Designer Manual

ADITO Software GmbH

Version 1.8 | 2025-02-19

Version	Changes
1.8	• Chapter Debugger extended and optimized for debugging of managed ADITO cloud systems. • Added appendix Version history. • Minor further improvements

Summaries of changes in previous versions of this document can be found in appendix Version history.

Character Formatting

The following signs will point you to specific sections:

 Hints and notes.

 Tips and tricks.

 This is important!

 Warning! These actions are dangerous and can result in data loss!

The following font formatting applies:

Font type	Meaning
Mask	The mask, table or button to which the section refers
"Mask"	Terms that originate from the system and that need to be emphasized in the reading flow
`code();`	Code and program parts

Preface

The ADITO Designer is the central tool to convert customers' requirements into impressing web-based ADITO projects. The Designer Manual is a glossary, in which the core functions of the ADITO Designer are explained, part by part - including the various plugins, which extend the Designer’s operational capabilities.

Happy reading!

The ADITO Academy

P.S.:
After reading this manual, you should go on by reading the ADITO Customizing Manual, which shows you, on the basis of a plain example, how to use the Designer in order to build and customize ADITO applications for multiple purposes.

Index

Character Formatting 1
Preface 2
1. Introduction 8
1.1. Relation to NetBeans 8
1.2. General Overview 8
1.3. Plugin concept 9
2. Installation 9
2.1. Connecting Designer and server 10
2.1.1. All operating systems 10
2.1.2. Additional steps for Linux and Mac 10
2.2. Start menu 10
3. Startup 11
3.1. Execution 11
3.2. Initialisation 11
3.3. Configuration 11
3.4. Ports 13
4. Menu bar 14
4.1. File 15
4.1.1. New - New Project 16
4.1.2. Import/Export Project 21
4.2. Edit 22
4.3. View 23
4.4. Navigate 26
4.5. Source 26
4.6. Refactor 26
4.7. Team 27
4.8. Tools 28
4.8.1. Overview 28
4.8.2. Export Project Structure 29
4.8.3. Options 30
4.8.3.1. ADITO 30
4.8.3.2. iReport 31
4.8.3.3. Other options 31
4.8.3.3.1. Janitor 31
4.9. Window 32
4.10. Help 34

5. Toolbar 35
6. Project Structure 36
6.1. System 36
6.1.1. Create a new System 37
6.1.2. Overview 38
6.1.3. Users 40
6.1.4. System Configuration 40
6.2. Preferences Project 41
6.3. Classic 43
6.4. Neon 43
6.4.1. Deletion of a context 44
6.5. Process 46
6.6. test 47
6.7. report 47
6.8. Internationalization 48
6.8.1. Basics 48
6.8.2. Special functions 49
6.8.3. Automated translation 50
6.8.4. Find unused keys 53
6.8.5. Export Keys with Place of Use 55
6.8.6. User help 56
6.8.7. Translation of source code 56
6.9. Roles 57
6.9.1. Internal roles 57
6.9.2. Project roles 58
6.10. alias 58
6.11. Others 62
7. Output Window 64
8. Execute SQL 65
9. Debugger 68
9.1. Preparations 68
9.2. Debugging options 72
9.3. Evaluate Expressions 75
9.4. Breakpoint Menu 76
9.5. Watches 77
10. Code quality support 78
10.1. Autocompletion 78

10.2. JSDoc 78
10.3. Errors and warnings 80
10.4. ESLint support 81
10.4.1. Prerequisites 81
10.4.2. Configuration 82
10.4.3. Executing ESLint 83
10.4.3.1. Analyze 83
10.4.3.2. Fix all 84
10.4.4. Examples of hints and solutions 85
10.5. Scan Services 87
10.5.1. Preferences 87
10.5.2. Result 88
11. Deploy 90
11.1. Deploy of selected data models 93
11.2. Deploy of opened data models 94
11.3. Deploy Tool 95
11.3.1. Prerequisites 95
11.3.2. Configuration and execution 95
11.3.3. Exit codes 96
12. Local Data 98
13. Updates 100
13.1. Update the ADITO project 100
13.1.1. Basics 100
13.1.2. Update of single ADITO models 101
13.2. Update the database 101
13.2.1. Basics 102
13.2.2. Maintaining system tables 102
13.2.2.1. ASYS_VERSIONHISTORY 103
13.2.2.2. List of system tables 103
13.2.2.3. Faulty system tables 104
13.2.2.4. Missing entry in ASYS_VERSIONHISTORY 104
13.2.2.5. Wrong structure 104
13.2.2.6. Table upgrade 105
13.2.2.7. Table missing 105
13.2.3. Action toolbar 105
13.2.4. Information area 106
13.2.5. Server startup prerequisites 107

14. Plugins 108
14.1. Installation 108
14.2. Advantages 110
14.3. Plugin AsciidoctorJ 111
14.4. Plugin Git 114
14.4.1. Auto-Merging 118
14.5. Plugin Liquibase 120
14.5.1. Basics 120
14.5.2. How to use Liquibase 120
14.5.2.1. Creating and naming changelog files 121
14.5.2.2. Creating folders 121
14.5.2.3. Liquibase folder structure 122
14.5.2.3.1. Common structure pattern 122
14.5.2.3.2. Liquibase folder structure of the ADITO xRM project 122
14.5.3. Changelog XML structure 123
14.5.4. Feature changelog XML structure 124
14.5.4.1. Data Definition Language (DDL) 124
14.5.4.1.1. CREATE 124
14.5.4.1.2. ALTER 127
14.5.4.1.3. DROP 128
14.5.4.1.4. PRECONDITIONS 130
14.5.4.1.5. RENAME 131
14.5.4.2. Data Manipulation Language (DML) 131
14.5.4.2.1. INSERT 132
14.5.4.2.2. UPDATE 132
14.5.4.2.3. DELETE 133
14.5.4.3. PREPARED STATEMENTS 133
14.5.5. ROLLBACKS 134
14.5.6. BLOB/CLOB 136
14.5.7. XML special characters 136
14.5.8. Enabling demo data 137
14.5.9. Create Liquibase files automatically 138
14.5.9.1. Create empty changelog 138
14.5.9.2. Create changelog with new DB table 138
14.5.9.3. Changelog to SQL 138
14.5.9.4. DB to changelog 139
14.5.10. Liquibase functions outside the Designer 141

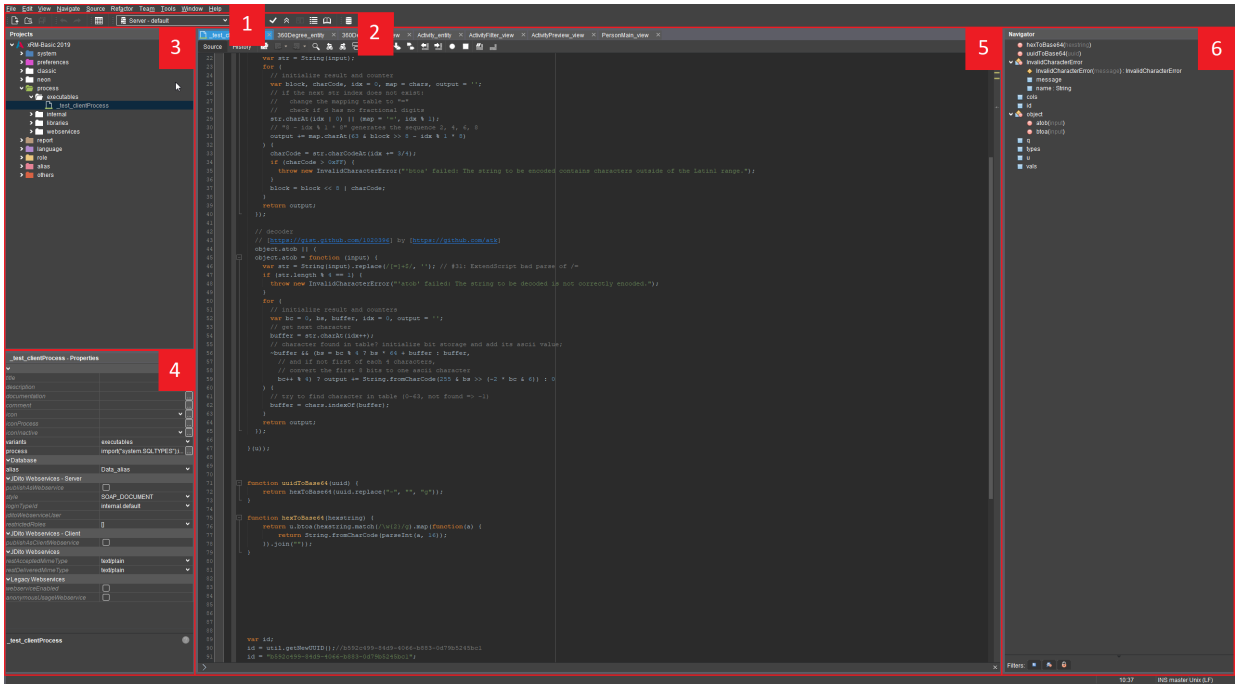
14.5.11. Liquibase with audit and offline synchronisation 141
14.5.12. Using Liquibase files for multiple database types 142
14.5.13. Liquibase troubleshooting 142
14.5.13.1. Updated database structure is not accessible 142
14.5.13.2. Handling Liquibase failures 143
14.6. Plugin Cloud Support 145
14.6.1. With SSP access 145
14.6.1.1. Connection via "load cloud system" 145
14.6.1.2. Connecting a local project to a cloud system 148
14.6.2. Without SSP access 150
14.7. Plugin NodeJS & TypeScript 151
14.8. Plugin ESLint 152
14.9. Plugin SQL Formatter 152
14.9.1. Functionality 152
14.9.2. Settings 152
14.10. Plugin Grouped Tabs 154
Appendix A: Configuration helpers 158
A.1. "+ New ..."-Button 158
A.2. Blueprints 158
A.3. Combobox content search 158
Appendix B: Create and upgrade system tables 160
B.1. Benefits 160
B.2. Execution 160
B.2.1. Parameter 160
B.2.2. Examples 161
B.2.2.1. Creating system tables 161
B.2.2.2. Upgrading system tables 161
B.2.2.3. Using short names 161
B.2.3. Exit codes 162
B.2.3.1. General exit codes 162
B.2.3.2. Exit codes of createTables 163
B.2.3.3. Exit codes of upgradeTables 163
Appendix C: Setting the path variables 164
Appendix D: UUID Generator 165
Appendix E: Tunneling 165
Appendix F: Version history 167

1. Introduction

1.1. Relation to NetBeans

The ADITO Designer is based upon the NetBeans IDE. Therefore, this manual focuses on the functions that are special features of the ADITO Designer. For a more general overview, please refer to the official NetBeans documentation.

1.2. General Overview



The ADITO Designer can generally be divided into six different sections. In the screenshot you can see the standard setting.

1. The menu bar on top.
   These subcategories are always the same and cannot be changed
2. The toolbar directly below.
3. The navigation window for the project ("Projects window", "Project tree").
4. The properties of the marked project part ("Properties window", "Property sheet").
5. The main screen (also called "Editor window"), where all code is displayed.
   This is the only window that cannot be removed completely. You can scale the font size of the code lines up and down by pressing the mouse wheel and then turning it back and forth.
6. The navigation window ("Navigator"), for the file viewed.

The content of all of these regions can be individually changed with different windows via drag and drop.

The properties shown in the "Properties" window are interdependent - meaning that some of them are only shown, if a specific property has been set to a certain value.
Examples:

- If the calender has been deactivated, then the other calender-specific properties are not shown.
- If a View's property "layout" is set to "HeaderFooterLayout", then 2 additional properties will appear, namely "header" and "footer".

For special information regarding the menu, toolbar, or the different windows, please refer to the following chapters.

If you have changed the size and position of the window and created a mess, then there is an easy way to get back to the standard windows setting: Choose "Window" in the menu bar and then select option "Reset Windows". The Designer will disappear for a second and reappear on you standard display, with all windows in standard setting - exception window "Output", which must be added manually again:
Window > Output.

1.3. Plugin concept

Part of the functionality of the ADITO Designer is offered via plugins, which can be installed/uninstalled (or activated/deactivated, respectivly) on demand. When starting the Designer for the first time, a dialog will appear that offers you to install all "standard" plugins, which usually should be installed in order to work with the Designer smoothly.

Please find further information in chapter "Plugins".

2. Installation

Up to and including ADITO versions 2023.1.x, both the Designer and the server are included in one installation file, named, e.g., ADITO_2023.1.0_windows-x64.exe.
Additionally, beginning with version 2023.1.0, the Designer is also supplied as ZIP file.
This should enable the user to get familiar with the new installation approach that will be standard from version 2023.2.0 and newer:

- The ADITO Designer is available as ZIP file.
- The ADITO server is available in an installation file, which does no longer include the Designer.

2.1. Connecting Designer and server

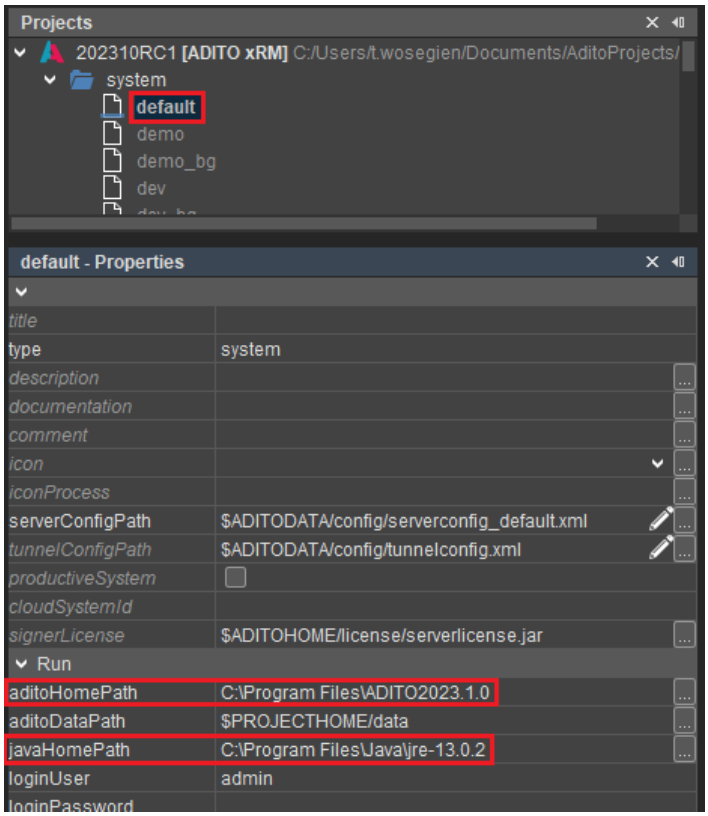
The following is only required, if you are working with a local server, not with a cloud system.

2.1.1. All operating systems

The handling of Designer and a local server remains the same. The connection between the unzipped Designer and the installed server is done via property "aditoHomePath". This property has a file chooser, with which you can select the path of your server, e.g., C:\Program Files\ADITO2023.1.0.

2.1.2. Additional steps for Linux and Mac

Users of Linux and Mac (after unzipping the Designer) additionaly need to specify the path to a Java JDK or JRE that is compatible with the Designer, in property "javaHomePath" e.g., C:\Program Files\Java\jre-13.0.2



Furthermore, they need to set the "jdkhome" attribute in the file ADITODesigner.conf to a valid JDK/JRE path, because the zip file by default includes only a Windows JRE. The Designer requires Java 13.

2.2. Start menu

As unzipping does not mean that something is included in the Windows start menu, we have to do this manually. In the Designer's directory, we navigate to folder "bin", right-click on file ADITOdesigner.exe, and choose option "Create shortcut". Now you can drag the shortcut on the desktop, into the task bar, include it in the start menu or wherever you want it to be available for starting the Designer.

3. Startup

3.1. Execution

Start the designer via the shortcut which you have created manually during the installation process - see previous chapter.

3.2. Initialisation

At the first Designer startup,

- an initial dialog appears, prompting you to specifiy which of the Designer plugins are to be installed. Usually, you can select all plugins.
- the initial content of the "default_userdir" (see below) is created.

3.3. Configuration

In the ADITO platform's installation directory "config", you can find a file named "ADITOdesigner.conf". This file includes startup configuration parameters of the ADITO Designer. Usually, you do not need to change the default configuration. However, if required, here is a description of the parameters:

- **default_userdir** (for macOS: **default_mac_userdir**): Basic subdirectory of the user directory. Here, the Designer automatically creates and manages its files, e.g., plugins and configuration files.

  From ADITO version 2022.1.0, folder ".aditodesigner" has one separate sub-folder (sub-user-directory) for every installed ADITO version, be it a major release (e.g., 2022.0), a minor release (e.g., 2022.0.1), or a hotfix (e.g., 2022.0.0.2). Earlier versions had sub-folders (sub-user-directories) only for every major release.

  Example: `"${HOME}/.aditodesigner/2021.2"` (which results in a folder like `"C:\Users\j.smith\AppData\Roaming\.aditodesigner\2021.2`). If you work with multiple Designer installations belonging to the same major version (e.g., 2021.2), you need to modify this directory's name (before the first Designer startup), in order to avoid that its content gets confused by 2 Designer instances writing into this directory. For example, specify `"${HOME}/.aditodesigner/2021.2_customerA"` in the ADITOdesigner.conf of the

first installation, and `"${HOME}/.aditodesigner/2021.2_customerB"` in the ADITOdesigner.conf of the second installation.

- **netbeans_default_cachedir**: Directory in which the Designer places cache data, e.g., indices, or the state of the Designer windows at last shutdown. Example: `"${HOME}/.aditodesigner/2021.1/cache"` (which results in a folder like `"C:\Users\j.smith\AppData\Roaming\.aditodesigner\2021.1\cache"`). If you work with multiple Designer installations belonging to the same version (e.g., 2021.2), you first need to modify this directory's name (see remark above).
- **default_options**: Java parameters passed to the Designer. Here you can, e.g., specify "-D" parameters, which always need the prefix "-J".
  Example: `-J-Dadito.home=\"C:\Program Files\ADITO2021.2.1\"`. Here are some further parameters that might be of special interest:
  - If you work with multiple projects in the same Designer instance: `-J-Xmx` defines the main memory to be reserved for the Designer. The default value is `-J-Xmx6144m` (same as `-J-Xmx6G`), meaning 2 gigabyte, which should be enough if you work with 1 or 2 projects at the same time. If you work with more projects or with very large projects, you may increase this value according to your requirements. But CAUTION: The Designer will always allocate the amount of RAM assigned to it via the `-J-Xmx` parameter, even if actually less RAM is required. Therefore, handle this parameter with care.
  - A parameter that is added occasionally is the "default database timeout". It enables you to define how long the system should "wait" for the completion of a logic execution. If this timespan is exceeded, a dialog will appear, prompting you to decide whether to continue waiting or stopping the execution. By default (= if not explicitely set in file "ADITOdesigner.conf"), this value is 30000 (i.e., 30 seconds).
    In some cases, e.g., when executing a "Drop all & update" command via Liquibase, this default value might be too low and you possibly want to avoid having to confirm the timeout dialog again and again. You can solve this by changing the timeout value: Add and set the following parameter:
    `-J-Dadito.designerdb.timeout=<value in milliseconds>`.
  - `-Dadito.designer.project` allows you to specify an absolute path of a project (or any file of it), which will then be opened at every Designer startup. If further projects had already been opened additionally, they will not be closed.
  - `-J-Djdk.http.auth.tunneling.disabledSchemes=\"negotiate,kerberos,digest,ntlm\"` deactivates all non-supported authentication methods when using a proxy. (The ADITO Designer exclusively supports the authentication method "BASIC".)

- **jdkhome**: Path to your local Java Runtime Environment (JRE); can be a relative or an absolute path. Default is the relative path "jre", as every ADITO installation comes with a current JRE, stored in a folder named "jre".

3.4. Ports

The Designer itself does not need ports to be opened. It depends on where the Designer wants to connect to. If everything runs locally (the system, the database, the project) you do not need to open any ports. If you, e.g., want to load your project from Git, then the Designer needs to connect to the Git repository, and thus it needs the Git (SSH/HTTPS) port. Similarly, in order to load data from the SSP, the Designer must connect to the SSP and thus needs the open port for HTTPS.

The following ports need to be opened, if the Designer needs to connect to non-local destinations (all ports TCP unless otherwise noted):

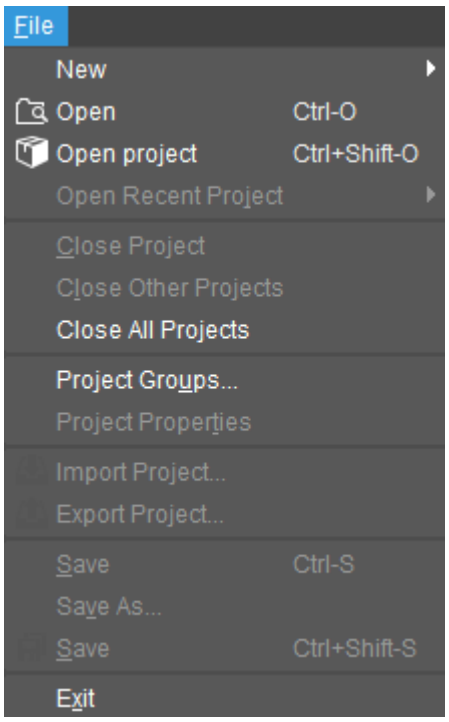
Destination	Port	URL	Comment
Plugins	443	https://aditopluginsonline.adito.de/2023.1.0/catalog.xml	Caution: Version-specific. It is recommended to share the following URL https://aditopluginsonline.adito.de/ → half-optional: nbm files can be downloaded elsewhere and then copied over → check correct version!
NodeJS	443	https://nodejs.org/dist/	→ half-optional: can be copied from existing installation (%appdata%/…/libraries/bundled_nodejs) → check correct version!
Analytics (optional)	9000	http://157.90.233.96:9000	might change in the future
GitLab	443 for https AND 22 for SSH	https://gitlab.adito.de/	

Destination	Port	URL	Comment
Cloud Plugin	443	https://ssp.adito.cloud/	
SSH tunnel	depending on system; check tunnelconfig.xml	depending on tunnelconfig, e.g., ssh.c2.adito.cloud	
database	MariaDB: 3306 MS SQL: 1433 Oracle: 1521 Postgres: 5432 Derby: 1527 DB2: 443		depending on manufacturer, check in serverconfig (→ host/port); if managed via SSH tunnel, then irrelevant
npm install	443	https://nexus.adito.cloud/repository/xrm	this is the standard; can be changed, if required, in .npmrc
Clear Datamodel Cache at deploy (optional)	neon HTTPS port standard is 8443	depending on ADITO server address	not to be mistaked with db cache, which is controlled via manager; if managed via SSH tunnel, then irrelevant
autom. translation of language keys (optional)	443	https://api.deepl.com/v2/translate https://translation.googleapis.com/language/translate/v2 https://translate.yandex.net/api/v1.5/tr.json/translate	via DeepL, Google Translate, or Yandex
Debugger	1098 and 1099	depending on ADITO server address	if managed via SSH tunnel, then irrelevant
Telnet logger	7722		
DNS	UDP/53		

4. Menu bar

Only menu items that differ from the NetBeans standard will be covered in this chapter.
For information regarding other options please refer to the NetBeans documentation.

4.1. File

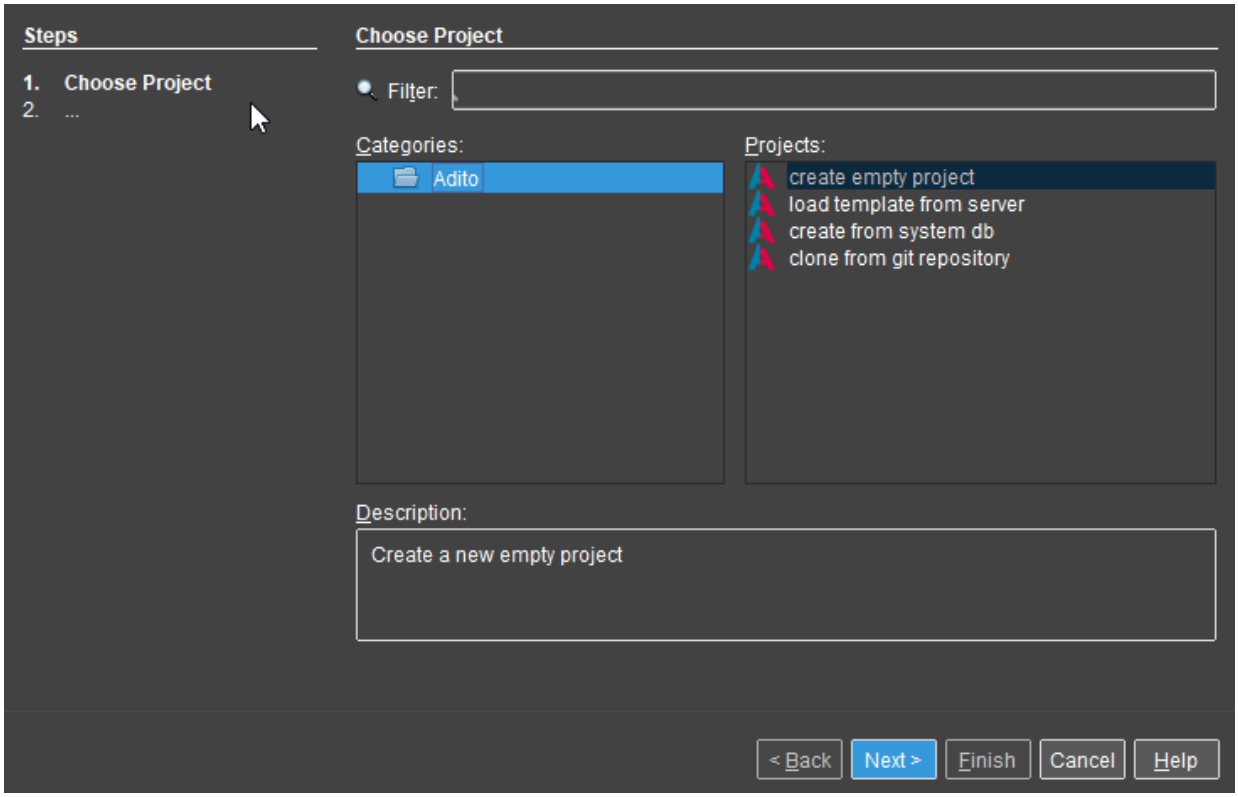


Name	Usage
New - New	Create a new data model (entity/process/role/etc.) within the selected project.
New - New Project	Create a new project. Additional Information below.
Open	Open/Search for one data model in a project.
Open Project	Open a project saved locally.
Open Recent Project	Open recently open projects.

Name	Usage
Close Project	Close the selected project.
Close Other projects	Close all projects but hte selected one.
Close All Projects	Close all open projects.
Project Groups	Group your projects together.
Import Project	Import data models or complete projects from a export file into your selected project. Additional Information below.
Export Project	Export one or more data models from your selected project. Additional Information below.
Save	Save the project.
Save As	Specify how you want to save the project
Save All	Save all projects.

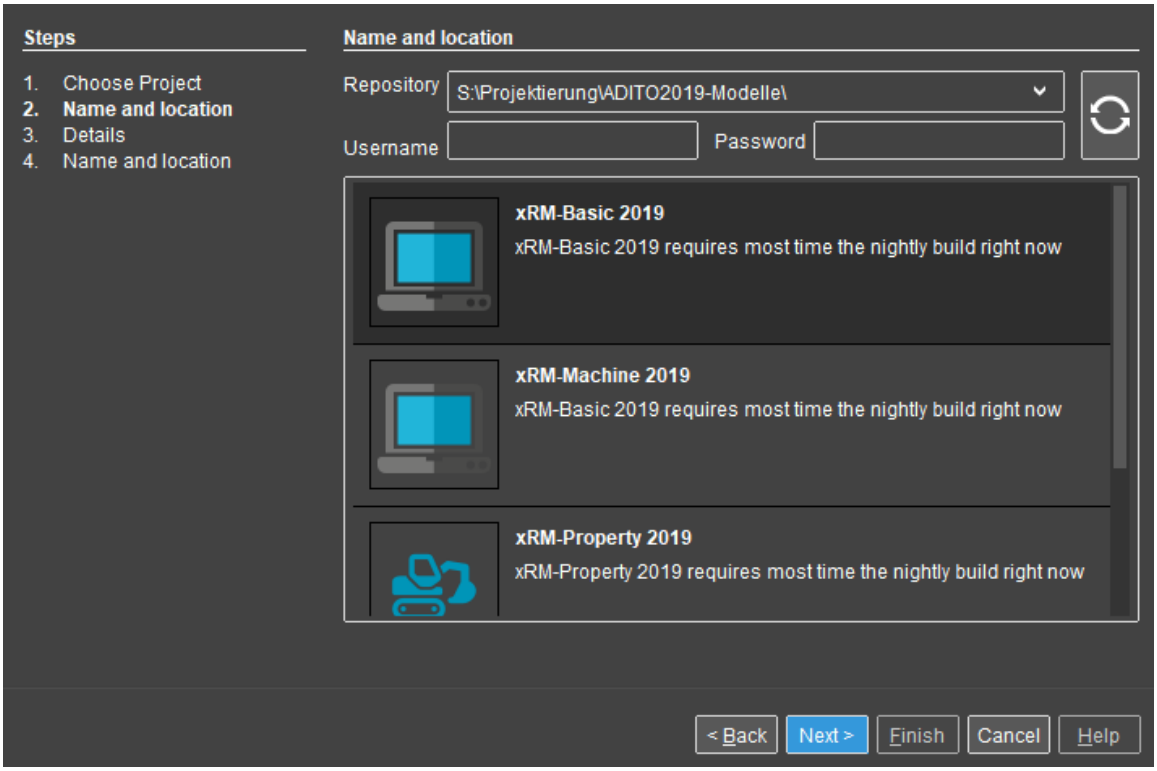
4.1.1. New - New Project

There are a few ways to create a new project.



To be able to use option number 3 the git plugin has to be installed.

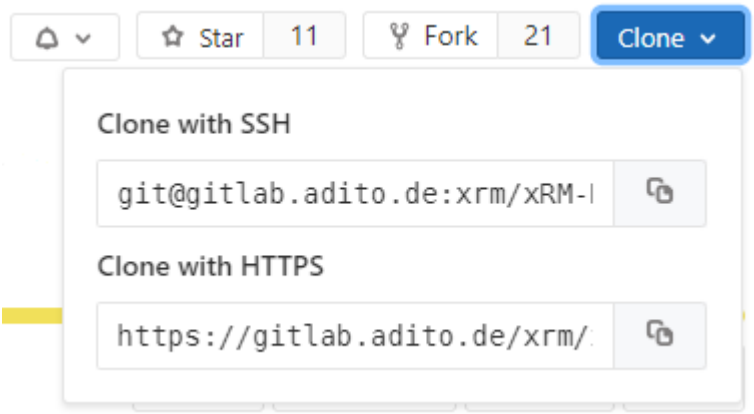
1. **create empty project**
   First you have to choose a name and the storage location.
   This will create, as the name suggests, a completely blank project with not a single data model with the exception of empty preferences.
2. **load template from server**



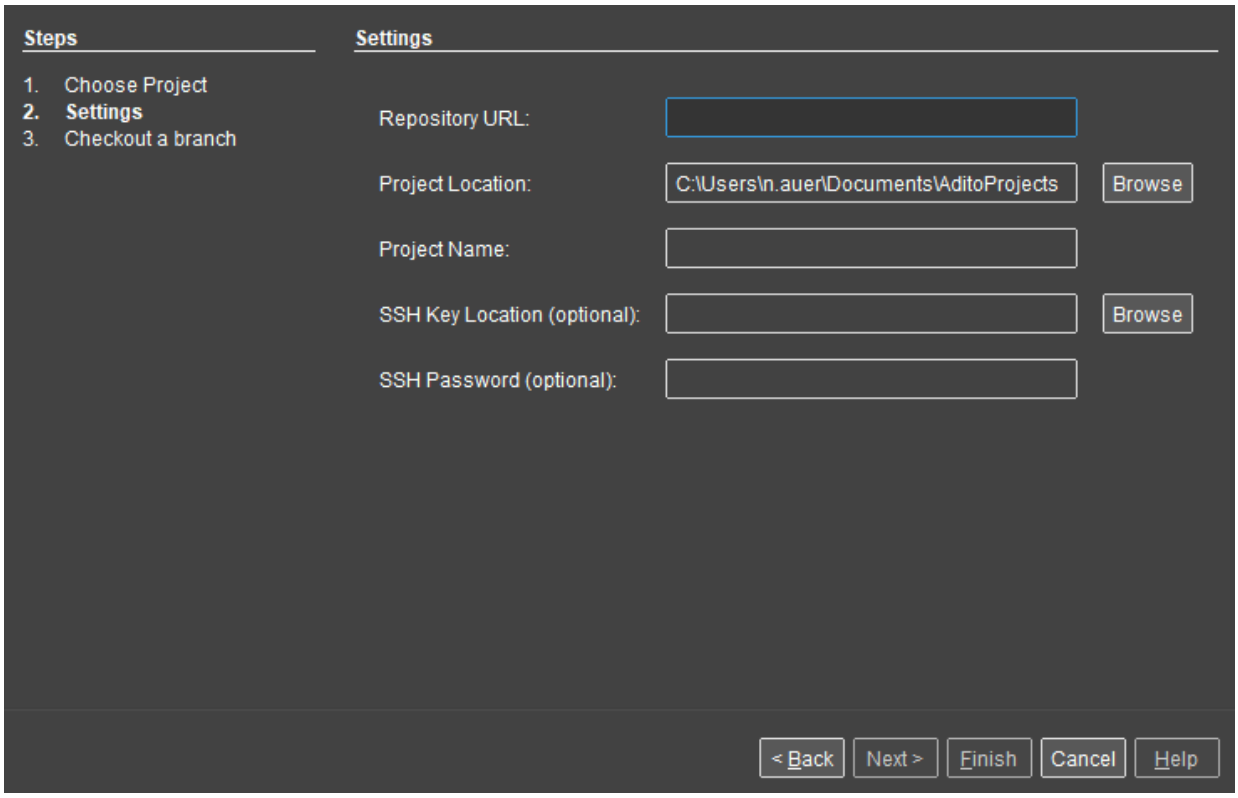
This option is used to create a new local project out of a already existing 'compressed' project. Copy the storage path into the field "Repository" and the project will show up. You may have to give a username and password when connecting to the "Repository".
Again choose a name and storage location in the following window.

3. **create from system db**
   Select the name and storage location for your project, but also the system db file, from which the project shall be generated.

4. **clone from git repository**
   Firstly you have to select the way to connect to the git server where the project is saved. Either via https or ssh. In both cases you will need the right address. Connect to git with your browser and navigate to your project. here there will be a button "clone" where you are able to obtain both keys.



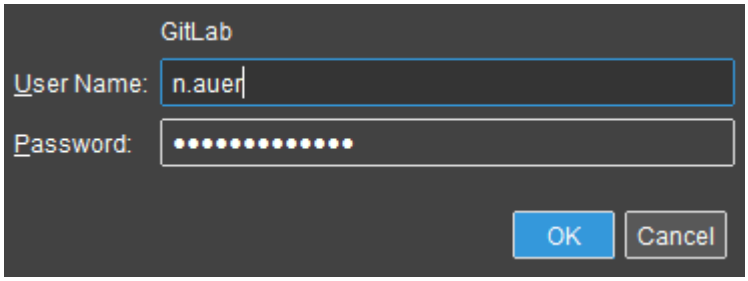
Copy the key for the preferred method into the "Repository URL" field.



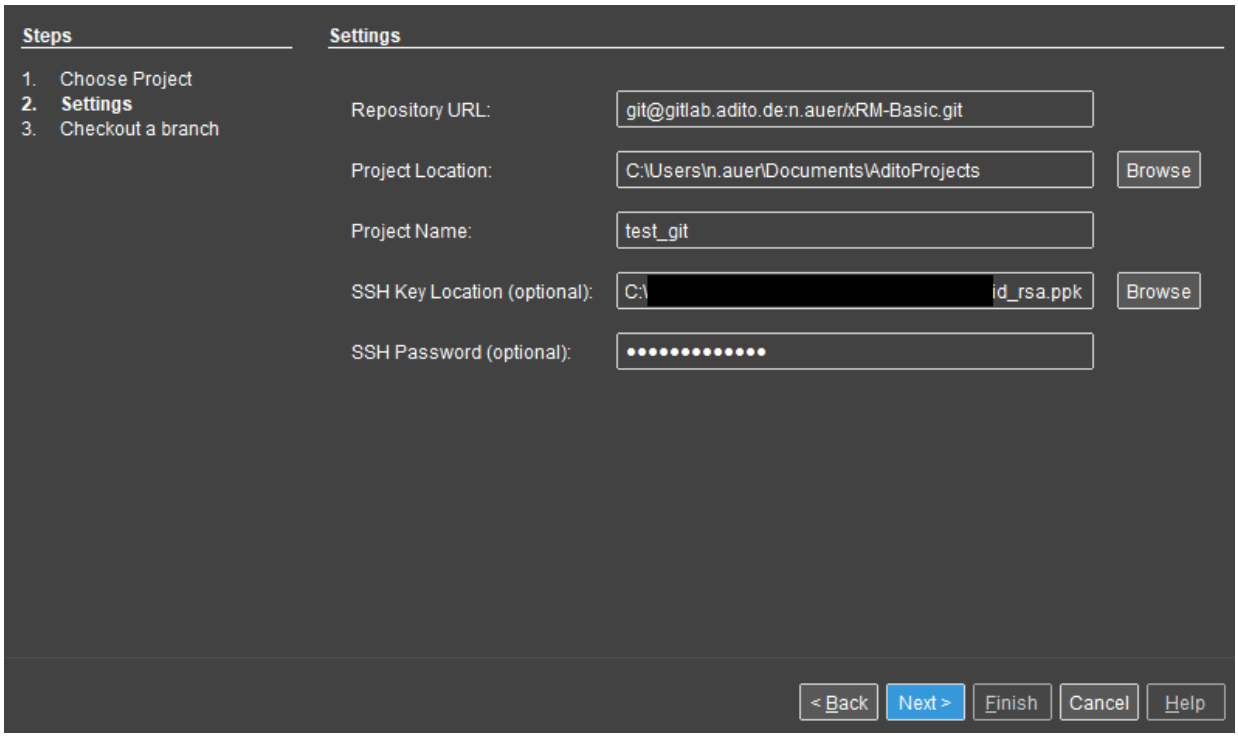
In both cases choose a name and storage location.

The name must not be "dist" or "node_modules", as these are names reserved for internal use.

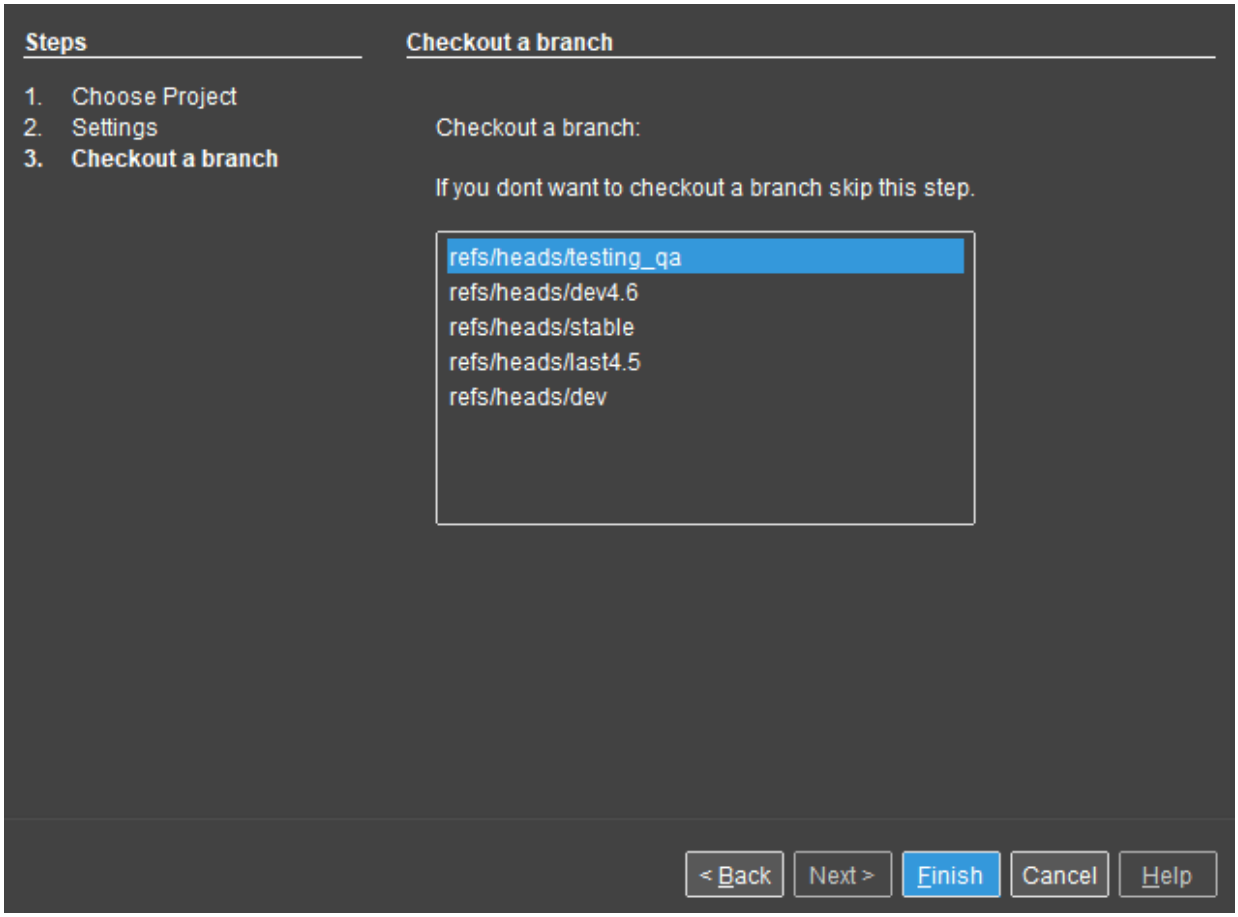
When working with the https key you will now be asked for you git login



In case of the the ssh key you have to select the physical key, generated with puttyGen and deposited in git, and its password.

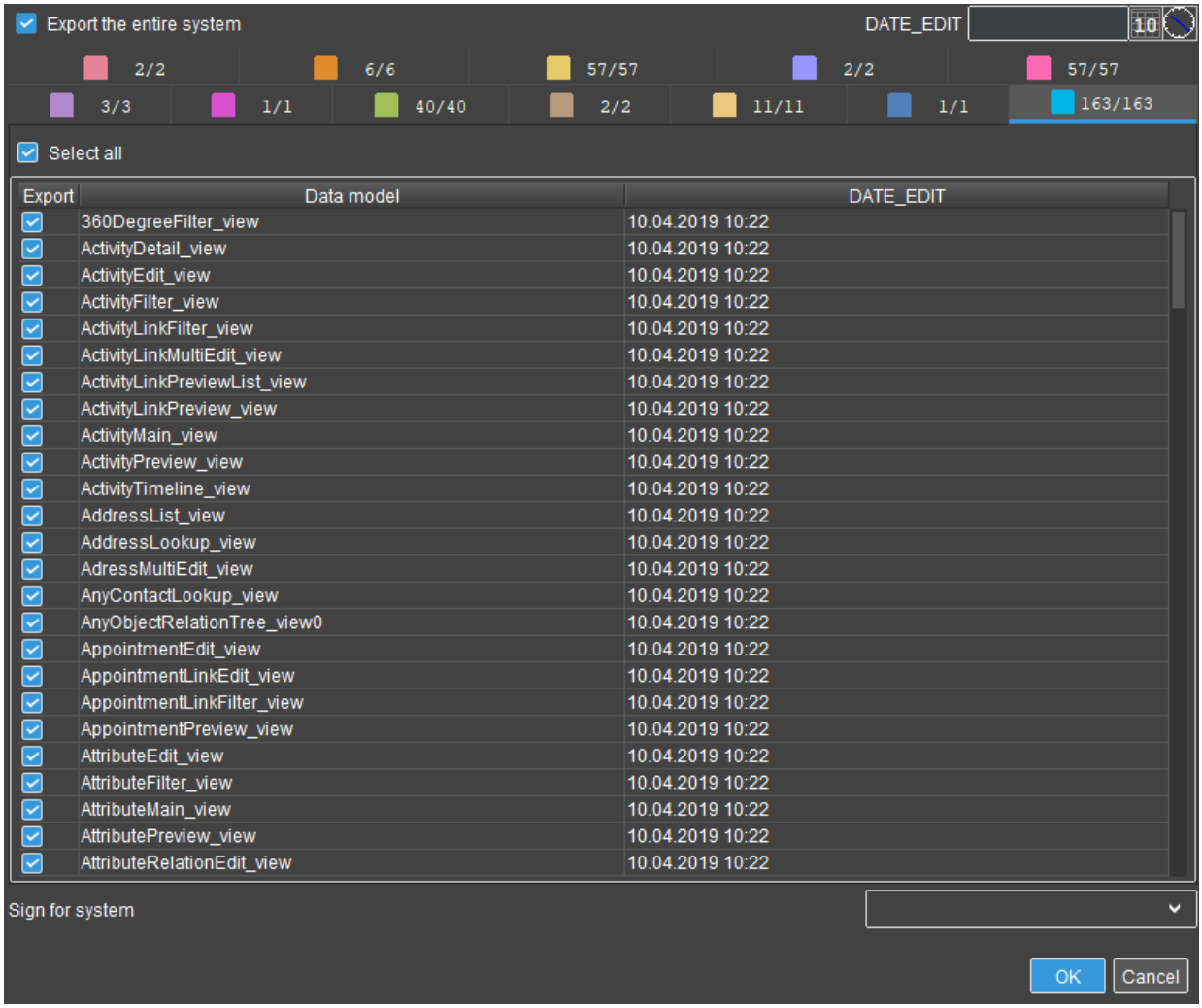


In both cases you will have the option to checkout with a specific branch of your project.



4.1.2. Import/Export Project

When importing/exporting a project, the process is mostly the same, just the other way around.



When exporting a project, this project has to be the currently selected project.

You may choose all, or just one datamodel which you want to export. Do this by simply clicking the checkbox.
In the next step choose the storage location for your export, and a zip-file will be generated.

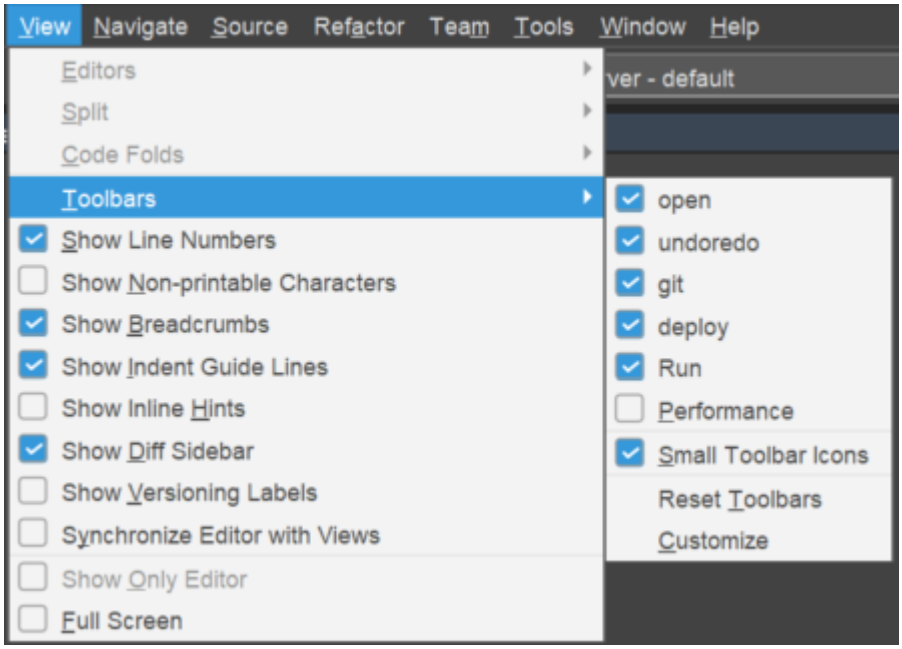
When importing a project you just have to choose the zip-file and the same screen shown above will appear.
You again have the option to deselect datamodels for the import.

Should one imported datamodel have the same name as a existing datamodel you have the option to override the existing datamodel or cancel the import.

4.2. Edit

There is no difference between this menu point in ADITO and NetBeans, with the exception you cannot use macros in ADITO. For this reason please use the NetBeans documentation for any questions.

4.3. View



With the menu point View - Toolbars, the toolbar beneath the menu bar can be configured.

The following options can be selected (most of them is selected by default):

Git

	Commit your changed files
	Pull the newest version
	Push your changed files
	Diff the local changes
	Show the project history
	Show local changes made

deploy

	Deploy of complete project

	Deploy of opened data models
	Open the SQL prompt

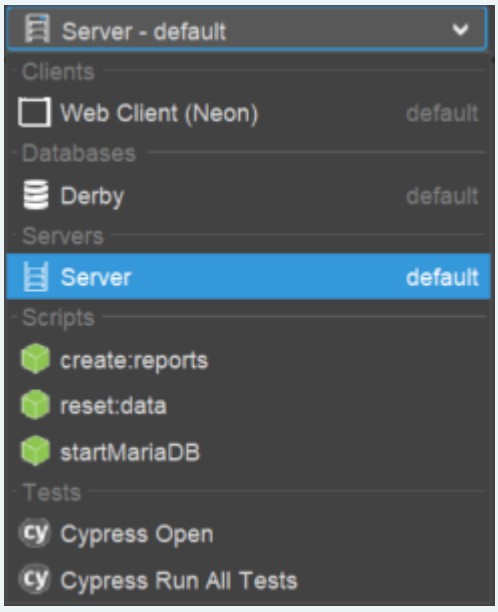
open

	Create a new data model
	Open the searchbar for data models
	Save your project

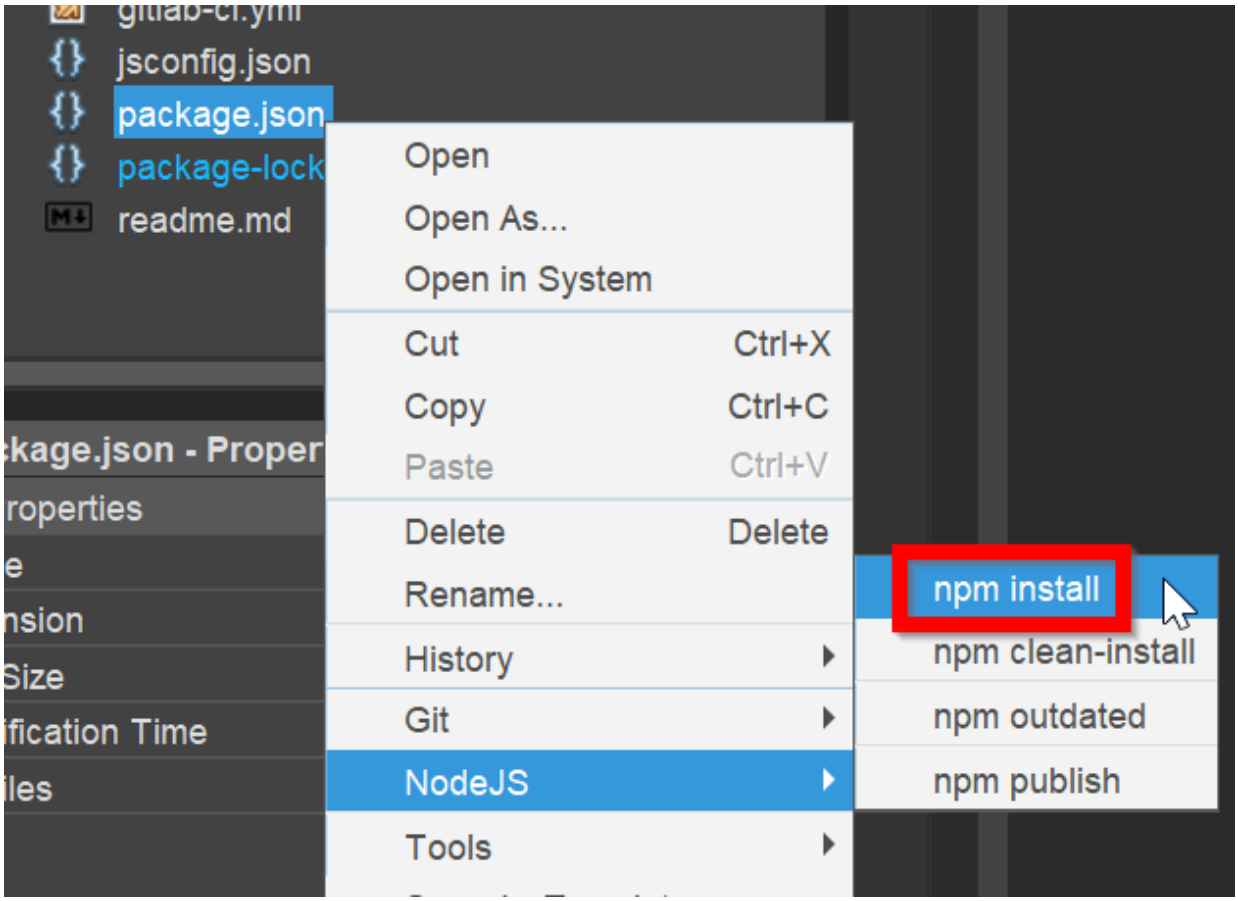
undo/redo

	Undo the last action
	redo the last undone action

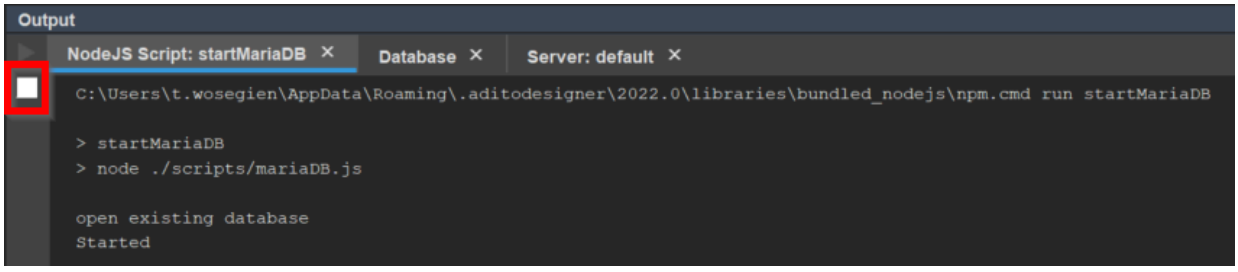
run

	Select the instance which you want to start
	Start the selected instance

The items selectable in this combo box are named "run configs". They are used, e.g., to start the ADITO server or the browser including the ADITO Web Client.
In section "Scripts" you can find the run configs of the NodeJS scripts, but only if you have installed the NodeJS plugin. (This plugin allows you to create NodeJS scripts, which will be stored, keeping with the standard, in package.json).
The NodeJS scripts will not work unless you have executed the "npm install" command via the context menu of the file "package.json" (in the lower part of the "Projects" window):



After executing a NodeJS script, its output is shown in the Designer's output window "NodeJS Script: <name of script>". Here, you can find a white, square button that enables you to stop the script:



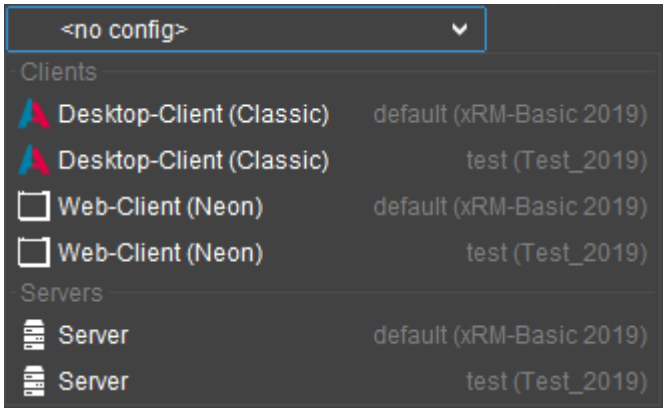
In section "Scripts" you can find the run configs of the frontend-related tests written with Cypress.

performance

	Shows the current Designer performance. A click activates the garbage collection
	Profile the application and create a screenshot
	Pauses and resumes I/O checks

The performance tool is disabled by default.
Please look into the NetBeans documentation for further information.

When more than one project is open the project name will show when selecting you instance



With the button "Reset Toolbars" all changes made to the toolbar will be reverted.

If you want to add additional functions to your toolbar, you can select many different functions in "Customize" and add these via drag and drop to your toolbar. Also you are able to move icon while this list is open via drag and drop.

4.4. Navigate

There is no difference between this menu point in ADITO and NetBeans. For this reason please use the NetBeans documentation for any questions.

4.5. Source

There is no difference between this menu point in ADITO and NetBeans, with the exception you cannot use macros in ADITO. For this reason please use the NetBeans documentation for any questions.

4.6. Refactor

This menu group provides options for refactoring, in particular, for deleting and renaming ADITO models, such as Contexts or Entitys. The functionality is the same as the corresponding items in the context menu of the respective ADITO models.

In some cases, particularly when deleting or renaming, a tab "Refactoring" will appear in the lower middle part of the Designer (it can very easily be overlooked!), which requires you to confirm the refactoring by clicking on button "Do Refactoring". Here you see all models affected by the refactoring, and on demand, you can uncheck part of them (not recommended!). If you miss to react to the refactoring prompt (or repeat the action that caused it) and continue working, your XML project source code might become confused and you will have to repair it manually.

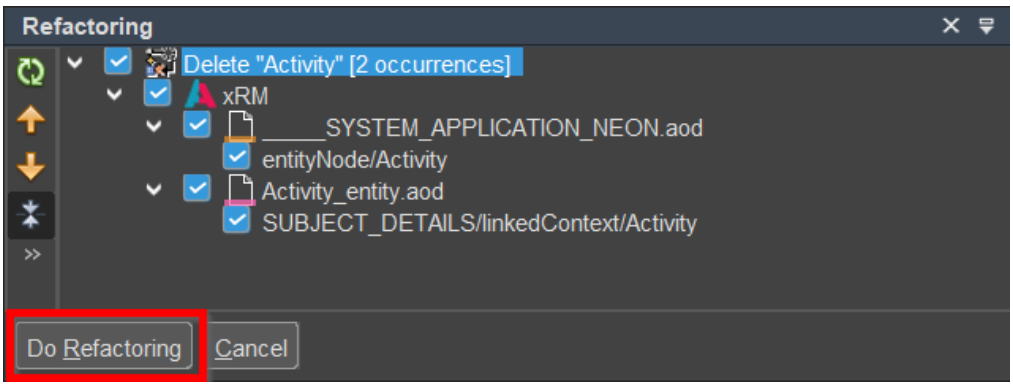


Figure 1. Example of content of tab "Refactoring"

The refactoring will fail or be executed incompletely, if you had cancelled the "Indexing" process - please refer to chapter "[Scan services]".

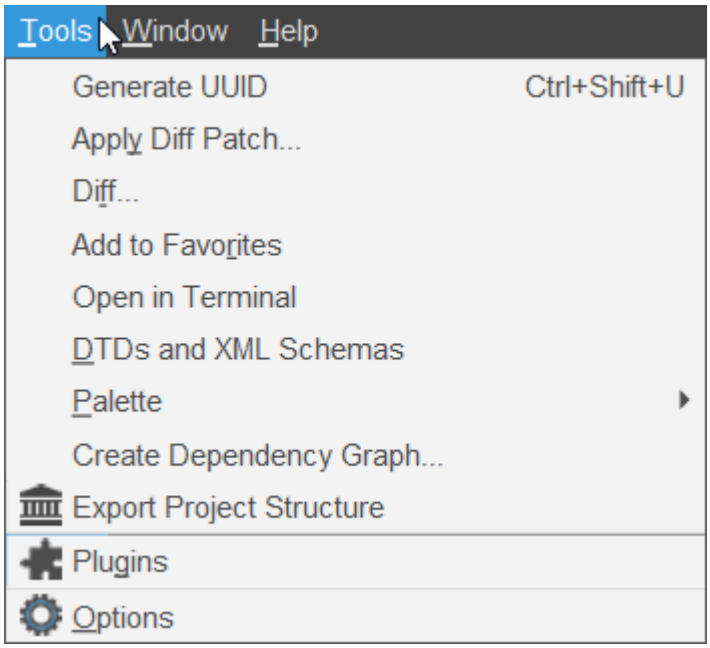
4.7. Team

The menu point team changes depending if the Git plugin is installed or not, refer to the Git-Plugin chapter for further information.
If it is not installed you have the option to "Shelve Changes", NetBeans Standard and use the "Versioning".

Should Git be installed on the other hand, you have will find the different Git-Commands.
For further questions about those please visit the official Git forum.

4.8. Tools

4.8.1. Overview



Generate UUID	Generates a random 36-digit hexadecimal UUID (see appendix UUID Generator)
Apply Diff Patch…	Update your project using a diff file selected by the user.
Diff…	Enables the comparison of the source of the currently marked model with a file selected by the user.
Add to Favorites	Mark a data model/system-component/project as a favorite. It will then be displayed in a sub-window of the "Projects" window.
Open in Terminal	Opens the Terminal window (requires cygwin to be installed).
DTDs and XML Schemas	see NetBeans documentation
Palette	see NetBeans documentation
Create Dependency Graph…	Creates an illustration of all dependencies of the complete project, in one large png file. **CAUTION**: This can take several minutes, while the Designer is blocked.

Export Project Structure	Exports the structure of the complete project, as csv file (see sub-chapter Export Project Structure below)
Plugins	See chapter Plugins and its several sub-chapters.
Options	Opens a tabbed dialog for configuring editors, colors, shortcuts, the debugger, and many more (see sub-chapter Options below)

4.8.2. Export Project Structure

The tool "Export Project Structure" enables you to export the structure of the complete project, as csv file, with comma as separator.

The design of the export structure reflects the requirement of a specific customer group and may thus not meet every user's expectations.

The csv file consists of several rows, one row per structure element (e.g., a table), and 6 columns, whose general meaning is as follows (exceptions possible):

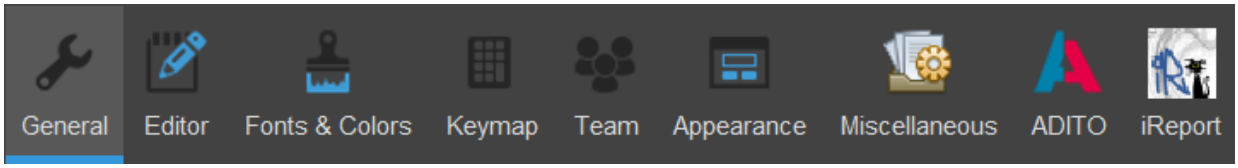
1. module: The module to which the structure element belongs, e.g. "Contact Management". In most cases, a "module" is identical to a specific menu group of the Global Menu.
2. Context - in most cases identical to a specific menu item of a menu group.
3. type of visual element, e.g., FilterView, PreviewView, "G Tab", "D Tab" (tab of a MainView). To simplify filtering in Excel, prefix "G" means that it is the first row of the view element (with colums 4, 5, and 6 being empty), and "D" marks the following rows.
4. name of the sub-component visual element, e.g.,
    - FilterView/PreviewView: title of the Entity
    - lable of tab in a MainView (= title of ViewTemplate)
5. The title of the sub-element, e.g., of a TableViewTemplate.
6. The lowest structural element(s), e.g., the columns of a TableViewTemplate (separated by semicolons), or the title of an Action.

Some cells have three slashes ("///") as prefix. This indicates that the cell's content could not be retrieved statically, because, e.g.

- the title is generated at runtime, via a titleProcess or
- there is no translation available

Instead, the name or the non-translated title of the object is given. This will help you to find the corresponding place in the web client, in order to fill in / correct the cell value by yourself.

4.8.3. Options



4.8.3.1. ADITO

The subject "ADITO" contains the following settings:

General	Settings regarding -Dialogs -System Editor -JDito Upgrader -Adito Hints - Scan Services (refer to the corresponding chapter)
NodeJS	Here you can change your installationpath for your NodeJS instance and also download a desired version.
ESLint	Enable code analyzis after you save a file.
Translators	Here it is possible to specify API keys for Google, DeepL and Yandex translators (refer to the corresponding chapter).
Debugger	You activate the Debugger here and choose to show the elapsed time (refer to the corresponding chapter).
Cloud	Change the buffer size for the logging with RXJava backpressure.
Encoding	Change your encoding in your ADITO Designer if necessary.
Data Sharing	Enable or Disable the sharing of anonymous usage statistics (recommended to enable it) .

4.8.3.2. iReport

The subject "iReport" contains the following settings regarding Jasper-Report:

General	Here you can define the standard unit and make general settings in the areas Report defaults, Designer, Expression editor, and Compatibility.
Classpath	Classpath lets you manage existing JARs or add more.
Fonts	Fonts can be managed or added .
Viewers	In this tab it is possible to specify the paths of the viewer to be used.
Wizard Templates	Here you can manage templates or add new ones.
Compilation and execution	In this tab, you can change the compilation default directory, customize the execution options, or make settings in the Parameters prompt or Virtualizer sections.
Query Executers	Here you can adapt the Factory Class and the Fields Provider class to the respective language or add new ones.
Export options	In this tab export settings in the areas common, PDF, HTML / XHTML, Excel, CSV and text can be made.
JasperReports Properties	Here it is possible to complete or modify properties for JasperReports.

4.8.3.3. Other options

For information regarding the subjects "General" till "Miscellaneous" please use the NetBeans documentation. In the following, selected options are explained.

4.8.3.3.1. Janitor

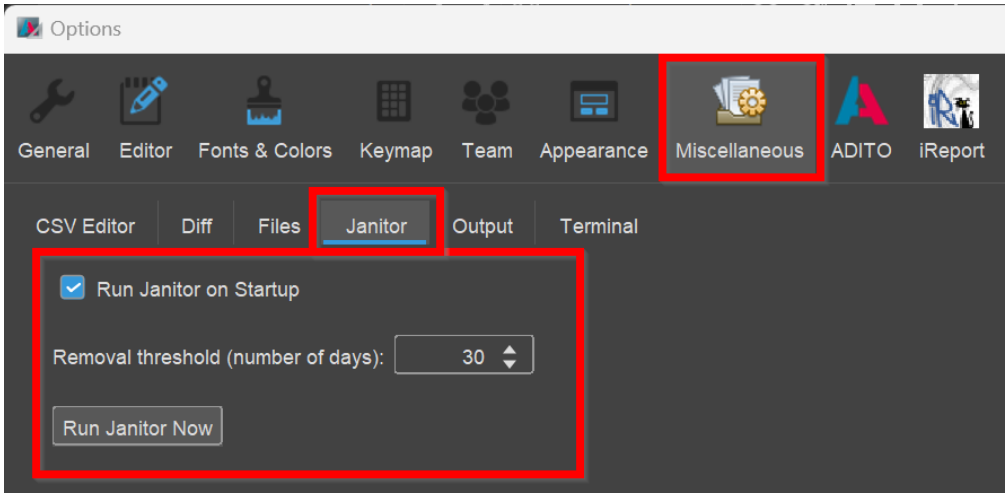
Janitor is a NetBeans feature that is also included in the ADITO Designer. It helps you to delete outdated version folders and thus free up disk space. Version folders are the folders holding cache data, plugin-related data etc. You can find these folders in your roaming directory:
C:\Users\<username>\AppData\Roaming\.aditodesigner\<version number>,
e.g., C:\Users\j.smith\AppData\Roaming\.aditodesigner\2023.0.2

Janitor runs one minute after startup and checks, if the version folder was not used for more than (by default) 30 days. In this case, a notification is shown that this folder might be deleted, which could free up a specific amount of disk space. If you click on the respective link in the notification, the folder can be removed immediately.

Under Tools > Options > Micellaneous > Janitor you can configure

- if Janitor should run on startup;
- the removal threshold (default: 30 days), i.e. the minimum number of days without changes, which makes Janitor classify a version folder as "outdated".

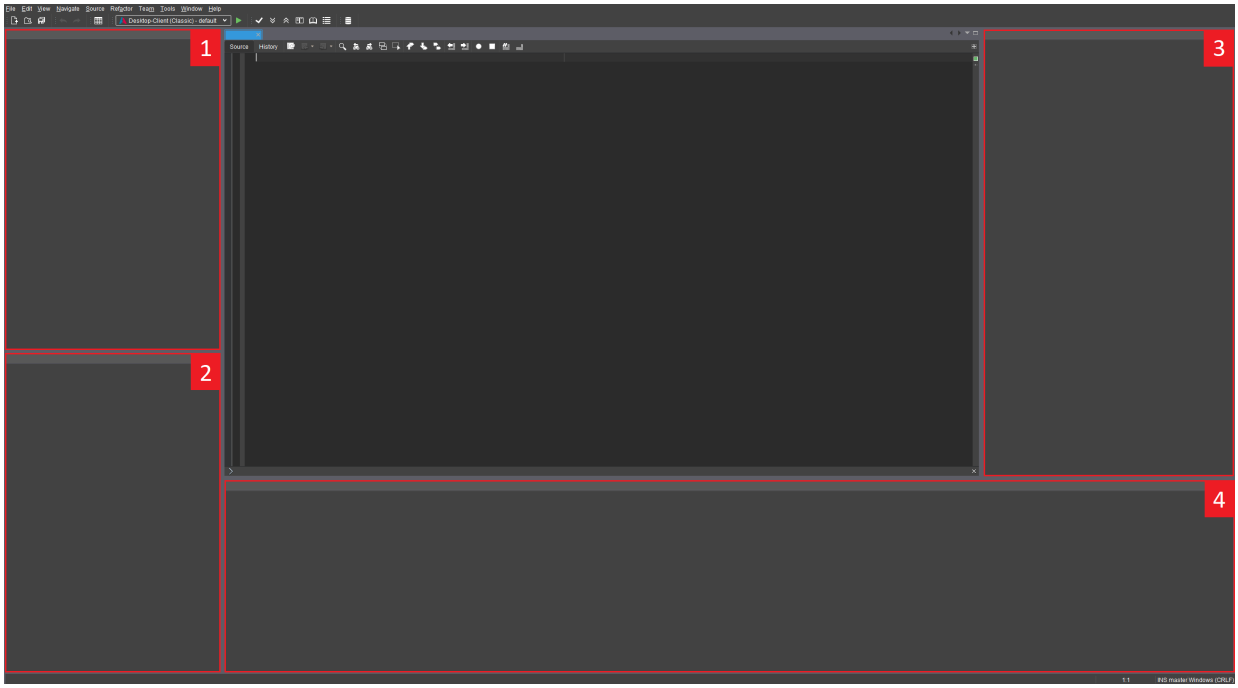
Furthermore, you can run Janitor manually via the button "Run Janitor Now".



4.9. Window

Using the "Window" menu item, you have the option to select/deselect from a variety of different windows with different functions.

The arrangement of all windows can be changes via drag and drop.

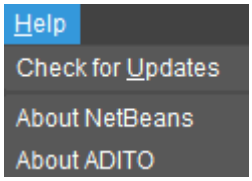


Name	Standard Position	Usage
Projects	1	Shows the open project(s) as a tree table, split into data models.
Files	1	Shows the open project(s) as a tree table, split into its folder structure
Favorites	1	Shows all as favorites marked files/data models/project/…
Output	4	Logs everything happening for the selected server/client/…
Services	1	Shows a logical overview of run-time resources e.g. database drivers
Scan Services	4	Lists all things detected that should be reworked/looked over like TODOs or warnings
Navigator	3	Gives an easy overview of the selected file (e.g. methods)
Debugger	4	See the corresponding chapter for further information

Name	Standard Position	Usage
Palette	3	Shows all components available for the selected data model
Properties	2	Shows the properties given to the selected data model
Bookmarks	4	Shows all toggled bookmarks (via menu point "Navigate")
Notifications	4	Shows all application wide messages which are not specific for a file or project
Terminal	4	A terminal window for specific command lines
Report designer		Multiple windows helping when creationg a report
Editor	Middle	The place where the code or opened frames will show up

For any questions regarding the remaining menu points please use the NetBeans documentation.

4.10. Help



1. **Check for Updates**
   Check for available Updates.
2. **About NetBeans**
   Show a short summary of the source NetBeans-Version.
3. **About ADITO**
   Info show the Designer-Version as a image.
   Licence show the used licence.
   System contains multiple values regarding a wide variety of information.
   This information is sorted via a key which shows its origin.
   For example java.runtime.version contains the used java version.

The values regarding ADITO are:

adito.designer.build	The exact buildnummer of your Designer version.
adito.designer.data	This is the storage location for database information like logs.
adito.designer.home	Here is the path for the opened Designer.
adito.designer.version	The installed ADITO is shown here.
adito.db.derpy.start	A boolean value showing if a derby database is/was started.
adito.home	The installation path for ADITO.
adito.netbeans.buildnummer	This is the combination of .build and .version

5. Toolbar

For instructions regarding the toolbar please take a look at the chapter Menu bar, subchapter View. Everything listed there for the "Toolbar" section is still relevant for this chapter.

Please note you can also access this menu point by right-clicking into the toolbar.
If you want more information on how to customize your toolbar or add a new one, please refer to the NetBeans documentation.

6. Project Structure

To create a new empty project, select File > New Project > create empty project. Specify its name and location, and press the "Finish" button. This will create a new project with a substructure, in which ten different sub-nodes appear:

- system
- preferences
- classic
- neon
- process
- report
- language
- role
- alias
- others

These substructure will be explained below.

In most cases, you will not create a new project from scratch, but you will use the standard project "ADITO xRM" (or one of its industry models) as template for a new project, and then customize it according to your requirements, e.g., by modifying existing Entities, or creating new Contexts, Entities, and Views. Nevertheless, in the following, you will find a description of how to create a completely new project and its sub-structure.

The general creation procedure usually is:

1. Create a new ADITO model (system, Entity, View, etc.) in the "Projects" window, the included "+ New …"-button in the entity or in the Navigator window, respectively.
2. Configure its properties in the "Properties" window.
3. If required: Create its substructure in the Navigator window.

In the Designer there are various mechanisms to simplify the configuration of new models - please refer to appendix "Configuration helpers".

6.1. System

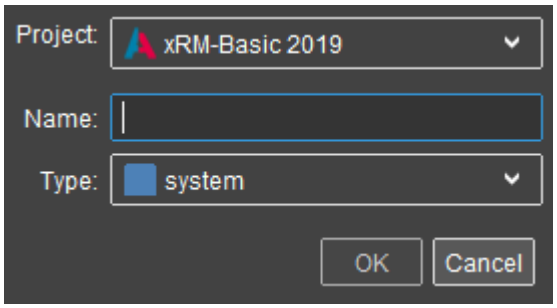
The system model is the access point for all external systems which can connect to ADITO like e.g.

1. Database Systems
2. Mail Server
3. Groupware Systems

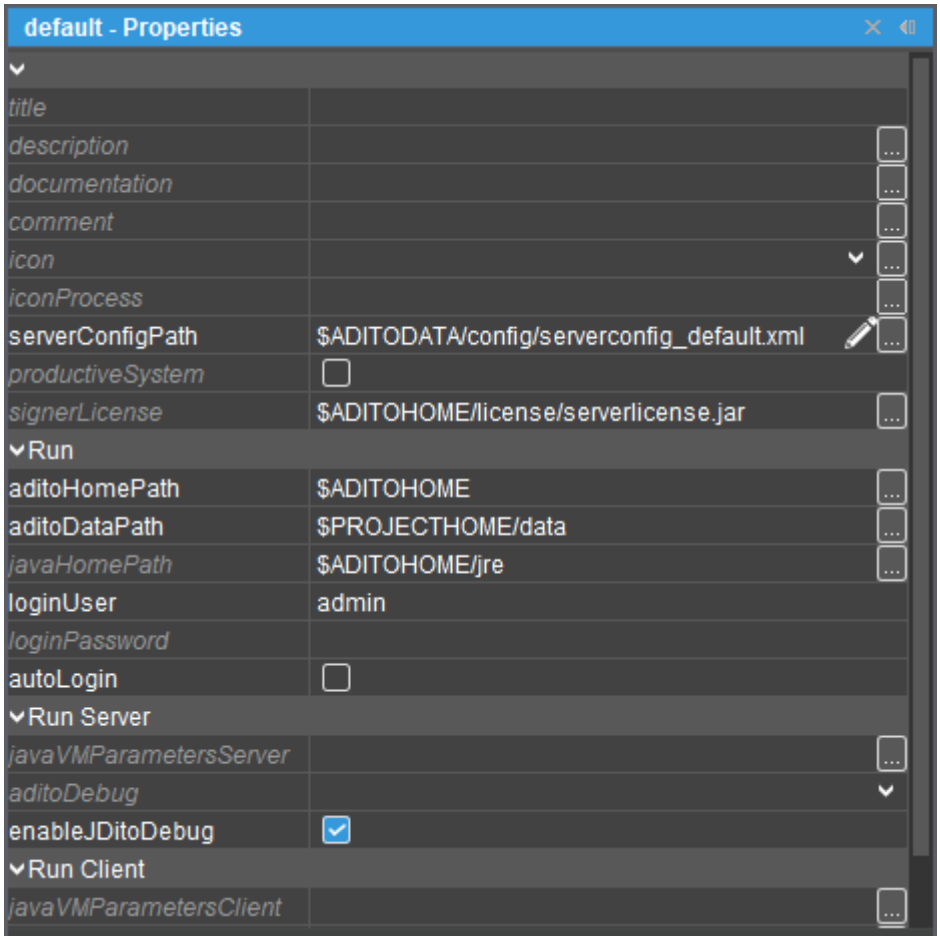
In addition to being the access point, the system settings also define the ADITO server itself.

6.1.1. Create a new System

To create a new system, right click onto the "system" folder and select "New". Alternatively you may also use the option "File" - "New" - "New", in this case please make sure that the right type is selected.

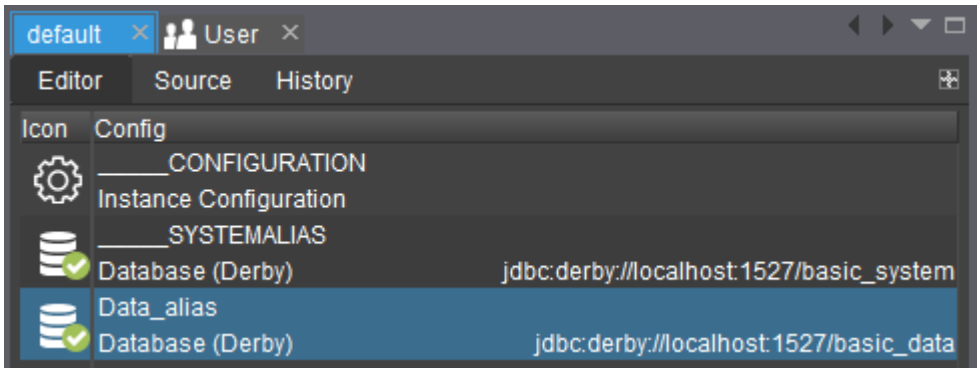


But before you can effectively do anything with it, the server has to be started.
So please make sure a serverconfig file is available at the location given under properties as well as the actual license file.



The server can be started using the buttons in the toolbar.

6.1.2. Overview

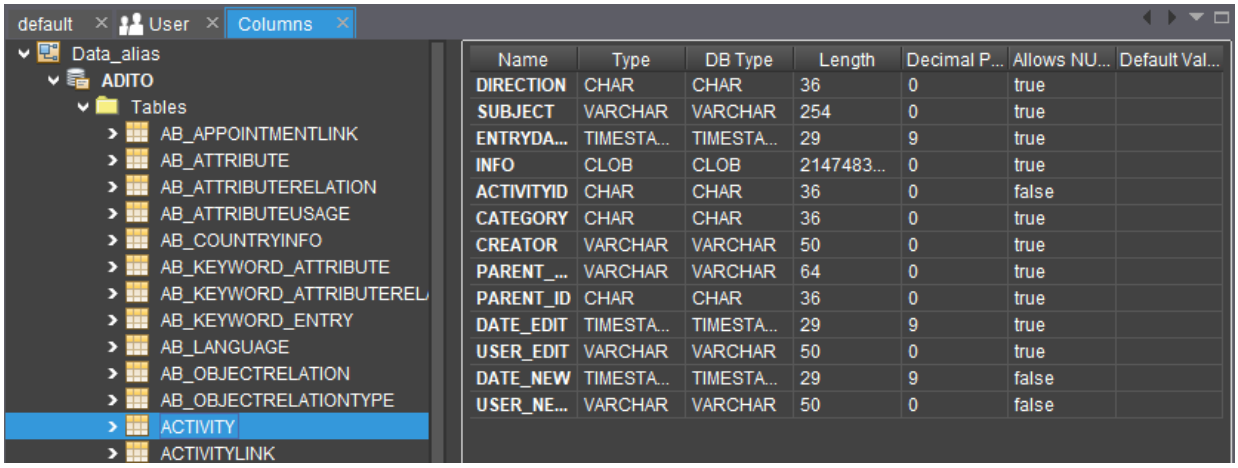


By default the connected systems together with the configuration menu will appear as well as a tab with all users present in ADITO. In this example there are two databases, both of which can be accessed via a double click in addition to "___CONFIGURATION". Should there be more external systems connected they would show up here together with their address.

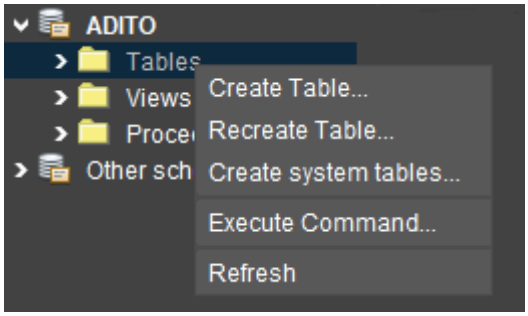
There will always be two connected data bases "___SYSTEMALIAS" and "Data_alias". All tables

regarding the system or the client, normaly with the prefix "ASYS_" are saved in "____SYSTEMALIAS". The data you normally work with in the client, like persons or contacts, is saved in a table in "Data_alias".

Via a double click on an instance, like a database, it can be accessed.



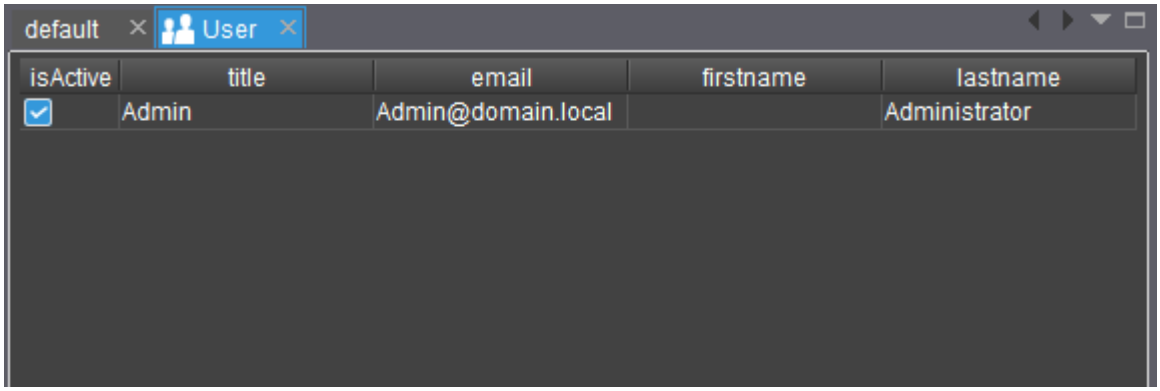
It is also possible to create or alter database tables here. With a right click you open the following window.



Simply select "Create Table" and you will get a pop up window where you are able to easily create a new window. The same way you may also create system tables with "Create System Tables" but it is recommended you don’t do this. There are more options to alter tables when directly clicking on a table.
You are also able to create views and store procedures in their respective folders. These work the same way in ADITO as in any other database system.

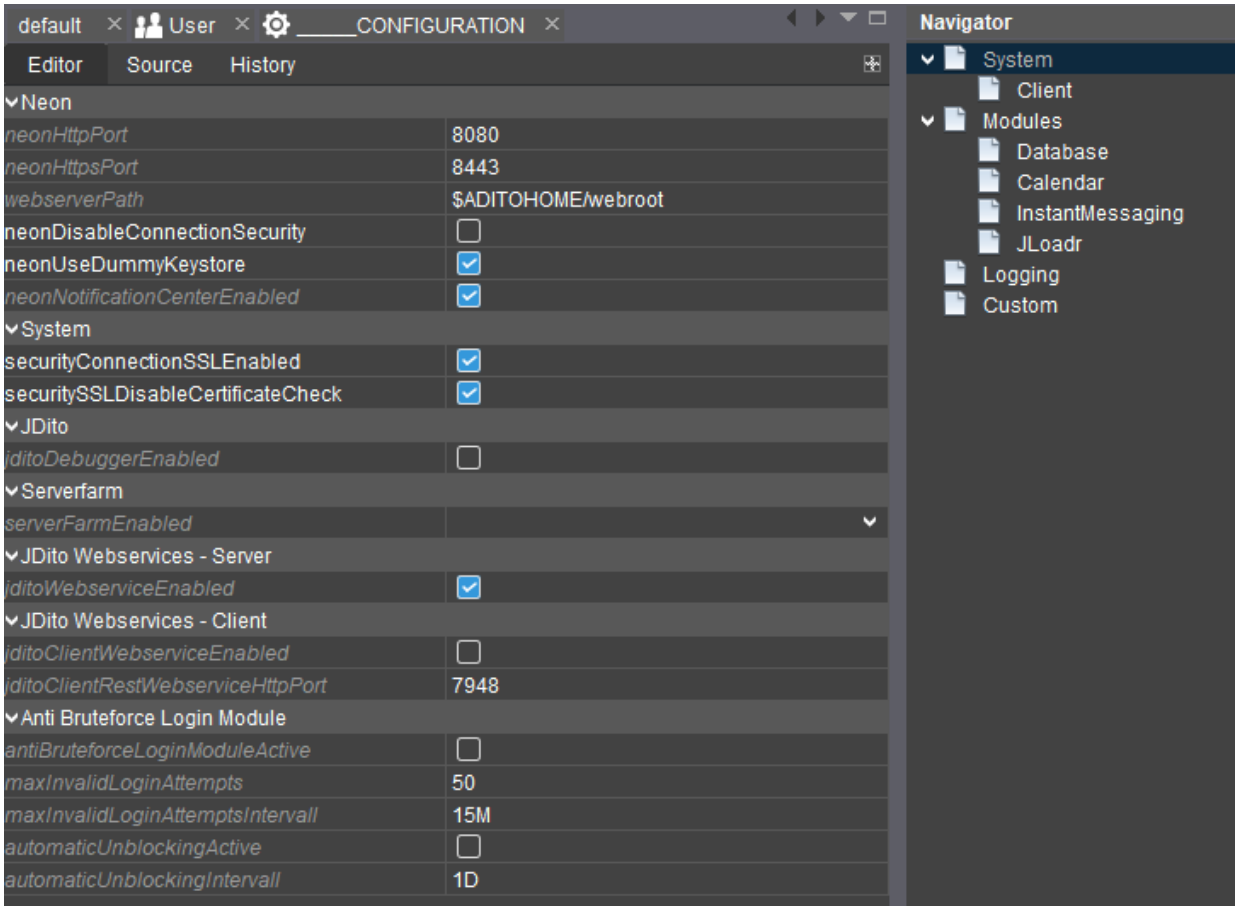
In addition it is also possible to create more different schemas in the "Other schemas" tab. These can be simply selected as the default with a right click.

6.1.3. Users



In the tab "User" all users are shown. Also you are able to create Users here via a right click. These functions are relatively the same when you create a user via the frame. But further there are some option which can only be selected in the Designer via the "Properties" window. For example the "INTERNAL_..." roles can only be assigned here.

6.1.4. System Configuration



With a double click on "____CONFIGURATION" you open the system settings regarding a multitude of options of the ADITO server as well as the ADITO client. All options are grouped in multiple categories

shown in the "Navigator" on the right side.

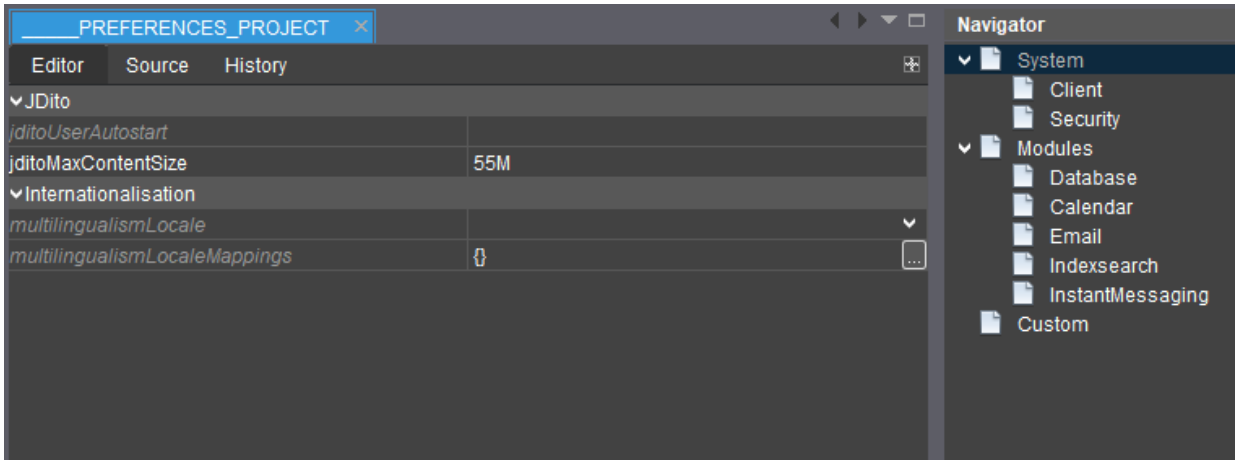
Do not mistake these 2 configuration options: The properties of "CONFIGURATION" all refer to the *system*, while the properties of "PREFERENCES_PROJECT" (in the project tree, under the node "preferences") all refer to the *project*.

System	information regarding the client/server ports as well as security measures
Client	Options to disable or enable the different client types, to modify the client appearance and to configure which login types are being used
Modules	Disable or enable Indexsearch, e-mail, telephony and snmp
Database	Options for database audits and timeouts and Master and Slave settings for offline synchronisation
Calendar	Determine calendar storage locations and the synchronisation with other calendar applications
InstantMessaging	Settings concerning the instant messaging in the client on the server level
JLoader	Settings concerning the download manager
Logging	De/activate the logging function for multiple different ways
Custom	Create own custom properties

6.2. Preferences Project

In addition to the system settings, there are more options for the client and server in the data model "preferences" (see second main node of the project tree in window "Projects").
These settings are specific to the project and are the same for all systems within the current project.

Do not mistake these 2 configuration options: The properties of "CONFIGURATION" all refer to the *system*, while the properties of "PREFERENCES_PROJECT" (in the project tree, under the node "preferences") all refer to the *project*.



System	Specify a language and the autostart
Client	Set a default browser and login limits as well as the client search plus feedback options
Security	Define requirements for user passwords
Modules	No content
Database	More options for the database audit and new options determining how to handle large data packages
Calendar	Enable a standard sorting and define categories (e.g. events)
Email	More specific options on how to handle e-mails
Indexsearch	Enable the index search and define the indexer (when indexes should be build) + general rules
InstantMessaging	Settings concerning the instant messaging in the client on the client level
Custom	Create own custom properties

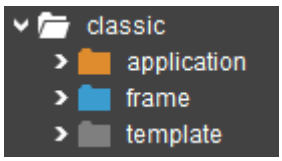
You can access these preferences via the methods of module `project`.

Example:

```
import { project } from "@aditosoftware/jdito-types";
```

```
JSON.parse(project.getPreferenceValue("custom.dsgvo.active", "true"));
```

6.3. Classic



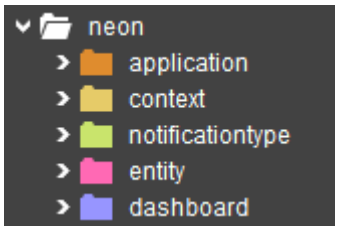
Classis in ADITO is the "swing"-client you might know as the default client of older ADITO versions using "Frames".
In "application" each frame is places in a location, which would create the sidebar in the client.

Frames contains all frames, now known as context-types.

When a component is placed inside a frame, you may often need exact measures or other setting for more than one component. To simplify this, you can create a template with all settings and apply the template for your component under "Properties" - "template". All templates are managed under "classic" - "template".

For further information on how to work with frames etc. please refer to the Coding-Guidlines.

6.4. Neon



The node "neon" includes groups containing data models that are used in the ADITO Web Client (sometimes refered to as "Neon"), which is the new standard client since version ADITO 2019.

The group **"application"** fulfils a similar role as "application" in "classic": A double-click on "____SYSTEM_APPLICATION_NEON" opens an editor to modify the Global Menu of the Web Client.

"context" contains Context models, which can then be referenced in the editor of the Global Menu (see above).

You may create custom notifications for your project. These are managed in **"notificationtype"**.

"entity" clusters ADITO Entities, which are used to model the data structure of the ADITO system.

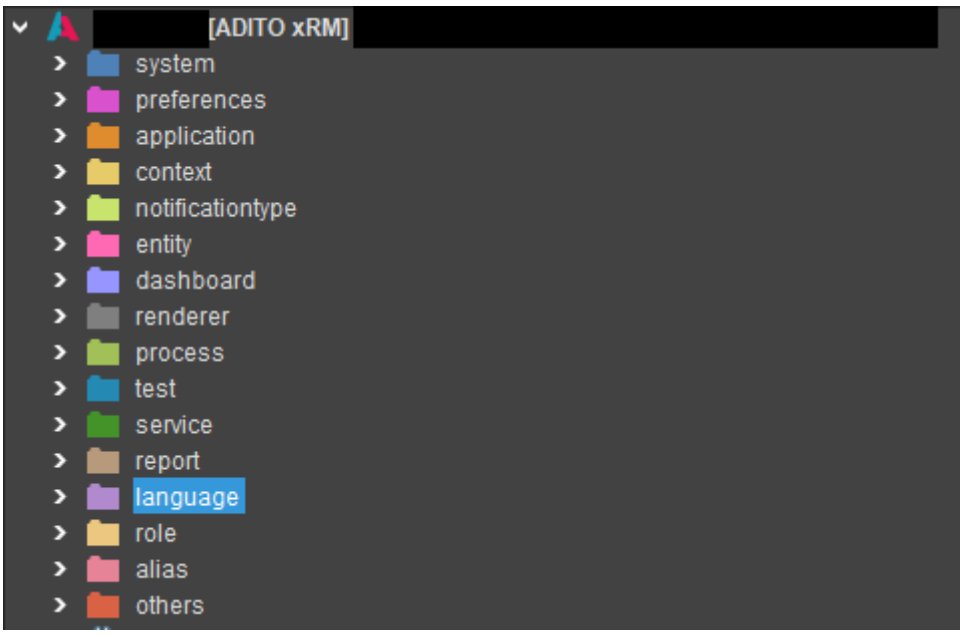
"dashboard" includes Dashboard models, which are used for showing web pages that include Dashlets,

which in turn included capsulated Views of specific Contexts. Dashboard "Home" is always shown after logging into the Web Client, while further Dashboards (e.g. Sales Dashboard) can be opened via the Global Menu.

"renderer" contains definitions of Renderers, which, e.g., are required for the ViewTemplate MultiEditTable.

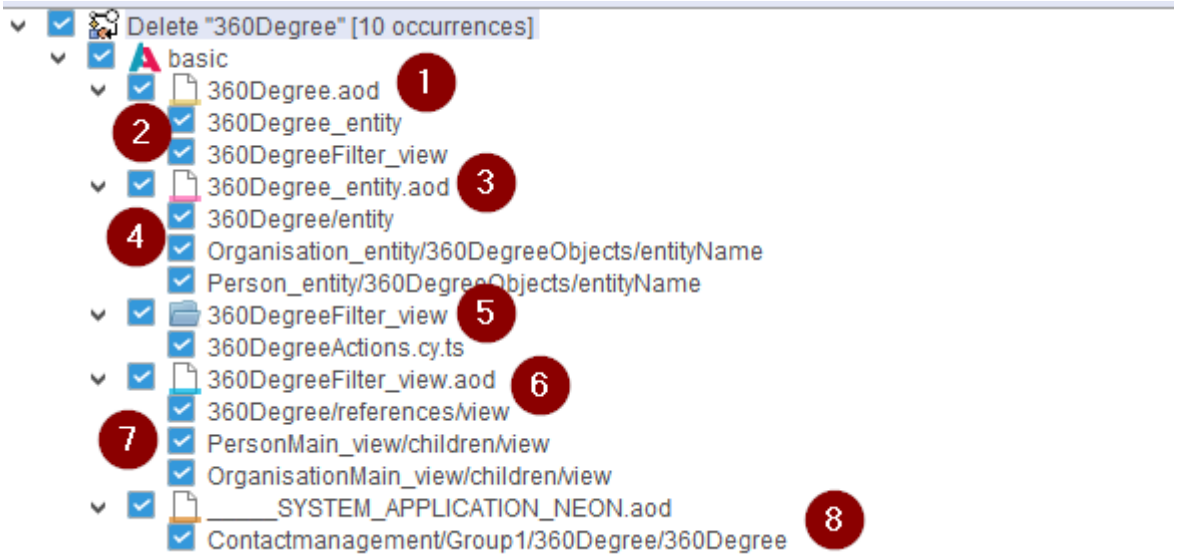
Find detailed information about how to handle the Neon data models in the Customizing Manual and in the various "ADITO Information Documents" (AIDs). Therefore, the above paragraphs are only an overview.

Newer version don’t support the "classic" ("swing"-client) folder structure. The project will look like this:



6.4.1. Deletion of a context

With 2022.2.2 you will be able to delete a context with the connected entity, views, SYSTEM_APPLICATION_NEON entry, cypresstests, consumer and references in your system



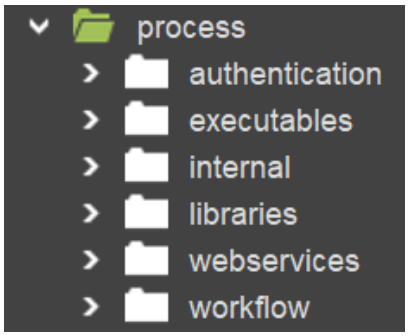
1. The Context to be deleted
2. The children hanging under the context
3. The entity of the context
4. The references of the entity
5. The order of the Cypress tests and the tests to it
6. The view of the context
7. The references of the view
8. SYSTEM_APPLICATION_NEON and its use in the menu.

Explicitly, this means:

- If all checkmarks are set (as in the image above), the Context, View, Entity, Cypress tests and the folder where the Cypress tests are located will be deleted. It also deletes the context entry from SYSTEM_APPLICATION_NEON and the view and entity references.
- If you want to keep the view, you have to uncheck (6). The check mark at (2) only indicates that in 306Degree.aod (the AOD of the context) the entry is removed, but not the view. Similarly for Entity (3) and Cypress tests (5)
- For Cypress the folder (under basic\cypress\e2e\singleTests) is deleted and the files under it. You can deselect child items here, which will not be deleted, or the whole folder, so that no Cypress tests will be deleted.
- If all checkmarks are removed and you press Refactor, the context is the only one deleted (default behavior).

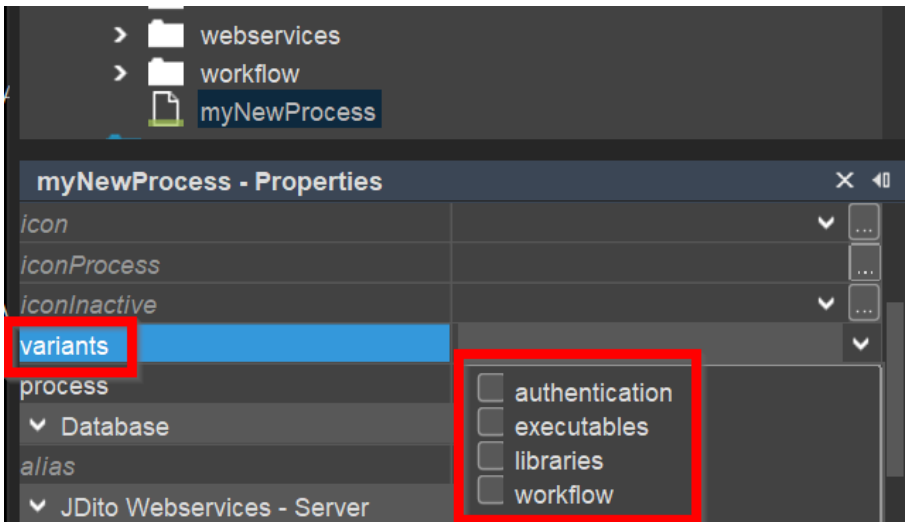
- Rule of Tumb: Parent elements in the tree that end in `_view` or `_entity` or have a folder icon will have been deleted after the refactor. Parent elements to which this does not apply and child elements will have lost the reference to the element to be deleted (which can be either the context for SYSTEM_APPLICATION_NEON; or the entity or view for its child elements).

6.5. Process



There are multiple reasons, why you should write code in a process instead of in the Context, e.g., reusability, server processes, etc. The processes are stored in sub-folders, according to purpose.

To create a new process, right-click on the folder "process" and choose "New" from the context menu, enter a suitable name, and click "OK". At first, the process will appear at the same level as the sub-folders. Via property "variants", you can assign your new process to one of 4 variants.



If you check a checkbox, the new process is immediately moved to the corresponding folder.

The purpose of these variants is:

- authentication: All processes responsible for authentication - see the ADITO document AID032 Authentication Methods

- executables: All processes which are used as an executeable, e.g. importer or other server processes.
- libraries: All standard and customer-specific libraries, which include methods for general purpose (e.g., Sql_lib) or for specific Contexts (e.g., Person_lib).
- workflow: All processes related to workflows - see the ADITO document AID110 Workflow Management

A fifth variant is "webservices". In this folder, there are all processes that have property "publishAsWebservice" set to true. Find more information about how to handle web services in the ADITO document AID059 Web services with ADITO.

The sixth variant, "internal", contains all processes which are provided by the ADITO platform ("core"), because they are needed for internal purposes, e.g., autostartNeon, blobHandler, or ctiCall. You cannot add your own processes in this folder, but there are multiple processes with names in grey font; if you need them, you can double-click on them or right-click and choose "Create Model". Then you can set their properties and insert code.

Processes can appear in multiple folders. E.g., the REST web service "mosaico_rest" appears not only in folder "webservices" (because its property "publishAsWebservice" is set to true), but also in the folder "executables" (because this variant is set in property "variants").

If you add new processes, you should name them according to the naming conventions, which you can learn from the names of the existing xRM processes. Libraries, e.g., are named in CamelCase, with the suffix "_lib"; REST webservices are named in camelCase, with the suffix "_rest", etc.

The processes included in the ADITO xRM project should never be renamed, and their content should only be modified in exceptional cases. If you need additional, customer-specific methods, you should store them in new, customer-specific processes, e.g., MyNewContext_lib or KeywordRegistry_custom. In particulary, this separation will help you to avoid merge conflicts when updating your project to a new version of the xRM project.

6.6. test

Node "test" include test files, which contain jDito code to test parts of the ADITO project code. The files can be executed via the context menu option "Execute tests".

6.7. report

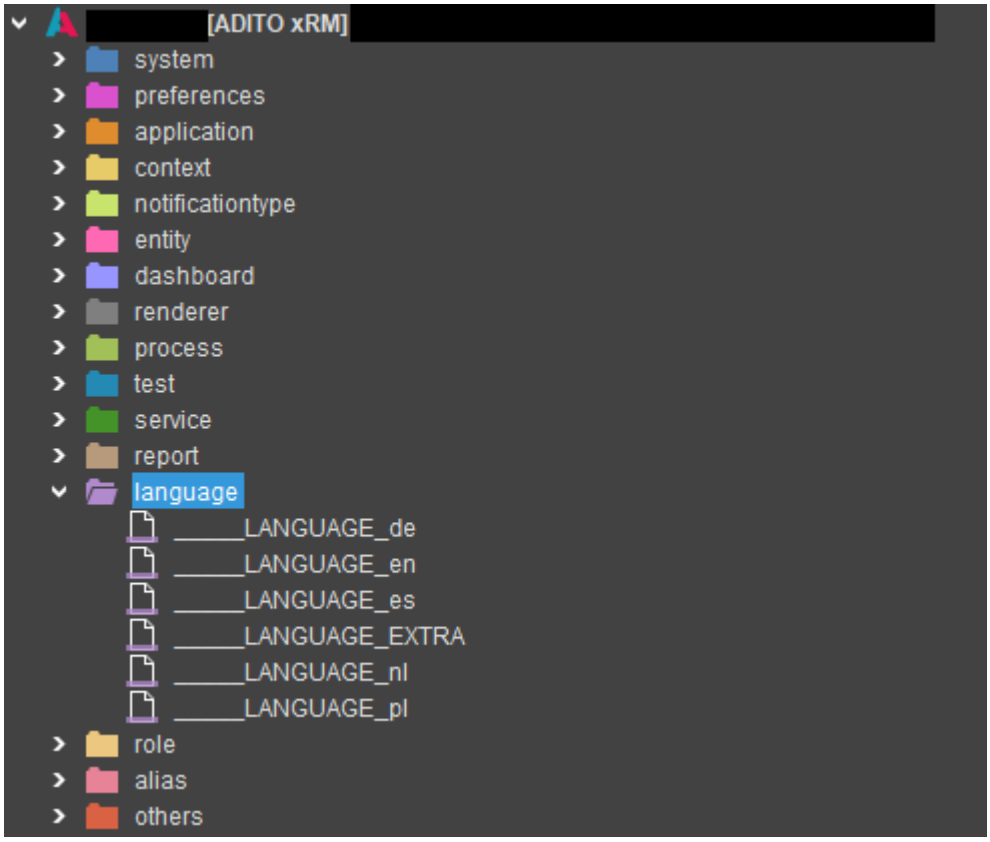
All report files in ADITO are based on JasperReports.

For further information, please refer to

- the ADITO Reporting Manual
- the documentation provided by the manufacturer of JasperReports.

6.8. Internationalization

ADITO is ready to be used in any international context. Most of the web client's elements can be shown in any language. This requires language-related settings in the project tree, under node "language":



In addition to the following sub-chapters, we recommend you to read also the complete chapter "Internationalization" of the Customizing Manual, where you will find additional information.

6.8.1. Basics

By default, the ADITO xRM project comes with English and German language settings. By right-clicking on the "language" node and choosing option "New", you can add settings for further languages.

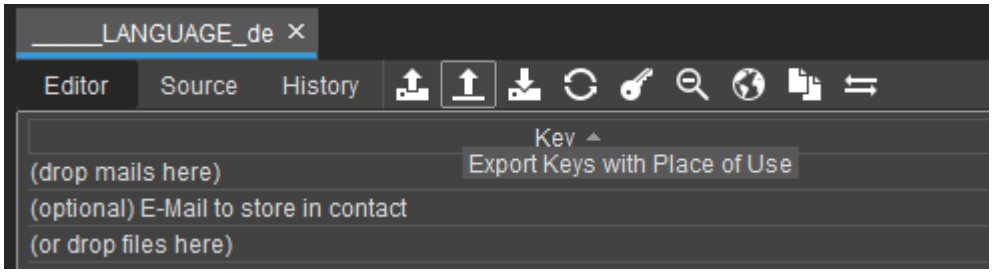


Figure 2. Language editor with settings of German language translation

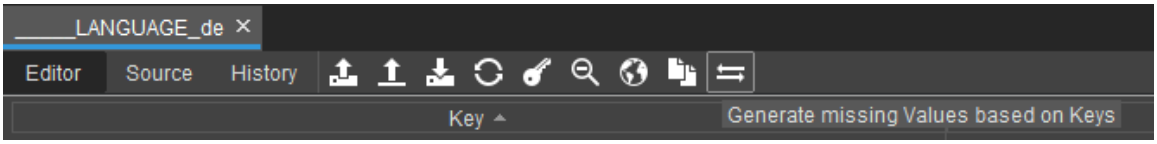
Languages in ADITO are always represented as tables, who are embedded in a language editor. In this table, you can enter the translations for the corresponding keys. Via a context menu you can create a new line, edit a line or delete an entry. Furthermore, the function bar above the table provides you with additional options, e.g. the automatic extraction of language keys from the project (button "Extract Keys").

1. **Languages and regions**
   A translation in ADITO is always divided into the language (language, e.g., en) and region (US). A language key is built from these parts and is also displayed in the Language Editor, e.g., LANGUAGE_en_US.
   Translations can be specified in ADITO either down to the region (see LANGUAGE_en_US) or for one language family (LANGUAGE_en).
2. **Fallback to other languages or translation keys**
   If a particular translation (e.g., LANGUAGE_en_US) does not exist for a language family (e.g., LANGUAGE_en), the special language falls back to the translations of the language family, if available (e.g., LANGUAGE_en_US falls back to LANGUAGE_en).
   However, if both the translation settings of the language (e.g., LANGUAGE_en_US) and the language family (e.g., LANGUAGE_en) are present and a special translation key is translated in the language family, but not in the language, the fallback to the actual translation is not to the language family, but only to translating keys.
   Example: Translation key "bonnet", English "Bonnet"; in the American translation, the key is not registered. An ADITO client then displays "hood". Only when you enter the corresponding term ("Hood") in the American translation, the term is displayed translated accordingly.

6.8.2. Special functions

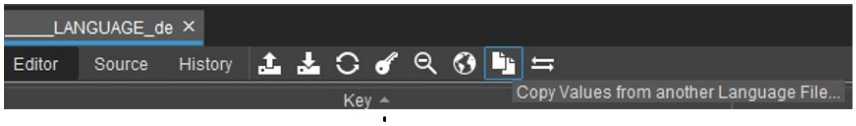
The following functions may help you to configure the language files more smoothly:

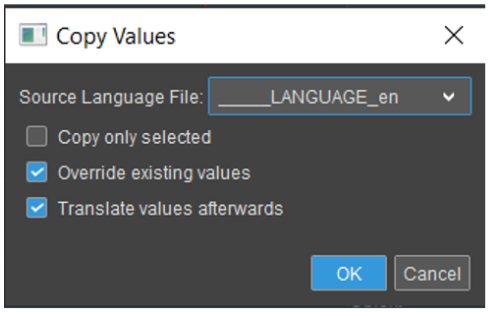
- Generate missing values based on keys

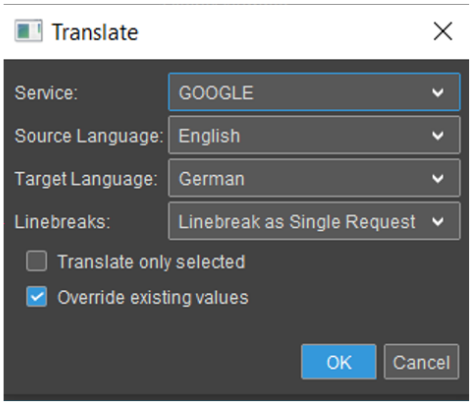


With this function, you can fill all empty values in column "Values" with the existing keys in column "Keys". Existing values will not be overwritten. This simplifies the configuration in cases, when key and value are identical (e.g., in the English language file).

- Copy values from another language file







If a new language is configured and maintained, the user can copy values from another language file (and, optionally, translate them subsequently). The relevant languages are already preselected in the translation dialog.

6.8.3. Automated translation

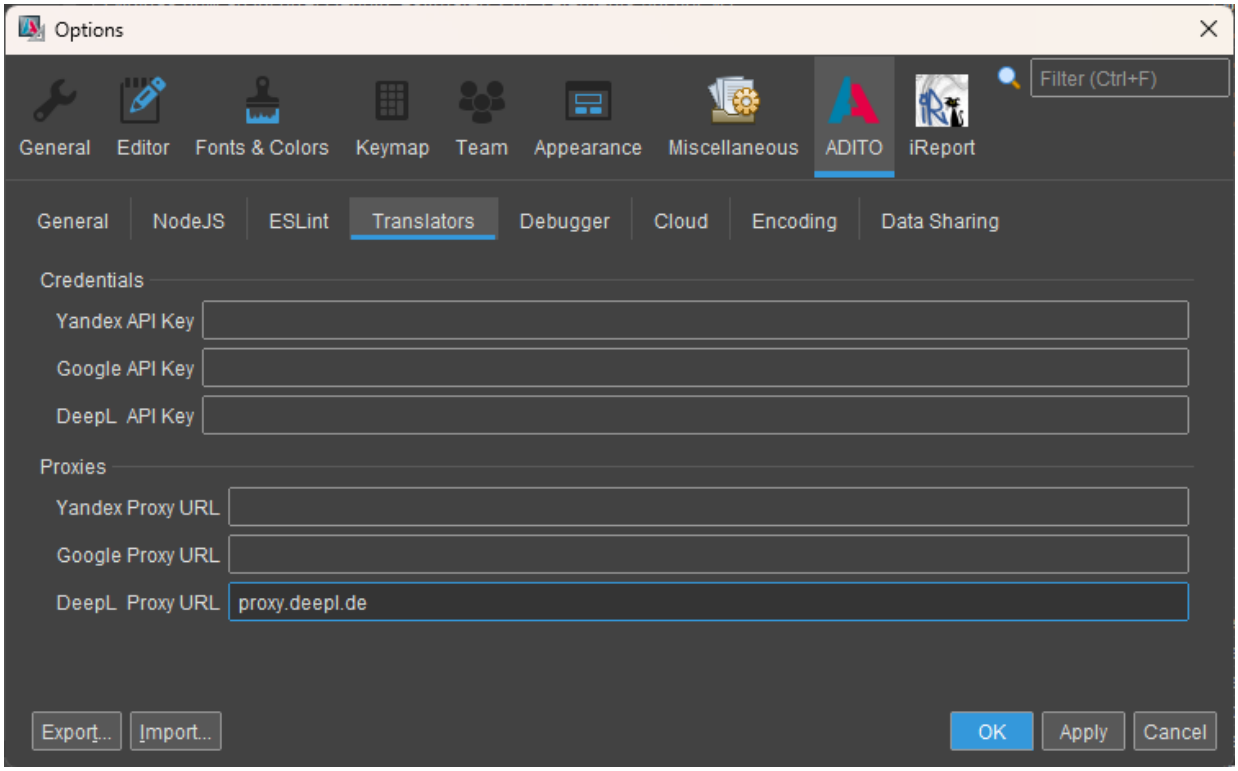
The ADITO Designer has interfaces to automated translation providers, which are realized via web services. The automatism exclusively translates the **values** in column "Value", NOT (as in earlier versions) the keys. (The latter led to problems, as "technical" keys, such as "KEY_PERSON_SALUTATION" cannot be translated.)

The following providers are supported:

1. Yandex Translate API, see https://tech.yandex.com/translate/
2. Google Cloud Translation API, see https://cloud.google.com/translate/
3. DeepL GmbH, see https://www.deepl.com/pro

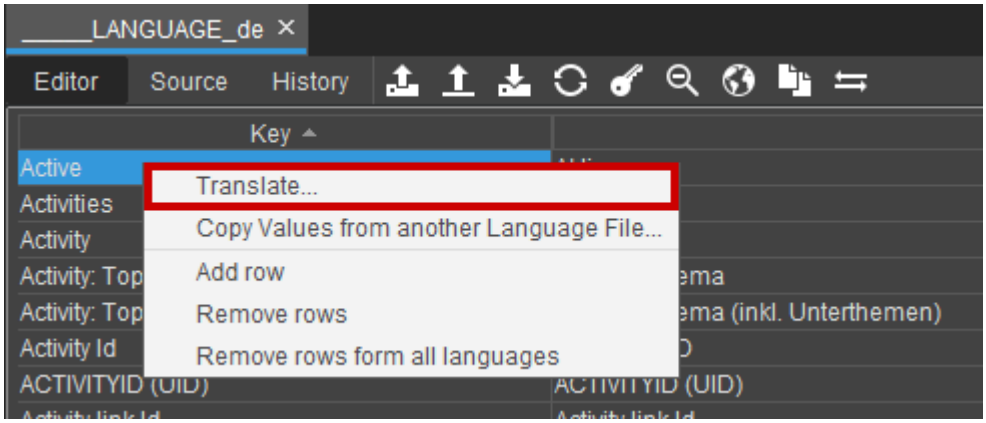
The API keys are not included in ADITO. You need to obtain a license directly from the providers. Some of them offer free licenses or limited test licenses, under certain conditions. (Please refer to the above web sites.)

The corresponding API key must be stored under Tools > Options > ADITO > Translators. In this tab, you can also set a proxy, if required.

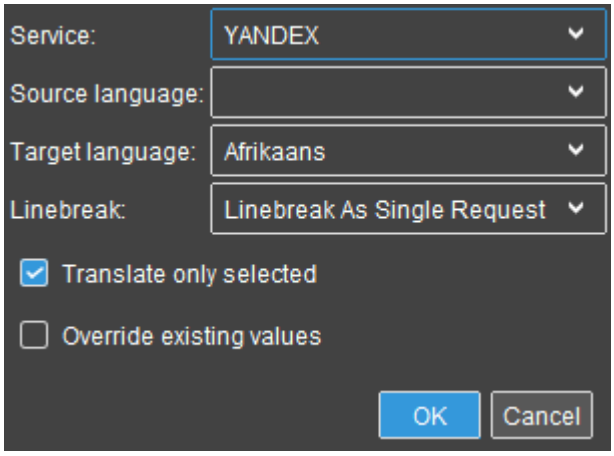


As soon as the API key is stored and (if required) the correct proxy is set, the "Translate all" button will work.

Alternatively, single terms can be translated via right-clicking on them and choosing "Translate" from the context menu.



In both cases, the following dialog is opened:

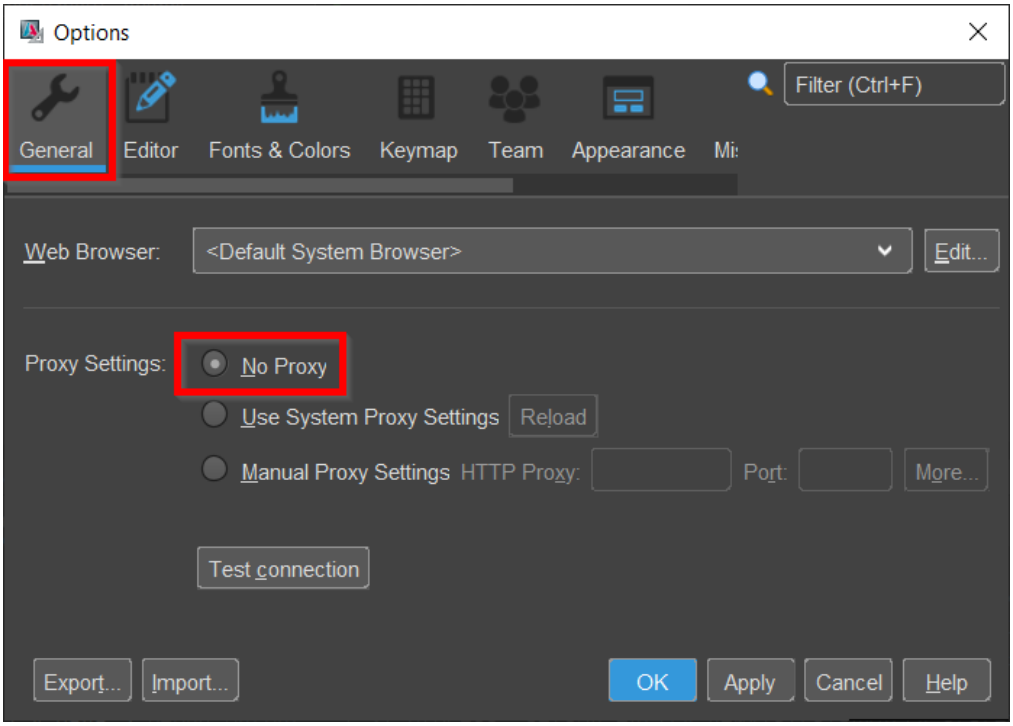


The settings are self-explanatory, except for the line break:

- Linebreak To Space: Line break is passed as a space.
- Linebreak As Single Request: each line is transferred individually (default).
- Disabled: String is passed as deposited.

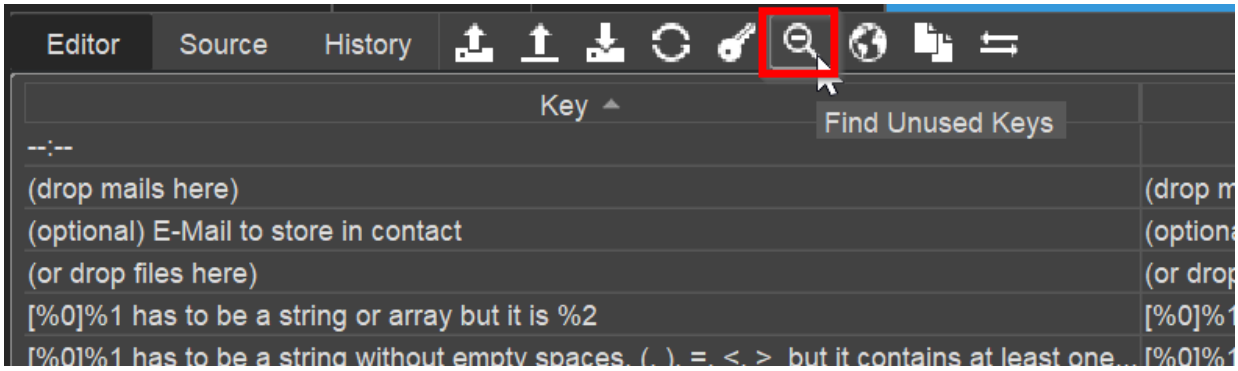
If you want to use DeepL, please note:

- Usually, you do not need a proxy. On the basis of the key, ADITO automatically detects if the pro API or the free API must be called.
- Make sure that the web API is accessible, e.g., by executing the following test URL: https://api-free.deepl.com/v2/translate?auth_key=<API-Key>&text=HelloWorld&target_lang=DE (insert your API key accordingly before executing)
- Some IT environments require you to deactivate the proxy in the Designer options (Tools > Options > General > "No Proxy"):

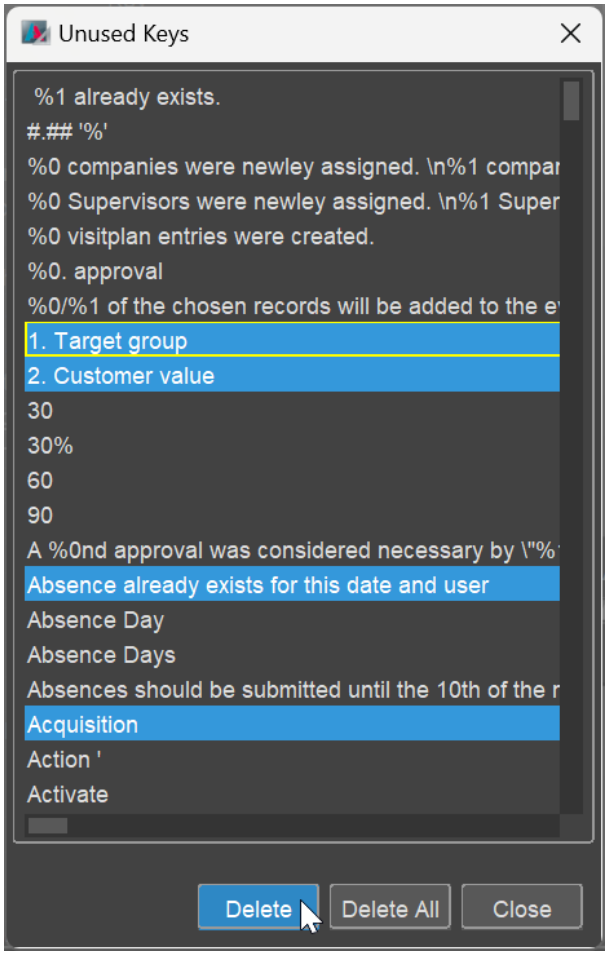


6.8.4. Find unused keys

In the development process, language files can get bigger than required, if they include unused keys. With button "Find unused keys" you can trigger a function that checks which of the keys are not used in the project:



No matter which language file you have opened to call this function, *all* language files will be scanned for unused keys. The result is shown in a dialog that enables you to remove all unused keys in one step or to select and remove only some of them (to select more than one, press CTRL while clicking on the respective keys).



All keys that you delete will be deleted in *all* language files, along with their values.

The scan that searches for unused keys works as follows:

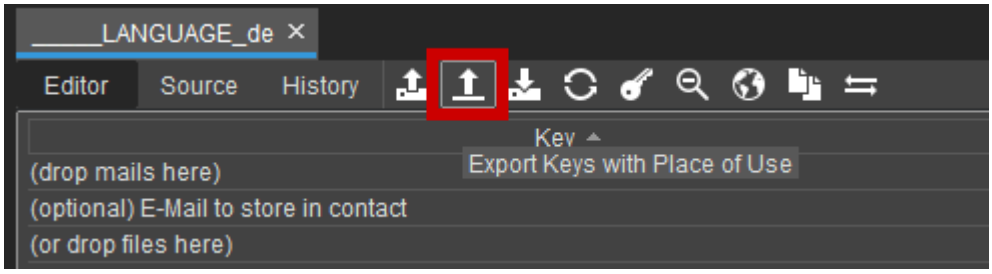
The search is for all keys that are included in the language files, but cannot be found in the project. There are 3 ways how the project is scanned:

1. Execution of SQL in order to read entries from the database (see project folder "language" > _____LANGUAGE_EXTRA > property "sqlModels" > 3-dot button > mark "Data_alias")
2. Static code analysis: Search for strings of the "translate.xxx" methods, e.g.
    - `translate.text("This will automatically be included in the translation.");`
    - `var msg = "This will *not* automatically be included in the translation."`
      `translate.text(msg);`
3. Read specific properties: These are all properties that need to be shown translated in the client, e.g., the property "title".

Keys that are not found/no longer needed can be removed from the language files via the subsequent dialog.

6.8.5. Export Keys with Place of Use

With 2022.2.1 you can export keys with the place of use via the button between im- and export.



After the dialog to select your project and system you see a progressbar. This process will define the place where sqlModels/Alias/DB, Datamodels and Jasper Reports are used. After finishing this process you can set the parameter for the generated CSV file and the place to store the file.

For the keys from the alias a database must be available. In addition, a query for determining the keys must be available in _____LANGUAGE_EXTRA in sqlModels. With this query applies:

- At least one, at most two columns are assumed.
- In the first column the key must be returned.
- In the second column the usage location is returned.
- If the second column is not filled (because it is either not there, or is null or empty), then only Database is written to the usage location in the CSV.

Example for the place of use:

- Attribute_entity/entityFields/ATTRIBUTE_ACTIVE/title
- TopicTree_lib/process
- FacebookTimeline_view/dashletConfigurations/AditoFacebookDashlet/title

For the keys from the Jasper, the folder where the Jasper file is located + the name of the Jasper file is exported, e.g. Organization_report/reportData.jrxml

One row in the CSV will then have exactly 3 columns.

- The key (exactly as in the normal export)
- The value in the respective language (exactly as in the normal export)

- The usage locations, separated with comma

6.8.6. User help

The "user help" are the texts and illustrations provided to client users via the "questionmark" icons. These texts are also submitted to internationalization, i.e., you can provide them in various langugages.

In ADITO document AID005_Userhelp.pdf, you can find more information of how to implement and maintain the "user help" via the ADITO Designer.

In the xRM project, this functionality is included, e.g., as "Context help":

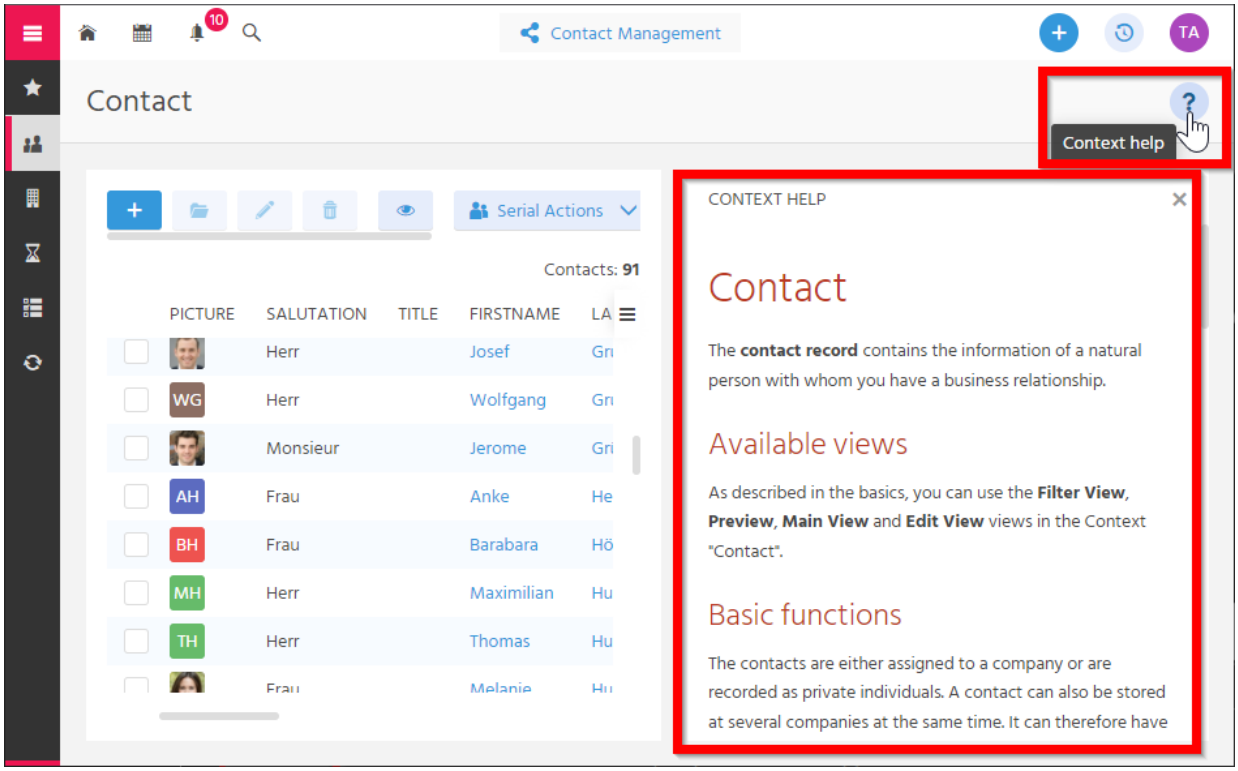
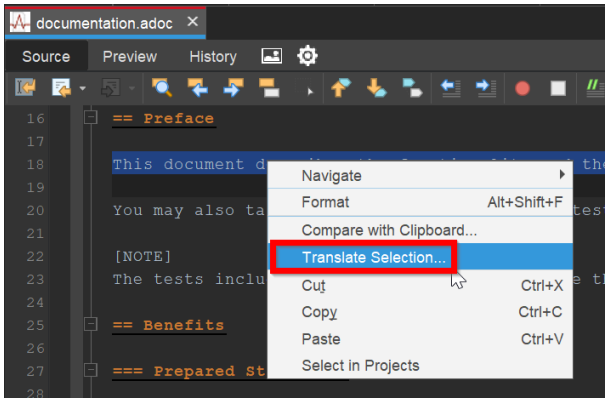


Figure 3. Example of facilitating the "user help" functionality

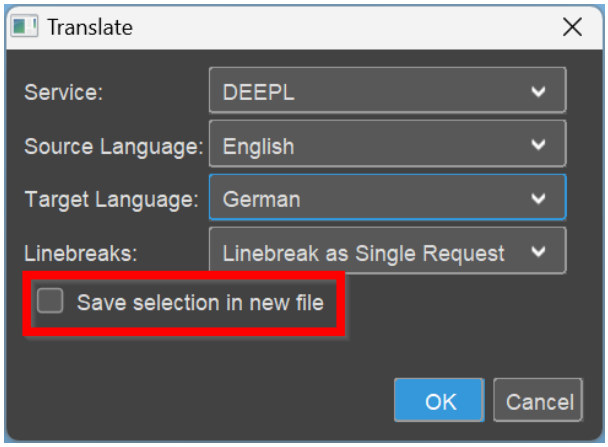
6.8.7. Translation of source code

Given that a valid API key is used (e.g., for DeepL, see above), you have the option to automatically translate selected source code (adoc, js, xml, and aod files). This can be, e.g., a helpful function when writing a User help.

Proceed as follows:
Open the code in the code editor, mark the part you want to translate, then right-click and choose "Translate selection..." from the context menu.



This option opens the translation dialog that you know from the translation of language files (see above). Mind the checkbox in the lower part of the dialog: Here, you can decide if the translated text should overwrite the selection, or if it should be saved in a separate file:



If you have chosen "Save selection in new file", a file browser will open subsequently to let you choose where and with which name the file is to be saved.

6.9. Roles

ADITO users (appearing as "Employees" in the Web Client) are always equipped with roles. There are two types of roles:

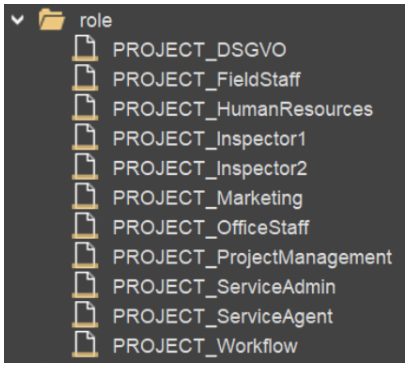
- Internal roles
- Project roles

6.9.1. Internal roles

Internal roles included in the ADITO platform ("core"), not in an ADITO project. They can only be assigned (e.g., to control what Contexts are shown in the Global Menu) but not changed. You can view all internal roles, if you navigate to neon > application > ___*SYSTEM_APPLICATION_NEON and then check "Role" in the Navigator window. Then, the Editor window's content is reduced to roles, with the*

internal roles' names starting with the prefix "INTERNAL".

6.9.2. Project roles

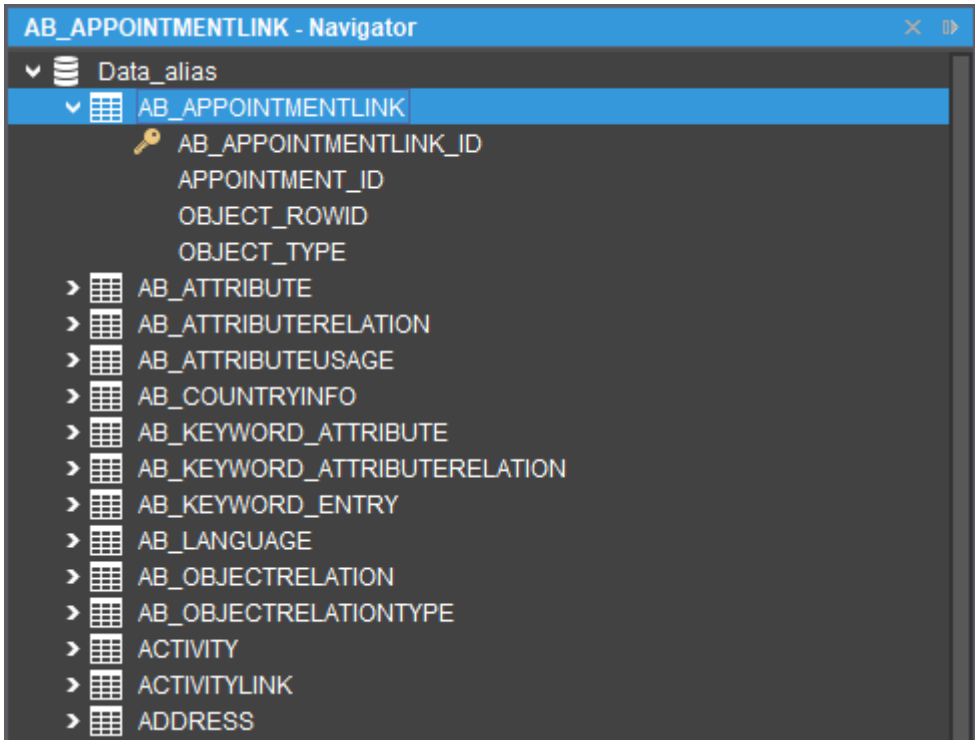


Project roles can be used to give specific rights to an user, e.g. ability to see certain data and/or editing it. These are heavily dependent on the intended use. For example, a role can be created for each department and users can be assigned to the department.

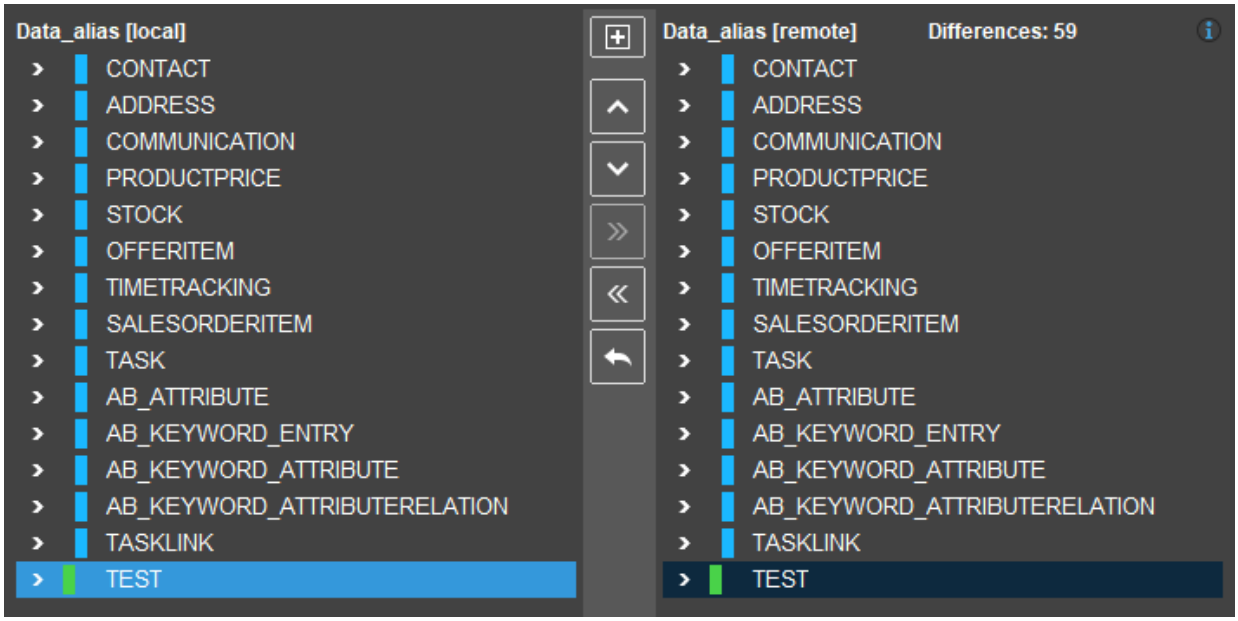
To create a new role right click "roles" and choose "new" - type = role and enter the name of the new role.

6.10. alias

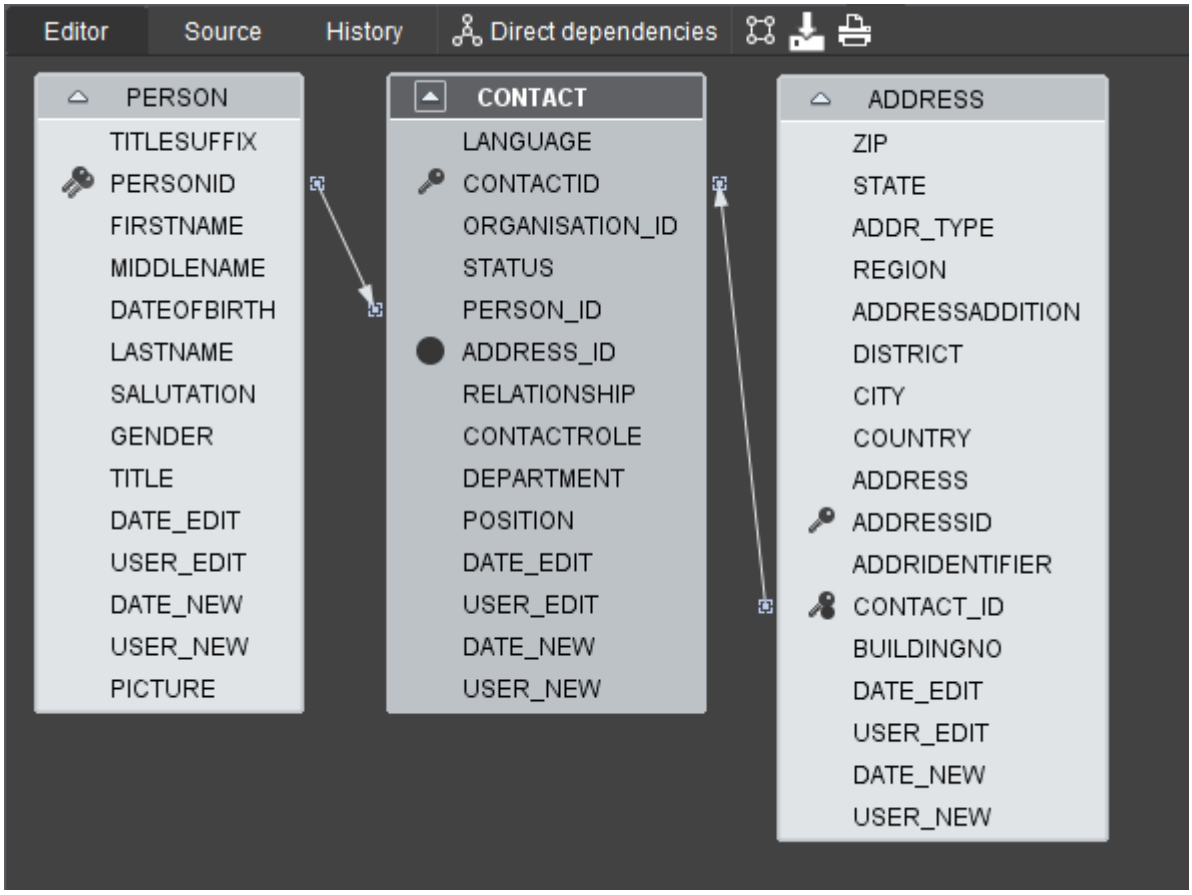
Node "alias" contains alias models, named according to the name of the database alias they are refering to. If you double-click on an alias model, the so-called "Alias Definition" is shown in the Navigator window. It is a junction layer between the database and the project and includes only the **structure** of the database, i.e., the names and configurations of the database's tables and their columns. It does *not* include datasets. The "Alias Definition" is also refered to as "The database structure of the project".



To synchronise the Alias Definition with the databas tables, right-click on the alias name (root node in the "Navigator" window) and select "Diff Alias Definition <> Database tables". Likewise, choose "Diff Alias Definition <> Platform", if you want to update the Alias Defintion with the table definitions hard-coded in the ADITO platform.



The Designer will compare the Alias Definition ("[local]")" with the database ("[remote]") and mark all differences (changes). These differences may be synchronized in either direction: Just click on the respective double-arrow button and then on "OK".



The icons to the left of a table column give additional information:

Key	Is either a primary key or a foreign key (if a dependency has been established).
Ball	Is part of the index.

The buttons at the top have the following functionality:

Direct dependencies	Toggles between showing only direct or also indirect dependencies of the selected table.
	Arrange the respective tables in a standardized way, with the dependency arrows drawn rectangularly.
	Export the selection, as ERD diagram (png file) or as well-structured text file in "Asciidoctor" format (adoc). See illustrations below.

	Directly print the selection.

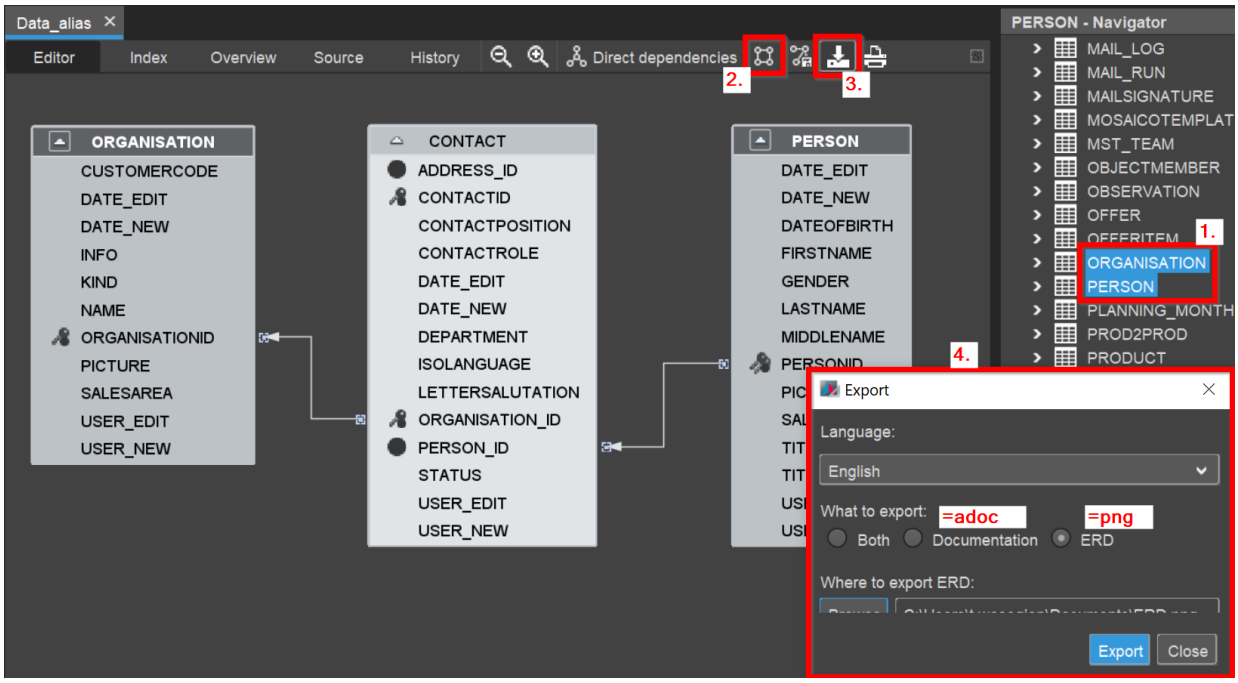


Figure 4. Example of how to export selected tables of the Alias Definition, as ERD diagram or as well-structured text file (adoc format)

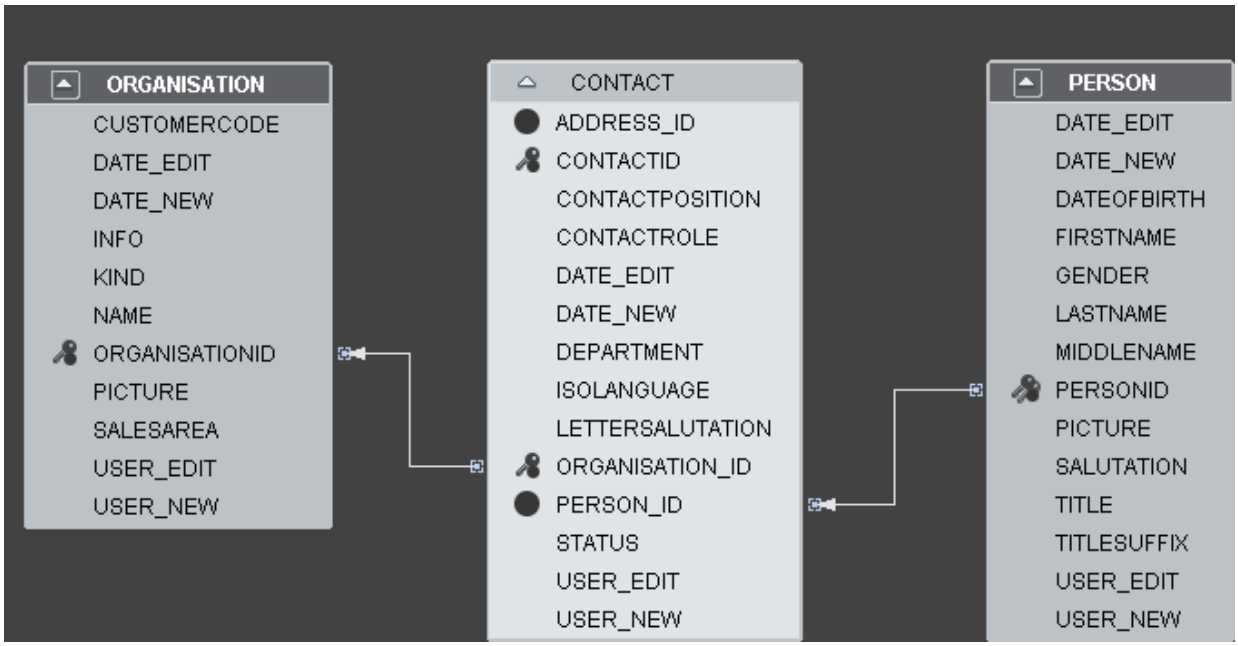


Figure 5. Export result: ERD diagram (png file)

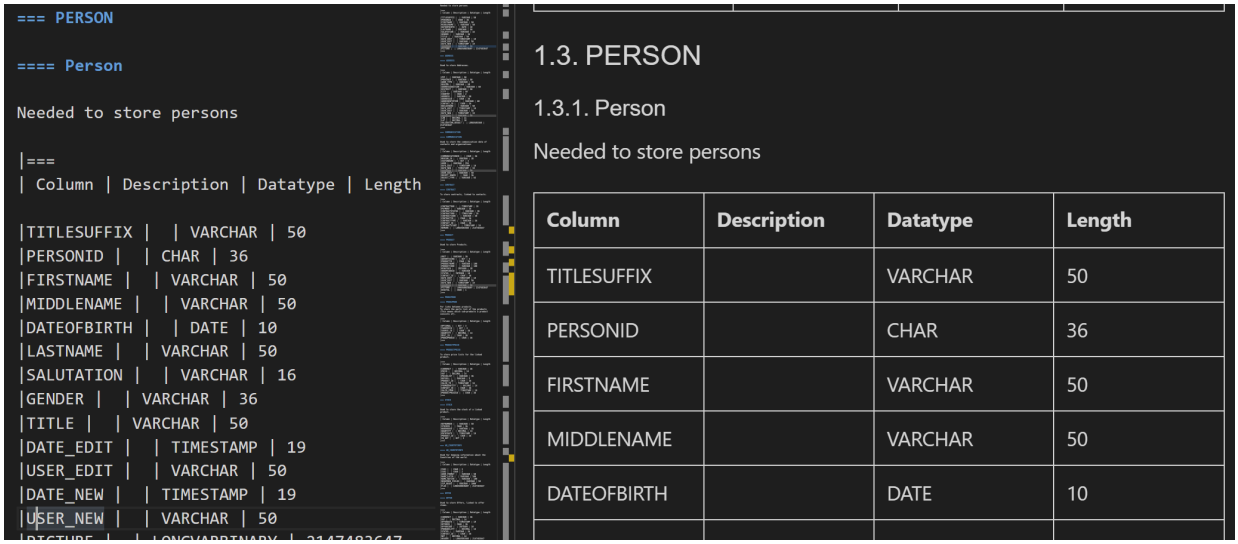
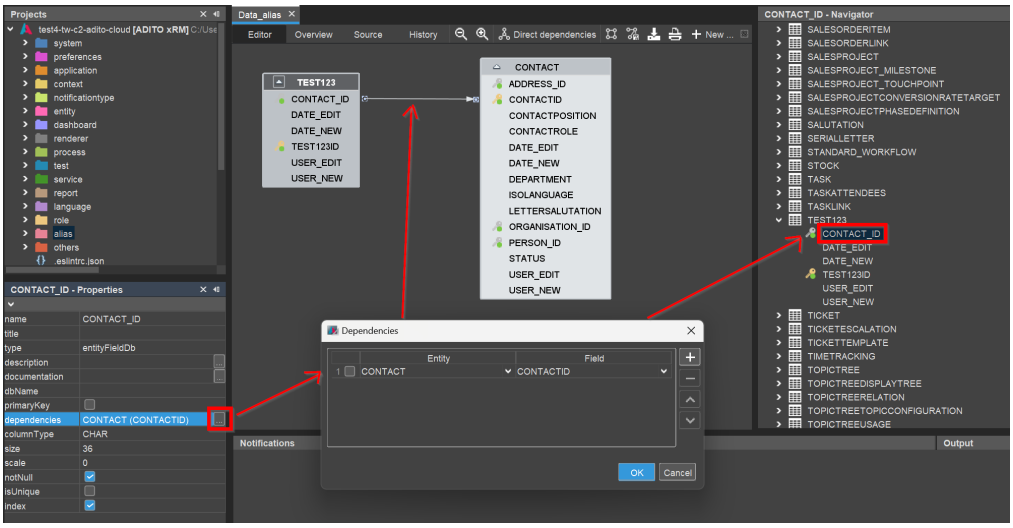


Figure 6. Export result: adoc file (to be viewed, e.g., with application "Visual Studio Code")

For technical reasons (e.g., because of asynchronous saving), it is unusual to configure foreign keys in ADITO data databases. Therefore, you need to maintain the dependencies shown in the ERD (as arrows between primary keys and factual "foreign keys") manually.

The dependencies are configured on the foreign key side of the relationship: Click on the column working as foreign key (e.g., CONTACT_ID) and enter the corresponding table and its primary key in property "dependencies". After a restart of the Designer, this table will appear additionaly in the ERD, connected to the other table with an arrow between primary key and foreign key. Furthermore, the foreign key will be marked with a key symbol to the left of it.

Note that this configuration of the Alias Definition does *not* influence the database table's configuration; it still has no technical foreign key.



6.11. Others

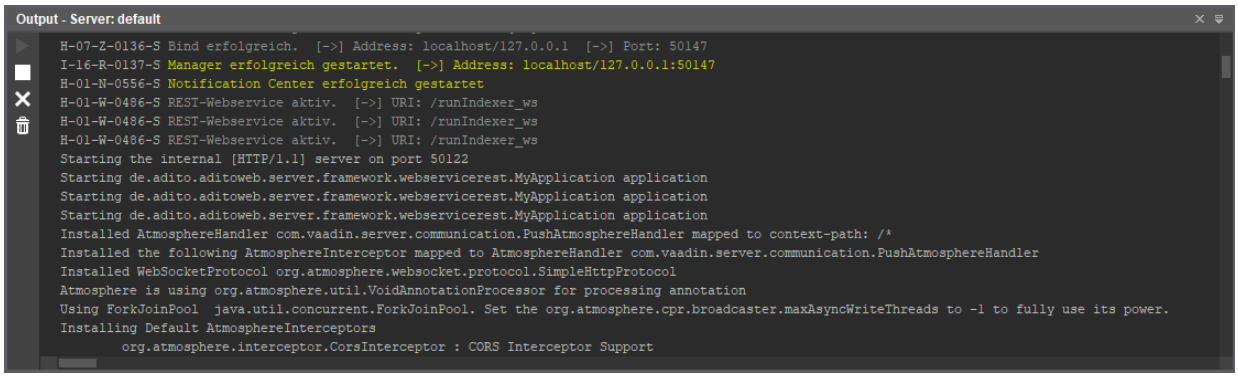
In folder "others", there are, e.g., files related to the "Cypress" test framework.

Furthermore, you may create folders and add files directly in your project’s file structure, via a file explorer tool (e.g., the Windows Explorer); it will then be visible and (depending on the file type) also be editable in the Designer.

Some Designer users utilize the folder "others" for, e.g., keeping specification files of their customizing tasks or log files for documenting changes they have done in the database.

7. Output Window

When a server or client is started using the Designer, the output window opens automatically.



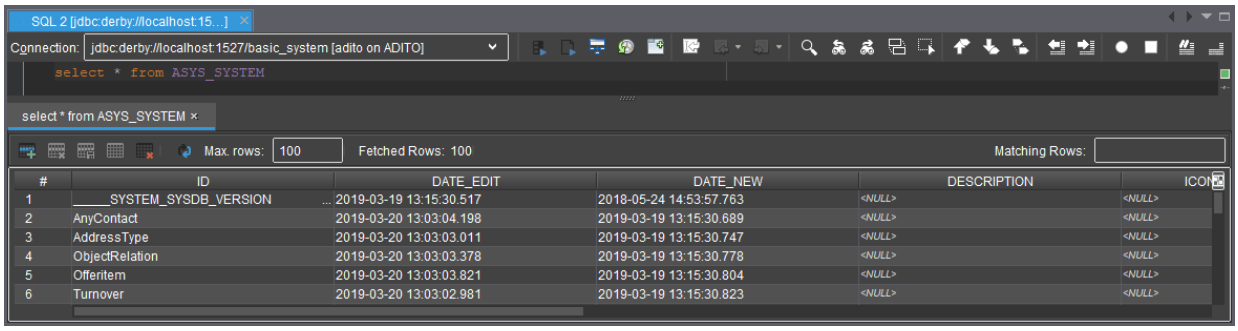
The specific server instance can also be seen in the window header (e.g., "Server: default").

The console output of the server or client is shown in the middle of this window. There are also some actions you can use via the buttons to the left of the console output:

	Starts the selected instance when paused or shut down
	Completely shuts down the instance (graceful shutdown)
	Instantly shuts down the instance
	Clears the cache of the selected instance

8. Execute SQL

Your are able to directly execute sql statements in the Designer.
All you have to do for this feature, is opening the sql window using the following button in your toolbar:



Top Bar:

Connection	Select the database instance where the statement should be executed
	Execute all written statements
	Execute your selected/marked sql statement
	Select the connection of the services
	Shows you the history of all executed sql statements
	Keep prior tabs
	Jump to the last edit
	Take back the last edit
	forward the last edit
	Search for a string

	Selects the previous occurrence of your search
	Select the next occurrence of your search
	Toggles the option to highlight your search findings
	Toggles a visible border when selecting
	Selects the previous bookmark
	Selects the next bookmark
	Sets a bookmark in the marked line
	Shifts the line one tabulator to the left
	Shifts the line one tabulator to the right
	Starts a macro recording. Macro execution is not supported by the Designer.
	Stops the macro recording. Macro execution is not supported by the Designer.
	Comments the selected line
	Uncomments the selected line

"select * from ASYS_SYSTEM" - In the text area below the top bar, you can write your sql statements.

After the sql statement is executed, the lower half of the picture will appear.
A tab with the results of each executed statement is shown, as well as the middle bar.

Middle Bar:

	Adds a new row, where you are able to insert the values
	Deletes the selected row
	Commit your applied edits
	Cancel your applied edits
	Deletes the table without protocolling it (See SQL truncate statement)
	Refreshes the output, when applying changes
Max. rows	Select the maximal amount of rows which should be displayed
Fetched Rows	Shows the amount of rows displayed (When may Rows is high enough all rows)
Matching Rows	Search in all rows and all columns after a string

If you commit your changes, those will be committed to the database and can't be rolled back.
Only use this, if your are really sure, not to destroy any production data!

9. Debugger

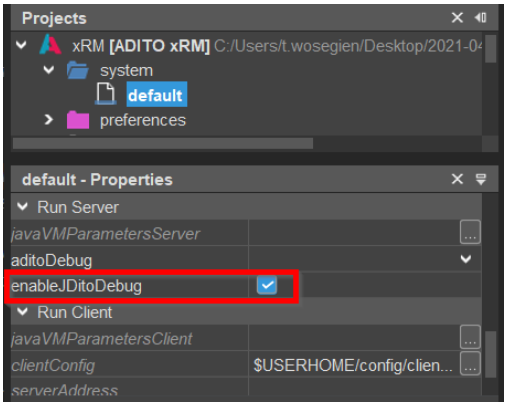
The debugger enables you to interrupt the execution of code at specific code lines, in order to obtain various information, such as the value of a specific variable.

As for modularized systems with versions from 2025.0.0, it is only possible to select a project as the "debugged" project in the Designer and not individual modules, as they are not independently executable projects.

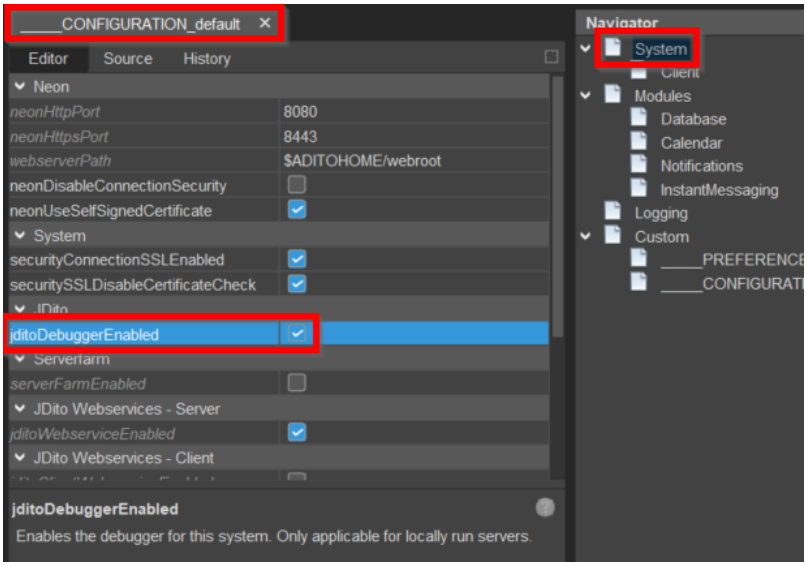
9.1. Preparations

Prerequisites for using the debugger with managed ADITO cloud systems are:

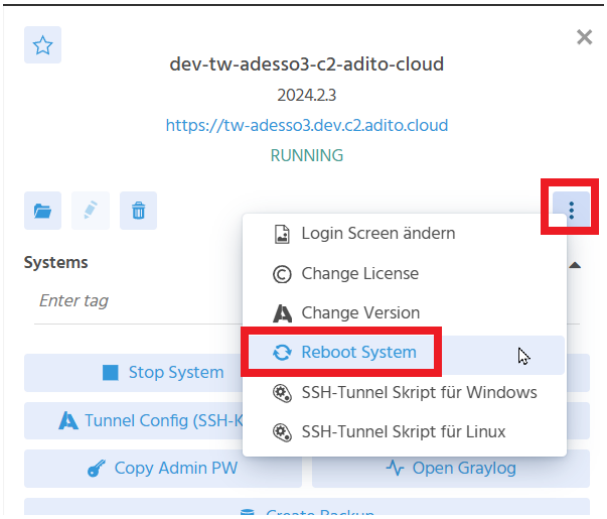
- System property "enableJDitoDebug" is set to true:



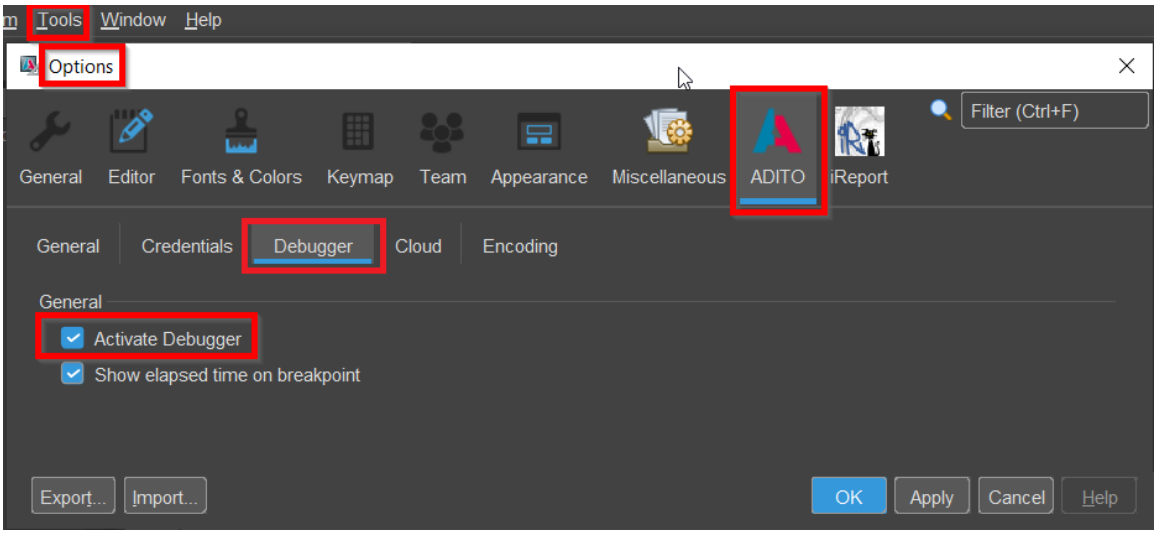
- The instance configuration's property "jditoDebuggerEnabled" is set to true:



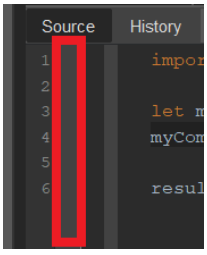
- Reboot the ADITO server and wait until it is accessible again in the client:



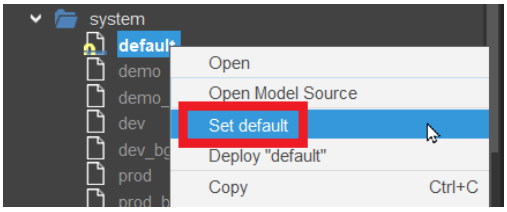
- Activate the debugger under Tools > Options > ADITO > Debugger:



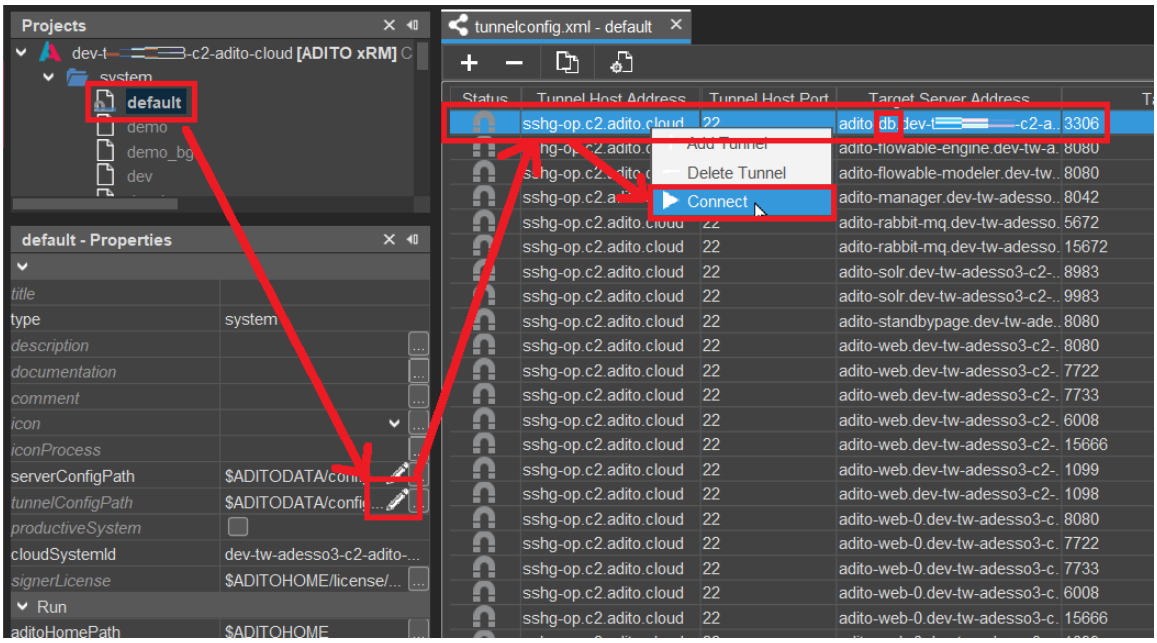
- As a result, you will see an additional vertical "breakpoint bar" to the left of each code windows (if you do not see this bar, simply close the code window and re-open it):



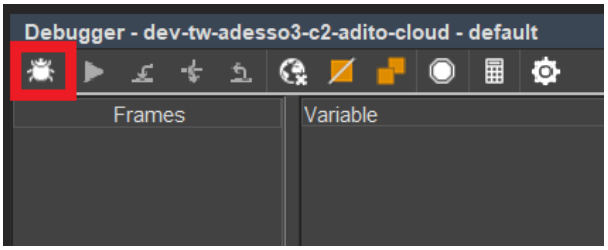
- Set the system you want to debug as default:



- Open the database (db) tunnel (cf. appendix Tunneling):



- Make sure that all local changes are deployed.
- Add (if not already opened) the "Debugger" window, via the menu item "Window" > "Debugger".
- In the "Debugger" window, press the little button with the bug icon:

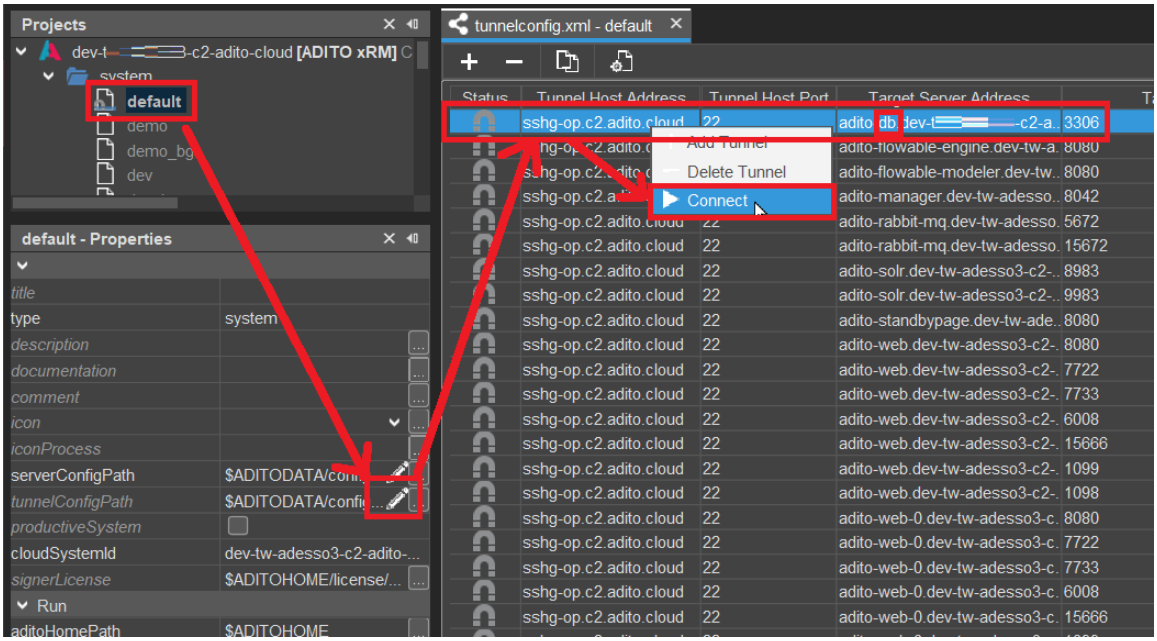


This button will start and connect the debugger.

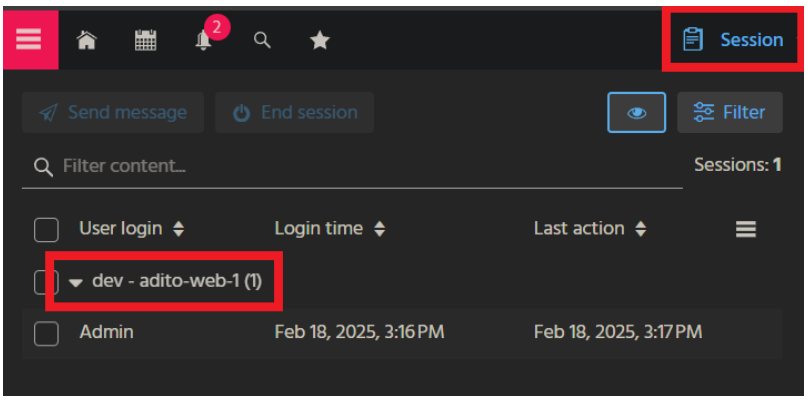
If the debugger throws an error message (e.g., "Connection refused"), immediatly after pressing the button with the bug icon, proceed as follows:

1. Restart your PC and your Designer.

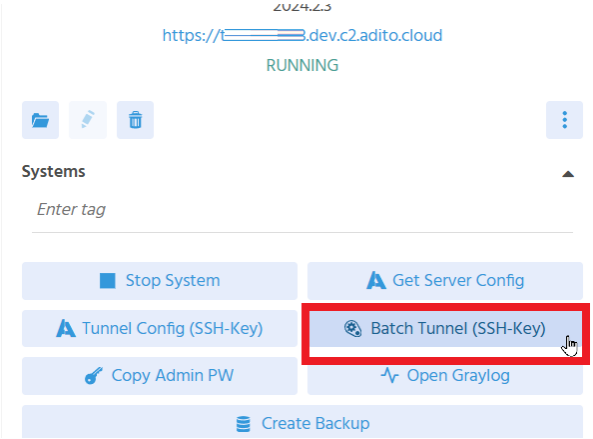
2. Do not open all tunnels, but connect only the database (db) tunnel:



3. Find out the web pod you are currently logged into, by using Context "Session" of menu group "Manager" in your ADITO client. There, you can see where your user session is active. Remember it:



4. Download the batch tunnel from your SSP system...



5. … and open it in a text editor, such as Notepad++. Remove all tunnel entries except those for the two ports 1098 and 1099 of the web pod you are currently logged into (e.g., `adito-web-1`). Then, replace the port numbers at the beginning of these entries with 1098 or 1099. Here is an example:

```
@echo off
set SERVER_ADR=sshg-op.c2.adito.cloud
set SERVER_PORT=22
set SERVER_USER=dev-t[redacted]-c2-adito-cloud

plink.exe ^
-C ^
-v ^
-N ^
-l %SERVER_USER% ^
-i "%USERPROFILE%\.ssh\id_rsa" ^
-P %SERVER_PORT% ^
-L 1099:adito-web-1.dev-t[redacted]-c2-adito-cloud.svc.cluster.local:1099 ^
-L 1098:adito-web-1.dev-t[redacted]-c2-adito-cloud.svc.cluster.local:1098 ^
%SERVER_ADR%
 PAUSE
```

6. Save the modified tunnel batch file and double-click it to open both tunnels. Follow the instructions displayed in the terminal.
7. Now, try again to start debugging via the button with the bug icon.

NOTE: If you want to debug again later, you will, again, first need to find out the web pod you are currently logged into and then execute the subsequent steps (see above).

9.2. Debugging options

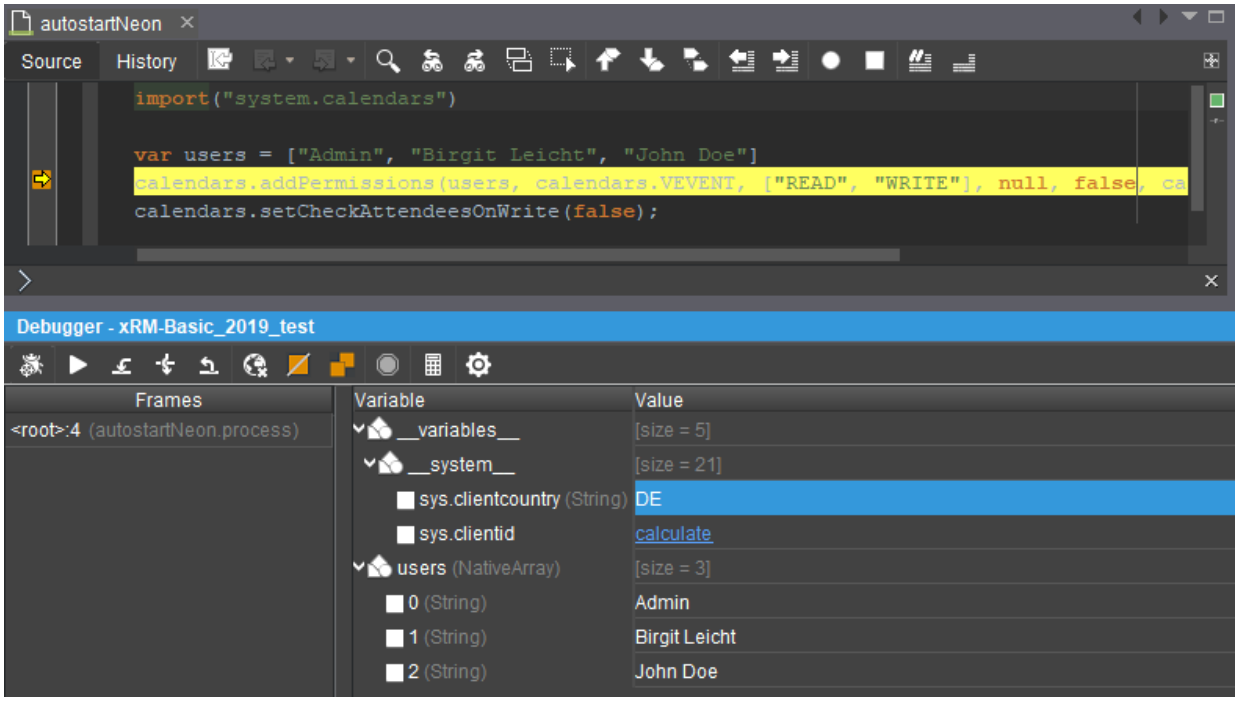
The response time of ADITO's debugger features is partly slow. This will be improved in future ADITO versions.

After you have started the debugger, set so-called breakpoints wherever you like the code execution to be stopped. To do this, click into the breakpoint bar (to the left of your code window), exactly in the height of the code line in which the execution should halt. Then, an a little orange square will appear (see the picture below). (In earlier ADITO versions, you need to click into the code window once and/or execute the code to be debugged once, before the color changes to orange.)

With a breakpoint you mark a line of code. Now, use your client to execute the code you want to debug. When the code execution reaches a breakpoint, the execution will completely stop until manually resumed via one of the respective buttons. (In earlier ADITO versions, you initially need to repeat the first code execution, before the code actually stops at the first breakpoint.)

The line of the halt is marked yellow. At the end of the line you can see (by default) the time elapsed since the start of the code execution.

A breakpoint may also be muted, then the execution will not stop, but it is easier to reactivate it later on. How to do this look further below.



On the left side in the table "Frames", the current stacktrace will be displayed (In this case the auostartNeon.process line 4). The right hand side shows all variables and their value, which exist at this point.

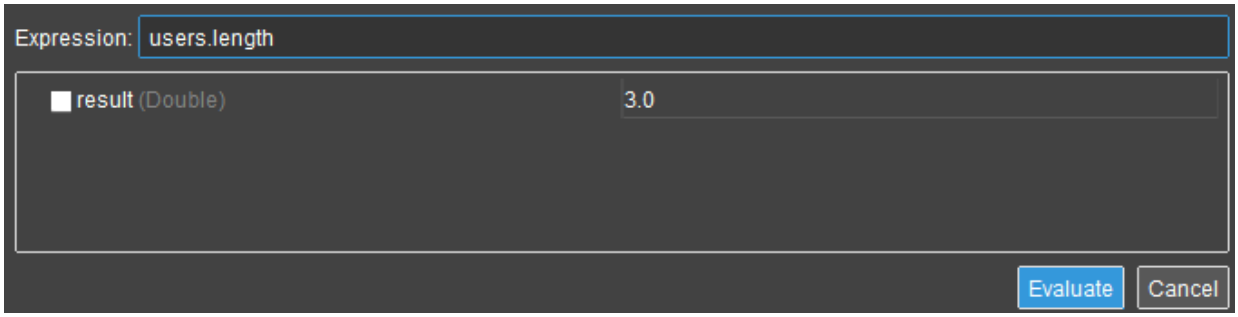
For one there are the variables, which are always in the system. These are listed under "_variables_" e.g. "_system_", others do not have values yet, as this is the first process started. The value for some variables are to complex to show immediately, in this case you have to click calculate to get a value. That was also the case for "sys.clientcountry" and "sys.clientid".

But also local variables will be shown, like e.g. "users" (array with the size 3), which gets created in line 3.

The buttons in the window:

	Manually resume the code execution after a breakpoint
	Step into a function when there is one declared while debugging line-by-line
	The debugger will step over this line, any functions will be executed but not debugged line-by-line
	Jumps to the line where the current function was called
	Remove all breakpoints in your project
	Mute all breakpoints in a project
	Shows all breakpoints in a project and their menu (see below)
	Stops the execution of the code without a breakpoint
	Opens a new window where you are able to evaluate custom expressions
	Opens the "Debugger" option in "Settings"

9.3. Evaluate Expressions



The option to evaluate a expression is available, when a breakpoint is reached. Any exprssion can be searched for. Just write it into the field "Expression", click on "Evaluate" and the results will show below.

An Expression can in principle be any valid single-line JavaScript.
Example: "var arr = []; for (var i = 0; i <10; i ++) arr.push (i); filelist = arr;" is executed and works and displays the result of the assignment at the end! The last return is always displayed.

Furthermore, one can simply use variable names to see their contents.
For example, fileList.

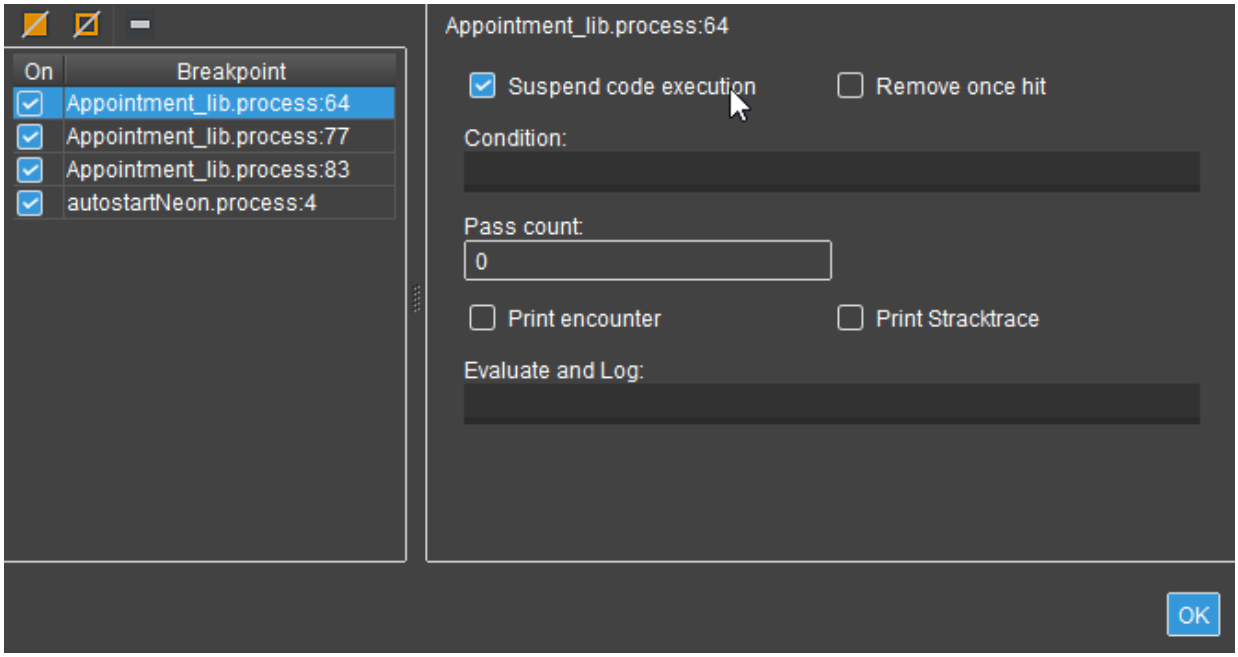
Or you can also inspect individual array / object elements if you know the right index.
For example, filelist[0] or fileObj["file.txt"]

Also JDito methods can be called.
For example, to see if a variable contains the correct date, you could use "datetime.toDate (pDate," dd.MM.yyyy HH: mm: ss ");" to temporarily convert the long value for display into a readable date. For the calls to work, the corresponding system module must be imported in the code.
Example: `import { mymodule } from "@aditosoftware/jdito-types";`
It is also sufficient if the import of the system module is imported into one of the imported libs.

You can also call functions from the code or imported libs, but it can be problematic that you only get the return value of the function call, but this may do more in the code sequence. For example, Database changes, triggering emails or actions. One should always keep in mind that a function call in many cases may have far reaching side effects!

Be careful with assignments: these are executed in the context of the current code and override the values that are in a variable at the moment of the break!

9.4. Breakpoint Menu



	Mute the selected breakpoint
	Unmute the selected breakpoint
	The debugger will step over this line, any functions will be executed but not debugged line-by-line

On the left side all breakpoints in a project will be listed, with a checkbox if they are muted and their location (name of the data model and the line).

The right side will open when you right-click on a single breakpoint.

When in the breakpoint menu for all breakpoints, the options on the right will only apply to the currently selected breakpoint

Most of the point in this menu are self explanatory. The most noteworthy are "Condition" and "Evaluate and Log".

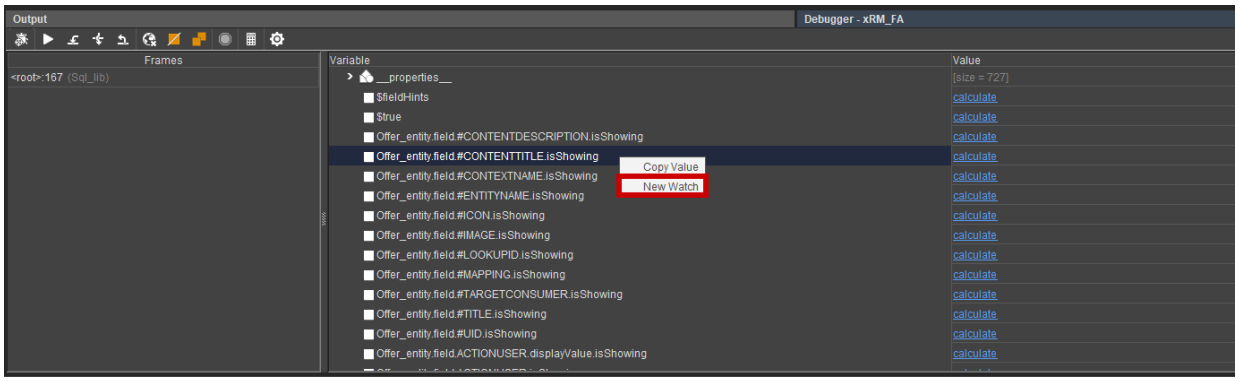
For "Condition" you can set a special condition for the breakpoint to be active. Like "array.length > 5" for example.

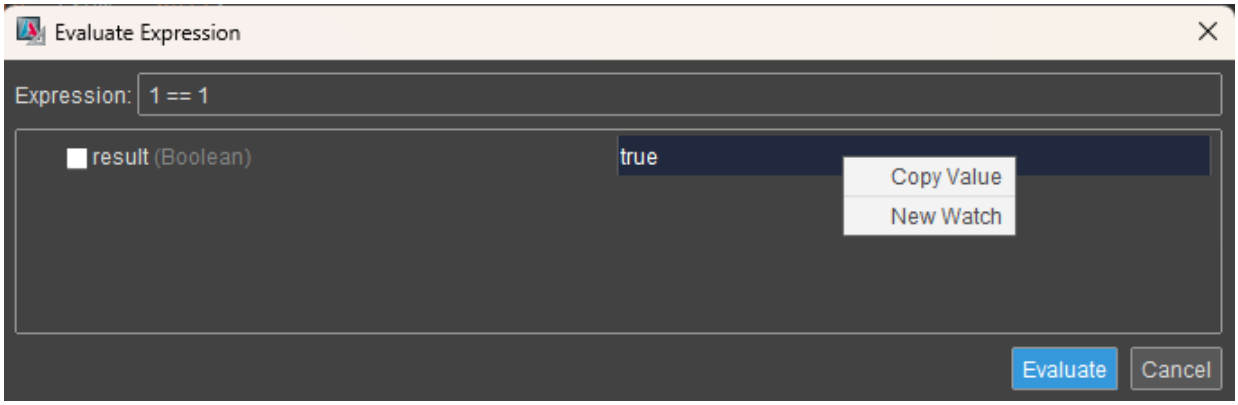
"Evaluate and Log" gives you the option to set things that should be logged.

9.5. Watches

You can set so called "watches" to permanently view the value of a field or result of an expression while you go throw the debug process. The value of these watches may change over the course of the process.

You can add watches via the right-click menu





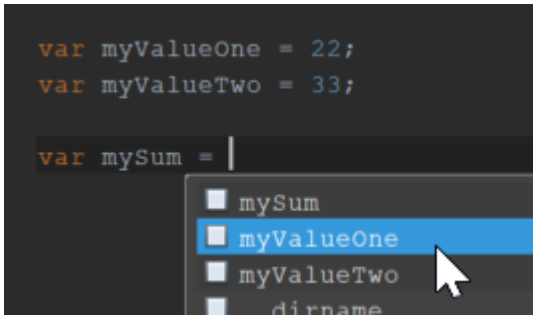
After adding watches you will see them in your "Variable"-tab in the debugger:

10. Code quality support

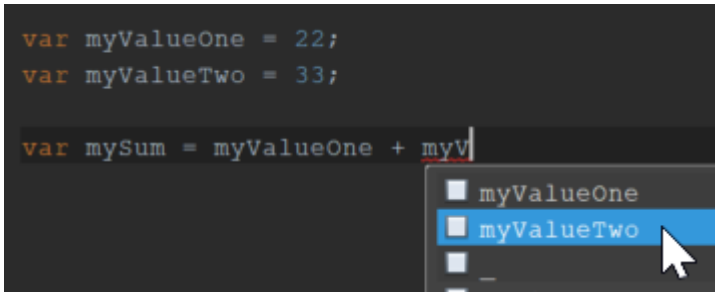
The ADITO Designer, particularly the code editor, offers a variety of functions that help you to optimize the quality of your code.

10.1. Autocompletion

Using the ADITO Designer's autocompletion functionality speeds-up coding and helps you to avoid syntax errors. If, e.g., you use a variable that you had declared earlier, you might make a typing error if you simplye re-type the variable's name. Instead, you can press CTRL-SPACE in order to open a combobox that includes all currently available variable namens - and then you can simply select from this combobox.



If you first type some characters and then press CTRL-SPACE then the combobox' content will show variables starting with these characters on top:



10.2. JSDoc

Related to the autocompletion is a documentation, the so-called JSDoc (JavaScript documentation). It appears below whenever you select an item in the autocompletion combobox (see above) and contains important information about the usage of the respective method, constant, etc. In particular, the JSDoc contains the following parts:

- general description
- parameters/arguments
- return value

- exception
- example(s)

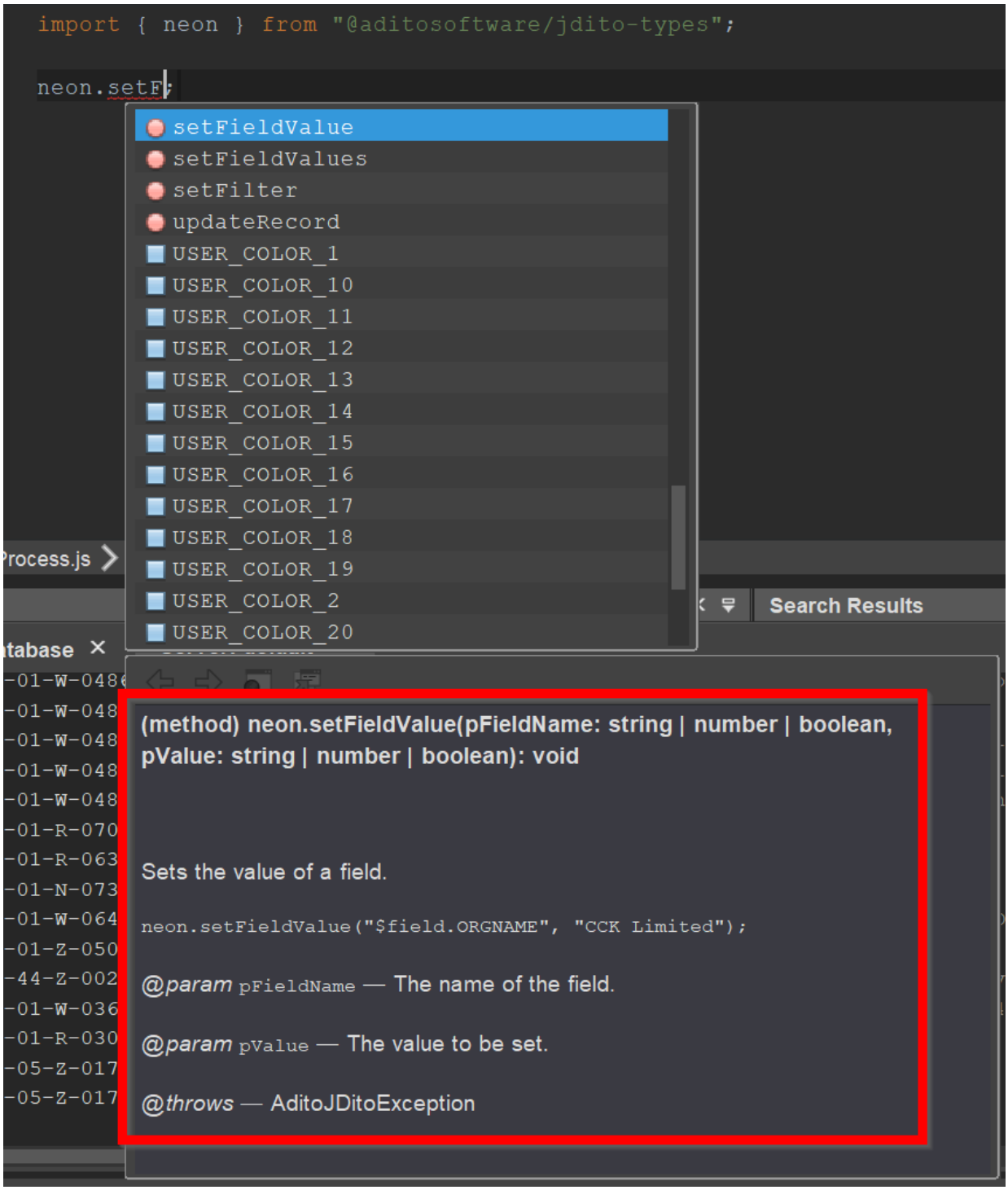
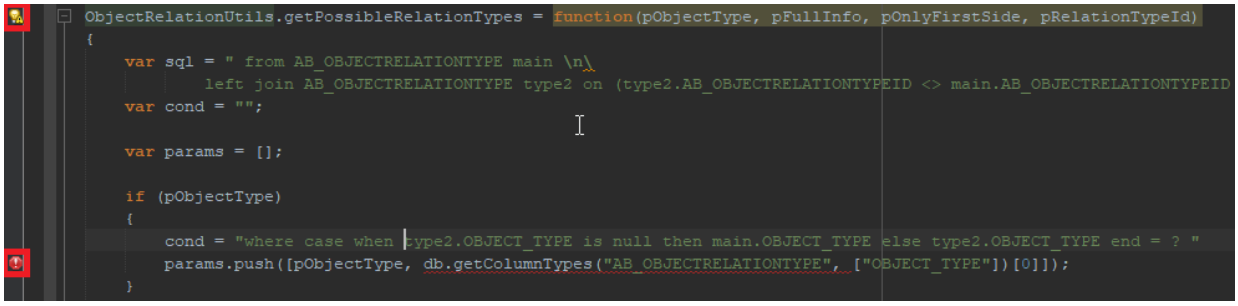


Figure 7. Example of a JSDoc

10.3. Errors and warnings

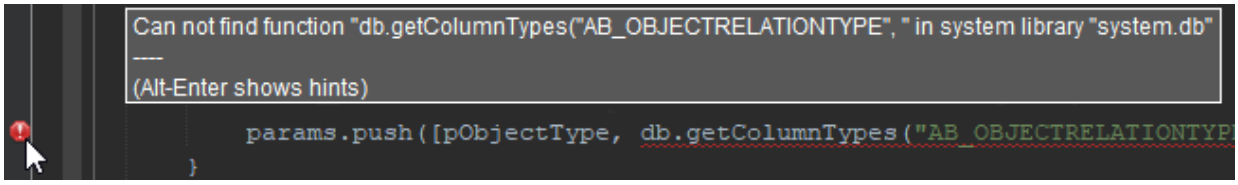
To the left of the code editor, in a vertical bar, icons appear, symbolizing errors or warnings in the respective line:

- A "light bulb" icon means "Warning".
- An icon showing an exclamation mark in a red circle means "Error". Additionally, the erroneous code is underlined in red.



If you *hover* over these icons or over the red line under the erroneous code, a small popup window will show you the details of the error/warning, which can be a hint to the problem's source. (Some of these hints are generated by the ESLint plugin and marked with the prefix "ESLint: ". See chapter [ESLint] for further details.)

Example:

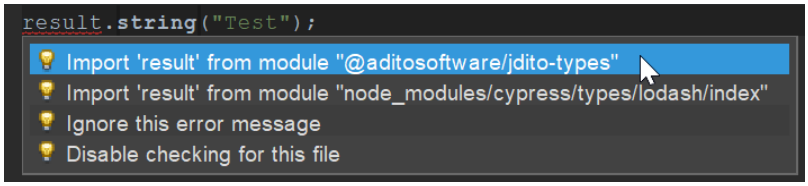


If you *click* on the light bulb, a popup list will appear, offering one or multiple automated solutions, one of which might solve the problem. Choose it carefully, as all other "solutions" will not be suitable and probably not only fail to solve the problem, but produce further problems.

(Some of the offered solutions are generated by the ESLint plugin and marked with the prefix "ESLint ". See chapter ESLint support for further details.)

In the following example, an automated generation of the missing "import" code line is offered as solutions (amongst others, which are not suitable).

Simply click on the suitable action to execute it:



→

```
import { result } from "@aditosoftware/jdito-types";

result.string("Test");
```

Most of the errors and warnings shown here can also be found using the "Scan Services.

10.4. ESLint support

If you have installed the Plugin "ESLint" (and the NodeJS plugin as well as some configuration, see chapter "Prerequisites" below), your code quality is supported by the linter ESLint, which is a tool for static code analysis of JavaScript code. It

- marks errors and warnings,
- can solve many of them automatically and
- can also be used for code formatting.

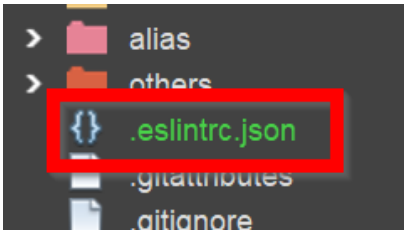
Besides the ESLint functionality as preconfigured by ADITO, ESLint can also be customized individually.

As it is a common tool in the software developer community, you can find good advice on its functions and configuration in the internet (not only on its homepage).

10.4.1. Prerequisites

To make ESLint work, the following must be given:

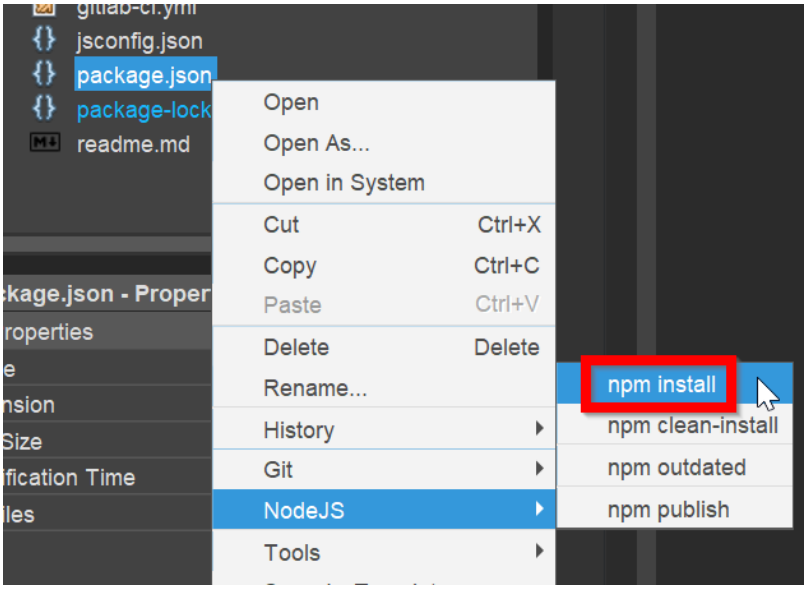
- Plugin "NodeJS & TypeScript" is active.
- Plugin "ESLint" is active.
- Configuration file ".eslintrc" is present in the ADITO xRM project (in the lower part of the "Projects" window).



- File "package.json" is present (in the lower part of the "Projects" window) and includes all required dependencies:


```
(...)
    "eslint": "8.12.0",
    "@typescript-eslint/parser": "^5.18.0",
    "@typescript-eslint/eslint-plugin": "^5.18.0",
    "eslint-plugin-brace-rules": "^0.1.6",
    "eslint-plugin-standard": "^4.0.2"
```

- command "npm install" has been executed on file "package.json"



10.4.2. Configuration

Each xRM project usually already includes an ESLint configuration file ".eslintrc" (see above). This file contains a lot of settings that (amongst others) respect the coding guidelines included in the ADITO document AID001 Coding Styles, e.g. the requirement that the end of each code line must be a semicolon.

It can be very helpful to add further settings according to your requirements, but these should always comply with ADITO's coding guidelines. ADITO does not provide a detailed documentation of the various configuration options, as ESLint that is already well-documented in the internet (see, e.g.,

here).

10.4.3. Executing ESLint

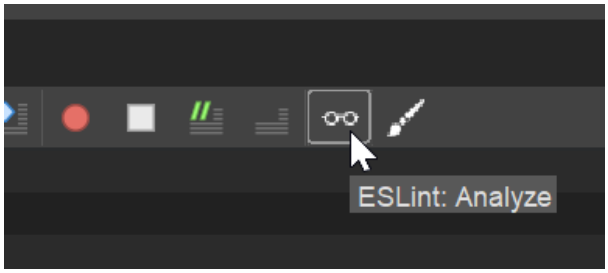
Basically, ESLint offers 2 options:

- Analyzing code
- Fixing problems (errors and warnings)

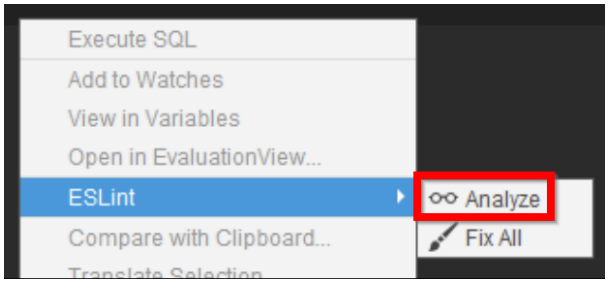
Both are generally related to the *complete* content of a *single* code window. There is no option to only fix one single problem separately from other problems existing in the same code window. And there is no option to fix all problems of the complete ADITO project (only for all files detected as "changed" by Git, see below).

10.4.3.1. Analyze

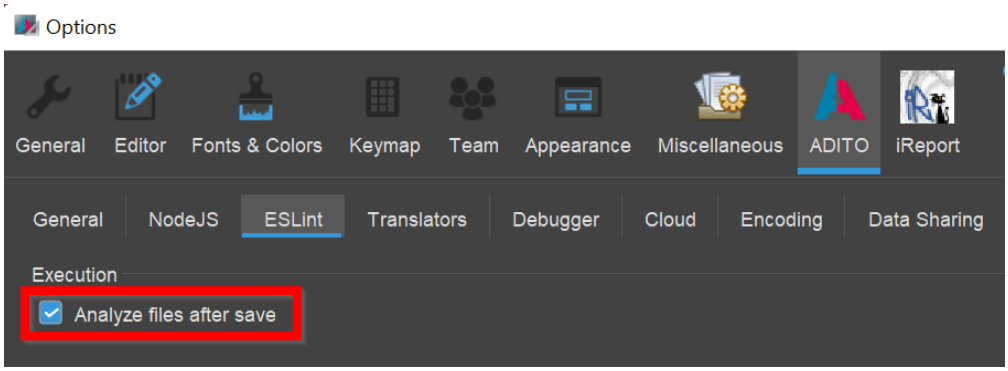
If you click button "ESLint: Analyze" (in the button bar above the code window), the code of the current code window is being analyzed by ESLint.



You can call this function also via the context menu, when right-clicking in the code window:



In the Designer options (select Tools > Options in the upper menu bar) you can activate that the ESLint analysis is automatically executed after every save:



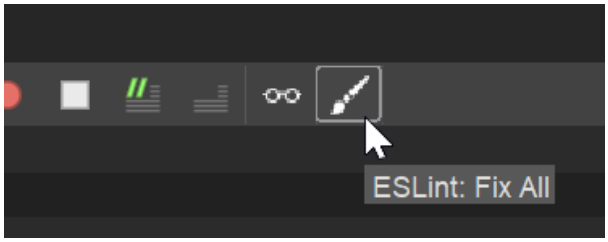
We recommend to activate this option.

If ESLint detects a problem, it is underlined in red, and an icon is shown to the left of the code. In most cases, this icon shows a combination of an error icon (= an exclamation mark circled in red) and the corresponding solution icon (= a light bulb).

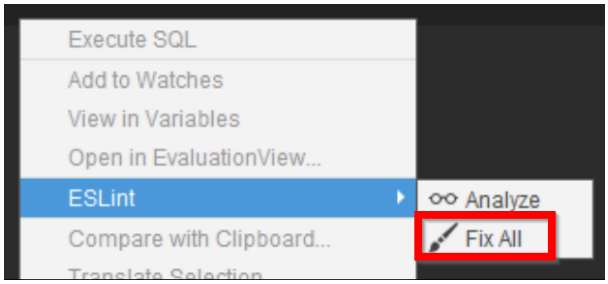
These results of the analysis are cached - thus, they are still available after closing and re-opening the Designer.

10.4.3.2. Fix all

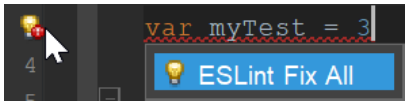
If you click button "ESLint: Fix all", all problems detected in the current code window are fixed automatically.



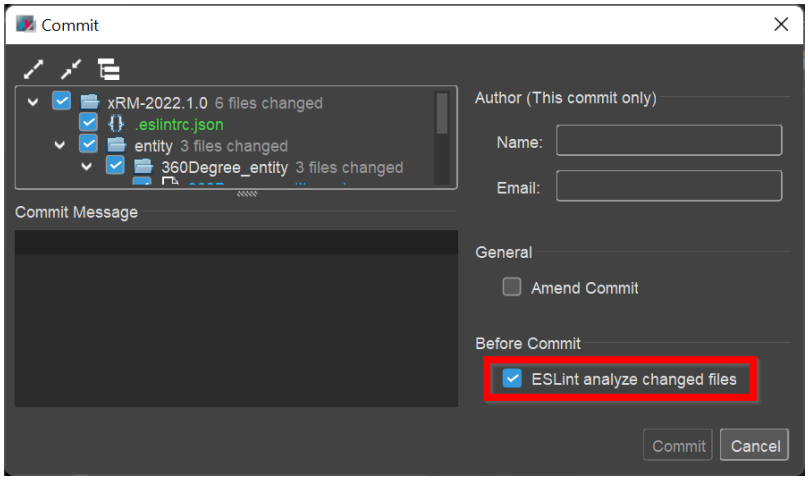
You can call this function also via the context menu, when right-clicking in the code window:



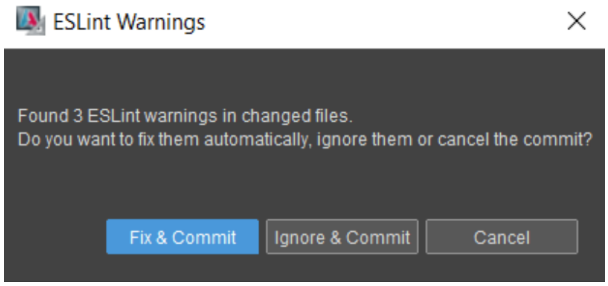
Furthermore, this function is also available if you click on the light bulb icon to the left of a code line in which ESLint has detected an error.



In the Git commit dialog you can select an option that the ESLint analysis is automatically executed on all changed files:



Then, if ESLint detects problems, they can optionally be fixed before commit:



We recommend to activate this option.

10.4.4. Examples of hints and solutions

As explained in chapter Errors and warnings, in the vertical bar to the left of the code editor, 2 kind of icons are shown whenever the Designer's code scanner detects a problem: An exclamation mark circled in red marks an error, while a yellow light bulb is a warning. In most cases, a combination of both icons is shown.

If you hover over these icons, you get a small popup window including hints to the possible source of the problem. If you click on the light bulb, various solutions are being offered in a popup window.

Those of these hints and solutions that are generated by ESLint can be identified by the prefix "ESLint:". You can use them similar to the other hints and solutions that are generated by the ADITO Designer itself, see chapter Errors and warnings - with the only difference that for ESLint-detected problems no

single fixes are possible, only "Fix all".

Here is are some examples:

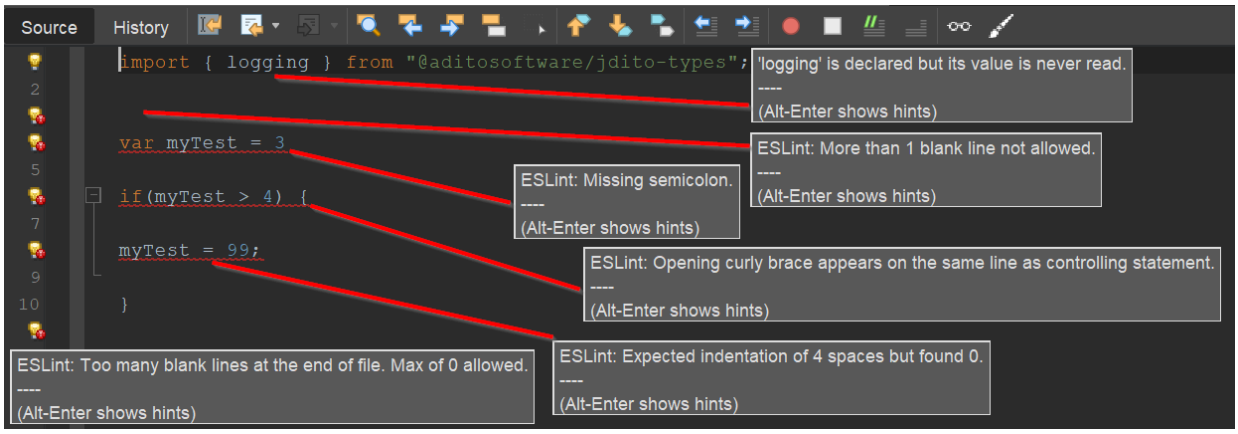


Figure 8. Examples of hints provided by the Designer (first hint) and ESLint (all other hints)

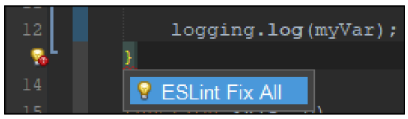
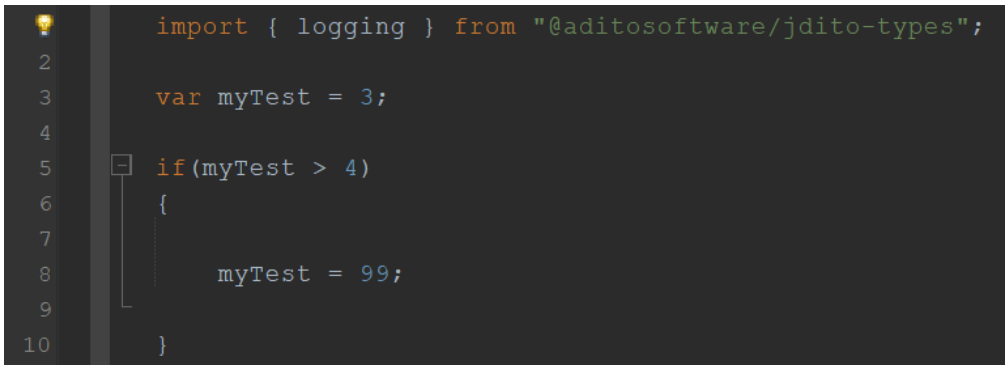
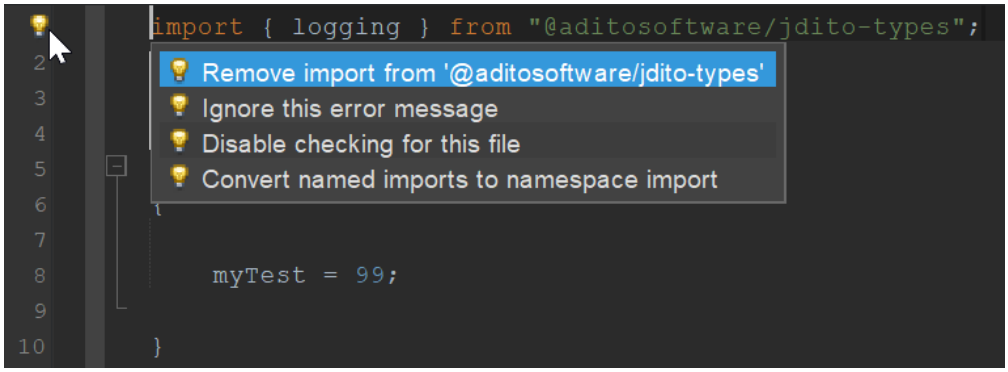


Figure 9. Solution "ESLint Fix all" (after clicking on one of the light bulb icons related to ESLint)

After the execution of the command "ESLint: Fix all", all ESLint-detected problems will be fixed within a few seconds. The result of the above example will look like this:



As you can see, the first problem was not solved, because this was not a problem detected by ESLint, but by the ADITO Designer. To solve also this problem, you need to click on the light bulb icon to the left of the respective code line and choose the suitable solution.



10.5. Scan Services

The ADITO Designer's "Scan Services" trace possible problems within your project and display them structured in errors and warnings in tab "Scan Services" (next to tab "Output").

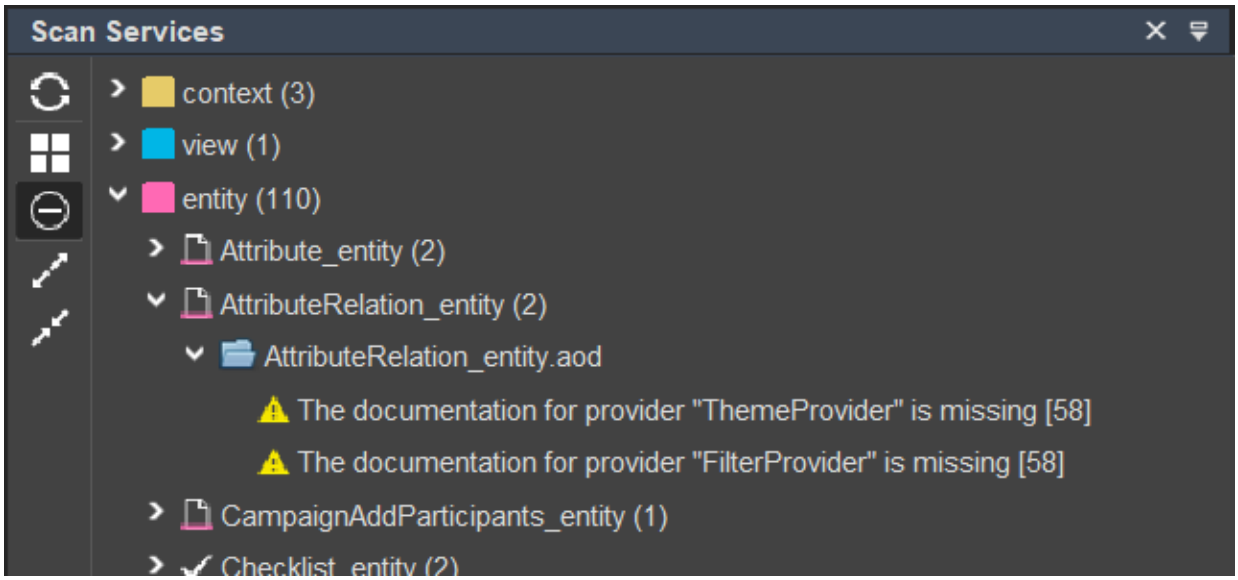


Figure 10. Example of the content of tab "Scan Services"

10.5.1. Preferences

By default, the Scan Services are

- executed completely, if
  - you open a project file for the first time, or
  - the opened version of the project file is not identical with the version you had opened before.

You can deactivate this by unchecking option "Refresh on file change" (under Tools > Options > ADITO > General > Scan Services)

- executed restricted to changes, if you
    - add elements to your project, e.g., a new EntityField or a new Action
    - change elements of your project, e.g., set a property value
- checked on every deploy. You can deactivate this by unchecking option "Check Scan Services on deploy (Errors, Warnings)" (under Tools > Options > ADITO > General > Dialogs)

The refreshing process of the Scan Services is part of the "Indexing Project" process (not to be mistaked with another meaning of "indexing" in the context of a IndexRecordContainer and the global "Index Search"), which consists of multiple subsequent phases that are indicated in the small message box in the lower middle part of the Designer:

1. "Collecting References…": References to "refactoring" feature are collected and indexed.
2. "Saving References…": The result of the collection process is persisted.
3. "Preparing Model Scanners…": Preparation of data model scanner for Scan Service indexing
4. "Analyzing Warnings and Errors…": Scan Services are evaluated
5. "Collecting Model Dependants…": Collection of dependencies between data models (also required for Scan Services)
6. "Scheduling Model Dependants…": Execution of a process for re-indexing the dependencies

If one of this processes is executed very fast (< 1 sec), it might not be visible in the message box.

It is possible to cancel running "Indexing" processes. This is explicitely NOT recommended, especially not during phase "Collecting References…", as then the "refactoring" feature will fail or being executed incompletely. If you cancel it anyway, a warning message will appear.

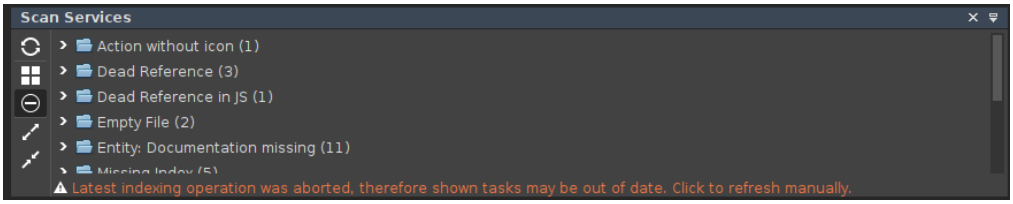


Figure 11. Warning message appearing when "Indexing" is cancelled

10.5.2. Result

By default, all types of messages (errors, warnings, hints, etc.) will be displayed, structured as a result tree. If you mark one or multiple tasks, you can make them disappear from the result tree via context menu option "Ignore selected tasks". This is not possible for results of type "Error". By clicking the respective switch (see table below), you can let these "ignored tasks" reappear and, if required, choose

option "Include selected tasks" in the context menu, in order to show them permanently again.

If you double-click on a result (or choose "Open in editor" in the context menu), the corresponding ADITO model will be opened in the "Editor" and "Navigator" windows, enabling you to quickly find the source of the related problem.

Content and design of the result tree are controlled via buttons to the left:

Table 1. Buttons of tab "Scan Services"

	Refresh all Scan Services for the complete project. This runs the same processes that are executed when you open a project (see list in previous chapter "Preferences").
	Enables you to select what type of information is to be included in the result tree and how the tree is structured.
	Switch to control whether or not those tasks are shown that have been marked as "ignored" (see above).
	Expand all nodes of the result tree.
	Collapse all nodes of the result tree.

Part of these options are also available via the context menu of the nodes of the result tree.

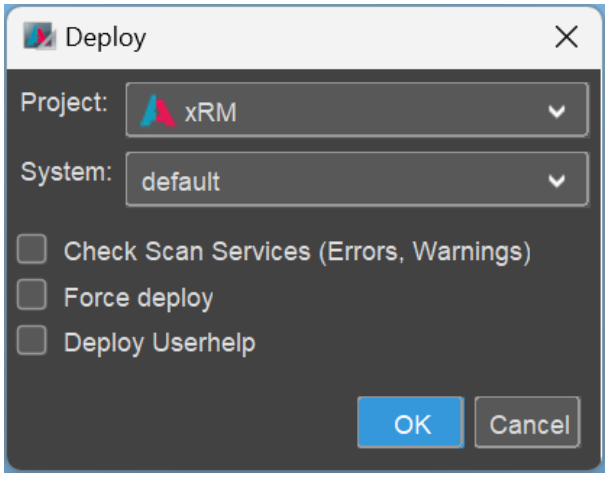
11. Deploy

Deploying a project means to transfer the local development status into the registered ADITO system database (in particular, the table ASYS_SYSTEM). Besides a deploy via the ADITO Designer, there is a Deploy Tool, which allows a standalone deploy, without the Designer.

Make sure that the deploy toolbar is shown in the Designer's button bar (menu "View" > "Toolbars" > check "deploy").

If you want to deploy all changes in your project, press button "Deploy project" in the button bar:

The following dialog will open:



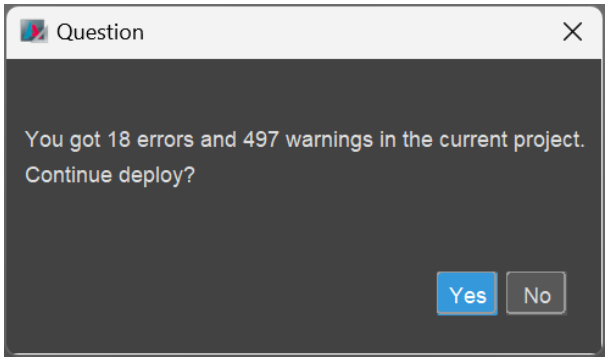
If you have more than one project opened, you need to choose the project you want to deploy via the combobox in line "Project".

One project can have several systems. If so, select the system to which database you want to deploy via the combobox in line "System".

Be very careful with the selection in line "System", in order to avoid, e.g., that by mistake you deploy into the productive system instead of your development system.

Furthermore, the deploy dialog includes 3 checkboxes.

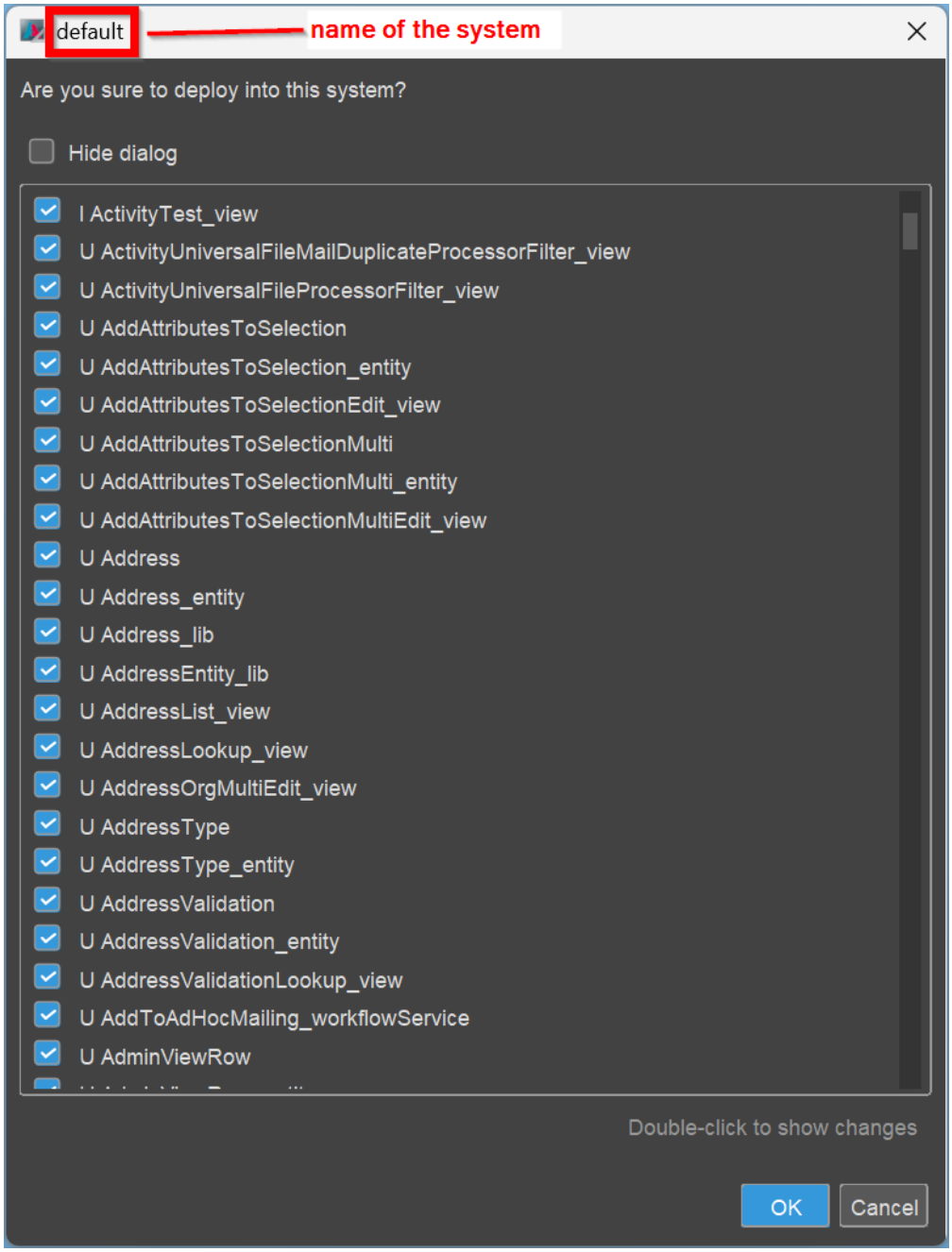
- "Check Scan Services (Errors, Warnings)" will show a dialog with the result of the latest run of the Scan Services, before the deploy starts.



This dialog enables you to abort the deploy, if you are not sure about the errors and warnings and want to analyze them first.

- "Force deploy": If this checkbox is unchecked, only those data models are deployed that have actual changes. The process that searches for these changes works with caches; in rare cases it can happen that some changes are not found and therefore not deployed. In this case, you can check "Force deploy", which deploys *all* data models of your project, no matter if they include changes or not.
- "Deploy Userhelp": If this checkbox is checked, the User help is included in the deploy.

Subsequently to this dialog, another dialog will show all data models that are prepared for being deployed.



The header of this dialog shows the name of the system into which the data will be deployed. Once again, check carefully if you have selected the correct system (see above).

If you double-click on the name of a data model or right-click on it and choose "Show changes", you can see the actual changes that will be deployed.

The letter before the name of a data model refers to the SQL statement that will be executed in the system database, being the actual deploy:

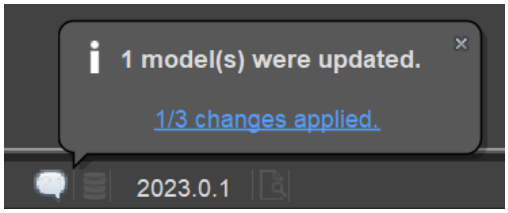
- I for Insert: This data model is new and will be deployed (inserted in the system database) for

the first time.

- U for Update: This data model is already in the system database, but was changed meanwhile, so it will be updated now.
- D for Delete: This data model is no longer present, so it will be deleted in the system database now.

If required, you can deselect data models that should not be deployed.

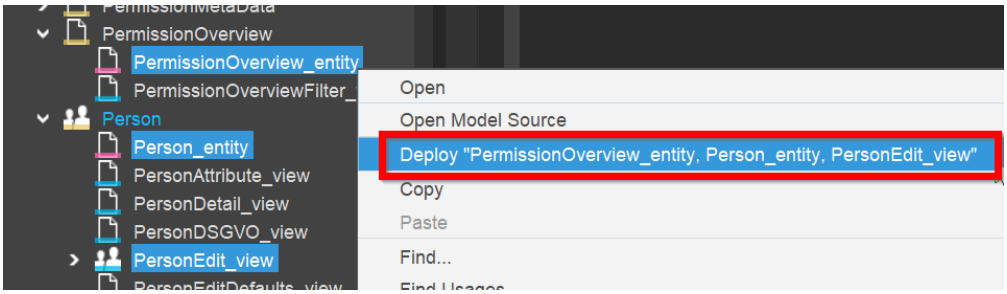
If you press the "OK" button, the deploy will start. When the deploy is finished, a message balloon will appear in the footer of the Designer:



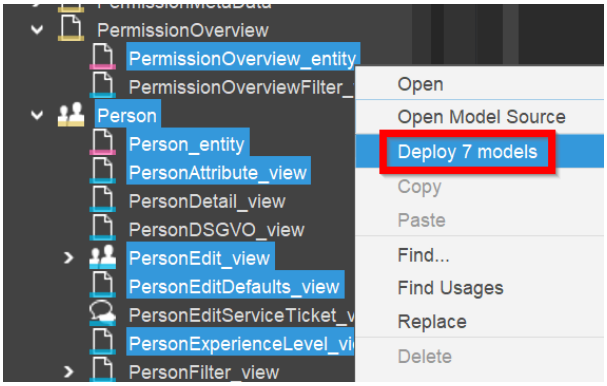
11.1. Deploy of selected data models

If you do not want to deploy all new or changed data models, you have 2 options:

- You can deselect data models in the deploy dialog (see previous chapter)
- In the project tree, you can mark all data models that should be deployed. (Hold CTRL to mark one data model after another or SHIFT to mark multiple data models in one step.) Then right-click on one of the marked data models and choose option "Deploy (...)" from the context menu. If you have marked up to 3 data models, there names are shown in the context menu:



If you have marked more than 3 data models, their sum is shown in the context menu:



Subsequently, the same deploy dialogs are shown that you know from a full deploy (see previous chapter).

11.2. Deploy of opened data models

In addition to the "Deploy Project" action (see above), you can also restrict the deploy to the data models opened in tabs in the "Editor" part of the Designer. There are 3 ways to do this:

- Press button "Deploy n opened model(s)" in the "deploy" part of the button bar:

- Right-click on any opened tab and choose option "Deploy n opened model(s)" from the context menu.
- Use the shortcut CTRL-ALT-F8.

11.3. Deploy Tool

The ADITO Deploy (aka "Deploy Tool") enables you to deploy your project via the command shell, i.e., without using the ADITO Designer. This is, e.g., a common requirement whenever automated deploy procedures are to be established.

11.3.1. Prerequisites

The following prerequisites apply:

- The ADITO server must be running.
- The Designer must be closed. (Technically, the Deploy Tool can be considered as a "Designer without a GUI". Therefore, conflicts are likely, if both applications would be running.)
- The path of the JRE must be set in the file ADITOdeploy.conf, which you can find in the "config" directory of the ADITO installation.
  Example: `jdkhome = "C:\Program Files\Java\jre1.8.0_291"`
  (Simply reference the JRE that has automatically been installed by the ADITO installer.)

11.3.2. Configuration and execution

The following example commandos are to be used under Windows. At the end of this chapter, you can find a description of the differences between execution under Windows and Linux.

In the "bin" directory of the ADITO installation you will find an executable file, which has the following name:
* under Windows 64 bit: "ADITOdeploy64.exe"
* under Linux and Mac: "ADITOdeploy"

To execute the deploy tool, call the executable files via command prompt (Windows Power Shell etc.), along with the following parameters:

Mandatory parameters:

- --projecthome (shortName = 'p', longName = "projecthome")
- --serverconfig (shortName = 's', longName = "serverconfig")

Additional parameters:

- --user (longName = "user")
- --password (longName = "password")

- --host (longName = "host")
- --port (longName = "port")
- --force (shortname = "-f", if "-f" is set, forcedeploy is active, if the flag is not set, forced assembly is not active)
- --userhelp If this parameter is set, then the "user help" will be deployed. (This is the help functionality available in the client via the little "questionsmark" icons.)

Example:

```
ADITOdeploy64.exe --projecthome "C:\Users\j.smith\Documents\AditoProjects\2021-05-31-xRM-2021.1.0" -f --serverconfig "C:\Users\
j.smith\Documents\AditoProjects\2021-05-31-xRM-2021.1.0\data\config\serverconfig_default.xml"
```

When using short names, please note that in this case, the equality sign (“=”) must not be written. Example:

```
--serverconfig="C:\Users\j.smith\Documents\AditoProjects\myProjectName\data\config\serverconfig_default.xml"
```

is equivalent to

```
-s"C:\Users\j.smith\Documents\AditoProjects\myProjectName\data\config\serverconfig_default.xml"
```

11.3.3. Exit codes

To request the exit code of the execution of the Deploy Tool under Windows, please note that - unlike under Linux - processes are executed asynchronously by default. In order to wait for the result blockingly, we suggest the following patterns:

- PowerShell (output of exit code in the command line):
  ```
  $p = Start-Process ".\ADITOdeploy64.exe" -ArgumentList "--
  projecthome '<YourProjectPath>\2021-05-31-xRM-2021.1.0'
  --serverconfig '<YourProjectPath>\2021-05-31-xRM-
  2021.1.0\data\config\serverconfig_default.xml' -PassThru -Wait;
  $p.ExitCode
  ```
- Cmd (output of exit code in a file):
  ```
  start /wait cmd /c "ADITOdeploy64.exe --projecthome
  '<YourProjectPath>\2021-05-31-xRM-2021.1.0' --serverconfig
  '<YourProjectPath>\2021-05-31-xRM-
  2021.1.0\data\config\serverconfig_default.xml' & call echo
  %^errorLevel% > .\exitcode.txt"
  ```

The following exit codes are possible:

- 0: Deploy successful
- 142: Deploy partially successful
- 150: Exception of other type

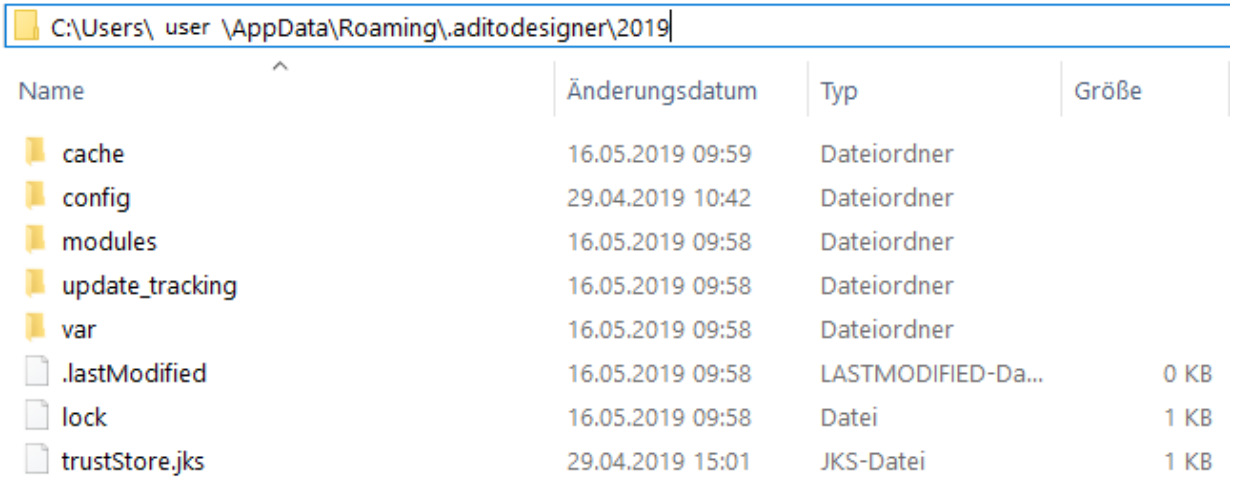
12. Local Data

The local project folder contains only the alias definitions. These include the name of the alias and the types of the alias. The connection data and the alias configuration are in the system database table ASYS_ALIASCONFIG of the respective ADITO system.

The user data also depends on the respective system. Therefore, these are only stored in the database and are located in the system database table ASYS_USERS.

These data are kept in the database because they belong directly to each system and should prevent local changes from hindering the system's functioning.

Local Designer Data



When using the ADITO Designer, the Designer will save local data under the path:

"C:\Users\<username>\AppData\Roaming\.aditodesigner\2020"

Replace "<username>" with your user on the computer.

These local files contain a multitude of different things. For example individually set colors for your Designer are saved here, but also things like the last update and installed plugins.

When there are problems with the Designer it often helps to force these local files to be created newly. Just create a differently named copy and delete the original folder or rename the original one.

The path is can also be found and changed in

"C:\Program Files\ADITO2020.0.1\config\ADITOdesigner.conf"

Simply use a text-editor like notepad to open the file.

In the same location you also find ADITOdesigner.clusters which lists all java-libraries which are used.

Just add infront of the path shown:

"C:\Program Files\ADITO2020.0.1"

13. Updates

In regular frequency, ADITO releases new versions both of the ADITO platform and of the ADITO xRM project. If your own project is based on the xRM project, it can be helpful to merge new xRM versions into your own project, in order to make use of the latest new features and bugfixes. Furthermore, please consider that the version of the ADITO platform (Designer, backend, etc.) must fit to the version of the project.

You will find all information required for handling ADITO updates in the Update Manual, which will in itself be updated whenever a new ADITO version is released. We recommend you to read the Update Manual regularly and carefully.

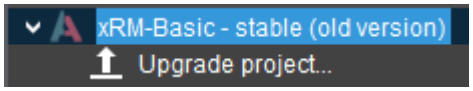
13.1. Update the ADITO project

13.1.1. Basics

Normally a ADITO project will be kept on the same version for as long as possible. But sometimes it might be necessary to upgrade the project to a newer version.

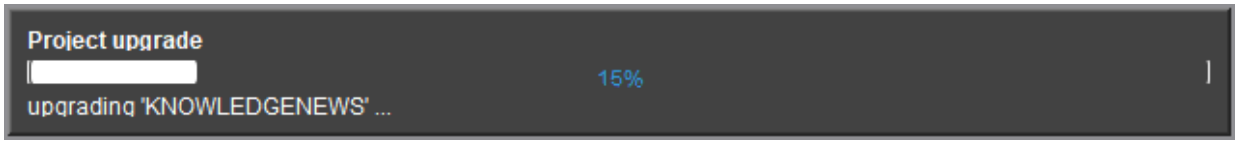
To start this procedure install the new ADITO version and start the Designer. Next open the project in the new Designer. It will appear greyed out as it is not the correct version for the Designer.

When loaded into the Designer it will still be greyed out with only the field "Upgrade project..." beneath.

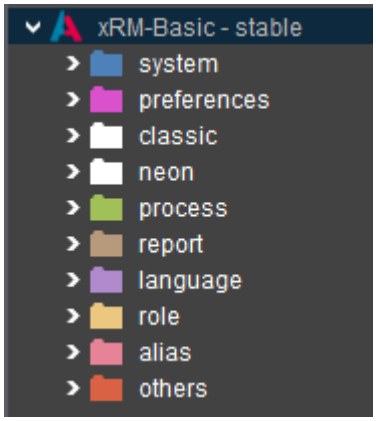


Double click this field to start the upgrade process. A pop up will open, asking for confirmation and informing, it won't be possible to open the project with older versions anymore. If not done so already, it is recommended to create a copy of your project before this step.

The automated process will start now.



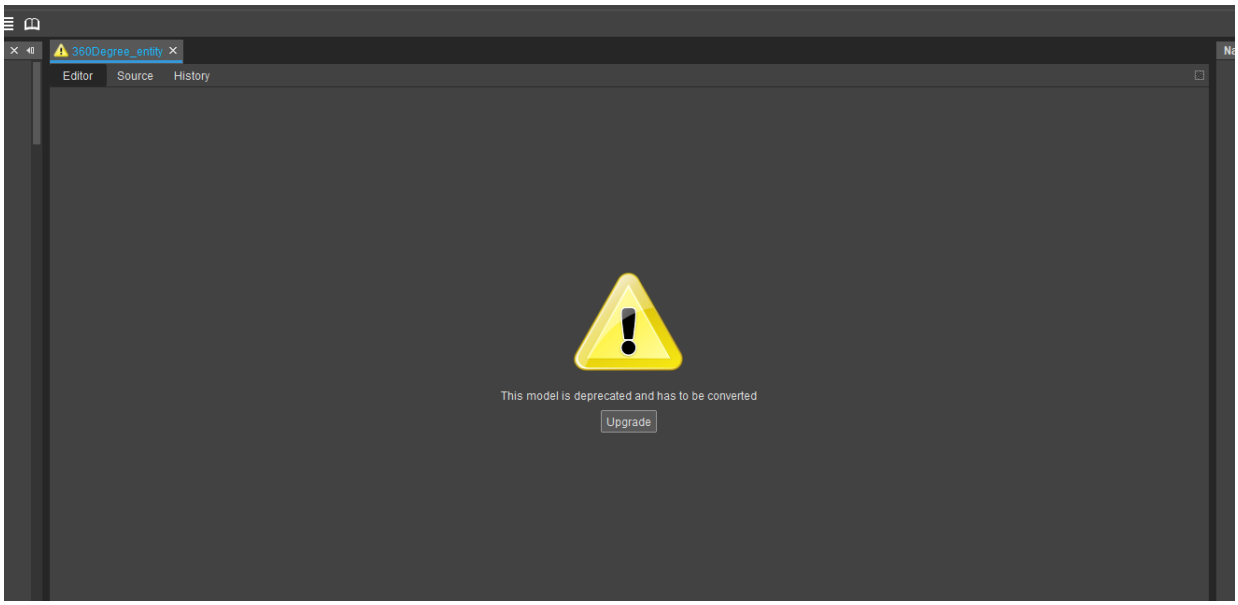
As soon as the process is finished the normal project structure will show again.



There still might be some problems in the code which can not be automatically fixed, so it is still necessary to manually check the client for errors to pop up.

13.1.2. Update of single ADITO models

If there is an updated available for a single model of the project (e.g., a specific Entity, which you have imported from an older project) and you double-click on it, the following message will be displayed in the editor:



To upgrade the model, click on button "Upgrade".

13.2. Update the database

When there are database changes between two updates or the system database requires a new version, the ADITO server cannot be started, but an error message will be logged.

13.2.1. Basics

To be able to update the database without starting the server, you need to start the database alone first: Simply select it in the combobox of the button bar and then press the "green triangle" button. In window "Output", you can view the log entries until the database is started.

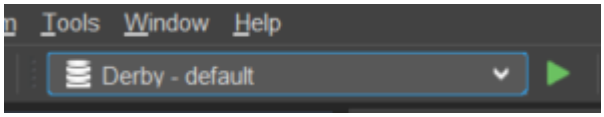


Figure 12. Starting the database without the ADITO server (example for a Apache Derby database)

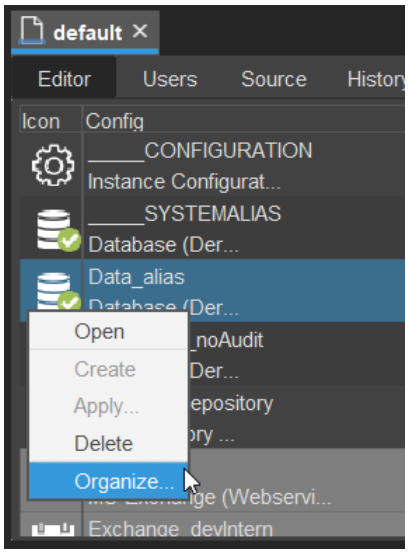
When the database is running, double-click on your system (e.g., system > default). Then the available databases will appear in the Editor area of the Designer, and you can continue, e.g., with the "Organize" function, see following chapters. (The option named "Upgrade", which was included in earlier ADITO versions, has been replaced by "Organize".)

13.2.2. Maintaining system tables

System tables are database tables that are essential or optional for running the ADITO system itself (e.g., ASYS_SYSTEM, ASYS_USERS, ASYS_BINARIES), indepentently from the tables holding data managed by the client user (e.g., PERSON, ORGANISATION, CONTACT).

While in ADITO versions earlier than 2021.0.0, system tables resided exclusively in the system alias, they can now reside in multiple aliases. This can be configured in the preferences of the project.

When the database is running, double-click on your system (e.g., system > default). Then the available databases will appear in the Editor area of the Designer. If you right-click on the database, you will find option "Organize System Tables" in the context menu.



After selecting this option, a dialog will open, named "System Table Control Center". It shows a list of all

system tables to the left, while further information on the currently selected system table is available in the right part of the dialog.

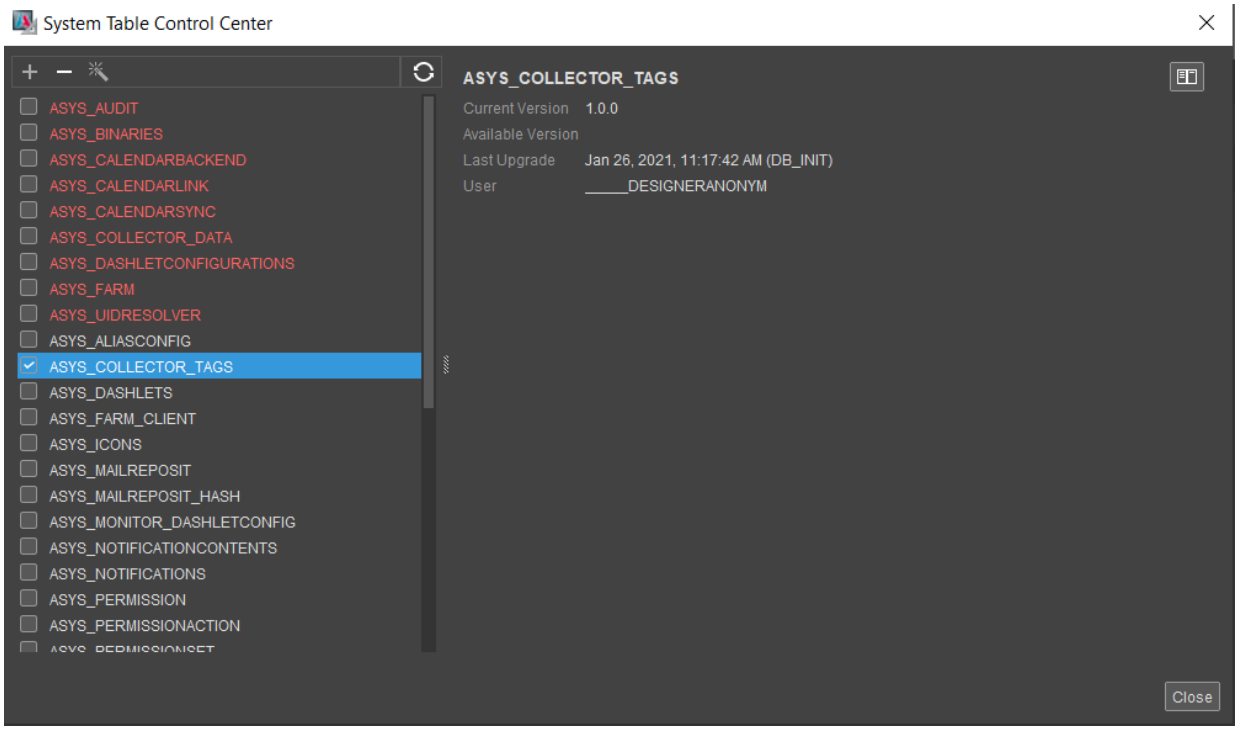


Figure 13. System Table Control Center

Besides the approach explained in this chapter, you can also create system tables

- via customizing, using Liquibase (see chapter Plugin Liquibase)
- via a tool named "ADITOdatabase", which enables you to create system tables via command line or batch file (see appendix Create and upgrade system tables)

13.2.2.1. ASYS_VERSIONHISTORY

Table ASYS_VERSIONHISTORY contains all version-related information on a system table. Initially, this table will automatically be set when executing an upgrade of the system database (see earlier chapter). If it is deleted, it will automatically be re-created on the next startup of the dialog.

13.2.2.2. List of system tables

The list in the left part of the System Table Control Center contains all system tables that

- exist or
- do not exist, but are essentially required

If a table is treated as "essentially required" depends on the settings in the preferences. Example: Property userdirectoryAlias (preferences > ____PREFERENCES_PROJECT > Modules) determines, in which DB alias the database tables for user directory and user data reside.

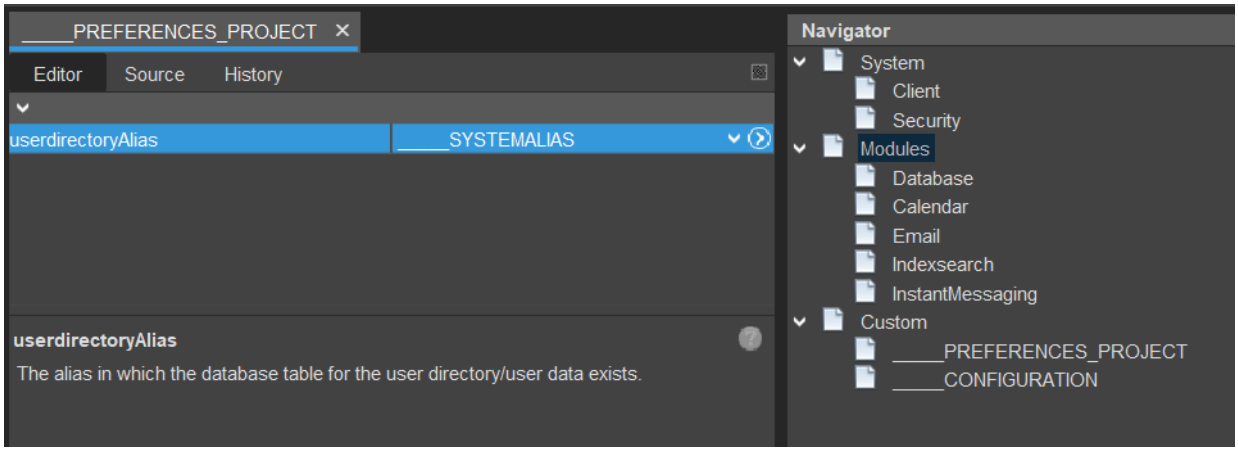


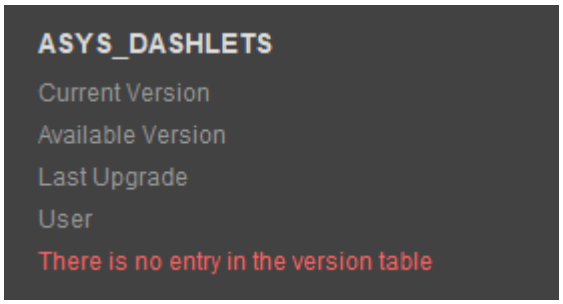
Figure 14. Example of a property determining if a table is treated as "essentially required"

13.2.2.3. Faulty system tables

If there is a problem with a system table, it will be marked red in the list. If you select it, further hints on the problem are shown to the right. The most common problems are as follows:

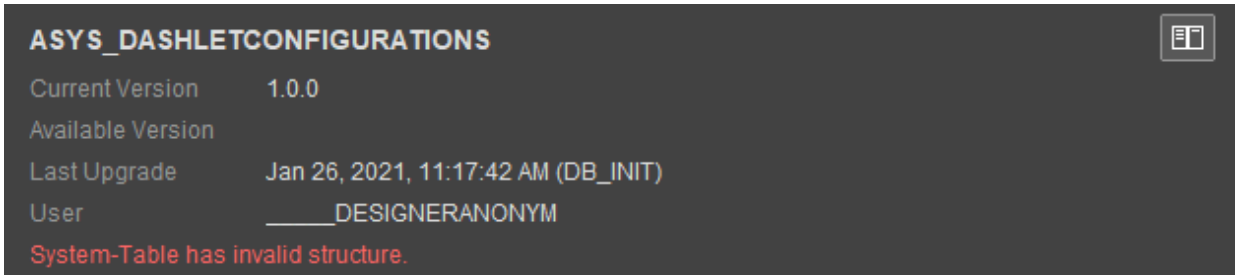
13.2.2.4. Missing entry in ASYS_VERSIONHISTORY

If a system table exists, but there is no related entry in table ASYS_VERSIONHISTORY, the following note is shown:



13.2.2.5. Wrong structure

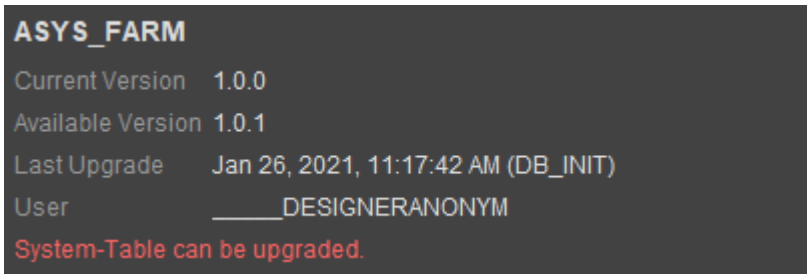
If the structure of the database table does not meet the structure required by the ADITO platform, the following note appears:



If you have the Plugin "Git" installed, you can inspect the structural differences by choosing action "Show Diff" (upper right side).

13.2.2.6. Table upgrade

Is a table ready for being upgraded, the following note is shown:



13.2.2.7. Table missing

If a table is missing that is essentially required, the following note will appear:

13.2.3. Action toolbar

There is a toolbar providing actions adding and deleting system tables, as well as for resolving the above problems. Generally, these actions are only applied to the currently selected system tables. If an action might lead to data loss, a warning dialog will be shown that enables you to cancel the action before it is executed.

The effect of the actions is as follows (description from left to right):

- Create optional tables: By default, the list contains only tables that already exist or that are essentially required. If you want to create an optional table, klick on the "+" button (plus sign) and select the table you want to create.

- Delete
  A click on the "-" button (minus sign) deletes all tables selected via the checkboxes.
- Resolve problems

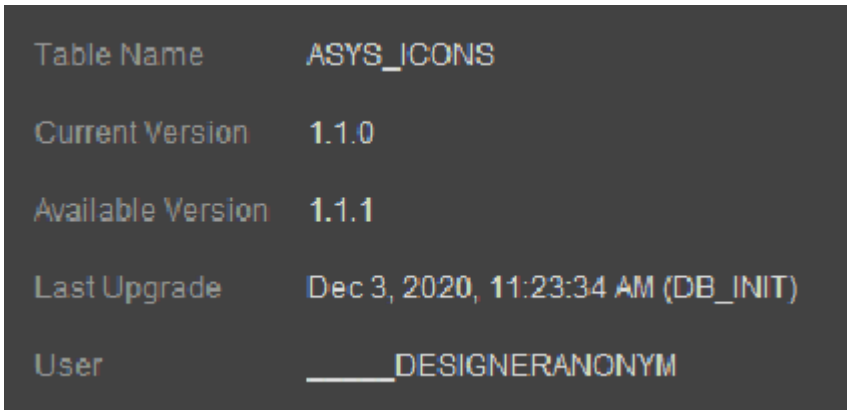
A click on the "Resolve problems" button ("magic wand" icon) resolves the problems related to the currently selected system tables (see chapter "Faulty system tables"), e.g., a table is created or upgraded.

- Refresh

A click on the "Refresh" button refreshes the complete System Table Control Center dialog. This can be required, if database changes have been performed in the background (i.e., outside the dialog).

13.2.4. Information area

In the right part of the dialog, additional information on the currently selected system tables is given.



This information is structured as follows:

- Table Name
- Current Version
- Available Version: Version that is available via upgrading.
- Last Upgrade: Date of the latest upgrade

- User: User that has performed the latest upgrade

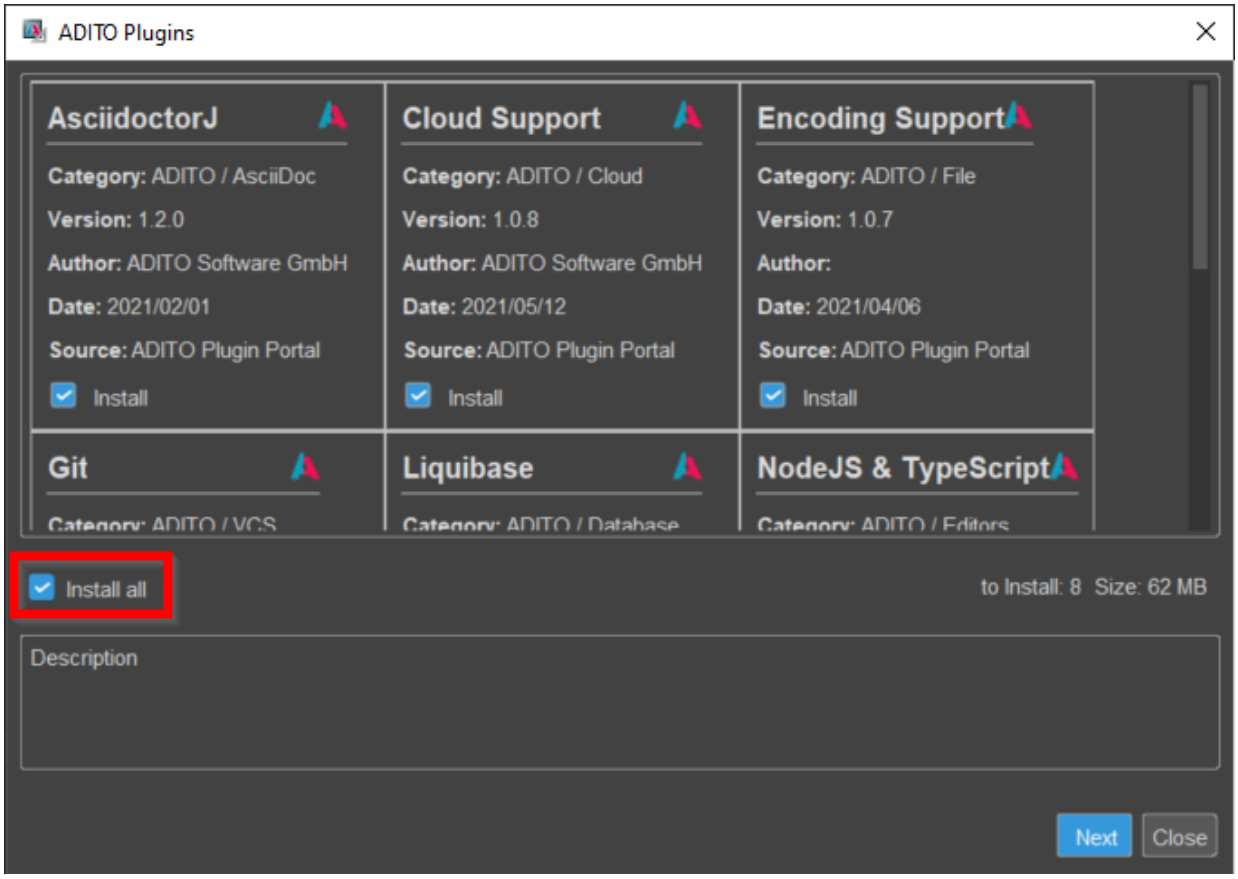
13.2.5. Server startup prerequisites

The ADITO server can only be run, if tables ASYS_ALIASCONFIG and ASYS_SYSTEM are present in their latest versions, i.e., they do not appear as "upgradeable".

14. Plugins

14.1. Installation

The ADITO Designer offers function packages to be added or removed on demand, in the form of plugins. When starting the Designer for the first time, a dialog will prompt you to decide which of the plugins you want to use; we recommend you to choose "Install all", in order to have the full Designer functionality available (you can deactivate them at any time later, see below).



We recommend you to install at least these plugins, as they will be required in most cases:

- AsciidoctorJ
- Encoding Support
- Git
- Liquibase

However, you can also add, update, or remove these or further plugins at a later time, via the Plugins dialog, available via menu Tools > Plugins.

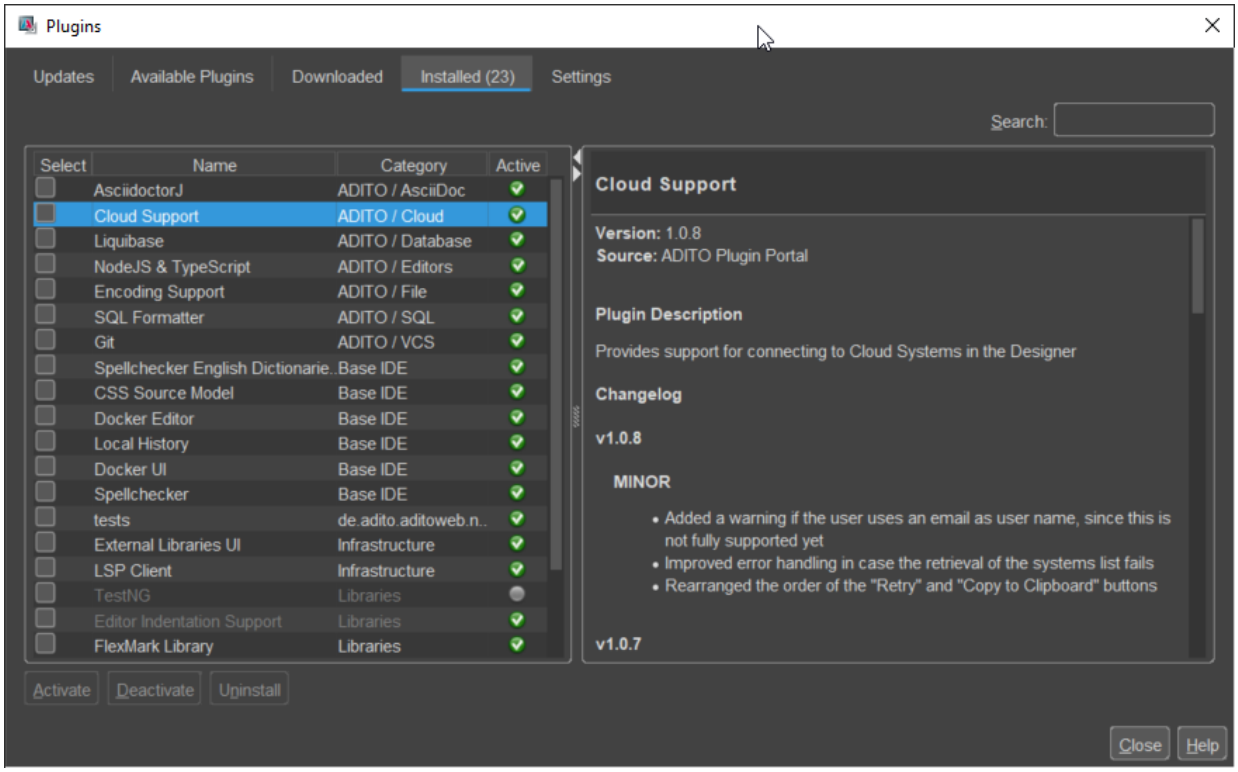


Figure 15. The Plugins dialog

The dialog consists of these tabs:

- "Updates" shows all updates available for the installed plugins.
- "Available Plugins" lists every plugin of the given source.
- "Downloaded" enables you to import other plugins as nbm-files.
- "Installed" shows all locally installed plugins (some of which had been automatically installed). You can also deactivate/reactivate plugins here.
- "Settings" contain the plugins' source, as well as the frequency to check for updates.

In case your firewall blocks the download of plugins, you may proceed as follows.
In the this catalogue you will find all plugins available for the ADITO Designer:
https://aditopluginsonline.adito.de/2020.2.0/catalog.xml
In attribute "distribution" of tag "module" you see the URL for downloading the respective plugin. The download result will be a file of format ".nbm" ("NetBeans Module").
If you want to open your firewall for downloading the plugin directly in the Designer, then you need to use the following parameters:
URL: "https" / see catalogue above
Port: 443
Applikcation: HTTP / Webserver

> Target port: 443
>
> If you do not want to open your firewall, then you can alternatively download the plugins from somewhere "outside" the firewall and install them manually - but please note, that in this case, the automation of plugin updates will not work.

14.2. Advantages

The plugin concept offers you the following advantages

- You can optimize the Designer’s performance by running only those plugins that you actually need.
- Enhancements, improvements, and bugfixes can be provided via plugin updates, independently from new ADITO version releases. By default, the Designer checks for new Plugin versions at every startup. You may change the frequency via tab "Settings" in the Plugins dialog (see screenshot above).
- In the rare case that a plugin-related bug disturbs your work, you can simply de-activate the respective plugin via Tools > Plugins > Installed > <Mark respective plugin> > button "Deativate" (or, if required, button "Uninstall")

In the future, ADITO will consequently extend the Designer’s functionality via plugins, which will then appear under Tools > Plugins > Available Plugins, ready to be installed.

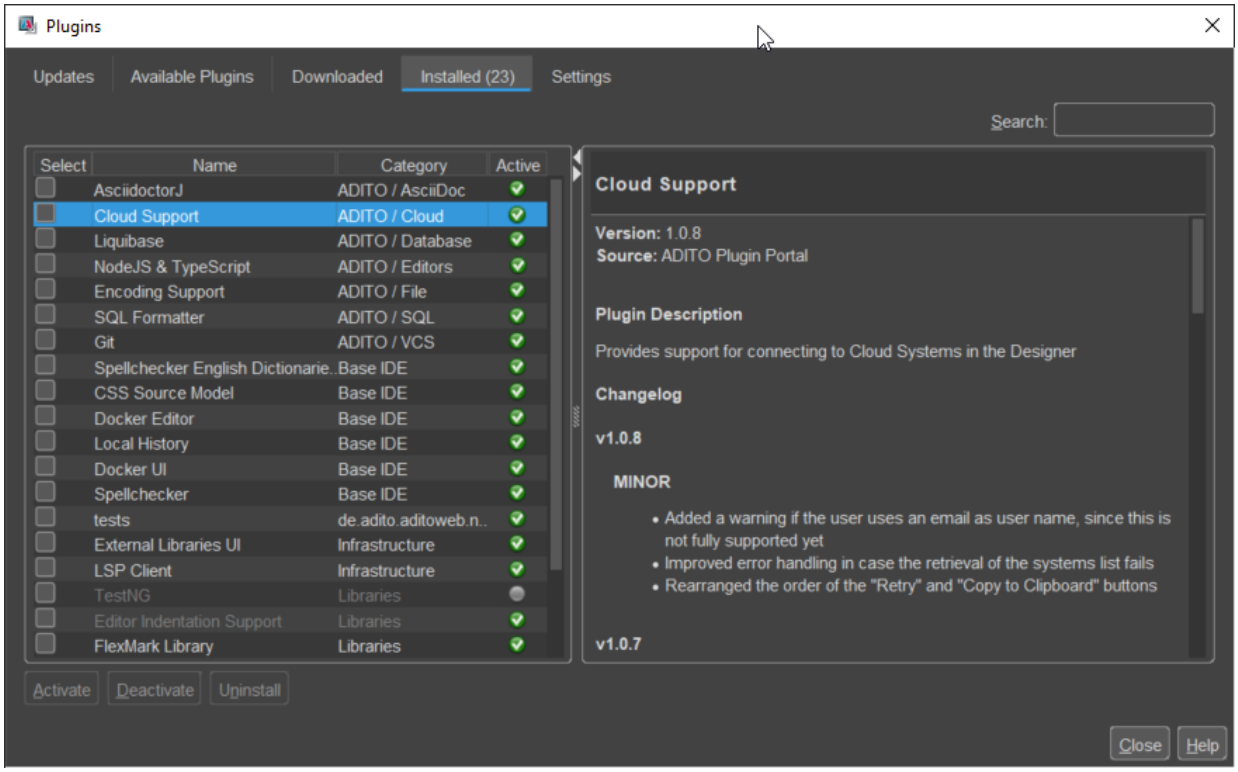


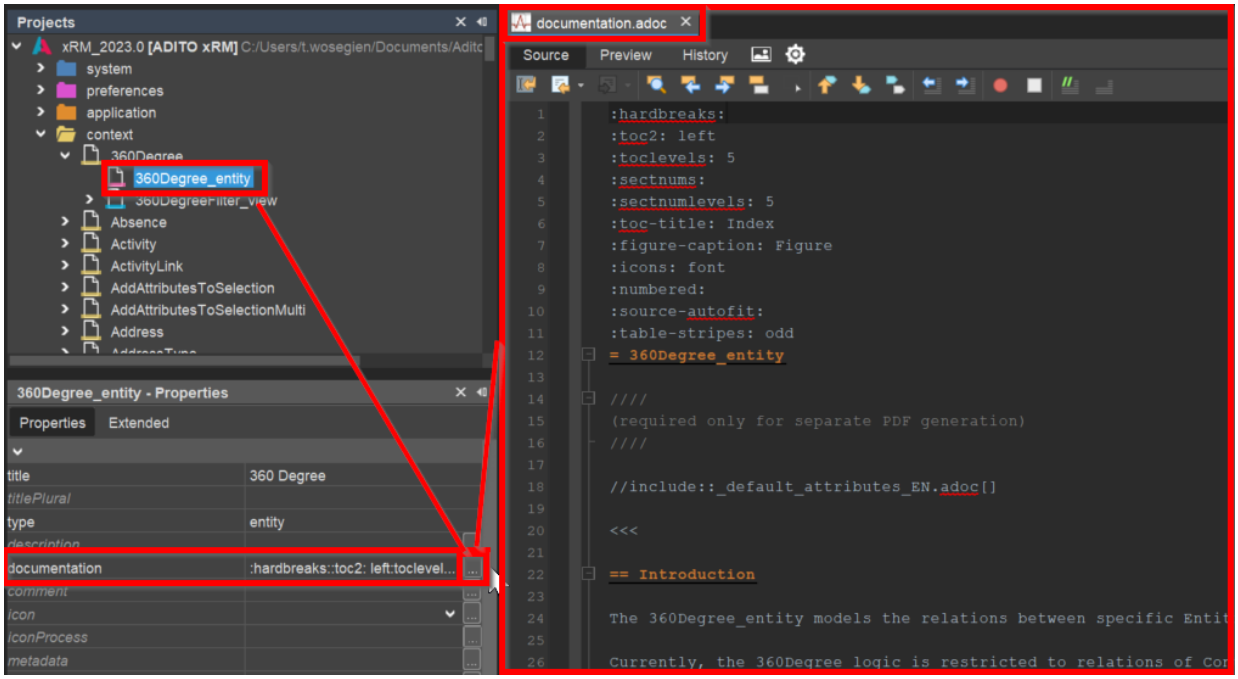
Figure 16. The Plugins dialog

In the following chapters, you will find a description of selected plugins. If you need assistance for further plugins, please contact ADITO via Service Client ticket.

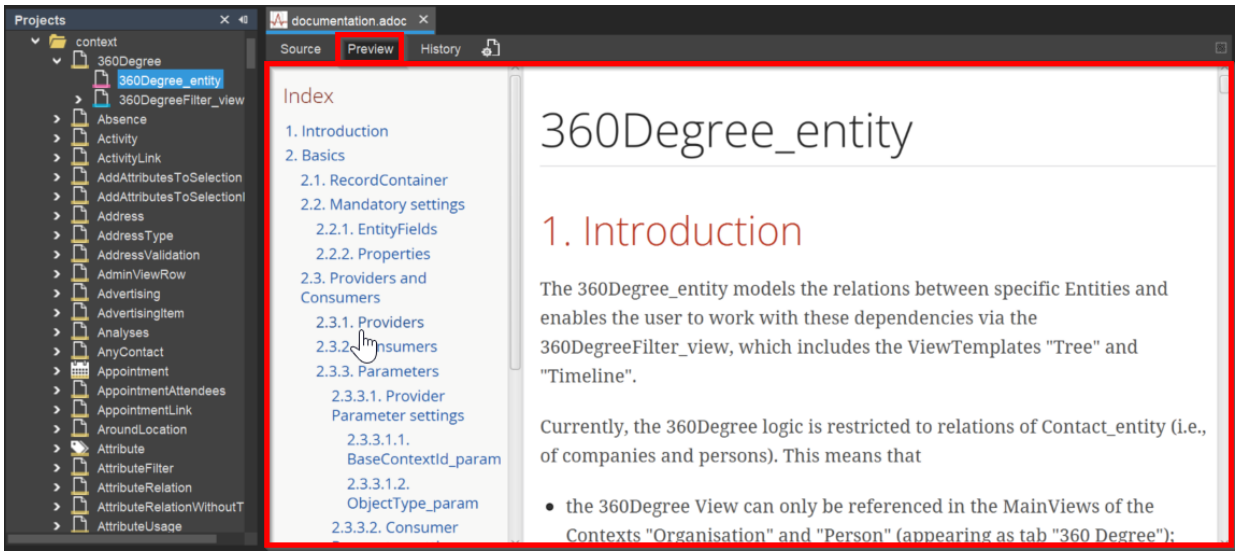
14.3. Plugin AsciidoctorJ

The plugin AsciidoctorJ provides preview functionality for Asciidoctor (adoc) files. In ADITO, these files are used for documentation purposes, mainly available via property "documentation" of various ADITO models or via the
ADITO Userhelp. Besides, you can create additional adoc files on demand, e.g., in the project folder "others" (choose New > Other… from the context menu of "other" and then select "Asciidoc" in the popup dialog).

A good example in the xRM project is the documentation property of 360Degree_entity:



As you can see, reading the documentation text via the source file is quite difficult, as there are a lot of control characters and terms of the AsciiDoc markup language. Now, when the plugin AsciidoctorJ is installed, a new tab "Preview" is available to the right of tab "Source". If you click on tab "Preview", the document is rendered and shown in a formatted, readable way:



You can learn more about the adoc language by

- scanning the source of examples from the xRM project, like the above;
- reading the official online documentation of the open-source tool Asciidoctor;
- use other online resources, such as the AsciiDoc Cheatsheet.

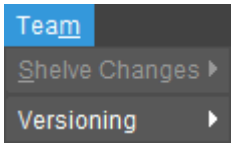
Adding images to adoc documents is currently only supported by ADITO for Userhelp documents. Find more information in chapter "Adding images and videos" of the ADITO manual AID005_Userhelp.pdf.

14.4. Plugin Git

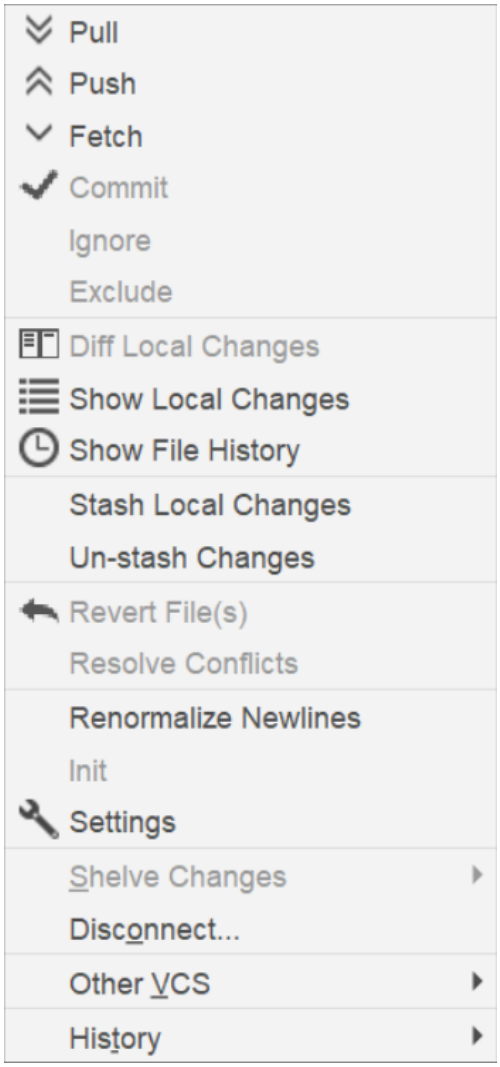
Also the ADITO Designer in theory has a version control tool in the menu group "Team". However, this tool is not used in most cases. Instead, nearly all ADITO projects with more than one developer use Git an are mostly hosted on GitLab repositories or on web sites like GitHub.

At a high level, GitHub is a website and cloud-based service that helps developers store and manage their code, as well as track and control changes in their code. It is a version control tool and a way to manage code across multiple developer devices.

When the Git plugin is not installed you will find the following under "Team".



If Git is installed, the content of this menu group changes as follows:



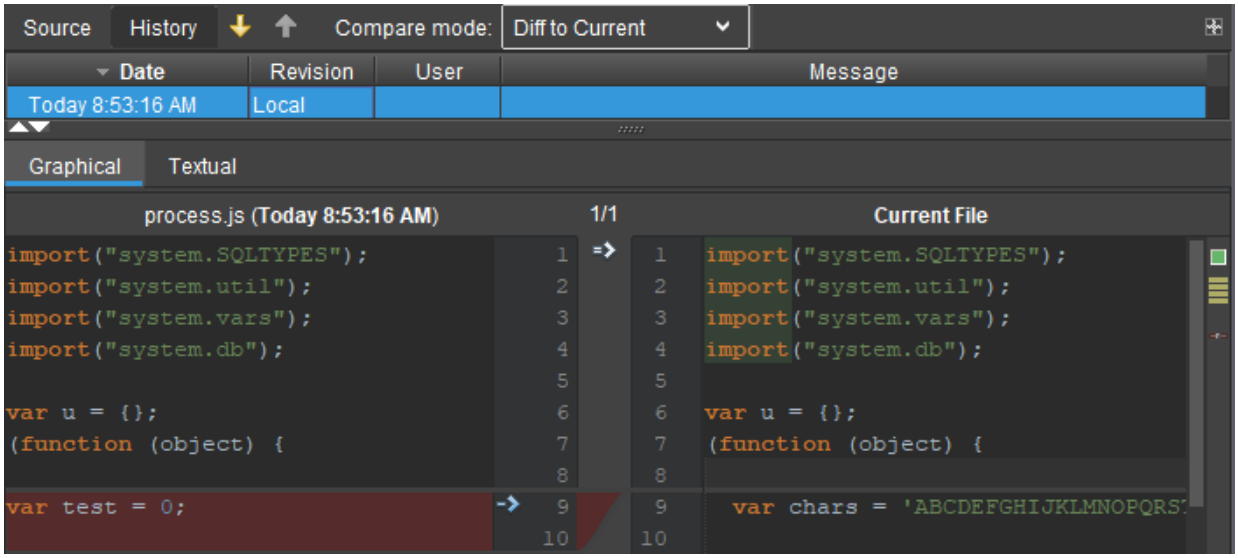
(At first start, in some cases, you first need to execute option "Init" in order to initialize the Git folder ".git" in your local project directory.)

In this menu you find nearly all options you need to operate Git via the Designer. The exact way Git is used may vary from project to project.

For comprehensive information on the use of each individual option please refer to the official Git documentation. The following explanations are only a rough introduction.

One function that is handled via the Git plugin, is the option "History".

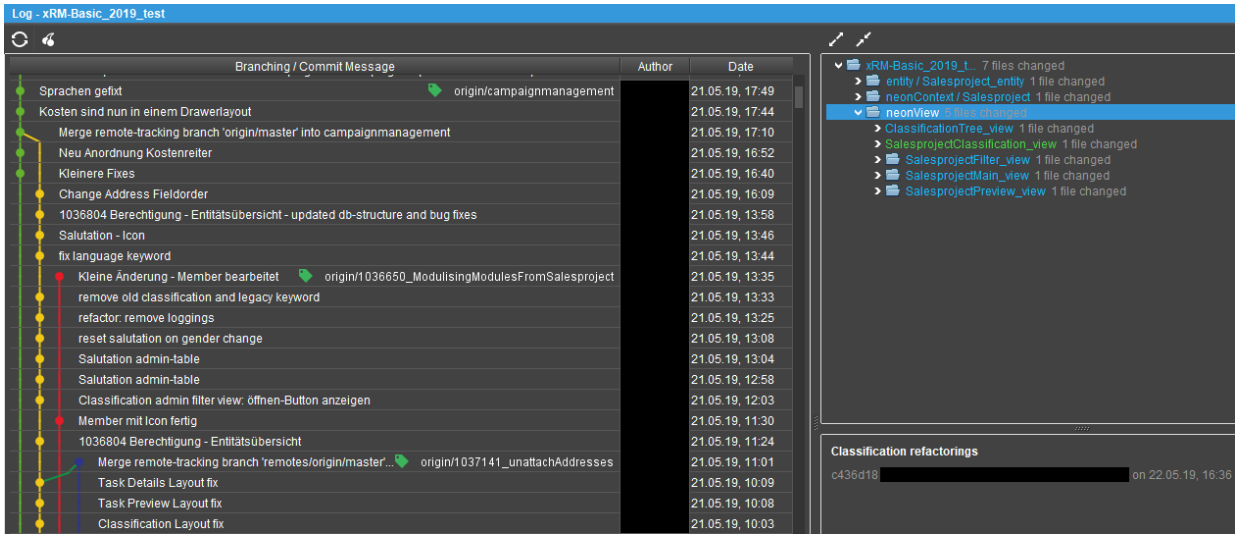
Here all changes made to the code are documented and made revertable. All of these changes are made either locally, but then only the local computer can see them in the history tab or via a Git commit. As soon as the commits are "pushed" (Option "Push"), they will show up on any computer using the same Git branch (as soon as the others have executed the Git option "Pull"), no matter where or by whom the changes were made.



"Revision", "User", and "Message" will be the Git user who committed the changes with the commit message as "Message". By using the arrows the changes can be reverted to the old state. The old state will always be shown on the left side.

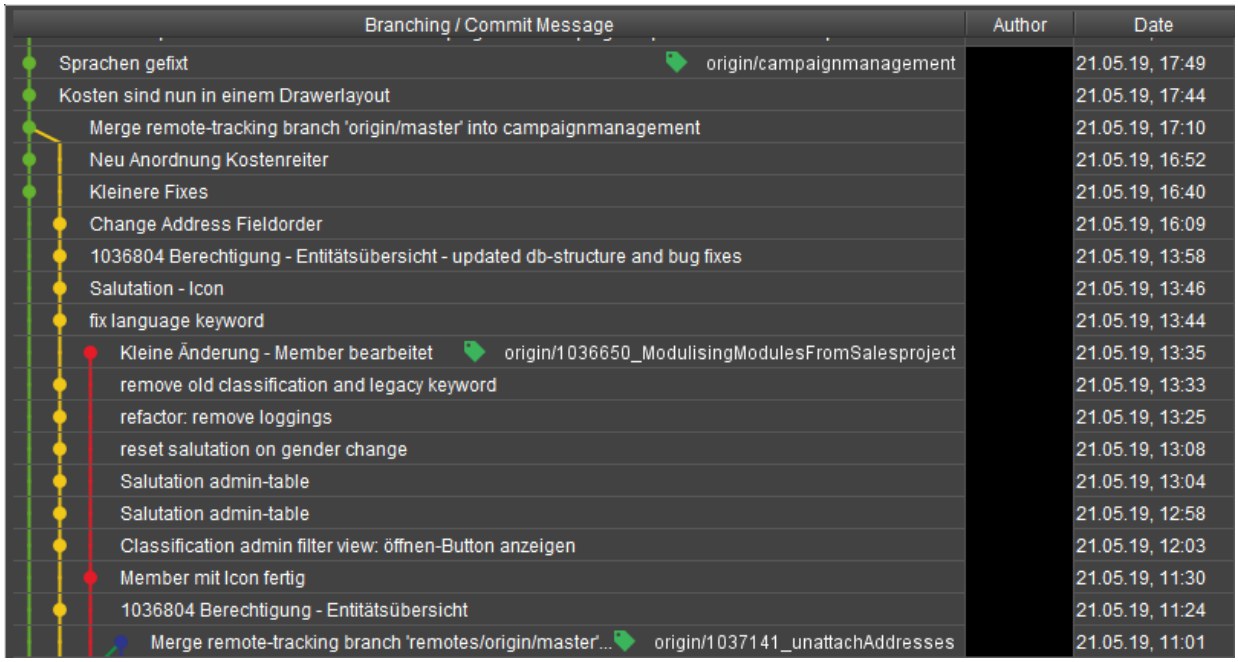
In addition to the file changes, the whole project history can be made visible with the button

This will open the following view:



Here are three major parts.

First:



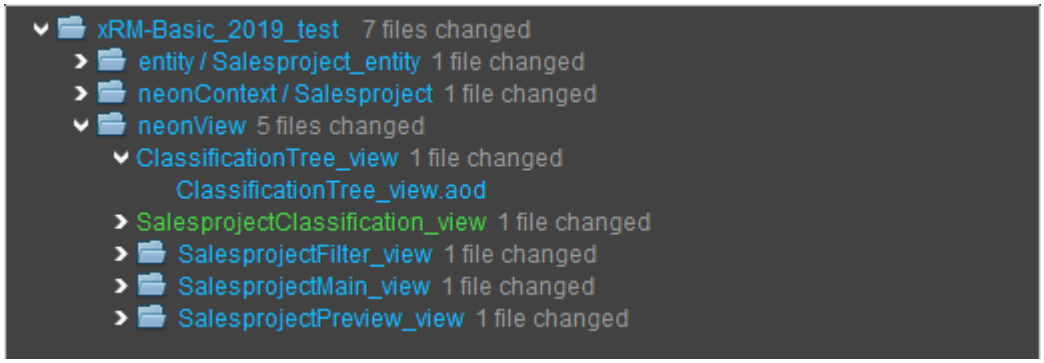
Every line on the left side represents a branch. At the moment when a branch gets merged into the main branch this will also happen in the picture (see the line "Merge remote tracking branch…), when a new branch is made this will also show the same but turned upside down. Each dot on the other hand shows a commit made in the project, together will the commit message next to it. The next line would show the username of the person which made the commit and the exact date the commit was made

on the right side.

If you want to revert to a old state, you have the option to do this here. Right click on a commit and select one of the following options.

Compare with HEAD	Shows all differences in all files from the local state to the state of the commit.
Reset current Branch to here	Resets the current branch! You lose all changes made when doing a hard reset.
Add Tag	Add a tag to a commit (the green flags)
Delete Tag	Delete a added tag
Cherry Pick	Select all files which should be merged into the current project with a dialogue on how to manage individual merge conflicts

Second:



This view shows all files where changes where made in a tree. With a double click another window will open which compares the current and old file.

Third:

Here the commiter and the commit message will be shown again.

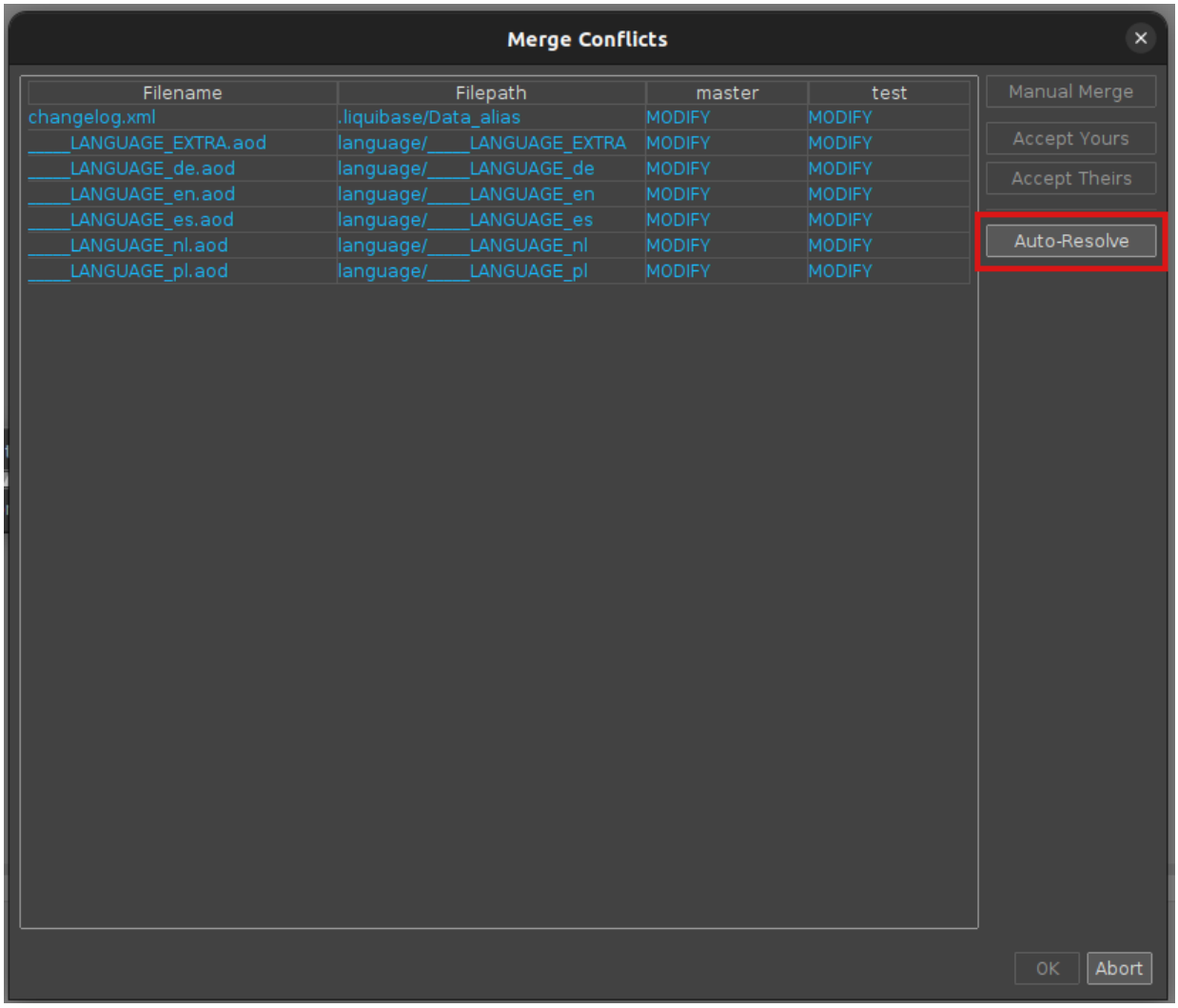
In addition to the toolbar windows, you may also access git for single files with a right click in the menu

"Git". Below the normal Git actions like pull/push there are some more options.

Diff local changes	Shows a side by side comparison between the current and former local file
Show local changes	Shows all locally changed files in a treetable
Show file history	Shows the files history in the same way as shown in "First" above
Stash local changes	Temporarily save your changes(stash) when you are not ready to commit yet (see Git documentation)
Un-stash changes	Un-stash your changes
Revert file(s)	Undo all changes in your open files
Resolve conflicts	Resolve merge conflicts which might appear when pulling a project

14.4.1. Auto-Merging

As mentioned in the table above you might encounter merge conflicts that have to be resolved. These conflicts are more likely to occur in Language and Liquibase files than any other file types. Therefore you can use the function "Auto-Resolve" in order to automatically merge these files. If there a conflicts that can’t be solved automatically, the normal merge dialog will appear and you have to manually merge.



At the moment you can expect the following auto-resolving to work with:

- Same changes
- Changes in which one side is contained in the other
- Word-based (instead of line-based) resolution of conflicts
- Imports: here the imports from both pages are merged together
- Liquibase: if different changelog files have been added, simply add all of them
- Language: in language files the conflicts are resolved if the "name" of the entries is different

14.5. Plugin Liquibase

14.5.1. Basics

Liquibase is an open-source database-independent library for tracking, managing, and applying database schema changes (see https://en.wikipedia.org/wiki/Liquibase). The Designer plugin encapsulates its original commandline calls and integrates its functionality into the ADITO Designer. Liquibase is very well-documented at https://docs.liquibase.com. Therefore, this manual contains only essential information.

Liquibase works by defining database structure and database content (datasets) in XML files, which are called "changelogs". Each changelog file contains

- XML "include" tags pointing to other (sub-)changelog files, or
- one or multiple so-called "changeSets", which XML nodes containing the actual configuration of database structure (e.g., create table, add column, etc.) or datasets (e.g., insert, update, or delete)

The core "job" of Liquibase is to convert these XML-based configuration into SQL commands - always starting with the master (top-level) "changelog.xml", which resides directly under the sub-nodes of project node "alias" (e.g., under alias > Data_alias) and points to "changelog.xml" files in sub-folders, which in turn point to the changelog files including the actual database change configurations.

As the changelog XML files are part of the ADITO project, they can also be versioned via Git. Liquibase is able to compare the current database with the definitions in the XML files, enabling to update the database accordingly.

The file names (not the content) of these XML files and the related folders must NOT contain any special characters, such as blanks, German "umlauts", French accents, Danish characters "å", "ø", etc. Otherwise, the Liquibase action might fail, resulting in an incomplete execution and a respective error message.

14.5.2. How to use Liquibase

The plugin adds a new menu item "Liquibase" to the context menu of the nodes under node "alias" (e.g. alias > Data_alias).

This item is a submenu, containing the following actions:

1. Update
   Checks the selected database against the configuration defined in the XML files and updates the database, if there are differences.

2. Drop All
   Deletes all tables of the database.
3. Drop All & Update
   Drops all database tables and (re-)creates all tables and datasets according to the XML files.

Depending of your project's size, these actions can take several minutes, resulting in a **"timeout" dialog** to be shown, possibly several times. Simply confirm these dialogs, in order to continue the execution of the Liquibase action. (If this bothers you, you can increase the timeout value in the "default options" of the Designer's startup file - see sub-chapter "Configuration" of chapter Startup.)

If, during your project's development phase, you have destroyed or confused your database tables or their datasets, you simply need to choose "Liquibase > Drop all & update", and in a few seconds your database will be reset to the original state, including the (demo) datasets that you may have defined in the XML files.

The Liquibase plugin does not provide any "undo" functionality. Once you have executed "Drop All" or "Drop All & Update", your complete database and all its datasets will be deleted permanently without any possibility to restore them!

14.5.2.1. Creating and naming changelog files

To create a new Liquibase changelog XML file, choose option "New" > "New Changelog" from the context menu of the nodes under node "alias" (e.g. alias > Data_alias). Name it according to the spelling conventions, e.g.,

- just "changelog.xml", if it is a file that only points to other (sub-)changelog files.
- according to their function, e.g., "insert_<whatIsToBeInserted>.xml", "alter_<whatIsToBeAltered>.xml", or "delete_<whatIsToBeDeleted>.xml", if database structure or datasets are created, altered, or deleted. Simply scan through the existing changelog files of the ADITO xRM project, in order to find out an suitable name.

Both types of XML files are commonly called "changelog files" - not only those that have the word "changelog" in their file name.

14.5.2.2. Creating folders

For a better overview, it is helpful to group the changelog files into several folders, with

- one file named "changelog.xml" in each folder, whose "include" tags point to each single changelog file in the folder (e.g., create_MyTable.xml, init_myTable.xml, etc.)

- one master file named "changelog.xml" on top-level, i.e., directly under "Data_alias". It contains "include" tags pointing to the (sub-)"changelog.xml" files of each folder.

This top-level "changelog.xml" file must be exactly named like this, as it is the hard-coded starting point of every Liquibase logic (except for "drop", which functions independently from the XML files and simply deletes the complete database). If this master file does not exist or is named differently from "changelog.xml", every Liquibase "update" logic will fail.

To create a new Liquibase folder, choose option "New" > "New Folder" from the context menu of the nodes under node "alias" (e.g. alias > Data_alias). Folder names should be self-explanatory, e.g., the name of the database table that is affected by the changeSets in the changelog files of this folder. You may also create sub-folders by choosing option "New" > "New Folder" from the context menu of a folder.

14.5.2.3. Liquibase folder structure

14.5.2.3.1. Common structure pattern

A common way to structure the Liquibase folders follows the following pattern:

(…)
<Name of Feature>
- data
- struct
- changelog.xml (sometimes also named init.xml)

- "struct" contains the changelog files that define the database tables for this feature
- "data" contains changelog files to insert datasets required for the initial status of this table or required new datasets in other tables (e.g., new keywords that are essential for this feature).
- "changelog.xml" points to every single changelog files in the "struct" and "data" folders.

14.5.2.3.2. Liquibase folder structure of the ADITO xRM project

In the ADITO xRM project, the changelog files are structured according to each single ADITO version. Existing Liquibase folders and changelog files are never changed, but whenever a new ADITO version is released, the folder structure is extended by new folders with new changelog files, specific for this new version. Even if, e.g., a database table is no longer required, its creation changelog file is not removed, but a delete changelog file is added. This means, whenever you execute Liquibase > Drop All & Update, the complete database history is worked through - starting from the database version of the first ADITO version ever, then through every single later version, until the current version. The advantage of this

approach is that, if required, you can re-create the database state specific for every single ADITO version of the past - simply by outcommenting the "include" tags of later versions in the master "changelog.xml" file.

To avoid failures or inconsitencies, it is strongly recommended to follow this principle for every further customizing-related database changes, be it related to structure or content (datasets): NEVER change or delete any of the existing Liquibase folder or changelog files, but ADD further folder or files according to your requirements.

14.5.3. Changelog XML structure

The basic enclosing element of a changelog file is a tag named "databaseChangeLog":

```
<?xml version="1.1" encoding="UTF-8" standalone="no"?>
<databaseChangeLog
xmlns="http://www.liquibase.org/xml/ns/dbchangelog"
                   xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xsi:schemaLocation="http://www.liquibase.org/xml/ns/dbchangelog
http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-3.6.xsd">

</databaseChangeLog>
```

In this XML structure you can add <changeSet> tags (then you call this file a "feature changelog"), or <include> tags pointing to other changelog files (then you call this file a "master changelog").

Example for adding a (sub-)master changelog to a master changelog:

```
<include relativeToChangelogFile="true"
file="basic/2021.0.3/changelog.xml"/>
```

Example for adding a feature changelog to the master changelog:

```
<include relativeToChangelogFile="true"
file="struct/create_product.xml"/>
```

Without the attribute relativeToChangelogFile="true" Liquibase is not able to locate the referenced changelog files correctly.

The file "changelog.xml" directly below the alias definition is called *the* master

changelog and contains only references (<include> tags) to (sub-)master changelog files, separately for every feature or ADITO version.

14.5.4. Feature changelog XML structure

Feature changelog files are Liquibase XML files that include one or multiple <changeSet> tags, nested in a <databaseChangeLog> tag. (Although, literally, a "changeSet" is an XML tag, some ADITO developers also call the whole XML file "changeset", if it is a feature changelog file including at least one changeSet.)

The basic structure of a feature changelog file is:

```
<?xml version="1.1" encoding="UTF-8" standalone="no"?>
<databaseChangeLog
xmlns="http://www.liquibase.org/xml/ns/dbchangelog"
                   xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xsi:schemaLocation="http://www.liquibase.org/xml/ns/dbchangelog
http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-3.6.xsd">
    <changeSet author="" id="">

    </changeSet>
</databaseChangeLog>
```

Here you have to manually set the author and id attributes.

- author should reflect who created and maintains this changeSet
- id has to be any *unique* string. Best you use a UUID generated by the Designer's integrated UUID Generator.

Within the <changeSet/> node you can add DDL nodes for structural changeSets or DML nodes for data changeSet.

It is not recommended to mix DDL and DML in a single changeSet.

The possible DDL and DML nodes are detailed below.

14.5.4.1. Data Definition Language (DDL)

14.5.4.1.1. CREATE

CREATE TABLE

E.g. createTable_info.xml

The following example of a create-file shows how to create a table with columntypes like CHAR, VARCHAR, NVARCHAR, SMALLINT, INT, DOUBLE, BOOLEAN, CLOB, NCLOB, LONGBLOB, DATETIME, DATE, TIME.

Primary key with multiple columns? You can't do that inside the createTable tag itself, but you can do it within the same changeSet as when the table is created.

```
<changeSet author="e.pollinger" id="1523697466364-1">
    <createTable tableName="INFO">
        <column name="INFOID" type="CHAR(36)">
            <constraints primaryKey="true"
primaryKeyName="PK_INFO_INFOID"/>
        </column>
        <column name="ORG_ID" type="CHAR(36)">
            <constraints foreignKeyName="FK_INFO_ORG_ID"
references="ORG(ORGID)"/>
        </column>
        <column name="USER_NEW" type="VARCHAR(50)">
            <constraints nullable="false"/>
        </column>
        <column name="NAME" type="NVARCHAR(255)">
            <constraints nullable="false" unique="true"
uniqueConstraintName="uqe_org_name"/>
        </column>
        <column name="TYPE" type="SMALLINT"/>"YYYY-MM-DD",
"hh:mm:ss" or "YYYY-MM-DDThh:mm:ss">
        <column name="AMOUNT" type="INT"/>
        <column name="PERCENT" type="DOUBLE"/>
        <column name="USEFUL" type="BOOLEAN"/>
        <column name="GOODTOKNOW" type="CLOB"/>
        <column name="INFOBIG" type="NCLOB"/>
        <column name="PICTURE" type="LONGBLOB"/>
        <column name="INFOMOMENT" type="DATETIME"/>
        <column name="INFODATE" type="DATE"/>
        <column name="INFOTIME" type="TIME"/>
        <column name="DATE_NEW" type="TIMESTAMP">
            <constraints nullable="false"/>
        </column>
    </createTable>
</changeSet>
```

If you want to have a composite constraint (like a composite primary key, unique constraint, ..) you can add the "constraints" tag with the "primaryKey" attribute to two columns.

When Liquibase is executed, Liquibase's data types (as included in the xml files) are mapped to data types proper to the specific database engine connected to the ADITO project. For example, Liquibase's data type NCLOB ((used for very large text fields) remains a NCLOB for Apache Derby databases, but is mapped to a LONGTEXT for MariaDB and MySQL, while in MicrosoftSQL it will be a NVARCHAR(MAX).
When customizing ADITO, you should always prefer the target data types of these Liquibase mappings, even if you do not use Liquibase itself.
You can find a list of the preferable data types, according to database system, in the document AID001 Coding Styles, chapter "Preferable data types".

CREATE INDEX

E.g. createIndex_org_name.xml

The structure of this file could look like this:

```
<changeSet author="j.brunner" id="1528803467864-1">
    <createIndex indexName="idx_org_name" tableName="ORG">
        <column name="NAME"/>
    </createIndex>
</changeSet>
```

By default, most DBMSs create all indices **as**cendingly. However, with respect to performance, the index should be **de**scending in specific cases. If, e.g., the datasets that are usually selected (= searched for) are relatively new, you could set a **de**scending index on the respective DATE column.

Here is an example, speeding-up the performance of the xRM project's Context "Activity":

```
(...)
<createIndex indexName="IDX_ACTIVITY_ENTRYDATE" tableName=
"ACTIVITY">
    <column name="ENTRYDATE" descending="true"/>
</createIndex>
(...)
```

As the client user usually searches for recent Activities, the DBMS finds the data faster, because a **de**scending index is set on column ENTRYDATE.

In particular, descending indices can speed-up the performance when set on columns that are date/time-related or holding sequential numbers.

14.5.4.1.2. ALTER

ADD COLUMN

E.g. addColumn_org_mailboxNumber.xml

The structure of this file could look like this:

```
<changeSet author="e.pollinger" id="1528803466371-1">
    <addColumn tableName="ORG">
        <column name="MAILBOXNUMBER" type="int"></column>
    </addColumn>
</changeSet>
```

Be careful if you want to set a "not nullable constraint" on the new column! If there are already records in the table, you'll not be able to add the column!

If you want to have a specific column order, have a closer look at the attribute "beforeColumn" on this website: https://www.liquibase.org/documentation/column.html

ADD PRIMARY KEY

E.g. addPrimaryKey_org_orgid.xml

The structure of this file could look like this:

```
<changeSet author="e.pollinger" id="1528703466371-1">
    <addPrimaryKey columnNames="ORGID" constraintName="pk_org"
tableName="ORG"/>
</changeSet>
```

You can also create a composite primary key by indicating two columns for the attribute "columnNames" (e.g. "ORGID, RELATIONID").

ADD FOREIGN KEY CONSTRAINT

E.g. addForeignKey_pers_org.xml

The structure of this file could look like this:

```
<changeSet author="e.pollinger" id="1528803466371-1">
    <addForeignKeyConstraint baseColumnNames="ORG_ID"
        baseTableName="PERS"
        constraintName="fk_pers_org"
```

```
        referencedColumnNames="ORGID"
        referencedTableName="ORG"/>
</changeSet>
```

If you have a composite primary key you want to refer to, you can indicate two columns at the attributes "baseColumnsNames" and "referencedColumnNames" (separated by a comma).

Be sure the two columns are the same type!

Here you could also add "onDelete" and "onUpdate" as a attribute.

ADD NOT NULL CONSTRAINT

E.g. addNotNull_org_name.xml

The structure of this file could look like this:

```
<changeSet author="e.pollinger" id="1528863466371-1">
    <addNotNullConstraint columnName="NAME" tableName="ORG"/>
</changeSet>
```

ADD UNIQUE CONSTRAINT

E.g. addUnique_org_customercode.xml

The structure of this file could look like this:

```
<changeSet author="e.pollinger" id="1528803466371-1">
    <addUniqueConstraint columnNames="CUSTOMERCODE" tableName="ORG"
constraintName="uqe_org_customercode"/>
</changeSet>
```

You can indicate two columns (separated by a comma) at the value of the attribute "columnNames" to get a composite unique constraint.

14.5.4.1.3. DROP

DROP TABLE

E.g. dropTable_org.xml

```
<changeSet author="j.brunner" id="1598813466379-1">
    <dropTable tableName="org"/>
</changeSet>
```

Watch out if there's a referential integrity constraint that refers to primary and unique keys in the dropped table. Then you won't be able to drop the table after all.

DROP COLUMN

E.g. dropcolumn_org_mailboxNumber.xml

The structure of this file could look like this:

```
<changeSet author="j.brunner" id="1528803466379-1">
    <dropColumn tableName="ORG" columnName="MAILBOXNUMBER"/>
</changeSet>
```

DROP FOREIGN KEY

E.g. dropForeignKey_pers_org_id.xml

The structure of this file could look like this:

```
<changeSet author="j.brunner" id="1526703466389-1">
    <dropForeignKeyConstraint baseTableName="PERS"
constraintName="fk_pers_org"/>
</changeSet>
```

DROP INDEX

E.g. dropIndex_org_name.xml

The structure of this file could look like this:

```
<changeSet author="j.brunner" id="1528803466389-1">
    <dropIndex indexName="idx_org_name" tableName="ORG"/>
</changeSet>
```

DROP NOT NULL CONTRAINT

E.g. dropNotNull_org_name.xml

The structure of this file could look like this:

```
<changeSet author="j.brunner" id="1528803466389-1">
    <dropNotNullConstraint columnName="type" tableName="testtable"/>
</changeSet>
```

DROP PRIMARY KEY

E.g. dropPrimaryKey_org_orgid.xml

The structure of this file could look like this:

```
<changeSet author="j.brunner" id="1528803466389-1">
    <dropPrimaryKey constraintName="PK_ORG_ORGID" tableName="ORG"/>
</changeSet>
```

14.5.4.1.4. PRECONDITIONS

Preconditions are changelog or changeset tags that control the execution of an update based on the state of the database.

Preconditions are typically used to:

- Document what assumptions the author of the changelog had when creating it.
- Enforce that those assumptions are not violated by users running the changelog.
- Perform data checks before performing an unrecoverable change such as dropTable.
- Control what changesets are run and not run based on the state of the database.

Example for using the tag "preConditions"

```
 <changeSet author="j.smith" id="d3ff85dc-278c-442a-8bde-
01e03b14ccb6">
    <preConditions onFail="MARK_RAN">
      <not>
        <tableExists tableName="MYTABLE" />
      </not>
    </preConditions>
    <createTable tableName="MYTABLE">
      <column name="MYTABLEID" type="CHAR(36)">
        <constraints nullable="false" primaryKey="true"
primaryKeyName="SQL0000000000-4191140c-017a-99e1-b89f-000014b5bcc5"
/>
      </column>
      <column name="MYCOLUMN1" type="VARCHAR(50)" />
      <column name="MYCOLUMN2" type="INTEGER" />
    </createTable>
  </changeSet>
```

In this example, tag "preConditions" makes sure that the changeSet can also be used for a Liquibase "update", without preceding "drop" (otherwise Liquibase would throw an exception after failing to create the same table a second time). The attributes and nested tags of this example mean:

- tableExists: Defines if the specified table exists in the database.
- not: You can apply conditional logic to preconditions using nestable <and>, <or>, and <not> tags. If no conditional tags are specified, the default value is AND.
- onFail: Controls what happens if the preconditions check fails.
- MARK_RAN: Skips over the *changeset* but marks it as executed. Continues with the *changelog*.

Find more information on preconditions on the official Liquibase documentation web site, see https://docs.liquibase.com/concepts/advanced/preconditions.html

14.5.4.1.5. RENAME

RENAME TABLE

E.g. renameTable_organisation.xml

The structure of this file could look like this:

```
<changeSet author="j.brunner" id="1723617466364-1">
    <renameTable newTableName="organisation" oldTableName="org"/>
</changeSet>
```

RENAME COLUMN

E.g. renameColumn_org_ccode.xml

The structure of this file could look like this:

```
<changeSet author="j.brunner" id="1723697466364-1">
        <renameColumn newColumnName="CCODE"
oldColumnName="CUSTOMERCODE" tableName="ORG"
columnDataType="INTEGER"/>
</changeSet>
```

The attribute `columnDataType` of the `renameColumn`-tag is required!

RENAME CONSTRAINT

It is not possible to rename a constraint. You have to drop the constraint and make a new one.

14.5.4.2. Data Manipulation Language (DML)

The following file cutouts do just contain the `<changeSet>`-part. Do not forget to add the `<?xml>`- and `<databaseChangeLog>`-Tags!

All those files need to be in the "data" folder.

14.5.4.2.1. INSERT

E.g. insert_org_data1.xml

The structure of this file could look like this:

```
<changeSet author="j.brunner" id="1528875814576-1">
    <insert tableName="INFOO">
        <column name="INFOID" value="8796528b-c053-557a-b8d9-
11063d6a50ae"/>
        <column name="ORG_ID" value="198b8a8b-c053-447a-bc69-
17963b6a60ae"/>
        <column name="USER_NEW" value="j.brunner"/>
        <column name="NAME" value="Infoblatt"/>
        <column name="TYPE" valueNumeric="1"/>
        <column name="AMOUNT" valueNumeric="16"/>
        <column name="PERCENT" valueNumeric="15.2"/>
        <column name="UNIQUE" valueBoolean="1"/>
        <column name="GOODTOKNOW"
valueClobFile="data\clob\example.txt" />
        <column name="GOODTOKNOW2"
valueClobFile="C:\Users\J.Brunner\Documents\AditoProjects\xRM-Basic

5.1\others\db_changes\data\clob\example.txt"/>
        <column name="PICTURE" valueBlobFile="data\blob\DS1.png"/>
        <column name="INFOMOMENT" valueDate="2007-08-09T13:14:15"/>
        <column name="INFODATE" valueDate="2007-08-09"/>
        <column name="DATE_NEW" valueDate="2007-08-09T13:14:15"/>
    </insert>
</changeSet>
```

Date and/or Time value to set the column tag `valueDate`. The value is specified in one of the following forms: "YYYY-MM-DD", "hh:mm:ss" or "YYYY-MM-DDThh:mm:ss".

14.5.4.2.2. UPDATE

E.g. update_org_data1.xml

The structure of this file could look like this:

```
<changeSet author="j.brunner" id="1528875814530-1">
    <update tableName="ORG">
```

```
        <column name="type" valueNumeric="1"/>
        <where>ORGID = '29990305-4ac6-4876-94b9-
0300ce6cefaa'</where>
    </update>
</changeSet>
```

14.5.4.2.3. DELETE

E.g. delete_org_data1.xml

The structure of this file could look like this:

```
<changeSet author="j.brunner" id="1528875814544-1">
    <delete tableName="ORG">
        <where>ORGID = '29990305-4ac6-4876-94b9-
0300ce6cefaa'</where>
    </delete>
</changeSet>
```

If you try to delete a record, whose primary key is used in another table, the statement will probably not be executed because of the foreign key contraint and the fact, that onDelete is "RESTRICT" on default. That doesn't allow you to delete records whose primary keys are as a foreign key in another table.

You could set the value of the attribute `onDelete` of the Tag `<addForeignKeyConstraint>` "CASCADE", "SET NULL", "SET DEFAULT", "RESTRICT" or "NO ACTION"

CLEAR TABLE

As there is no Truncate in Liquibase, you simply have to run a delete-skript without a where-tag. E.g.:

```
    <changeSet author="j.brunner" id="1528995814544-1">
        <delete tableName="org"/>
    </changeSet>
```

14.5.4.3. PREPARED STATEMENTS

If you want to execute a delete/an update with the where-clause as prepared statement, the XML could look like that:

```
<changeSet author="j.brunner" id="1528875814830-1">
    <delete tableName="ORG">
```

```
        <where>ORGID = ?</where>
        <whereParams>
            <param value="29990305-4ac6-4876-94b9-0300ce5cefaa"/>
        </whereParams>
    </delete>
</changeSet>
```

Update works analog to delete

14.5.5. ROLLBACKS

Liquibase allows you to undo changes you have made to your database, either automatically or via custom rollback SQL. Rollback support is available in command_line, Ant, and Maven (in our case: **command_line**).

Many refactorings such as “create table”, “rename column”, and “add column” can automatically create rollback statements. If your change log contains only statements that fit into this category, your rollback commands will be generated automatically.

Other refactorings such as “drop table” and “insert data” have no corresponding rollback commands that can be automatically generated. In these cases, and cases where you want to override the default generated rollback commands, you can specify the rollback commands via the tag within the changeSet tag. If you do not want anything done to undo a change in rollback mode, use an empty tag.

Auto Rollback	No Auto Rollback
RenameColumn	Insert
RenameTable	Update
CreateTable	Delete
CreateIndex	DropColumn
AddColumn	DropTable
AddPrimaryKey	DropIndex
AddForeignKey	DropPrimaryKey
AddNotNull	DropForeignKey

Auto Rollback	No Auto Rollback
AddUniqueConstraint	DropUniqueConstraint
DropNotNullConstraint	

So if you create a statement where no Auto Rollback can be created, you have to proceed like that:

For example if you want to drop a table, you can reference to the changeSet which originally created the statement:

```
<changeSet id="1523697466364-2" author="jbrunner">
    <dropTable tableName="info"/>
    <rollback changeSetId="1523697466364-1"
changeSetAuthor="jbrunner"/>
</changeSet>
```

So *1523697466364-1* is the id of the "create" changeSet of the table *info*.

If you create an "insert" changeSet, you put a "delete" changeSet in a rollback tag and vice versa.

If you create a "dropColumn" changeSet, you put the "addColumn" changeSet in a rollback tag.

If you create a "dropTable" changeSet, you put the "createTable" changeSet in a rollback tag.

If you create a "dropIndex" changeSet, you put the "createTable" changeSet in a rollback tag.

If you create a "dropPrimaryKey" changeSet, you put the "addPrimaryKey" changeSet in a rollback tag.

If you create a "dropForeignKey" changeSet, you put the "addForeignKey" changeSet in a rollback tag.

If you create a "dropUniqueConstraint" changeSet, you put the "addUniqueConstraint" changeSet in a rollback tag.

If there is no changeSet which could reverse your new changeSet or you create a "update" ChangeSet, just reverse the statement by your own. Set it to the value it has been before.

For example an "update" changeSet:

```
<changeSet author="j.brunner" id="1528875814530-1">
        <update tableName="ORG">
            <column name="type" valueNumeric="1"/>
            <where>ORGID = '29990305-4ac6-4876-94b9-
```

```
0300ce6cefaa'</where>
        </update>
        <rollback>
            <update tableName="ORG">
                <column name="type" valueNumeric="0"/>
                <where>ORGID = '29990305-4ac6-4876-94b9-
0300ce6cefaa'</where>
            </update>
        </rollback>
    </changeSet>
```

Or an dropPrimaryKey-changeSet:

```
<changeSet author="j.brunner" id="1528803466389-1">
    <dropPrimaryKey constraintName="PK_ORG_ORGID" tableName="ORG"/>
    <rollback>
        <addPrimaryKey columnNames="ORGID"
constraintName="PK_ORG_ORGID" tableName="ORG"/>
    </rollback>
</changeSet>
```

14.5.6. BLOB/CLOB

If you want to insert data in a table and one of the columns are type "CLOB" or "BLOB", you need to store the value of the data in a separate file. Therefore add two folders ("blob" and "clob") in you "data" folder and store your pictures, textfiles, etc. in these.

In your insert-statement you need to indicate the file path as a relative or absolute path for the "BLOB"/"CLOB" data. An example for both:

Relative:

```
valueClobFile="data\clob\example.txt"
```

Absolute:

```
valueClobFile="C:\Users\J.Brunner\Documents\AditoProjects\xRM-Basic
5.1\others\db_changes\data\clob\example.txt"
```

14.5.7. XML special characters

Adding control characters ('<', '>', '"', '"', '&') into xml data can cause the parser to miss understand the resulting data. The solution is to escape the control characters so that the parser can interpret them

correctly as data, and not confuse them for markup.

Characters	Escape String
<	&lt;
>	&gt;
&	&amp;
"	&quot;
'	&apos;

Example:

Data	In XML
He said "OK"	attributeName="He said "OK""

For any further, more complicated examples, google "escaping xml data examples". You'll find plenty of them.

14.5.8. Enabling demo data

The Liquibase files of the ADITO xRM project include the configuration of demo data, e.g., example persons or companies. However, by default, these demo datasets are excluded from any Liquibase operations, by being commented out in the master "changelog.xml" file:

changelog.xml (under alias > Data_alias) with deactivated demo data

```
<databaseChangeLog>
...
    <!--enable this only when you definetly want to overwrite the existing data with demo records:-->
    <!--<include relativeToChangelogFile="true" file="basic/_demoData/changelog.xml" context="example"/>-->
</databaseChangeLog>
```

As you can already read in the remark of the xml code, activating (uncommenting) the demo data will result in a **complete loss of any productive data** - **even if you only choose option "Liquibase - update"** (without "drop all"). Therefore, this xml line must NEVER be activated (= uncommented) in a productive system or whenever

> you have entered own data that must not be deleted.

As we want to use the demo data for our carpool project, we uncomment the corresponding xml line:

changelog.xml (under alias > Data_alias) with activated demo data

```
...
    <!--enable this only when you definetly want to overwrite the existing data with demo records:-->
    <include relativeToChangelogFile="true" file="basic/_demoData/changelog.xml" context="example"/>
</databaseChangeLog>
```

But do not forget to deactivate (comment out) this line again, as soon as you have performed the Liquibase update. Otherwise data loss is possible (see warning above).

14.5.9. Create Liquibase files automatically

The ADITO Designer includes some very helpful functionality that simplifies the generation of new Liquibase files and thus reduces the probability of "copy & paste" mistakes.

14.5.9.1. Create empty changelog

You can create a new changeLog file with the "author" attribute and a UUID already set by right-clicking on a Liquibase folder and choosing "New with Blueprint" > "Create empty changeLog". A dialog will then be opened, requesting the name of the new file and (optionally) the name of the author.

14.5.9.2. Create changelog with new DB table

You can create a new changeLog file for defining a new table, including primary key column, "author" attribute and UUID by right-clicking on a Liquibase folder and choosing "New with Blueprint" > "Create changeLog with new DB table". A dialog will then be opened, requesting the name of

- new file
- author (optional)
- table
- primary key column (optional)
- standard column (USER_NEW, USER_EDIT, DATE_NEW, DATE_EDIT; optional)

14.5.9.3. Changelog to SQL

If you want to get the SQL statements

- that Liquibase generates on the basis of a specific changelog XML file,

- right-click on that file and choose "Liquibase" > "Generate Update SQL".
- A dialog will open requesting you to select the corresponding database alias (e.g. "…Data_alias…").
- Section "Available contexts": In addition, you can (if present) check checkboxes to include those tags having a specific "context" attribute, e.g., "example":

```
(...)
    <include relativeToChangelogFile="true" file="basic/2021.0.1/changelog.xml"/>
    <include relativeToChangelogFile="true" file="basic/2021.0.2/changelog.xml"/>
    <include relativeToChangelogFile="true" file="basic/2021.0.3/changelog.xml"/>

    <include relativeToChangelogFile="true" file="basic/_demoData/changelog.xml" context="example"/>
(...)
```

If, in this case, the checkbox "Available contexts > example" is not checked, then the line

```
    <include relativeToChangelogFile="true" file="basic/_demoData/changelog.xml" context="example"/>
```

would be skipped, when a Liquibase "Update" action is executed.

- After selecting and clicking "OK", a popup window shows you the requested SQL statements.

- for rolling back the "Update" actions related to a specific changelog XML file, right-click on that file and choose "Liquibase" > "Generate Future Rollbqack SQL". A dialog will open requesting you to select the corresponding database alias (e.g. "…Data_alias…"). After selecting and clicking "OK", a popup window shows you the requested SQL rollback statements.

14.5.9.4. DB to changelog

In earlier chapters, you have already learned how to use Liquibase XML files, in order to create database structure (tables and columns) and insert data into tables.

In practice, it can sometimes be easier to do it the other way round:

- You **first** create the database structure manually (via the ADITO database editor, via the Alias Definition, or directly via SQL commands) and **then** you create the Liquibase files. The latter can be done automatically.
- In the same way, you can also **first** insert data (example data etc.) manually (via the ADITO client or directly via SQL commands) and **then** you create the Liquibase files. The latter can also be done automatically.

You can easily generate changelog XML files from an existing database table: Open the ADITO database

editor (e.g., via double click on system > default > Data_alias > ADITO > Tables) and right-click on the respective database table. In the context menu, choose "Generate changelog". A dialog will appear that allows you to select path and name of the changelog file to generate, fill in the author's name and, if required, select options to include (as required for some database management systems) catalog or schema. In section "Types" check

- "Table structure", if you want a changelog file for creating the database table's structure, i.e., its name, columns (with data type), contraints, indices, etc.
- "Data", if you want a changelog file for inserting the database table's datasets.
- You can combine both types by checking both checkboxes. Then the database table will first be generated and then be filled with datasets.

Example of a Liquibase XML file automatically generated on the basis of an existing database table

```
<?xml version="1.0" encoding="UTF-8"?>
<databaseChangeLog xmlns="http://www.liquibase.org/xml/ns/dbchangelog" xmlns:ext="http://www.liquibase.org/xml/ns/dbchangelog-ext"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.liquibase.org/xml/ns/dbchangelog-ext
http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-ext.xsd http://www.liquibase.org/xml/ns/dbchangelog
http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-4.1.xsd">
  <changeSet author="j.smith" id="d3ff85dc-278c-442a-8bde-01e03b14ccb6">
    <preConditions onFail="MARK_RAN">
      <not>
        <tableExists tableName="MYTABLE" />
      </not>
    </preConditions>
    <createTable tableName="MYTABLE">
      <column name="MYTABLEID" type="CHAR(36)">
        <constraints nullable="false" primaryKey="true" primaryKeyName="SQL0000000000-4191140c-017a-99e1-b89f-000014b5bcc5" />
      </column>
      <column name="MYCOLUMN1" type="VARCHAR(50)" />
      <column name="MYCOLUMN2" type="INTEGER" />
    </createTable>
  </changeSet>
</databaseChangeLog>
```

As you can see, this XML file includes a tag called "preConditions" that makes sure that the file can also be used for a Liquibase "update", without preceding "drop" (otherwise Liquibase would throw an exception after failing to create the same table a second time). The attributes and nested tags of this tag mean:

- tableExists: Defines if the specified table exists in the database.
- not: You can apply conditional logic to preconditions using nestable <and>, <or>, and <not> tags. If no conditional tags are specified, the default value is AND.
- onFail: Controls what happens if the preconditions check fails.
- MARK_RAN: Skips over the *changeset* but marks it as executed. Continues with the *changelog*.

Find more information on preconditions on the official Liquibase documentation web site, see https://docs.liquibase.com/concepts/advanced/preconditions.html

See also chapter PRECONDITIONS.

14.5.10. Liquibase functions outside the Designer

As already mentioned, you can find a detailed documentation of all Liquibase functions on the developer's web site, see https://docs.liquibase.com/ . Besides the functionality already integrated in the Designer, there are several further features that are available outside the Designer, to be executed via the command line. For example, it is possible to generate a "diff" between different database snapshots (see Liquibase web site). This requires Liquibase to be installed from the developer's website, https://www.liquibase.org/ . (Having the ADITO plugin installed is *not* enough!)

14.5.11. Liquibase with audit and offline synchronisation

Ignoring this chapter can result in data inconsistency!

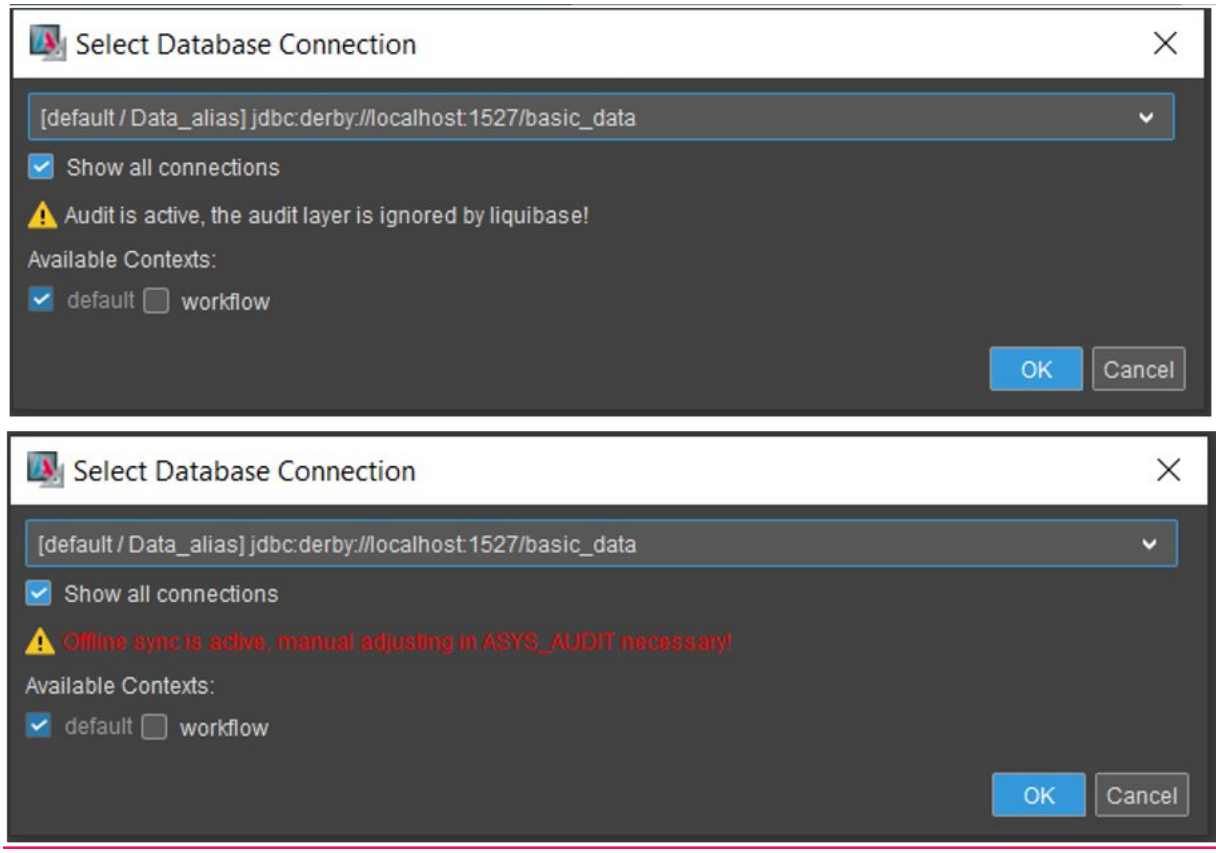
If you execute a Liquibase function on a database table that is included in

- audit with or without
- offline synchronisation

warning messages will be shown, which need to be confirmed with "OK", before Liquibase will actually be executed. (Alternatively, you may choose "Cancel".)

The reason of these warnings is that Liquibase does not communicate its database changes to ADITO's audit layer - neither changes of structure, nor changes of content (datasets). Therefore, if audit is active, a warning message will be shown in white font color, meaning you will have an incomplete audit protocol unless you perform manual adjustments of table ASYS_AUDIT subsequently.

If, in addition to audit, the offline sychronization is active, then subsequent adjustments of ASYS_AUDIT are compulsory. If you miss to do this, you will encounter data inconsistencies on devices having been offline during the execution of Liquibase. Therefore, a warning message will be shown in red font color.



14.5.12. Using Liquibase files for multiple database types

All Liquibase files are database type-specific, i.e., you cannot 1:1 use files created for, e.g., a Derby database also for an Oracle database - and vice versa. However, in many cases, only minor changes of the xml content is required, in order to meet the specifications of other database types (e.g., change the name of the column type).

14.5.13. Liquibase troubleshooting

14.5.13.1. Updated database structure is not accessible

Whenever you change the structure of the database (e.g., add a new table or a new column), you need to clear the server's cache (which has been implemented in order to optimize the system's performance):

- If you have started the server in the Designer,
  - click the garbage bin button in the vertical button bar in the left part of the output window "Server:default" or
  - right-click into the same window and choose "Clear cache" (do not mistake this option with another option called "Clear", which only clears the displayed log entries).

- Otherwise, clear the cache via the ADITO Web Manager (see button "Clear cache" in menu "Overview"; find more details on the Web Manager in the Operating Manual).

If you miss to clear the cache, the structure (new table, new column, etc.) will not be accessible by your Entities, but you will get an error message.

14.5.13.2. Handling Liquibase failures

If the execution of a Liquibase "Update" action fails, please inspect the error message in the ADITO Designer's log (not the ADITO server log!). In most cases you will find hints about the problem, e.g., messages like

```
liquibase.exception.ChangeLogParseException: The file .liquibase/Data_alias/basic/myFolder/mySubfolder/myChangelogFile.xml was not
found
```

(which, in this example, most probably means that a file named "myChangelogFile.xml" was referenced in a "changelog.xml" file, but a file of that name does not exist.)

What happens to the database state, if a Liquibase "Update" action fails while processing a specific changeLog file?

- Liquibase attempts to execute each changeSet in a transaction that is committed at the end, or rolled back if there is an error. Some databases will auto-commit statements which interferes with this transaction setup and could lead to an unexpected database state. Therefore, it is best practice to have just one change per changeSet unless there is a group of non-auto-committing changes that you want to apply as a transaction such as inserting data (see https://docs.liquibase.com/concepts/basic/changeset.html).
- changeSets (meaning changeSet nodes in a changeLog file) preceding the erroneous changeSets are NOT rolled back.
- changelog files that have been executed before the changelog file including the errorours changeSet are NOT rolled back.
- Subsequent changelog files are not executed.

Thus, your database will be in an incomplete, not-functioning condition, if a Liquibase "Update" actions fails - especially after action "Drop All & Update". In most cases, you can fix this by

- tracing and repairing the errorours changelog file;
- executing "Liquibase > "Update" - "Drop" should not be required, because Liquibase will automatically detect which of the changeSets have already been executed before and which of them still need to be executed.

There is an optional attribute available for the changeSet tag: "failOnError". This defines whether the whole Liquibase "Update" action will fail (= stop) if an error occurs while executing the changeSet. (Default value is true.) Example:

```
<changeSet id="3" author="j.smith" failOnError="false" dbms="oracle">
    (...myChanges...)
</changeSet>
```

This can be helpful if you know in advance that in some cases a specific changeSet might fail - without that being a reason to stop the whole Liquibase "Update" action: If the statement resulting from the changeSet throws an error, the changeSet will still be marked as ran, and the rest of the "Update" action will continue. The downside of this approach is that the changeSet will also be marked as ran and continue if it fails for some reasons you had not expected (e.g., bad permissions, connection failure, invalid SQL, etc.). Therefore, the more accurate approach is to prevent Liquibase failures by defining so-called **preconditions**: Preconditions are changeSet tags that control the execution of a Liquibase "Update" action, based on the state of the database. Find more information at https://docs.liquibase.com/concepts/advanced/preconditions.html

14.6. Plugin Cloud Support

ADITO's cloud plugin simplifies the connection of the ADITO Designer to an ADITO cloud system, as well as the customizing of it. If the cloud system is integrated in ADITO's Self-Service Portal (SSP) and you have access to the SSP, then the approach is very simple - if not, you have to do several configuration steps.

Prerequisite is that you have installed the Plugin either at the initial Designer start or via menu Tools > Plugins > Available Plugins > Cloud Support > Install.

For performance reasons, we recommend you not to open more than one cloud system in the Designer at the same time.

If you only want to exchange the default background image or logo of the login web page, you can do this directly in the SSP, via Action "Change login screen" (see Action button in your system's PreviewView). No further customizing required.

14.6.1. With SSP access

14.6.1.1. Connection via "load cloud system"

If the cloud system is integrated in ADITO's Self-Service Portal (SSP) and you have access to the SSP, you can simply download the corresponding project from the cloud and immediately start to customize it and deploy your changes. Just proceed as follows:

- File > New Project > load cloud system
- Enter your credentials (username and passwort) for accessing the SSP - these are the same as those for accessing the SSP via the web client (https://ssp.adito.cloud/client/Home/full). Actually, the credentials are identical with credentials for accessing ADITO's GitLab.
- If you click on the "refresh" button to the right of the password field, all SSP cloud systems that you have access to are shown in the large field below:

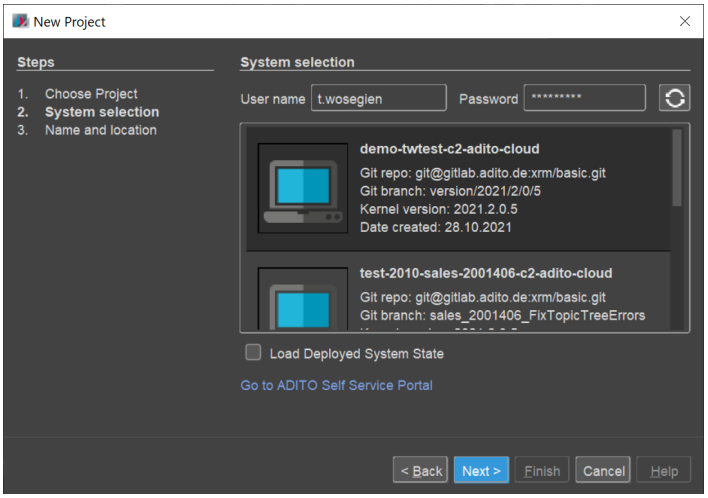


Figure 17. Selection box of available cloud systems

- Choose the cloud system that you want to connect with.
- By checking/unchecking the checkbox "Load Deployed System State", you can choose the required state of the new local project that you want to load from the cloud system:
    - Leave the checkbox empty, if you want to checkout the Git branch with which the system was originally set up.
    - Select "Load Deployed System State", if you want to have the system "as it is" on the server. This will also check out the Git branch that was active when the last deploy was made.
    - Technical details: When a user performs a deploy, the currently active Git branch is stored in the system database. When the "Load Deployed System State" checkbox is selected, that information is read from the database and the branch is checked out. Afterwards, the files are overridden with the files of the system database (ASYS_SYSTEM etc.). This way, changes that were deployed but not yet committed to the Git repository can be retrieved.
- If required, change the default project name or the default project location.
- Click "Finish"
- A popup dialog will appear, in which you must enter the location of the file holding your SSH key (RSA private key, usually a file named "id_rsa") as well as the corresponding password and then press "Save". Then, these credentials are saved (cashed), and you will not be prompted to enter them a second time.
- Possibly, a warning dialog appears that tells you that the SSH fingerprint of the server is unknown. As this is probably your first time connecting to the server you can safely ignore this warning.
- The download of the project of your cloud system will take several minutes, depending on the speed of your internet connection.

- After the download is finished, the project will appear in the "Projects" window of the Designer. This project contains the same data as the corresponding cloud system (i.e., identical to the content of the cloud system's system database table ASYS_SYSTEM).
- The connection of your project with the cloud system is established via a "tunnel": Navigate to your system (e.g., system > default) and choose "Open Tunnels" in the context menu. (Sometimes, option "Open Tunnels" is only active, if you open the context menu a second time.)
- Watch the tunnel icon to the left of the system name: Its color will change to yellow (= tunnel connection in progress) and then to green (= tunnel connection successfully established).
- See also appendix Tunneling.
- Possibly, a warning dialog appears (see above).
- Now you can start with your customizing work. You are always connected to the cloud - no local server is running.
- In order to see the server log in the Designer's window "Output", choose "Cloud Server - <system name>" (e.g., "Cloud Server - default") from the combobox in middle of the button bar and press the green "triangle" button to the right of it. After a few seconds, you can read a confirmation of the connection in a sub-window of window "Output", e.g., titeled "Cloud Server: default").
- If you click on the "Deploy" button in the button bar, the changes of your local project will be deployed into the cloud system. If you do this for the first time, the deploy dialog will also show changes that you have not caused. If you double-click on them, you can see that these are tiny changes (e.g., that an empty value process has been reset to default state) caused by internal automatisms, which you can ignore - just include them in the deploy, then they will not appear in the next deploy.
- You can open the client browser, connected to your cloud system, by choosing "Web Client (Neon) Cloud - <system name>" (e.g., "Web Client (Neon) Cloud - default") from the same combobox. If this URL leads to an error, remove the port information ":8443" from the URL in the address line of your browser and then call the URL in your browser again. Then you will reach the ADITO login page. Here, enter "admin" as user and below the admin password of your SSP system: In the SSP, click on your system and then press button "Copy admin password" in the PreviewView. Then the password is in the clipboard, from which you can simply paste it into the password window.

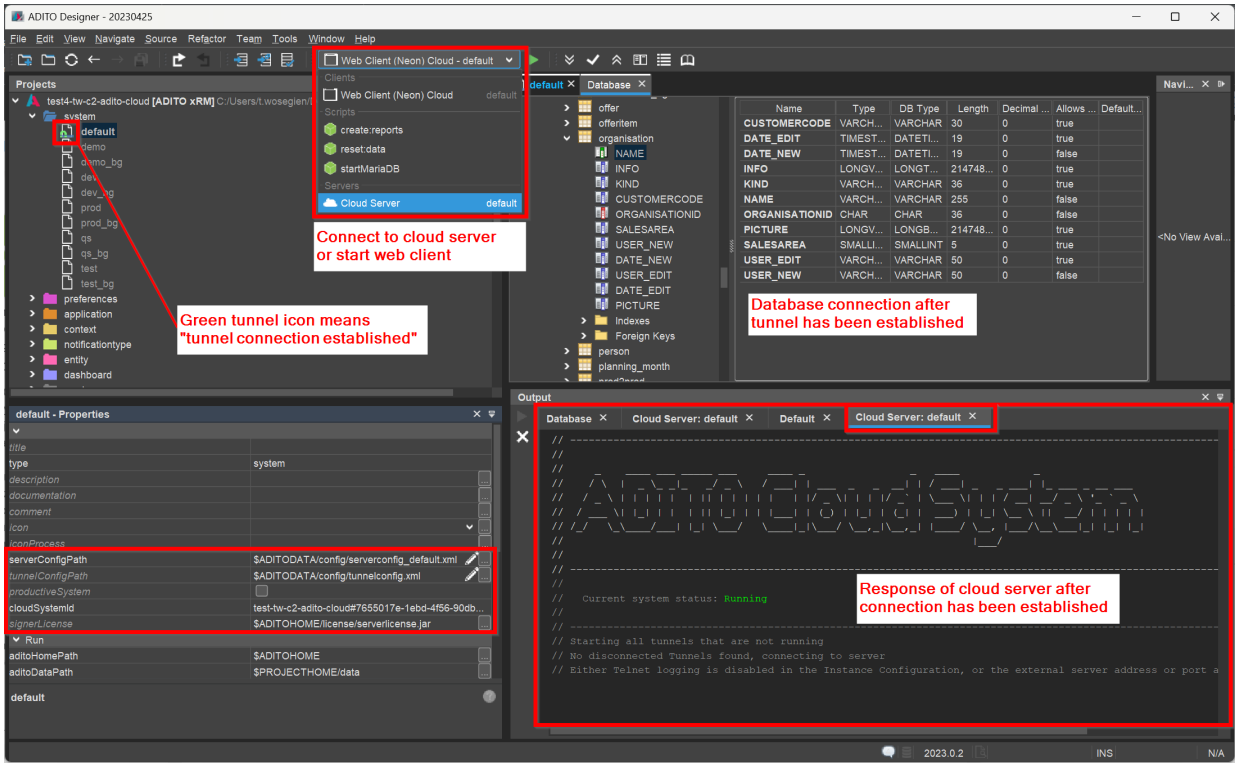


Figure 18. Overview of functionality provided by plugin Cloud Support

14.6.1.2. Connecting a local project to a cloud system

Even if you have not downloaded the cloud system into your designer (see previous chapter), you can connect any local project to any cloud system, simply by setting property "cloudSystemId" in the properties of your system (e.g., system > default) and establishing a tunnel connection. (See also appendix Tunneling.)

The cloud system ID cannot be entered directly, but you need to use option "Link to Cloud System" in the context menu of your system (e.g., system > default). Then, proceed similar to "load cloud system" (see previous chapter):

- Enter your credentials (username and password) for accessing the SSP - these are the same as those for accessing the SSP via the web client (https://ssp.adito.cloud/client/Home/full). Actually, these credentials are identical with credentials for accessing ADITO's GitLab.
- If you click on the "refresh" button to the right of the password field, all SSP cloud systems that you have access to are shown.

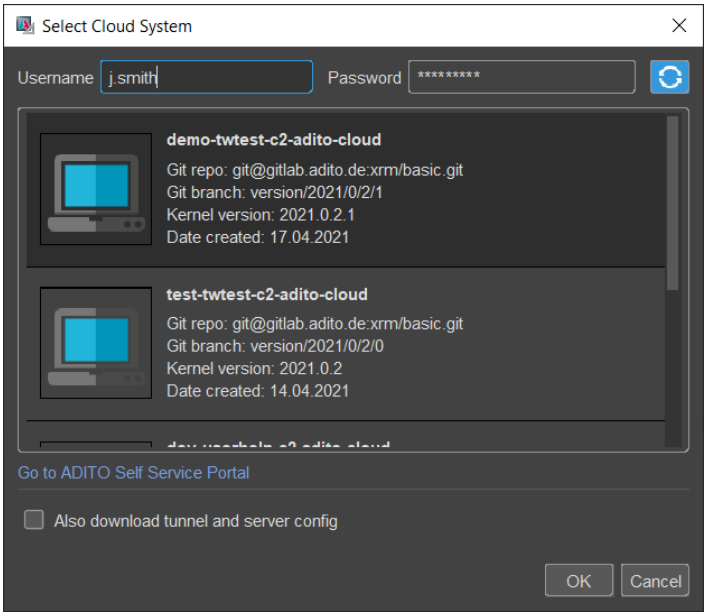


Figure 19. Selection box of available cloud systems, including the option to download the config files

- Choose the cloud system that you want to connect with.
- If you have not yet downloaded and referenced the cloud system's config files in your system's properties (e.g., system > default), check option "Also download tunnel and server config".
- After clicking the OK button
    - The cloud system ID will then automatically be inserted in property "cloudSystemId". This means, your local project is now linked to this cloud system.
    - The 2 config files (tunnelconfig and serverconfig) will then automatically be downloaded into the project folder "...\data\config". (If there are already config files in this folder, you can overwrite them on request. But if you want to continue working with a local system at a later time, you should backup the local system's serverconfig file to some other location first, before overwriting it.) Make sure that the downloaded files are referenced correctly in property "serverConfigPath", and "tunnelConfigPath", respectively (see the properties of your system, e.g., system > default).
- Now you can establish the required tunnel connection - please proceed as described in the previous chapter. (See also appendix Tunneling.)
- You can view the cloud system's server log in the Designer's "Output" window, if you choose "Cloud Server - default" from the combobox in middle of the button bar and press the green "triangle" button to the right of it.
- Depending on your system configuration (serverconfig file), any deploy will, from now on, not be directed to your local database, but to the database of the cloud system.

Any deploy will overwrite the current condition (database content) of your cloud system without any warning. There are no plausibility checks or similar security mechanisms. This means, if your cloud system contains unversioned/unbackuped data, these will be completely lost after the deploy, without any "undo" option. Therefore, be very careful when selecting the correct cloud system to connect with.

Of course, it is possible to revert the connection to your cloud system:

To disconnect from the cloud system

- temporarily:
  - Choose option "Close opened tunnels" in the context menu of your system (e.g., system > default).
  - You may choose "Open Tunnels" later, if you want to re-establish the connection.
- permanently:
  - Remove the cloud system ID by right-clicking on property "cloudSystemId" (see the properties of your system, e.g., system > default) and choosing "Restore Default Value" in the context menu.
  - If you want to continue working with a local system, remember to reference your local system's "serverconfig" file in property "serverConfigPath" again (see properties of your system, e.g., system > default).

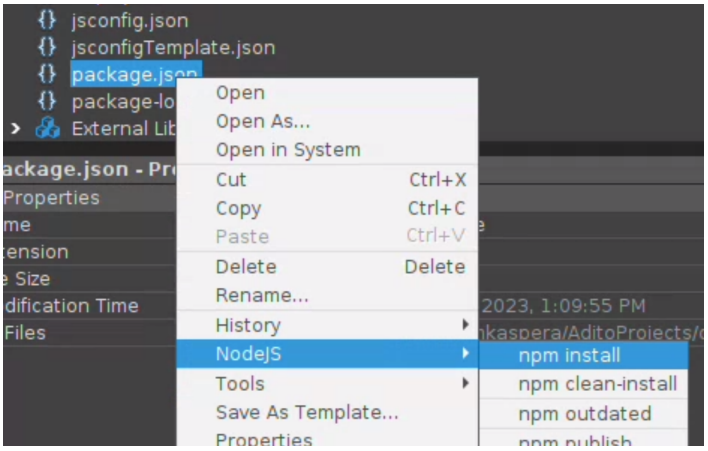
14.6.2. Without SSP access

If you have no access to ADITO's Self-Service Portal (SSP) you have to do several configuration steps. Please contact ADITO for further instructions.

14.7. Plugin NodeJS & TypeScript

The plugin NodeJS & TypeScript provides

- syntax highlighting, autocomplete, and go-to actions in js/ts files
- ScanServices and hints in the context of js/ts files
- npm install/npm clean install actions executed on package.json



- comment/uncomment actions

If the plugin has been installed and set active (and the ADITO Designer has been restarted), new buttons will appear on top of the code editor:

Check the buttons' tooltips to understand their function.

Additional information on Comment/Uncomment:

If you have marked one or multiple lines and press button "Comment", then a double slash ("//") is appended in every marked line before the first non-whitespace character - independent from whether the line was already marked as comment or not. This enables you to deactivate lines including both comments and code - and later, via the "Uncomment" button, restore the original state of the code: The "Uncomment" button works exactly opposite to the "Comment" button; in every marked line, it removes the first instances of "//". If, e.g., there are 4 slashes in a code line, the button must be pressed twice in order to activate the code again. If a line does not start with at least 2 slashes before the first non-whitespace character, "Uncomment" will ignore this line.

14.8. Plugin ESLint

The plugin ESLint integrates the linter ESLint into the ADITO Designer. It is a very valuable tool that helps you to optimize your JavaScript code quality. Please find detailled information in chapter ESLint support.

14.9. Plugin SQL Formatter

The Plugin "SQL Formatter" enables you to easily format SQL code. To use it, it must first be installed via the Designer’s plugin portal: Tools > Plugins > Available Plugins > SQL Formatter > Install.

14.9.1. Functionality

Once installed, you can execute the plugin’s functionality directly in every SQL window, via the context menu (right-click in the SQL window). There are 2 options related to the SQL Formatter plugin:

- Format (shortcut: SHIFT+ALT+F): Formats the SQL code according to the current settings (see below)
- Copy to JS String: The SQL code will be
    - "cut" into concatinated Strings, line by line, and
    - copied to the clipboard, ready to be inserted in JDito/JavaScript code.

14.9.2. Settings

The formatting options can be set in menu Tools > Options > Editor > SQL Formatter:

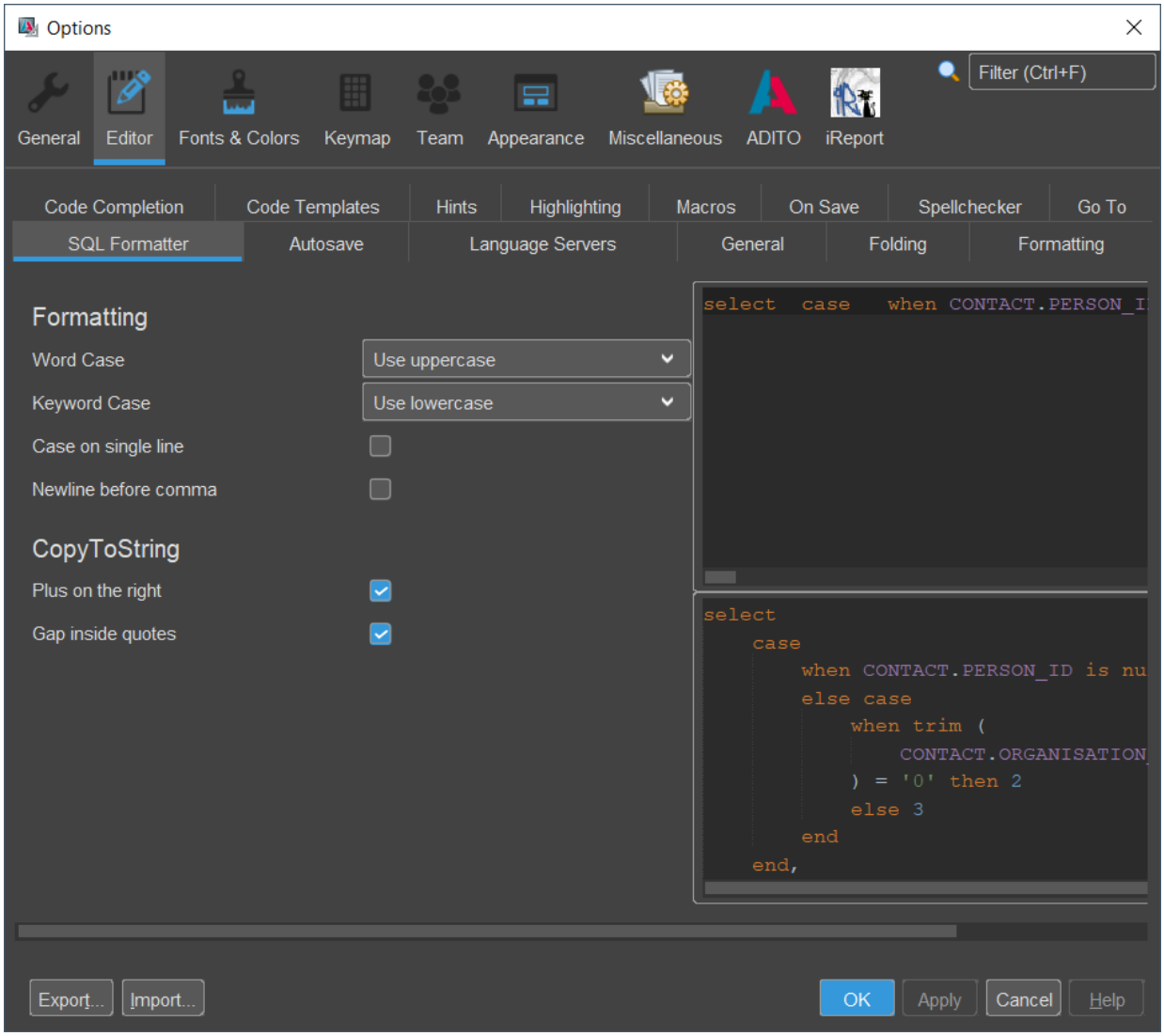


Figure 20. Options of the SQL Formatter

- Formatting
  - Word Case: Setting whether the names of tables and columns are to be formatted in uppercase or lowercase letters.
  - Keyword Case: Setting whether SQL keywords (e.g. "select") are to be formatted in uppercase or lowercase letters.
  - Case on single line: If set to true, then "case" statements are written in one single line.
  - Newline before comma: If set to true, then a newline will be set before every comma (i.e., commas will then be on the left).

In the "before" and "after" windows (with the first being editable) you can watch a preview of the effects of the currently set formatting options.

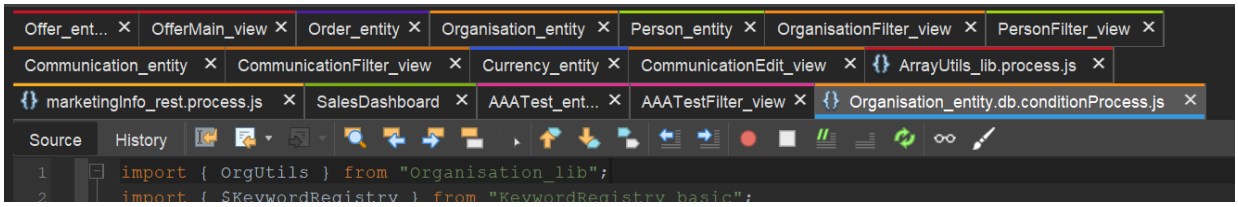
- CopyToString
    - Plus on the right: If set to true, then the "plus" sign used for concatinating the SQL code lines, will be set to the right - otherwise to the left.
    - Gap inside quotes: Indentations will be inserted inside the quotation marks.

The source code of this plugin is publicly available on ADITO's GitHub and can, if required, be used as template for building further plugins.

14.10. Plugin Grouped Tabs

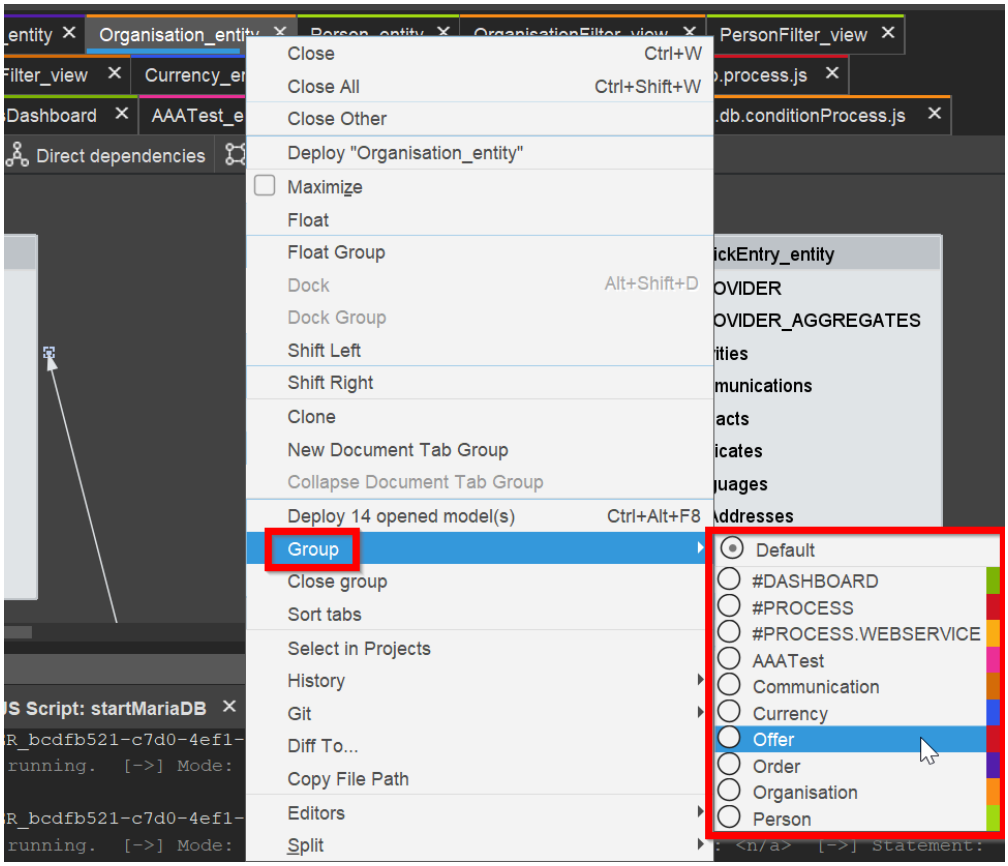
The Plugin "Grouped Tabs" helps you to keep a good overview over the tabs that you have opened in the Editor area of the Designer. Thin colored horizontal bars are being placed on top of each tab, with the color being the grouping criterion: Tabs that logically belong together show the same color, e.g.,

- Views and processes of the same Context (e.g. PersonFilter_view, PersonEdit_view and Person_entity.db.conditionProcess)
- processes of the same variant (e.g. the libraries Organisation_lib and Person_lib)



You can modify the grouping manually:

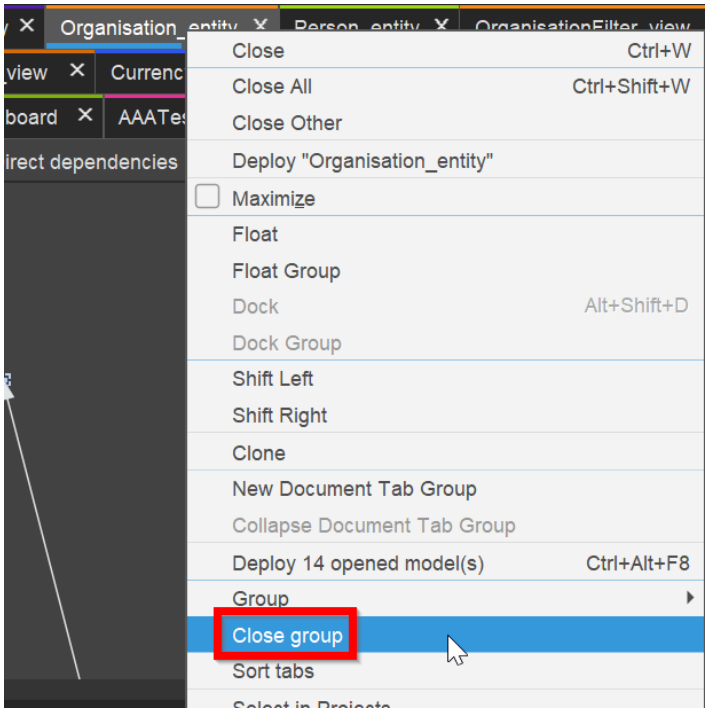
- Click on a tab, in order to move it to the foreground.
- Right-click on the tab, to open its context menu, and here choose "Group".
- Then, the groups of all currently opened tabs are shown, along with the corresponding colors.
- You can now change the group assignment (including the color) of the current tab by selecting another group of your choice.



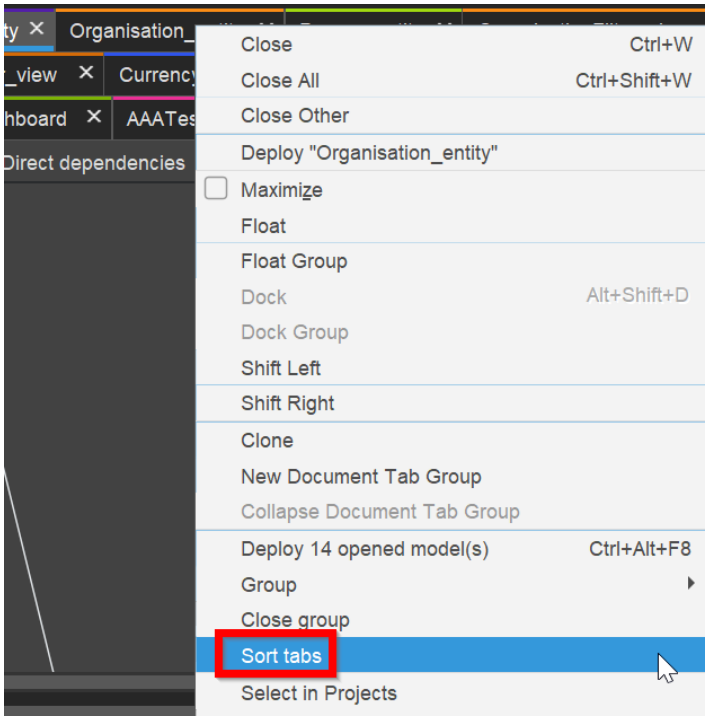
In the above screenshot, you see an option "Deploy n opened model(s)". Although this option is located directly above the group-related options, it does not refer to a group, but deploys *all* opened models of *all* groups.

You can close all tabs of a specific group in one step, if you

- click on any tab of the respective group, in order to move it to the foreground;
- right-click on the tab, to open its context menu, and here choose "Close group".



If you want to sort the tabs, to make tabs of the same group being placed together, right-click on *any* open tab and then choose "Sort tabs" from the context menu.



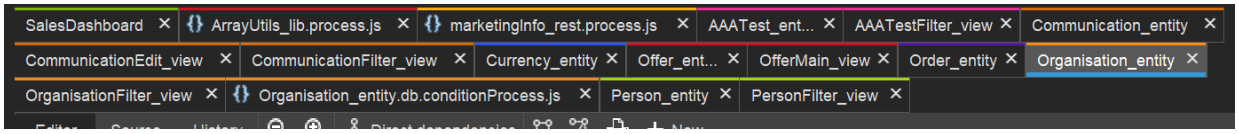
After the automated sorting, the tab order is as follows:

- The groups are sorted by model type, e.g.,

1. Dashboards
2. processes
3. Context-related models

- Groups related to the same type of model are sorted in alphabetical order, e.g. "Organisation_xxx" will be sorted before "Person_xxx".
- Inside each group, the order is as follows:
  - Context-related models:
    1. Entity
    2. Views
    3. processes
  - All other sorting (e.g., among libraries, or among Views of the same Context) is done in alphabetical order.

The above example appears after sorting like this:



Models are colored via an algorithm using the group name (e.g., the Context name), i.e., if 2 different Designer users

- open, e.g., Person_entity, their tabs will get the same color.
- create a new Entity with the same name, their tabs will also get the same color.

The number of colors is limited to about 25. This means, if you open a very large number of tabs, it is possible that 2 different groups have the same color. Nevertheless, they will be treated differently when being sorted.

There are shortcuts available:

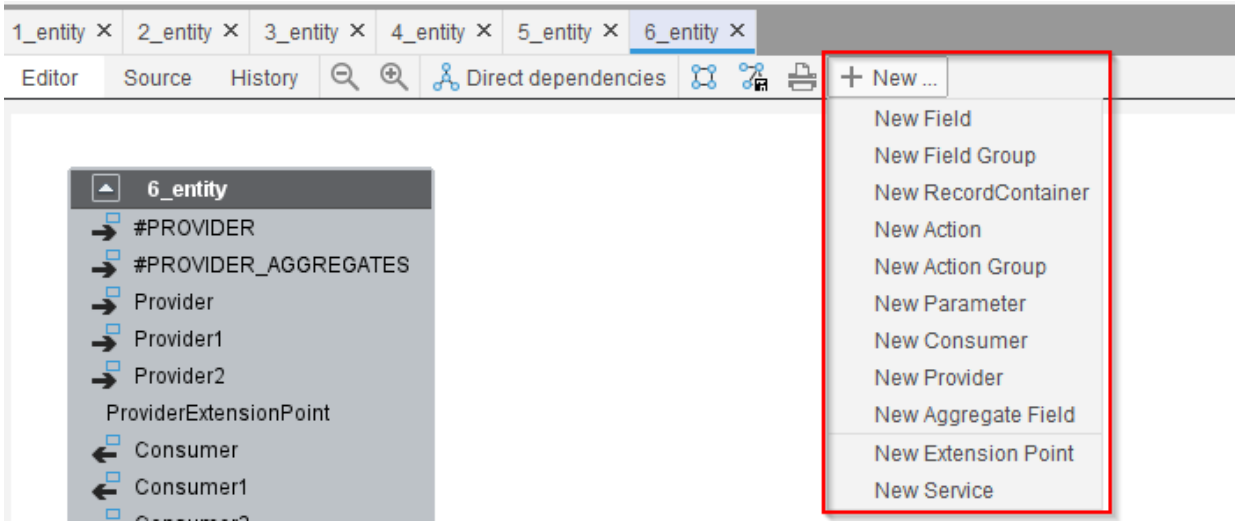
- Close group: Ctrl+Alt+W
- Sort tabs: Ctrl+Alt+S

Appendix A: Configuration helpers

In the Designer there are various mechanisms to simplify the creation and configuration of new models.

A.1. "+ New ..."-Button

With ADITO 2022.2.1 you have a new button to create a new model in an Entity. This button contains all the "New"-actions that were otherwise available when right-clicking on the entity node in the Navigator and that are located above "New with Blueprint". The right-click actions on the entity node still work.For modularized projects, the last two actions "New Extension Point" and "New Service" are still available in addition. For non-modularized projects, these two points are missing.



A.2. Blueprints

"Blueprint" is a functionality to simplify the creation of ADITO models, e.g., Entities, EntityFields, Contexts, and Views. You can execute Blueprints in the "Projects" window, via the context menu of the nodes "context" or "entity", respectively. Please find details in chapter "Blueprints" of the Customizing Manual.

Besides, you can create additional Blueprints, according to your own requirements. This will be explained in a separate manual, to be released soon. (If you need this information in advance, please contact ADITO).

A.3. Combobox content search

Some properties can be set via selection from a combo box, e.g., property "icon" (included in various ADITO models), or "recordfield" (in "xxx.value" and "xxx.displayValue" nodes of the RecordContainer). Now, instead of searching for the required list item by scrolling down the long list of combo box

content, you can simplify the search by using a wildcard.

Example:
You can quickly find a suitable icon, if you filter the "icon" combo box using the starlet ("*") as preceding wildcard.
Example: You are looking for an icon that has something to do with a "circle". To find all possibly suiting icons, proceed as follows:

1. Open the "icon" combo box by clicking on the small "arrow" to the left of the 3-points button: A long list of all icon names, along with a preview of all icons, will appear.
2. Type the following string: *circle
   → The focus will jump to the first icon with its name containing the word "circle"
3. Use the arrow keys (down/up) to navigate to all other icons having names containing "circle". (This is better than scanning the whole list using the scroll bar.)
4. Press the "Enter" key to select the icon of your choice.

Appendix B: Create and upgrade system tables

Besides creating system tables via Liquibase (see chapter Plugin Liquibase), ADITO includes a tool named "ADITOdatabase" that enables you to create and upgrade system tables via command line or batch file.

B.1. Benefits

The advantage of using this tool is that, in this case, the ADITO platform (core) determines the structure of the system tables. Thus, there is no need for additional maintenance on customizing side, and it is guaranteed that the latest structure version is being applied. Another advantage is the possibility to automatize the creation and upgrade of system tables - like, e.g., it is done in ADITO's "Self-Service Portal" (SSP).

B.2. Execution

The tool ADITOdatabase can be installed via the ADITO installer. Depending on your operating system, you will find the following executable file in sub-directory "bin":

- under Windows 32 bit: "ADITOdatabase.exe"
- under Windows 64 bit: "ADITOdatabase64.exe"
- under Linux and Mac: "ADITOdatabase"

B.2.1. Parameter

Table 2. Parameter for creating tables

Parameter	Call
Serverconfig	shortName = 's', longName = "serverconfig"
Tables for creation	longName = "createTables"

Table 3. Parameter for upgrading tables

Parameter	Call
Serverconfig	shortName = 's', longName = "serverconfig"

Parameter	Call
Projecthome	shortName = 'p', longName = "projecthome"
Tables for upgrade	longName = "upgradeTables"

B.2.2. Examples

The following examples are based on a Windows 64 bit operating system.

B.2.2.1. Creating system tables

Create all system tables

```
./ADITOdatabase64.exe
  --serverconfig="C:\Users\j.smith\Documents\AditoProjects\myProjectName\data\config\serverconfig_default.xml"
  --createTables=""
```

Create only system tables ASYS_SYSTEM and ASYS_ALIASCONFIG

```
./ADITOdatabase64.exe
  --serverconfig="C:\Users\j.smith\Documents\AditoProjects\myProjectName\data\config\serverconfig_default.xml"
  --createTables="ASYS_SYSTEM,ASYS_ALIASCONFIG"
```

B.2.2.2. Upgrading system tables

Upgrade all system tables

```
./ADITOdatabase64.exe
  --serverconfig="C:\Users\j.smith\Documents\AditoProjects\myProjectName\data\config\serverconfig_default.xml"
  --projecthome="C:\Users\j.smith\Documents\AditoProjects\myProjectName"
  --upgradeTables=""
```

Upgrade only system tables ASYS_SYSTEM and ASYS_ALIASCONFIG

```
./ADITOdatabase64.exe
  --serverconfig="C:\Users\j.smith\Documents\AditoProjects\myProjectName\data\config\serverconfig_default.xml"
  --projecthome="C:\Users\j.smith\Documents\AditoProjects\myProjectName"
  --upgradeTables="ASYS_SYSTEM,ASYS_ALIASCONFIG"
```

B.2.2.3. Using short names

Whenever the short name of a parameter is used, the equality sign (“=”) must not be written.

Example:

```
--serverconfig="C:\Users\j.smith\Documents\AditoProjects\myProjectName\data\config\serverconfig_default.xml"
```

is equivalent to

```
-s"C:\Users\j.smith\Documents\AditoProjects\myProjectName\data\config\serverconfig_default.xml"
```

B.2.3. Exit codes

The following exit codes of the tool are a useful feedback in order to verify the success of the execution, and to react accordingly - especially when you want to automatize the creation of the system tables.

B.2.3.1. General exit codes

Table 4. General exit codes

Exit code	Meaning
0	Success
50	parameters "createTables" and "upgradeTables" are both missing
51	parameter "createTables" and "upgradeTables" have both been specified (this is not possible; you can only specify one of this parameters)
100	path to serverconfig file missing
101	specified path to serverconfig file does not exist
102	specified path to serverconfig file is a directory, not a file
103	specified serverconfig file could not be read (invalid format)
200	other error (error message will be logged)

B.2.3.2. Exit codes of createTables

Table 5. Exit codes of createTables

Exit code	Meaning
10	Success, but at least one of the tables had already existed
150	at least one table could not be created

B.2.3.3. Exit codes of upgradeTables

Table 6. Exit codes of upgradeTables

Exit code	Meaning
160	parameter projecthome has not been specified
161	the specified projecthome does not reference a valid ADITO projekt
162	at least one of the specified tables could not be upgraded, because it does not exist
163	at least one of the specified tables could not be upgraded, because it is not a system table

Appendix C: Setting the path variables

You can specify different paths in the properties of a system. These paths can also be set at any time with the Java VM variables:

- -Dadito.home and
- -Dadito.data

In the properties of a system these are available as properties:

- aditoHomePath
- aditoDataPath

The following path variables can be used in the above properties:

1. $PROJECT HOME
Home directory of the project. Usually created in the development system for the aditoDataPath and points to $ PROJECTHOME / data.

2. $ADITOHOME
The ADITO home directory. In development systems, the Designer's home directory if the server is started inside the Designer.

3. $ADITODATA
Refers to the directory set in aditoDataPath or -Dadito.data. In aditoDataPath or -Dadito.data this variable can not be set.

Appendix D: UUID Generator

Most of ADITO datasets are identified by a random 36-digit hexadecimal "Universally unique identifier" (abbreviated as "UUID", "UID", or "GUID"), see https://en.wikipedia.org/wiki/Universally_unique_identifier. This identifier is generated automatically for every new dataset, if you have set flag "UID column" to true, in property "linkInformation" of a dbRecordContainer (see Customizing Manual, chapter "Connecting EntityFields with database columns (RecordContainer)").

However, for some tasks you need to generate a UUID manually - e.g., if you create a Liquibase XML file. Now, instead of using online UUID generators available on several web sites, you can also use the Designer-integrated UUID generator, which is accessiable via the following ways:

- menu "Tools" > "Generate UUID" (will be copied into the clipboard)
- shortcut: CTRL + Shift + U (will be copied into the clipboard)
- A direct insert of a UUID into a code window is available via the "Generate" menu: Alt + Ins > UUID.

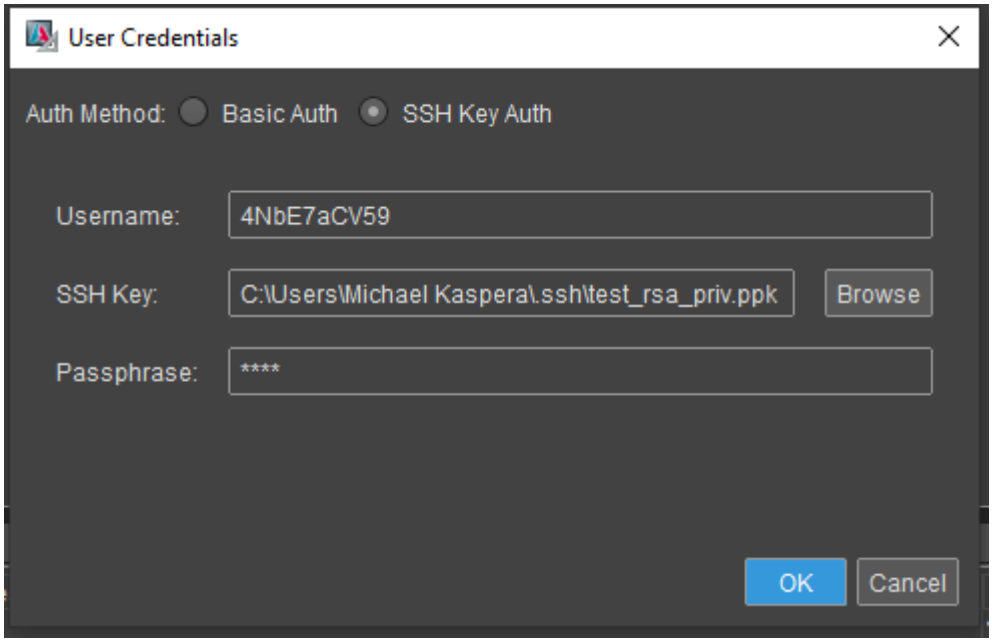
Appendix E: Tunneling

The Designer supports tunneling to remote servers in various use cases, e.g., in combination with ADITO's Self-Service Portal (SSP), see sub-chapter With SSP access, and chapter Debugger.

Currently, the ADITO Designer supports the following authentication schemes:

- Basic auth (user has to provide a username and a password)
- Key auth (user has to provide a SSH key)

The user can choose which authentication method is used by selecting the wanted scheme with the radio buttons in the auth dialog.



The selected auth method has to be supported by the server, the SSP only supports basic auth at the moment

>>>

Appendix F: Version history

Version	Changes
1.7	• New chapter Installation • New chapter Ports • Chapter alias: New note box explaining how to maintain ERD diagrams • New chapter about option Janitor • Chapter Internationalization: New sub-chapter Translation of source code • Chapter Connection via "load cloud system" improved • Chapter Process refactored and extened • Chapter Deploy refactored, including new sub-chapters Deploy of selected data models and Deploy of opened data models • Chapter Find unused keys extended • New Chapter Plugin Grouped Tabs • Chapter New - New Project: Warning box regarding invalid project names • New chapter Plugin NodeJS & TypeScript • Chapter Plugin AsciidoctorJ renewed, including a reference to AID005 Userhelp.
1.6	• Chapter Export Keys with Place of Use: working with language files and there usage for modularization • Chapter Refactor: Warning box regarding refactoring dialog optimized • New chapter Code quality support, including sub-chapter ESLint support • Info box about one [userDirectory] being generated for every ADITO version (major/minor/hotfix) • Chapter Configuration: New -D parameter for deactivating non-supported authentication methods when using a proxy • Chapter Watches: new debugger information added • Chapter Auto-Merging: new feature of the Git plugin • Chapter Deletion of a context • New images, typo fixes

Version	Changes

1.5	• chapter Deploy Tool: explanation of Exit codes
	• appendix "Create system tables" renamed to Create and upgrade system tables and extended by description of how to upgrade system tables
	• chapter Deploy extended by remark on how to use short names
	• chapter Plugin Liquibase
	  ○ sub-chapter "Basics": Warning of using special characters in XML file names included
	  ○ sub-chapter How to use Liquibase extended by advice on how to avoid timeout dialogs
	• chapter Startup, sub-chapter "Configuration": Extended description of how to set the timeout value
	• chapter Debugger updated: No restart of the Designer required, code execution stops not before the second execution, orange color requires click into code window.
	• chapter CREATE extended by description of how to create a sorted index (ascending or descending)
	• new chapter Plugin SQL Formatter
	• chapter Project Structure updated, including description of ERD/adoc export function of Alias Definition in sub-chapter about [AliasDefinition]
	• chapter Internationalization:
	  ○ new sub-chapter User help, including reference to AID005
	  ○ new sub-chapter Special functions, describing how to simplify the setting and translation of column "Value" for existing and new language files
	  ○ Update of chapter Automated translation: Values are translated instead of (in earlier versions) the keys.

Version	Changes
1.4.1	• chapter Connection via "load cloud system": description of option "Load Deployed System State" • chapter Plugin Liquibase extended by description of preferable data types, including reference to AID001 Coding Styles • chapter View: paragraph "run" extended by description of run configs, especially those of NodeJS scripts • new appendix Tunneling • adaptions regarding new "import" syntax of version 2022.0

1.4	• new chapter Liquibase with audit and offline synchronisation
	• chapter Plugin Git: updates, explanation of option "init" added
	• new chapter Startup, including description of Designer startup configuration parameters in file "ADITOdesigner.conf"
	• chapter Debugger extended by notes that property "jditoDebuggerEnabled" must be activated for local servers and that both server and Designer need to be restarted.
	• new appendix [Create system tables], explaining command line tool "ADITOdatabase" for automated creation of system tables.
	• description of conditional execution of Liquibase functions via tag "preConditions", included in chapters
	  ◦ PRECONDITIONS (sub-chapter of chapter Data Definition Language (DDL)) and
	  ◦ DB to changelog (sub-chapter of chapter Create Liquibase files automatically)
	• chapter Automated translation extended by notes on settings required for DeepL API.
	• reference to the new appendix in chapter Maintaining system tables

Version	Changes
1.3	• chapter Tools revised and extended by option "Export Project Structure" • chapter Updates: Restructured and enhanced by sub-chapter on upgrading system tables • chapter Debugger: Included info box on debugging cloud systems • chapter Deploy Tool: Revision, update (new parameter "userhelp"), description of differences between execution under Windows and Linux • chapter "Language" renamed to Internationalization, refactored, and enhanced by including a reference to the DeepL API.

1.2	• new chapter Connecting a local project to a cloud system
	• chapter Plugin Liquibase:
	  ○ content partly updated and restructured;
	  ○ new sub-chapter Create Liquibase files automatically
	  ○ new sub-chapter Liquibase troubleshooting
	• new appendix UUID Generator
	• chapter [Scan services] extended (deactivation option, description of "Indexing" phases)

1.1	• new chapter Plugin Cloud Support, on how to connect the Designer with a cloud system • new appendix Configuration helpers; improved overview of how to create new ADITO models • chapter Plugin concept: added note on firewall problems when installing plugins • Preface added • version history table: hyperlinks to chapters added; formatting optimized • restructuring, renaming
1.0	• first official version