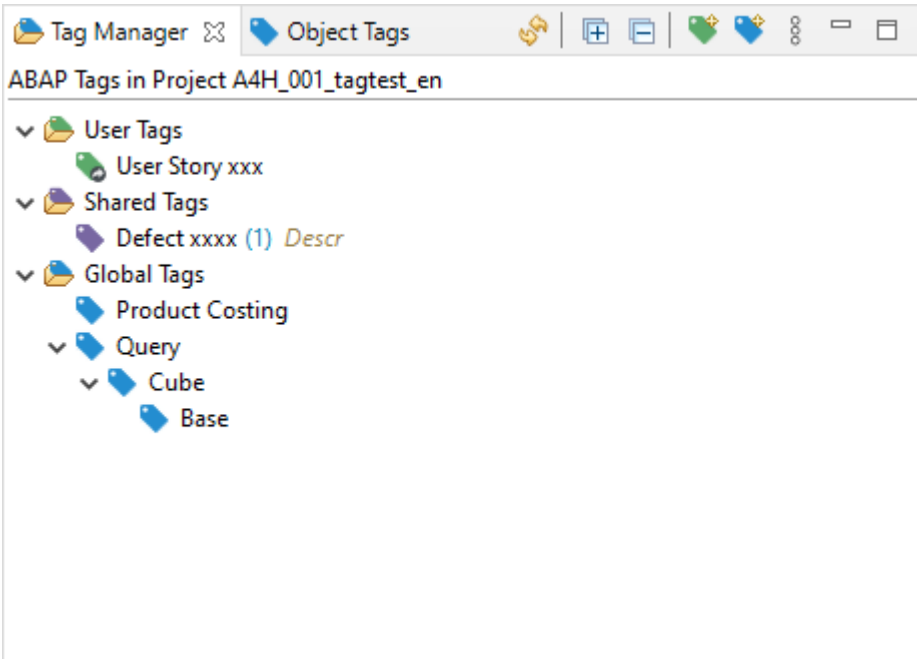


Figure 176 View “Tag Manager”

Tagging objects is intuitively possible via the context menu from the editor or the Project Explorer. The tagged objects can then be searched for and displayed either from the "Tag Manager" view using the context menu action, or via the "ABAP Tagged Object Search" integrated in the "Search" dialog.

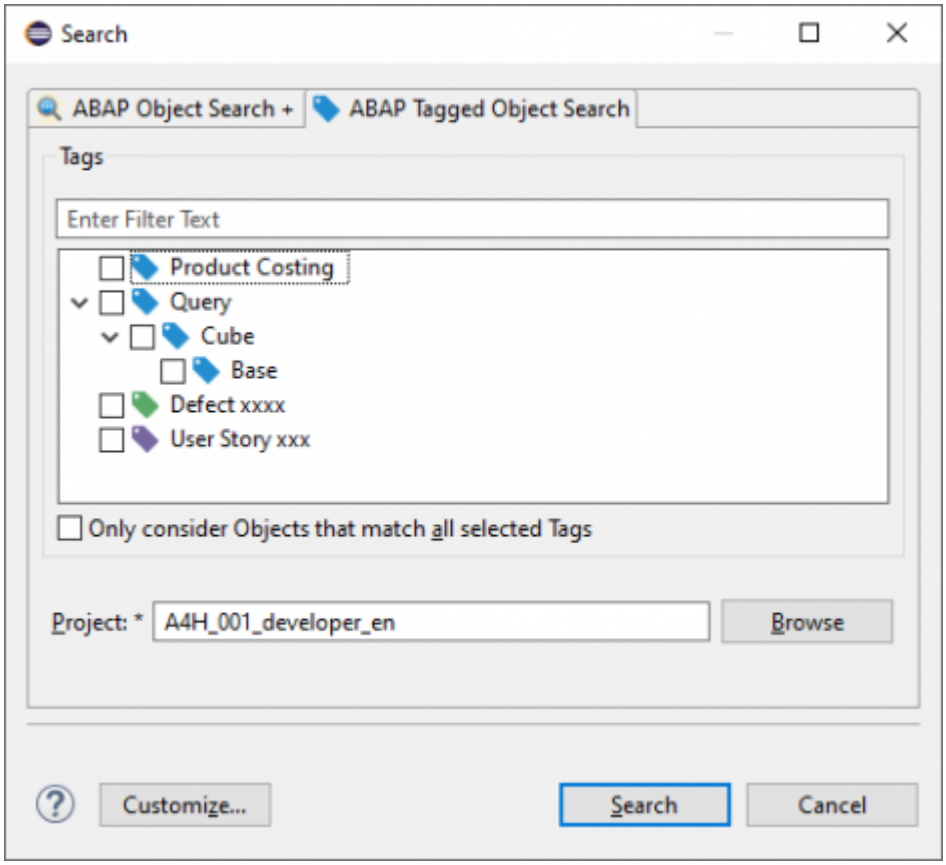


Figure 177 Search-Dialog with “ABAP Object Search”

Prerequisites Eclipse:

- Eclipse Platform Runtime or Eclipse IDE for Java Developers
- ADT

Prerequisites ABAP:

- SAP NetWeaver 7.40 SP08 or newer
- abapGit repository abap-tags-backend must be installed

Links:

- Source-Code on GitHub
- Eclipse Marketplace (Marketplace Client must be installed)

7.3.8 ABAP Search and Analysis Tools

This plug-in extends ADT with additional search and analysis capabilities for the following object types:

- Class/Interface
- Database table/view
- CDS View

The search functions are integrated into the Eclipse search dialog (Ctrl+**H**). The operation is similar to the "Open ABAP Development Object" dialog (**Ctrl+Shift+A**). The object type can be changed using a dropdown. Among other things, this controls the available filters in the "Search Filters" field.

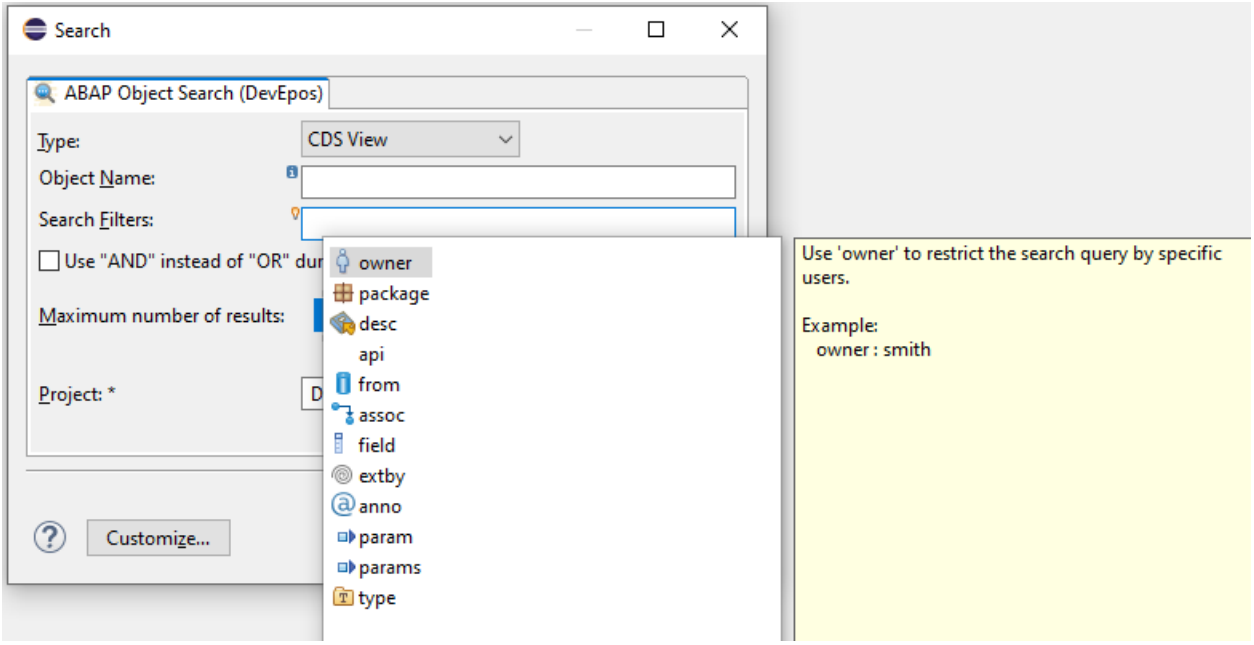


Figure 178 Search Dialog on Page "ABAP Object Search"

In addition to the search functions, the plug-in also provides the "CDS Analyzer" view, which allows the following analyses on CDS views:

- Top-Down
- Evaluation of all used entities of a CDS View
- Where-used list of database entities as data source ("*select from*" or "*association")*

- Field-level analysis
  - Top-down (origin determination)
  - Bottom-up (use of a field in fields of other CDS views)

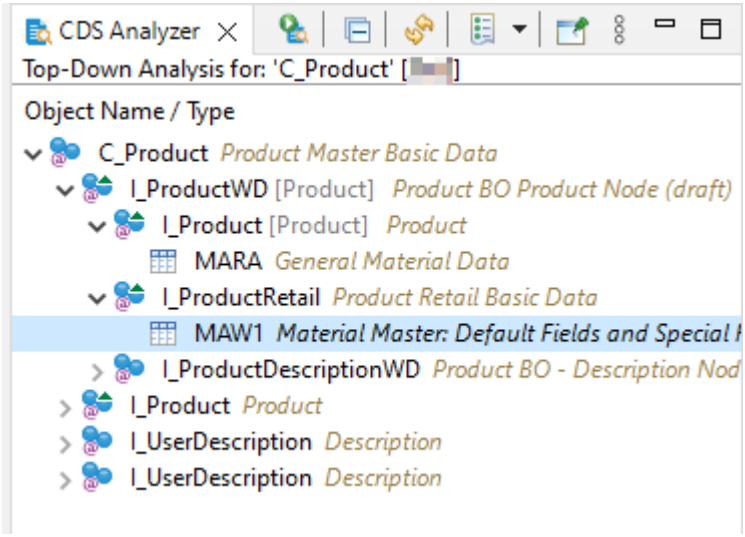


Figure 179 View “CDS Analyzer” - Top-Down-Analysis

Prerequisites Eclipse:

- Eclipse Platform Runtime or Eclipse IDE for Java Developers
- ADT

Prerequisites ABAP:

- SAP NetWeaver 7.40 SP08 or newer
- abapGit repository abap-search-tools must be installed

Links:

- Source-Code on GitHub
- Eclipse Marketplace (Marketplace Client must be installed)

7.3.9 ABAP Code Search

This plug-in brings the well-known SAP GUI transaction CODE_SCANNER to Eclipse. As the name suggests, the "ABAP Code Search" is also integrated into the Eclipse Search dialog. In addition to the use of regular expressions, there are also special

search modes such as: *Single Pattern mode* or *Sequential Matching*. Other features of the search are:

- Parallel execution (optionally controllable per user)
- Search can be stopped at any time
- Complete system search possible, since only small packets are processed on the application server at a time
- ABAP Tags plug-in Tags can be used for object selection → requires installation of the ABAP Tags plug-in

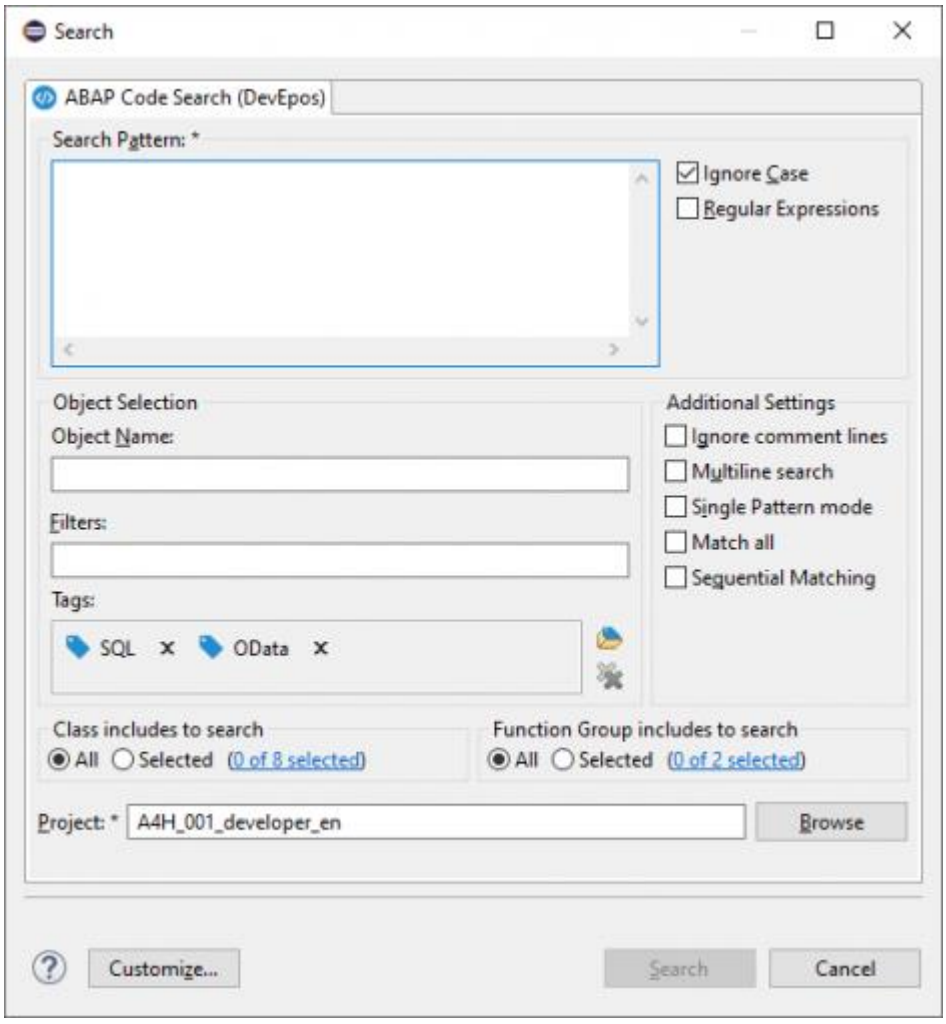


Figure 180 Search-Dialog with "ABAP Code Search"

Prerequisites Eclipse:

- Eclipse Platform Runtime or Eclipse IDE for Java Developers
- ADT

Prerequisites ABAP:

- SAP NetWeaver 7.40 SP08 or newer
- abapGit repository abap-code-search-tools must be installed

Links:

- Source-Code on GitHub
- Eclipse Marketplace (Marketplace Client must be installed)

7.3.10 abapGit Eclipse Plug-in

Provides the functions of the abapGit SAP GUI transaction as an Eclipse plug-in. However, the full range of functions is currently (2022) only guaranteed via the SAP GUI transaction.

Prerequisites Eclipse:

- Eclipse IDE for Java Developers
- ADT

Prerequisites ABAP:

- SAP NetWeaver 7.50 or newer
- Complete abapGit Installation
- abapGit repository ADT_Backend must be installed

Links:

- Source-Code on GitHub
- Installation via Update-Site https://eclipse.abapgit.org/updatesite/

7.4 Develop own ADT Plug-ins

7.4.1 Prerequisites

Since plug-ins for Eclipse are to be developed in Java, it is advisable to already have some knowledge there. The necessary level of knowledge depends on the plug-in you want to develop.

7.4.2 Setting up the development environment

7.4.2.1 Installation Eclipse for RCP/RAP Development

To develop plug-ins for Eclipse, you need a specific variant of the Eclipse platform: Eclipse *IDE for RCP and RAP Developers* (RAP = Remote Application Platform). This variant provides a complete toolset to develop plug-ins for Eclipse as well as rich client applications (RCP) based on Eclipse. It can be obtained directly from eclipse.org.

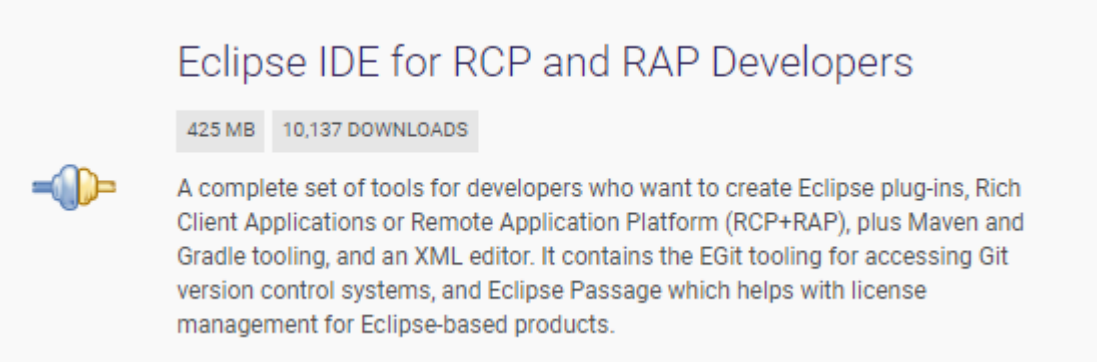


Figure 181 Eclipse Bundle "Eclipse IDE for RCP and RAP Developers"

Furthermore, an installation of the Java Development Kit (JDK) is required. This can be obtained, for example, from the following sources:

- https://adoptium.net/de/
- https://openjdk.org/
- https://sap.github.io/SapMachine/

Note: In the more recent Eclipse versions, this is already included.

Which Java Version?

Since Eclipse v2020-09 or ADT v3.16, Java 11 is the minimum requirement, and therefore the JDK should also be installed in at least this version.

7.4.2.2 Installation of ADT

The next step is to install ADT in Eclipse, otherwise it will not be possible to develop against the ADT SDK.

In order to have the best possible compatibility, it is recommended to always install ADT in the same version as Eclipse. The latest ADT version can be obtained from https://tools.hana.ondemand.com/latest. For older versions of ADT, you simply have to replace the *latest* in the path with the desired Eclipse version (for Eclipse 2020-09, for example, it would be: https://tools.hana.ondemand.com/2020-09).

7.4.2.3 Installation of useful Plug-ins (optional)

In addition to the Eclipse installation, it is recommended to install the following plug-ins:

Enhanced Class Decompiler

Marketplace Link: https://marketplace.eclipse.org/content/enhanced-class-decompiler.

This plug-in allows compiled source code to be displayed in a readable way. It is even possible to set breakpoints in such de-compiled classes and parse the code at runtime.

Web Developer Tools

Marketplace Link: https://marketplace.eclipse.org/content/eclipse-web-developer-tools-0/promo.

If you also want to offer help for your plug-in, this plug-in extends Eclipse with editors with syntax highlighting for the typical web file extensions (css, html etc.).

WindowBuilder

Marketplace Link: https://www.eclipse.org/windowbuilder/

Creating GUI elements, such as dialogs or custom views, can sometimes be very time-consuming. The WindowBuilder can help with this and allows you to create GUI elements with the help of a graphical editor.

7.4.3 Key Concepts/Artifacts

7.4.3.1 Plug-in

A plug-in is used to aggregate code into a modular, extensible, and shareable unit. The entire Eclipse application consists of many such plug-ins.

7.4.3.2 Feature

A feature is used to group one or more plug-ins into a single installable and updatable unit.

7.4.3.3 Update Site

Update sites are used to organize and export features so that they can be installed in Eclipse applications.

7.4.4 Creation of a Plug-in Project

A new plug-in project can be created via File → New → Plug-in Project. The Project Wizard plug-in will open:

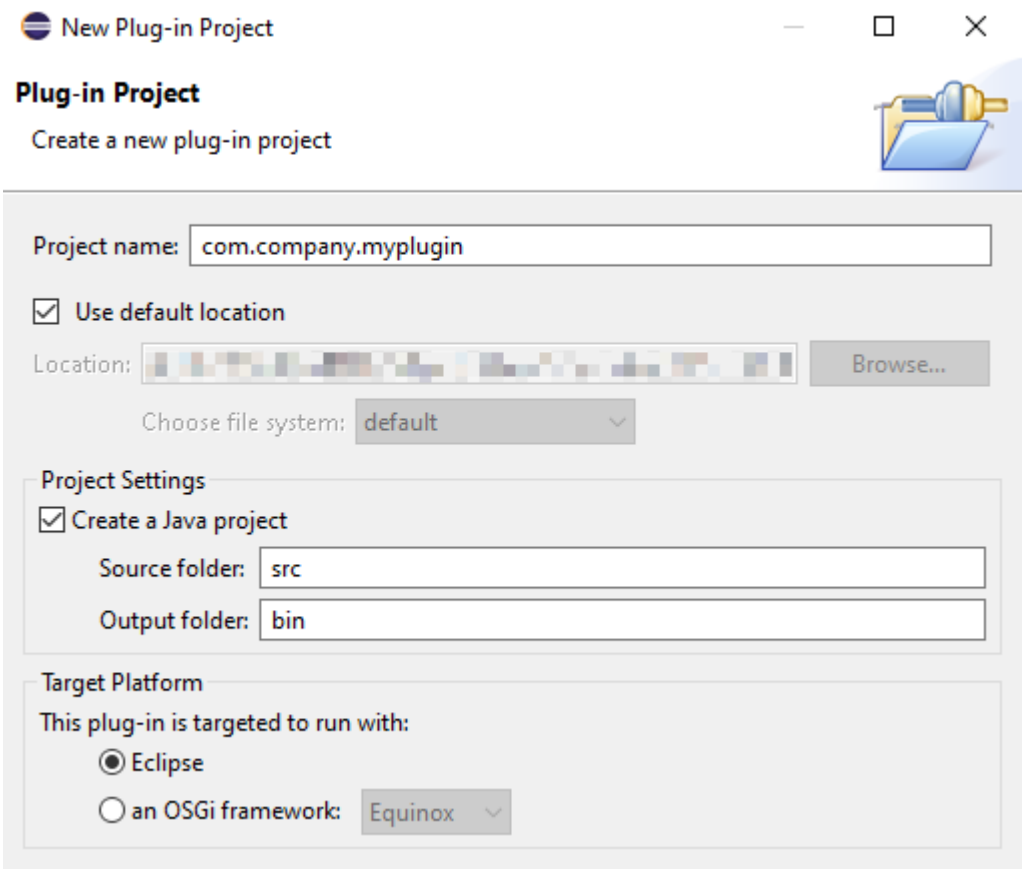


Figure 182 Plug-in Project Wizard – Entrance

Here, a name for the project must first be assigned. In the case of the name, the so-called. *Reverse naming domain notation* is recommended (e.g.com.company.myplugin), but any other naming convention can be chosen here.

By default, a plug-in project is always created as a Java project because most plug-ins contribute code. However, this option can also be deselected, e.g. for plug-ins that only provide documentation.

Since this guide explicitly discusses plug-in development for Eclipse, the *target platform* is always Eclipse.

Clicking on *Next* takes you to the next page of the wizard.

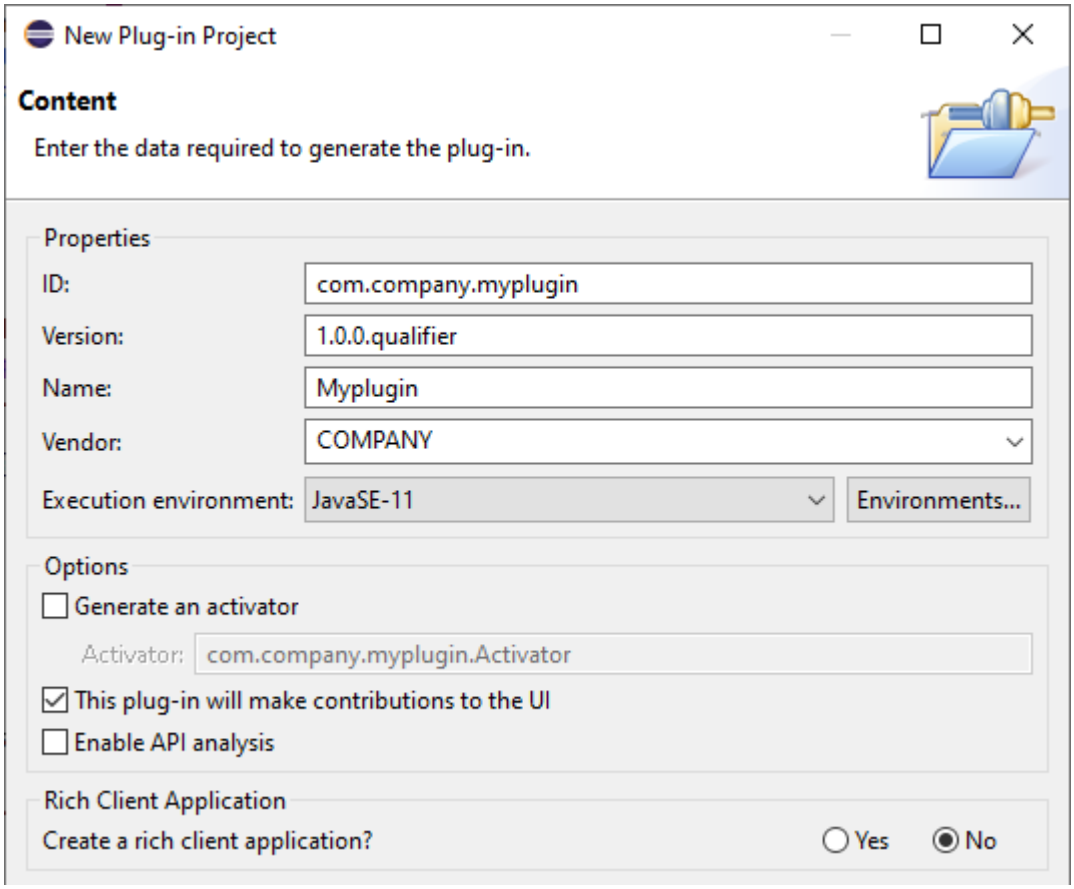


Figure 183 Plug-in Project Wizard – Content

This is where the plug-in-specific properties are recorded. It is recommended to use the project name for the ID, but this is not mandatory. The version must follow the pattern *major.minor.micro.qualifier.* The *.qualifier* part is optional. It is replaced with a timestamp during the build (e.g. 1.3.0.202205011550).

The *Name* and *Vendor* fields are translatable and represent the plug-in name and its vendor.

For the *Execution Environment*, the minimum required Java version must be entered. Java 11 is currently the minimum requirement for ADT plug-ins and should therefore also be set for your own plug-ins.

If the *Generate an activator option is set,* an *activator* class is generated. Such a class can exist exactly once per plug-in and is only necessary if activities are necessary when starting or stopping the plug-in. The option *This plug-in will make contributions to the UI* regulates the available templates, which can be selected on the next page of the wizard.

As a last optional step, a template can be selected on the next page. Templates exist, for example, for creating your own views or editors.

After completing the wizard with *Finish* , the plug-in project is generated at the selected location in the file system and then displayed in the Eclipse Workspace.

7.4.4.1 Structure of a Plug-In Project

A plug-in project always has the following structure. The plugin.*xml* file and the *OSGI-INF* folder are optional and only exist if necessary.

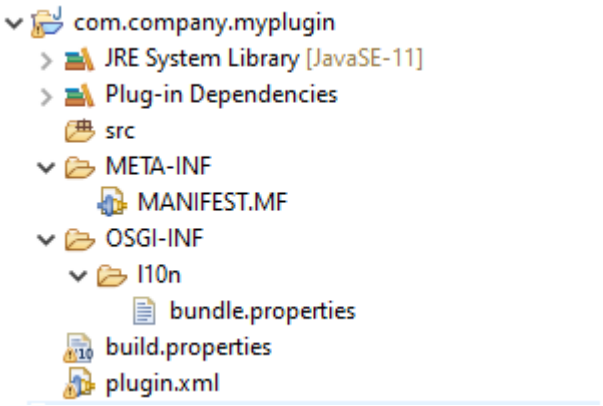


Figure 184 Plug-in-Projekt in the Project Explorer View

The most important files are *manifest.mf,* build.properties, and *plugin.xml*. If you open one of these three files, the plug-in manifest editor is opened by default. This editor enables the maintenance of all metadata of a plug-in, which the editor subdivides into the following areas:

- **Overview**
  This view serves as a general entry point. It allows the maintenance of basic plug-in data such as name, version, etc. and the jump to the other views.

- **Dependencies**
  All plug-ins that are required in this plug-in must be listed here.

- **Runtime**
  Used to specify the Java packages that should be visible to other plug-ins.
  An API status can also be set for each package.

- **Extensions**
  This is where the actual extension of Eclipse takes place, e.g. with new menus, commands, views, etc.

- **Extension Points**
  Definition of the extensibility points that this plug-in provides to others.

- **Build**
  Configuration of which files should be included in the build result.

7.4.5 Creation of a Feature Project

A new feature can be created via File → New → Feature Project. The Feature Project Wizard opens:

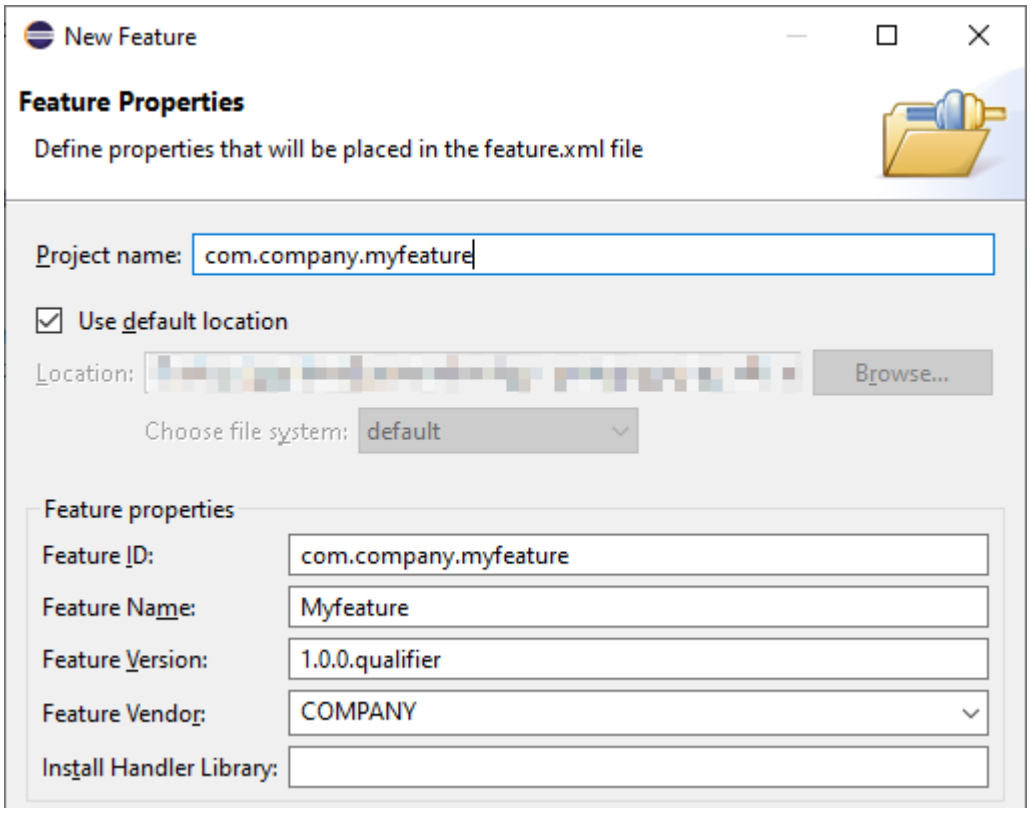


Figure 185 Feature Project Wizard - Entrance

The properties of a feature project are similar to those of a plug-in project, so the same rules apply to e.g. *ID, name,* or *version*.

Now the wizard can be closed or you can navigate to the next page, where you can select the plug-ins that should be included in this feature:

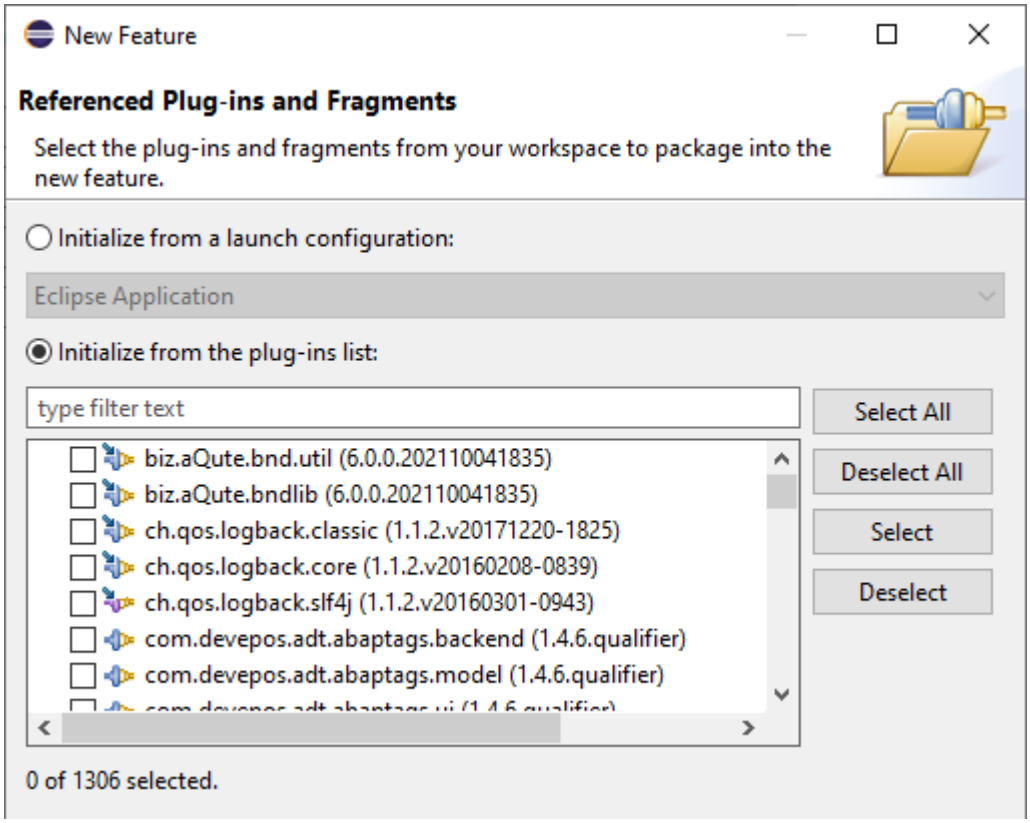


Figure 186 Feature Project Wizard - Plug-in Selection

Upon completion, the feature project is generated and displayed in the workbench.

7.4.5.1 Structure of a Feature Project

A feature project has a very simple structure. It contains only the two files *feature.xml* and *build.properties.* As with the plug-in project, there is a separate manifest editor for maintaining the feature metadata, which opens automatically when one of the two files is opened.

This is divided into the following sections:

- **Overview**
  This view serves as a general entry point. It allows the maintenance of basic plug-in data such as name, version, etc. and the jump to the other views.

- **Information**
  Description, copyright notice and license agreement can be maintained here.

- **Included Plug-ins**
  Selection of plug-ins that are included in this feature.

- **Included Features**
  Features can be used to group other features and thus the included features can be listed here.

- **Dependencies**
  Normally, all dependencies are computed during the build. This is done by analyzing the dependencies of all included plug-ins. However, it is also possible to maintain the dependencies manually here.

- **Build**
  see Plug-in Manifest

7.4.6 Creation of an Update Site

A new update site project can be created via File → New → Project… → Plug-in Development →> Update Site Project. The Update Site Project Wizard opens:

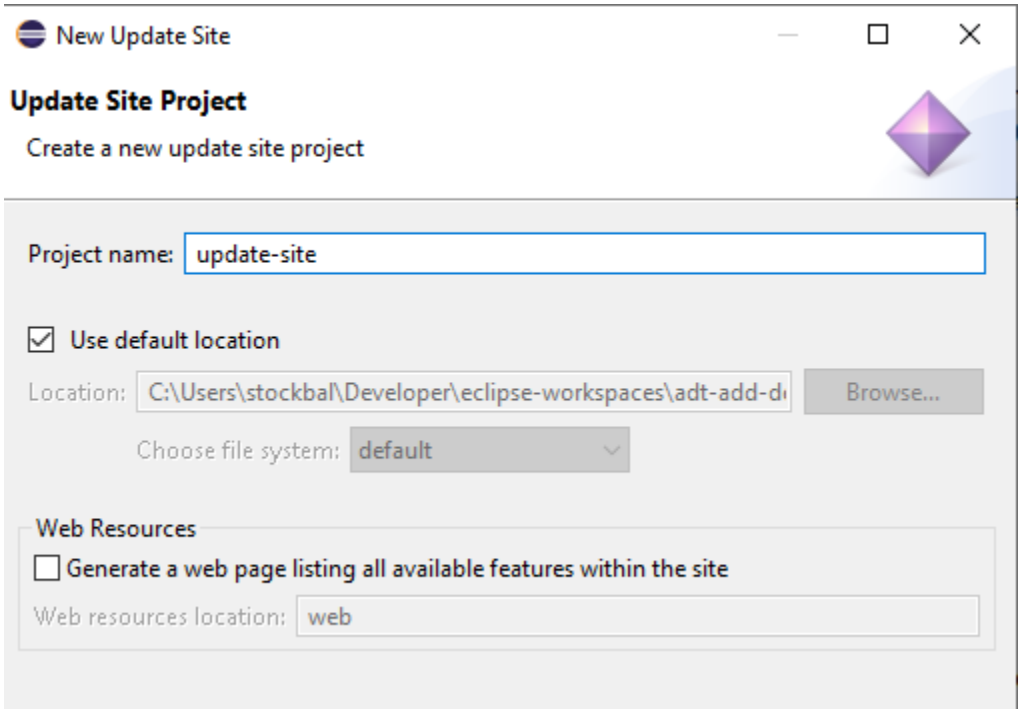


Figure 187 Update Site Wizard

The wizard contains only one page where you can give the project a name and select the storage location. After completing the wizard, there should be a folder in the workspace with the selected project name. This folder contains the update site's manifest file, called *site .xml*.

Double-clicking this file opens the manifest editor for the update site. In this editor, you can now add the features that you want to publish to the site. For a better overview, the features should be divided into categories.

After the content of the update site has been configured, it can be created using the *Build all* button in the editor. It is also possible to create only individual or selected features.

IMPORTANT: Before you create the update site, you should check the Java compiler settings again via Window → Preferences → Java → Compiler. These should be set to the same Java version that has been defined as the minimum requirement for the plug-ins:

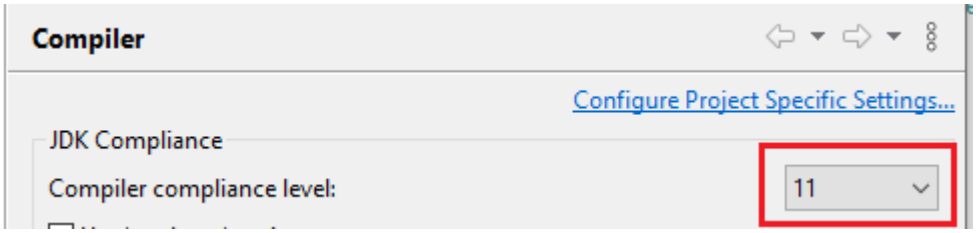


Figure 188 Compiler Settings in the Eclipse Settings Dialog

If the creation is successful, the following files/folders will be located in the project folder of the update site:

- features (contains jar files of the features)
- plugins (contains jar files of the plug-ins)
- artifacts.jar
- content.jar

In addition, an archive with the name *logs.zip* may have been created. This is where all the messages that occurred during compilation are located.

7.4.6.1 Testing the Update Site

Before the created site is uploaded to a web server, you may want to test it first. To do this, you should get a new Eclipse installation. The Eclipse *IDE for Java Developers* variant is completely sufficient for this. In this installation, ADT are now installed, after a restart you now add the new – so far still local – update site via Help → Install New Software… → Add…:

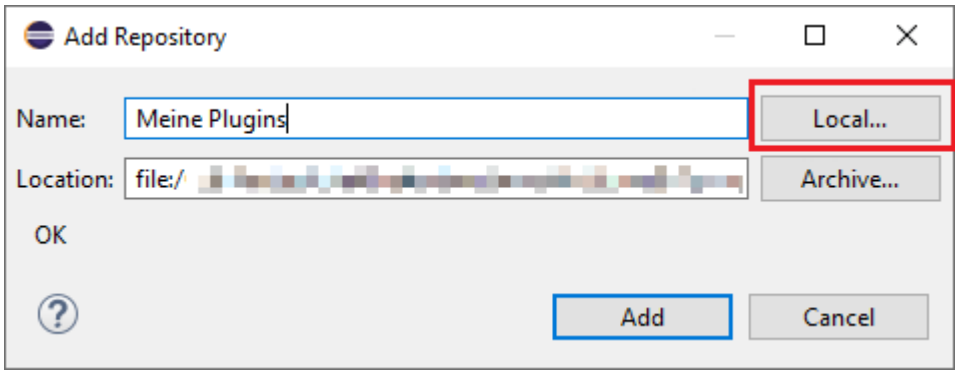


Figure 189 Dialog for Adding an Update Site

The "Local..." button is used to select the directory of the update site project. After clicking on the "Add" button, the categories and the assigned features of the update site should be listed and ready to be installed.

7.4.6.2 Deployment

If the test of the update site is successful, it can now be uploaded to a web server to make the artifacts available to others. If you shy away from the cost of your own web server, there is a free way to provide your update site via GitHub Pages, for example. To do this, initialize a new Git repository in the project directory of the update site and publish it in a public repository on GitHub. Afterwards, the option "GitHub Pages" can be activated in the repository settings on GitHub:

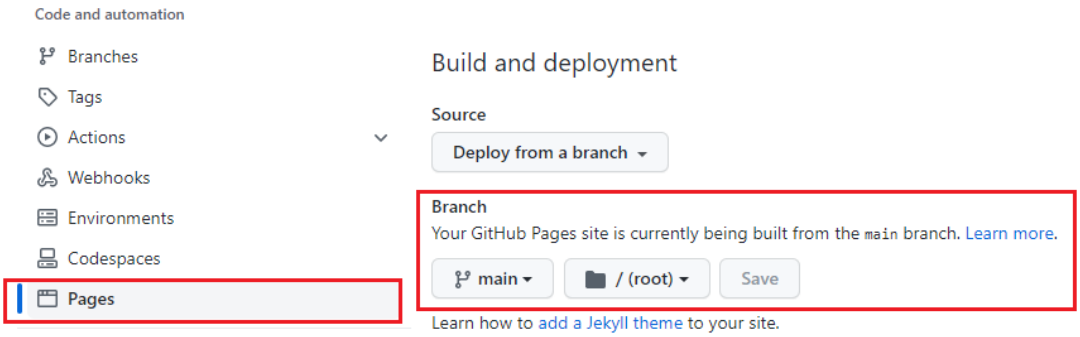


Figure 190 GitHub Repository Settings for GitHub Pages

After GitHub has finished creating the GitHub Pages page, it can be registered with the URL *https://<username>.github.io/<repository-name>* in Eclipse as an update site

7.4.7 Extension of the ADT Backends with ABAP Code

For communication from Eclipse to the ABAP Server, ADT uses RESTful APIs. How to develop such APIs yourself can be found in the SAP Guide How To... Create RESTful APIs and consume them in ADT .

Although we are talking about RESTful APIs here, there is no communication via HTTP from ADT. It is done using the RFC protocol. At the lowest level of the communication layer of ADT, an RFC call takes place that calls a specific RFC function module on the ABAP server.

Thus, it is open to everyone to either develop their own RESTful APIs with the BAdI extension concept, as described in the guide, or alternatively to develop an RFC-

capable function module and call it with the RFC Java Connector. The Java Connector API can be integrated and used via the "com.sap.conn.jco" plug-in.

Especially for smaller plug-ins, the BAdI method may seem like quite an overhead, both on the ABAP and Java side. However, the BAdI approach also has its advantages. Especially through the use of EMF (Eclipse Modelling Framework), it is possible to easily convert a string serialized from ABAP to XML into objects in Java. Simply described, you need a "Simple Transformation" (object type XSLT) on the ABAP page to transform ABAP data to XML.

```
<?sap.transform simple?>
<tt:transform xmlns:tt="http://www.sap.com/transformation-templates"
              xmlns:cst="http://www.devepos.com/adt/cst"
              xmlns:adtbase="http://www.devepos.com/adt/base"
              xmlns:adtcore="http://www.sap.com/adt/core"
              xmlns:cl="http://www.sap.com/abapxml/types/class-pool/ZIF_ADCOSET_TY_ADT_TYPES">

  <tt:root name="root" type="cl:ty_code_search_result"/>

  <tt:include name="sadt_object_reference" template="objectReferenceAttributes"/>
  <tt:include name="sadt_main_object" template="main_object"/>

  <tt:template>
    <tt:apply name="codeSearchResult">
      <tt:with-root name="code_search_result" ref="root"/>
    </tt:apply>
  </tt:template>


  <!-- Template for the result -->
  <tt:template name="codeSearchResult">
    <tt:context>
      <tt:root name="code_search_result"/>
    </tt:context>

    <cst:result tt:extensible="deep">
      <tt:attribute name="cst:numberOfResults" value-ref="code_search_result.number_of_results"/>
      <tt:attribute name="cst:numberOfSearchedObjects" value-ref="code_search_result.number_of_searched_objects"/>
      <tt:attribute name="cst:numberOfSearchedSources" value-ref="code_search_result.number_of_searched_sources"/>
      <tt:attribute name="cst:linesOfSearchedCode" value-ref="code_search_result.loc"/>
      <tt:attribute name="cst:queryTimeInMs" value-ref="code_search_result.query_time_in_ms"/>

      <tt:apply name="codeSearchObjects">
        <tt:with-root name="code_search_objects" ref="code_search_result.code_search_objects"/>
      </tt:apply>
```

Figure 191 Example of a Simple Transformation for the Transformation of ABAP <-> XML

And on the part of Java, an EMF model is necessary.

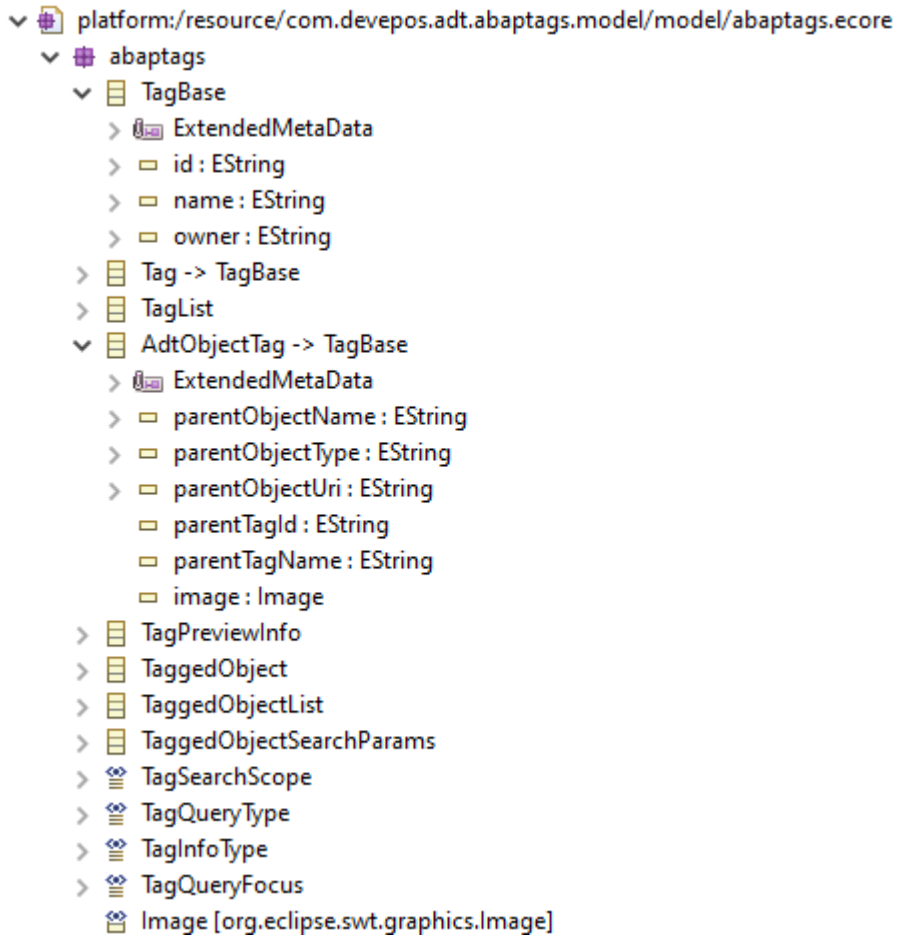


Figure 192 Sample EMF Model for Serializing XML <-> Java Object

This approach allows the data transformation to be built very generically, and you also save development time.

7.4.8 Java Code Snippets for recurring tasks in ADT

7.4.8.1 Calling an RFC function module with the Java Connector

```
// 1) Reading the Destination Id for an ABAP Project Instance

String destinationId =
AdtCoreProjectServiceFactory.createCoreProjectService().getDestinationId(project);

// 2) Reading the JCo destination to the Destination Id

JCoDestination destination = JCoDestinationManager.getDestination(destinationId);

// 3) Reading the RFC block
```

```
JCoFunction function =
destination.getRepository().getFunction("name_of_rfc_function");

// 4) Setting an Importing Parameter

function.getImportParameterList().getField("I_PARAM1").setValue(“PARAM_VALUE”);

// 5) Executing the function

function.execute(destination);

// 6) Reading an Exporting Table Parameter

JCoTable objectTree = function.getExportParameterList().getTable("E_PARAM1");
```

7.4.8.2 Reading the ABAP project depending on the current selection in the workbench

```
// 1) Determination of the active page in the workbench

IWorkbenchPage page =
PlatformUI.getWorkbench().getActiveWorkbenchWindow().getActivePage();

// 2) Reading the Workbench Window Instance

IWorkbenchWindow window = page.getWorkbenchWindow();

// 3) Determination of the current selection in the window

ISelection adtSelection = window.getSelectionService().getSelection();

// 4) Determination of the active ABAP project

IProject project = ProjectUtil.getActiveAdtCoreProject(adtSelection, null, null,
IAbapProject.ABAP_PROJECT_NATURE);
```

7.4.8.3 Determine the source code oft he active editor

```
// 1) Determine the active editor

IAdtFormEditor editor = (IAdtFormEditor)PlatformUI.getWorkbench()

                                .getActiveWorkbenchWindow().getActivePage()

                                .getActiveEditor();

// 2) Determine the editor's document

IDocument document = editor.getAdapter(AbapSourcePage.class).getDocument();
```

```
// 3) Determine the source code

String code = document.get();
```

7.4.8.4 Execute the transaction code

```
// 1) Determine the user setting of navigation to the Eclipse Editor for supported
development objects

boolean navigateToEclipse =

       com.sap.adt.sapgui.ui.internal.Activator.getDefault()

                     .getPreferenceStore()

                     .getBoolean(com.sap.adt.sapgui.ui.internal.PreferenceInitializ
              er

                                   .PREF_KEY_USE_ECLIPSE_NAVIGATION);

// 2) Execution of the transaction code

AdtSapGuiEditorUtilityFactory

       .createSapGuiEditorUtility()

       .openEditorAndStartTransaction(project,TRANSACTION_NAME, avigateToEclipse);
```

7.4.8.5 ABAP Scan Services – Check whether the token is a keyword

```
// 1) Determine the instance of AbapSourceUI
IAbapSourceUi sourceUi = AbapSourceUi.getInstance();

// 2) Determine the instance of SourceScannerServices

IAbapSourceScannerServices = sourceUi.getSourceScannerServices();

// 3) Determine the active ADT editor

editor = (IAdtFormEditor)PlatformUI.getWorkbench()

                     .getActiveWorkbenchWindow().getActivePage().getActiveEditor();

// 4) Locate the document from the editor

IDocument document = editor.getAdapter(AbapSourcePage.class).getDocument();


// 5)  Check if the token at the offset position is a keyword (or not)
```

```
Boolean isKeyword = scannerServices.isKeyword(document,OFFSET);

// 6)  Determine the next token based on the offset position

Token nextToken = scannerServices.getNextToken(document,OFFSET);

// 7)  Check if the next token is a keyword (or not)

isKeyword = scannerServices.isKeyword(document,nextToken.offset);
```

7.4.8.6 Identify the project and show the selection dialog

```
// 1) Determine the shell

Shell shell = PlatformUI.getWorkbench().getActiveWorkbenchWindow().getShell();

// 2) Show the selection dialog and determine the selected project

IProject chosenProject = AbapProjectSelectionDialog.open(shell, null);
```

7.4.8.7 Identify the users and call up the selection dialog

```
// 1) Determine User Service

IAdtUserServiceUI adtUserService =
AdtUserServiceUIFactory.createAdtUserServiceUI();

// 2) Call up the user selection dialog and determine the selected users

String[] users = adtUserService.openUserNameSelectionDialog(null, false,
project,"");
```

7.4.8.8 Logging on tot he ABAP System

```
// 1) Adapting an IProject Object to an IAbapProject Object

final IAbapProject abapProject = project.getAdapter(IAbapProject.class);

// 2) Check the login status with automatic login dialog if no one is already
available

//    Registration is available

IStatus logonStatus = AdtLogonServiceUIFactory.createLogonServiceUI()

       .ensureLoggedOn(abapProject.getDestinationData(), PlatformUI.getWorkbench()
```

```
            .getProgressService())
        .isOK();
```

7.4.9 Project Set-up with Maven

In addition to the standard options for developing plug-ins, features and update sites that come with *the Eclipse IDE for RCP and RAP Developers*, there is also the option to use Eclipse Tycho. Eclipse Tycho is a collection of plug-ins for Apache Maven. Further information can be found on Tycho's project page. A tutorial on plug-in development with Tycho can be found here.

Authors

Finally, we would like to introduce ourselves as a team of authors and continuous contact persons regarding any questions in the context of the ABAP development tools.

But first of all, we would like to take this opportunity to express our special thanks to **Marc Zimmek** (Claas KGaA), **Jens Zähringer** (Nagarro) and the DSAG office, who supported us in the final phases of the creation with their active proofreading.

In addition, we do not want to miss the opportunity to highlight the close, constructive and uncomplicated cooperation with colleagues at SAP. Many thanks to **Thomas Fiedler** (Product Owner ADT), **Wolfgang Wöhrle** (ADT documentation incl. configuration and installation), **Yannic Soethoff** and **Sebastian Ratz** (both ADT under Mac OS).

Name	Company and job title	Brief description of your activities and experiences with ADT
Michael Biber	R+V Allgemeine Versicherungen AG System Developer and Application Manager	First beginnings in 2016 in self-study. Gradually, increased use. Exclusive use today. Since approx. 2018 support of oomph configurations for the SAP area.
Jörg Brandeis	Brandeis Consulting GmbH Managing Director, Trainer, Developer and Consultant	Jörg Brandeis is Managing Director of Brandeis Consulting GmbH in Mannheim, which offers training and consulting in the areas of SAP BW/4HANA, SAP HANA, Modern ABAP and SQLScript. He is the author of SQLScript for SAP HANA (SAP Press)

Uwe Fetzer	Silesia Gerhard Hanke GmbH & Co. KG Team Lead Global SAP Development & Analytics	ADT has been used privately since the beginning (co-author of the SAPLink extension for Eclipse), and has been using ADT exclusively with employers for five years
Thomas Foehn	Zürcher Kantonalbank	Responsible for ABAP development guidelines and best practices. No widespread use of ADT yet. Use depending on employee preference, even now an intensive user
Sebastian Freilinger-Huber	msg systems ag Principal IT Consultant DSAG Speaker Committee Development	programming, designing, architecture; Use of Eclipse (ADT) since its introduction; Pushing the use of ADT within the company and divisions Organizational management of the DSAG-ADT guide
Florian Henninger	FIS Informationssysteme und Consulting GmbH Senior Consultant Development	programming, designing, architecture; Use of Eclipse (ADT) since the beginning; Main area Odata/fiori elements
Armin Junge	msg systems ag Lead IT Consultant	Architecture, Design, Implementation Intensive use of ADT for many years. Pushing the use in the area and at the customer's site.

Michael Keller	educatedBytes GmbH CEO, Consultant, Developer, Trainer	Michael Keller has been working for many years as a consultant, developer and trainer in the logistics environment of SAP ERP systems. Through development with the ABAP programming language, he became acquainted with the ABAP Development Tools for Eclipse in 2017 and has been working with this development environment ever since.
Peter Luz	Robert Bosch GmbH Development Engineer and -architect	ABAP developer and coordinator for centrally developed components and solutions in the area of SAP Logistics Execution. Fan of ABAP-OO and ABAP Unit Tests. Entry into ADT 2015. With ADT, the creation of high-quality and modern code has become much more efficient and faster.
Dominik Panzer	Intense AG Senior Manager/ Head of Development/ Agile Technical Coach	Long-time ADT users and often buried deep in legacy codebases to transform them and bring them into the new cloud world. Therefore, he really appreciates the refactoring features of ADT. In addition to technical topics, clean code and software architecture, team-oriented topics such as agile games, coding dojos, pair and mob programming are also close to his heart and passes them on as a coach.
Lukasz Pegiel	Hager Group Digital & Information Manager Poland	I'm responsible among others for: - local teams management, which includes ABAP Development Team, Net Development Team, IT support and BIM (Building Information Modeling Team) - Collaboration with External Resources Providers - S/4HANA custom code adaptation.

Dr. Wolfgang Röckelein	Axians NEO Solutions & Technology Senior Architect Product Development	ADT are used by me and my colleagues in product development and in customer projects for our products across the board.
Björn Schulz	REWE digital GmbH Product Owner (SAP Development & Technology)	Product owner and developer for ABAP, Steampunk and BTP. Only ADT for almost four years and always striving to motivate colleagues in the direction of ADT and to "sell" the many advantages.
Ludwig Stockbauer-Muhr	msg systems ag Senior IT Consultant	Architecture, Design, Implementation ADT has been in use for many years Passionate add-on developer for ADT
Bärbel Winkler	Alfred Kärcher SE & Co. KG Consultant SAP Development	ABAP Development & Guidelines. Not yet a widespread use of ADT and even only "rudimentary" experience, but I am using Eclipse more and more often and would like to motivate others to do the same.

Table 6 Authors

Imprint

We expressly point out that this document cannot anticipate and cover every regulatory requirement of all DSAG members in all business scenarios. In this respect, the topics and suggestions addressed must naturally remain incomplete. The DSAG and the authors involved cannot assume any responsibility for the completeness and suitability of the suggestions for success.

This publication is protected by copyright.

Unless expressly stated otherwise, all rights are vested in:

Deutschsprachige SAP® Anwendergruppe e.V.
Altrottstraße 34 a
69190 Walldorf | Deutschland
Telefon +49 6227 35809-58
Telefax +49 6227 35809-59
E-Mail info@dsag.de
dsag.de

Any unauthorized use is not permitted. This applies in particular to duplication, editing, distribution, translation or use in electronic systems/digital media.

© Copyright 2023 DSAG e.V.