



Elektrobit

# EB tresos® Studio for ACG8 developer's guide

product release 8.8.4





Elektrobit Automotive GmbH  
Am Wolfsmantel 46  
91058 Erlangen, Germany  
Phone: +49 9131 7701 0  
Fax: +49 9131 7701 6333  
Email: info.automotive@elektrobit.com

## Technical support

<https://www.elektrobit.com/support>

## Legal disclaimer

Confidential information.

ALL RIGHTS RESERVED. No part of this publication may be copied in any form, by photocopy, microfilm, retrieval system, or by any other means now known or hereafter invented without the prior written permission of Elektrobit Automotive GmbH.

All brand names, trademarks, and registered trademarks are property of their rightful owners and are used only for description.

Copyright 2021, Elektrobit Automotive GmbH.



# Table of Contents

Begin here .....	16
1. If you are upgrading from a previous version .....	16
2. If you are looking for a particular API .....	16
3. If you are a first-time EB tresos Studio developer .....	16
4. If you need help/more information .....	17
1. About this documentation .....	18
1.1. Introduction .....	18
1.2. Required knowledge and system environment .....	18
1.2.1. Required system environment .....	19
1.3. Interpretation of version information .....	19
1.3.1. Product version number .....	19
1.3.2. Qualification of basic software .....	20
1.4. Typography and style conventions .....	21
1.5. Naming conventions .....	22
1.5.1. AUTOSAR XML schema .....	22
1.5.2. Configuration parameter names .....	22
1.6. EB tresos Studio naming conventions .....	23
1.6.1. EB tresos Studio command shell .....	23
2. Safe and correct use of EB tresos Studio .....	24
2.1. Intended usage of EB tresos Studio .....	24
2.2. Possible misuse of EB tresos Studio .....	24
2.3. Target group and required knowledge .....	24
2.4. Quality standards compliance .....	24
3. Introduction .....	25
3.1. What is the Public Java API? .....	25
3.2. What benefits do I have from using the Public Java API? .....	27
3.3. Public Java API documentation .....	27
3.4. How to use the Public Java API .....	28
3.5. Restrictions to the usage of the Public Java API .....	28
4. Working with Eclipse .....	29
4.1. Prerequisites .....	29
4.1.1. General preparations to use the Public API .....	29
4.2. How to install Eclipse .....	29
4.3. How to install the EB tresos Studio Eclipse tooling .....	31
4.4. How to set up the target platform .....	31
4.5. How to open extension point descriptions .....	34
4.6. How to create a project with Eclipse .....	37
4.6.1. Errors and warnings within the Plugin Development Environment (PDE) .....	41
4.6.2. Creating a EB tresos Studio ModulePlugIn .....	42



4.6.2.1. Schema-file and templates .....	42
4.6.2.2. Extension points .....	43
4.6.2.2.1. Extension point dreisoft.tresos.launcher2.plugin.module .....	43
4.6.2.2.2. Extension point dreisoft.tresos.launcher2.plugin.configuration .....	44
4.6.2.2.3. Extension point dreisoft.tresos.launcher2.plugin.generator .....	44
4.7. How to set up a run configuration .....	45
4.8. How to define default preferences .....	48
4.9. How to install a Public API demo plugin .....	49
5. Developing modules .....	50
5.1. Concepts: The XDM format .....	50
5.1.1. Concepts .....	50
5.1.2. Top level structure .....	50
5.1.3. Nodes .....	51
5.1.4. Attributes .....	52
5.1.4.1. Passive attributes .....	52
5.1.4.2. Generic attributes .....	53
5.1.4.2.1. AdminData .....	53
5.1.4.2.2. Implementation Configuration Variant .....	53
5.1.4.2.3. Implementation Config Class .....	54
5.1.4.2.4. Custom data attribute .....	55
5.1.4.3. Automatic attributes .....	55
5.1.4.3.1. Bound attribute .....	56
5.1.4.3.2. Match attribute .....	57
5.1.4.3.3. Range attribute .....	58
5.1.4.3.3.1. Tests defining range boundaries .....	58
5.1.4.3.3.2. Tests allowing single values .....	58
5.1.4.3.3.3. Tests excluding single values .....	58
5.1.4.3.4. XPath attribute .....	59
5.1.4.3.5. Multi test attribute .....	60
5.1.4.3.6. EcucCond attribute .....	61
5.1.4.3.7. EcucValidCond attribute .....	63
5.1.4.3.8. XDM attribute examples .....	64
5.1.4.3.8.1. Check the value of another node .....	64
5.1.4.3.8.2. Negation .....	65
5.1.4.3.8.3. Logical Operators: and, or .....	65
5.1.4.3.8.4. Range-check using the Match-attribute .....	65
5.1.4.3.8.5. Range-check using the XPath-attribute .....	66
5.1.4.3.8.6. Only allow values configured in another list .....	66
5.1.4.3.8.7. DEFAULT-value using the Bound-Attribute .....	66
5.1.4.3.8.8. Only allowing certain values using the Range-Attribute .....	67
5.1.4.3.8.9. Modulus operator .....	67
5.1.4.3.8.10. Calculating with node-values .....	67



5.1.4.3.8.11. Regular expressions .....	68
5.1.4.3.8.12. References .....	68
5.1.4.3.8.13. Preconfigured values for list-entries .....	69
5.1.4.3.9. Restrictions .....	69
5.1.4.4. Data attributes .....	70
5.1.4.5. Common attributes .....	71
5.1.5. Generic nodes .....	71
5.1.5.1. link .....	71
5.1.5.1.1. Attributes .....	71
5.1.5.1.2. Valid children .....	71
5.1.6. Schema-nodes .....	72
5.1.6.1. ctr .....	72
5.1.6.1.1. Attributes and data attributes .....	72
5.1.6.1.2. Valid children .....	73
5.1.6.2. chc .....	73
5.1.6.2.1. Attributes and data attributes .....	73
5.1.6.2.2. Valid children .....	74
5.1.6.3. lst .....	74
5.1.6.3.1. Attributes and data attributes .....	74
5.1.6.3.2. Valid children .....	74
5.1.6.4. var .....	75
5.1.6.4.1. Attributes and data attributes .....	75
5.1.6.4.2. Valid children .....	76
5.1.6.5. ref .....	76
5.1.6.5.1. Attributes and data attributes .....	76
5.1.6.5.2. Valid children .....	76
5.1.7. Data-nodes .....	77
5.1.7.1. ctr .....	77
5.1.7.1.1. Attributes .....	77
5.1.7.1.2. Valid Children .....	77
5.1.7.2. chc .....	77
5.1.7.2.1. Attributes .....	78
5.1.7.2.2. Valid Children .....	78
5.1.7.3. lst .....	78
5.1.7.3.1. Attributes .....	78
5.1.7.3.2. Valid Children .....	78
5.1.7.4. var .....	78
5.1.7.4.1. Attributes .....	79
5.1.7.4.2. Valid Children .....	79
5.1.7.5. ref .....	79
5.1.7.5.1. Attributes .....	79
5.1.7.5.2. Valid Children .....	79

5.1.8. Factories .....	79
5.1.9. Path addressing .....	80
5.2. Concepts: Mapping AUTOSAR ECU Configuration to XDM .....	81
5.2.1. Node mapping .....	81
5.2.1.1. Schema-Nodes .....	81
5.2.1.1.1. AR-PACKAGE .....	81
5.2.1.1.2. MODULE-DEF .....	82
5.2.1.1.3. PARAM-CONF-CONTAINER-DEF .....	82
5.2.1.1.4. MULTIPLE-CONFIGURATION-CONTAINER .....	82
5.2.1.1.5. BOOLEAN-PARAM-DEF .....	83
5.2.1.1.6. INTEGER-PARAM-DEF .....	83
5.2.1.1.7. FLOAT-PARAM-DEF .....	83
5.2.1.1.8. STRING-PARAM-DEF .....	83
5.2.1.1.9. LINKER-SYMBOL-DEF .....	84
5.2.1.1.10. FUNCTION-NAME-DEF .....	84
5.2.1.1.11. ENUMERATION-PARAM-DEF .....	84
5.2.1.2. CHOICE-CONTAINER-DEF .....	85
5.2.1.3. CHOICE-REFERENCE-DEF .....	85
5.2.1.4. REFERENCE-DEF .....	86
5.2.1.5. SYMBOLIC-NAME-REFERENCE-DEF .....	86
5.2.1.6. INSTANCE-REFERENCE-DEF .....	86
5.2.1.7. FOREIGN-REFERENCE-DEF .....	87
5.2.1.8. ECUC-URI-REFERENCE-DEF .....	87
5.2.1.9. DERIVED-BOOLEAN-PARAM-DEF .....	88
5.2.1.10. DERIVED-INTEGER-PARAM-DEF .....	88
5.2.1.11. DERIVED-FLOAT-PARAM-DEF .....	88
5.2.1.12. DERIVED-STRING-PARAM-DEF .....	89
5.2.1.13. DERIVED-ENUMERATION-PARAM-DEF .....	89
5.2.1.14. Data-Nodes .....	89
5.2.1.14.1. ECU-CONFIGURATION .....	89
5.2.1.14.2. MODULE-CONFIGURATION .....	90
5.2.1.14.3. CONTAINER .....	90
5.2.1.14.4. Choices .....	90
5.2.1.14.5. BOOLEAN-VALUE .....	91
5.2.1.14.6. INTEGER-VALUE .....	92
5.2.1.14.7. FLOAT-VALUE .....	92
5.2.1.14.8. STRING-VALUE .....	92
5.2.1.14.9. LINKER-SYMBOL-VALUE .....	93
5.2.1.14.10. FUNCTION-NAME-VALUE .....	93
5.2.1.14.11. ENUMERATION-VALUE .....	93
5.2.1.14.12. REFERENCE-VALUE .....	94
5.2.1.14.13. INSTANCE-REFERENCE-VALUE .....	94



5.2.1.14.14. Example: .....	95
5.2.2. Attribute Mapping .....	95
5.2.2.1. CONFIGURATION-CLASS-AFFECTION .....	95
5.2.3. XPath-addressing of ECU-CD using ECU-PD .....	96
5.2.3.1. Path in Template .....	97
5.2.4. Top level Structure .....	98
5.2.5. List-Nodes .....	99
5.2.5.1. Optional elements .....	100
5.2.5.2. Optional elements within the commandline .....	101
5.2.5.3. Optional elements within the GUI .....	101
5.2.6. Choice-Nodes .....	101
5.2.7. References .....	103
5.2.7.1. URI References .....	103
5.2.7.1.1. XDM representation .....	105
5.3. How to create a new module .....	105
5.3.1. Prerequisites .....	106
5.3.2. Creating a vendor-specific module definition .....	107
5.3.3. Converting files .....	107
5.3.4. Templates for new modules .....	107
5.3.5. Setup of the file structure .....	108
5.3.6. Adapting MANIFEST.MF .....	108
5.3.7. Adapting plugin.xml .....	108
5.3.7.1. dreisoft.tresos.launcher2.plugin.module .....	108
5.3.7.2. Optional: dreisoft.tresos.launcher2.api.plugin.modulecluster .....	111
5.3.7.3. dreisoft.tresos.launcher2.plugin.configuration .....	112
5.3.7.3.1. Parameter definition, pre-, recommended configuration .....	112
5.3.7.3.2. Module configuration .....	114
5.3.7.3.2.1. Package structure of the module configuration .....	114
5.3.7.3.3. Registering editors .....	115
5.3.7.3.4. Code sample .....	115
5.3.7.4. dreisoft.tresos.launcher2.plugin.generator .....	116
5.3.7.5. Module versions .....	117
5.3.7.5.1. Software version .....	119
5.3.7.5.2. Specification version .....	120
5.3.7.5.3. Release version .....	120
5.3.8. Code templates .....	120
5.3.9. Adding the module to EB tresos Studio .....	120
5.4. How to migrate your module from one AUTOSAR version to another .....	121
5.4.1. How to migrate an AUTOSAR 2.x module to 3.x .....	121
5.4.1.1. Choices .....	122
5.4.1.2. Configuration variant .....	122
5.4.1.3. ADMIN-DATA .....	123



5.4.2. How to migrate your AUTOSAR module to AUTOSAR version 4.x .....	124
5.5. How to sign modules with the crypto command .....	124
5.5.1. Purpose .....	124
5.5.2. Prerequisites .....	125
5.5.3. Commands .....	125
5.6. Module Transformer API .....	126
5.6.1. Purpose .....	126
5.6.2. API .....	126
5.6.3. Registering a module transformer .....	126
5.6.3.1. Examples .....	127
5.6.4. Implementing a module transformer Class .....	128
5.6.5. Chaining of module transformers .....	129
5.7. Variant handling and post-build loadable support .....	129
5.7.1. Purpose .....	129
5.7.2. Prerequisites .....	129
5.7.3. Module with support for variant selection .....	130
5.7.4. Marking elements as post-build selectable variant aware .....	130
5.7.5. AUTOSAR attribute postBuildVariantUsed .....	131
5.7.6. Marking elements as post-build loadable variant-aware .....	132
5.7.7. Variant handling API .....	132
5.7.8. Variant-aware code generators .....	133
5.7.9. Variant handling demo plug-in .....	133
5.7.9.1. The EcuC module .....	134
5.7.9.2. The variant module .....	135
5.7.9.3. The code generators .....	136
5.7.9.4. Switching variants .....	137
5.7.9.5. Managing variants .....	138
5.8. Post-build loadable support via Multiple Configuration Containers .....	139
5.8.1. Providing a post-build configuration management module .....	140
5.8.2. Additional VSMD rules regarding post-build enabled configuration modules .....	140
6. Configuration models .....	142
6.1. Concepts: The DataModel .....	142
6.1.1. Schematic-tree .....	144
6.1.1.1. Example .....	144
6.1.1.2. Features of Schema-Nodes .....	145
6.1.2. Data-Tree .....	145
6.1.3. Node-Attributes .....	145
6.1.4. List-Representation within the DataModel .....	146
6.1.5. Choice-representation within the DataModel .....	147
6.2. How to add GUI annotation in the XDM file .....	148
6.2.1. Purpose .....	148
6.2.2. Generic Configuration Editor Layout .....	148



6.2.3. Module GUI configuration .....	151
6.2.4. GUI Elements .....	152
6.2.4.1. Attributes .....	152
6.2.4.2. Boolean values .....	153
6.2.4.3. Integer Values, Float Values, String Values, Enumerations and References .....	156
6.2.4.4. Container .....	159
6.2.4.5. Choices .....	163
6.2.4.6. Lists .....	167
6.2.4.7. Tab assignment .....	172
6.3. XPath API .....	172
6.3.1. Purpose .....	172
6.3.2. Prerequisites .....	173
6.3.3. Concepts .....	174
6.3.3.1. Conventions .....	174
6.3.3.2. Data-Types within XPath .....	174
6.3.3.3. Additional Data-Types for the DataModel .....	175
6.3.3.4. DataModel-Nodes and their XPath-Value .....	176
6.3.3.5. Example-Tree .....	176
6.3.4. API .....	176
6.3.4.1. Pitfalls when using XPath .....	177
6.3.4.1.1. Non-existing nodes .....	177
6.3.4.1.1.1. Expressions .....	177
6.3.4.1.1.2. Functions .....	178
6.3.4.1.2. Type conversion with Boolean expressions .....	178
6.3.4.1.3. Implementation errors in the handling of Boolean values .....	179
6.3.4.1.4. Strings and numbers .....	179
6.3.4.1.5. Integers .....	180
6.3.4.1.6. Target node filter .....	182
6.3.4.1.7. Conclusion .....	182
6.3.4.2. XPath expressions .....	182
6.3.4.3. XPath predicates .....	184
6.3.4.4. XPath operations .....	185
6.3.4.5. XPath functions .....	187
6.3.4.5.1. Node-Set functions .....	187
6.3.4.5.2. String-functions .....	188
6.3.4.5.3. Boolean-functions .....	191
6.3.4.5.4. Number-functions .....	192
6.3.4.5.5. Additional functions provided by the XPath Engine .....	193
6.3.4.5.5.1. Node functions .....	193
6.3.4.5.5.2. Text functions .....	201
6.3.4.5.5.3. Number functions .....	206
6.3.4.5.5.3.1. Number conversions .....	208



6.3.4.5.5.4. Bit functions .....	212
6.3.4.5.5.5. Variable functions .....	214
6.3.4.5.5.6. License functions .....	214
6.3.4.5.5.7. Util functions .....	214
6.3.4.5.5.8. Documentation functions .....	215
6.3.4.5.5.9. Variant functions .....	217
6.3.4.5.5.10. Custom data attribute functions .....	219
6.3.4.5.6. Additional XPath-functions for AUTOSAR .....	219
6.4. Custom XPath Functions API .....	222
6.4.1. Registering custom XPath functions .....	222
6.4.2. Implementing custom XPath functions .....	223
6.5. ECU Resource Manager API .....	225
6.5.1. Purpose .....	225
6.5.2. Registering ECU resource properties .....	225
6.5.2.1. Advanced parameters .....	227
6.5.3. ECU Resource Finder .....	228
6.5.3.1. Advanced parameters .....	228
6.5.3.2. Default ECU Resource Finder Implementation .....	229
6.5.4. ECU Resource XPath Functions .....	231
6.5.5. Characteristics when using the legacy commandline .....	232
6.5.5.1. Defining parameter values for the DefaultEcuResourceFinder .....	232
6.5.5.2. Defining configured modules for the DefaultEcuResourceFinder .....	232
6.5.5.3. Converting a module schema with ECU Resources to AUTOSAR .....	233
6.6. Resources API .....	233
6.7. System Model Access API .....	234
6.7.1. System Model Access Demo .....	235
6.8. Custom Attribute API .....	235
6.8.1. Registering a custom attribute .....	236
7. Generating code .....	237
7.1. General concept .....	237
7.1.1. Generating module-independent code .....	237
7.1.2. Specifying generation steps .....	238
7.1.3. Defining variant-aware code generators .....	238
7.1.4. Order of execution .....	239
7.1.5. Running generators in parallel .....	241
7.1.5.1. Prevent generator classes from running in parallel .....	242
7.1.5.2. Prevent single generators from running in parallel .....	242
7.2. Template-based Code Generator API .....	242
7.2.1. Concepts .....	242
7.2.1.1. Addressing nodes using XPath .....	243
7.2.1.2. Identical generated files .....	243
7.2.2. Codetemplate console .....	244



7.2.3. Comments .....	245
7.2.4. Evaluating variables and expressions .....	245
7.2.5. Including other files .....	245
7.2.6. Conditions .....	246
7.2.7. Loops and selects .....	247
7.2.8. Leaving loops .....	249
7.2.9. Variables and macros .....	249
7.2.10. Spaces, newlines and indentation .....	253
7.2.10.1. Tab stops .....	253
7.2.10.2. Whitespaces .....	254
7.2.10.3. Newlines .....	254
7.2.10.4. Auto-spacing .....	254
7.2.10.5. Indentation .....	256
7.2.11. Excluding files from the generation .....	257
7.2.12. Asserts, errors and warnings .....	258
7.3. Public Generator API .....	259
7.3.1. Registering A Public API Generator .....	259
7.3.2. Implementing a Public API Generator .....	260
7.3.2.1. API .....	261
7.4. NG Generator API .....	261
7.4.1. Purpose .....	261
7.4.2. JET Support .....	261
7.4.3. Standalone JET compiler .....	262
7.4.4. C Data Structures Generator (CDS) .....	262
7.4.5. Apache Ant support .....	263
7.4.6. Source delivery .....	266
7.5. EPC File Generator API .....	267
7.5.1. Purpose .....	267
7.5.2. Generating EPC files .....	267
7.6. External Generator API .....	269
7.6.1. Purpose .....	269
7.6.2. Functionality .....	269
7.6.3. Prerequisites .....	270
7.6.4. Adding the Generator to the plug-in .....	270
7.6.5. Registering an External Generator .....	270
7.6.6. Generator Commandline .....	273
7.6.7. Parsing the output of the generator .....	275
8. Extending the user interface .....	278
8.1. How to use the National Language Support (NLS) / Operation Status (OS) .....	278
8.1.1. Localizing the language of the user interface .....	278
8.1.1.1. Concepts: National Language Support (NLS) .....	278
8.1.1.2. Workflow for NLS-support in Java code .....	279



8.1.1.3. Config file format .....	280
8.1.1.4. NLS examples .....	282
8.1.1.4.1. Example for the usage in a Plugin.xml .....	282
8.1.1.4.2. NLS source code usage example .....	283
8.1.2. Defining APIOperationStatus subclasses .....	283
8.1.2.1. Concepts: APIOperationStatus .....	284
8.1.2.1.1. Glossary .....	284
8.1.2.1.2. Functionality .....	284
8.1.2.2. Workflow for OS-support in Java code .....	285
8.1.2.3. Config file format .....	286
8.1.2.4. Operation Status Examples .....	288
8.1.3. The NLS/OS builder .....	290
8.1.3.1. Commandline support .....	290
8.2. Guided Configuration API .....	290
8.2.1. Purpose .....	290
8.2.2. Prerequisites .....	291
8.2.2.1. Knowledge required .....	291
8.2.2.2. Plug-ins required .....	291
8.2.3. Design overview .....	291
8.2.3.1. Wizard type overview .....	295
8.2.3.1.1. Standalone editor .....	297
8.2.3.1.2. Dialog .....	298
8.2.3.1.3. Unattended wizard .....	300
8.2.3.1.4. Custom module editor .....	302
8.2.3.2. Selection context .....	303
8.2.3.2.1. The ECUConfigContext class .....	303
8.2.3.2.2. The ModuleConfigContext class .....	305
8.2.3.2.3. The SystemConfigContext class .....	306
8.2.3.3. Lifecycle .....	306
8.2.3.4. Data representation .....	308
8.2.3.4.1. The MementoOperationHandler class .....	309
8.2.3.4.2. The Memento interface .....	310
8.2.3.4.3. MementoFactory .....	310
8.2.3.5. GUI abstraction .....	311
8.2.3.5.1. Multi-page support .....	311
8.2.3.5.2. Layout .....	311
8.2.3.5.3. Validation .....	312
8.2.3.5.4. Navigation .....	313
8.2.3.5.5. Predefined widgets .....	313
8.2.3.5.6. Widget lifecycle .....	317
8.2.3.5.7. Results view .....	317
8.2.3.6. The push service pattern .....	319



8.2.3.7. Locking mechanism .....	320
8.2.4. API .....	321
8.2.4.1. Extensibility .....	322
8.2.4.1.1. Model .....	322
8.2.4.1.2. View .....	322
8.2.4.1.3. Controller .....	323
8.2.4.1.4. Additional extensible classes .....	323
8.2.4.1.4.1. Context .....	323
8.2.4.1.4.2. Widget .....	324
8.2.4.1.4.3. Push service .....	326
8.2.4.1.4.4. Triggers .....	326
8.2.5. Demos .....	327
8.2.5.1. Setting up the demo plug-in .....	327
8.2.5.1.1. Making the Sidebar view visible .....	328
8.2.5.1.2. Customizing the view .....	328
8.2.5.2. Demo1 .....	330
8.2.5.2.1. Restricting the visibility .....	330
8.2.5.2.2. Using multi-page support .....	331
8.2.5.2.3. Using XForm GUI description .....	331
8.2.5.2.4. Validating .....	331
8.2.5.2.5. Navigating .....	331
8.2.5.2.6. Creating your own <code>MementoFactory</code> class .....	332
8.2.5.3. Demo2 .....	332
8.2.5.3.1. Restricting the visibility .....	333
8.2.5.3.2. Creating a custom widget .....	334
8.2.5.3.3. Using the push service .....	334
8.2.5.3.4. Updating widget enablement .....	334
8.2.5.4. Demo3 .....	334
8.2.5.4.1. Using the <code>PushService</code> class .....	336
8.2.5.4.2. Registering a custom <i>result</i> widget .....	336
8.2.5.5. Demo4 .....	337
8.2.5.5.1. Restricting the visibility .....	338
8.2.5.5.2. Displaying predefined widgets .....	338
8.2.5.5.3. Validating .....	338
8.2.5.5.4. Using selection events .....	339
8.2.5.6. Demo5 .....	340
8.2.5.6.1. Using multi-page support .....	342
8.2.5.6.2. Validating .....	342
8.2.5.7. Demo6 .....	343
8.2.5.7.1. Restricting the visibility .....	343
8.2.5.7.2. Disabling the Undo/redo support in dialogs .....	344
8.2.5.7.3. Using hierarchical Sidebar labels .....	344



8.2.5.8. Demo7 .....	345
8.2.5.9. Demo8 .....	346
8.2.5.9.1. Using an XForm GUI description .....	347
8.2.5.9.2. Using a custom MementoFactory .....	347
8.2.5.9.3. Validating .....	348
8.2.5.9.4. Using the push service .....	348
8.2.5.10. Demo9 .....	348
8.2.5.10.1. Working with the demo .....	349
8.2.5.10.2. Placing guided configuration widgets on an external view or dialog .....	351
8.2.6. Common use cases .....	352
8.2.6.1. Registering a wizard .....	352
8.2.6.2. Registering the docking point .....	354
8.2.6.2.1. Adding an editor or dialog to the <b>Sidebar</b> view .....	354
8.2.6.2.2. Creating a custom module editor .....	356
8.2.6.2.3. Registering an unattended wizard .....	357
8.2.6.3. Restricting the visibility of a wizard .....	358
8.2.6.3.1. Via extension point .....	358
8.2.6.3.2. Via Backend implementation .....	359
8.2.6.4. Using XForms as GUI description .....	359
8.2.6.5. Creating the page .....	361
8.2.6.6. Creating a custom widget .....	362
8.2.6.7. Validating the data .....	363
8.2.6.8. Using selection events .....	364
8.2.6.9. Creating a MementoFactory .....	365
8.2.6.10. Using the push service .....	366
8.2.6.10.1. Registering a <i>push operation</i> .....	366
8.2.6.10.2. Sending a <i>push event</i> .....	367
8.2.6.11. Registering a custom result widget .....	367
8.3. Workflow API .....	368
8.3.1. Purpose .....	368
8.3.2. Prerequisites .....	368
8.3.2.1. Knowledge required .....	368
8.3.2.2. Plug-ins required .....	368
8.3.3. Design overview .....	369
8.3.3.1. Workflow description .....	369
8.3.3.2. Workflow hyperlinks .....	373
8.3.3.3. Workflow registration .....	373
8.3.3.3.1. Registration via extension point .....	374
8.3.3.3.2. Registration via workflows class .....	374
8.3.3.3.3. Registration via project .....	374
8.3.4. API .....	376
8.3.5. Commands .....	376

---

8.3.5.1. Config Editor Command .....	377
8.3.5.2. Generator Command .....	377
8.3.5.3. Importer/Exporter Command .....	377
8.3.5.4. Module Wizard Command .....	378
8.3.5.5. Sidebar Trigger Command .....	378
8.3.5.6. Autoconfigure Dialog Command .....	378
8.3.5.7. Autoconfigure Command .....	379
8.3.6. Demos .....	380
8.3.6.1. Setting up the demo plug-in .....	380
8.3.6.1.1. Making the Workflows view visible .....	381
8.3.6.2. Demo Overview .....	382
8.3.6.3. Demo step by step .....	382
8.3.6.3.1. Workflow <i>Create the project</i> .....	383
8.3.6.3.1.1. Precondition .....	384
8.3.6.3.1.1.1. Create a configuration project .....	385
8.3.6.3.1.1.2. Add required module .....	386
8.3.6.3.1.1.3. Import Autosar file .....	386
8.3.6.3.1.1.4. Inspect the changes .....	386
8.3.6.3.1.2. Workflow <i>Configure the project</i> .....	387
8.3.6.3.1.2.1. Module configuration .....	387
8.3.6.3.1.2.2. Calculate Handhelds .....	388
8.3.6.3.1.2.3. Edit module configuration .....	388
8.3.6.3.1.2.4. Generate project .....	389
Bibliography .....	390
A. Third party license .....	391
Index .....	397



# Begin here

## 1. If you are upgrading from a previous version

- ▶ What's new in this EB tresos Studio version?

Refer to the document EB tresos Studio new and noteworthy, located in your EB tresos Studio <INSTALL\_PATH>/doc directory.

- ▶ Which known problems, fixed problems, incompatibilities to previous releases, limitations and restrictions have been reported for the current EB tresos Studio version?

Refer to the EB tresos Studio release notes, located in your EB tresos Studio <INSTALL\_PATH>/doc directory.

The documents EB tresos Studio new and noteworthy and EB tresos Studio release notes also contain information in changes in the APIs.

## 2. If you are looking for a particular API

All APIs are available in the respective topic oriented chapter, e.g. information on the Template-based Generator API is available in [Chapter 7, “Generating code”](#).

See [Section 3.1, “What is the Public Java API?”](#) for an overview about the provided APIs.

See [Section 3.3, “Public Java API documentation”](#) for information about how to access the Java documentation.

## 3. If you are a first-time EB tresos Studio developer

If this is the first time you extend the functionalities of EB tresos Studio, you may start with basic information on:

- ▶ Installing Eclipse, first steps in Eclipse and setting up a Public API demo plugin at [Chapter 4, “Working with Eclipse”](#).
- ▶ What is the Public Java API and which APIs are available for me? Refer to [Chapter 3, “Introduction”](#).
- ▶ Background information on all relevant concepts is available in the subchapters with the prefix Concepts:

- ▶ [Section 5.1, “Concepts: The XDM format”](#)
- ▶ [Section 5.2, “Concepts: Mapping AUTOSAR ECU Configuration to XDM”](#)
- ▶ [Section 6.1, “Concepts: The DataModel”](#)
- ▶ Getting started with application examples:
  - ▶ Information on installing a demo is available in [Section 4.9, “How to install a Public API demo plugin”](#).
  - ▶ Information on the particular demos is available in the corresponding topic oriented chapters, e.g. in [Section 8.2.1, “Purpose”](#) and [Section 8.3, “Workflow API”](#).
- ▶ How do I work with EB tresos Studio without extending any functionalities? Refer to the EB tresos Studio user's guide, which is available in your <INSTALL\_PATH>/doc/2.0\_EB\_tresos\_Studio directory.

## 4. If you need help/more information

- ▶ Technical support

Receive technical support via email or phone from the support hotline.

- ▶ [Chapter 1, “About this documentation”](#)

Find out about:

- ▶ Required knowledge, tools, and system environment
- ▶ Typographical and style conventions used throughout this documentation. Defines usage of special fonts in the documentation.
- ▶ Naming conventions used in this documentation. Defines usage of special names and small/capital lettering in the documentation.
- ▶ EB tresos glossary

You do not understand what a word or abbreviations means? Find out in the EB tresos glossary.

- ▶ [“Bibliography”](#)

Would you like to read more detailed information? Find bibliographic references and further reading suggestions in the [“Bibliography”](#).



# 1. About this documentation

## 1.1. Introduction

Welcome to the EB tresos Studio documentation.

This chapter *About this documentation* provides you with

- ▶ Required knowledge, tools, and system environment
- ▶ Typographical and style conventions used throughout this documentation. Defines usage of special fonts in the documentation.
- ▶ Naming conventions used in this documentation. Defines usage of special names and small/capital lettering in the documentation.

---

**NOTE**

Note that there is a separate documentation for the EB tresos AutoCore if you also purchased and installed the EB tresos AutoCore modules.



## 1.2. Required knowledge and system environment

### 1.2.1. Required knowledge

If you want to configure an AUTOSAR stack with EB tresos Studio, you need to know the following:

- ▶ the EB tresos Studio user's guide
- ▶ the documentation of any other products from the EB tresos product line that you purchased along with EB tresos Studio
- ▶ the documentation of any MCAL products that you purchased along with EB tresos Studio

If you want to extend the EB tresos Studio functionality, you need to know the following:

- ▶ the EB tresos Studio developer's guide



- ▶ Java programming language
- ▶ basic knowledge about developing Eclipse plug-ins

## 1.2.1. Required system environment

Item	Requirement
Operating system	Microsoft Windows 10 (tested with Microsoft Windows 10 Enterprise 2016 (Version 10.0.14393 Build 14393)) or Linux Ubuntu, 64-bit (tested with Ubuntu 16.04 LTS)
Processor	Dual-core (minimal) Quad-core (recommended)
RAM	2 GB (minimal) 8 GB (recommended)
Connectors	USB port, if dongled licenses are used
Network	Network connection, if network licenses are used

Table 1.1. Minimal and recommended system requirements

## 1.3. Interpretation of version information

### 1.3.1. Product version number

Each product within the product line is assigned an individual *product version number*.

The product version number scheme is made of three parts:

1. The major number

An increment of the *major* version number indicates that the release contains major new features and changes compared to the previous major version.

2. The minor number



An increment of the *minor* version number indicates that the release contains minor new features and changes compared to the previous minor version.

### 3. The patch number

An increment of the *patch* version number indicates that the release fixes known problems.

The numbers are separated by dots in the following naming scheme: <major>. <minor>. <patch>.

Examples for spelled out product version numbers are:

- ▶ EB tresos AutoCore OS 6.0
- ▶ EB tresos AutoCore OS 6.1
- ▶ EB tresos AutoCore Generic 8.6
- ▶ EB tresos AutoCore Generic 8.7

You can find the product version number for example on the documentation cover and the release notes cover.

Within GUIs, e.g. in EB tresos Studio, you can find the product version number on the splash screen and the **EB tresos details** dialog next to the keyword *Studio*.

There may be a *qualifier* additionally to the product version number. This *qualifier* is provided in brackets and displayed after the product version number.

## 1.3.2. Qualification of basic software

Basic software of the EB tresos product line are the products: EB tresos AutoCore Generic and EB tresos AutoCore OS.

The quality level of the basic software is provided in the quality statement (Q statement), which is part of each delivery. In the quality statement, the application area is documented based on the quality level of the delivery. The different quality levels and their associated quality criteria are documented in the EB tresos AutoCore Generic Quality Level documentation.

The Quality Level documentation can be found in \$TRESOS\_BASE/doc/4.0\_EB\_tresos\_AutoCore\_Generic/AutoCore\_Generic\_Quality\_Level\_documentation.pdf where \$TRESOS\_BASE refers to the location of your EB tresos Studio installation.

The quality statement is provided in parallel to the installation file of the basic software and on the command server (EB Command) in the naming scheme: EB\_tresos\_AutoCore-quality\_statement-<RelName>-<Target>-B<BuildNr>(\_<Suffix>).doc, e.g. EB\_tresos\_AutoCore-quality\_statement-ACG-6.4-WIN32X86-B64208.doc



## 1.4. Typography and style conventions

The signal word **WARNING** indicates information that is vital for the success of the configuration.

---

**WARNING** **Source and kind of the problem**



What can happen to the software?

What are the consequences of the problem?

How does the user avoid the problem?

---

The signal word **NOTE** indicates important information on a subject.

---

**NOTE** **Important information**



Gives important information on a subject

---

The signal word **TIP** provides helpful hints, tips and shortcuts.

---

**TIP** **Helpful hints**



Gives helpful hints

---

Throughout the documentation, you find words and phrases that are displayed in **bold**, *italic*, or monospaced font.

To find out what these conventions mean, see the following table.

All default text is written in Arial Regular font.

Font	Description	Example
Arial italics	Emphasizes new or important terms	The <i>basic building blocks</i> of a configuration are module configurations.
Arial boldface	GUI elements and keyboard keys	<ol style="list-style-type: none"> <li>In the <b>Project</b> drop-down list box, select <b>Project_A</b>.</li> <li>Press the <b>Enter</b> key.</li> </ol>
Monospaced font (Courier)	User input, code, and file directories	<p>The module calls the <code>BswM_Dcm_RequestSessionMode()</code> function.</p> <p>For the project name, enter <code>Project_Test</code>.</p>



Font	Description	Example
Square brackets [ ]	Denotes optional parameters; for command syntax with optional parameters	insertBefore [<opt>]
Curly brackets {}	Denotes mandatory parameters; for command syntax with mandatory parameters	insertBefore {<file>}
Ellipsis ...	Indicates further parameters; for command syntax with multiple parameters	insertBefore [<opt>...]
A vertical bar	Indicates all available parameters; for command syntax in which you select one of the available parameters	allowinvalidmarkup {on off}

## 1.5. Naming conventions

The naming conventions listed below will enable you to understand meanings of words and word groups that may otherwise be confusing without explanation. This chapter is useful as a reference point if you are unsure what a specific spelling (e.g. capital letters in parameters, a word in boldface that appears on a screen) means. Browse through to see the conventions and return if you need an explanation for a phenomenon you do not understand.

### 1.5.1. AUTOSAR XML schema

Parameters spelled in capital letters (e.g. RECOMMENDED-CONFIGURATION-REF and BSW-MODULE-DESCRIPTION) refer to the AUTOSAR XML schema.

XML tags are presented as specified in AUTOSAR Model Persistence Rules for XML V2.1.2 R3.-0 Rev 0001, specifically in chapter 3.6 XML names [7]:

- ▶ All XML elements, XML attributes, XML groups and XML types used in the AUTOSAR XML schema are written in upper case letters.
- ▶ In order to increase the readability of the XML names, hyphens are inserted in the XML names which separate the parts of the names.

### 1.5.2. Configuration parameter names

All configuration parameter names used in EB tresos Studio could be used for code generation and therefore potentially conflict with names used in the EB tresos AutoCore.



---

**WARNING** To avoid naming conflicts, do not use any of the following names as configuration parameter name:



- ▶ any C keywords
  - ▶ name of AUTOSAR modules, e.g. Rte, Os, Com.
- 

## 1.6. EB tresos Studio naming conventions

### 1.6.1. EB tresos Studio command shell

System commands and properties in the EB tresos Studio command shell are spelled with mixed lower- and upper-case letters, e.g. -DmergeConfigs.

Where the AUTOSAR XML schema uses hyphens (–) to separate parameter parts, EB tresos Studio uses the camelback case to separate different parts of the parameters.



#### Example 1.1. Example instructions that use the AUTOSAR XML schema and the EB tresos Studio spelling convention

Define the `SHORT-NAME` of the parameter `ECU-PARAMETER`. Example: –  
`DecuParamDefName=<name>`

## 2. Safe and correct use of EB tresos Studio

### 2.1. Intended usage of EB tresos Studio

EB tresos Studio is intended to be used for configuring and generating EB tresos OsekCore and ECU basic software compliant to the AUTOSAR standard (see [www.autosar.org](http://www.autosar.org)).

### 2.2. Possible misuse of EB tresos Studio

- ▶ If you use the product in non-automotive projects or in automotive projects that are not based on OSEK or AUTOSAR technology (see [www.autosar.org](http://www.autosar.org)), the product and its technology may not conform to the requirements of your application. Elektrobit Automotive GmbH is not liable for such misuse.
- ▶ EB tresos Studio must only be used and extended as described in this EB tresos Studio developer's guide. If you use the APIs in a way or for purposes not described in this EB tresos Studio developer's guide, EB is not liable for this misuse or for the consequences of this misuse. Further claims are excluded explicitly.

### 2.3. Target group and required knowledge

- ▶ Automotive software engineers
- ▶ Programming skills and experience in programming AUTOSAR- and OSEK/VDX-compliant ECUs

### 2.4. Quality standards compliance

EB tresos Studio has been developed following processes that have been assessed and awarded ISO 9001:2008. These processes are based on Automotive SPICE. Should it be necessary to use any part of EB tresos Studio for a safety relevant project, contact EB first.



## 3. Introduction

This chapter is going to introduce you to the concepts of the Public Java API and will explain to you which kinds of API EB tresos Studio is shipped with and how you may extend your EB tresos Studio functionalities with these APIs.

### 3.1. What is the Public Java API?

The Public Java API is a set of Java classes and Eclipse extension points provided by EB for you to use and to extend the EB tresos Studio functionalities with. With the Public Java API you may:

- ▶ define your own dialogs, editors, wizards, etc.
- ▶ extend the EB tresos Studio command line
- ▶ make changes to the ECU configuration data model
- ▶ automate tasks
- ▶ generate code/write your own code generator
- ▶ write XPath functions
- ▶ etc.

The Public Java API consists of several APIs which are named according to their functionality:

#### Guided Configuration API

With this API you may extend the user interface of EB tresos Studio by defining your own GUI widgets such as dialogs, wizards, editors etc. For information on the Guided Configuration API see [Section 8.2, “Guided Configuration API”](#).

#### Workflow API

With this API you may create workflows that will be visible in the **Workflows** view of EB tresos Studio. These workflows may be created to semi-automate tasks by guiding the users through a specific task such as e.g. configuring a Com stack. For information on the Workflow API see [Section 8.3, “Workflow API”](#).

#### XPath API

XPath is a W3C-standard which can be used to navigate within the DataModel, select nodes and retrieve the values of nodes. To retrieve the value of a node, the node must be selected with a corresponding XPath expression. For information on the XPath API, see [Section 6.3, “XPath API”](#).

#### Custom XPath Functions API

Use this API to define your own XPath functions with which you may navigate within the DataModel. For information on the Custom XPath Functions API see [Section 6.4, “Custom XPath Functions API”](#).



## ECU Resource Manager API

With the ECU Resource Manager API you may register several values for ECU resources such as RAM and ROM. The ECU Resource Manager provides XPath functions to query these resource values. For information on the ECU Resource Manager API, see [Section 6.5, “ECU Resource Manager API”](#).

## Resources API

With the Resources API you may load, store and convert configuration data into standard file formats and merge data. For information on the Resources API see [Section 6.6, “Resources API”](#).

## System Model Access API

The System Model Access API provides support methods to access and modify the system model. For information on the System Model Access API see [Section 6.7, “System Model Access API”](#).

## Module Transformer API

With the Module Transformer API you may register module transformers with which you may convert an existing module configuration into another AUTOSAR version or into another derivative. For information on how to register a module transformer with the Module Transformer API, see [Section 5.6, “Module Transformer API”](#).

## Public Generator API

The Public Generator API provides you with a basic Java based Interface to Implement a Generator and/or Verifier. For information on Public Generator API, see [Section 7.3, “Public Generator API”](#). If you look for a more advanced Generator API have all Look at [Section 7.4, “NG Generator API”](#).

## Template-based Code Generator API

The DataModel provides a code generator which takes a code template, enhanced with particular macros that refer to variables in the configuration data, and which generates code. For information on this macro language with which code can be generated dynamically, see [Section 7.2, “Template-based Code Generator API”](#). If you look for a more advanced Generator API have all Look at [Section 7.4, “NG Generator API”](#).

## NG Generator API

With the NG Generator API you may define your own code generator. The NG Generator API contains more features than the External Generator API and the Template-based Generator API. For information on the NG Generator API see [Section 7.4, “NG Generator API”](#).

## External Generator API

With the External Generator API you may integrate external code generators into EB tresos Studio. The users can then start these integrated external generators via the EB tresos Studio GUI and via the EB tresos Studio command line. For information on the External Generator API see [Section 7.6, “External Generator API”](#).

## EPC File Generator API

With the EPC Generator API you may register and implement an EPC file generator with which you may generate EPC files. These EPC files are necessary if you want to work with external code generators. For information on the EPC File Generator API see [Section 7.5, “EPC File Generator API”](#).



#### Custom Attribute API

With the Custom Attribute API you may define your own custom attribute. You can store your non AUTOSAR related informations about a data node in this attribute. For information on the Custom Attribute API see [Section 6.8, “Custom Attribute API”](#).

## 3.2. What benefits do I have from using the Public Java API?

The main benefit of the Public Java API is that you may extend EB tresos Studio with functionalities of your needs. This guarantees quick results for standard use cases and saves you time because with the Public Java API you may reduce recurring, tedious tasks. Moreover, you will have solutions tailored exactly at your needs.

If you extend the EB tresos Studio functionality yourself, you become independent of the release cycles of EB tresos Studio. On the other hand, all upcoming EB tresos Studio releases will be syntactically and semantically backward-compatible, which means that APIs might be added but not changed. Some Methods are marked as `deprecated` may not be used. Those Methods may change its behaviour or may be removed without notice.

The Public Java API comes with various demos and documentation. Information on how to use the demos is located in the respective chapter of this EB tresos Studio developer's guide. If you wish to use the Public Java API, you will have to extract it first.

## 3.3. Public Java API documentation

The Java documentation is contained in your EB tresos Studio installation folder at `doc/2.0_EB_tresos_Studio/Public API Java.zip`. After you have unzipped the file, the Java documentation can be accessed by opening the file `index.html`.



---

**NOTE****Integrated Public Java API documentation**

The recommended way to access the Java documentation is to directly view it from within Eclipse. This is possible after you have installed the EB tresos Studio Eclipse tooling (see [Section 4.3, "How to install the EB tresos Studio Eclipse tooling"](#)).

The Java documentation is also included in the EB tresos Studio online help at [EB tresos Studio Public API Java](#). To access the online help, open the EB tresos Studio GUI and press F1.

---

## 3.4. How to use the Public Java API

To use the Public Java API you must always perform the following two steps:

1. Register the API in the corresponding extension point of the file `plugin.xml`.
2. Implement the Public Java API class.

## 3.5. Restrictions to the usage of the Public Java API

In general, the public Java API is not to be considered thread-safe, unless explicitly stated otherwise in the Java documentation.

If you decide to execute code in parallel (e.g. by scheduling multiple threads), you have to ensure that any call to the EB tresos Studio public Java API is synchronized in a way that it is never executed in parallel.



## 4. Working with Eclipse

### 4.1. Prerequisites

EB tresos Studio is built on Java and Eclipse, and you also need these tools to develop extensions for it.

We recommend to use the *Eclipse IDE for Eclipse Committers 4.6.2* package. You can download the 64-bit version from the [Eclipse Downloads](#) page.

Windows [eclipse-committers-neon-2-win32-x86\\_64.zip](#).

Linux [eclipse-rpc-photon-RC3-linux-gtk-x86\\_64.tar.gz](#).

The Java version used for EB tresos Studio development is 8.0 Update 121. You should use the same version for the development of extensions. If the exactly same version is not available, later update levels of version 8.0 should also work. You can download the Java 8 SE Development Kit (JDK) for 64-bit Windows/Linux from the Oracle Technology Network [Java SE Downloads](#) page, i.e. from [Java SE Development Kit 8 Downloads](#). Make sure to download the JDK and not just the JRE to also have the Java sources for debugging purposes.

#### 4.1.1. General preparations to use the Public API

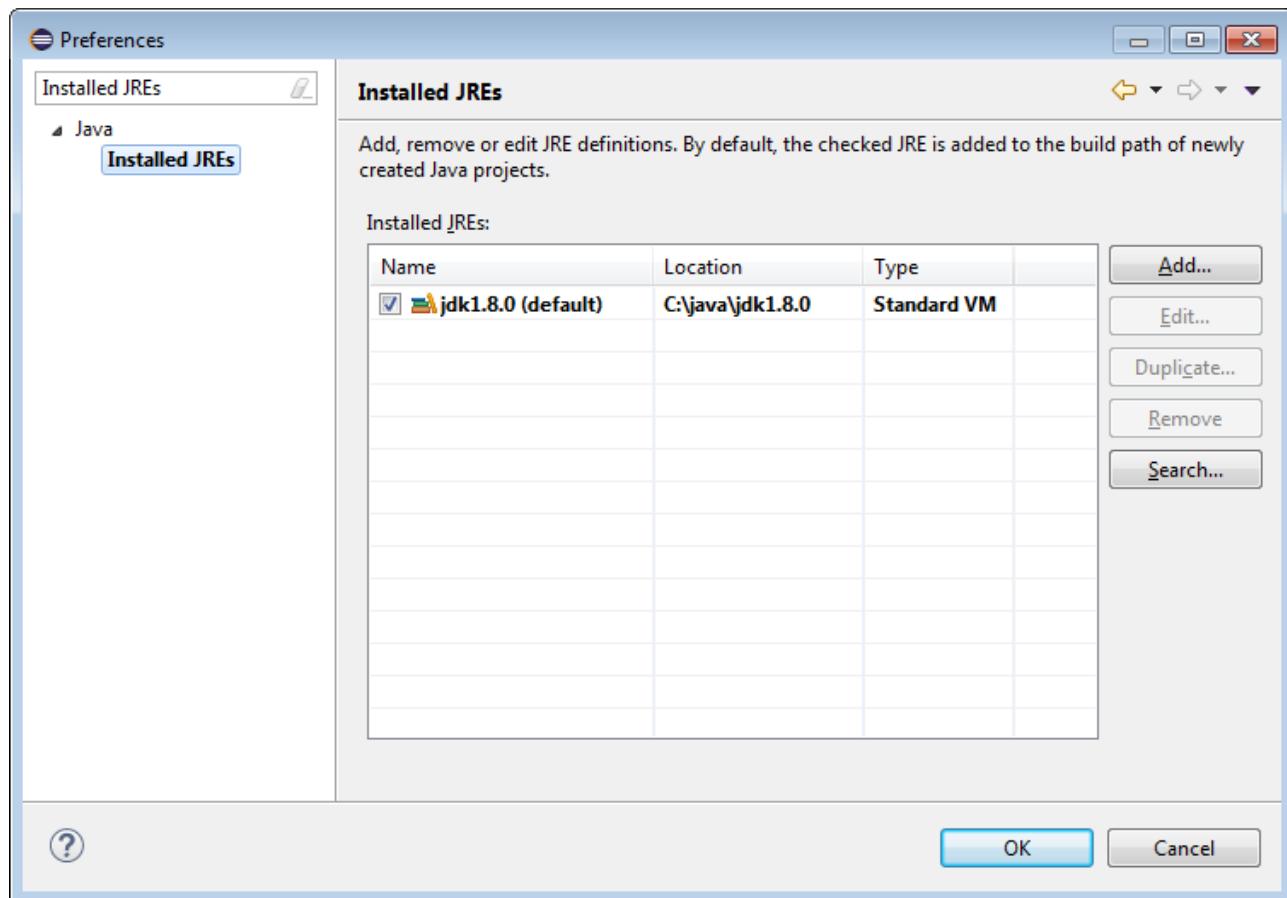
A plug-in which shall be fully Public API capable should have dependencies to the following EB tresos Studio plug-ins.

- ▶ dreisoft.tresos.autosar2.api.plugin
- ▶ dreisoft.tresos.datamodel2.api.plugin
- ▶ dreisoft.tresos.launcher2.api.plugin
- ▶ dreisoft.tresos.lib2.api.plugin
- ▶ dreisoft.tresos.generator.api.plugin

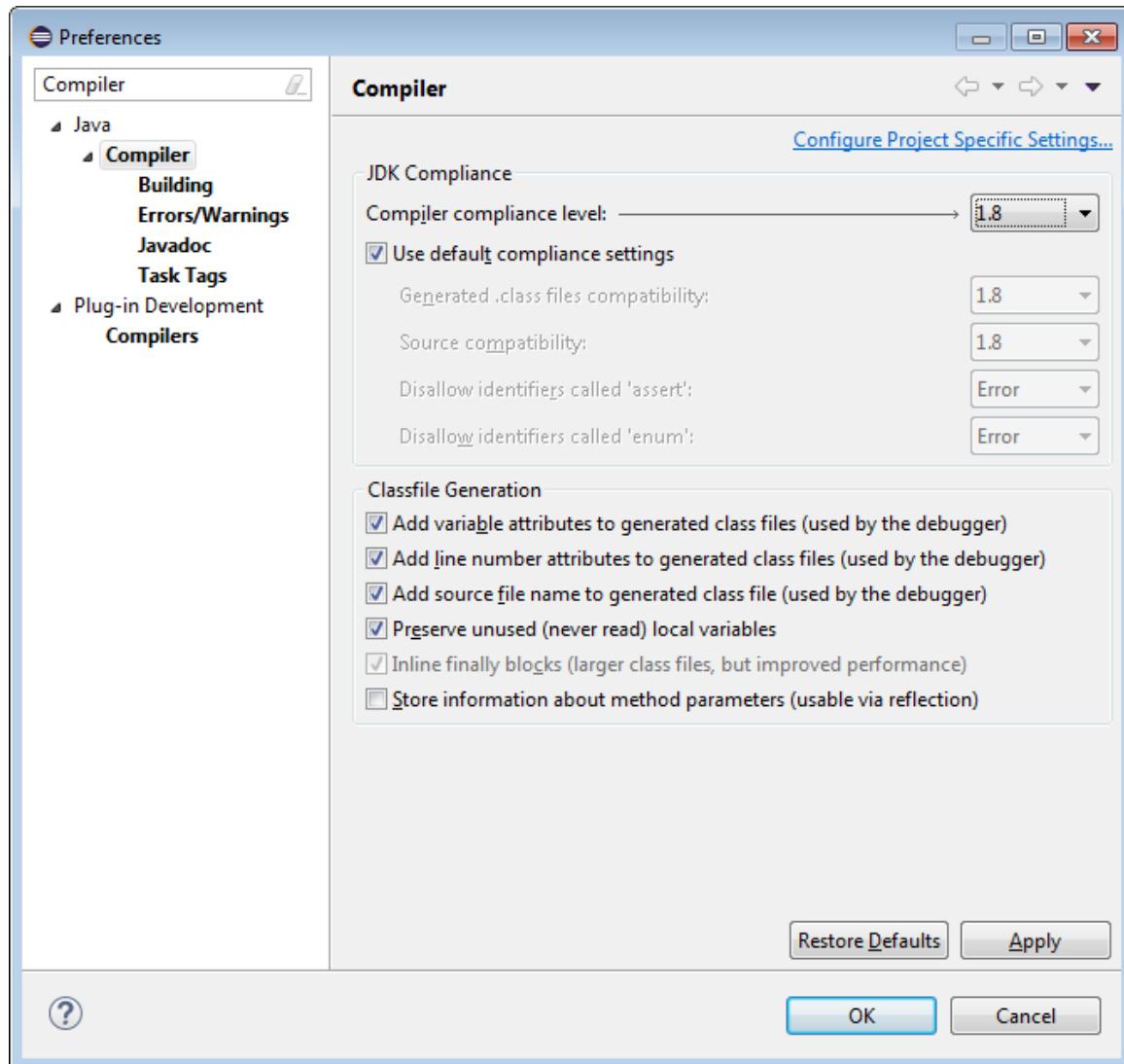
## 4.2. How to install Eclipse

To install Eclipse just unzip the downloaded zip file to your harddrive.

Now the Java environment has to be setup. Start Eclipse, go to *Window -> Preferences -> Java -> Installed JREs* and add a new JRE definition located at the home directory of your *JDK* installation. To avoid confusion, it is safe to remove any other existing JRE definitions.



Still in the **Preferences** dialog, navigate to **Java -> Compiler** and make sure that the Java compiler compliance level is set to 1.8.



## 4.3. How to install the EB tresos Studio Eclipse tooling

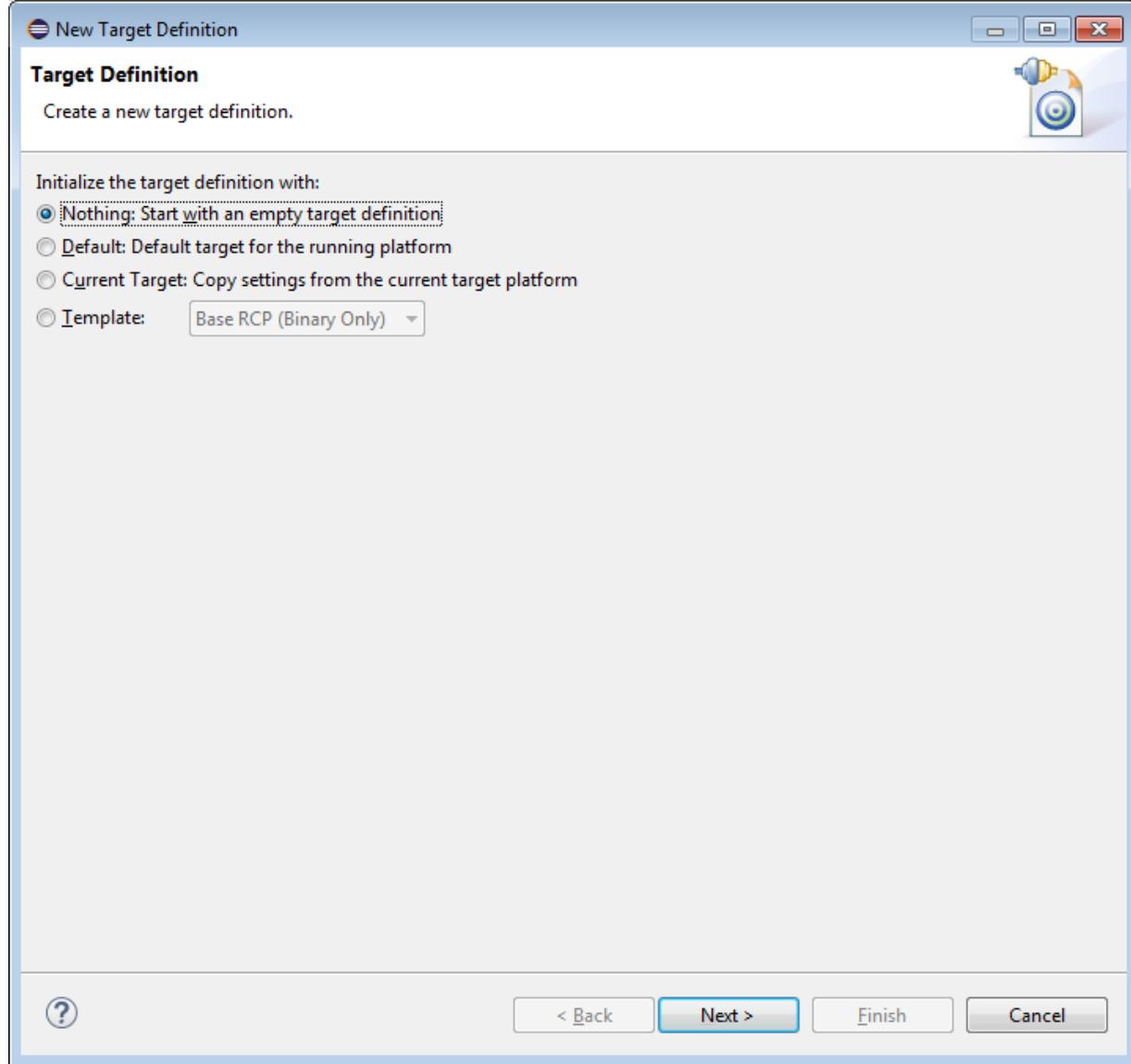
To use some special utilities that are necessary for programming with the EB tresos Studio Public API, go to the EB tresos Studio installation directory and unzip the file *tools/eclipse\_tools.zip* to your Eclipse installation. The zip-file contains several plugins.

## 4.4. How to set up the target platform

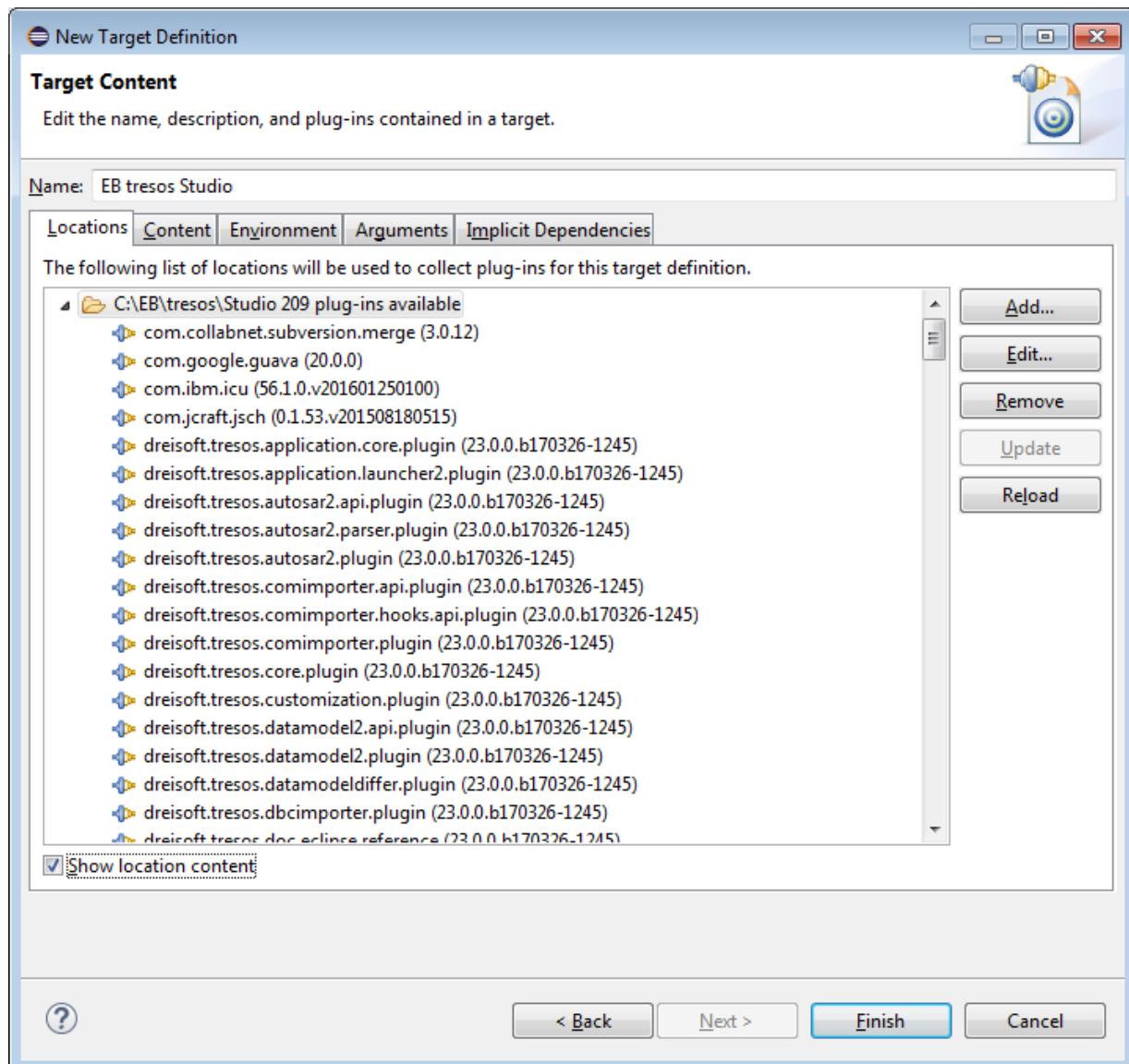


To be able to start EB tresos Studio from within Eclipse you need to set up the target platform accordingly. To do this, open the **Preferences** dialog on page *Window -> Preferences -> Plug-In Development -> Target Platform*.

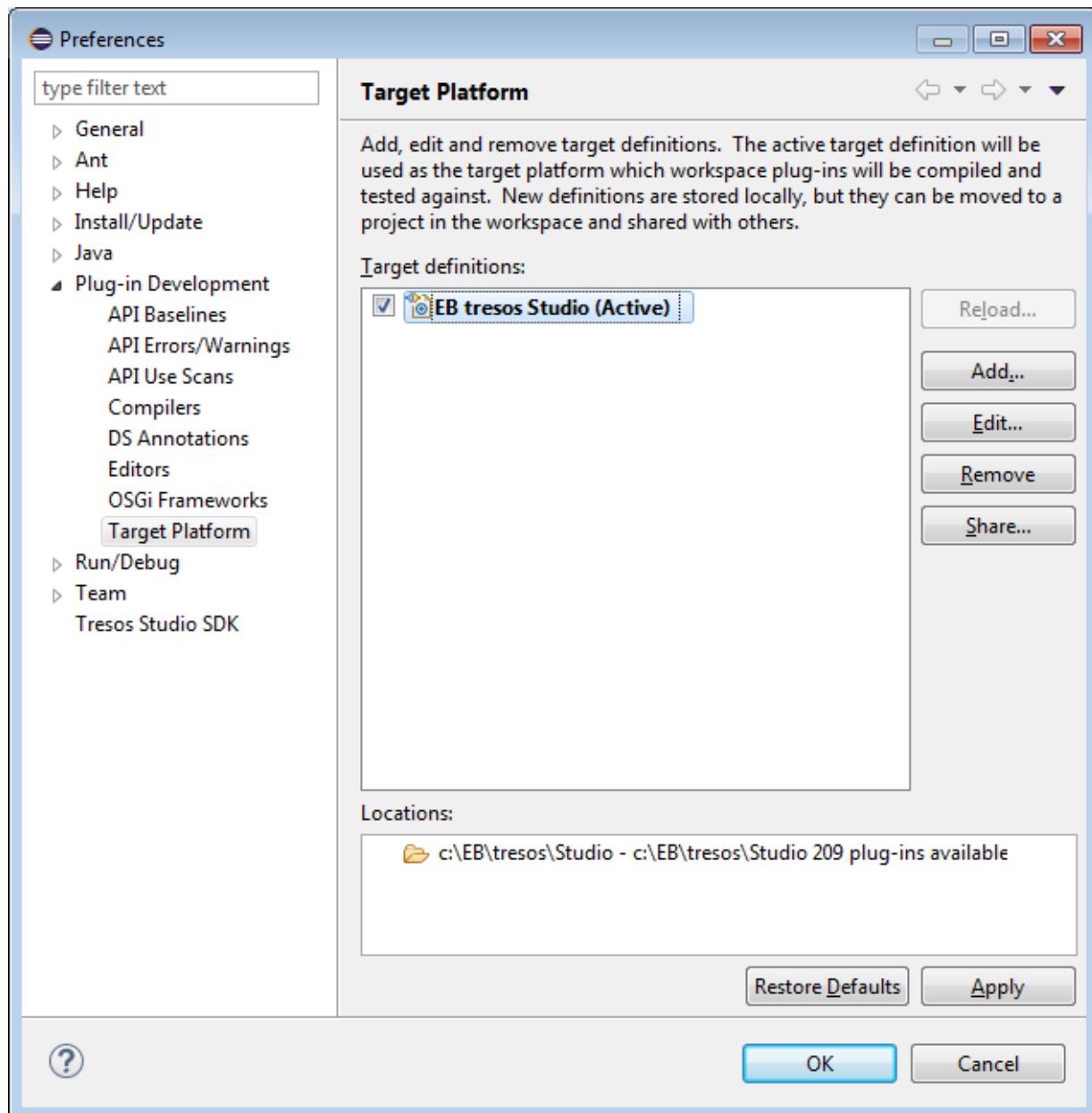
Now, press the **Add...** button to open the **New Target Definition** dialog, and press next to start with an empty target definition.



Enter "EB tresos Studio" into the **Name** field, and add the directory where you installed EB tresos Studio to the list of **Locations**. After selecting the option to **Show location content** the dialog should look like the following screenshot.

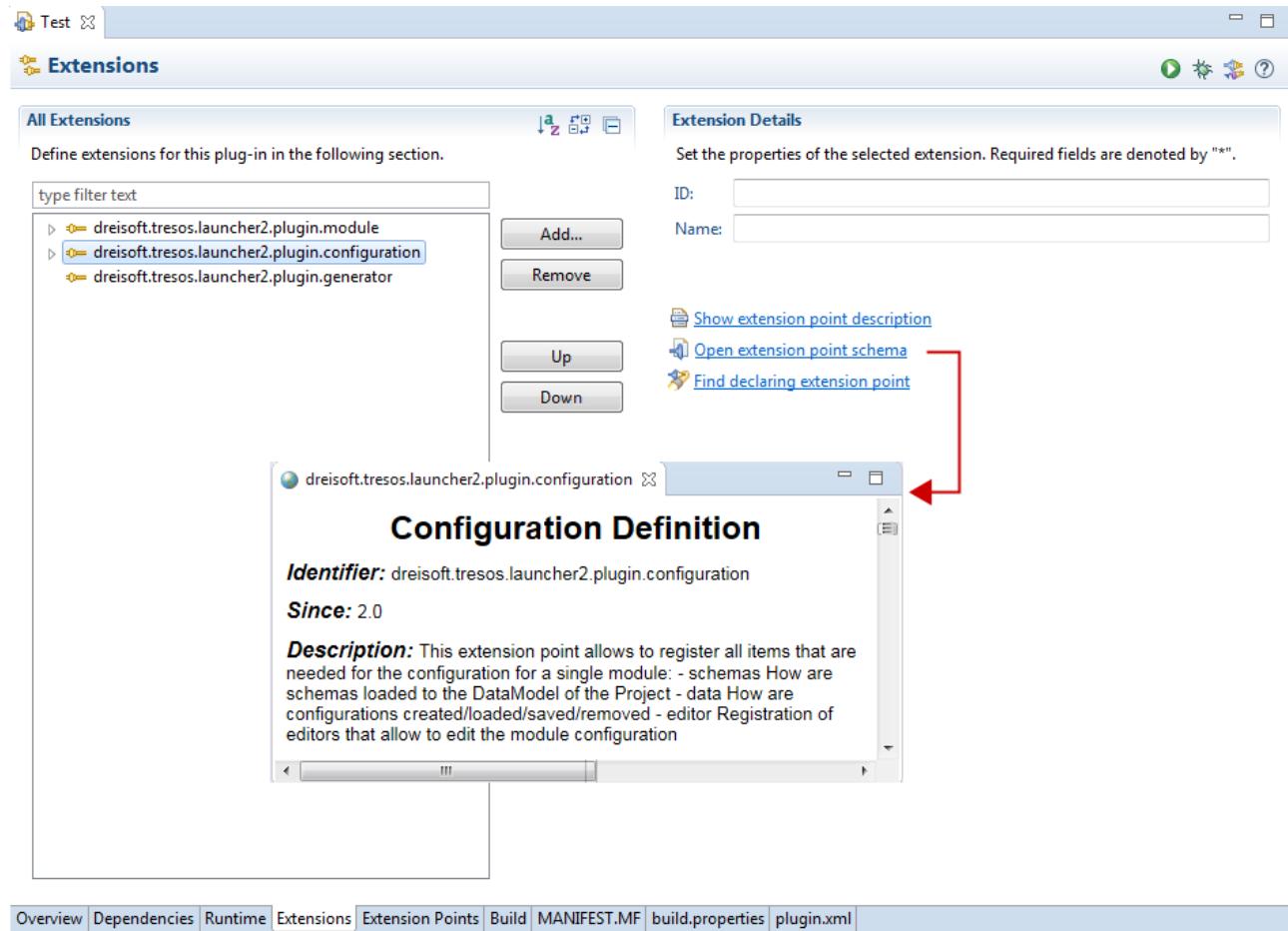


Click **Finish** to add the new target definition, and activate it in the **Preferences** dialog. To avoid confusion, you may want to remove any other target definition.



## 4.5. How to open extension point descriptions

If you select an extension within the PDE-editor, you can click on the link **Open extension point description** to get a detailed description of all elements which belong to it.



The description-editor contains a description of the extension point and each of its elements which should be read first before using it.



The screenshot shows the Eclipse PDE Editor interface. On the left, the 'Test' tab is active, displaying the following XML code:

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <eclipse version="3.4"?>
3 <plugin>
4   <extension
5     point="dreisoft.tresos.launcher2.plugin.module">
6     <module
7       allowMultiple="false"
8       categoryCategory="Com"
9       categoryComponent="ECUC"
10      categoryLayer="Driver"
11      categoryType="Can"
12      copyright="(c) 2006"
13      description="Textual Description"
14      id="Test.module1"
15      mandatory="false"
16      specVersionMajor="1"
17      specVersionMinor="0"
18      specVersionPatch="0"
19      swVersionMajor="1"
20      swVersionMinor="0"
21      swVersionPatch="0">
22     </module>
23   </extension>
24
25 </plugin>
26

```

On the right, the 'dreisoft.tresos.launcher2.plugin.module' tab is active, showing an example XML snippet:

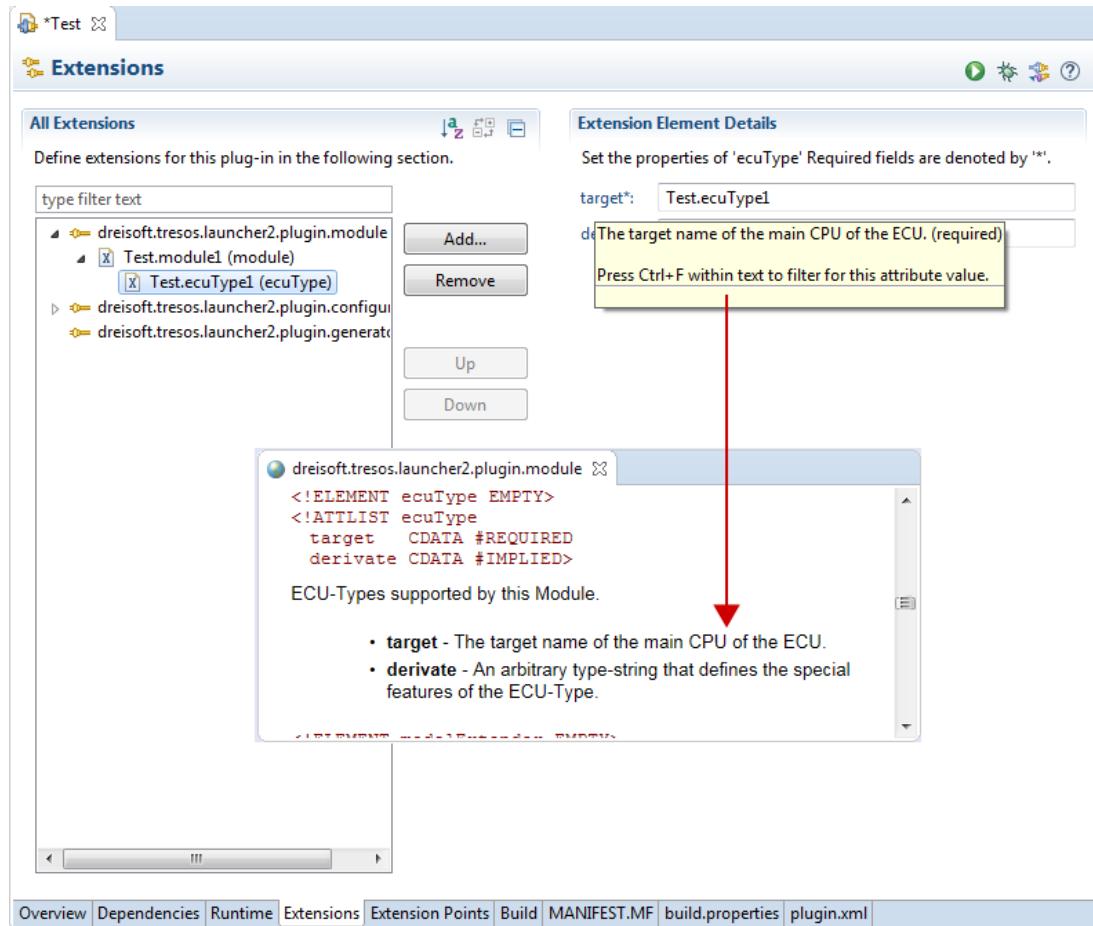
```

<extension id="mymoduleext"
label="My Module"
point="dreisoft.tresos.launcher2.plugin.module">
<module id="MyModule_TS16D4M2I0R0"
name="MyModule (TC1896)"
mandatory="false"
allowMultiple="false"
defaultConfigName="MyModule"
description="Can Driver Module for V850"
copyright="(c) 2006 Elektrobit Automotive GmbH"
swVersionMajor="1"
swVersionMinor="0"
swVersionPatch="0"
specVersionMajor="2"
specVersionMinor="0"
specVersionPatch="0"
categoryComponent="ECUC"
categoryCategory="Com"
categoryLayer="Driver"
categoryType="Can">
<ecuType target="V850" derivate="V850EGP1"/>
<moduleDependency id="CanIf"
conflict="false"
recommend="false">
<moduleProperties categoryType="CanIf"
specVersionMajor=">=2"/>
</moduleDependency>
<moduleDependency id="Can2"
conflict="true">

```

A red double-headed arrow points between the two tabs, indicating they are linked.

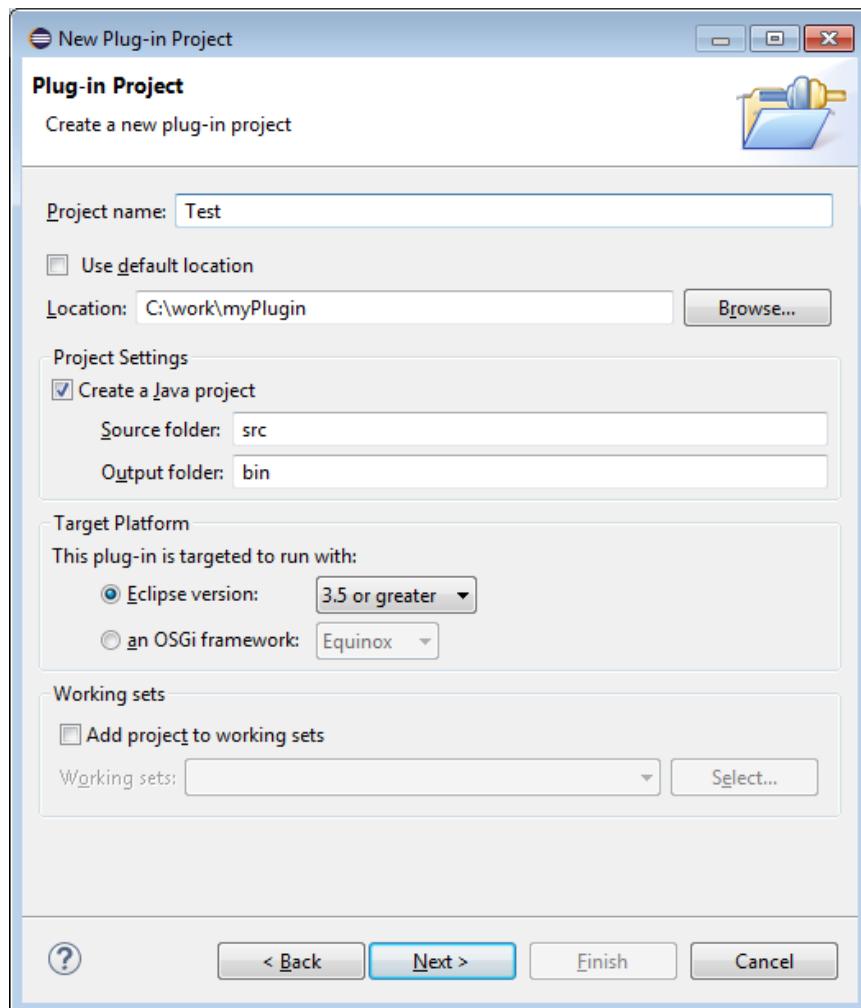
In most cases, there is an example at the bottom of the description-editor for each extension point. If you open the tab-page **plugin.xml** within your PDE-editor, you can compare your extension with this example, which is shown an xml-snippet.



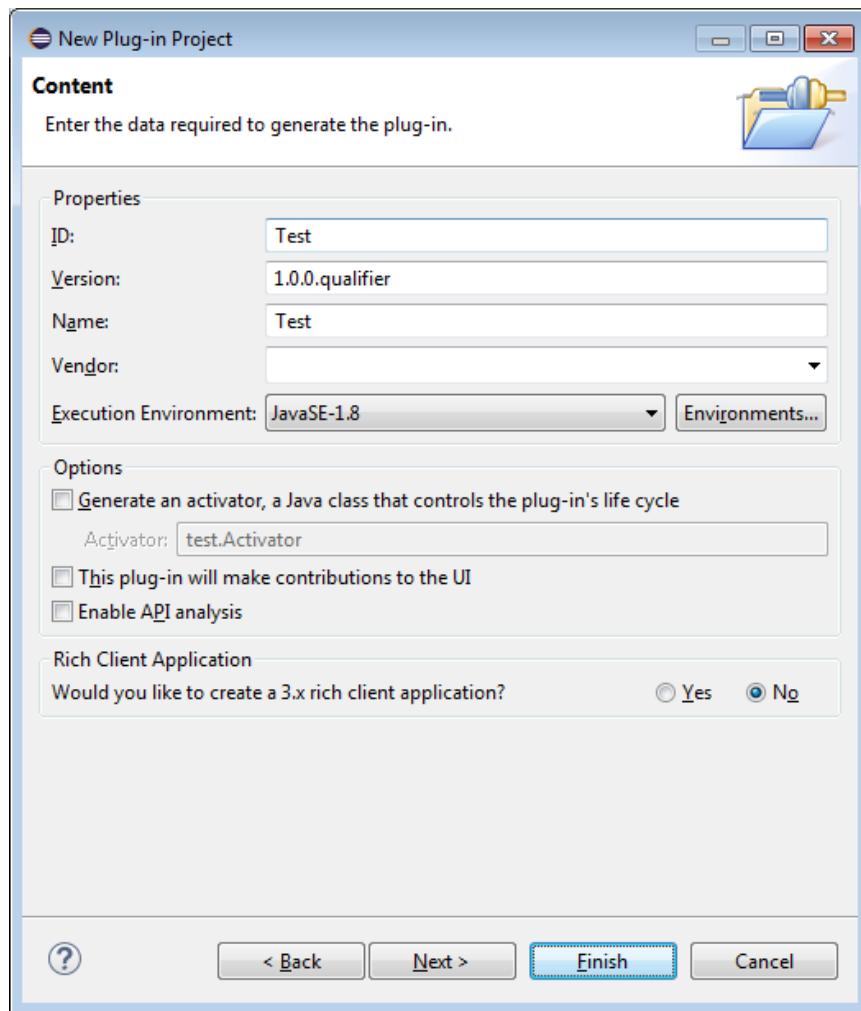
Additionally, the tool tip of each element contains the description of each element which is a short version of the description shown in the description-editor.

## 4.6. How to create a project with Eclipse

Start Eclipse and create a new Plug-in Project by selecting **File/New/Project...** and choose **Plug-in Project**.



Choose the directory, where you want to create your new plugin. You only have to enable the check box **Create a Java project** if you want to develop Java-classes.



Deselect the check boxes within the group **Options** and click **Finish**.

Within the PDE-editor, you can now change the configuration of your plugin and create extensions.



**Test**

**Overview**

**General Information**  
This section describes general information about this plug-in.

ID: Test  
Version: 1.0.0.qualifier  
Name: Test  
Vendor: Elektrobit  
Platform Filter:  
Activator:

Activate this plug-in when one of its classes is loaded  
 This plug-in is a singleton

**Execution Environments**  
Specify the minimum execution environments required to run this plug-in.

[Configure JRE associations...](#)  
[Update the classpath settings](#)

**Plug-in Content**  
The content of the plug-in is made up of two sections:

[Dependencies](#): lists all the plug-ins required on this plug-in's classpath to compile and run.  
 [Runtime](#): lists the libraries that make up this plug-in's runtime.

**Extension / Extension Point Content**  
This plug-in may define extensions and extension points:

[Extensions](#): declares contributions this plug-in makes to the platform.  
 [Extension Points](#): declares new function points this plug-in adds to the platform.

**Testing**  
Test this plug-in by launching a separate Eclipse application:

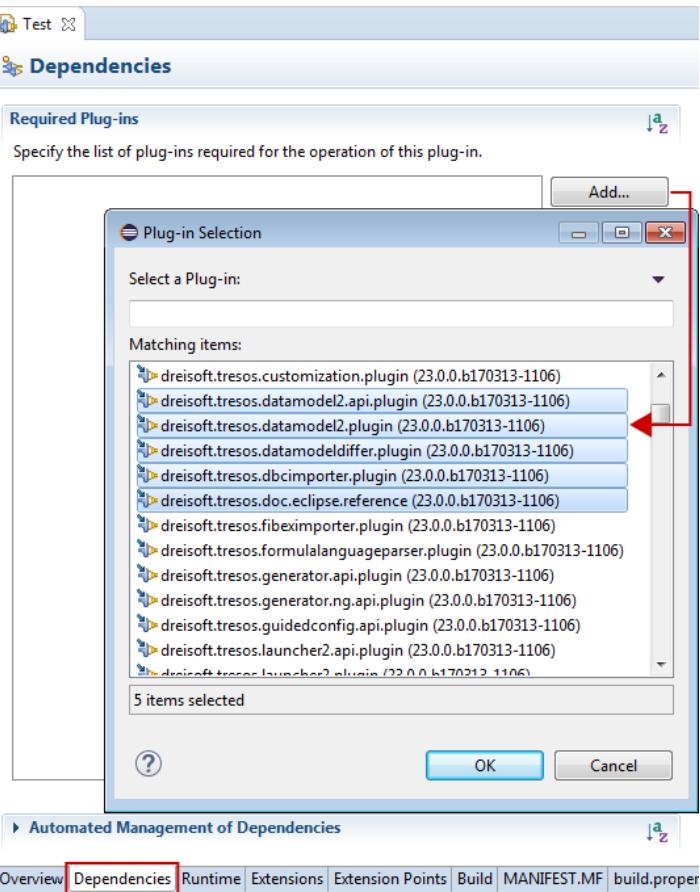
[Launch an Eclipse application](#)  
 [Launch an Eclipse application in Debug mode](#)

**Exporting**  
To package and export the plug-in:

- Organize the plug-in using the [Organize Manifests Wizard](#)
- Externalize the strings within the plug-in using the [Externalize Strings Wizard](#)
- Specify what needs to be packaged in the deployable plug-in on the [Build Configuration](#) page
- Export the plug-in in a format suitable for deployment using the [Export Wizard](#)

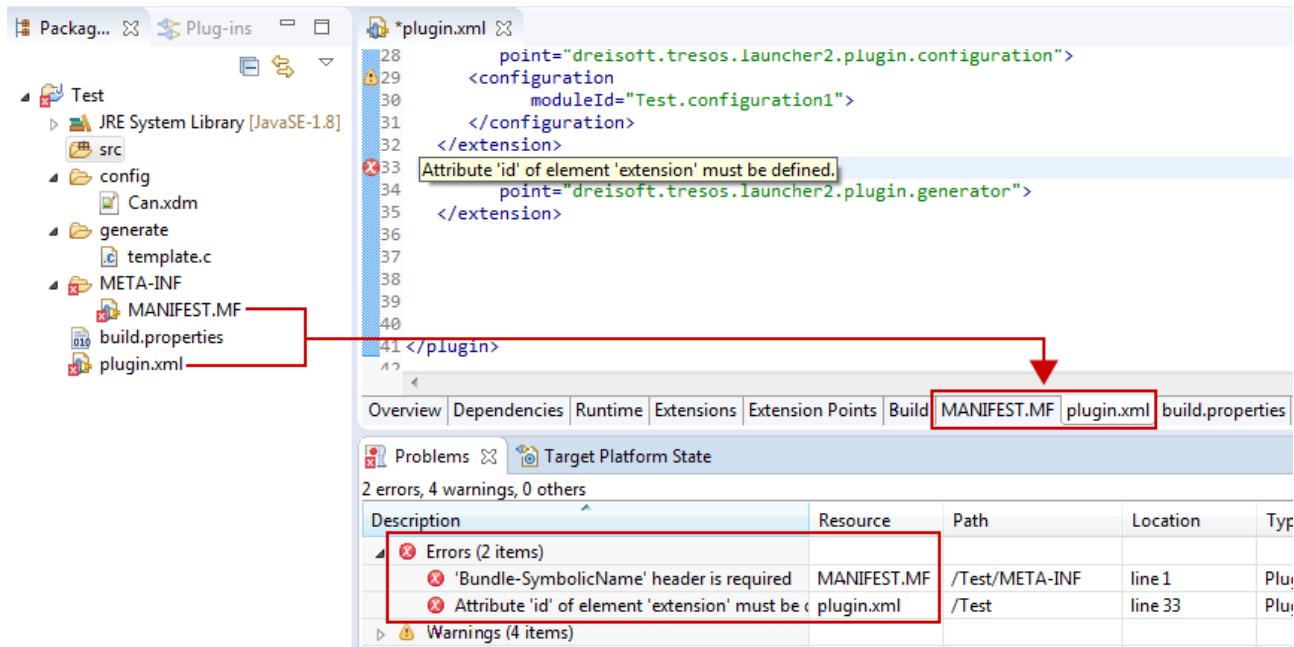
[Overview](#) | [Dependencies](#) | [Runtime](#) | [Extensions](#) | [Extension Points](#) | [Build](#) | [MANIFEST.MF](#) | [build.properties](#) | [plugin.xml](#)

Use the tab-pages at the bottom of your editor to switch to the "Dependencies" configuration-page.



Add all dreisoft.\* plugins to your dependencies. Now you can use all EB tresos Studio extension-points within your plugin.

#### 4.6.1. Errors and warnings within the Plugin Development Environment (PDE)



If there is an error within your `plugin.xml` or `MANIFEST.MF` file, the file is marked with a red overlay-icon in the Package Explorer. Additionally, all errors are listed within the Problems-view (where you can click one to jump to the error). A description for each error is provided within the Problems-view and as tool tip in the editor.

## 4.6.2. Creating a EB tresos Studio ModulePlugIn

### 4.6.2.1. Schema-file and templates

First, you need your vendor specific module definition (VSMD) as xdm-file. Just put the file into a new directory "config" in your plugin.

Next, you need your code-templates, which you put into the directory "generate". You can put any directory-structure with any files into this directory - all of them will be generated, taking on your directory-structure.

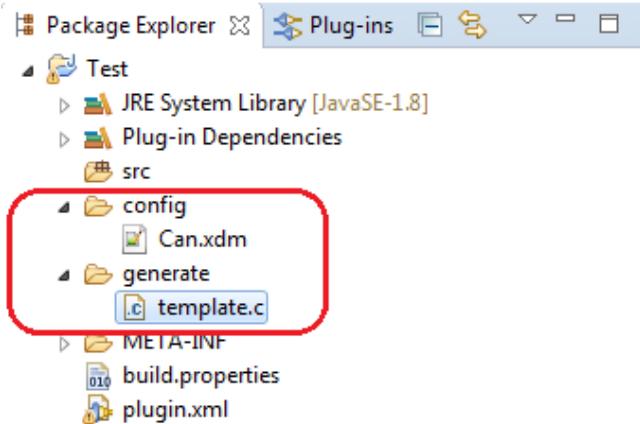


Figure 4.1. Files and folders

#### 4.6.2.2. Extension points

Since there is a detailed description available for each element of the extension points, this chapter only describes the most interesting extension points and their elements. To create a new ModulePlugIn, you have to add at least the following three extensions to your plugin.

For more details, see the description of these extension points. See also [Section 4.5, “How to open extension point descriptions”](#)

##### 4.6.2.2.1. Extension point dreissoft.tresos.launcher2.plugin.module

This extension point defines the module of your plugin.

The most important element to configure is the `ModuleId` which will be referenced by the configuration- and generator-extension point.

Since the `ecuType` is mandatory, create at least one by choosing **New/ecuType** from the context menu of the module.

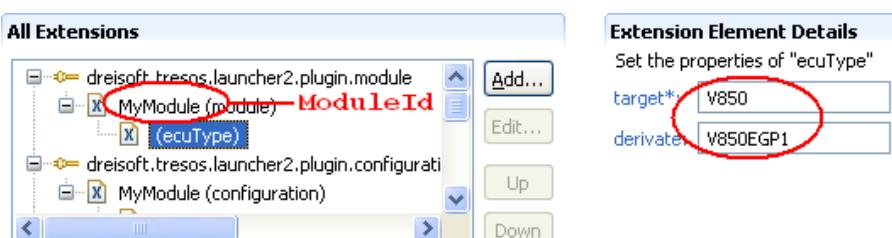


Figure 4.2. Module extension point



#### 4.6.2.2.2. Extension point dreisoft.tresos.launcher2.plugin.configuration

This extension point defines your schema loader, dataloader and editor for your ModulePlugIn.

Element	Description
configuration	Needs the id of your module defined before (ModuleId)
schema/manager	must be dreisoft.tresos.autosar2.resourcehandling.AutosarSchemaManager
schema/resource	the value must be the relative path to your schema-file, the type must be "xdm"
data/manager	must be dreisoft.tresos.autosar2.resourcehandling.AutosarConfigManager
data/schemaNode	the SHORT-NAME-path to your MODULE-DEFINITION prefixed with "ASPath:"
editor/class	must be dreisoft.tresos.launcher2.editor.GenericConfigEditor
editor/parameter	name must be "schema", the value must be the SHORT-NAME-path to your MODULE-DEFINITION prefixed with "ASPath:"



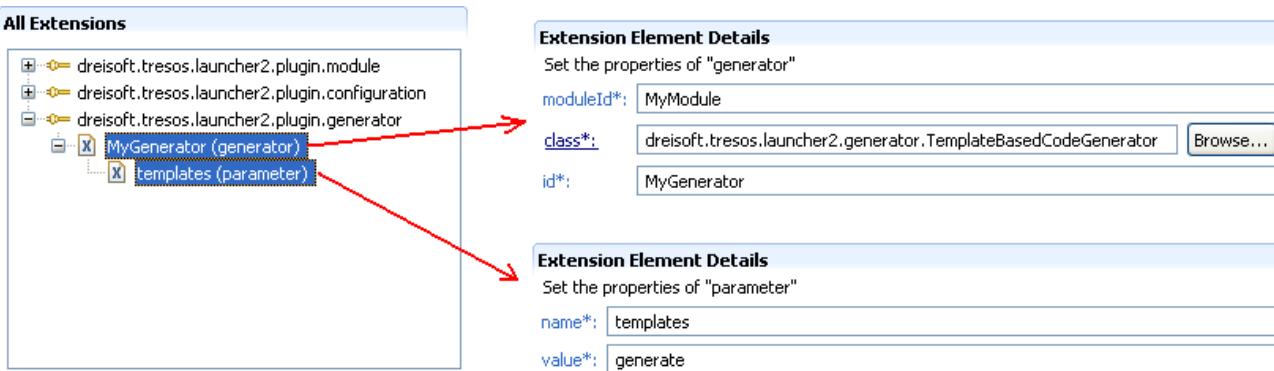
Figure 4.3. Configuration extension point

#### 4.6.2.2.3. Extension point dreisoft.tresos.launcher2.plugin.generator

This extension point defines the generator which can be used for your ModulePlugIn.

Again, you have to declare the id of your module (ModuleId) defined above. The class of your editor must be dreisoft.tresos.launcher2.generator.TemplateBasedCodeGenerator.

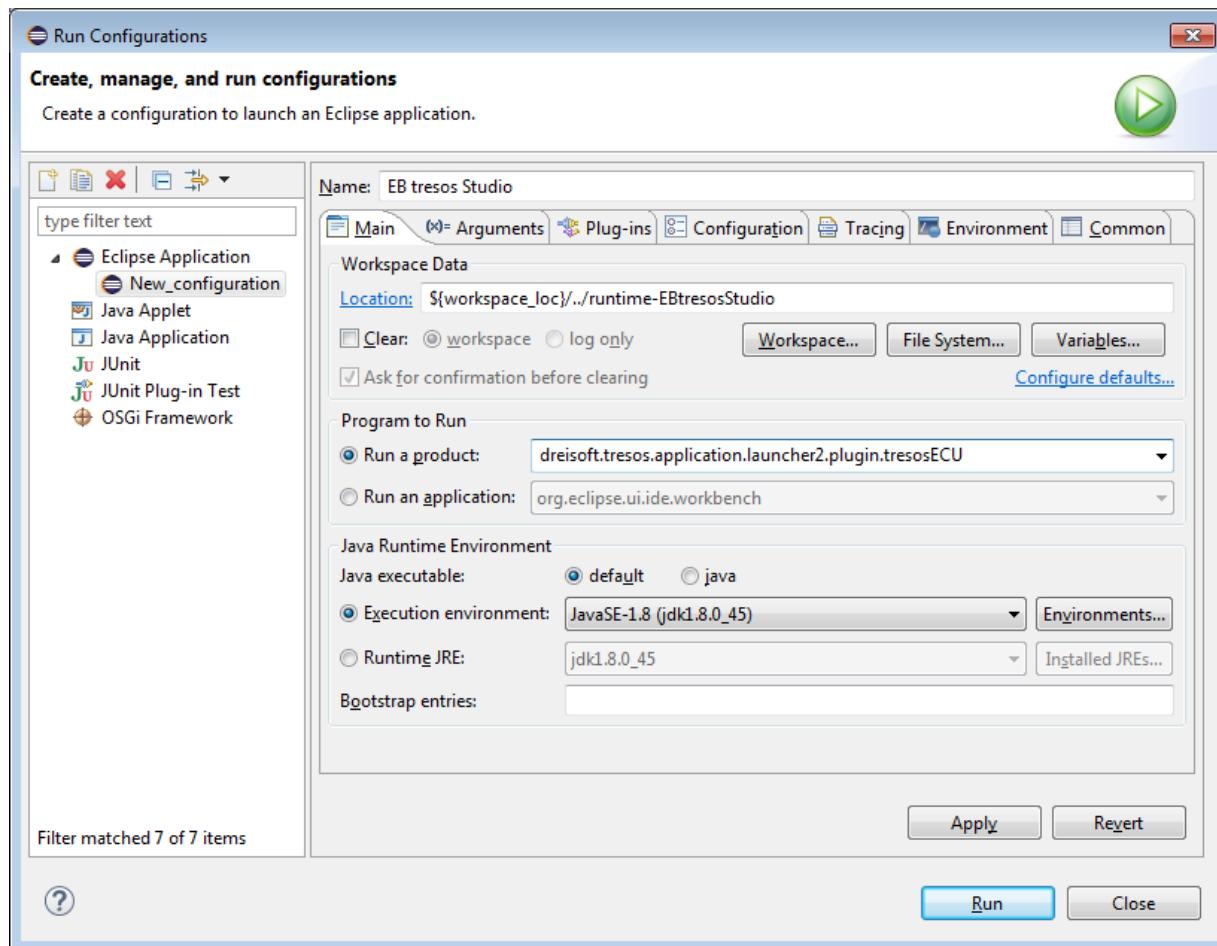
This generator needs the parameter "templates" with the relative path to your code-templates.



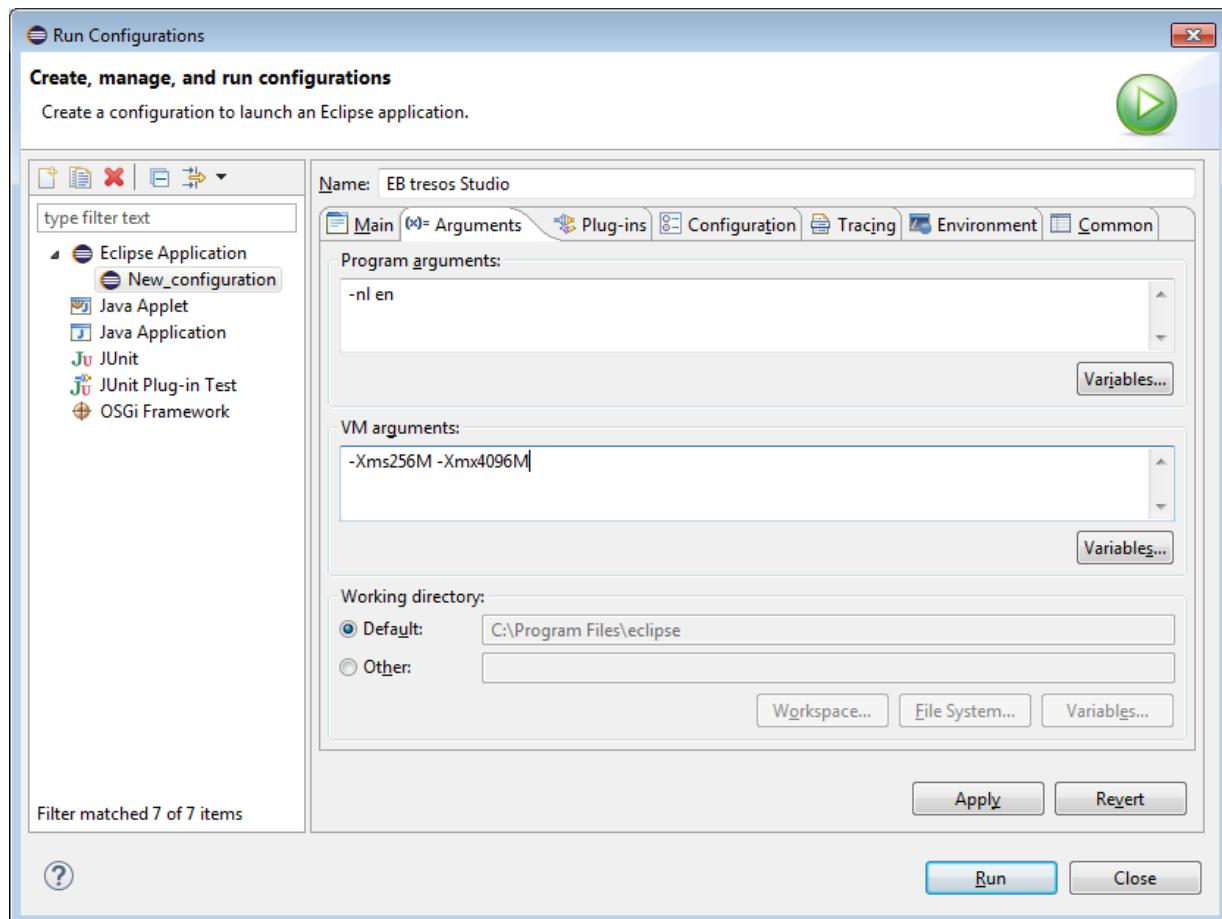
## 4.7. How to set up a run configuration

The last step for setting up Eclipse is to create a **Run Configuration** for starting EB tresos Studio directly from the IDE. To do so, choose **Run/Run Configurations...** from the menu. Double Click on *Eclipse Application* and give it a name.

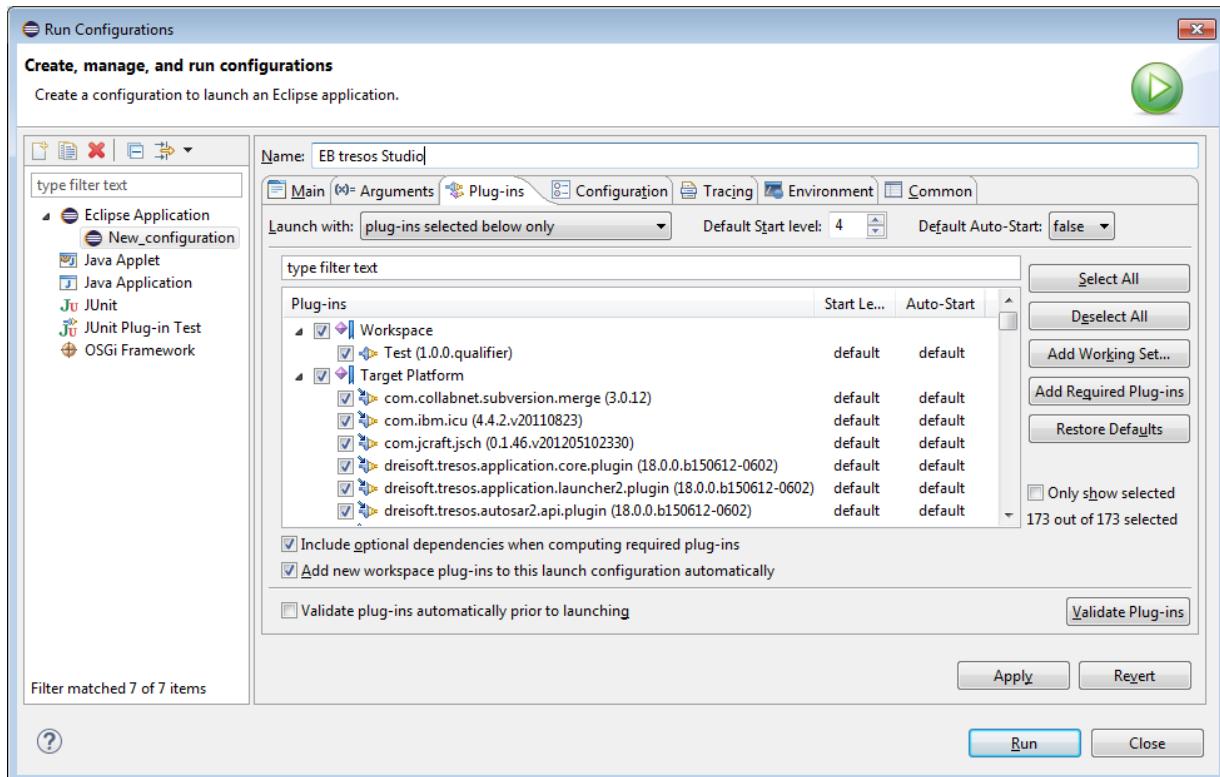
The following images show what has to be done to setup a correct **Run Configuration**.



Choose "dreisoft.tresos.application.launcher2.plugin.tresosECU" from the **Run a product:** combo box and adjust the Java Runtime Environment setting as shown in the picture above.



On the **Arguments** tab, set the **Program arguments** to `-nl en` and add `-Xms256M -Xmx4096M` to the **VM arguments**.



Set *Launch with* to *plug-ins selected below only*. Select *all* Target Platform plug-ins. From your Workspace select those plug-ins you actually want to run with EB tresos Studio.

## 4.8. How to define default preferences

EB tresos Studio manages many global preferences. Most of these preferences are available in the **Preferences** dialog.

To set preferences via the **Preferences** dialog,

- ▶ Click on the **Window** menu.
- ▶ Select the entry **Preferences**.

The **Preferences** window opens up.

However, some preferences need to be set via the configuration dialogs of several EB tresos Studio views. EB tresos Studio initially sets the preferences to reasonable defaults.

To supply different default values for preferences for different customers, it is possible to customize the defaults during installation of EB tresos Studio by installing a customized default preferences properties file in `$TRESOS_BASE/configuration/defaultPreferences.properties`.



A standard EB tresos Studio installation already provides a `defaultPreferences.properties` with standard default settings.

The properties file uses the Java property file format and has the following characteristics:

- ▶ Empty lines are allowed.
- ▶ Comments start with # as the first character in a line.
- ▶ Preferences are defined as tag/value pairs of the form `tag ":" value`.

The list of available preferences and allowed values is described in the `defaultPreferences.properties` file shipped with each EB tresos Studio installation.

## 4.9. How to install a Public API demo plugin

---

**NOTE**

*Precondition:* Eclipse has to be setup as described here [Section 4.3, “How to install the EB tresos Studio Eclipse tooling”](#)



---

All Public API examples in this developers guide can also be found in the demo folder `<tresos-install-dir>/demos/Studio`. The demo plugins can be imported to your Eclipse installation by right-clicking in the Package Explorer and selecting *Import... -> Import Existing Project Into Workspace*

Now the plugin is ready for use. You can also export it, to use it directly within the EB tresos Studio installation. To do so just right-click the plug-in project and hit export. Select *Deployable plug-ins and fragments* and hit *next*. Check the project, select a folder to where it shall be exported and uncheck the option *package plugins as individual JAR archives*. That's it. Now copy the exported plugin to your EB tresos Studio plugins folder. After a restart of EB tresos Studio, a new project can be created using the demo plugins.



# 5. Developing modules

## 5.1. Concepts: The XDM format

The native persistency format for storing a DataModel is the XDM-format. The XDM-format is an XML data-format that allows storing the node structure including all data (attributes, node names) of a DataModel on disk. The XDM format was also used in the first version of the DataModel. The XDM format presented in this chapter therefore is XDM version 7.0. See the EB tresos Studio user's guide, chapter [Content types for XDM files](#) for an overview of available XDM versions and corresponding EB tresos Studio versions.

### 5.1.1. Concepts

The XDM format stores:

- ▶ Nodes (schema-nodes, data-nodes...)
- ▶ Node Attributes

To keep the format extensible, multiple XML-Namespace are used to differentiate elements of the DataModel.

The DataModel supports different kinds of nodes (for which different name spaces are defined):

generic nodes:

Nodes that are neither schema-nodes nor data-nodes.

schema-nodes:

Schema-nodes define configuration parameters and the structure of configuration parameters. They define type, structure and attribute defaults for data-nodes.

data-nodes:

Data-nodes represent configuration parameters. A data node is *bound* to a schema-node that describes valid parameter ranges for the data-node.

### 5.1.2. Top level structure

An XDM-file is enclosed by the *datamodel* XML-tag:

```
'<datamodel'
```



```
'version="7.0"
[ 'xmlns="http://www.tresos.de/_projects/DataModel2/16/root.xsd' ]
[ 'xmlns:a="http://www.tresos.de/_projects/DataModel2/16/attribute.xsd' ]
[ 'xmlns:v="http://www.tresos.de/_projects/DataModel2/06/schema.xsd' ]
[ 'xmlns:d="http://www.tresos.de/_projects/DataModel2/06/data.xsd' ]
'>

...
'</datamodel>'
```

XML attribute	Description
version	Must be set to 7.0
xmlns	Declares the XML-namespaces for the tags of the DataModel. The namespace for generic node-tags and the datamodel-tag is:  http://www.tresos.de/_projects/DataModel2/16/root.xsd
factory	The name of the factory to be used to create nodes.

Child-tags:

- ▶ All node-tags

The datamodel-tag is not represented by its own node in the datamodel.

### 5.1.3. Nodes

Node tags are used to create nodes in the datamodel. There are various node tags available, one for each kind of node (container, list, variable). The XML-namespace is used to group nodes to categories (data-nodes, schema-nodes...). All node tags have a common form:

```
'<' namespace-prefix ':' core-node-type [ 'name="' node-name '"']
[ 'type="' type-string '"']
[ 'value="' value '"']
[ 'factory="' factory-name '"'
[ xmlns:...]
]
'>

'</' namespace-prefix ':' core-node-type '>'
```



The possible namespace-prefixes (e.g. 'd', 'v', 'a') are described in the chapters [Section 5.1.4, “Attributes”](#), [Section 5.1.5, “Generic nodes”](#), [Section 5.1.6, “Schema-nodes”](#) and [Section 5.1.7, “Data-nodes”](#).

See chapter [Section 5.1.8, “Factories”](#) for details about the factory and xmlns attributes.

XML attribute	Description
name	The name of the node. Any node can have a name. Some nodes require that their child nodes have a unique name.
type	The optional type attribute in addition to the node-tag determines the node type (the class that gets created for the node, allowed child nodes, attributes...).
value	Each node can have a value which is set with this attribute.
factory	The name of the factory to be used to create nodes.

Child-tags:

- ▶ All node-tags
- ▶ All attribute-tags

## 5.1.4. Attributes

Feature	Description
Namespace	<a href="http://www.tresos.de/_projects/DataModel2/16/attribute.xsd">http://www.tresos.de/_projects/DataModel2/16/attribute.xsd</a>
Default Prefix	a

Nodes can have attributes which add meta-data to a node. DataModel attributes must not be confused with XML-Attributes. The attributes here are added to a node via special XML-tags.

Attributes have a name and can have multiple string values.

Attributes come in three forms:

1. Passive attributes
2. Generic attributes
3. Automatic attributes

### 5.1.4.1. Passive attributes

Passive attributes are attributes for which the values are defined in the XDM-file:



```
'<a:da name="" attribute-name "" value="" attribute-value ""/>'

'<a:da name="" attribute-name "">
  '<a:v>' value '</a:v>'
  ...
'</a:da>'
```

In case of multiple values, or if the value contains line-feeds, the values are set via the `a:v` tag.

#### 5.1.4.2. Generic attributes

So-called generic attributes are able to store any (also quite complex) data structures. There is an own namespace for every attribute-type inside the `a:a` tag and therefore each generic attribute looks different. Thus, all currently defined generic attributes are explained in the following.

##### 5.1.4.2.1. AdminData

The generic attribute "ADMIN-DATA" is especially used to hold the complex data-structures of that attribute defined by Autosar in a generic way. For this, all TAGs and XML-attributes are taken over from the Autosar format and prefixed with the namespace "ad" in the following form:

```
<a:a name="ADMIN-DATA" type="ADMIN-DATA">
  <ad:ADMIN-DATA>
    <ad:LANGUAGE>DE</ad:LANGUAGE>
    <ad:USED-LANGUAGES>
      <ad:L-10 L="FR">francais</ad:L-10>
    </ad:USED-LANGUAGES>
    <!-- see Autosar format -->
  </ad:ADMIN-DATA>
</a:a>
```

Instead of the comment shown above, there may be more tags as defined by AUTOSAR.

##### 5.1.4.2.2. Implementation Configuration Variant

Each "MODULE-DEF" may contain the parameter "IMPLEMENTATION\_CONFIG\_VARIANT" which defines the supported configuration variants of the module via its RANGE. The configuration class of the parameter



itself is set to "PreCompile" for each variant to limit the changeability of the selected configuration variant. See the example below:

```
<v:var name="IMPLEMENTATION_CONFIG_VARIANT" type="ENUMERATION">
  <a:a name="IMPLEMENTATIONCONFIGCLASS" type="IMPLEMENTATIONCONFIGCLASS">
    <icc:v class="PreCompile">VariantPostBuild</icc:v>
    <icc:v class="PreCompile">VariantPreCompile</icc:v>
  </a:a>
  <a:a name="LABEL" value="Config Variant"/>
  <a:a name="UUID" value="ccf29ffd-fde7-41db-a5d8-4aeb7476d2e2"/>
  <a:da name="DEFAULT" value="VariantPostBuild"/>
  <a:da name="RANGE">
    <a:v>VariantPostBuild</a:v>
    <a:v>VariantPreCompile</a:v>
  </a:da>
</v:var>
```

If only one configuration variant is declared for the module, it may be omitted to set a configuration class for the "IMPLEMENTATION\_CONFIG\_VARIANT" parameter, see the example below:

```
<v:var name="IMPLEMENTATION_CONFIG_VARIANT" type="ENUMERATION">
  <a:a name="LABEL" value="Config Variant"/>
  <a:a name="UUID" value="ccf29ffd-fde7-41db-a5d8-4aeb7476d2e2"/>
  <a:da name="DEFAULT" value="VariantPostBuild"/>
  <a:da name="RANGE">
    <a:v>VariantPostBuild</a:v>
  </a:da>
</v:var>
```

### 5.1.4.2.3. Implementation Config Class

The attribute IMPLEMENTATIONCONFIGCLASS holds the class and the variant as value as shown below:

```
<a:a name="IMPLEMENTATIONCONFIGCLASS" type="IMPLEMENTATIONCONFIGCLASS">
  <icc:v class="PreCompile">VariantPreCompile</icc:v>
  <icc:v class="Link">VariantLinkTime</icc:v>
  <icc:v class="PostBuild">VariantPostBuild</icc:v>
  <icc:v class="PostBuildSelectable">VariantPostBuildSelectable</icc:v>
</a:a>
```



#### 5.1.4.2.4. Custom data attribute

The attribute `CUSTOMDATA` is used to hold custom attributes, which users may attach to data nodes, e.g. to tag a node with one or more **Labels**.

```
<a:a name="CUSTOMDATA" type="CUSTOMDATA">
  <cd:v id="Label">Some Label</cd:v>
  <cd:v id="Label">Another Label</cd:v>
</a:a>
```

There may be more tags as based on the number of custom attributes attached to a data node. To define your own custom data attributes, see [Section 6.8, “Custom Attribute API”](#).

#### 5.1.4.3. Automatic attributes

Automatic attributes calculate their own values by querying the configuration stored in the DataModel. Automatic attributes have a type that determines the algorithm via which the values are calculated and they are equipped with test expressions that describe how to calculate the values.

Automatic attributes are created with the following XML-tags:

```
'<a:da name="" attribute-name ""
  type="" attribute-type ""
  expr="" test-expression ""
  [ 'true=' true-value ''
  ]
  [ 'false=' false-value ''
  ]
  />'

'<a:da name="" attribute-name ""
  type="" attribute-type ""
  '>
    '<a:tst expr="" test-expression ""
      [ 'true=' true-value ''
      ]
      [ 'false=' false-value ''
      ]
      />
    ...
  '</a:da>'
```

The second form allows to define multiple test-expressions.



The Attribute type is mapped via the current DataModelFactory to an implementation. The type-attribute determines the type of the Automatic Attribute. The syntax of the XML-attribute test expression is determined by the type of the automatic attribute.

Each Automatic Attribute has one to n tests. A test is composed of a test expression and an optional fixed true value and an optional fixed false value. Each test evaluates to either no value or a list of values depending on the test expression. The test expression is evaluated by the attribute each time a value is requested from the attribute. If either `true` or `false` or both are set, the result of the expression is interpreted as a Boolean value. If the expression evaluates true, the value given with the `true`-attribute is returned. Otherwise the value of the `false`-attribute is returned. If the corresponding attribute is missing, no value is returned. If both, the `true` and the `false`-attributes are not set, the value of the expression is returned. The value of the whole Automatic Attribute is the list of all values returned by all individual tests.

By default the following types are supported:

- ▶ Bound
- ▶ Match
- ▶ Range
- ▶ XPath
- ▶ EcucCond
- ▶ EcucValidCond

#### **NOTE**



#### **Automatic attributes and configuration classes**

Automatic attributes must not access values of nodes with a later configuration class than the configuration class of the own node.

##### **5.1.4.3.1. Bound attribute**

The `Bound` attribute is the simplest form of an Automatic Attribute. The expression always evaluates the values of another node in the tree (the value of a variable, the member names of a map, the indices in a list, the variable names of a container). The bound node is referred to by an absolute or relative XPath expression.

Example:

```
<v:var name="Int1" type="INTEGER"/>
<v:var name="Int2" type="INTEGER"/>
  <a:da name="DEFAULT" type="Bound" expr="../Int1"/>
</v:var>
```

The `DEFAULT`-value of node `Int2` is now the value of `Int1`.



#### 5.1.4.3.2. Match attribute

The second form, the `Match` Attribute is a slightly more generic form of the `Bound`-attribute. It allows to compare the value of a node referred to by an XPath expression to a fixed value. The expression obeys the rules of a full XPath expression:

```
xPath-to-the-variable compare-operator 'string-value'  
xPath-to-the-variable compare-operator number-value
```

The variable to be bound to is selected with an XPath. The value of the variable is matched to the value following the operator. If the value is enclosed in single quotes, it is interpreted as a string. Otherwise it is interpreted as a number.

If the value is a string the following operators are supported:

=  
Returns true if the value of the XPath matches the value in the test.

!=  
Returns true if the value of the XPath does not match the value in the test.

If the value is a number the following operators are supported:

=  
Returns true if the value of the XPath matches the value in the test.

!=  
Returns true if the value of the XPath does not match the value in the test.

<  
Returns true if the value of the XPath is smaller than the value in the test.

<=  
Returns true if the value of the XPath is smaller or equal than the value in the test.

>  
Returns true if the value of the XPath is greater then the value in the test.

>=  
Returns true if the value of the XPath is greater or equal then the value in the test.

Example:

```
<a:da name="TITLE" type="Match">  
  <a:tst expr="//Var1 = 3"  
    true="Do not enter something"  
    false="Do enter something"/>  
</a:da>
```



The TITLE of the node which owns this attribute depends on the value of another node (//Var1).

#### 5.1.4.3.3. Range attribute

The Range attribute is used to define range checks. The Range attribute works similar to the Match attribute. It defines the range that the value of the node with the Range attribute may have. If the value is not within the defined range the Range attribute produces a corresponding error message as its value. The different tests expressions that are supported can be categorized as follows.

##### 5.1.4.3.3.1. Tests defining range boundaries

<valueOrExpression

Checks the value of the node to be numerically smaller than the value or the evaluated expression.

<=valueOrExpression

Checks the value of the node to be numerically smaller than or equal to the value or the evaluated expression.

>valueOrExpression

Checks the value of the node to be numerically greater than the value or the evaluated expression.

>=valueOrExpression

Checks the value of the node to be numerically greater than or equal to the value or the evaluated expression.

##### 5.1.4.3.3.2. Tests allowing single values

=valueOrExpression

Checks the value of the node to be numerically equal to the value or the evaluated expression.

'string literal'

Checks the value of the node to be equal to the given string literal. The surrounding apostrophes can also be omitted.

##### 5.1.4.3.3.3. Tests excluding single values

!=valueOrExpression

Checks the value of the node to be numerically unequal to the value or the evaluated expression.

You can specify a numerical constant or an XPath expression for all numerical tests. Numerical constants can be integers or 64bit floating point numbers.



Integers can optionally be enclosed in `i( )` as indicator. Negative integers can only be given in decimal notation, for positive integers also hexadecimal notation ('`0x<hex-number>`'), octal notation ('`\0<octal-number>`') or binary notation ('`b<binary-number>`') is allowed.

XPath expressions must evaluate to a numerical value. If the schema file shall be converted to the AUTOSAR format, EB tresos Studio must be able to evaluate the expression without accessing other nodes. For further information how to achieve this, see function `node:fallback()` in chapter [Section 6.3.4.5.5.1, "Node functions"](#).

The tests are treated with different semantics, depending on the type:

- ▶ All tests excluding single values must always match AND
- ▶ One of the following must be true
  - ▶ One of the tests allowing single values matches OR
  - ▶ All of the tests defining range boundaries match

If this restriction is not fulfilled, an appropriate error message is returned as node value. Each test type is only considered if at least one such test is defined. Thus if there are only tests allowing single values and none of them matches, an error will be reported.

e.g.

```
<a:da name="INVALID" type="Range">
  <a:tst expr=">=i(-15)" />
  <a:tst expr="<=i(30)" />
  <a:tst expr="!=0" />
</a:da>
```

The example shows the use of the Range-attribute. It states that the value of the node must be between -15 and 30 excluding the value 0.

#### NOTE

Do not confuse automatic attributes of type "Range" with the attribute "RANGE" which restricts the valid values of its node.



#### 5.1.4.3.4. XPath attribute

The fourth predefined Automatic Attribute is the `XPath` type. With this type the full power of the `XPath` language can be used (not only Path expressions as in the previous cases). The result of the `XPath` evaluation is used as the Attribute value. Caution: This is the slowest form of Automatic Attribute.



Example:

```
<a:da name="EDITABLE" type="XPath" expr="../Integer2 < 20"/>
```

The node now is only editable if the value of node "Integer2" is lower than 20.

#### 5.1.4.3.5. Multi test attribute

The fifth predefined Automatic Attribute is the `Multi` type. AUTOSAR 4.0 introduced new restriction tags for Strings (MIN-LENGTH, MAX-LENGTH, REGULAR-EXPRESSION) that together with MIN and MAX must be convertible from and to the AUTOSAR format even if there are XPath restriction defined for the parameter. The DataModel therefore allows to define multiple test expressions that are evaluated by different query languages in one attribute:

```
<a:a name="INVALID" type="Multi">

<!-- String length -->
<mt:length>
    <mt:tst expr="≥6"/>
    <mt:tst expr="≤10"/>
</mt:length>

<!-- defining the Autosar MIN/MAX range for a parameter -->
<mt:range>
    <mt:tst expr="≤255"/>
    <mt:tst expr="≥0"/>
</mt:range>

<!-- regexp for strings -->
<mt:regex>
    <mt:tst false="The value does not match the first regex."
        expr="H[ae]l{2}o.*"/>
    <mt:tst false="The value does not match the second regex."
        expr=".*(Du|Sie)!"/>
</mt:regex>

<!-- XPath expression -->
<mt>xpath expr=". ≥ 42"/>
</a:a>
```



#### 5.1.4.3.6. EcucCond attribute

**NOTE**
**Usage restriction**


The EcucCond automatic attribute can only be used in combination with the ENABLE data attribute!

The sixth predefined automatic attribute is the EcucCond type. With this type it is possible to use expressions (AUTOSAR EcucConditionSpecification) dependent on EcuC parameter values. The rules for how the condition is evaluated are not defined directly in the expression but the expression refers to one or more EcucQueryExpressions. Those can be located anywhere in the model and can be referred from several EcucCond attributes.

The following shows how an example from the *Specification of ECU Configuration AUTOSAR CP Release 4.-3.0* is converted to XDM.

Original:

```
<ECUC-COND>
  <CONDITION-FORMULA>
    <ECUC-QUERY-REF DEST="ECUC-QUERY">
      /path/to/the/definitionelement/GetTTCanEnabled
    </ECUC-QUERY-REF>
  </CONDITION-FORMULA>
  <ECUC-QUERYYS>
    <ECUC-QUERY>
      <SHORT-NAME>GetTTCanEnabled</SHORT-NAME>
      <ECUC-QUERY-EXPRESSION>
        value(
          refvalue(
            <CONFIG-ELEMENT-DEF-LOCAL-REF DEST="ECUC-BOOLEAN-PARAM-DEF">
              /path/to/some/other/definitionelement/CanIfSupportTTCAN
            </CONFIG-ELEMENT-DEF-LOCAL-REF>
          )
        )
      </ECUC-QUERY-EXPRESSION>
    </ECUC-QUERY>
  </ECUC-QUERYYS>
</ECUC-COND>
```

Converted XDM:

```
<a:da name="ENABLE" type="EcucCond">
  <f:formulaExpr>
    <f:formula>
```



```

        reference ("ECUC-QUERY:/path/to/the/definitionelement/GetTTCEnabled")
    </f:formula>
    <f:query name="GetTTCEnabled">
value(
    refvalue(
        localRef (
            "ECUC-BOOLEAN-PARAM-DEF:/path/to/some/other/definitionelement/CanIfSupportTTCAN"
        )
    )
)
</f:query>
</f:formulaExpr>
</a:da>
```

A parameter with this `ENABLE` attribute is only enabled if the referred query expression returns true.

The `GetTTCEnabled` query itself is evaluated from inside out:

1. The `localRef` specifies the **`EcucDefinitionElement`**.
2. The `refvalue` function delivers a set of elements from the *ECU Configuration Value* description which share the **`definition`** role of the provided **`EcucDefinitionElement`**.
3. In case of the `value` function, the `refvalue` may only return one element. Then the `value` function returns the parameter's value as a numerical value.

To prevent mixed content in XDM, some references will be converted in the following way:

from ARXML: `<TAG DEST="X">asPath</TAG>`

to XDM: `convertedTag ("X":absoluteAsPath)`

The following mapping list shows how the ARXML Tags get converted:

ARXML	XDM
ECUC-QUERY-REF	reference
ECUC-QUERY-STRING-REF	stringReference
CONFIG-ELEMENT-DEF-LOCAL-REF	localref
CONFIG-ELEMENT-DEF-GLOBAL-REF	globalref



---

**NOTE****Path resolving**

To refer parameters from the EcucQuery or the EcucQuery from the EcuCCCond attribute, it is necessary to do one of the following:

- ▶ Specify an absolute AUTOSAR path.
- ▶ Specify a relative AUTOSAR path in combination with a referenceBase.

In XDM only the absolute path is stored. When converting it back to the AUTOSAR format, also only the absolute path without referenceBase is stored.

---

Note that there are XPath expressions available to execute AUTOSAR formulas and ecuc query expressions: [Section 6.3.4.5.5, “Additional functions provided by the XPath Engine”](#). With that, it is e.g. possible to test them via the XPath console.

**5.1.4.3.7. EcucValidCond attribute**

---

**NOTE****Usage restriction**

The EcucValidCond automatic attribute can only be used in combination with the IN-VALID data attribute.

---

The seventh predefined automatic attribute is the EcucValidCond type. With this type it is possible to define validity constraints (AUTOSAR EcucValidationConditions). The rules for how the validation is performed are not defined directly in the expression but the expression refers to one or more EcucQueryExpressions. Those can be located anywhere in the model and can be referred from several EcucValidCond attributes.

The following shows how an example according to the *Specification of ECU Configuration AUTOSAR CP Release 4.3.0* is converted to XDM.

Original:

```
<ECUC-VALIDATION-COND>
  <ECUC-VALIDATION-CONDITION>
    <SHORT-NAME>ecucValidCond</SHORT-NAME>
    <VALIDATION-FORMULA>
      <ECUC-QUERY-REF DEST="ECUC-QUERY"> /path/to/the/definitionelement/GetTTCEnabled
      </ECUC-QUERY-REF>
    </VALIDATION-FORMULA>
    <ECUC-QUERYS>
      <ECUC-QUERY>
        <SHORT-NAME>GetTTCEnabled</SHORT-NAME>
        <ECUC-QUERY-EXPRESSION> value( refvalue( <CONFIG-ELEMENT-DEF-LOCAL-REF
          DEST="ECUC-BOOLEAN-PARAM-DEF">
```



```

        /path/to/some/other/definitionelement/CanIfSupportTTCAN
    </CONFIG-ELEMENT-DEF-LOCAL-REF> ) ) </ECUC-QUERY-EXPRESSION>
</ECUC-QUERY>
</ECUC-QUERYS>
</ECUC-VALIDATION-CONDITION>
</ECUC-VALIDATION-COND>
```

Converted XDM:

```

<a:da name="INVALID" type="EcucValidCond">
    <f:formulaExpr name="ecucValidCond">
        <f:formula>
            reference("ECUC-QUERY:/path/to/the/definitionelement/GetTTCanEnabled")
        </f:formula>
        <f:query name="GetTTCanEnabled">
            value(
                refvalue(
                    localRef (
                        "ECUC-BOOLEAN-PARAM-DEF:/path/to/some/other/definitionelement/CanIfSupportTTCAN"
                    )
                )
            )
        </f:query>
    </f:formulaExpr>
</a:da>
```

A parameter with this INVALID attribute is only valid if all validation formulas evaluate to true.

#### **NOTE**



It is possible to define more than one validation condition for an **EcucDefinitionElement**. The formula defined in the validation condition must yield a boolean expression depending on ecuc queries.

Note that there are XPath expressions available to execute AUTOSAR formulas and ecuc query expressions: [Section 6.3.4.5.5, “Additional functions provided by the XPath Engine”](#). With that, it is e.g. possible to test them via the XPath console.

#### 5.1.4.3.8. XDM attribute examples

##### 5.1.4.3.8.1. Check the value of another node



```
<v:var name="OnlyFor16MHz" type="INTEGER">
  <a:da name="ENABLE" type="XPath" expr="../Frequency = '16MHz'"
    true="true" false="false"/>
</v:var>
```

Only enables the node if the node 'Frequency' contains the value '16MHz'.

#### 5.1.4.3.8.2. Negation

```
<v:var name="Negation" type="INTEGER">
  <a:da name="INVALID" type="XPath" expr="not(. = ../OnlyFor16MHz)"
    false="May not be the same value as OnlyFor16MHz"/>
</v:var>
```

Marks the node as invalid if its value is not the same as 'OnlyFor16MHz'. In this case, the "false"-value is given to the user as an error.

#### 5.1.4.3.8.3. Logical Operators: and, or

```
<v:var name="BiggerThanMHz" type="INTEGER">
  <a:da name="INVALID" type="XPath">
    <a:tst expr="(.. / Frequency = '8MHz' and . > 8)
      or (.. / Frequency = '16MHz' and . > 16)
      or (.. / Frequency = '32MHz' and . > 32)">
      false="Value must be > the configured MHz"/>
  </a:da>
</v:var>
```

Restricts the value of the node depending on the string-value of another node. If the value violates the restriction, the "false"-value is given to the user as error.

#### 5.1.4.3.8.4. Range-check using the Match-attribute

```
<v:var name="SmallIntegerOnly" type="INTEGER">
  <a:da name="INVALID" type="Match">
    <a:tst expr=". >= 1" false="Value must be >=1 and <=9"/>
    <a:tst expr=". <= 9" false="Value must be >=1 and <=9"/>
```



```
</a:da>  
</v:var>
```

Marks the node as invalid if its value is not >=1 and <=9. In this case, the "false"-value is returned as an error.

#### 5.1.4.3.8.5. Range-check using the XPath-attribute

```
<v:var name="NegativeOrBigIntegerOnly" type="INTEGER">  
  <a:da name="INVALID" type="XPath"  
    expr=". &lt; 0 or . &gt; 9"  
    false="Value must be &lt;0 or &gt;9"/>  
</v:var>
```

Marks the node as invalid if its value is not <0 or >9. In this case, the "false"-value is given to the user as error.

#### 5.1.4.3.8.6. Only allow values configured in another list

```
<v:var name="OnlyValuesFromList" type="INTEGER">  
  <a:da name="INVALID" type="XPath"  
    expr="count(text:grep(..//PreConfiguredValues/*, concat('^',.,'$')))>0"  
    false="This node may only have values configured in list  
          'PreConfiguredValues'"/>  
</v:var>
```

Only the value configured within the list 'PreConfiguredValues' are valid values for this node. If another value is given, the "false"-value is given to the user as error.

#### 5.1.4.3.8.7. DEFAULT-value using the Bound-Attribute

```
<v:var name="DefaultToOnlyValuesFromList" type="INTEGER">  
  <a:da name="DEFAULT" type="Bound" expr="..//OnlyValuesFromList"/>  
</v:var>
```

Sets the DEFAULT-value of this node to the value configured for the node 'OnlyValuesFromList'. A **Calculate** button is shown to the user so that he can re-calculate the value on demand since it is only calculated once automatically.



#### 5.1.4.3.8.8. Only allowing certain values using the Range-Attribute

```
<v:var name="Only2or4or8or20" type="INTEGER">
  <a:da name="INVALID" type="Range">
    <a:tst expr="=2" false="Only the values 2, 4, 8 and 20 are allowed"/>
    <a:tst expr="=4" false="Only the values 2, 4, 8 and 20 are allowed"/>
    <a:tst expr="=8" false="Only the values 2, 4, 8 and 20 are allowed"/>
    <a:tst expr="=20" false="Only the values 2, 4, 8 and 20 are allowed"/>
  </a:da>
</v:var>
```

Only allows certain values for the node. If another value is configured, the "false"-value is given to the user as error.

#### 5.1.4.3.8.9. Modulus operator

```
<v:var name="DivisibleBy3" type="INTEGER">
  <a:da name="INVALID" type="XPath" expr=". mod 3 = 0"
        false="Value must be divisible by 3"/>
</v:var>
```

Marks the node as invalid if the configured value is not divisible by 3. In this case, the "false"-value is given to the user as error.

#### 5.1.4.3.8.10. Calculating with node-values

```
<v:var name="Mul2" type="INTEGER">
  <a:da name="INVALID"
        type="XPath"
        expr=". * ../Mul1 &gt;= 5 and . * ../Mul1 &lt;= 25"
        false="Mul1 * Mul2 must be between 5 and 25"/>
</v:var>
```

The value of the node multiplied with the value of node 'Mul1' and must be between 5 and 25. If not, the "false"-value is given to the user as error.

```
<v:var name="Mul3" type="INTEGER">
  <a:da name="INVALID"
```



```

    type="XPath"
    expr=". * ../Mul1 = 5
          or . * ../Mul1 = 10
          or . * ../Mul1 = 20
          or . * ../Mul1 = 40"
    false="Mul1 * Mul3 must be one of 5 10 20 or 40"/>
</v:var>
```

The value of the node multiplied with the value of node 'Mul1' must be 5, 10, 20 or 40. If not, the "false"-value is given to the user as error.

#### 5.1.4.3.8.11. Regular expressions

```

<v:var name="Regex" type="STRING">
  <a:da name="INVALID" type="XPath">
    expr="text:grep(., '^[A-Z][a-zA-Z0-9]*$')"
    false="Must start with a capital letter"/>
</v:var>
```

Marks the value of the node as invalid if it does not match a regular expression. In this case, the "false"-value is given to the user as error.

#### 5.1.4.3.8.12. References

```

<v:ref name="Reference" type="REFERENCE">
  <a:da name="REF" value="ASPathDataOfSchema:/AUTOSAR/Adc/Container/Int"/>
  <a:da name="RANGE" type="XPath">
    <a:tst expr="text:difference(
      node:paths(node:refs('ASPathDataOfSchema:/AUTOSAR/Adc/Container/Int')),
      node:paths(node:refs(node:refs(
        'ASPathDataOfSchema:/AUTOSAR/Adc/Container/Reference4'
      )))
    )"/>
  </a:da>
  <a:da name="INVALID"
    type="XPath"
    expr="node:refvalid(.)"
    false="The configured node does not exist or may not be referenced."/>
</v:ref>
```



Both, the REF- and the RANGE-attribute can be used to define which nodes are valid to be configured by the user. If none of these attributes is given, any node can be configured. If both attributes are given, the intersection of the returned values is calculated.

The REF-attribute may only hold paths (no xpath-expression). In the example above, all configured values (DataNodes) for the SchemaNode with the AUTOSAR ShortName-path "/AUTOSAR/Adc/Container/Int" are allowed to be configured.

The RANGE-attribute is an automatic attribute and thus may contain complex xpath-expressions. The example above does not allow to configure a node which is referenced by a ReferenceNode of SchemaNode "/AUTOSAR/Adc/Container/Reference4".

The INVALID-attribute in the example above creates an error if the node does not reference a valid node (defined by REF and/or RANGE). By default, only a warning is created in that case.

Additional Explanation: To get the nodes of a prefixed path (e.g. ASPathDataOfSchema), the path must be surrounded by apostrophes and give in to the function "node:ref". To get the target-nodes of a ReferenceNode, also the function "node:ref" must be used. Thus, the example above uses this function twice. Because we do not need nodes, but paths, the function "node:paths" must be used. Finally, we have two lists of paths from which we need the "remaining quantity". This is done by "text:difference". Alternatively you could use "node:difference" to the list of nodes and surround it by "node:paths".

#### 5.1.4.3.8.13. Preconfigured values for list-entries

```
<v:lst name="PreConfiguredValues">
  <v:var name="Int" type="INTEGER">
    <a:da name="DEFAULT" type="XPath" expr=
"text:split('1 2 4 8 10', ' ') [position()-1 = node:current() / @index]">
    />
  </v:var>
</v:lst>
```

The DEFAULT-value of each value added to the list depends on its index within the list. By this several pre-configured values can be defined.

#### 5.1.4.3.9. Restrictions

There are some things you cannot do with automatic attributes in the DataModel:

- ▶ There is no way to calculate a value of a node all the time, so it is always up to date. Instead, you can use the DEFAULT-attribute which gives the user the possibility to recalculate the value of the node. For this all nodes with an automatic DEFAULT attribute have a **Calculate** button.



- ▶ You cannot use the attribute LABEL as automatic attribute. The LABEL of an entry-field always has a static value. The only automatic value may be the TITLE of the page.
- ▶ To avoid infinite loops, you cannot access a node from within its own ENABLE attribute. This happens usually accidentally, e.g. if you try to reference to a sibling with a wildcard:

```
<v:var name="abc">
  <a:da name="ENABLE" type="XPath" expr="./*[@name='xyz'] = 3"/>
</v:var>
```

In this case you would get the following error message:

```
(1092) Detected a possible infinite loop because the expression accesses
its own node: /.../abc in expression "./*[@name='xyz'] = 3"
See the developer's guide for further information on
this error (INFINITE_LOOP_DETECTED).
```

#### 5.1.4.4. Data attributes

In addition to attributes of a node schema-nodes can have so called *data-attributes*. Data attributes are attributes that are inherited by data-nodes that are bound to the schema-node.

Automatic data attributes are therefore never evaluated in the context of the schema-node but only in the context of the data-node (essential for relative path expression).

Data attributes are defined like normal attributes but with the XML-tag a:da.

Example:

```
<v:ctr name="myCtrDef">
  <a:da name="ENABLE" type="XPath" expr="do_enable = 'TRUE'" />
</a:da>
</v:ctr>

...
<d:ctr name="myCtrData">
  <a:a name="DEF" value="XPath:/.../myCtrDef"/>

  <d:var name="do_enable" value="FALSE"/>
</d:ctr>
```

In the example above the ENABLE data attribute is evaluated in the context of myCtrData and therefore evaluates 'false' as the variable do\_enable is set to false.



### 5.1.4.5. Common attributes

All nodes can be equipped with the following attributes:

Attribute	Type	Description
IMPORTER_INFO	string array	Used by importers to add information from where a node was imported.

## 5.1.5. Generic nodes

Generic nodes are neither data nor schema-nodes.

Feature	Description
Namespace	<a href="http://www.tresos.de/_projects/DataModel2/16/root.xsd">http://www.tresos.de/_projects/DataModel2/16/root.xsd</a>
Default Prefix	None

The DataModel defines a single generic node type:

### 5.1.5.1. link

Link nodes are proxies for other nodes that are referred to by the link node. Link nodes can be used to define schema-nodes that, for example, contain themselves:

```
<v:ctr name="myContainer">
  <link name="subContainer" value="XPath:..." />
</v:ctr>
```

#### 5.1.5.1.1. Attributes

None

#### 5.1.5.1.2. Valid children

None



## 5.1.6. Schema-nodes

Schema-nodes define available configuration parameters. Schema-nodes form trees that are referred to (bound) by data-node trees that conform to the schema. Schema-nodes can have data attributes in addition to normal attributes. Schema-nodes always have a unique name concerning their parent. Schema-nodes never have a value.

Feature	Description
Namespace	<a href="http://www.tresos.de/_projects/DataModel2/06/schema.xsd">http://www.tresos.de/_projects/DataModel2/06/schema.xsd</a>
Default Prefix	v

Data attributes that can be used for all schema-nodes:

Attribute	Type	Description
ENABLE	Boolean	If set to false the data-node is treated as not being part of the tree.
INVALID	string array	This must be an automatic attribute that evolves a list of error messages. If so the data-node is considered as being erroneous.
WARNING	string array	This must be an automatic attribute that evolves a list of warning messages. If so the data-node is considered as being possibly incorrect.
DESC	string	Description of the node (e.g. for tool tips).
ORIGIN	string	Description of where the parameter originates (vendor, module name, version, target).

### 5.1.6.1. ctr

The v:ctr define a container schema-node. A container groups (contains) a fixed list of other nodes all identified by a unique name.

#### 5.1.6.1.1. Attributes and data attributes

Schema attribute	Type	Description
NAME_PATTERN	String	The optional attribute NAME_PATTERN affects the naming of a newly created instance of the container. If this attribute is not explicitly set, a default name pattern is created for any container consisting of the container's name and a unique index.



Schema attribute	Type	Description	
		<p><b>NOTE</b></p>  <p><b>Existing nodes</b></p> <p>Existing instances of containers are not checked for violations of the NAME_PATTERN. The pattern attribute is only used during the creation of an instance of a container.</p> <p>The value of the attribute must consist of a valid node name. A valid AUTOSAR node name begins with a letter and may be followed by letters, numbers and underscore (Regular expression: ^[a-zA-Z][a-zA-Z_0-9]+\$).</p> <p>The value of attribute NAME_PATTERN must additionally contain a question mark unless the container can only be created once. The question mark will be replaced with an incremental index by the system to assure the uniqueness of the name. Note, that the container cannot be created if the resulting name is not unique.</p> <p>Example: MyContainerName_-, My?Container</p>	

### 5.1.6.1.2. Valid children

- ▶ v:ctr
- ▶ v:chc
- ▶ v:lst
- ▶ v:var
- ▶ v:ref

### 5.1.6.2. chc

A v:chc defines a choice schema-node. A choice schema node contains container schema-nodes. In the configuration only one container data-node can be added to a choice that must conform to one of the containers of the choice schema-node. A choice is therefore an element with which the user can select different alternatives for the element representation.

#### 5.1.6.2.1. Attributes and data attributes

None



### 5.1.6.2.2. Valid children

- ▶ v:ctr

### 5.1.6.3. Ist

A `v:lst` creates a list schema-node. A list in the schema has only one child node which can be replicated in the data multiple times. List schema-nodes can be used to define tables.

Lists come in two variations: lists and maps which are selected via the `TYPE`-attribute of the list schema-node:

""

The empty type creates a list node. A list just contains an ordered list of entries. This type of list always contains one column besides the index column.

"MAP"

A list of the type MAP creates a map. A map is an ordered list where the names of all entries are unique within the list. A map can contain several columns where the visibility of the default columns can be configured.

Available Data attributes in addition to the ones that can be used for all schema-nodes:

#### 5.1.6.3.1. Attributes and data attributes

Data Attribute	Type	Description
MIN	long	Defines the minimum number of nodes in the (data-node) list. It is an error if the list contains fewer entries. When the (data-node) list is created it has a MIN number of entries.
MAX	long	Defines the maximum number of nodes in the (data-node) list. It is an error if the list contains more entries.

#### 5.1.6.3.2. Valid children

For list with type="":

- ▶ v:var
- ▶ v:ref

For list with type="MAP":

- ▶ v:ctr
- ▶ v:chc



### 5.1.6.4. var

A `v:var` creates a variable schema-node. A variable schema node defines a configuration parameter that can store a single value.

The type of the value is determined by the `type` XML-attribute of the `v:var`-tag. The following types are supported:

**INTEGER:**

Can store a 64 bit signed integer. Integer values are encoded in decimal format.

**BOOLEAN:**

Stores a Boolean value. The encoding of true and false is determined by the `RANGE` attribute. The first value of the `RANGE`-attribute defines the true value, the second defines the false value. True defaults to the string "TRUE", false defaults to "FALSE".

**FLOAT:**

Stores a 64 bit signed float value. Float values are encoded in decimal format ([ "—" ] <digits> ". " <digits>).

**STRING:**

Stores an arbitrary ASCII string.

**MULTILINE-STRING:**

Stores an arbitrary ASCII string that exceeds one line, e.g. in a multiple-line text box.

#### 5.1.6.4.1. Attributes and data attributes

Attribute name	Type	Description
WITH-AUTO	integer	If set to <code>true</code> , the user can choose between manually entering the value or letting the generator calculate it during code-generation. If set to <code>false</code> , the user has to edit this parameter manually.

Data attribute	Type	Description
DEFAULT	string	Defines the default value of the data-node. This may also be an automatic attribute.
RANGE	string array	<p>Restricts valid value ranges. The interpretation of the range depends on the type of the schema-node.</p> <p>For the types <code>INT</code> and <code>FLOAT</code>, the <code>RANGE</code> can restrict the values with the following expressions:</p> <pre>[ '—' ] number '—' [ '—' ] number</pre>



Data attribute	Type	Description
		<pre>'&lt;' [ "-" ] number '&lt;=' [ '-' ] number '&gt;' [ '-' ] number '&gt;=' [ '-' ] number</pre> <p>Type <b>STRING</b> variables can be restricted by either a list of values or by a regular expression. A regular expression must be prefixed with ~ (tilde).</p> <p>Type <b>BOOLEAN</b> variables define true and false values with their range attribute. The first value of the range attribute is the true value whereas the second is the false value.</p>

#### 5.1.6.4.2. Valid children

none

#### 5.1.6.5. ref

A v:ref creates a reference schema-node. A reference schema node defines a configuration parameter that refers to another configuration parameter.

##### 5.1.6.5.1. Attributes and data attributes

Data attribute	Type	Description
REF	string array	A list of paths that point to schema-nodes. Data-nodes that refer to these schema-nodes can be selected.
RANGE	string array	A list of paths that point to schema-nodes. Data-nodes that refer to these schema-nodes can be selected. This is usually an automatic attribute which calculates the nodes. For examples see <a href="#">Section 5.1.4.3.8, “XDM attribute examples”</a> .

If none of these attributes is given, all nodes can be selected. If both attributes are given, the intersection of the returned values is used to calculate the nodes which can be selected.

##### 5.1.6.5.2. Valid children

none



## 5.1.7. Data-nodes

Data-nodes store configuration data. Data-nodes form trees that refer to (or bind) schema-nodes which describe their data ranges etc.

Feature	Description
Namespace	<a href="http://www.tresos.de/_projects/DataModel2/06/data.xsd">http://www.tresos.de/_projects/DataModel2/06/data.xsd</a>
Default Prefix	d

Attributes that can be used for all data-nodes:

Attribute	Type	Description
DEF	string	Node reference to the schema-node that describes the data-node.

### 5.1.7.1. ctr

The `d:ctr` defines a container data-node. A container groups (contains) a fixed number of other nodes all identified by a unique name.

#### 5.1.7.1.1. Attributes

none

#### 5.1.7.1.2. Valid Children

- ▶ `d:ctr`
- ▶ `d:chc`
- ▶ `d:lst`
- ▶ `d:var`
- ▶ `d:ref`

### 5.1.7.2. chc

A `d:chc` defines a choice data-node. A choice data node contains one container node. The container data-node must conform to one of the container schema-nodes of the choice schema node to which the choice data-node is bound.



#### 5.1.7.2.1. Attributes

none

#### 5.1.7.2.2. Valid Children

- ▶ d:ctr

### 5.1.7.3. lst

A `d:lst` creates a list data-node. A list in the data has children of a single type (container, choice, variable, reference). The children of a list data-node must conform to the child of the bound schema node.

Lists come in two variations: lists and maps which are selected via the `TYPE`-attribute of the list schema-node:

""

The empty type creates a list node. A list contains an ordered list of entries.

"MAP"

A list of the type MAP creates a map. A map is an ordered list where the names of all entries are unique within the list.

#### 5.1.7.3.1. Attributes

none

#### 5.1.7.3.2. Valid Children

- ▶ d:ctr
- ▶ d:chc
- ▶ d:var
- ▶ d:ref

### 5.1.7.4. var

A `d:var` creates a variable data-node. A variable data node stores a single configuration value.



#### 5.1.7.4.1. Attributes

none

#### 5.1.7.4.2. Valid Children

none

### 5.1.7.5. ref

A `d:ref` creates a reference data-node. A reference data node is a configuration parameter that refers to another configuration parameter. The reference to that node is stored in the value of the data-node as a node path expression.

#### 5.1.7.5.1. Attributes

none

#### 5.1.7.5.2. Valid Children

none

## 5.1.8. Factories

The DataModel uses factories to create nodes. Each factory has knowledge of different node types and may tailor created nodes (add specific attributes etc.). Additionally, each factory may define new namespaces for nodes and attributes defined by this factory.

The following factories are supported:

`default`:

A factory that creates default node instances for all node types.

`autosar`:

This factory must be used to create nodes for the AUTOSAR ECU Configuration, ECU Parameter Definition and the other AUTOSAR formats.

The factory of a node can be set with the XML-Attribute `factory` which can be applied to all node tags. When a node has a factory-attribute, the named factory is used to create the node and all subsequent nodes that do not have such a tag, i.e. if the same factory shall be used for all nodes, it only has to be specified for the root- node.



If a factory defines attributes or nodes with a special namespace, the namespace(s) must be defined at the node which specifies the factory, e.g.:

```
<d:ctr type="AUTOSAR" factory="autosar"
  xmlns:ad="http://www.tresos.de/_projects/DataModel2/08/admindata.xsd"
  xmlns:icc="http://www.tresos.de/_projects/DataModel2/08/implconfigclass.xsd">
```

## 5.1.9. Path addressing

Reference data-nodes, the `DEF` attribute of data-nodes and the `REF` data attribute of reference schema-nodes refer to other nodes in their string values. This is be done by a path expression of the form:

`path-type ':' path`

The `DEF`-attribute of a data-node therefore might look like the following with XPath addressing:

```
<d:ctr name="myCtr">
  <a:a name="DEF" value="XPath:/AUTOSAR/
    TOP-LEVEL-PACKAGES/myPkg/ELEMENTS/myModuleDef/myCtr"/>
</d:ctr>
```

or with AUTOSAR addressing:

```
<d:ctr name="myCtr">
  <a:a name="DEF" value="ASPath:/myPkg/myModuleDef/myCtr"/>
</d:ctr>
```

The following addressing schemas are supported:

**XPath:**

References a node via an XPath path (not a full XPath expression).

**XPathDataOfSchema:**

References all data-nodes that are bound to the schema node to which the XPath expression points.

**SchemaViaParentByIdx:**

Takes the schema-node of the parent node and returns the schema-child of the node that has the value of the expression as it's index. If expression is empty a value of 0 is used.



SchemaViaParent:

Takes the schema-node of the parent node and returns the schema-child of the node that has the value of the expression as it's name. If expression is empty the node-name is used. If the parent is no list node, the first schema-parent-child is used.

ASPath:

Refers to a node by an AUTOSAR path expression. This path expression is only available for nodes that were created by the AUTOSAR factory.

ASPathDataOfSchema:

Refers to all data-nodes that are bound to the schema node to which the AUTOSAR path expression points. This path expression is only available for nodes that were created by the AUTOSAR factory.

ASPathParentNode:

References the parent node of a node that is referred to by an AUTOSAR path. This path expression is only available for nodes that were created by the AUTOSAR factory.

ASPathDataOfSchemaChoice:

Takes the schema-node of the parent node and returns the schema-child of the node that has the value of the expression as it's name. If expression is empty the node-name is used. If the parent is no list node, the first schema-parent-child is used. This path expression is only available for nodes that were created by the AUTOSAR factory.

## 5.2. Concepts: Mapping AUTOSAR ECU Configuration to XDM

### 5.2.1. Node mapping

The ECU Parameter Definition will be represented as a schema-tree within the DataModel and the ECU Configuration Description as a data-tree. All AUTOSAR-nodes will be mapped to the nodes of the DataModel:

#### 5.2.1.1. Schema-Nodes

##### 5.2.1.1.1. AR-PACKAGE

There is no corresponding data node.

simple

**simple**

```
<d:ctr type="AR-PACKAGE"/>
```

**5.2.1.1.2. MODULE-DEF**

Corresponding data node: [Section 5.2.1.14.2, “MODULE-CONFIGURATION”](#).

**simple**

```
<v:ctr name="myModule" type="MODULE-DEF"/>
```

**5.2.1.1.3. PARAM-CONF-CONTAINER-DEF**

Corresponding data node: [Section 5.2.1.14.3, “CONTAINER”](#).

<b>simple</b>	<b>with surrounding list</b>
<pre>&lt;v:ctr name="myCtr"        type="IDENTIFIABLE"/&gt;</pre>	<pre>&lt;v:lst name="myCtr" type="MAP"&gt;   &lt;v:ctr name="myCtr"          type="IDENTIFIABLE"/&gt; &lt;/v:lst&gt;</pre>

**5.2.1.1.4. MULTIPLE-CONFIGURATION-CONTAINER**

The tag MULTIPLE-CONFIGURATION-CONTAINER may only be contained within a PARAM-CONF-CONTAINER-DEF and marks this container as multiple configuration set. This information cannot be seen in the data-tree. Within the xdm file, there is always a list within the schema and data-tree! Corresponding data node: [Section 5.2.1.14.3, “CONTAINER”](#).

**with surrounding list**

```
<v:lst name="DioConfiguration" type="MULTIPLE-CONFIGURATION-CONTAINER">
  <a:da name="MIN" value="1"/>
  <v:ctr name="DioConfiguration" type="MULTIPLE-CONFIGURATION-CONTAINER">
    ...
  </v:ctr>
</v:lst>
```



### 5.2.1.1.5. BOOLEAN-PARAM-DEF

Corresponding data node: [Section 5.2.1.14.5, “BOOLEAN-VALUE”](#).

simple	with surrounding list
<pre>&lt;v:var name="bit" type="BOOLEAN"/&gt;</pre>	<pre>&lt;v:lst name="bit"&gt;     &lt;v:var name="bit" type="BOOLEAN"/&gt; &lt;/v:lst&gt;</pre>

### 5.2.1.1.6. INTEGER-PARAM-DEF

Corresponding data node: [Section 5.2.1.1.7, “FLOAT-PARAM-DEF”](#).

simple	with surrounding list
<pre>&lt;v:var name="myInt" type="INTEGER"/&gt;</pre>	<pre>&lt;v:lst name="myInt"&gt;     &lt;v:var name="myInt" type="INTEGER"/&gt; &lt;/v:lst&gt;</pre>

### 5.2.1.1.7. FLOAT-PARAM-DEF

Corresponding data node: [Section 5.2.1.14.7, “FLOAT-VALUE”](#).

simple	with surrounding list
<pre>&lt;v:var name="myFloat" type="FLOAT"/&gt;</pre>	<pre>&lt;v:lst name="value"&gt;     &lt;v:var name="value" type="FLOAT"/&gt; &lt;/v:lst&gt;</pre>

### 5.2.1.1.8. STRING-PARAM-DEF

Corresponding data node: [Section 5.2.1.14.8, “STRING-VALUE”](#).

simple	with surrounding list
<pre>&lt;v:var name="myStrg" type="STRING"/&gt;</pre>	<pre>&lt;v:lst name="myStrg"&gt;     &lt;v:var name="myStrg" type="STRING"/&gt;</pre>



simple	with surrounding list
	</v:lst>

### 5.2.1.1.9. LINKER-SYMBOL-DEF

Corresponding data node: [Section 5.2.1.14.9, “LINKER-SYMBOL-VALUE”](#).

simple	with surrounding list
<v:var name="LinkerSymbol" type="LINKER-SYMBOL"/>	<v:lst name="LinkerSymbol"> <v:var name="LinkerSymbol" type="LINKER-SYMBOL"/> </v:lst>

### 5.2.1.1.10. FUNCTION-NAME-DEF

Corresponding data node: [Section 5.2.1.14.10, “FUNCTION-NAME-VALUE”](#).

simple	with surrounding list
<v:var name="myFunction" type="FUNCTION-NAME"/>	<v:lst name="myFunction"> <v:var name="myFunction" type="FUNCTION-NAME"/> </v:lst>

### 5.2.1.1.11. ENUMERATION-PARAM-DEF

Corresponding data node: [Section 5.2.1.14.11, “ENUMERATION-VALUE”](#).

simple	with surrounding list
<v:var name="enum" type="ENUMERATION"> <a:da name="RANGE"> <a:v>Enum1</a:v> <a:v>Enum2</a:v> </a:da> </v:var>	<v:lst name="enum"> <v:var name="enum" type="ENUMERATION"> <a:da name="RANGE"> <a:v>Enum1</a:v> <a:v>Enum2</a:v> </a:da> </v:var> </v:lst>



### 5.2.1.2. CHOICE-CONTAINER-DEF

Up to AUTOSAR 2.1, there is no corresponding TAG for a data node in AUTOSAR. Nevertheless there is a data node in tresos: [Section 5.2.1.14.4, “Choices”](#). Since AUTOSAR 3.0, there is a corresponding Data-Node-tag which has the same tag as containers (CONTAINER).

simple	with surrounding list
<pre>Autosar 2.x: &lt;v:chc name="myChc"&gt;   &lt;v:ctr name="myCtr1"/&gt;   &lt;v:ctr name="myCtr2"/&gt; &lt;/v:chc&gt;</pre>	<pre>&lt;v:lst name="myChc" type="MAP"&gt;   &lt;v:chc name="myChc"&gt;     &lt;v:ctr name="myCtr1"/&gt;     &lt;v:ctr name="myCtr2"/&gt;   &lt;/v:chc&gt; &lt;/v:lst&gt;</pre>
<pre>Autosar 3.x: &lt;v:chc name="myChc"        type="IDENTIFIABLE"&gt;   &lt;v:ctr name="myCtr1"/&gt;   &lt;v:ctr name="myCtr2"/&gt; &lt;/v:chc&gt;</pre>	<pre>&lt;v:lst name="myChc" type="MAP"&gt;   &lt;v:chc name="myChc"        type="IDENTIFIABLE"&gt;     &lt;v:ctr name="myCtr1"/&gt;     &lt;v:ctr name="myCtr2"/&gt;   &lt;/v:chc&gt; &lt;/v:lst&gt;</pre>

### 5.2.1.3. CHOICE-REFERENCE-DEF

Corresponding data node: [Section 5.2.1.14.12, “REFERENCE-VALUE”](#).

simple	with surrounding list
<pre>&lt;v:ref name="myChrRef"        type="CHOICE-REFERENCE"&gt;   &lt;a:da name="REF"         value="ASPathDataOfSchema:/..."/&gt;   &lt;a:da name="REF-TYPE"         value="ASPath"/&gt; &lt;/v:ref&gt;</pre>	<pre>&lt;v:lst name="myChrRef"&gt;   &lt;v:ref name="myChrRef"        type="CHOICE-REFERENCE"&gt;     &lt;a:da name="REF"       value="ASPathDataOfSchema:/..."/&gt;     &lt;a:da name="REF-TYPE"       value="ASPath"/&gt;   &lt;/v:ref&gt; &lt;/v:lst&gt;</pre>



#### 5.2.1.4. REFERENCE-DEF

Corresponding data node: [Section 5.2.1.14.12, “REFERENCE-VALUE”](#).

simple	with surrounding list
<pre>&lt;v:ref name="myRef" type="REFERENCE"&gt;   &lt;a:da name="REF"     value="ASPathDataOfSchema:/..."&gt;   &lt;a:da name="REF-TYPE"     value="ASPath"/&gt; &lt;/v:ref&gt;</pre>	<pre>&lt;v:lst name="myRef"&gt;   &lt;v:ref name="myRef" type="REFERENCE"&gt;     &lt;a:da name="REF"       value="ASPathDataOfSchema:/..."&gt;     &lt;a:da name="REF-TYPE"       value="ASPath"/&gt;   &lt;/v:ref&gt; &lt;/v:lst&gt;</pre>

#### 5.2.1.5. SYMBOLIC-NAME-REFERENCE-DEF

Corresponding data node: [Section 5.2.1.14.12, “REFERENCE-VALUE”](#).

simple	with surrounding list
<pre>&lt;v:ref name="mySrd"   type="SYMBOLIC-NAME-REFERENCE"&gt;   &lt;a:da name="REF"     value="ASPathDataOfSchema:/..."&gt;   &lt;a:da name="REF-TYPE"     value="ASPath"/&gt; &lt;/v:ref&gt;</pre>	<pre>&lt;v:lst name="mySrc"&gt;   &lt;v:ref name="mySrd"     type="SYMBOLIC-NAME-REFERENCE"&gt;     &lt;a:da name="REF"       value="ASPathDataOfSchema:/..."&gt;     &lt;a:da name="REF-TYPE"       value="ASPath"/&gt;   &lt;/v:ref&gt; &lt;/v:lst&gt;</pre>

#### 5.2.1.6. INSTANCE-REFERENCE-DEF

Corresponding data node: [Section 5.2.1.14.13, “INSTANCE-REFERENCE-VALUE”](#).

simple	with surrounding list
<pre>&lt;v:ctr name="myInst" type="INSTANCE"&gt;   &lt;v:ref name="TARGET"     type="REFERENCE"&gt;     &lt;a:da name="REF"       value="ASTyped:RunnableEntity"/&gt;</pre>	<pre>&lt;v:lst name="myInst"&gt;   &lt;v:ctr name="myInst" type="INSTANCE"&gt;     &lt;v:ref name="TARGET"       type="REFERENCE"&gt;       &lt;a:da name="REF"</pre>



simple	with surrounding list
<pre> &lt;/v:ref&gt; &lt;v:lst name="CONTEXT"&gt;   &lt;v:ref name="CONTEXT"     type="REFERENCE"&gt;     &lt;a:da name="RANGE"       type="IRefCtxt"&gt;       &lt;a:tst expr="ComponentPrototype"/&gt;       &lt;a:tst expr="PortPrototype*" /&gt;     &lt;/a:da&gt;   &lt;/v:ref&gt; &lt;/v:lst&gt; &lt;/v:ctr&gt; </pre>	<pre>   value="ASTyped:RunnableEntity"/&gt; &lt;/v:ref&gt; &lt;v:lst name="CONTEXT"&gt;   &lt;v:ref name="CONTEXT"     type="REFERENCE"&gt;     &lt;a:da name="RANGE"       type="IRefCtxt"&gt;       &lt;a:tst expr="ComponentPrototype"/&gt;       &lt;a:tst expr="PortPrototype*" /&gt;     &lt;/a:da&gt;   &lt;/v:ref&gt; &lt;/v:lst&gt; &lt;/v:ctr&gt; &lt;/v:lst&gt; </pre>

### 5.2.1.7. FOREIGN-REFERENCE-DEF

Corresponding data node: [Section 5.2.1.14.12, “REFERENCE-VALUE”](#).

simple	with surrounding list
<pre> &lt;v:ref name="myRef"   type="FOREIGN-REFERENCE"&gt;   &lt;a:da name="REF"   value="ASPathDataOfSchemaChoice:/..."/&gt;   &lt;a:da name="REF-TYPE"   value="ASPath"/&gt; &lt;/v:ref&gt; </pre>	<pre> &lt;v:lst name="myRef"&gt;   &lt;v:ref name="myRef"     type="FOREIGN-REFERENCE"&gt;     &lt;a:da name="REF"     value="ASPathDataOfSchemaChoice:/..."/&gt;     &lt;a:da name="REF-TYPE"       value="ASPath"/&gt;   &lt;/v:ref&gt; &lt;/v:lst&gt; </pre>

### 5.2.1.8. ECUC-URI-REFERENCE-DEF

Corresponding data node: [Section 5.2.1.14.12, “REFERENCE-VALUE”](#).

Please have a look at chapter [Section 5.2.7.1, “URI References”](#) for the concepts behind URI references.

simple	with surrounding list
<pre> &lt;v:ref name="UriRef"   type="URI-REFERENCE"&gt;   &lt;a:da name="REF" </pre>	<pre> &lt;v:lst name="UriRef"&gt;   &lt;v:ref name="UriRef"     type="URI-REFERENCE"&gt; </pre>



simple	with surrounding list
<pre>value="ASPath:/...."/&gt; &lt;/v:ref&gt;</pre>	<pre>&lt;a:da name="REF"       value="ASPath:/...."/&gt; &lt;/v:ref&gt; &lt;/v:lst&gt;</pre>

### 5.2.1.9. DERIVED-BOOLEAN-PARAM-DEF

Corresponding data node: [Section 5.2.1.14.5, “BOOLEAN-VALUE”](#).

simple	with surrounding list
<pre>&lt;v:var name="myBool" type="BOOLEAN"&gt;   &lt;a:a name="DERIVED" value="true"/&gt; &lt;/v:var&gt;</pre>	<pre>&lt;v:lst name="myBool"&gt;   &lt;v:var name="myBool" type="BOOLEAN"&gt;     &lt;a:a name="DERIVED" value="true"/&gt;   &lt;/v:var&gt; &lt;/v:lst&gt;</pre>

### 5.2.1.10. DERIVED-INTEGER-PARAM-DEF

Corresponding data node: [Section 5.2.1.14.6, “INTEGER-VALUE”](#).

simple	with surrounding list
<pre>&lt;v:var name="myInt" type="INTEGER"&gt;   &lt;a:a name="DERIVED" value="true"/&gt; &lt;/v:var&gt;</pre>	<pre>&lt;v:lst name="myInt"&gt;   &lt;v:var name="myInt" type="INTEGER"&gt;     &lt;a:a name="DERIVED" value="true"/&gt;   &lt;/v:var&gt; &lt;/v:lst&gt;</pre>

### 5.2.1.11. DERIVED-FLOAT-PARAM-DEF

Corresponding data node: [Section 5.2.1.14.7, “FLOAT-VALUE”](#).

simple	with surrounding list
<pre>&lt;v:var name="myFloat" type="FLOAT"&gt;   &lt;a:a name="DERIVED" value="true"/&gt; &lt;/v:var&gt;</pre>	<pre>&lt;v:lst name="myFloat"&gt;   &lt;v:var name="myFloat" type="FLOAT"&gt;     &lt;a:a name="DERIVED" value="true"/&gt;   &lt;/v:var&gt;</pre>



simple	with surrounding list
	</v:lst>

### 5.2.1.12. DERIVED-STRING-PARAM-DEF

Corresponding data node: [Section 5.2.1.14.8, “STRING-VALUE”](#).

simple	with surrounding list
<pre>&lt;v:var name="myStrg" type="STRING"&gt;   &lt;a:a name="DERIVED" value="true"/&gt; &lt;/v:var&gt;</pre>	<pre>&lt;v:lst name="myStrg"&gt;   &lt;v:var name="myStrg" type="STRING"&gt;     &lt;a:a name="DERIVED" value="true"/&gt;   &lt;/v:var&gt; &lt;/v:lst&gt;</pre>

### 5.2.1.13. DERIVED ENUMERATION-PARAM-DEF

Corresponding data node: [Section 5.2.1.14.11, “ENUMERATION-VALUE”](#).

simple	with surrounding list
<pre>&lt;v:var name="enum" type="ENUMERATION"&gt;   &lt;a:a name="DERIVED" value="true"/&gt; &lt;/v:var&gt;</pre>	<pre>&lt;v:lst name="enum"&gt;   &lt;v:var name="enum"     type="ENUMERATION"&gt;     &lt;a:a name="DERIVED" value="true"/&gt;   &lt;/v:var&gt; &lt;/v:lst&gt;</pre>

## 5.2.1.14. Data-Nodes

### 5.2.1.14.1. ECU-CONFIGURATION

There is no corresponding schema node.

simple
<pre>&lt;d:ctr name="myCfg" type="ECU-CONFIGURATION"&gt;   ... &lt;/d:ctr&gt;</pre>



## simple

### 5.2.1.14.2. MODULE-CONFIGURATION

Corresponding schema node: [Section 5.2.1.1.2, “MODULE-DEF”](#).

## simple

```
<d:ctr name="myMod" type="MODULE-CONFIGURATION">
  ...
</d:ctr>
```

### 5.2.1.14.3. CONTAINER

Corresponding schema node: [Section 5.2.1.1.3, “PARAM-CONF-CONTAINER-DEF”](#), [Section 5.2.1.1.4, “MULTIPLE-CONFIGURATION-CONTAINER”](#).

simple	with surrounding list
<pre>&lt;d:ctr name="myCtr"       type="IDENTIFIABLE"&gt;   ... &lt;/d:ctr&gt;</pre>	<pre>&lt;d:lst name="myCtr" type="MAP"&gt;   &lt;d:ctr name="myCtrl1"         type="IDENTIFIABLE1"&gt;     ...   &lt;/d:ctr&gt;   &lt;d:ctr name="myCtrl2"         type="IDENTIFIABLE2"&gt;     ...   &lt;/d:ctr&gt; &lt;/d:lst&gt;</pre>

### 5.2.1.14.4. Choices

AUTOSAR 2.0 and 2.1 defines no XML-tag for Choices within the data. They are simply omitted. Since AUTOSAR 3.0, a choice is represented in data as CONTAINER. Nonetheless, there always is a choice in the data-tree in xdm file format. Corresponding schema node: [Section 5.2.1.2, “CHOICE-CONTAINER-DEF”](#).

Simple	With surrounding list
Autosar 2.x:	<pre>&lt;d:lst name="MyChoice" type="MAP"&gt;</pre>



Simple	With surrounding list
<pre>&lt;d:chc name="MyChoice"       type="CHOICE"       value="Ctrl1"&gt; &lt;d:ctr name="Ctrl1"       type="IDENTIFIABLE"/&gt; &lt;d:ctr name="Ctrl2"       type="IDENTIFIABLE"/&gt; &lt;d:ctr name="Ctrl3"       type="IDENTIFIABLE"/&gt; &lt;/d:chc&gt;</pre>	<pre>&lt;d:chc name="MyChoice"       type="CHOICE"       value="Ctrl1"&gt; &lt;d:ctr name="Ctrl1"       type="IDENTIFIABLE"/&gt; &lt;d:ctr name="Ctrl2"       type="IDENTIFIABLE"/&gt; &lt;d:ctr name="Ctrl3"       type="IDENTIFIABLE"/&gt; &lt;/d:chc&gt; &lt;/d:lst&gt;</pre>
<p>Autosar 3.x:</p> <pre>&lt;d:chc name="MyChoice"       type="IDENTIFIABLE"       value="Ctrl1"&gt; &lt;d:ctr name="Ctrl1"       type="IDENTIFIABLE"/&gt; &lt;d:ctr name="Ctrl2"       type="IDENTIFIABLE"/&gt; &lt;d:ctr name="Ctrl3"       type="IDENTIFIABLE"/&gt; &lt;/d:chc&gt;</pre>	<pre>&lt;d:lst name="MyChoice" type="MAP"&gt; &lt;d:chc name="MyChoice"       type="IDENTIFIABLE"       value="Ctrl1"&gt; &lt;d:ctr name="Ctrl1"       type="IDENTIFIABLE"/&gt; &lt;d:ctr name="Ctrl2"       type="IDENTIFIABLE"/&gt; &lt;d:ctr name="Ctrl3"       type="IDENTIFIABLE"/&gt; &lt;/d:chc&gt; &lt;/d:lst&gt;</pre>

### 5.2.1.14.5. BOOLEAN-VALUE

Corresponding schema node: [Section 5.2.1.1.5, “BOOLEAN-PARAM-DEF”](#), [Section 5.2.1.9, “DERIVED-BOOLEAN-PARAM-DEF”](#).

simple	with surrounding list
<pre>&lt;d:var name="myBool" value="true"       type="BOOLEAN"/&gt;</pre>	<pre>&lt;d:lst name="myBool"&gt; &lt;d:var name="myBool1" value="true"       type="BOOLEAN"/&gt; &lt;d:var name="myBool2" value="true"       type="BOOLEAN"/&gt; &lt;/d:lst&gt;</pre>



### 5.2.1.14.6. INTEGER-VALUE

Corresponding schema node: [Section 5.2.1.1.6, “INTEGER-PARAM-DEF”](#), [Section 5.2.1.10, “DERIVED-INTEGER-PARAM-DEF”](#).

simple	with surrounding list
<pre>&lt;d:var name="myInt" value="5"       type="INTEGER"/&gt;</pre>	<pre>&lt;d:lst name="myInt"&gt;   &lt;d:var name="myInt1" value="5"         type="INTEGER"/&gt;   &lt;d:var name="myInt2" value="5"         type="INTEGER"/&gt; &lt;/d:lst&gt;</pre>

### 5.2.1.14.7. FLOAT-VALUE

Corresponding schema node: [Section 5.2.1.1.7, “FLOAT-PARAM-DEF”](#), [Section 5.2.1.11, “DERIVED-FLOAT-PARAM-DEF”](#).

simple	with surrounding list
<pre>&lt;d:var name="myFloat" value="5.5"       type="FLOAT"/&gt;</pre>	<pre>&lt;d:lst name="myFloat"&gt;   &lt;d:var name="myFloat1" value="5.5"         type="FLOAT"/&gt;   &lt;d:var name="myFloat2" value="5.5"         type="FLOAT"/&gt; &lt;/d:lst&gt;</pre>

### 5.2.1.14.8. STRING-VALUE

Corresponding schema node: [Section 5.2.1.1.8, “STRING-PARAM-DEF”](#), [Section 5.2.1.12, “DERIVED-STRING-PARAM-DEF”](#).

simple	with surrounding list
<pre>&lt;d:var name="myStr" value="Hello"       type="STRING"/&gt;</pre>	<pre>&lt;d:lst name="myFloat"&gt;   &lt;d:var name="myStr1" value="Hello"         type="STRING"/&gt;   &lt;d:var name="myStr2" value="Hello"         type="STRING"/&gt; &lt;/d:lst&gt;</pre>



### 5.2.1.14.9. LINKER-SYMBOL-VALUE

Corresponding data node: [Section 5.2.1.1.9, “LINKER-SYMBOL-DEF”](#).

simple	with surrounding list
<pre>&lt;d:var name="LinkerSymbol"       type="LINKER-SYMBOL"       value="MyLinkerSymbol"/&gt;</pre>	<pre>&lt;d:lst name="LinkerSymbol"&gt;   &lt;d:var name="LinkerSymbol1"         type="LINKER-SYMBOL"         value="MyLinkerSymbol"/&gt;   &lt;d:var name="LinkerSymbol2"         type="LINKER-SYMBOL"         value="MyLinkerSymbol"/&gt; &lt;/d:lst&gt;</pre>

### 5.2.1.14.10. FUNCTION-NAME-VALUE

Corresponding schema node: [Section 5.2.1.1.10, “FUNCTION-NAME-DEF”](#).

simple	with surrounding list
<pre>&lt;d:var name="myFunc" value="func1"       type="FUNCTION-NAME"/&gt;</pre>	<pre>&lt;d:lst name="myFunc"&gt;   &lt;d:var name="myFunc1" value="func1"         type="FUNCTION-NAME"/&gt;   &lt;d:var name="myFunc2" value="func1"         type="FUNCTION-NAME"/&gt; &lt;/d:lst&gt;</pre>

### 5.2.1.14.11. ENUMERATION-VALUE

Corresponding schema node: [Section 5.2.1.1.11, “ENUMERATION-PARAM-DEF”](#), [Section 5.2.1.13, “DERIVED-ENUMERATION-PARAM-DEF”](#).

simple	with surrounding list
<pre>&lt;d:var name="myEnum" value="Enum1"       type="ENUMERATION"/&gt;</pre>	<pre>&lt;d:lst name="myEnum"&gt;   &lt;d:var name="myEnum1" value="Enum1"         type="ENUMERATION"/&gt;   &lt;d:var name="myEnum2" value="Enum1"         type="ENUMERATION"/&gt;</pre>



simple	with surrounding list
	</d:lst>

#### 5.2.1.14.12. REFERENCE-VALUE

Corresponding schema node: [Section 5.2.1.4, “REFERENCE-DEF”](#), [Section 5.2.1.5, “SYMBOLIC-NAME-REFERENCE-DEF”](#), [Section 5.2.1.7, “FOREIGN-REFERENCE-DEF”](#), [Section 5.2.1.3, “CHOICE-REFERENCE-DEF”](#), [Section 5.2.1.8, “ECUC-URI-REFERENCE-DEF”](#).

simple	with surrounding list
<pre>&lt;d:ref name="myRef"       value="ASPath:/..."       type="REFERENCE"/&gt;</pre>	<pre>&lt;d:lst name="myRef"&gt;   &lt;d:ref name="myRef1"         value="ASPath:/..."         type="REFERENCE"/&gt;   &lt;d:ref name="myRef2"         value="ASPath:/..."         type="REFERENCE"/&gt; &lt;/d:lst&gt;</pre>

#### 5.2.1.14.13. INSTANCE-REFERENCE-VALUE

Corresponding schema node: [Section 5.2.1.6, “INSTANCE-REFERENCE-DEF”](#).

simple	with surrounding list
<pre>&lt;d:ctr name="iRef" type="INSTANCE"&gt;   &lt;d:lst name="CONTEXT"&gt;     &lt;d:ref value="ASPath:/..."           type="REFERENCE"/&gt;     &lt;d:ref value="ASPath:/..."           type="REFERENCE"/&gt;   &lt;/d:lst&gt;   &lt;d:ref name="TARGET"         value="ASPath:/..."         type="REFERENCE"/&gt; &lt;/d:ctr&gt;</pre>	<pre>&lt;d:lst name="iRef"&gt;   &lt;d:ctr name="iRef1" type="INSTANCE"&gt;     &lt;d:lst name="CONTEXT"&gt;       &lt;d:ref value="ASPath:/..."           type="REFERENCE"/&gt;       &lt;d:ref value="ASPath:/..."           type="REFERENCE"/&gt;     &lt;/d:lst&gt;     &lt;d:ref name="TARGET"           value="ASPath:/..."           type="REFERENCE"/&gt;   &lt;/d:ctr&gt; &lt;/d:lst&gt;</pre>

#### 5.2.1.14.14. Example:

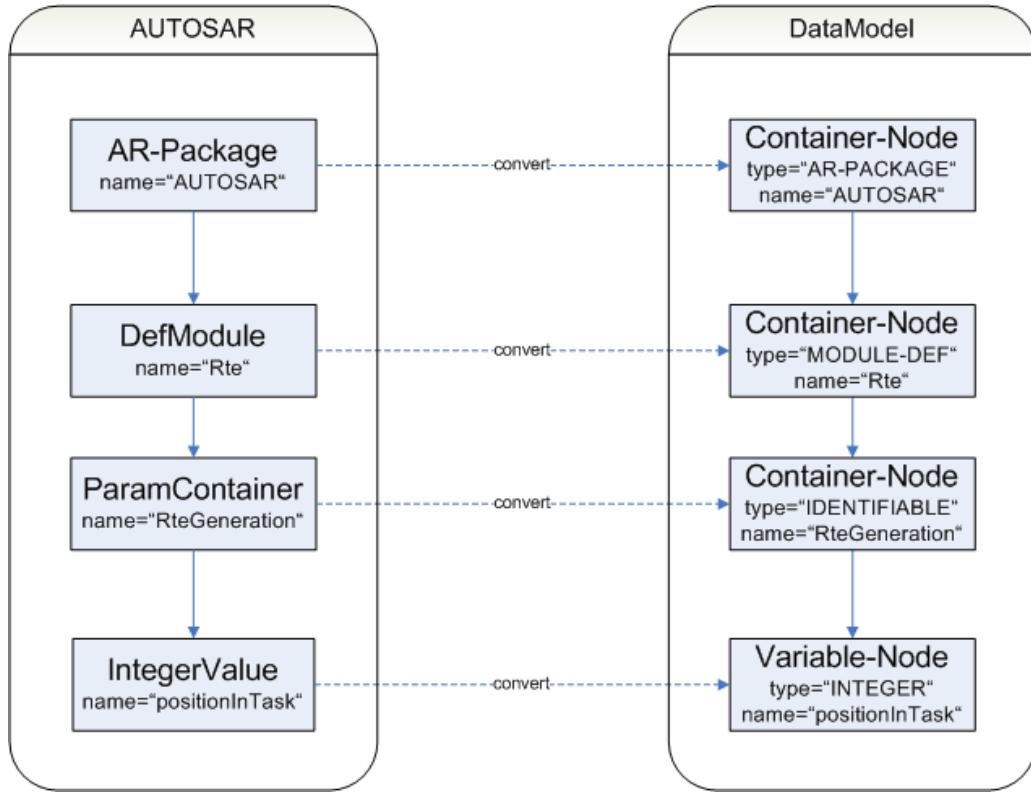


Figure 5.1. AUTOSAR-Modes to DataModel-Nodes

Every ECU Configuration Description-node refers to one ECU Parameter Definition-node. The tree-structure within the DataModel looks like the structure of the ECU-PD/CD except for the top level structure, lists and choices (see the following chapters).

## 5.2.2. Attribute Mapping

There are some complex attributes in AUTOSAR which have their own mapping.

### 5.2.2.1. CONFIGURATION-CLASS-AFFECTATION

AUTOSAR	XDM
<pre> &lt;CONFIGURATION-CLASS-AFFECTION&gt;   &lt;AFFECTED-REFS&gt;     &lt;AFFECTED-REF       DEST="BOOLEAN-PARAM-DEF"&gt;       /abc..     &lt;/AFFECTED-REF&gt;   </pre>	<pre> &lt;a:a name="AFFECTED"&gt;   &lt;a:v&gt;ASPathDataOfSchema:/abc..&lt;/a:v&gt;   &lt;a:v&gt;ASPathDataOfSchema:/xyz..&lt;/a:v&gt; &lt;/a:a&gt; &lt;a:a name="AFFECTION-KIND"   value="PCAffectsPB"/&gt;   </pre>

AUTOSAR	XDM
<pre> &lt;AFFECTED-REF   DEST="STRING-PARAM-DEF"&gt;   /xyz.. &lt;/AFFECTED-REF&gt; &lt;/AFFECTED-REFS&gt; &lt;AFFECTION-KIND&gt;   PC-AFFECTS-PB &lt;/AFFECTION-KIND&gt; &lt;/CONFIGURATION-CLASS-AFFECTION&gt; </pre>	

### 5.2.3. XPath-addressing of ECU-CD using ECU-PD

Because there can be many configurations (using the same definition) and each configuration may use its own short-names for its nodes, addressing configuration-nodes should be done using only the names of the schema-nodes. Otherwise, the addressing would only work for a single configuration.

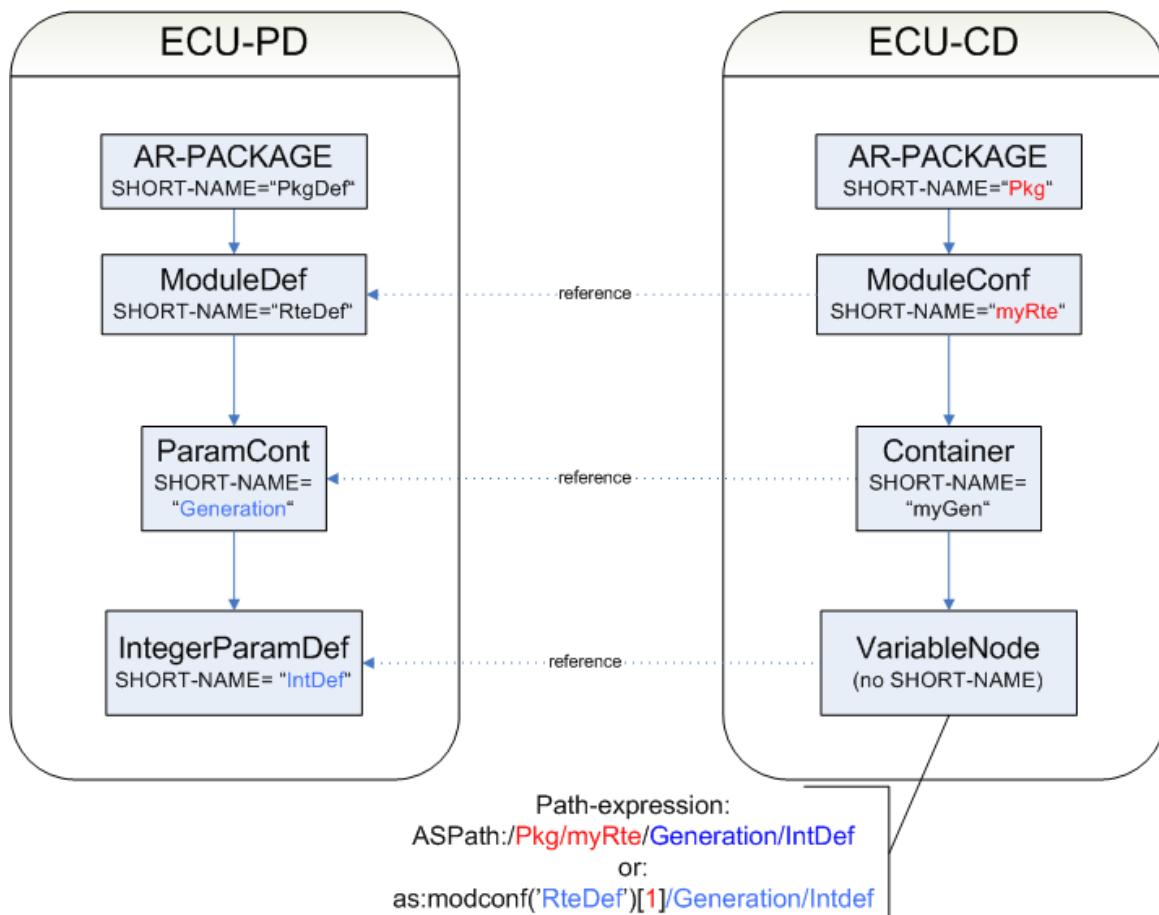


Figure 5.2. XPath-addressing within ECU-CD



Possible XPath-paths to address the ModuleConf:

- ▶ ASPPath:/Pkg/myRte/Generation/IntDef
- ▶ as:modconf('RteDef')[1]/Generation/Intdef

As you can see, the ModuleConf can be addressed using its configuration-name. However if the schema-addressing is used instead, it will work for any configuration - regardless of the ModuleConf's name. You only have to define which configuration you want to address (if you have exactly one configuration at a time, this will always be the first configuration).

Note, that the path following "as:modconf()" is an XPath - not a SHORT- NAME-path (otherwise you cannot address the VariableNode). So if the IntegerParamDef would have a lower or upper multiplicity != 1, there will be a surrounding list and the path would be .../Generation/IntDef/\*[1].

The following possibilities to address the VariableNode are only mentioned for completeness:

- ▶ as:modconf('RteDef')[name(.)='myRte']/Generation/Intdef
- ▶ as:ref( concat( as:path( as:stod('/PkgDef/RteDef')[1] ), '/Generation/IntDef' ) )
- ▶ /AUTOSAR/TOP-LEVEL-PACKAGES/\*/ELEMENTS/myRte/Generation/Intdef

### 5.2.3.1. Path in Template

The configurator provides a feature for developers showing the path to address a configuration node using the schema-names. To enable this feature, select the preference **Show template path in Properties view**. For instructions about how to change preferences, see the EB tresos Studio user's guide, chapter **Setting Developer features preferences**.

Now, the "Information"-view shows the "Path in Template" for the selected configuration-node in the current editor.

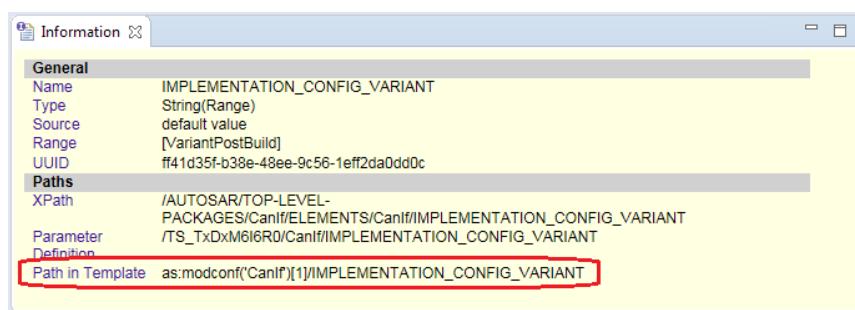


Figure 5.3. Path in template

Note, that you should NOT use this path "as is". If the path contains e.g. ".../\*[5]..." it addresses the fifth element within a list. Instead of directly accessing the fifth element, you should loop over all list-elements and use relative path within the loop.

Example:

```
[!LOOP "as:modconf('SchM') [1]/SchMMainFunctionMapping/*"!]
[!"./SchMMainFunctionRef"!]
[!ENDLOOP!]
```

## 5.2.4. Top level Structure

The top level node-structure of the datamodel for AUTOSAR-nodes is based on the AUTOSAR-metamodel as depicted below.

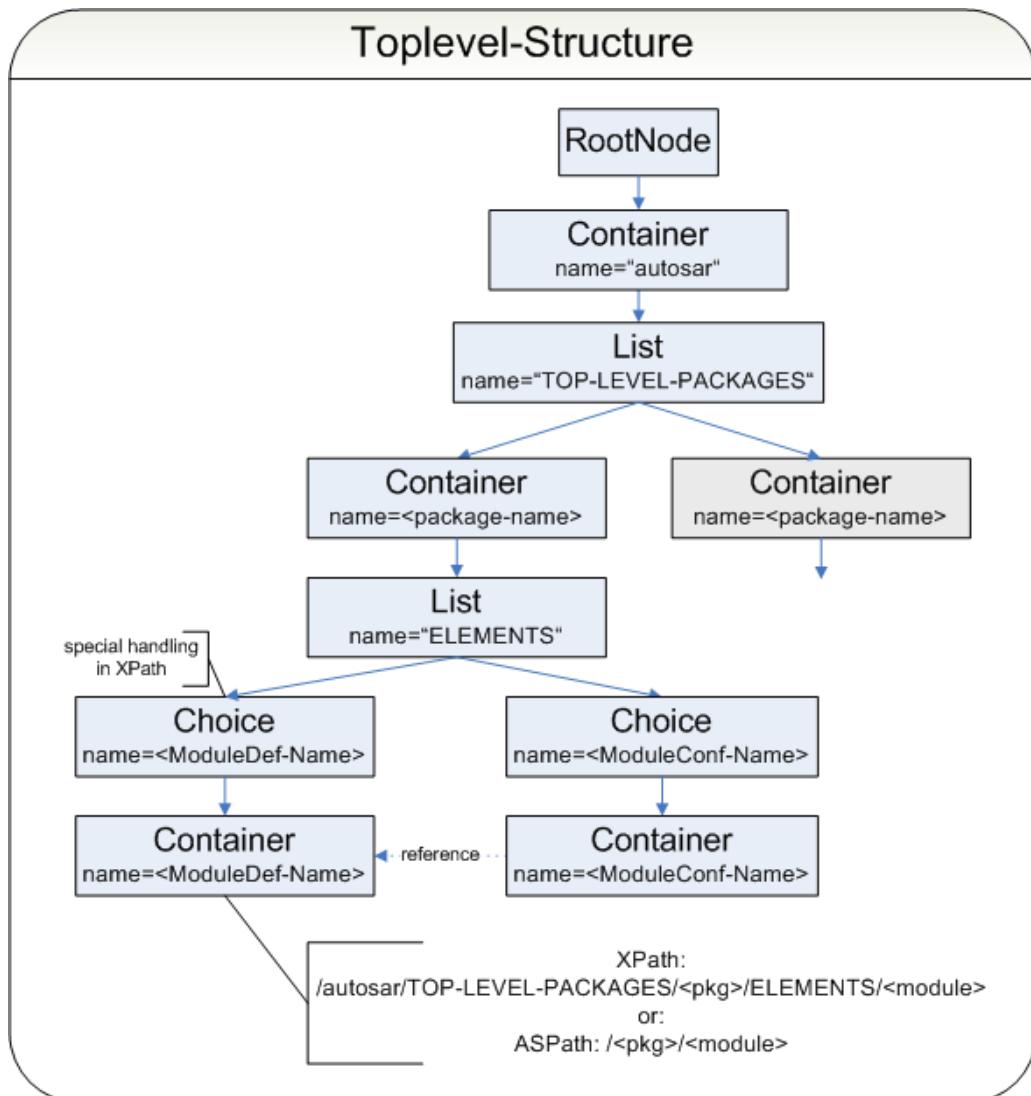


Figure 5.4. Top level structure for AUTOSAR-nodes

XPath	Description
/AUTOSAR/TOP-LEVEL-PACKAGES/<pkg>	Selects the package with the given name.
/AUTOSAR/TOP-LEVEL-PACKAGES/* ELEMEN-TS/<module>	Selects the module-container (not the choice!) with the given name, regardless if it is a definition or a configuration, and regardless of the package.
as:modconf(<module>)	Selects the module-configuration with the given name, regardless in which package it will be found.
ASPath:/<pkg>/<module>	Selects the module-container with the given name in the given package .

## 5.2.5. List-Nodes

Each node with a lower or upper multiplicity != 1 will be wrapped into a list-node (except optional nodes - see [Section 5.2.5.1, “Optional elements”](#)). The so created list gets the SHORT-NAME of the ECU Parameter Definition-node as its name. This happens both in the schema-tree as well as in the data-tree.

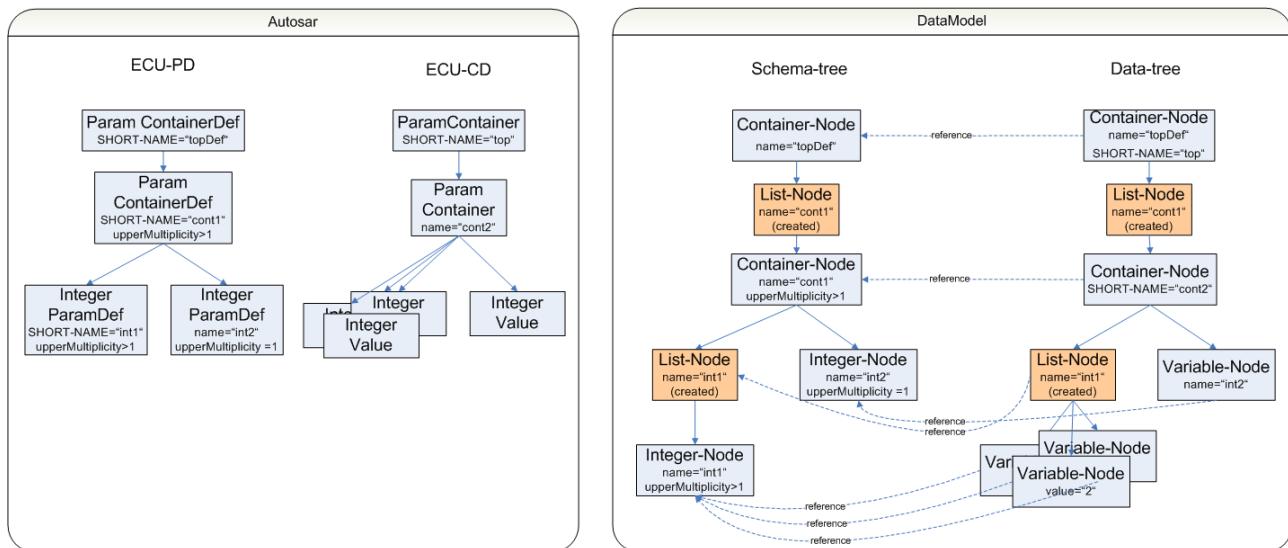


Figure 5.5. List-Representation

XPath (within the data-tree)	Description
topDef/cont1	selects the list "cont1". Note that the selection is done within the data using the schema-name.
topDef/cont1/*[1]	selects the first container within the list (here "cont1"). Note, that the selection is done within the data using the schema-name.
ASPath:top/cont2	selects the container-node "cont2"



XPath (within the data-tree)	Description
topDef/cont1/*[1]/int2	selects the variable-node "int2"
topDef/cont1/*[1]/int1	selects the list "int1"
topDef/cont1/*[1]/int1/*[2]	selects the second variable-node within the list "int1"

There is no SHORT-NAME-path to a variable-node as this does not have a SHORT-NAME.

### 5.2.5.1. Optional elements

AUTOSAR is able to define "optional" elements in the ECU Parameter Definition by setting LOWER-MULTIPLICITY == 0 and UPPER-MULTIPLICITY == 1 to an element of the ECU Parameter Definition.

```
<PARAM-CONF-CONTAINER-DEF>
  <SHORT-NAME>MyOptionalContainer</SHORT-NAME>
  <LOWER-MULTIPLICITY>0</LOWER-MULTIPLICITY>
  <UPPER-MULTIPLICITY>1</UPPER-MULTIPLICITY>
</PARAM-CONF-CONTAINER-DEF>
```

There are two different possible representations of this construct within the DataModel. The old one (which is no longer recommended) wraps the optional element within a list which can only have one child:

```
<v:lst name="MyOptionalContainer" type=MAP">
  <a:da name="MIN" value="0"/>
  <a:da name="MAX" value="1"/>
  <v:ctr name="MyOptionalContainer"/>
</v:lst>
```

This representation has two major disadvantages:

- ▶ The GUI represents each element within its own tab.
- ▶ When optional elements are taken over from the StMD to the VSMD as mandatory elements, the Xpath to this element changes. Thus, within a codegenerator- template, two different statements (one for each representation) are needed to safely access this element:

```
[ !IF "node:exists(ElementName)"] ... [!ENDIF!]
[ !IF "node:exists(ElementName/*[1])"] ... [!ENDIF!]
```

The new (and highly recommended) representation simply marks the element as optional (without wrapping it into a list):



```
<v:var name="MyOptionalInteger" type="INTEGER">
  <a:a name="OPTIONAL" value="true"/>
  <a:da name="ENABLE" value="false"/>
</v:var>
```

The xpath to an element does not depend on whether this element is optional or not:

```
[!IF "node:exists(ElementName)"] ... [!ENDIF!]
```

### 5.2.5.2. Optional elements within the commandline

To be backward-compatible to existing scripts, the default behavior of the commandline is the "old" list-representation of optional elements. To activate the new recommended representation, the option "MapOptionalAsList" must be set to false. The following example converts a file from the "old" to the new recommended representation:

```
tresos_cmd.bat -DMapOptionalAsList=false legacy convert old.xdm new.xdm
```

### 5.2.5.3. Optional elements within the GUI

When using the GUI, it depends on the used plugin if optional elements are used. To let a plugin use the optional-feature, its xdm-files must be converted as described above. After that, "old" configuration data files can still be loaded.

## 5.2.6. Choice-Nodes

Until AUTOSAR 3.0, there was no choice-construct within the ECU Configuration Description. Nonetheless, there is always a choice-data-node within the DataModel. But the Choice-Node within the data-tree cannot be addressed via XPath. Instead, the XPath "topDef/abc" (within the data-tree) returns the chosen container.

For completeness, this choice-data-node refers to the choice-schema-node. The choice-data-node gets the name of the choice-schema-node and the SHORT-NAME of its child as value (the selected container).

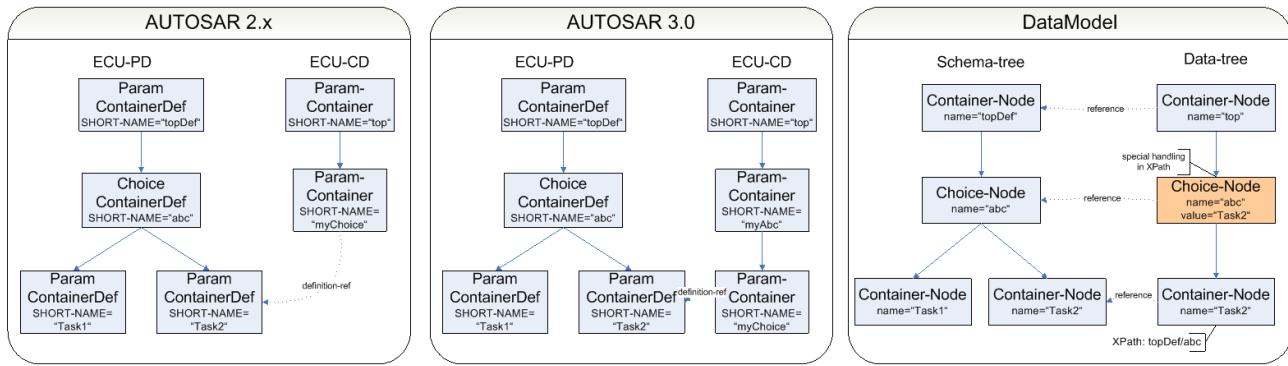


Figure 5.6. Choice-Representation

**NOTE**

As described above, choices with a lower or upper multiplicity != 1 will also be wrapped within a list.

The examples in the following table operates on the data-tree shown above.

XPath (within the data-tree)	Description
topDef/abc	selects the container-node "Task2" (the selected container)
ASPath:top/myChoice (Autosar 2.x)	selects the container-node "Task2" (the selected container)
ASPath:top/myAbc/myChoice (Autosar 3.x)	selects the container-node "Task2" (the selected container)

## 5.2.7. References

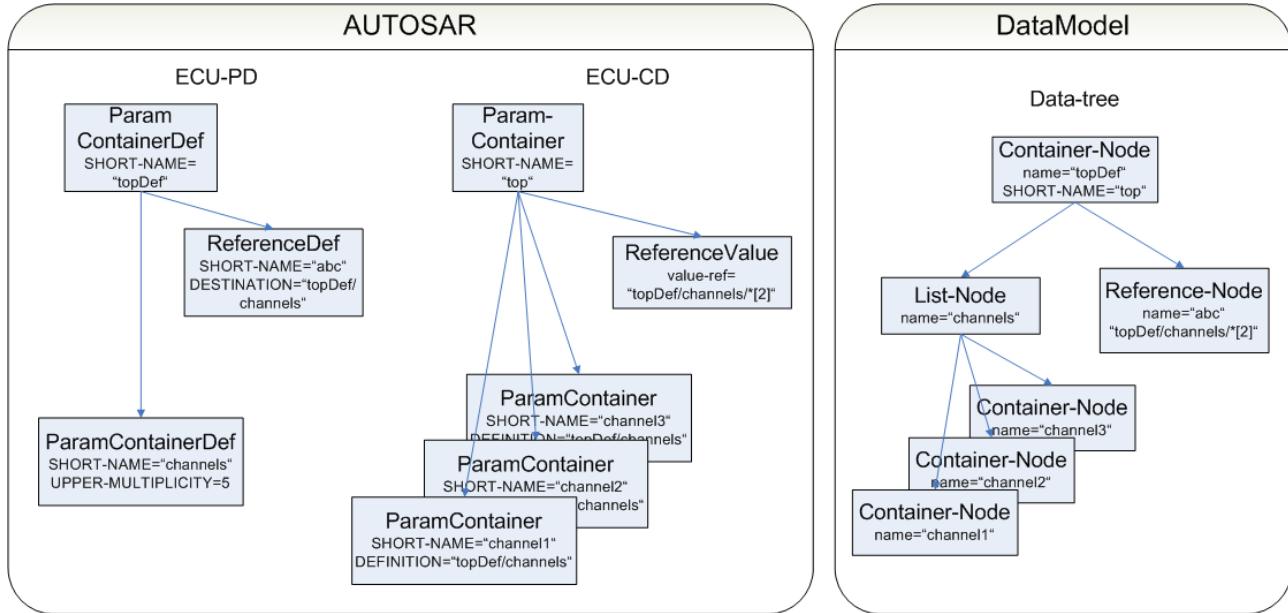


Figure 5.7. Reference-Example

The diagram above shows a part of a data-tree containing a reference-node. The value of the reference-node is the SHORT-NAME-path to the referenced node. The reference-node gets the name from the SHORT-NAME of the ReferenceDef.

XPath (within the data-tree)	Description
topDef/channels/*[2]	selects the container-node 'channel2'
topDef/abc	selects the reference-node
as:ref(topDef/abc) node:ref(topDef/abc)	selects the container-node 'channel2' - the parameter is a reference-node
as:ref('top/channel2')	selects the container-node 'channel2' - the parameter is the SHORT-NAME-path, not the XPath-path!

### 5.2.7.1. URI References

With AUTOSAR 4.2.1, URI references have been introduced. This chapter summarizes the concept behind URI references. Please have a look at the AUTOSAR "Specification of ECU Configuration" for details.

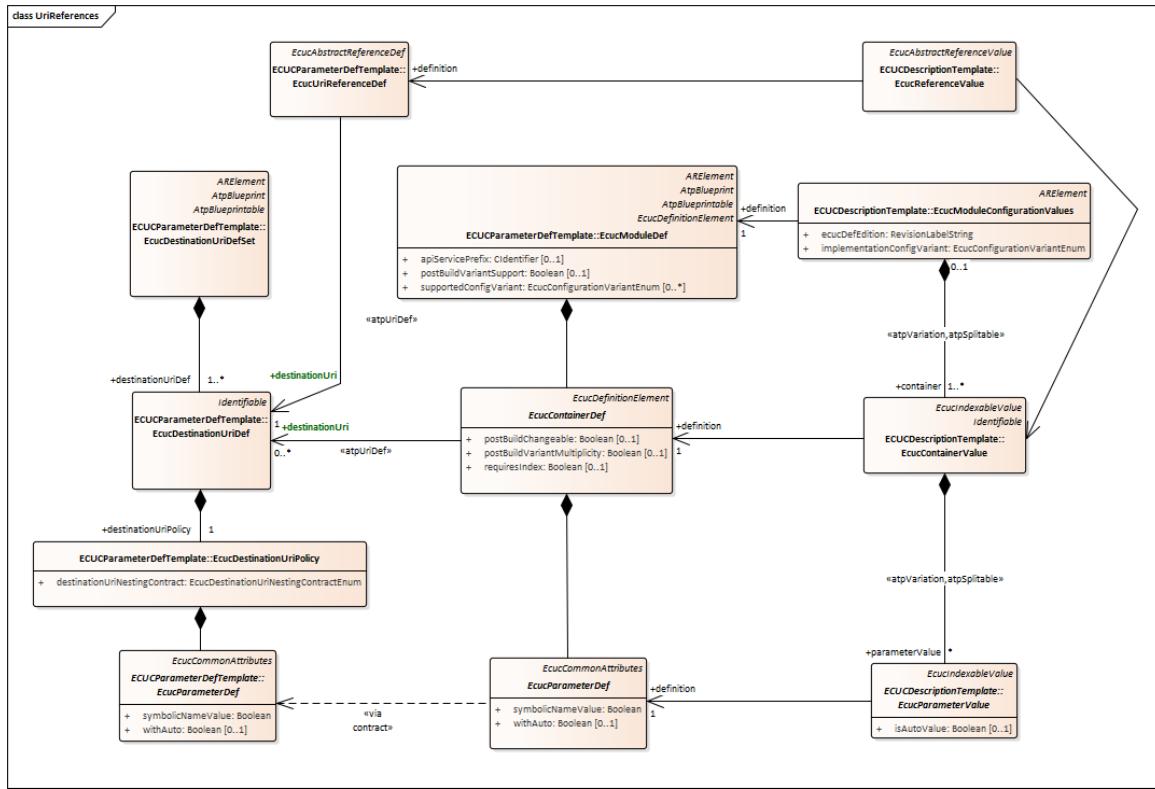


Figure 5.8. URI References in AUTOSAR

The idea behind URI references is, that you can define, which specific configuration parameters are required for a specific feature. Any module which wants to support this feature, can then provide this configuration.

URI references enable you to get this configuration from any module which supports your feature. You dont have to know which modules do actually support your feature beforehand.

This involves the following steps:

- ▶ 1. you have to define the containers/parameters/structure you need beneath an element called "DestinationUriPolicy"
- ▶ 2. any module can now create the same containers/parameters/structure and link it to your DestinationUri (which is basically just the parent-container of your "DestinationUriPolicy")

You can now provide URI references pointing to your DestinationUriPolicy. Valid reference targets for such a reference are all containers (**EcucContainerValue**), whose definitions point to the same DestinationUriPolicy.

Following those references, you can now access all such configurations regardless of the module they are defined in.



#### 5.2.7.1.1. XDM representation

EB tresos Studio does not support the definition of DestinationUriPolicies. Even though this seems to be an essential part of the concept, it is actually not really required for this feature as it only defines the parameter structure as a policy, that the implementing modules shall define. The only thing which is required is the path to the destination URI. This path must be the same for the URI references and the container definition, which defines this parameter structure - and therefore conforms to the policy, regardless if the policy is available in the model or not.

Therefore there are just two elements involved: the reference and the container - both using the same destination URI:

```
<v:ref name="UriRef" type="URI-REFERENCE">
  <a:da name="REF" value="ASPath:/Path/To/DestinationUri"/>
</v:ref>
```

```
<v:ctr name="Container" type="IDENTIFIABLE">
  <a:a name="DESTINATION-URI" value="ASPath:/Path/To/DestinationUri"/>
</v:ctr>
```

The reference above can refer to any container, which has such definition.

## 5.3. How to create a new module

EB tresos AutoCore can be extended with your specific software modules. This chapter provides an overview on how to create a new module in EB tresos Studio.

Example: The Input Output Hardware Abstraction (`IoHwAb`) module provides an interface to internal and external I/O devices of an ECU. As each ECU has its particular I/O capabilities, a specific `IoHwAb` module must be implemented for each ECU. Due to its individual character, the `IoHwAb` is not included in the EB tresos AutoCore product. Of course, you can integrate the module in the same way as an SWC or CDD based on the appropriate SWCD and C source file. But if you want to have a configurable `IoHwAb` or connect it to the `PduR`, you can create the `IoHwAb` module in EB tresos Studio as described in this chapter.

For more information on the `IoHwAb`, see the AUTOSAR Requirements on I/O Hardware Abstraction.

**TIP****Learn more about IoHwAb**

The purpose and integration of the `IoHwAb` module is part of the EB tresos Classic AUTOSAR training.

### 5.3.1. Prerequisites

You need either a standardized module definition (StMD) or a vendor specific module definition (VsMD).

EB tresos Studio contains a copy of the AUTOSAR StMDs from each supported AUTOSAR release in the directories:

- ▶ `autosar/2.0`
- ▶ `autosar/2.1`
- ▶ `autosar/3.0`
- ▶ `autosar/3.1`
- ▶ `autosar/3.2`
- ▶ `autosar/4.0.2`
- ▶ `autosar/4.0.3`
- ▶ `autosar/4.1.3`
- ▶ `autosar/4.2.1`
- ▶ `autosar/4.2.2`
- ▶ `autosar/4.3.0`
- ▶ `autosar/4.3.1`
- ▶ `autosar/4.4.0`
- ▶ `autosar/R19-11`
- ▶ `autosar/R20-11`

To get single files for each module, change e.g. to the directory `autosar/R20-11` of your EB tresos Studio installation and execute the following command:

```
create_epds_460.pl AUTOSAR_MOD_ECUConfigurationParameters.arxml
```

This creates several `.epd` files which contain the StMDs of the AUTOSAR-defined modules.



## 5.3.2. Creating a vendor-specific module definition

According to the AUTOSAR-specification, a vendor has to be derived from a VsMD from the corresponding AUTOSAR StMD. For more details see the users guide, chapter "Creating VendorSpecific module definitions (VSMD)".

To convert a StMD-file to a VsMD-file use the following command:

```
tresos_cmd.bat legacy vsmd <StMD> <VsMD> <package> [<ORIGIN>]
```

## 5.3.3. Converting files

To utilize special features provided by EB tresos Studio, parameter definition files should be shipped in XDM format. The XML format allows to define cross parameter checks, GUI annotation and to add other meta-information. To convert an AUTOSAR file to XDM format or vice versa the `convert` command can be used:

```
tresos_cmd.bat -DMapOptionalAsList=false
    legacy convert <infile>... <outfile>
```

The convert command can also be used to merge different files into one file by just providing multiple input files. E.g.:

```
tresos_cmd.bat -DMapOptionalAsList=false
    legacy convert PduR.epd@asc:2.1 PduR.bmd@asc:2.1 PduR.xdm
```

## 5.3.4. Templates for new modules

EB tresos Studio ships with two templates for new modules that can be found in the directory `demos/Studio`:

`TemplateGenerator_TS_T00D0M0I0R0`

This template should be used when the template based code generator of EB tresos Studio should be used for generating code.

`PublicApiTest_TS_T00D0M0I0R0`

This template should be used if a generator should be implemented in Java or other Java functionality should be used for the module.



The following sections describe how to set up a module with a template-based code generator.

### 5.3.5. Setup of the file structure

Start by copying to your development environment and renaming the directory `TemplateGenerator_TS_T00D0M0I0R0`.

Replace the file `config/TemplateGenerator_TS_T00D0M0I0R0.xdm` by your own VsMD-file that was created as described above.

Remove all files and directories from within the generate-directory. This template-files can only be generated with the unchanged schematic-file of the template-plugin.

### 5.3.6. Adapting MANIFEST.MF

Change the following entries within `META-INF/MANIFEST.MF` according to your needs:

**Bundle-Name**

Human-readable name for your plug-in.

**Bundle-SymbolicName**

Unique name for your plug-in - a good practice is to use the directory-name of the plugin.

**Bundle-Vendor**

The name of the vendor of the plugin

### 5.3.7. Adapting plugin.xml

Change the `plugin.xml` to your needs. The `plugin.xml` is an XML file used to register resources with EB tresos Studio. Registrations are grouped by so called *extensions*. For a first module setup adapting the three extensions defined in the `TemplateGenerator_TS_T00D0M0I0R0` demo module is sufficient.

#### 5.3.7.1. dreisoft.tresos.launcher2.plugin.module

The extension `dreisoft.tresos.launcher2.plugin.module` registers a new module. The most important attribute in this extension is the attribute `id` beneath the `module` tag. This id is used in other extension in the form of an `moduleId` attribute to refer to this module.



The following attributes of the `extension` tag should be adapted:

#### id

A unique id of your plug-in (this is NOT the moduleId).

#### name

The human readable name of your plug-in.

The actual information of the extension can be found as attributes and sub-tags of the `module` tag.  
Adapt the following attributes:

#### id

The `module-id` for the module, this id is referenced by the extensionpoints `dreisoft.tresos.launcher2.api.plugin.modulecluster`, `dreisoft.tresos.launcher2.plugin.configuration` and `dreisoft.tresos.launcher2.plugin.generator`.

#### label

Provides a human readable name for the module. This is the module name that will be presented to the user in the GUI. If this value starts with a percent-sign, it must be written as key-value-pair (separated by a colon) within the file `plugin.properties`.

#### swVersion

Provides the software version on which this module is based. See [Section 5.3.7.5, “Module versions”](#) for details.

#### specVersion

Provides the version of the specification, this module is based. See [Section 5.3.7.5, “Module versions”](#) for details.

#### relVersion

Provides the release version on which this module is based. See [Section 5.3.7.5, “Module versions”](#) for details.

#### categoryType

Autosar type of the module: e.g. Can, CanIf, Lin, LinIf, Com, Os...

#### categoryLayer

Layer of the module: should be one of "Service", "ECU Abs." or "MCAL"; used in the Configurator to group the configurations.

#### categoryCategory

Category of the module: should be one of "IO", "Com", "Diag", "Mem" or "Sys"; used in the Configurator to group the configurations

#### categoryComponent

Must be "ECUC"

With the `ecuType` sub-tag of the `module` tag, the ECU targets/derivatives for which the module is valid can be restricted. If no ECU type is defined, the module is valid for all targets. If the module is valid for several targets, several ecu types can be specified. The following attributes are known for the `ecuType`:



### target

The supported target of your module. The combination of target and derivative must be selected by the user when creating a new configuration-project if your module is to be used.

### derivate (optional)

The supported derivative of your module.

Optionally, you can assign the module to one or to several clusters by adding the `cluster` sub-tag to the `module` tag. The given cluster names are used for grouping modules in the module configuration GUI. You can also provide default cluster assignment and override original cluster assignment using the `modulecluster` extension point. See [Section 5.3.7.2, “Optional: dreisoft.tresos.launcher2.api.plugin.modulecluster”](#) for details.

Examples for cluster names are: Basic Services, OS, Libraries, Memory Stack, Diagnostic, FlexRay Stack, Can Stack etc.

Here you can see a sample of the module extension:

```
<extension point="dreisoft.tresos.launcher2.plugin.module"
  id="YourID"
  name="Your plugins name">

  <module id="Your_Module_Id"
    label="Your modules human readable name"
    mandatory="false"
    legacy="false"
    allowMultiple="false"
    description="Your modules description"
    copyright="YourCopyright"

    swVersionPrefix="2007.b.pl1"
    swVersionMajor="2"
    swVersionMinor="0"
    swVersionPatch="1"
    swVersionSuffix=""

    specVersionPrefix=""
    specVersionMajor="1"
    specVersionMinor="9"
    specVersionPatch="0"
    specVersionSuffix="Rev.17"

    relVersionPrefix="AUTOSAR"
    relVersionMajor="2"
    relVersionMinor="1"
    relVersionPatch="0"
    relVersionSuffix=""
```



```

        categoryType="YourType"
        categoryLayer="Service|ECU Abs.|MCAL"
        categoryCategory="IO|Com|Diag|Mem|Sys"
        categoryComponent="ECUC">
    <ecuType target="YourTarget" derivate="YourDerivate"/>
    <cluster name="Basic Services"/>
    <cluster name="Diagnostic"/>
</module>
</extension>
```

### 5.3.7.2. Optional: dreisoft.tresos.launcher2.api.plugin.modulecluster

With the extension `dreisoft.tresos.launcher2.plugin.modulecluster` you may assign modules to clusters from outside of the `dreisoft.tresos.launcher2.plugin.module` extension point:

- ▶ To provide default cluster assignments, use the `<default>` tag for modules that do not define cluster membership themselves.
- ▶ To supersede existing cluster assignments, use the `<override>` tag.
- ▶ To name one or more clusters, use the `<cluster>` tag.

You may use the `<cluster>` tag also with the `module` extension point. Information on the `module` extension point is available at [Section 5.3.7.1, “dreisoft.tresos.launcher2.plugin.module”](#).

- ▶ To assign modules to the named clusters, use the `<module>` tag.

The `<module>` tag recognizes the attributes `type` and `regExpModuleId` to identify the modules that should be assigned to the given clusters. The `type` attribute must match the module's `categoryType` attribute exactly. The `regExpModuleId` attribute is matched against module IDs as a regular expression pattern as of the standard Java package `java.util.regex`. If you specify both attributes, `type` and `regExpModuleId`, within one `<module>` tag, both must match (i.e. Boolean AND). If more than one `<module>` tag is supplied, only one must match (i.e. Boolean OR).

Usually you define contributions to this extension point in other plugins than those contributing the original modules, i.e. in plugins for organizing third-party modules. If multiple contributions match a given module, union sets are applied.

Example for a `modulecluster` extension:

```
<extension point="dreisoft.tresos.launcher2.api.plugin.modulecluster">
<modulecluster>
    <default>
        <cluster name="Can Stack"/>
```



```

<cluster name="Fr Stack"/>
<module type="PduR"/>
<module type="Com"/>
</default>
<default>
  <cluster name="Can Stack"/>
  <module type="CanIf"/>
</default>
<override>
  <cluster name="Can Stack"/>
  <module type="Can" regExpModuleId="^.*_T16D4M[0-9]I[0-9]R[0-9]$/>
</override>
</modulecluster>
</extension>

```

### 5.3.7.3. dreisoft.tresos.launcher2.plugin.configuration

The extension `dreisoft.tresos.launcher2.plugin.configuration` registers configuration specific resources for the module like the parameter definition, the provided editors, pre- and recommended configuration.

The attributes of the extension tag are standard attributes already described for the module tag.

The toplevel tag beneath the extension tag is called `configuration`. This tag makes the association the module defined by a `module` extension via the attribute `moduleId`. The value of this attribute must match the `id` of the `module` tag of a previous module registration.

Configuration information registered via the configuration extension is grouped into various sub-tags of the `configuration` tag.

#### 5.3.7.3.1. Parameter definition, pre-, recommended configuration

Via the sub-tag `schema` the parameter definition and the pre- and recommended configuration can be registered.

The various information registered by the `schema` tag is grouped into sub-tags.

One mandatory sub tag named `manager` names the Java class that loads a schema resources. The class is selected via the attribute `class` which must be set to `dreisoft.tresos.autosar2.resourcehandling.AutosarSchemaManager`.

Schema information is loaded from files within the module plug-in. The files to load are registered with the tag `resource`. Multiple files can be registered. The `resource` tag accepts the following attributes:



#### id (optional)

The `id` is optional. If pre- or recommended configuration should be supplied this attribute can be used to reference to the file from which to load the pre- or recommended configuration.

#### value

The `value` attribute contains a path relative to the root directory of the module plug-in that says where to find the XDM file from which to load the schema information.

#### type

The type of the resource must be set to XDM.

EB tresos Studio supports multiple pre- configurations per module which are registered via the `plugin.xml`. If at least one pre- configuration is specified the user must select one pre- configuration when creating a module configuration for the module. If multiple pre- configurations are specified the user is free to select one of the provided.

A pre- configuration is described via the sub-tag `preconfig` of the `schema` tag. The following attributes are available:

#### name

The `name` attribute provides the name of the pre- configuration that is shown to the user.

#### resourceId

The `resourceId` attribute selects the resource file from which to load the pre- configuration. The supplied id must match an id of a `resource` tag supplied for this module.

#### path

The `path` attribute provides an XPath or ASPPath to the module configuration in the referenced XDM that forms the pre- configuration.

#### description (optional)

The `description` attribute provides a supplementary text that is shown to the user when selecting the pre- configuration in the GUI.

#### default (optional)

The `default` attribute selects the pre- configuration that is presented to the user a initially selected pre- configuration when adding a new module configuration.

EB tresos Studio supports multiple recommended configurations per module which are registered via the `plugin.xml`. If multiple pre- configurations are specified the user is free to select one of the provided. The user also is allowed to not load any recommended configuration.

A recommended configuration is described via the sub-tag `reccconfig` of the `schema` tag. The following attributes are available:

#### name

The `name` attribute provides the name of the recommended configuration that is shown to the user.

**resourceId**

The `resourceId` attribute selects the resource file from which to load the recommended configuration. The supplied id must match an id of a `resource` tag supplied for this module.

**path**

The `path` attributes provides an XPath or ASPPath to the module configuration in the referenced XDM that forms the recommended configuration.

**description (optional)**

The `description` attributes provides a supplementary text that is shown to the user when selecting the recommended configuration in the GUI.

**default (optional)**

The `default` attributes selects the recommended configuration that is presented to the user a initially selected recommended configuration when adding a new module configuration. If none of the recommended configuration is the default one no recommended configuration is loaded by default.

### 5.3.7.3.2. Module configuration

How the module configuration of a new instance of the module is created is defined by the `data` tag.

Via the tag `manager` the Java class that manages the module configuration can be selected. The attribute `class` must be set to `dreisoft.tresos.autosar2.resourcehandling.AutosarConfigManager`.

The tag `schemaNode` selects the module definition that should be used to create the module configuration. The attribute `path` must contains an XPath or ASPPath pointing to a module definition in one of the registered schema files.

#### 5.3.7.3.2.1. Package structure of the module configuration

Per default, a newly created module configuration will be placed in a package with the same name as the module configuration itself. For example the "Can" module configuration will be contained in a package named "Can". Its ASPPath will therefore be "/Can/Can".

To change this default behavior, you can add a `parameter` sub-tag to the `manager` tag. The sub-tag's attribute `name` must be set to "package" and the attribute `value` must contain the desired package name.

With this tag, it is also possible to define that the module configuration shall be contained within a sub-package by separating the names of the packages using a slash as shown in the example below.

```
<manager
    class="dreisoft.tresos.autosar2.resourcehandling.AutosarConfigManager">
    <parameter name="package" value="parentPkg/subPackage"/>
</manager>
```



With the example above, the module configuration will be contained within the package "subPackage" which itself is contained within the package "parentPkg".

### 5.3.7.3.3. Registering editors

The tag `editor` allows to register editors that are presented to the user for editing the module configuration. EB tresos Studio provides an implementation of a generic configuration editor that can be registered via this tag. An editor registration for a generic configuration editor has the following form:

```
<editor id="AnyUniqueId"
       label="AnyLabel"
       tooltip="AnyTooltip">
  <class class="dreisoft.tresos.launcher2.editor.GenericConfigEditor">
    <parameter name="schema" value="ASPath to your MODULE-DEF"/>
  </class>
</editor>
```

The following attributes must be edited module specific:

**id**

An id that must be unique to identify the editor.

**label**

A label for the editor shown in the GUI.

**tooltip**

A tooltip for the editor shown in the GUI.

The generic configuration editor required the following parameters to be set via the `parameter` tag:

**schema**

Path to the module definition used to create the module configuration. The value must match the path given in the `data` tag.

### 5.3.7.3.4. Code sample

The following sample shows a complete configuration extension:

```
<extension point="dreisoft.tresos.launcher2.plugin.configuration"
          id="AnyUniqueId"
          name="AnyName">
```



```

<configuration moduleId="YourModuleId">
  <schema>
    <manager class="dreisoft.tresos.autosar2.resourcehandling.
      AutosarSchemaManager"/>
    <resource value="RelativePathToYourVsMD" type="xdm"/>
  </schema>
  <data>
    <manager
      class="dreisoft.tresos.autosar2.resourcehandling.AutosarConfigManager"/>
    <schemaNode path="ASPath to your MODULE-DEF"/>
  </data>
  <editor id="AnyUniqueId"
    label="AnyLabel"
    tooltip="AnyTooltip">
    <class class="dreisoft.tresos.launcher2.editor.GenericConfigEditor">
      <parameter name="schema" value="ASPath to your MODULE-DEF"/>
      <parameter name="title" value="AnyTitle"/>
    </class>
  </editor>
</configuration>
</extension>

```

#### 5.3.7.4. dreisoft.tresos.launcher2.plugin.generator

Via the generator extension a code generator can be registered for the module. A module can have any number of code generators. The generator extension supplies the tag `generator` as its single tag. The `generator` tag supports the following attributes:

`id`

A generator can be given a unique id with this attribute.

`moduleId`

The id of the module extension used to register the module.

`class`

The Java class that implements the code generator.

`step (optional)`

The attribute `step` allows to select the phase of the code generation in which the generator is called. Code generation runs in three phases or steps: pre, default, post that are run in this order. Via the values `pre` and `post` the generator can be scheduled into the pre or into the post step.

The template based code generator is implemented in the Java class `dreisoft.tresos.launcher2.generator.TemplateBasedCodeGenerator`. Via the tag parameter the template based code generator can be customized. The following parameters can be used:



#### templates

Selects a path relative to the module plug-in root directory beneath which the code templates can be found.

```
<extension point="dreisoft.tresos.launcher2.plugin.generator"
           id="StandardModule_TS16D4M2I0R0GenExt"
           name="Module PlugIn TemplateGenerator">

<generator
    class="dreisoft.tresos.launcher2.generator.TemplateBasedCodeGenerator"
    id="AnyUniqueId"
    moduleId="YourModuleId"
    <parameter name="templates" value="RelativeTemplatePath"/>
</generator>
</extension>
```

#### 5.3.7.5. Module versions

In the `plugin.xml`, the software version, the specification version and the release version can be specified for each module. This information is shown to the user in the **EB tresos Details** dialog which shows all installed modules. To open the **EB tresos Details** dialog:

- ▶ Select **EB tresos Details...** from the **Help** menu.

The **EB tresos Details** dialog opens up.

- ▶ Switch to the **Module Registry** tab.

The software version, the specification version and the release version - among further information - are displayed for all installed modules.



EB tresos Details

EB tresos Details										
<a href="#">General</a> <a href="#">Module Registry</a> <a href="#">Bundle Registry</a>										
<input type="text" value="Filter:"/>										
I	Id	Label	Type	Category	Layer	Component	SWVer	SpecVer	Release	ECUTypes
	NvM_TS_TxDxM6IOR0	NvM (V6.0.0, AS4.0.2)	NvM	Memory	Memo...	ECUC	6.0.0	3.1.0 2	4.0.2	[ ]
	Fls_TS_T0D1M4I1R0	Fls (V4.1.0, AS4.0.2)	Fls	Stub	Stub	ECUC	4.1.0	3.1.0 0002	4.0.2	[ ]
	Gpt_TS_T0D1M4I1R0	Gpt (V4.1.0, AS4.0.2)	Gpt	Stub	Stub	ECUC	4.1.0	3.1.0 0002	4.0.2	[ ]
	IpduM_TS_T0D1M4I1R0	IpduM (V4.1.0, AS4.0.2)	IpduM	Stub	Stub	ECUC	4.1.0	2.1.0 0002	4.0.2	[ ]
	Ts5Atl_TS_TxDxM2I0R0	Ts5Atl (V2.0.0, AS4.0.2)	Ts5Atl	Test	Service	ECUC	2.0.0	0.0.0 0000	4.0.2	[ ]
	WdgM_TS_T0D1M2IOR0	WdgM (V2.0.2, AS3.1.0)	WdgM	Stub	Stub	ECUC	2.0.2	1.2.0 0003	3.1.0	[ ]
	PduR_TS_T0D1M4I1R0	PduR (V4.1.0, AS4.0.2)	PduR	Stub	Stub	ECUC	4.1.0	3.1.0 0002	4.0.2	[ ]
	Cdd2_TS_T0D1M4I1R0	Cdd2 (V4.1.0, AS4.0.2)	Cdd2	Stub	Stub	ECUC	4.1.0	3.1.0 0002	4.0.2	[ ]
	Can_TS_T19D1M2IOR0	Can (V2.0.1, AS4.0.2)	Can	CAN	Com...	ECUC	2.0.1	3.1.0 0000	4.0.2	[WINDOWS/WI...]
	FrNm_TS_TxDxM5IOR0	FrNm (V5.0.0, AS4.0.2)	FrNm	FlexRay	Com...	ECUC	5.0.0	4.1.0 0002	4.0.2	[ ]
	IpduM_TS_TxDxM3IOR0	IpduM (V3.0.3, AS4.0.2)	IpduM	Com Services	Com...	ECUC	3.0.3	2.1.0 0002	4.0.2	[ ]
	Wdg_TS_T0D1M2IOR0	Wdg (V2.0.2, AS3.1.0)	Wdg	Stub	Stub	ECUC	2.0.2	2.2.0 0003	3.1.0	[ ]
	Base_TS_TxDxM5IOR0	Base (V5.0.2, AS4.0.2)	Base	EB	EB Bas...	ECUC	5.0.2	0.0.0 0000	4.0.2	[ ]
	Platforms_TS_T19D1M2IOR0...	Platforms (V2.0.0, AS4....)	Platforms	EB	EB Bas...	ECUC	2.0.0	0.0.0 0000	4.0.2	[WINDOWS/WI...]

???

?

OK

Figure 5.9. The **EB tresos Details** dialog

Alternatively, the user can view the version information in the **Details** area on the right side of the module configurations UI, e.g. in the **New Project Wizard** or in the **Module Configurations** dialog. To open the **Details** area, the user has to select the module either in the **Available Modules** tree or in the **Module Configurations** table of the module configurations UI.

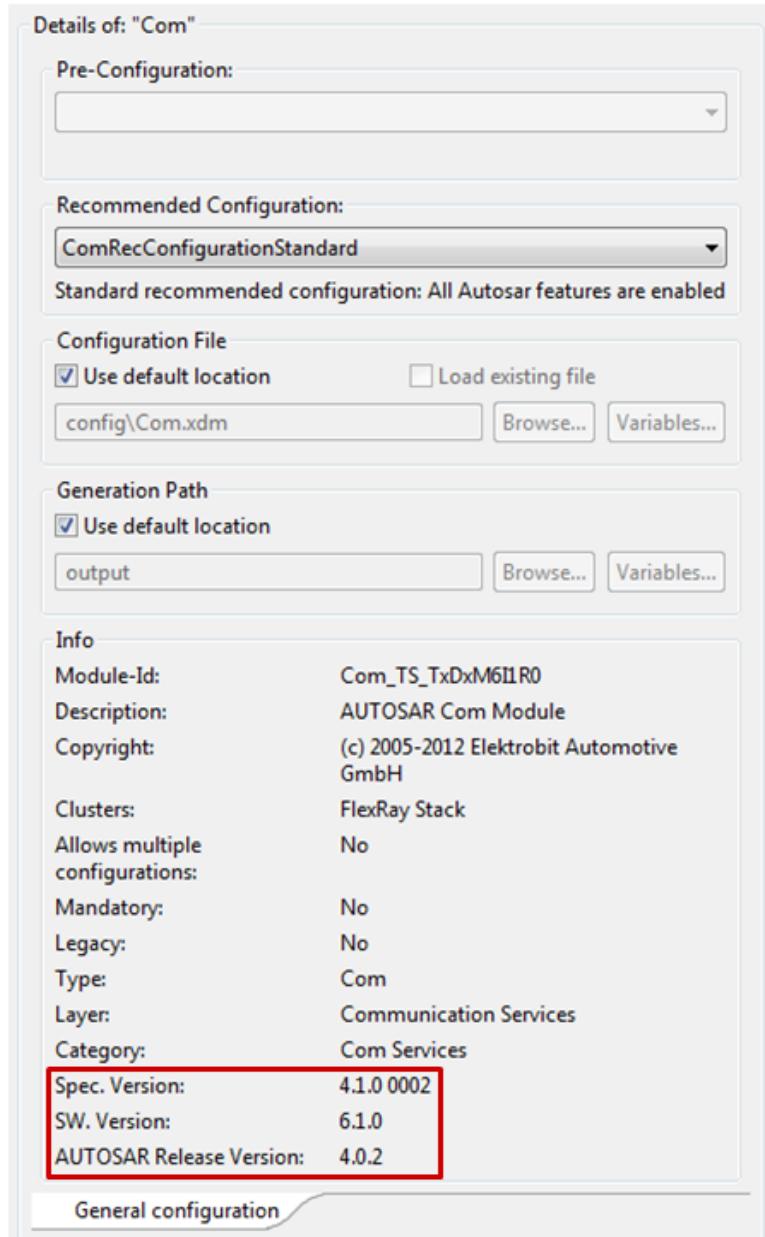


Figure 5.10. Viewing module version information (of the module configuration selected)

Each of the versions optionally may have a prefix and/or a suffix. This is only informal information and not evaluated in any way. Nevertheless, this information is presented to the user and thus should be self-explanatory.

#### 5.3.7.5.1. Software version

Provides the software version on which this module is based on. This might be e.g. software version of the contained code-generator or of the software which handles the generated code. This might also be used to define module-dependencies in future releases.



### 5.3.7.5.2. Specification version

Provides the version of the specification, this module is based on. This might be used to define module-dependencies in future releases.

### 5.3.7.5.3. Release version

Provides the release version on which this module is based on. The user has to define a "Release Version" for each newly created project which restricts the available modules for that project. For this, only the major and minor version is used.

## 5.3.8. Code templates

If not changed in the `plugin.xml` the code generator searches the code templates within the plug-in underneath the directory `generate`.

A good practice is to create a directory within the generation-path with the name of your plugin since there might be several modules having the same output-path.

## 5.3.9. Adding the module to EB tresos Studio

To install the plug-in to EB tresos Studio, copy the whole directory to the EB tresos Studio installation into the directory `plugins`.

Due to a bug in Eclipse, all directories named `org.eclipse.*` underneath the configuration directory of the EB tresos Studio installation have to be removed.

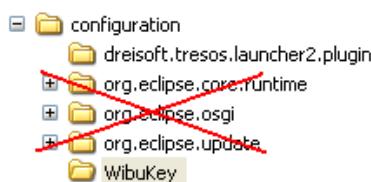


Figure 5.11. Directories to delete

The `org.eclipse.*` directories contain cached information about installed plugins. Not deleting them can cause several undefined effects.

After restarting EB tresos Studio the new plug-in will be available.

If you have purchased EB tresos AutoCore together with EB tresos Studio, you can also integrate the plugin-in with the User Build environment. You will find the instructions on how to do this in the EB tresos AutoCore documentation: EB tresos AutoCore documentation/EB tresos AutoCore user's guide/Build environment.

## 5.4. How to migrate your module from one AUTOSAR version to another

### 5.4.1. How to migrate an AUTOSAR 2.x module to 3.x

Converting an AUTOSAR 2.x module into an AUTOSAR 3.x must be done with the following order:

1. Use the "legacy vsmd" commandline command to create a VSMD XDM file from the AUTOSAR 3.x StMD file.
2. Merge all vendor specific changes from the existing AUTOSAR 2.x parameter definition into the newly created AUTOSAR 3.x vendor specific parameter definition file.
3. Use the "legacy vsmdcheck" commandline command to ensure that the new parameter definition conforms to AUTOSAR 3.x.

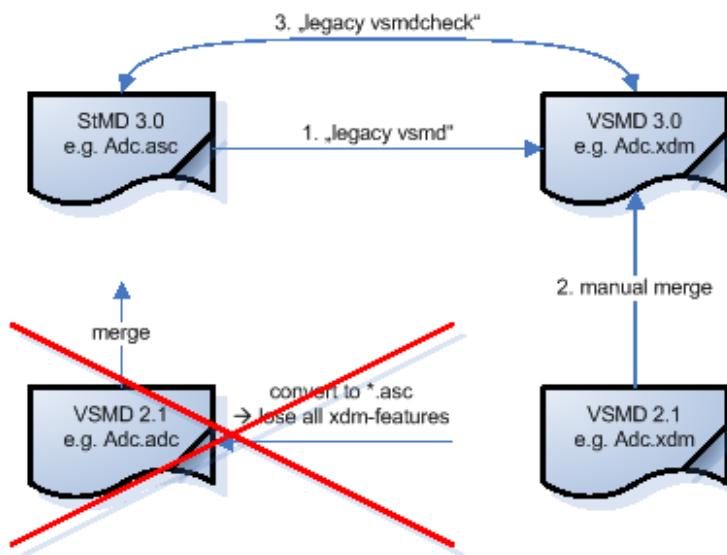


Figure 5.12. EB tresos Studio details

**NOTE**

Do not convert existing XDM-file to the AUTOSAR format as all XDM specific features would be lost in this way.



In the following sub sections the changes between AUTOSAR 2.1 and AUTOSAR 3.0 which must be considered when switching a parameter definition to AUTOSAR 3.x are presented. For further information see the AUTOSAR documents.



### 5.4.1.1. Choices

Since AUTOSAR 3.0 choice within XDM files are of type `IDENTIFIABLE` (in previous AUTOSAR they had been of type `CHOICE`):

wrong:

```
<v:chc name="CanTrcvAccess" type="CHOICE">
```

correct:

```
<v:chc name="CanTrcvAccess" type="IDENTIFIABLE">
```

This is especially important when merging the old and the new vsmd XDM file. The VSMD checker will mark invalid `v:chc` tags as an error.

---

**NOTE**



As choices are now represented as `IDENTIFIABLE` elements in the ECU Configuration Description all AUTOSAR paths to elements beneath a choice in the configuration must be adjusted. This means that existing configuration cannot be used without change even if the schema itself does not change between AUTOSAR 2.1 and AUTOSAR 3.x. The same holds for pre configurations and recommended configurations.

---

### 5.4.1.2. Configuration variant

With AUTOSAR 3.0 configuration variants have been introduced. With this new features one module parameter definition can be used to describe a compile time, and a link time and a post build module implementation. This allows the user to select the actual variant of the module when editing the module configuration. Which configuration variants a module actually supports is described in the VSMD. So the module developer is in charge to decide which variants the module will support.

The AUTOSAR StMD normally defines all variants for each module. The creator of a VSMD module definition has to decide which variants are actually supported and remove the excessive variants from the VSMD file. To do this the variable `IMPLEMENTATION_CONFIG_VARIANT` has to be adjusted:

```
<v:var name="IMPLEMENTATION_CONFIG_VARIANT" type="ENUMERATION">
  <a:a name="LABEL" value="Config Variant"/>
  <a:da name="DEFAULT" value="VariantPreCompile"/>
  <a:da name="RANGE">
    <!-- Remove not supported variants from this list -->
    <a:v>VariantPreCompile</a:v>
```



```

<a:v>VariantLink</a:v>
<a:v>VariantPostBuild</a:v>
<a:v>VariantPostBuildSelectable</a:v>
</a:da>
</v:var>

```

Each configuration parameter specifies its configuration class in each configuration variant. For all parameters taken over from the StMD these lists have to be adjusted. For all vendor specific parameters a corresponding mapping has to be added. The configuration class mapping looks as follows in the XDM:

```

<v:var name="AdcHwUnitId" type="INTEGER">
  <a:a name="IMPLEMENTATIONCONFIGCLASS"
    type="IMPLEMENTATIONCONFIGCLASS">
    <!-- adjust list of variant mappings -->
    <icc:v class="PreCompile">VariantPreCompile</icc:v>
    <icc:v class="Link">VariantPostBuild</icc:v>
    <icc:v class="PostBuild">VariantPostBuild</icc:v>
    <icc:v class="PostBuildSelectable">VariantPostBuildSelectable</icc:v>
  </a:a>
</v:var>

```

#### 5.4.1.3. ADMIN-DATA

Ever since the release AUTOSAR 3.0, ADMIN-DATA must be added beneath MODULE-DEF like this:

```

<d:chc name="CanIf" type="AR-ELEMENT" value="MODULE-DEF">
  <v:ctr type="MODULE-DEF">
    <a:a name="ADMIN-DATA" type="ADMIN-DATA">
      <ad:ADMIN-DATA>
        <ad:LANGUAGE>EN</ad:LANGUAGE>
        <ad:DOC-REVISIONS>
          <ad:DOC-REVISION>
            <ad:REVISION-LABEL>1.2</ad:REVISION-LABEL>
            <ad:ISSUED-BY>anonymous</ad:ISSUED-BY>
            <ad:DATE>2008-5-26T21:32:52</ad:DATE>
            <ad:MODIFICATIONS>
              <ad:MODIFICATION>
                <ad:CHANGE>
                  <ad:L-2 L="DE">ChangeTitle</ad:L-2>
                </ad:CHANGE>
                <ad:REASON>
                  <ad:L-2 L="EN">Reason</ad:L-2>
                </ad:REASON>
              </ad:MODIFICATION>
            </ad:MODIFICATIONS>
          </ad:DOC-REVISION>
        </ad:DOC-REVISIONS>
      </ad:ADMIN-DATA>
    </a:a>
  </v:ctr>
</d:chc>

```



```

        </ad:REASON>
    </ad:MODIFICATION>
</ad:MODIFICATIONS>
</ad:DOC-REVISION>
</ad:DOC-REVISEONS>
</ad:ADMIN-DATA>
</a:a>

```

## 5.4.2. How to migrate your AUTOSAR module to AUTOSAR version 4.x

To migrate your module to AUTOSAR 4.x, perform the folowing tasks:

1. In your XDM file, add the line `<a:a name="RELEASE" value="asc:4.0"/>` within the xml tag `<v:ctr type="MODULE-DEF">`

```

<d:ctr type="AUTOSAR" factory="autosar" ... >
<d:lst type="TOP-LEVEL-PACKAGES">
<d:ctr name="AUTOSAR" type="AR-PACKAGE">
<d:lst type="ELEMENTS">
<d:chc name="Adc" type="AR-ELEMENT" value="MODULE-DEF">
<v:ctr type="MODULE-DEF">
<a:a name="RELEASE" value="asc:4.0"/>
<v:lst name="AdcConfigSet" type="IDENTIFIABLE">
<v:ctr name="AdcConfigSet" type="IDENTIFIABLE">

```

2. The AUTOSAR 4.0 format redefined most of the XML-tags. EB tresos Studio handles this tag renaming transparently in XDM files, so there is you do not have to change your XDM files manually.

## 5.5. How to sign modules with the crypto command

### 5.5.1. Purpose

This chapter describes how to sign a module.



Partners of EB receive special redistributable licenses that allow them to redistribute their modules on their own. To do this, these modules must be signed with a special cryptographic key. Only modules signed with this key can be redistributed with this special license.

In the future EB may also provide the possibility for partners to encrypt parts of their modules but currently this feature has not been made available for outside EB.

## 5.5.2. Prerequisites

Partners of EB must use the crypto tool to sign their modules. To be able to operate the tool, partners receive the following:

1. Redistributable licenses for the partner's customers
2. A provider-specific feature license for using the crypto tool. This license is not for redistribution
3. A provider-ID string, which is required for signing modules. For example, the name of your company

## 5.5.3. Commands

Modules are signed by using the crypto command via the commandline.

Assuming the module is installed in the directory `plugins/Can_Vendor` with the following file structure:

```
config/Can_Vendor.xdm  
generate/Can_Cfg.c
```

In this case the module can be signed as follows:

```
$ tresos_cmd.bat crypto plugins/Can_Vendor\  
      -v \  
      -ski VendorCompany \  
      -files \  
      config/Can_Vendor.xdm;generate/Can_Cfg.c
```

`VendorCompany` (`-ski` option) is the provider-ID you received from EB, see above.

The tool signs all files given with the `-files` commandline option. Files are separated by `'.'`. The paths are relative to the module plug-in directory (the Base-Path), given as the first parameter in this example. If the `-`



files option is not used, the list of files to be processed is expected from stdin. The tool creates a file META-INF/CRYPTOMANIFEST.MF that contains the signatures.

The following files must be signed:

- ▶ Parameter definition files (.xdm)
- ▶ All code templates and macro files (generally speaking all files that are processed by the code generator)

## 5.6. Module Transformer API

### 5.6.1. Purpose

Module transformers are used to convert an existing module configuration to another version or target/derivative. If you have an AUTOSAR 2.1 configuration project and want to import it to an AUTOSAR 3.0 project, the module specific transformers are called to convert the configuration to the newer version.

### 5.6.2. API

If you installed the eclipse tools from the EB tresos Studio tools folder, the documentation of the DCtxt (data-model Public API) can be found in the Eclipse Help Contents with the title `Public API Java Documentation`. If you have EB tresos Studio set as target platform, the help is also available as javadoc in the `java eclipse` editor.

### 5.6.3. Registering a module transformer

To register a module transformer, the `dreisoft.tresos.launcher2.api.plugin.moduletransformer` extension point has to be registered in the new module's `plugin.xml`. Conduct the following steps:

Add the extension point via the `plugin.xml` editor.

Enter the module id of the module to which it belongs and assign an id for the transformer. The module id is `MyModule_TS_T16D4M2I1R0` then the id for the transformer should be something like `MyModuleTransformer_TS_T16D4M2I1R0`. The `moduleId` specifies the target module to which an old configuration shall be transformed. The version of the source module can be specified within the `enablement` attribute which is explained in one of the next paragraphs.



Add subelement to the moduletransformer named `type`. The type how good the transformer is (fully, partially, fallback). "fully" wins against a "partially" and "partially" wins against "fallback".

Add subelement to the moduletransformer named `transformer`. This element defines the class, that will do the transformation. The class must implement `dreisoft.tresos.launcher2.module.transform.IModuleTransformer`. To support the full functionality of the extension-point it must implement `dreisoft.tresos.launcher2.module.transform.AbstractModuleTransformer`. Clicking on the `class:` link will open the class creation wizard, with the right abstract class already added.

Add subelement to the moduletransformer named `transformation`. Enter the kind of transformation that will be performed. Possible conversions are "version" or "target".

Add subelement to the moduletransformer named `enablement`. Within the enablement you can enter expression, that can query some variables of a module and compare them to values you have entered. If the expressions evaluate to true, the transformer will be enabled for this module. The following variables can be used:

- ▶ `id`: Module-Id (e.g. PduR\_TS\_T16D4M2I0R0)
- ▶ `type`: Type of the module (e.g. PduR)
- ▶ `category`: The module category (e.g. Com)
- ▶ `layer`: The module layer (e.g. Service)
- ▶ `relVersion`: The release version (e.g. 2.1.1, 2.1 = 2.1.\* , 2 = 2.\* = 2.\*.\* )
- ▶ `specVersion`: The module spec version (e.g. 2.1.1, 2.1 = 2.1.\* , 2 = 2.\* = 2.\*.\* )
- ▶ `swVersion`: The software version of the module (e.g. 2.1.1, 2.1 = 2.1.\* , 2 = 2.\* = 2.\*.\* )
- ▶ `ecuTypes`: The target/derviates supported (e.g. V850/V850EGP1, V850, S12X)

### 5.6.3.1. Examples

Expression Examples:

```
<enablement>
<and>
<with variable="ecuTypes"><iterate value="V850"/></with>
<with variable="relVersion"><equals value="2.0.*"/></with>
</and>
</enablement>

<enablement>
<or>
<with variable="ecuTypes"><iterate value="V850"/></with>
<with variable="ecuTypes"><iterate value="S12X"/></with>
</or>
```



```

</enablement>

<enablement>
<and|or|not>
<with variable="ecuTypes">
<iterate value="TRICORE/TC1766"/>
</with>
</and|or|not>
</enablement>

<enablement>
<with variable="type"><equals value="Can"/></with>
</enablement>

```

The following example registers a module transformer, that transforms a module with name MyModule from version 2.0 to 2.1.

```

<extension id="MyModuleTransformerExtId"
           point="dreisoft.tresos.launcher2.api.plugin.moduletransformer">
  <moduletransformer id="MyModuleTransformer_TS_T16D4M2I1R0"
                     moduleId="MyModule_TS_T16D4M2I1R0">
    <type value="partially"/>
    <transformation type="version"/>
    <enablement>
      <and>
        <with variable="type"><equals value="MyModule"/></with>
        <with variable="relVersion"><equals value="2.0"/></with>
      </and>
    </enablement>
    <transformer class="eb.autocore.mymodule.transformer.Rel20Transformer"/>
  </moduletransformer>
</extension>

```

## 5.6.4. Implementing a module transformer Class

If you followed the steps in the previous section, you should have a java class that extends the AbstractModuleTransformer. The only method you have to implement is the abstract method

```

public boolean doTransform( ModuleDescription sourceModule,
                           ModuleDescription targetModule,
                           DCtxt moduleConfiguration )

```

The parameters will give you access on information about the source and the target modules, as versions, target, etc. The DCtxt contains the source configuration of the module, that shall be transformed. The config-



uration is not bound to a schema file, so you can delete complete nodes as containers, variables, references from the config and add them at another location. The DCtxt gives you access to methods that allow you to conduct those operations. The methods to operate on an unbound configuration all start with the prefix "ns" (e.-g. nsAddContainer).

## 5.6.5. Chaining of module transformers

Module transformers can also be chained. That means, that there must not be one transformer for each possible combination of version-to-version conversion. If you already have a transformer that transforms from version 2.0 to 2.1 and now you want to transform your module configuration from version 2.0 to 3.0 you only need to implement a transformer that transforms from version 2.1 to 3.0. At last you need to register a *fully* transformer, that just calls the right transformers to convert from 2.0 to 3.0.

# 5.7. Variant handling and post-build loadable support

## 5.7.1. Purpose

Read this chapter to be able to develop a module with variant handling and/or post-build loadable support. Thus you know the following:

- ▶ How to create a module that supports the selection of post-build selectable and loadable variants
- ▶ How to mark parameters as variant-aware in ARXML and XDM
- ▶ How to access variant functionality via XPath functions
- ▶ How to mark a code generator as variant-aware

## 5.7.2. Prerequisites

Before you read this chapter, have a look into the EB tresos Studio user's guide starting from chapter *Working with variants and Post-build loadable*. It describes the variant handling concept:

- ▶ The difference between PreBuild and PostBuild variation points



- ▶ The Post-build loadable and Post-build selectable concept
- ▶ The definition of PredefinedVariants
- ▶ Restrictions in EB tresos Studio

**NOTE**

**Read the user's guide chapter *Working with variants and Post-build loadable first***



The knowledge in these chapters is a prerequisite when you develop a module with variant support.

### 5.7.3. Module with support for variant selection

To support variant handling in one or multiple modules of a project, one dedicated module is required that supports the selection of variants. Elektrobit Automotive GmbH (EB) provides this functionality e.g. within the EcuC module.

The module must contribute to the `dreissoft.tresos.autosar2.api.plugin.postbuildSetup` extension point. Within the extension point you must define the paths to the configuration parameters for the following:

- ▶ The list of available selectables
- ▶ The current active selectable
- ▶ The current active loadable

For more information on the extension point, see the extension point description. Follow the hints in the EB tresos Studio developer's guide chapter *How to open extension point descriptions*.

An example of a module that contributes to this extension point is contained in the provided Demo plugin. See [Section 5.7.9, “Variant handling demo plug-in”](#).

### 5.7.4. Marking elements as post-build selectable variant aware

A container or parameter may only be equipped with a post-build selectable variation point if it is explicitly marked as variant-aware in the module definition like this:

```
<POST-BUILD-VARIANT-MULTIPLICITY>true</POST-BUILD-VARIANT-MULTIPLICITY>
<POST-BUILD-VARIANT-VALUE>true</POST-BUILD-VARIANT-VALUE>
```

The same can be represented in XDM in the following format:



```
<a:a name="POSTBUILDVARIANTMULTIPLICITY" value="true"/>
<a:a name="POSTBUILDVARIANTVALUE" value="true"/>
```

**NOTE****Restrictions for variant awareness**

You can define `variant-multiplicity` only for elements with a lower multiplicity different from its upper multiplicity. You can define `variant-value` only for variables.

Examples of these parameters are contained in the provided demo. See [Section 5.7.9, “Variant handling demo plug-in”](#).

## 5.7.5. AUTOSAR attribute postBuildVariantUsed

With AUTOSAR 4.4, the attribute `postBuildVariantUsed` has been introduced for module configurations. In case, the user has or plans to have (i.e., introduced at link or post-build time) post-build selectable variation points, this attribute has to be set to true by the user.

Even if there are no variation points yet at pre-compile time, the code generator can decide via this attribute, if there will be variation points at link or post-build time.

In EB tresos Studio, this attribute is represented via a boolean parameter and therefore has to be present in the module definition xdm schema as well. When you convert an arxml file to xdm, this parameter is automatically created. In case you manually create your xdm or update it to support version AUTOSAR 4.4, you have to manually add this parameter inside the module definition xml tag:

```
<v:var name="POST_BUILD_VARIANT_USED" type="BOOLEAN">
  <a:a name="LABEL" value="Post Build Variant Used"/>
  <a:a name="TOOLTIP" value="Indicates whether ..."/> <!-- from ASR spec -->
  <a:a name="DESC" value="Indicates whether ..."/> <!-- from ASR spec -->
  <a:da name="DEFAULT" value="false"/>
  <!--shall be readonly in case postBuildVariantSupport is set to false -->
  <a:da name="READONLY" value="false"/>
  <a:a name="POSTBUILDVARIANTVALUE" value="false"/>
  <a:a name="IMPLEMENTATIONCONFIGCLASS" type="IMPLEMENTATIONCONFIGCLASS">
    <!-- one entry for each supported ConfigVariant, all set to PreCompile -->
    <icc:v mcclass="PreCompile">VariantPreCompile</icc:v>
    <icc:v vclass="PreCompile">VariantPostBuild</icc:v>
  </a:da>
</v:var>
```



## 5.7.6. Marking elements as post-build loadable variant-aware

The loadable and the selectable approach are decoupled by AUTOSAR in release 4.2 and you can configure and handle them completely independent from each other. For the loadable approach, a container or parameter needs to specify MULTIPLICITY-CONFIG-CLASSES and/or VALUE-CONFIG-CLASSES for POST-BUILD:

```
<MULTIPLICITY-CONFIG-CLASSES>
  <ECUC-MULTIPLICITY-CONFIGURATION-CLASS>
    <CONFIG-CLASS>POST-BUILD</CONFIG-CLASS>
    <CONFIG-VARIANT>VARIANT-POST-BUILD</CONFIG-VARIANT>
  </ECUC-MULTIPLICITY-CONFIGURATION-CLASS>
  ...
</MULTIPLICITY-CONFIG-CLASSES>

<VALUE-CONFIG-CLASSES>
  <ECUC-VALUE-CONFIGURATION-CLASS>
    <CONFIG-CLASS>POST-BUILD</CONFIG-CLASS>
    <CONFIG-VARIANT>VARIANT-POST-BUILD</CONFIG-VARIANT>
  </ECUC-VALUE-CONFIGURATION-CLASS>
  ...
</VALUE-CONFIG-CLASSES>
```

The same is represented in XDM like this:

```
<a:a name="IMPLEMENTATIONCONFIGCLASS" type="IMPLEMENTATIONCONFIGCLASS">
  <icc:v vclass="PostBuild">VariantPostBuild</icc:v>
  <icc:v mclass="PostBuild">VariantPostBuild</icc:v>
</a:a>
```

*vclass* corresponds to ECUC-VALUE-CONFIGURATION-CLASS

*mclass* corresponds to ECUC-MULTIPLICITY-CONFIGURATION-CLASS

If you mark an element in this way as POST-BUILD, the user is able to configure multiple configurations for different loadables. For this, the user must configure the loadable use case as described in chapter **Post-build loadable configuration using variant handling** of the EB tresos Studio user's guide.

## 5.7.7. Variant handling API

Several XPath functions are available to access variant handling functionality for post-build selectables. These functions are described in [Section 6.3.4.5.5.9, “Variant functions”](#) which are all prefixed with `variant::`. For details about the single commands, see the table in this chapter.



Keep in mind that you can also access the XPath API always by using the DCtxt API, for example:

```
DCtxt myDCtxt;  
String variantName = myDCtxt.xpath.getValue("variant:name()");
```

By intention, it is not possible to programmatically switch between the different variants. It is also not possible to access values of a different variant. The reason behind this is the concept that every code always only works on one consistent variant. For example a code generator shall always only generate code for one, the currently selected variant. Also a code generator is automatically called once for each variant. For an example how to write a variant-aware code generator that uses this API, see the provided demo in [Section 5.7.9, “Variant handling demo plug-in”](#).

---

**NOTE****Only a selectable use case is targeted**

The variant handling API only targets the selectable use case because it is not necessary to access different loadable configurations at once.

---

## 5.7.8. Variant-aware code generators

If your code generator needs to run once for each configured Post-build selectable variant, you need to mark this code generator as variant-aware. For more information, see [Section 7.1.3, “Defining variant-aware code generators”](#).

The provided demo [Section 5.7.9, “Variant handling demo plug-in”](#) has several examples for variant-aware code generators.

## 5.7.9. Variant handling demo plug-in

The demo plug-in is available in the folder <tresos-install-loc>/demos/Studio/VariantsDemo.

If you want to adapt the demo, install the plug-in of the demo according to the instructions in [Section 4.9, “How to install a Public API demo plugin”](#).

If you just want to have a look at the demo and the variant configuration described in this chapter, you can just copy the VariantsDemo folder to the plugins folder of your EB tresos Studio installation and restart EB tresos Studio.

The project directory contains a fully configured configuration project that has two loadable and two selectable variants configured. Additionally, the containers and parameters of the VariantsDemo module are assigned to different variants.



To import the project, choose **File/Import...** from the menu and select **General/Existing Projects into Work-space**. Then browse to the <tresos-install-loc>/demos/Studio/VariantsDemo/project directory and click **Finish**. For detailed instructions, see the EB tresos Studio user's guide chapter *Importing a project*.

After you imported the project the variants configuration looks like this:

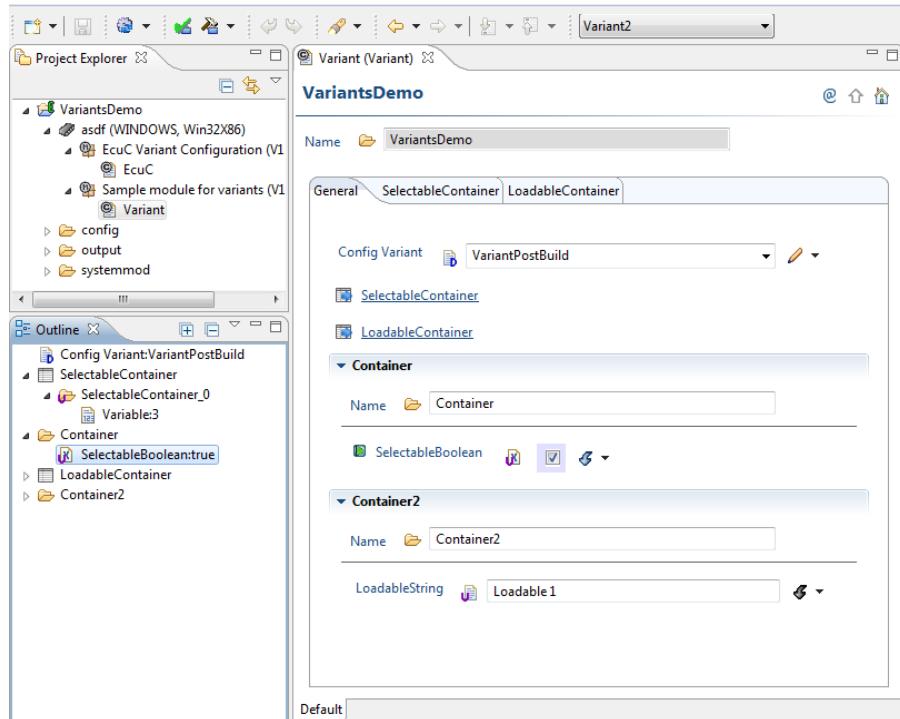


Figure 5.13. The Variant module (Variant2 selected)

### 5.7.9.1. The EcuC module

The contained **EcuC** module contains the registration point for a variant configuration, see [Section 5.7.3, “Module with support for variant selection”](#), and a configuration stripped to the variant handling use case.

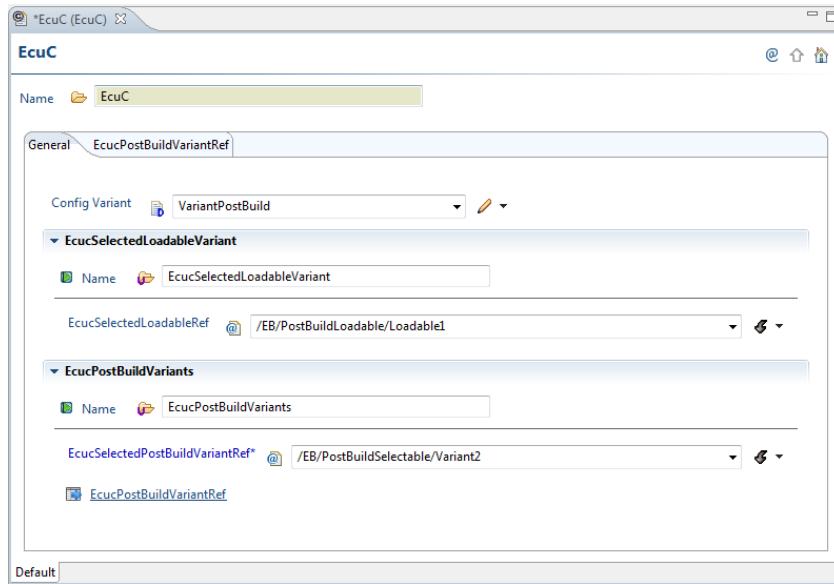


Figure 5.14. The EcuC module

There is the optional reference `EcucSelectedLoadableRef` with which you can enable the loadable use case by referring a loadable `PredefinedVariant`.

There is the list `EcucPostBuildVariantRef` where all `PredefinedVariants` must be configured which are part of the ECU and which are therefore generated together. This list is only enabled if the reference `EcucSelectedPostBuildVariantRef` is enabled.

And there is the optional reference `EcucSelectedPostBuildVariantRef` with which you can enable the selectable use case. With this reference you can specify which variant is currently selected. This must be one of the variants configured in the list `EcucPostBuildVariantRef`.

### 5.7.9.2. The variant module

The variant-aware parameters are marked with a **V** icon in the generic configuration editor and in the **Outline** view. For more information, see [Figure 5.13, “The Variant module \(Variant2 selected\)”](#).

For the selectable use case, this module contains a list of containers named `SelectableContainer` and a boolean variable named `SelectableBoolean`. As soon as selectable variants are configured, you can equip these elements with variation points.

For the loadable use case, this module contains a list of containers named `LoadableContainer` and a string variable named `LoadableString`. As soon as a loadable variant is configured, you can equip these elements with variation points.

The module definition is provided both as XDM file `VariantsDemo.xdm` and ARXML file `VariantsDemo.arxml`.



### 5.7.9.3. The code generators

The demo provides code generators for different types of generators: `TemplateBasedCodeGenerator`, `NGGenerator` and `ExternalGenerator`. All code generators are marked as variant-aware in the `plugin.xml`:

```
<parameter name="variantAware" value="true"/>
```

Variant-aware code generator run once for each selectable variant, that is configured in the `EcuC` module in the `EcucPostBuildVariantRef` list.

If you click the **Generate** button, all available generators run. However, it is recommended to delete the output folder and afterwards select one generation mode via the drop-down box in the toolbar. The following generation modes are available:

- ▶ `Template_withPostRename`

A variant-aware `TemplateBasedCodeGenerator` is registered for step `default`. It generates a file `output/VariantsTestTemplate.txt`. But this file would be overwritten as soon as the generator is called for the next variant. Therefore a variant-aware `NGGenerator` generator is registered for the `post` step which just renames this file to `output/VariantsTestTemplate_<variantName>.txt`.

The generated file contains the name of the generated variant by using the variable and the XPath function `variant:name()`.

- ▶ `External_outputWithPostfix`

A variant-aware `ExternalGenerator` is registered. It generates a file for each variant named `VariantsTestExternal_<variantName>.txt` which contains the name of the generated variant.

The External Generator API provides a commandline variable `postBuildVariant` that stores the short-name of the currently built post-build variant.

- ▶ `NG_PostfixAndMove`

A variant-aware `NGGenerator` is registered which calls an ant script which does 3 different things:

1. It creates a file for each variant named `VariantsTestNG_ant_<variantName>.txt` directly from the ant script.
2. It calls the `TemplateBasedCodeGenerator` creating a file with the name `VariantsTestNGTmpl_<variantName>.txt`. The difference to the `TemplateBasedCodeGenerator` mentioned above is, that the file name is directly calculated - you do not need to rename it.
3. It also defines an external generator to create a file which is renamed by the `move` command of ant to `VariantsTestNGExt_moved_<variantName>.txt`.



A common generator variable `postBuildVariant` exists that stores the short-name of the currently built post-build variant.

#### 5.7.9.4. Switching variants

If you switch the loadable or selectable variant, the configuration changes immediately and shows the configuration for the selected variant.

You may switch loadable or selectable variants in the EcuC module [Section 5.7.9.1, “The EcuC module”](#).

Additionally the **Fast Access** tool bar provides a fast possibility to switch selectable variants. In [Figure 5.13, “The Variant module \(Variant2 selected\)”](#) you see the currently selected selectable variant is **Variant2**.

When you select the parameter **SelectableBoolean** in the generic editor you see it is set to true. And in the **Properties** view you can see the `PostBuildVariantConditions` for this parameter:

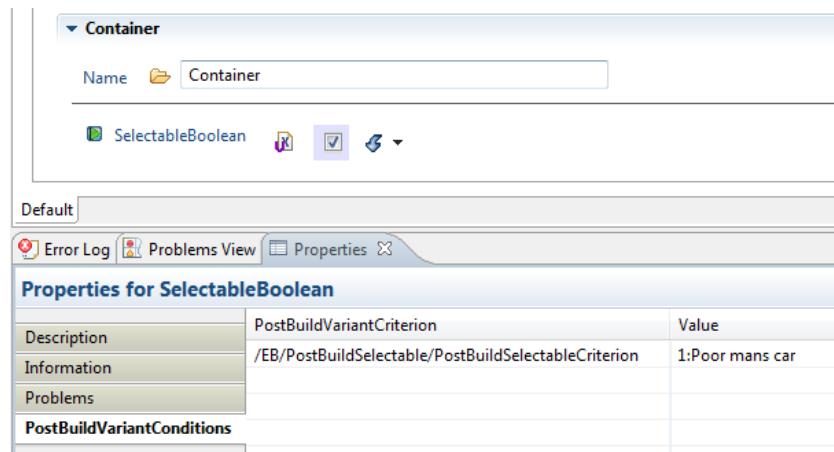
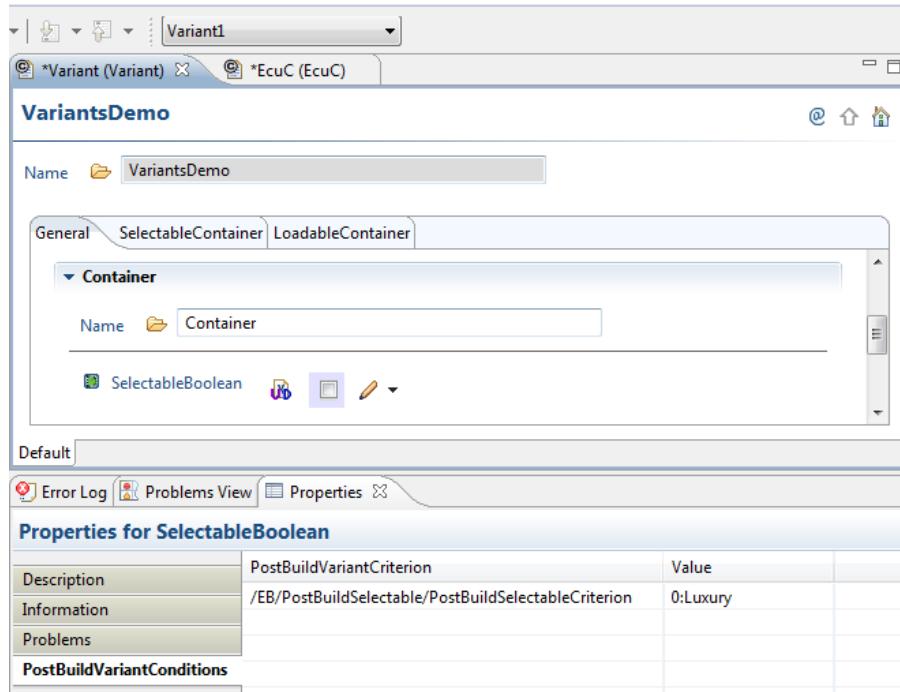


Figure 5.15. The properties for `SelectableBoolean`

You can now change the selected selectable variant to **Variant1** in the **Fast Access** tool bar.

Figure 5.16. The variant module (**Variant1** selected)

Immediately the parameter **SelectableBoolean** switches to false and in the **Properties** view it shows that the assigned criterion value is 0 which is associated with **Luxury**.

### 5.7.9.5. Managing variants

Two wizards exist below category **System** in the **Sidebar** view:

- ▶ **Edit Loadable PostBuildVariants** to create and manage loadable variants
- ▶ **Edit Selectable PostBuildVariants** to create and manage selectable variants

Both wizards consists of two tabs:

- ▶ **Predefined Variants**

Here you can manage predefined variants and refer variants to criterions.

- ▶ **Variant Criterions**

Here you can manage post-build variant criterions and edit the textual representation (**CompuMethod**) for the criterions.

When you open the **Edit Selectable PostBuildVariants** wizard from the **Sidebar** view, then you can see that the selectable predefined variant **Variant2** refers the **PostBuildSelectableCriterion** with value 1 which has the textual representation **Poor mans car**.

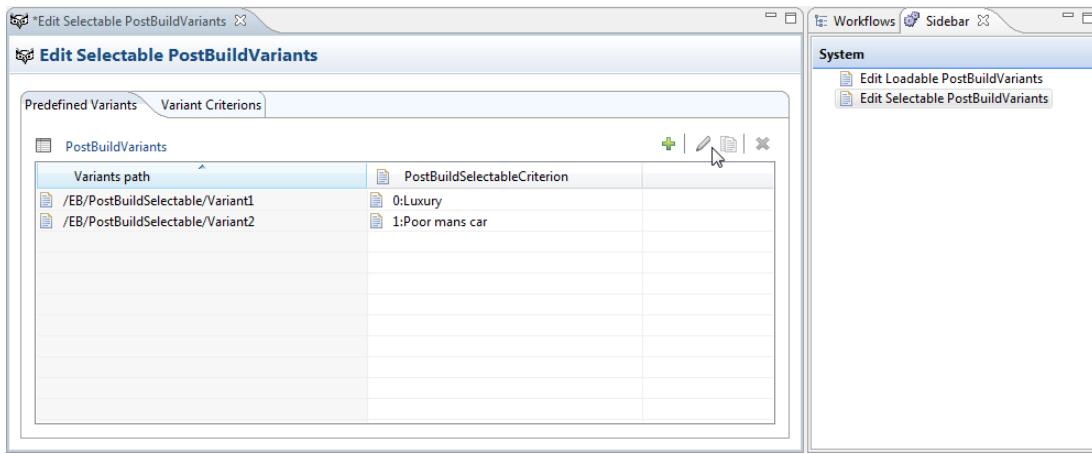


Figure 5.17. The **Predefined Variants** tab of the **Edit Selectable PostBuildVariants** wizard

When you switch to the **Variant Criterions** tab and select one **PostBuildVariantCriterion** from the left table, you can manage the textual representation for the selected criterion.

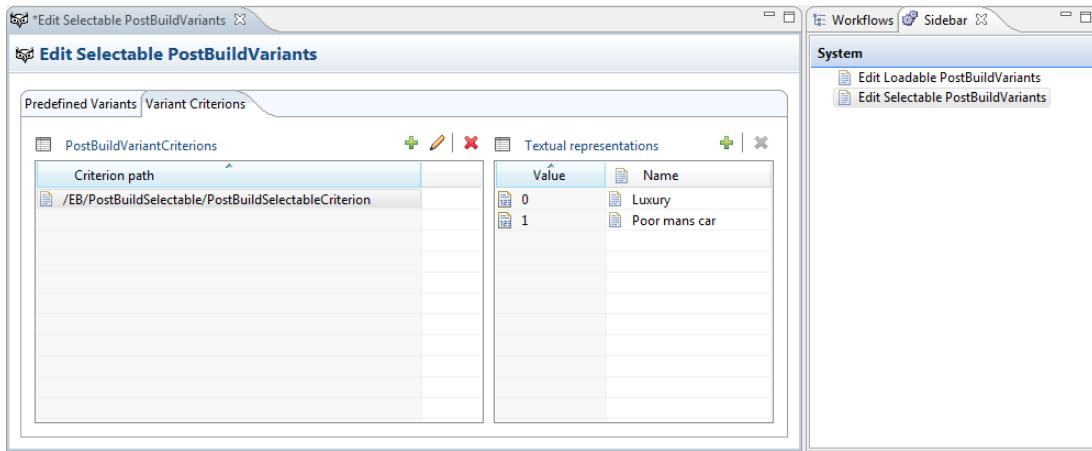


Figure 5.18. The **Variant Criterions** tab of the **Edit Selectable PostBuildVariants** wizard

## 5.8. Post-build loadable support via Multiple Configuration Containers

**NOTE****Multiple Configuration Container support is deprecated since AUTOSAR 4.2**

The Multiple Configuration Container support has been replaced by AUTOSAR 4.2 by newly introduced configuration classes. See [Section 5.7.6, “Marking elements as post-build loadable variant-aware”](#) for details about the new concept.

Nevertheless, Multiple Configuration Containers may still be used in EB tresos Studio.

EB tresos Studio has built-in support for AUTOSAR's Multiple Configuration Container (MCC) set approach to realize variants of compatible sets of configuration data for the *post-build loadable* scenario. This includes special handling of parameters and references with configuration class `PostBuild` and containers which have the attribute `postBuildChangeable` set to `true`.

For more information about the standard, see the AUTOSAR document *Specification of ECU Configuration*.

EB tresos Studio extends the AUTOSAR approach by the notion of the *current* MCC set.

For more information about this concept in EB tresos Studio, see the EB tresos Studio user's guide, section "Post-build loadable configuration using Multiple Configuration Containers".

## 5.8.1. Providing a post-build configuration management module

To enable post-build loadable configuration support of EB tresos Studio, you need to provide a contribution to the extension point `dreisoft.tresos.autosar2.api.plugin.multipleConfigurationContainerPostbuildSetup`, e.g. within some *Post-build configuration Management* module (PbcfgM) plugin.

With this additional meta-data EB tresos Studio is able to identify the current set of MCCs for referenced module configurations.

If users actually configured the PbcfgM module, EB tresos Studio uses those current MCCs for different tasks like code generation or data import in your configuration project. EB tresos Studio also applies more strict constraints regarding the module configuration data after users switched the configuration project into configuration time post-build (only).

Refer to the extension point description for more information and an example.

For further details, see the EB tresos Studio user's guide, section *Post-build loadable configuration using Multiple Configuration Containers*.

## 5.8.2. Additional VSMD rules regarding post-build enabled configuration modules



To verify the compliance of your post-build enabled Vendor Specific Module Definition (VSMD) against relevant AUTOSAR requirements and constraints, EB tresos Studio provides a special rule set named `asc:4.0.3.-RFCs`.

# 6. Configuration models

## 6.1. Concepts: The DataModel

The DataModel provides a very flexible in-memory representation of configuration data and other information. In EB tresos Studio, the DataModel is used to represent the AUTOSAR data structures, but the data contained in the DataModel goes beyond this. The model is split into three major parts:

- ▶ The *schematic-tree*, which contains the meta information about the represented data. In the AUTOSAR context, this is comparable to the module definition.
- ▶ The *data-tree*, which contains the configuration values. For each data node exists a corresponding node in the schematic-tree. In the AUTOSAR context, this is comparable to the module configuration.
- ▶ The *preferences-tree*, which stores other general information which is neither related to the data tree nor the schema tree. E.g. the importer settings of a project are stored in this part of the model. The preferences tree is no official API.

**NOTE**

**One model contains the complete ECU configuration and definition**

In EB tresos Studio, all AUTOSAR module configurations and their module definitions are located in the same model. This means that you have access to all configuration data, and even the definition, independent from which module you are accessing the model.

All nodes are stored within a tree-structure. Thus each node has one parent and as many child-nodes as desired. Only the top-level-node of the tree does not have a parent.

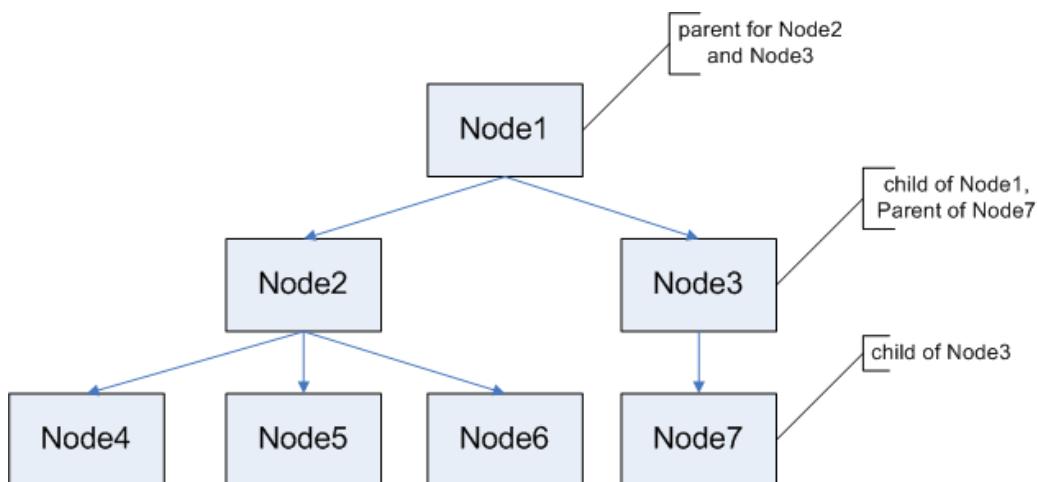


Figure 6.1. Simple Node-Tree

Configuration data is stored in a data-tree. The structure of this tree is described by a schematic-tree. The schema-tree and the data-tree always have the same structure which means that:

- ▶ Every data-node has a single schema-node which describes it.
- ▶ Each data-node has a reference to the schematic-node that describes it.
- ▶ The parent of a data-node refers to the parent of the corresponding schema-node. There never is a leap node between them. This ensures the similarity of the schematic- and data-tree-structure.

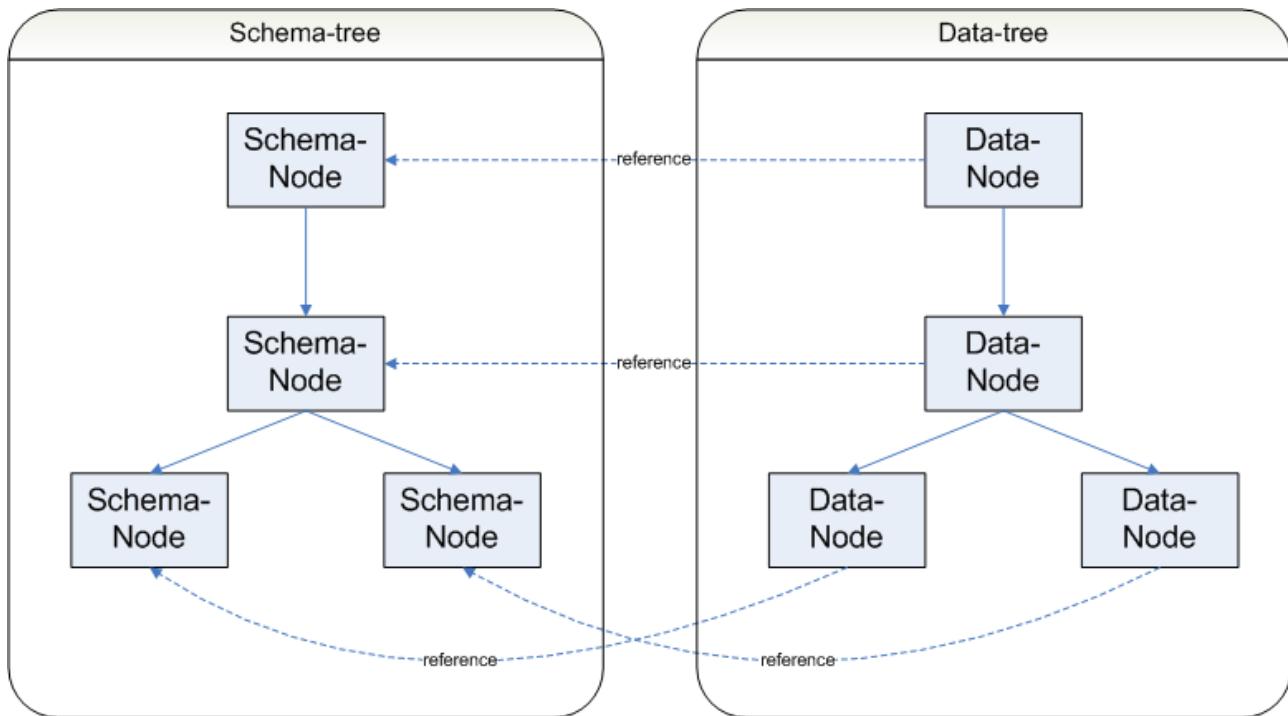


Figure 6.2. Simple schematic- and data-tree

The schematic- and data-trees are represented by different node types:

<b>Schema</b>	<b>Data</b>	<b>Description</b>
container-node	container-node	Stores a fixed list of nodes. The schematic-node contains the same amount of children as the data-node
list-node	list-node	Stores a variable list of nodes. The schematic-node always contains only a single node. The data-node contains one node for each configured (added) value.
choice-node	choice-node	Stores a fixed list of nodes. The schematic-node contains all available choices as children. The data-node contains the selected node as a child.
Boolean-node integer-node string-node float-node	variable-node	The schema-node describes the data-type and range of the data-node. The data-node stores one configuration entry. These nodes cannot contain other nodes.
reference-node	reference-node	The data-node refers to another node. These nodes cannot contain other nodes.



## 6.1.1. Schematic-tree

The schematic-tree itself does not store any configuration data. It just describes the data-tree and provides meta-information about:

- ▶ types and ranges of configuration data variables
- ▶ default-values for configuration data variables
- ▶ layout information for the visual component
- ▶ tool-tip
- ▶ online help
- ▶ etc.

Values of data-nodes are stored as strings without further type-information. Thus, the value of a data-node may only be interpreted correctly with the meta-information of the schematic-node referred.

### 6.1.1.1. Example

The four data-nodes in the following diagram hold the same string-value. With support of the schematic-node, the values get their data-type: integer, float, string and an invalid integer (attributes like RANGE will be explained in [Section 5.1, “Concepts: The XDM format”](#)).

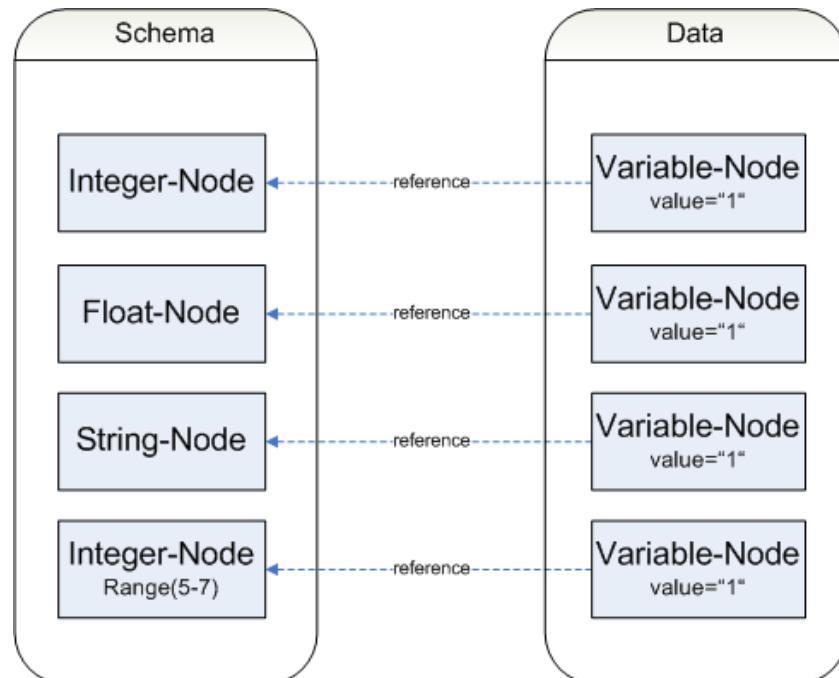


Figure 6.3. Nodes in Schema and Data

### 6.1.1.2. Features of Schema-Nodes

A schema-node permits:

- ▶ automatically instantiating corresponding configuration data-nodes
- ▶ filling in default values
- ▶ verifying entered data
- ▶ rendering dialogs which allow the user to fill in the configuration.

Multiple data-trees may refer to the same schema-tree, since the schema-tree only holds meta-information which is unique for all data-trees.

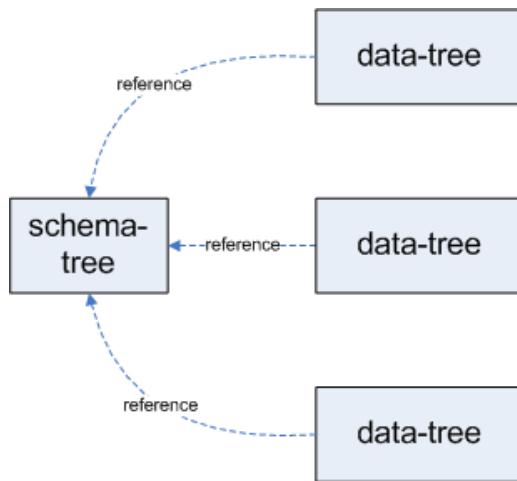


Figure 6.4. Multiple data-trees referring to one schematic-tree

## 6.1.2. Data-Tree

A data-node stores the configured value(s). For each node within the schema-tree, corresponding data-nodes can be created within the data-tree.

## 6.1.3. Node-Attributes

Each node can have several attributes. Each attribute may store multiple string values and is referred to by name.

## Nodes and Attributes

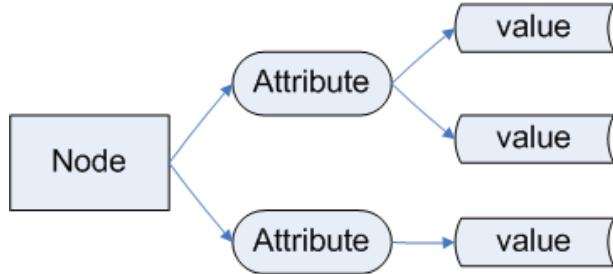


Figure 6.5. Nodes and attributes

For example, the DEFAULT-attribute holds the default-value for a data-node. The RANGE-attribute may contain several values that define ranges for a data-node.

When schema or configuration data is loaded from XDM-files, attributes are explicitly set as they are defined in the XDM-file. When an AUTOSAR configuration format is loaded, attributes are set implicitly (e.g. the RANGE-attribute will be calculated from ENUM-LITERAL-DEF).

The DataModel recognizes two types of attributes:

- ▶ Passive attributes. The values are set explicitly.
- ▶ Automatic attributes. Their values will be calculated each time the data in the tree changes as their value depends on the current configuration. Their values cannot be set explicitly.

### 6.1.4. List-Representation within the DataModel

A particular feature of a list-node is that there may be more children within the data than in the schema. Within the schema-tree, a list-node always has exactly one child. This child describes the children of the corresponding list-node within the data-tree. All list-children within the data-tree refer to the single list-child within the schema-tree.

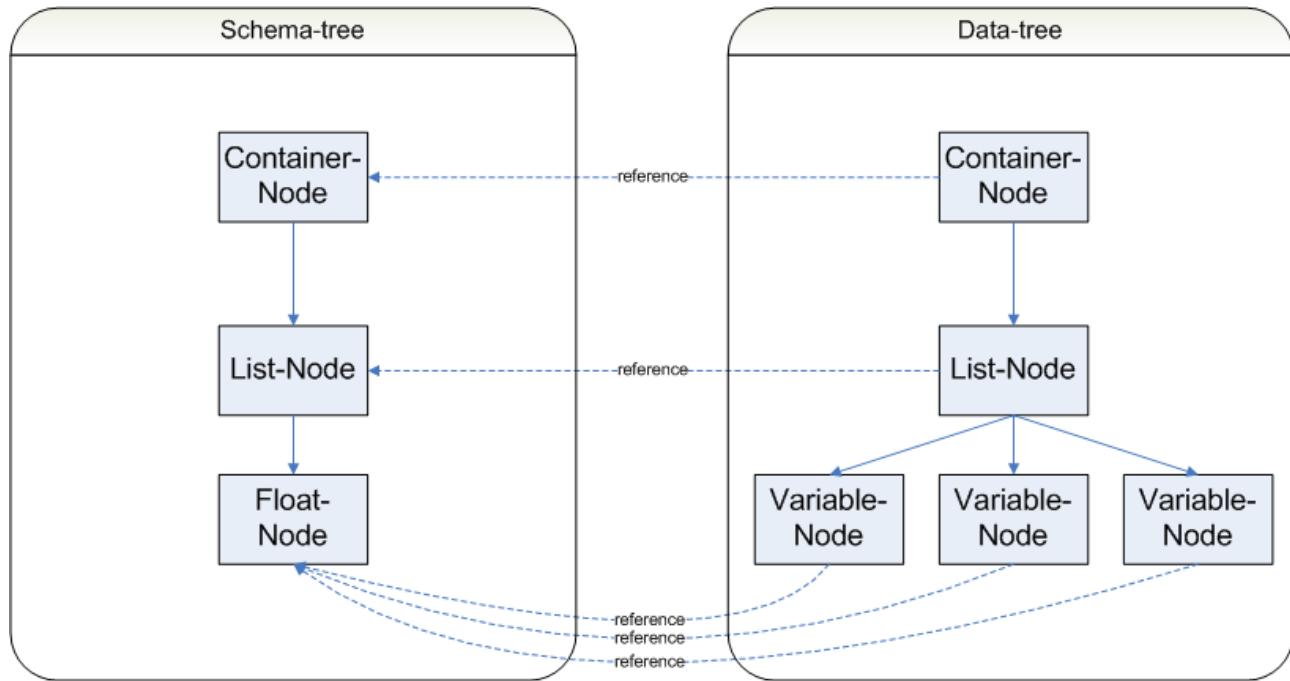


Figure 6.6. List-Representation

### 6.1.5. Choice-representation within the DataModel

A choice provides the possibility to choose a single node from of a list of sub-nodes. Within the schema-tree all available nodes are children of the choice-node. Within the data-tree the choice contains a child that represents the node which the user selected. The choice-node within the data provides the name of the chosen node as the node-value.

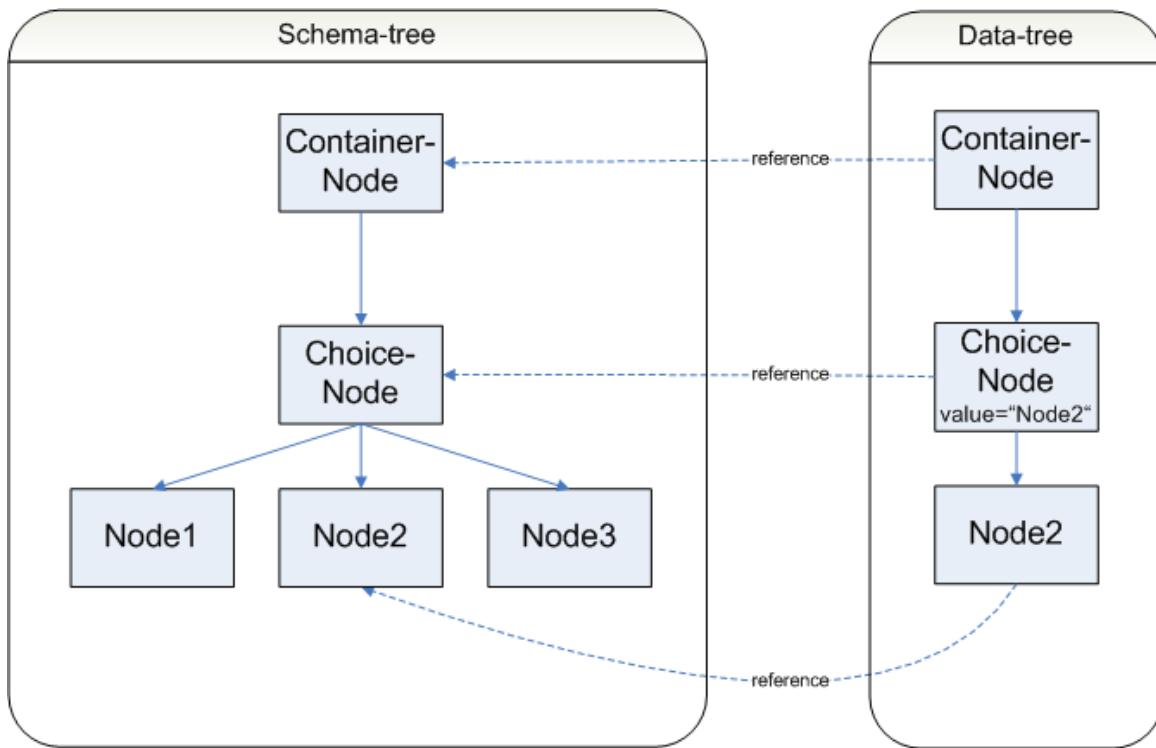


Figure 6.7. Choice-Representation

## 6.2. How to add GUI annotation in the XDM file

### 6.2.1. Purpose

The DataModel and therefore the XDM format supports adding GUI Annotations in the parameter definition to allow a more finely granulated control about how the configuration is rendered in a generic configuration editor. This chapter introduces all these constructs and describes how they are formulated in XDM.

### 6.2.2. Generic Configuration Editor Layout

This chapter details how a configuration model is laid out in the GUI. It assumes that an AUTOSAR parameter definition already forms the basis of the configuration.

The generic configuration editor reads a parameter definition and creates a respective GUI.



The generic configuration editor is composed of two parts:

1. The Editor itself
2. The Node Outline

The editor displays pages of which only one is visible at a time. The generic configurator creates a page for the `MODULE-DEF` element and one for each container that is part of a list (`UPPER-MULTIPLICITY` or `LOWER_MULTIPLICITY != 1`).

A page has a title and an entry that allows editing the name (`SHORT-NAME` of the underlying container). If the page represents the module definition, the name is not editable.



Figure 6.8. Top of page

The page contains editing elements for all configuration parameters, sub-containers and choices beneath the container represented by the page.

The page content is grouped into tabs. A `General` tab is created if there are parameters with `UPPER-MULTIPLICITY` or `LOWER_MULTIPLICITY != 1` (in this case the parameter, container or choice is represented by a list in the DataModel).

All list elements (`UPPER-MULTIPLICITY` or `LOWER_MULTIPLICITY != 1`) as well as instance references are stored on an own page which is labeled with the name of the element.

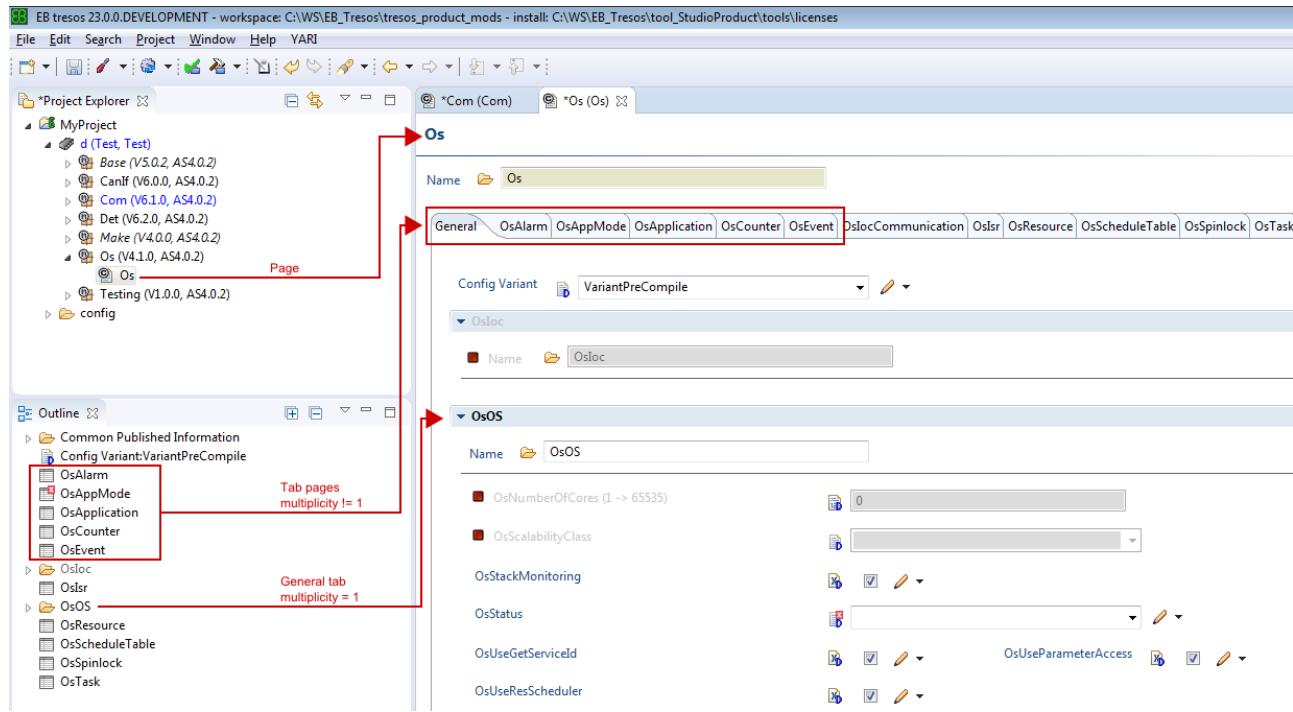


Figure 6.9. MODULE-DEF in the editor

Double clicking the index column in a table that contains containers switches the editor to a new page representing the container.

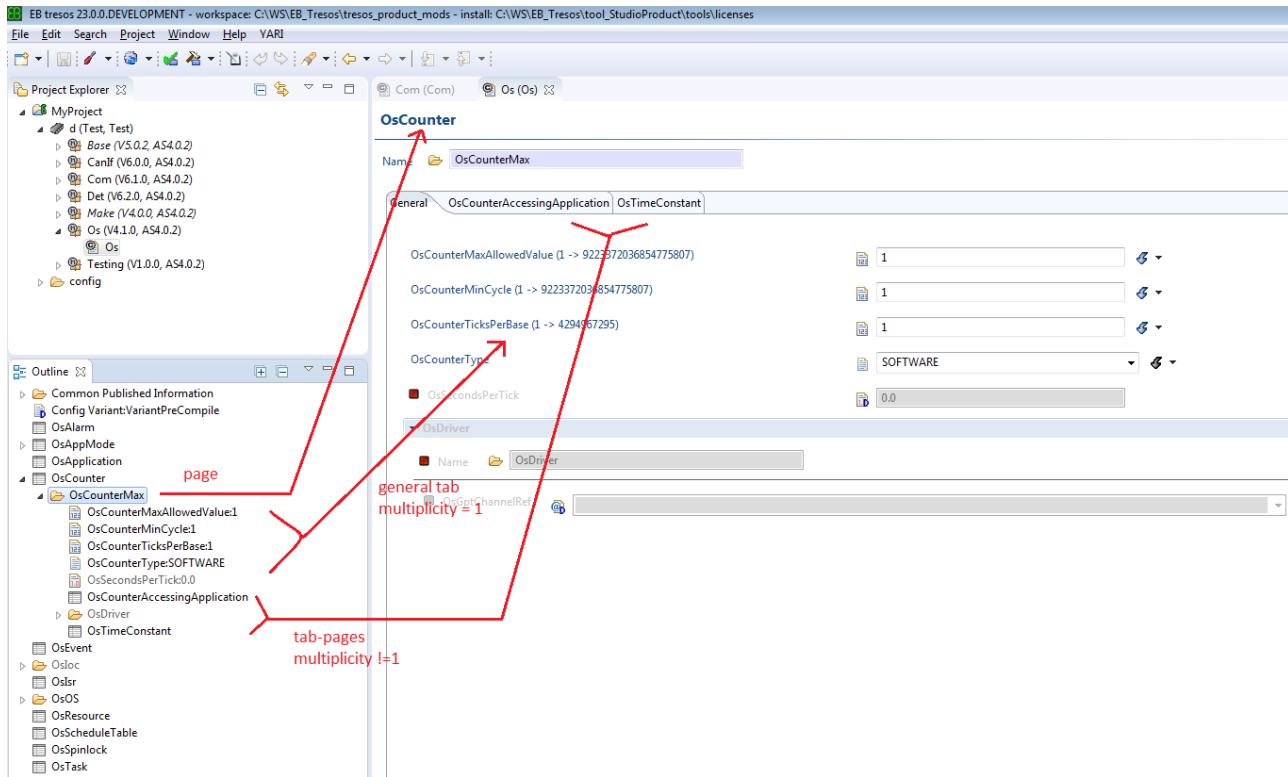


Figure 6.10. PARAM-CONF-CONTAINER-DEF in the GUI

### 6.2.3. Module GUI configuration

There are some parameters which change the GUI-representation of a whole module. Thus, these parameters are stored in the file `plugin.xml` of the module.

```

<extension point="dreisoft.tresos.launcher2.plugin.configuration"
          id="ConfigId" name="Configuration">
    <editor id="EditorId">
        <class
            class="dreisoft.tresos.launcher2.editor.GenericConfigEditor">
            <parameter name="schema" value="ASPath:/TS_T17D9M2I0R0/EcuC"/>
            <parameter name="noLinks" value="false"/>
            <parameter name="groupLinks" value="false"/>
            <parameter name="groupContainers" value="true"/>
            <parameter name="groupTables" value="true"/>
            <parameter name="optionalGeneralTab" value="true"/>
        </class>
    </editor>
</configuration>

```



Parameter	Default	Description
noTabs	false	If set to true tables will not be placed on own tabs.
noLinks	false	Do not create forward and backward hyperlinks. For each element which is not shown on the tab of its enclosing node (e.g. when the TAB-attribute is used or for tables), a hyperlink is shown between its sibling elements pointing to the tab where the element is shown. Another hyperlink is created on the tab where the element is shown which is pointing back.
groupLinks	false	Group all hyperlink elements after all configuration parameters instead of creating the hyperlinks at the position where the element occurs in the schema (between its sibling elements).
groupContainers	true	Group all containers after variables and references. Usually, each element is shown at the position where the element occurs in the schema (between its sibling elements). Setting this option to true renders containers after variables, references and tables.
groupTables	true	Group all tables after the variables and references. Usually, each element is shown at the position where the element occurs in the schema (between its sibling elements). Setting this option to true renders tables after variables and references.
optionalGeneralTab	true	Omit General Tab if only links would be placed on it.

## 6.2.4. GUI Elements

The following chapter describes the various elements in a parameter definition and how they are shown in the GUI.

The visualization of the elements in the GUI can be tailored by setting respective attributes in the XDM file. The chapter will outline all these attributes.

### 6.2.4.1. Attributes

There are several attributes which can be added to a schema-file to enrich the GUI.



	TOOLTIP	VISIBLE	DESC	EDITABLE	LABEL	FRAME	LAYOUT	TITLE	COLUMNS	COLUMN_TITLE	COLUMN_TOOLTIP	WDTH	TAB	DEFAULT_RADIX
Boolean values	x	x	x	x	x								x	
Integer values	x	x	x	x	x								x	x
Float values	x	x	x	x	x								x	x
String values	x	x	x	x	x								x	x
Enumerations	x	x	x	x	x								x	x
References	x	x	x	x	x								x	x
Container		x	x		x	x	x	x					x	
Choices	x	x	x	x	x	x	x						x	x
Lists	x	x	x	x	x				x	x	x		x	

Figure 6.11. Attributes

**NOTE**

Note that an automatic attribute can be used instead of a fixed value for all data-attributes (tag <a:da/>).

```
<a:da name="" type="XPath"
expr="..../Integer2 &lt; 20"/>
```

For schematic-attributes (tag <a:a/>) this is not possible, so for instance LABEL you can't use automatic attributes.

### 6.2.4.2. Boolean values

Each Boolean value defined within the schematic-xdm will be represented by a check box in the editor.

```
<v:var name="Boolean1" type="BOOLEAN"/>
```



Figure 6.12. Check box

Attribute	Description
TOOLTIP	<p>Used to add a tooltip to the check box.</p> <pre>&lt;a:da name="TOOLTIP" value="Example tooltip"/&gt;</pre>



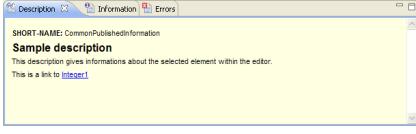
Attribute	Description
	
VISIBLE	<p>With this attribute, you can hide the check box within the editor.</p> <pre data-bbox="409 669 921 696">&lt;a:a name="VISIBLE" value="false"/&gt;</pre>
DESC	<p>Used to add a description for the Boolean value. The description is visible for the user within the "Description"-view if the check box is selected.</p> <p>As you can see in the following examples, there can be single-line, multi-line or html-description optional containing links to other configuration elements. A click on the link will show the target-element within the editor.</p> <pre data-bbox="409 1051 1151 1078">&lt;a:a name="DESC" value="The parameter depicts..."/&gt;</pre> <pre data-bbox="409 1118 754 1253">&lt;a:a name="DESC"&gt;&lt;a:v&gt;   Example description   taking multiple lines &lt;/a:v&gt;&lt;/a:a&gt;</pre> <pre data-bbox="409 1298 1246 1684">&lt;a:a name="DESC"&gt;&lt;a:v&gt;&lt;![CDATA[ &lt;html&gt;   &lt;h1&gt;Sample description&lt;/h1&gt;   This description gives informations about   the selected element within the editor.&lt;br&gt;   &lt;p&gt;     This is a link to &lt;a href="path:/AUTOSAR/TOP-LEVEL-     PACKAGES/Can/ELEMENTS/ModulePlugIn/Container/Integer1"/&gt;     Integer1&lt;/a&gt;   &lt;/p&gt; &lt;/html&gt; ] ]&gt;&lt;/a:v&gt;&lt;/a:a&gt;</pre> 

Figure 6.14. Check box Description



Attribute	Description
	<p>To create a link, you have to prefix the XPath of the target-element with <code>path:</code> and use this as the value of the <code>href</code> tag as shown in the example.</p> <p><b>NOTE</b> To get the path to a configuration element, just select it. Then, its XPath is shown in the <b>Information View</b>.</p> 
EDITABLE	<p>Used to allow the user to edit the element or not. If this attributes value is set to <code>false</code>, the check box will be greyed out.</p> <pre data-bbox="414 804 949 833">&lt;a:da name="EDITABLE" value="false"/&gt;</pre> 
READONLY	<p>Used to allow the user to edit the element or not. If this attributes value is set to "true", the check box will be greyed out and the value is reset to the default value. A warning is shown if the value is changed via other means (e.g. if the internal <code>.xdm</code> file is edited directly).</p> <pre data-bbox="414 1260 949 1289">&lt;a:da name="READONLY" value="false"/&gt;</pre> 
LABEL	<p>Used to change the label of the check box.</p> <pre data-bbox="414 1603 1006 1632">&lt;a:a name="LABEL" value="Example label"/&gt;</pre> 
TAB	<p>Via this attribute, an element can be placed on a different tab than its default.</p> <pre data-bbox="414 1931 962 1960">&lt;a:a name="TAB" value="BooleanStuff"/&gt;</pre>



Attribute	Description

Figure 6.18. Check box TAB

#### 6.2.4.3. Integer Values, Float Values, String Values, Enumerations and References

Enumerations and references are represented by a combo box. Integer values, float values, and string values are represented by a text field within the editor.

```
<v:var name="Integer1" type="INTEGER">
<v:var name="Float1" type="FLOAT">
<v:var name="String1" type="STRING">
<v:ref name="Reference1" type="REFERENCE">
<v:var name="Enumeration1" type="ENUMERATION">
```

Figure 6.19. Field elements

Attribute	Description
TOOLTIP	<p>Used to add a tooltip to the field.</p> <pre>&lt;a:da name="TOOLTIP" value="Example tooltip"/&gt;</pre>
VISIBLE	With this attribute, you can hide the field within the editor.



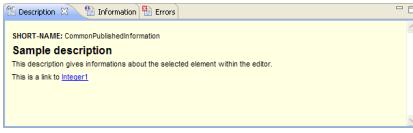
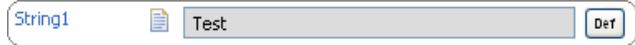
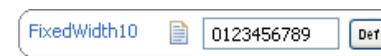
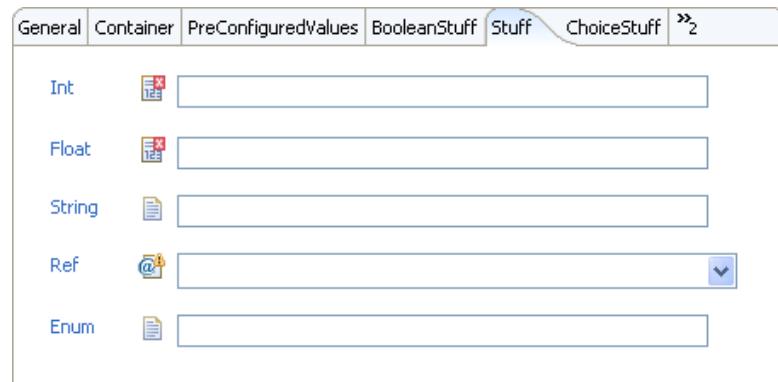
Attribute	Description
	<a:a name="VISIBLE" value="false"/>
DESC	<p>Used to add a description for the configuration element. The description is visible for the user within the <b>Description View</b> if the check box is selected.</p> <p>See <a href="#">Section 6.2.4.2, "Boolean values"</a> for details, how to create descriptions. As you can see there, you can create single-line, multi-line or html-descriptions containing links to other configuration elements.</p> 
	<p>Figure 6.21. Parameter Description</p> <p>To create a link, you have to prefix the XPath of the target-element with <code>path:</code> and use this as the value of the <code>href</code> tag as shown in the example.</p> <p><b>NOTE</b> To get the path to an configuration element, just select it. Then, its XPath is shown in the <b>Information View</b>.</p>
EDITABLE	<p>Used to allow the user to edit the element or not. If this attribute value is set to <code>false</code>, the check box will be greyed out.</p> <pre>&lt;a:da name="EDITABLE" value="false"/&gt;</pre> 
READONLY	<p>Used to allow the user to edit the element or not. If this attribute value is set to "true", the field will be greyed out and reset to its default value. A warning is shown if the value is changed via other means (e.g. if the internal <code>.xdm</code> file is edited directly).</p> <pre>&lt;a:da name="READONLY" value="true"/&gt;</pre> 

Figure 6.23. Read-only Parameter



Attribute	Description
LABEL	<p>Used to change the label of the field.</p> <pre>&lt;a:a name="LABEL" value="Example label"/&gt;</pre> 
WIDTH	<p>The width of a text field or a combo box in terms of characters.</p> <pre>&lt;a:a name="WIDTH" value="10"/&gt;</pre> 
TAB	<p>Via this attribute, an element can be placed on a different tab than its default.</p> <pre>&lt;a:a name="TAB" value="Stuff"/&gt;</pre> 
DEFAULT_RADIX	<p>Sets an initial numeric representation for the GUI element of an INTEGER node.</p> <p>Valid values for the DEFAULT_RADIX attribute are: DEC (decimal, default), HEX (hexadecimal), OCT (octal) and BIN (binary).</p> <pre>&lt;a:a name="DEFAULT_RADIX" value="HEX" /&gt;</pre>



Attribute	Description
	<p>DioPortId (0 -&gt; ...)</p> <p> 0x0 </p>

Figure 6.27. DEFAULT\_RADIX attribute for INTEGER nodes

#### 6.2.4.4. Container

A container creates a new section within the editor. You may edit the SHORT-NAME of the container.

A container should always contain other configuration elements.

```
<v:ctr name="Container" type="IDENTIFIABLE">
```

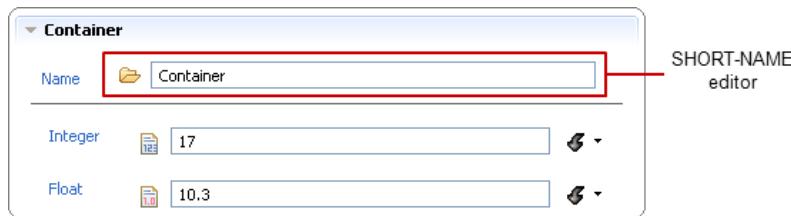


Figure 6.28. Container configuration element

Container attribute	Description
VISIBLE	<p>With this attribute, you can hide the container within the editor.</p> <pre>&lt;a:a name="VISIBLE" value="false"/&gt;</pre>
DESC	<p>Used to add a description for the configuration element. The description is visible for the user within the <b>Description View</b> if the check box is selected.</p> <p>See <a href="#">Section 6.2.4.2, “Boolean values”</a> for details, how to create descriptions. As you can see there, you can create single-line, multi-line or html-descriptions containing links to other configuration elements.</p> <p></p>

Figure 6.29. Container Description

To create a link, you have to prefix the XPath of the target-element with `path:` and use this as the value of the `href` tag as shown in the example.

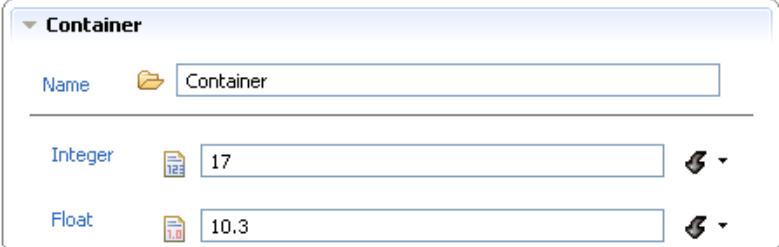
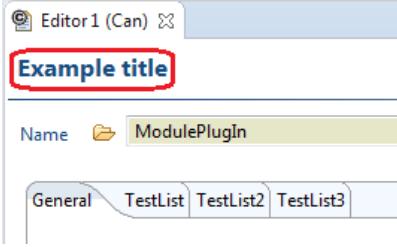
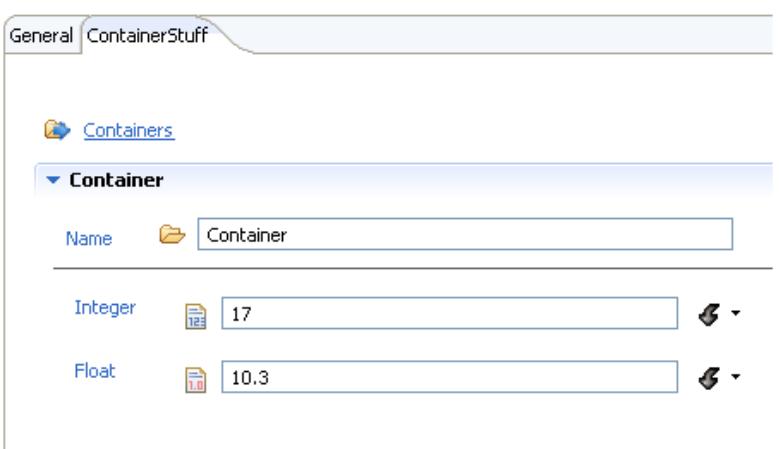


Container attribute	Description
	<p><b>NOTE</b> To get the path to an configuration element, just select it. Then, its XPath is shown in the <b>Information View</b>.</p> 
LABEL	<p>Used to change the label of the container.</p> <pre>&lt;a:a name="LABEL" value="Example label"/&gt;</pre> <div style="text-align: center;">  </div> <p>Figure 6.30. Container Label</p>
FRAME	<p>Used to change the frame around the configuration element. Possible values are:</p> <ul style="list-style-type: none"> <li>▶ <b>NONE</b> no frame. This is the default for subcontainers of choices (see <a href="#">Section 6.2.4.5, “Choices”</a>).</li> </ul> <pre>&lt;a:da name="FRAME" value="NONE"/&gt;</pre> <div style="text-align: center;">  </div> <p>Figure 6.31. FRAME with value NONE</p> <p><b>NOTE</b> You cannot edit the SHORT-NAME of the container then.</p>  <ul style="list-style-type: none"> <li>▶ <b>LINE</b> lined border with a title.</li> </ul> <pre>&lt;a:da name="FRAME" value="LINE"/&gt;</pre>



Container attribute	Description
	<p>Figure 6.32. FRAME with value LINE</p> <ul style="list-style-type: none"> <li>▶ TITLE (default) lined border with a title which you can expand and collapse when you click the title.</li> </ul> <pre>&lt;a:da name="FRAME" value="TITLE"/&gt;</pre> <p>Figure 6.33. FRAME with value TITLE</p>
LAYOUT	<p>Used to change the layout of the configuration element. Possible values are:</p> <ul style="list-style-type: none"> <li>▶ HORIZONTAL layouts the elements inside the container horizontal.</li> <li>▶ VERTICAL(default) layouts the elements inside the container vertical.</li> </ul> <pre>&lt;a:a name="LAYOUT" value="HORIZONTAL"/&gt;</pre> <p>Figure 6.34. LAYOUT with value HORIZONTAL</p>



Container attribute	Description
	 <p>Figure 6.35. LAYOUT with value VERTICAL</p>
TITLE	<p>Used to set the title of a page. You can only change the title of containers which creates a new page.</p> <pre data-bbox="412 842 1024 871">&lt;a:da name="TITLE" value="Example title"/&gt;</pre>  <p>Figure 6.36. Title of a Container</p>
TAB	<p>Via this attribute, an element can be placed on a different tab than its default.</p> <pre data-bbox="412 1370 992 1399">&lt;a:a name="TAB" value="ContainerStuff"/&gt;</pre>  <p>Figure 6.37. Container TAB</p>



#### 6.2.4.5. Choices

A choice creates a new section within the editor. You can edit the SHORT-NAME of the choice.

The choice element itself consists of a drop-down list box which is followed by selectable subcontainers. These subcontainers are displayed in separate tabs. When you change the selection of the drop-down list box, you automatically select one of the subcontainers. The title of the tab of the selected subcontainer is highlighted by bold font. The elements which the subcontainer contains are enabled then. To inspect content of the disabled tabs you may still switch to them.

```
<v:chc name="Choice">
```

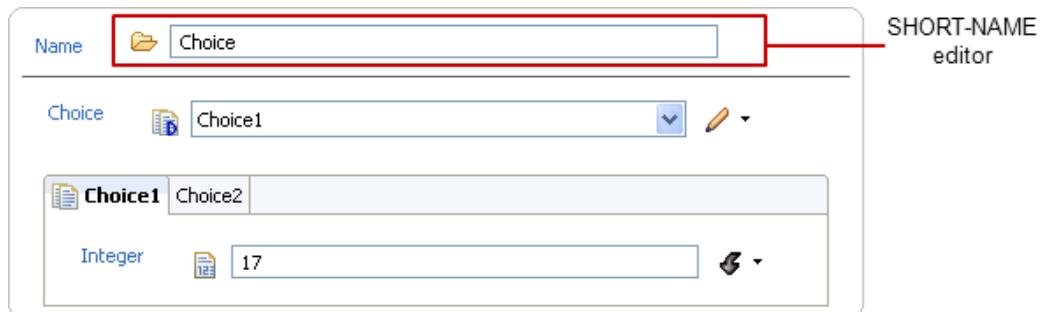
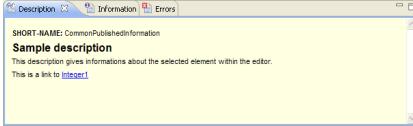
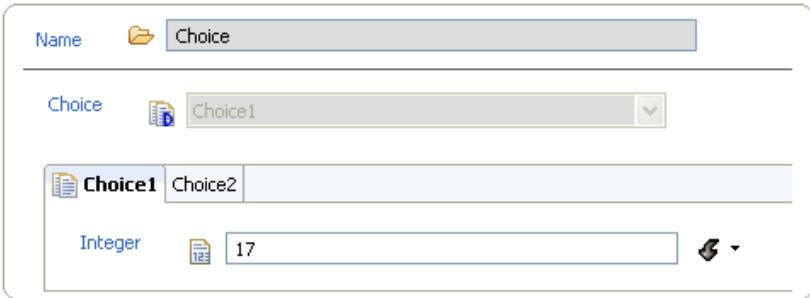


Figure 6.38. Choice configuration element

Choice attribute	Description
TOOLTIP	<p>Used to add a tooltip to the choice.</p> <pre>&lt;a:da name="TOOLTIP" value="Example Tooltip"/&gt;</pre>
VISIBLE	<p>With this attribute, you can hide the choice within the editor.</p> <pre>&lt;a:a name="VISIBLE" value="false"/&gt;</pre>



Choice attribute	Description
DESC	<p>Used to add a description for the configuration element. The description is visible for the user within the <b>Description View</b> if the choice is selected.</p> <p>See <a href="#">Section 6.2.4.2, "Boolean values"</a> for details, how to create descriptions. As you can see there, you can create single-line, multi-line or html-descriptions containing links to other configuration elements.</p> 
	<p style="text-align: center;">Figure 6.40. Description of a Choice</p> <p>To create a link, you have to prefix the XPath of the target-element with <code>path:</code> and use this as the value of the <code>href</code> tag as shown in the example.</p> <div style="border: 1px solid black; padding: 10px; margin-top: 10px;"> <p><b>NOTE</b> To get the path to an configuration element, just select it. Then, its XPath is shown in the <b>Information View</b>.</p>  </div>
EDITABLE	<p>Used to allow the user to edit the element or not. If this attributes value is set to <code>false</code>, the choice will be greyed out.</p> <pre>&lt;a:da name="EDITABLE" value="false"/&gt;</pre> 
READONLY	<p>Used to allow the user to edit the element or not. If this attributes value is set to "true", the choice will be greyed out and the value is reset to the default value. A warning is shown if the value is changed via other means (e.g. if the internal .xdm file is edited directly).</p>



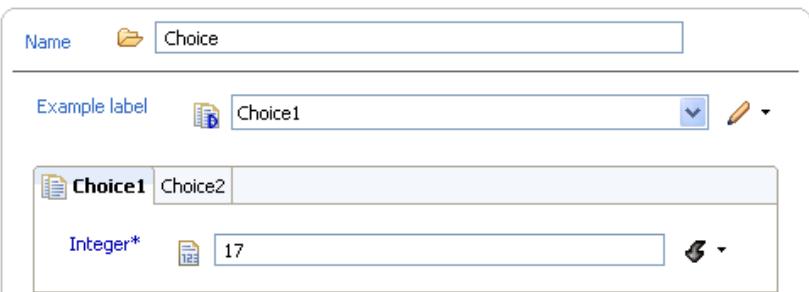
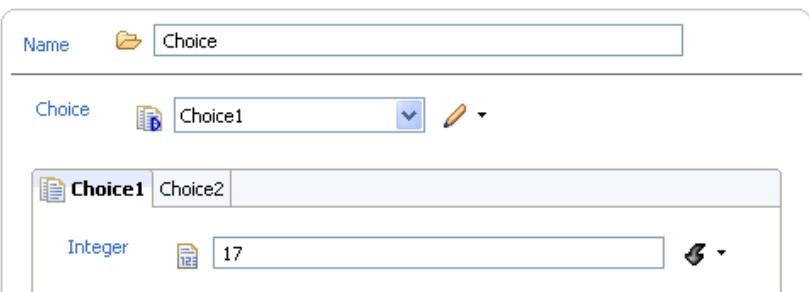
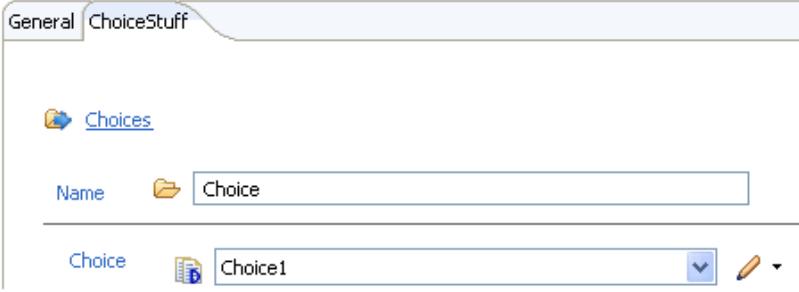
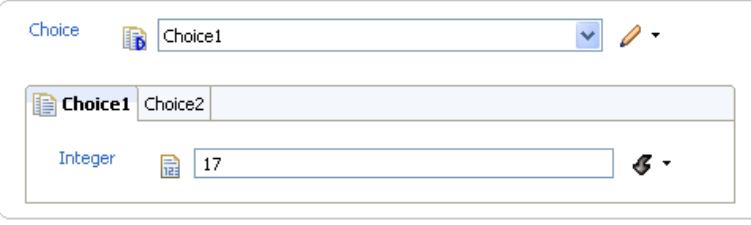
Choice attribute	Description
	<pre data-bbox="409 363 936 393">&lt;a:da name="READONLY" value="true"/&gt;</pre>  <p>The screenshot shows a configuration interface for a 'Choice' element. At the top, there is a 'Name' field containing 'Choice'. Below it is a 'Choice' field with a dropdown menu showing 'Choice1'. Underneath these are two sub-fields: 'Choice1' and 'Choice2'. At the bottom of the configuration panel is an 'Integer' field with the value '17'.</p>
	<p>Figure 6.42. Read-only choice</p> <pre data-bbox="409 857 857 887">&lt;a:a name="LABEL" value="Example label"/&gt;</pre>  <p>The screenshot shows a configuration interface for a 'Choice' element. The 'Name' field is 'Choice'. The 'Choice' field has a dropdown menu showing 'Choice1'. Below it are two sub-fields: 'Choice1' and 'Choice2'. The 'Integer' field at the bottom has a label 'Integer*' preceding its value '17'. There is also a small edit icon next to the dropdown menu.</p>
	<p>Figure 6.43. Choice Label</p> <pre data-bbox="409 1441 952 1471">&lt;a:a name="WIDTH" value="20"/&gt;</pre>  <p>The screenshot shows a configuration interface for a 'Choice' element. The 'Name' field is 'Choice'. The 'Choice' field has a dropdown menu showing 'Choice1'. Below it are two sub-fields: 'Choice1' and 'Choice2'. The 'Integer' field at the bottom has a label 'Integer' preceding its value '17'. The width of the 'Choice' field is explicitly set to 20 characters.</p>

Figure 6.44. Choice Width



Choice attribute	Description
TAB	<p>TAB on a different tab than its default.</p> <pre data-bbox="414 467 952 496">&lt;a:a name="TAB" value="ChoiceStuff"/&gt;</pre>  <p>Figure 6.45. Choice TAB</p>
FRAME	<p>Used to change the frame around the configuration element. Possible values are:</p> <ul style="list-style-type: none"> <li>▶ NONE no frame.</li> </ul> <pre data-bbox="462 1125 949 1154">&lt;a:da name="FRAME" value="NONE"/&gt;</pre>  <p>Figure 6.46. FRAME with value NONE</p> <hr/> <p><b>NOTE</b> You cannot edit the SHORT-NAME of the choice then.</p>  <ul style="list-style-type: none"> <li>▶ LINE (default) lined border with a title.</li> </ul> <pre data-bbox="462 1837 949 1866">&lt;a:da name="FRAME" value="LINE"/&gt;</pre>



Choice attribute	Description
	<p>Figure 6.47. FRAME with value LINE</p> <ul style="list-style-type: none"> <li>▶ TITLE lined border with a title which you can expand and collapse when you click the title.</li> </ul> <pre>&lt;a:da name="FRAME" value="LINE"/&gt;</pre> <p>Figure 6.48. FRAME with value TITLE</p>

#### 6.2.4.6. Lists

Each list gets its own tab-page within the editor and each configured value gets its own row within the list. Per default, the index (position within the list) of the configured element gets an own column within the list.

There exist two different types of lists:

- ▶ List with type="MAP":

```
<v:lst name="TestList" type="MAP">
<v:ctr name="ListContainer1" type="IDENTIFIABLE">
<v:var name="ListString" type="STRING"/>
<v:var name="ListInteger" type="INTEGER"/>
```



```
</v:ctr>
</v:lst>
```

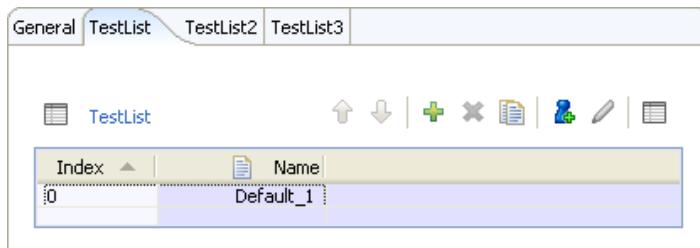


Figure 6.49. List with type="MAP"

► **List with type="":**

```
<v:lst name="TestList2" type="">
    <v:var name="ListString2" type="STRING">
        <a:da name="DEFAULT" value="test"/>
    </v:var>
</v:lst>
```

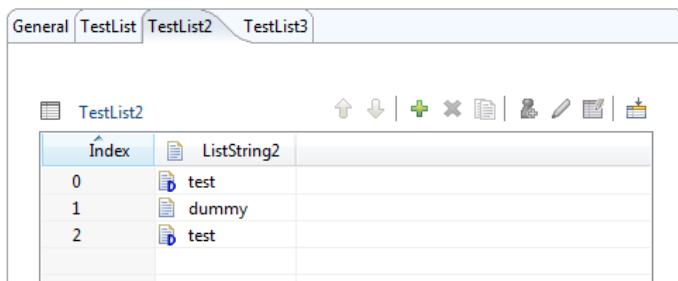
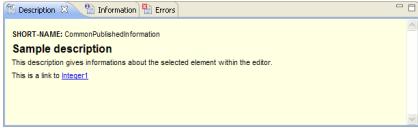
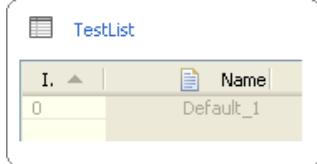


Figure 6.50. List with type=""

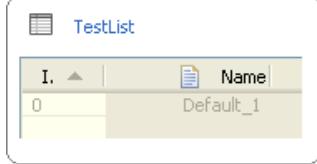
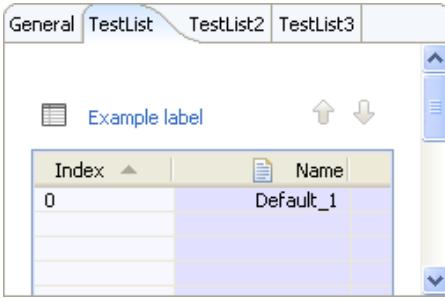
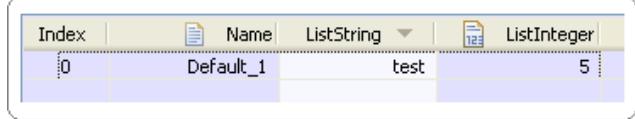
List attribute	Description
TOOLTIP	<p>Used to add a tooltip to the configuration element.</p> <pre>&lt;a:da name="TOOLTIP" value="Example tooltip"/&gt;</pre>

Figure 6.51. Tooltip inside a list

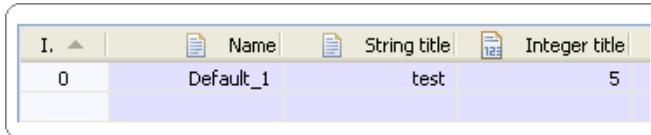


List attribute	Description
VISIBLE	<p>With this attribute, you can hide the tab-page of the list within the editor.</p> <pre>&lt;a:a name="VISIBLE" value="false"/&gt;</pre>
DESC	<p>Used to add a description for the configuration element. The description is visible for the user within the <b>Description View</b> if the list icon is clicked.</p> <p>See <a href="#">Section 6.2.4.2, “Boolean values”</a> for details, how to create descriptions. As you can see there, you can create single-line, multi-line or html-descriptions containing links to other configuration elements.</p> 
EDITABLE	<p>Used to disable the configuration list.</p> <p><b>NOTE</b> You cannot use the editor to select a list-entry of a disabled list.</p>  <p><code>&lt;a:da name="EDITABLE" value="false"/&gt;</code></p> 
READONLY	<p>You cannot use the editor to add a list-entry of a disabled list. An error is reported if you try to add a child via other means (eg. via DCtxt.Any.copy(...))</p> <p><code>&lt;a:da name="READONLY" value="true"/&gt;</code></p>

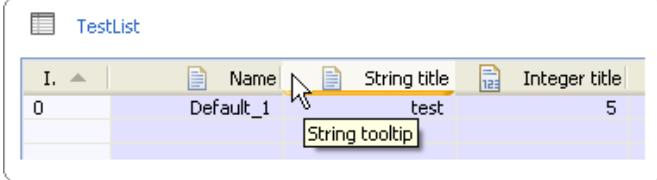


List attribute	Description
	 <p>Figure 6.54. Read-only list</p>
LABEL	<p>Used to change the label of the list element.</p> <pre>&lt;a:a name="LABEL" value="Example label"/&gt;</pre>  <p>Figure 6.55. Label of a list</p>
COLUMNS	<p>Columns of a table are defined via the <code>COLUMNS</code> attribute if the list contains containers or choices (only for lists of type="MAP"). The values of the attribute can reference parameters or references (that have <code>UPPER-MULTIPLICITY = LOWER_MULTIPLICITY == 1</code>) relative to the container or choice inside the list via an XPath.</p> <pre>&lt;a:a name="COLUMNS"&gt;   &lt;a:v&gt;ListString&lt;/a:v&gt;   &lt;a:v&gt;SubContainer/ListOfInteger&lt;/a:v&gt; &lt;/a:a&gt;</pre>  <p>Figure 6.56. List Columns</p>
COLUMN_TITLE	<p>Used to set the title of a custom column within a table.</p> <pre>&lt;v:lst name="TestList" type="MAP"&gt;   &lt;a:a name="COLUMNS"&gt;</pre>



List attribute	Description
	<pre> &lt;a:v&gt;ListString&lt;/a:v&gt; &lt;a:v&gt;ListInteger&lt;/a:v&gt; &lt;/a:a&gt;  &lt;v:ctr name="ListContainer1" type="IDENTIFIABLE"&gt;   &lt;v:var name="ListString" type="STRING"&gt;     &lt;a:a name="COLUMN_TITLE" value="String title"/&gt;     &lt;a:a name="COLUMN_TOOLTIP" value="String tooltip"/&gt;   &lt;/v:var&gt;   &lt;v:var name="ListInteger" type="INTEGER"&gt;     &lt;a:a name="COLUMN_TITLE" value="Integer title"/&gt;     &lt;a:a name="COLUMN_TOOLTIP" value="Integer tooltip"/&gt;   &lt;/v:var&gt; &lt;/v:ctr&gt; &lt;/v:lst&gt;</pre> 
COLUMN_TOOLTIP	<p>Used to add a tooltip for a custom column within a table.</p> <pre> &lt;v:lst name="TestList" type="MAP"&gt;   &lt;a:a name="COLUMNS"&gt;     &lt;a:v&gt;ListString&lt;/a:v&gt;     &lt;a:v&gt;ListInteger&lt;/a:v&gt;   &lt;/a:a&gt;    &lt;v:ctr name="ListContainer1" type="IDENTIFIABLE"&gt;     &lt;v:var name="ListString" type="STRING"&gt;       &lt;a:a name="COLUMN_TITLE" value="String title"/&gt;       &lt;a:a name="COLUMN_TOOLTIP" value="String tooltip"/&gt;     &lt;/v:var&gt;     &lt;v:var name="ListInteger" type="INTEGER"&gt;       &lt;a:a name="COLUMN_TITLE" value="Integer title"/&gt;       &lt;a:a name="COLUMN_TOOLTIP" value="Integer tooltip"/&gt;     &lt;/v:var&gt;   &lt;/v:ctr&gt; &lt;/v:lst&gt;</pre>



List attribute	Description
	 <p>Figure 6.58. Column Tooltip</p>
TAB	<p>Via this attribute, an element can be placed on a different tab than its default.</p> <pre>&lt;a:a name="TAB" value="ListStuff"/&gt;</pre>
ROWS	<p>Via this attribute, the number of visible list rows can be configured.</p> <pre>&lt;v:lst name="TestList" type="MAP"&gt;   &lt;a:a name="ROWS" value="3"/&gt; &lt;/v:lst&gt;</pre>

#### 6.2.4.7. Tab assignment

As soon as 100 elements are rendered to the same tab page, the next container is rendered to a new tab. This default behavior ensures that the editor is capable of displaying any number of elements in a concise way and with good performance.

You can manually change this default. To do so, change the file `$TRESOS_BASE/configuration/default-Preferences.properties`, edit the entry `maxElementsPerTab` and restart EB tresos Studio.

## 6.3. XPath API

### 6.3.1. Purpose

XPath is a W3C-Standard and is described in several tutorials available on the web:

[W3C-Standard: http://www.w3.org/TR/xpath](http://www.w3.org/TR/xpath)



[https://www.w3schools.com/xml/xpath\\_intro.asp](https://www.w3schools.com/xml/xpath_intro.asp)

<http://www.zvon.org/xxl/XPathTutorial/General/examples.html>

The basis of how to use XPath within the DataModel is described in this chapter. XPath can be used to navigate within the DataModel, select nodes and retrieve the values of nodes. To retrieve the value of a node, the node must be selected with a corresponding XPath path-expression.

### 6.3.2. Prerequisites

To use the **XPath Console**, select the preference **Show actions 'XPath console' and 'Codetemplate console' in Outline view**. For instructions about how to change preferences, see the EB tresos Studio user's guide, chapter Setting Developer features preferences.

You can open the **XPath Console** from the context-menu of a node within the **Outline view** by choosing "XPath console". The selected node will be used as context-node for all executed XPath expressions.

The path of the context-node is displayed above the text field. The part of the path that points to the MODULE-CONFIGURATION is displayed as hyperlink. To set the MODULE-CONFIGURATION as context-node, click on the hyperlink.

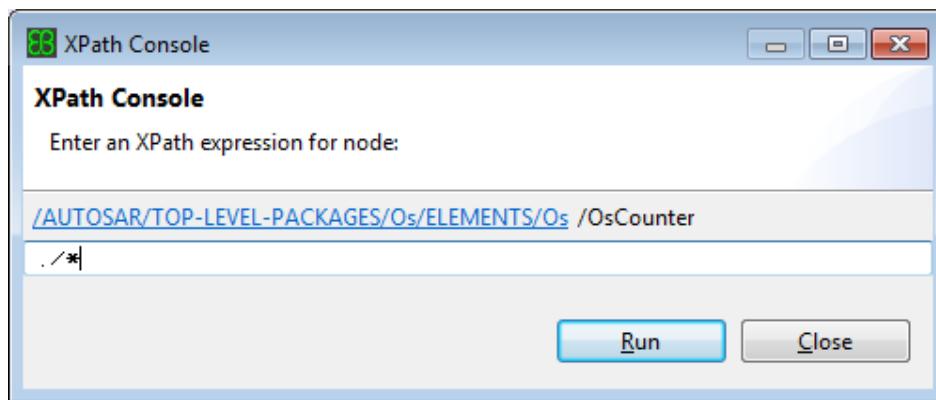


Figure 6.59. XPath Console

Type in any XPath expression and click **Run** to execute it. The result will be shown within the **Search View**.

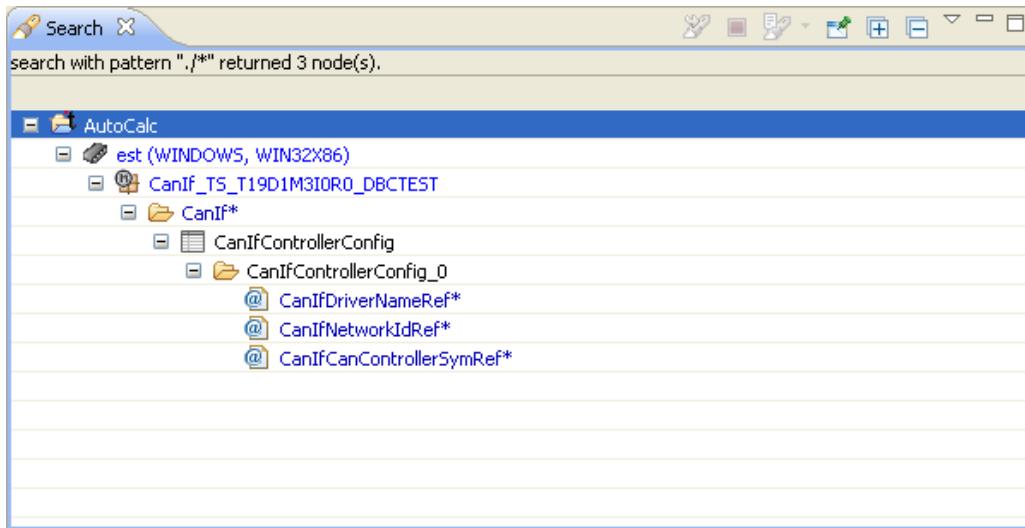


Figure 6.60. XPath Console Result

### 6.3.3. Concepts

#### 6.3.3.1. Conventions

XPath-functions are described in the following form: *return-value* function-name([*parameter*[?]] [,*parameter*[?]]/\*) Example for function 'name' with an optional parameter *node-set* and a *string*-return-value: *string* name(*node-set*?) An *italic* font is used to indicate XPath-data-types (return-value and all parameters). If a parameter is followed by a "?", this parameter is optional - otherwise it is mandatory. A "\*" means, that any number of parameters can be given.

#### 6.3.3.2. Data-Types within XPath

##### **XPath defines the following Data-Types:**

1. *node*: a node can be an element, an attribute or simply text
2. *node-set*: an unordered collection of nodes without duplicates.
3. *Boolean*: may be true or false
4. *number*: a floating-point number
5. *string*: a sequence of characters



6. *object*: this is no separate data-type but a generalization of the data-types listed above. It is therefore used if any data-type can be returned or given as an argument.

XPath defines rules to automatically convert an expression value to the appropriate type. For example if a function expects a string as parameter, but got a number, the given number will automatically be converted into a string.

---

**WARNING**

**Use the XPath engine with care**

Some expressions may return unexpected results due to the loose typing of the XPath language. There are implementation errors known in the engine that EB tresos Studio uses. To see further information about this see chapter [Section 6.3.4.1, “Pitfalls when using XPath”](#).

It is recommended to use the engine with care and to thoroughly test any expressions you use.

---

### 6.3.3.3. Additional Data-Types for the DataModel

For the use within the DataModel, new data-types are introduced in XPath. For example 'int' is derived from the XPath-data-type 'number' and ensures that there is no floating-point-value.

`xpath`

... a *string* representing a valid XPath- expression

`refnode`

... a *node* representing a reference-node which refers to another node

`numlist`

... a *node-set* of *numbers* or *strings* which represent *numbers*

`intlist`

... a *node-set* of *numbers* without floating-point-value

`strlist`

... a *node-set* of strings

`int`

... a *number* without floating-point-value

`hex`

... a *string* representing a hexadecimal value. The *string* always starts with '0x'

`oct`

... a *string* representing an octal value. The *string* always starts with '\0'

`bin`

... a *string* representing a binary value. The *string* always starts with 'b'

`dec`

... a *string* representing a decimal value.

#### 6.3.3.4. DataModel-Nodes and their XPath-Value

The following table describes which DataModel-node will return which value.

Node-type	Value in schema	Value in data
Container	empty <i>string</i>	the default from schema or empty <i>string</i>
Variable	empty <i>string</i>	the configured value or the default from the schema
List	empty <i>string</i>	the default from schema or empty <i>string</i>
Choice	empty <i>string</i>	the name of the selected child-node or the default from the schema
Reference	empty <i>string</i>	the XPath to the referenced node or the default from the schema

#### 6.3.3.5. Example-Tree

The tree in the picture below is used for all following examples.

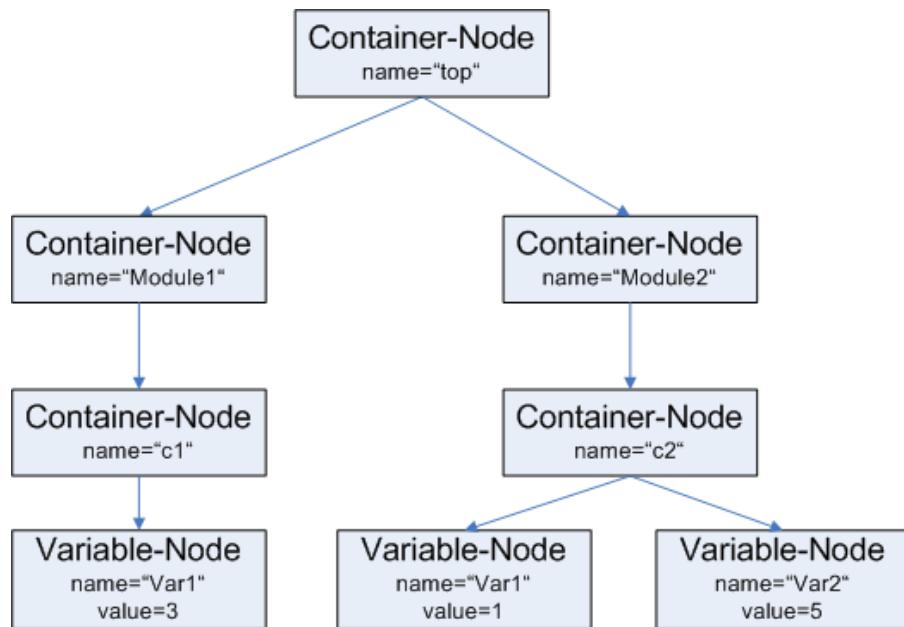


Figure 6.61. Example-Tree

#### 6.3.4. API



### 6.3.4.1. Pitfalls when using XPath

When using XPath (within EB tresos Studio or elsewhere), you must be aware of some anomalies. In the following, some of those are presented and workarounds or recommended uses are given.

#### 6.3.4.1.1. Non-existing nodes

It is possible to select nodes by XPath expressions that do not exist, or which are optional and are disabled in the current DataModel. If you use such nodes in XPath expressions, which do calculations based on the value of the nodes, the XPath engine implicitly assumes values for the nodes even if the nodes do not exist, instead of resulting in an error.

In case of optional nodes in the DataModel, that are disabled in the actual module configuration, this may lead to unexpected results, e.g. when using these expressions within the template-based code generator.

##### 6.3.4.1.1.1. Expressions

During evaluation, XPath uses implicit datatypes and values for non-existing nodes as required by the operands and the operator-type of the expression. Have a look at the following examples, where `NonExistingNode` is a non-existing node:

- ▶ In arithmetical operations, i.e. if the expression uses one of the operators: `+`, `-`, `div`, `*`, or `mod`, a value of `0.0` is taken for non-existing nodes.

For example, the expression `NonExistingNode + 3.0` evaluates to `3.0`.

- ▶ Equality comparisons (using the `=` operator), in which at least one of the operands is a non-existing node, always evaluate to `false`, for example:

- ▶ `NonExistingNode = 'somevalue'`
- ▶ `NonExistingNode = ''`
- ▶ `NonExistingNode = 0`
- ▶ `0.0 = NonExistingNode`
- ▶ `NonExistingNode = false()`
- ▶ `NonExistingNode = true()`
- ▶ `NonExistingNode = NonExistingNode`

Less surprisingly if the expression is an inequality check (using the `!=` operator), the result is always `true`:

- ▶ `NonExistingNode != 'somevalue'`
- ▶ `NonExistingNode != ''`
- ▶ `NonExistingNode != 0`



- ▶ `0.0 != NonExistingNode`
- ▶ `NonExistingNode != false()`
- ▶ `NonExistingNode != true()`
- ▶ `NonExistingNode != NonExistingNode`
- ▶ In a relational expression, using i.e. one of the operators `<`, `<=`, `>` or `>=`, the value `0.0` is taken for non-existing nodes.

For example the expressions `NonExistingNode < 0` and `NonExistingNode > 0` both return `false`. However, the expressions `NonExistingNode <= 0` and `NonExistingNode >= 0` both return `true`.

Although the non-existing node behaves as being equal to `0.0`, Keep in mind that the equality comparison does return `false`, as stated in the second example of this list.

- ▶ For the unary operator `-`, the value `0.0` is used in evaluation, and `-NonExistingNode` evaluates to `-0.0`.
- ▶ If the expression uses the union operator `|` for concatenating node sets, an empty node-set is taken.

#### 6.3.4.1.1.2. Functions

In functions defined by the XPath standard if a non-existing node is passed as an argument, the value is converted to the datatype required by the function (e.g. to `0.0`, the empty node-set, or to `false`). Functions defined by the XPath standard can be recognized by an empty namespace or the namespace `fn`.

In functions supplied by EB tresos Studio as an extension of the XPath engine if a non-existing node is passed as an argument to the function, an error message is shown. However, this only applies to nodes which are being provided arguments without any additional calculations. If an expression is passed as an argument, it is evaluated first.

You can use the XPath standard functions `node:empty(...)` or `node:exists(...)` or `node:accessible(...)` to explicitly test for the existence of nodes. As you can see from the examples above, relying on the implicit behaviour for non-existing nodes can otherwise lead to unexpected results.

#### 6.3.4.1.2. Type conversion with Boolean expressions

Due to the loosely typed nature of the XPath language, some expressions involving Boolean values might return unexpected results. The following items have been reported by users to be unexpected. If known, a best practice for formulating alternate, better comprehensible expressions is provided.

- ▶ Any string other than the empty string evaluates to `true` in equality comparisons (using the `=` operator). For example, `true() = 'false'` evaluates to `true`. Any node or variable value is handled as a string, therefore the following expressions are examples of discouraged use:

▶ `$var1 = true()`



► `./myBool = true()`

Use the following expressions instead, which do string comparisons:

► `text:tolower($var1) = 'true'`

► `text:tolower(./myBool) = 'true'`

- Comparing an empty node set to any Boolean value always evaluates to false (same as with non-existing nodes).

When comparing a non-empty node set to a Boolean value, the result is true if at least one of the nodes compares to the given Boolean value successfully.

- Less surprisingly, any number other than `0.0` evaluates to `true` in equality comparisons. For example, `true() = 0.1` will evaluate to `true`.

#### 6.3.4.1.3. Implementation errors in the handling of Boolean values

The XPath engine does not handle Boolean values correctly in some calculations and comparisons. Other operands than the ones listed in the examples below may also be affected if you use Boolean values. The behavior does not comply to the XPath specification.

- Calculations are affected. For example, calling `true() * 3.0` yields `0.0`, although `true() = 1.0` returns `true`. On the other hand, `false() = 0.0` holds `true`, but `false() * 3.0` yields `3.0`.
- Affected comparisons: `false() = 0.0` returns `true`, but at the same time `false() > 0.0` and `false() < 1.1` return `true`. However, `false() < 2.0` does not return `true`.

##### TIP



##### Mitigating problems with Boolean values

If you use Boolean values in XPath expressions, verify their correctness by checking the return values of the expressions in the **XPath Console** dialog or in the **Codetemplate console** dialog. To open these dialogs, select **Show actions 'XPath console' and 'Codetemplate console'** in **Outline view** in the EB tresos Studio preferences. The **XPath Console** dialog and the **Codetemplate console** dialog are then available in the **Node Outline** view as context menu entries of the nodes.

#### 6.3.4.1.4. Strings and numbers

Although XPath's internal representation for node and variable values uses strings, it is often safe to use them directly in relational or equality comparisons as well as in arithmetic operations, as long as they *actually do* contain numbers. There are still some things to consider though, for example:

- When comparing two strings containing numbers, the strings are compared, not the numbers. i.e. will `'23.0' = '23'` return `false`. Explicitly applying the XPath function `number` to at least one of the strings will help in that case, i.e. `'23.0' = number('23')`.



- ▶ When comparing a number with string, that is not a number (referred to `NaN` in the following) the result will always be `false`. For instance '`0x0b`' = `11` evaluates to `false`.
- ▶ If an arithmetic expression contains an `Nan`-String, the result will also be `Nan`, i.e. '`0x0b`' \* `2.0` will result in `Nan`.

Using number strings in XPath predicates (`[<expression>]`) will most likely not provide the expected result. Remember that the predicate expression is evaluated with each node of a given node set, with the context node set to each node one after another, filtering the set down to all "matching" nodes. Whether the node remains in the resulting node set depends on the result type of the predicate expression:

- ▶ When the result of the predicate expression is of type Boolean, the node remains in the set if the result value is `true`.
- ▶ When the result of the predicate expression is of type number, the node remains in the set if the position matches (Hints: counting starts with 1 here, the order depends on the XPath axis of the applied node-test).
- ▶ Otherwise the XPath function `boolean` is applied to the result of the predicate expression and the node again remains in the set if `true`.

That said, any string other than the empty one will result in not filtering anything. If for instance the value of a variable, which is always of type string, should be taken as an index for addressing one node of a node set, you must explicitly convert the string into a number. An example of using such expressions within a code generator template is:

```
[!FOR "i" = "1" TO "3"!]
[!"node:name(./*[${i}]")!] // wrong (always returns the name of the first
child node)
[!"node:name(./*[ number(${i}) ])"]! // ok
[!ENDFOR!]
```

#### 6.3.4.1.5. Integers

XPath handles all number literals as double-precision (64-bit) *floating points* as defined by the IEEE 754-1985 standard.

This leads to a loss of precision for signed integers that require more than 53 bits. These integers are for example:

- ▶ smaller than `-9007199254740992`, or
- ▶ greater than `9007199254740991`.

The additional functions provided by EB tresos Studio (see [Section 6.3.4.5.5, “Additional functions provided by the XPath Engine”](#)) can however handle signed integers with up to 64 bits (i.e. from `-9223372036854775808` to



9223372036854775807) without a loss of precision. In order to make use of this, make sure that XPath does not evaluate the numbers before. You can achieve this if you provide the integer parameters to those functions as:

- ▶ *string literals* (e.g. `num:i('9223372036854775807')`, or
- ▶ node values from the DataModel (e.g. `num:i(./myIntegerNode)`), or
- ▶ the result of another *custom* XPath function, like one of the additional functions provided by EB tresos Studio (e.g. `num:min(num:i(./*))`).

Be aware that as soon as you use any XPath operators like `+` (add), `=` (equality comparison), or the built-in XPath number functions (see [Section 6.3.4.5.4, “Number-functions”](#)), the 53 bits limitation applies again.

Some examples:

Expression	Rationale
Expression: -9223372036854775808  Result: -9.223372036854776E18	XPath interprets the value as a double precision float. The 53 bit precision limit applies.
Expression: num:i(-9223372036854775807)  Result: -9.223372036854776E18	XPath interprets the value and passes it on as a double precision float to the <code>num:i</code> function. The 53 bit precision limit applies.
Expression: num:i('-9223372036854775807')  Result: -9223372036854775807	XPath does not interpret the value as a number but provides it to the <code>num:i</code> function as a string. The 53 bit precision limit does not apply.
Expression: num:i('-9223372036854775807') + 1  Result: -9.223372036854776E18	XPath interprets the result of <code>num:i</code> before it passes it on to the <code>+</code> operator as a double precision float. The 53 bit precision limit applies.
Expression: -9223372036854775808 > num:i(./myIntNode)  Node-value: -9223372036854775807  Result: false	XPath interprets the result of <code>num:i</code> before it passes it on to the <code>&gt;</code> operator as a double precision float. The 53 bit precision limit applies.
Expression: num:inttohex(9223372036854775806)  Result: 0x7fffffffffffff	XPath interprets the value and passes it on as a double precision float to the <code>bit:getbit</code> function. The 53 bit precision limit applies.
Expression: num:inttohex('9223372036854775806')  Result: 0x7fffffffffffffe	XPath does not interpret the value as a number but provides it to the <code>num:inttohex</code> function as a string. The 53 bit precision limit does not apply.



Expression	Rationale
Expression: num:inttohex(./myIntNode, 0)  Node-value: 9223372036854775806  Result: 0x7fffffffffffffe	The value is not interpreted, but directly read from the data model. The 53 bit precision limit does not apply.
Expression: bit:getbit(9223372036854775806, 0)  Result: true	XPath interprets the value and passes it on as a double precision float to the <code>bit:getbit</code> function. The 53 bit precision limit applies.
Expression: bit:getbit(./myIntNode, 0)  Node-value: 9223372036854775806  Result: false	The value is not interpreted, but directly read from the data model. The 53 bit precision limit does not apply.

#### 6.3.4.1.6. Target node filter

To evaluate the Xpath nested comparison expression `node:ref(.) = node:ref(node:current())`, which is executed on the target node, will always returns all the nodes. This happens because the method which is used for comparison of the nodes will compare the literal value behind the nodes. Since the literal value is NULL for both of the nodes (system model nodes), this method will return true.

As workaround, use expression `node:value(.) = node:value(node:current())` to filter the target nodes, as here `node:value` function on a reference data node returns the short name path of the target node and the short name paths of the target nodes are now used for comparison.

#### 6.3.4.1.7. Conclusion

Most of the time it is safe to be explicit, e.g. by testing for a probably non-existing node first i.e. with the function `node:exists` before using its value in an expression or function argument.

However, there are other traps in the XPath functions. For example `boolean('false')` evaluate to `true`, and `number('0x0b')` will still be not a number, but fortunately EB tresos Studio provides additional XPath functions for converting integers from other radix into the decimal representation.

#### 6.3.4.2. XPath expressions

To select a node, absolute or relative paths can be used with the following expressions:

Expression	Description
/	Selects from the root node



Expression	Description	
	Example	Result
	/top/Module1	Selects the node 'Module1'
//	Selects all nodes in the document from the current node that match the selection no matter where they are.	
	<b>NOTE</b> 	<b>Performance</b> Using the expression // can result in memory and performance problems. Prefer to use absolute and relative path expressions rather than //. Due to the fact that the expression // is not local to your module, potential performance problems might even not be discovered in unit or module testing.
	<b>NOTE</b> 	<b>Do not use the expression //*</b> Using the expression //* selects <i>all</i> nodes from the data model. This also contains internal EB tresos Studio data (see chapter <a href="#">Section 6.1, "Concepts: The DataModel"</a> ) which is not relevant for the AUTOSAR models. In most cases this is not what you want and can also result in big performance and memory problems.
	Example	Result
	//Var1	Selects both(!) nodes with name 'Var1'
.	Selects the current node	
	Example	Result
	/top/Module1/.	Selects the node 'Module1'
..	Selects the parent of the current node	
	Example	Result
	/top/Module1/c1/..	Selects the node 'Module1'
*	Selects all child-nodes of the current path	
	Example	Result
	/top/*	Selects the nodes 'Module1' and 'Module2'
@name	Holds the name of the node (for use in predicates - see chapter <a href="#">Section 6.3.4.3, "XPath predicates"</a> )	



Expression	Description	
	Example	Result
	//*[@name='Var1']	Selects both(!) nodes with name 'Var1'
@type	Holds the type of the node (for use in predicates - see chapter <a href="#">Section 6.3.4.3, "XPath predicates"</a> )	
	Example	Result
	//*[@type='BOOLEAN']	Selects all nodes with type Boolean
@index	Holds the index (within its parents children-list) of the node (for use in predicates - see chapter <a href="#">Section 6.3.4.3, "XPath predicates"</a> ). Unlike 'position', the index starts at 0 (zero). This predicate does not return the index of an element within a list - it explicitly returns the position of an element within its parents children-list. Therefore it cannot be used for an expression like split('200 600 1200')[@index=2].	
	Example	Result
	//*[@index>0]	Selects node 'Module2' and 'Var2'
@POSTBUILD-FIXED	Fetches the information whether an <code>postBuildChangeable</code> container (by schema) is actually fixed (i.e. non post-build changeable) during post-build configuration time (a string value; for use in predicates - see chapter <a href="#">Section 6.3.4.3, "XPath predicates"</a> ).  Only applicable during post-build configuration time, and if a post-build configuration management module is configured in the project. For more information about post-build handling in EB tresos Studio, see <a href="#">Section 5.8, "Post-build loadable support via Multiple Configuration Containers"</a> .	
	Example	Result
	//*[@POSTBUILD-FIXED!=""]	Selects nodes which are fixed (i.e. not changeable) during post-build configuration time.

### 6.3.4.3. XPath predicates

Predicates are used to find a specific node or a node that contains a specific value. Predicates are always in square brackets.

Syntax	Description
<code>node/*[position]</code>	Selects the child of the node with the given position. Note, that you cannot use a generator-variable here directly. Instead, you have to use the following notation:
<code>node-set/*[position]</code>	<code>node/*[position() = \$var]</code>



Syntax	Description	
	Example	Result
	/top/Module2/c2/*[1]	node 'Var1'
node/*[condition]	Selects the child(ren) of the node/node-set which fulfill the given condition	
node-set/*[condition]	Example	Result
	/top/Module2/c2/*[last()]	node 'Var2'
	/top/Module2/c2/*[last()-1]	node 'Var1'
	/top/Module2/c2/*[position()=2]	node 'Var2'
	//*[@>2]	nodes 'Var1' and 'Var2'

#### 6.3.4.4. XPath operations

Below is a list of the operators that can be used in XPath expressions:

Operator	Description	
	Combines two node-sets	
	Example	Result
	/top/Module1   //Module2	nodes 'Module1' and 'Module2'
+	Addition	
	Example	Result
	//Var2 + 7	12.0
-	Subtraction	
	Example	Result
	//Var2 - 1	4.0
*	Multiplication	
	Example	Result
	//Var2 * 2	10.0
div	Division	
	Example	Result
	//Var2 div 1	5.0



Operator	Description	
=	Equal	
	<b>Example</b>	<b>Result</b>
	//Var2 = 5	true
!=	Not equal	
	<b>Example</b>	<b>Result</b>
	//Var2 != 5	false
<	Less than	
	<b>Example</b>	<b>Result</b>
	//Var2 < 5	false
<=	Less than or equal to	
	<b>Example</b>	<b>Result</b>
	//Var2 <= 5	true
>	Greater than	
	<b>Example</b>	<b>Result</b>
	//Var2 > 5	false
>=	Greater than or equal to	
	<b>Example</b>	<b>Result</b>
	//Var2 >= 5	true
or	or	
	<b>Example</b>	<b>Result</b>
	//Var2 = 7 or /top/Module1/c1/Var1 = 3	true
and	and	
	<b>Example</b>	<b>Result</b>
	//Var2 = 7 and /top/Module1/c1/Var1 = 3	false
mod	Modulus (div remainder)	



Operator	Description	
	Example	Result
	//Var2 mod 2	1 . 0

### 6.3.4.5. XPath functions

XPath contains a small set of build-in-functions. The most useful ones are listed in the following tables.

#### 6.3.4.5.1. Node-Set functions

Function	Description	
<i>number</i> last()	Returns the position-number of the last child of the context-node	
	Example	Result
	//c2/*[last()]	node 'Var2'
<i>number</i> position()	Returns the position- <i>number</i> of the context-node	
	Example	Result
	//c2/*[position() > 0]	All child-nodes of 'c2': 'Var1' and 'Var2'
<i>number</i> count( <i>node-set</i> )	Returns the <i>number</i> of nodes within the given <i>node-set</i>  Hint: use <i>num:i</i> to get integer-values.	
	Example	Result
	count(//*)	8 . 0
	count( //Module2/*)	1 . 0
<i>string</i> name( <i>node-set</i> )	Returns the name of the first context-node.  If the node has no name, the position of the node within it's parent's child-list containing two underlines in front of the node's position number is returned, e.g.: ' <u>__</u> 2'.	



Function	Description				
	<p><b>WARNING Deviation from the W3C XPath specification</b></p>  <p>The XPath function <code>name</code> does not work as specified by the XPath specification from W3C but is implemented the following way: If no parameter is given, the current context node is returned instead of its name. Depending on where you use this expression, this may lead to different behavior.</p> <p>For example if you call <code>name()</code> on a String parameter in a code template, you get the parameter value. If you call <code>name(.)</code> you get the parameter name.</p> <p>You can see the differences in the two function calls on a concrete configuration if you execute the following code as code template:</p> <pre>[ !SELECT "///*"! ] *[ !"name()"!] * vs. * [ !"name(.)"!] * [ !ENDSELECT! ]</pre> <p>This deviation from the specification is currently kept for backward-compatibility reasons. It may be changed in the future. To avoid problems in the future, always specify a parameter to the function. We recommend you to use <code>name(.)</code> instead of <code>name()</code>.</p>				
	<table border="1"> <thead> <tr> <th>Example</th> <th>Result</th> </tr> </thead> <tbody> <tr> <td><code>name("//c2/*")</code></td> <td>'Var1'</td> </tr> </tbody> </table>	Example	Result	<code>name("//c2/*")</code>	'Var1'
Example	Result				
<code>name("//c2/*")</code>	'Var1'				

#### 6.3.4.5.2. String-functions

Function	Description				
<code>string(string)</code>	Converts the given <i>object</i> into a <i>string</i>				
	<table border="1"> <thead> <tr> <th>Example</th> <th>Result</th> </tr> </thead> <tbody> <tr> <td><code>string(5)</code></td> <td>5</td> </tr> </tbody> </table>	Example	Result	<code>string(5)</code>	5
Example	Result				
<code>string(5)</code>	5				
<code>string concat( string, string* )</code>	Concatenates the given <i>strings</i>				
	<table border="1"> <thead> <tr> <th>Example</th> <th>Result</th> </tr> </thead> <tbody> <tr> <td><code>concat('a', 'b', 'c')</code></td> <td>abc</td> </tr> </tbody> </table>	Example	Result	<code>concat('a', 'b', 'c')</code>	abc
Example	Result				
<code>concat('a', 'b', 'c')</code>	abc				



Function	Description					
<code>boolean starts-with( string, string )</code>	Returns true if the first argument starts with the second argument					
	<table border="1"> <thead> <tr> <th>Example</th> <th>Result</th> </tr> </thead> <tbody> <tr> <td><code>starts-with('abctest', 'abc')</code></td> <td><code>true</code></td> </tr> </tbody> </table>	Example	Result	<code>starts-with('abctest', 'abc')</code>	<code>true</code>	
Example	Result					
<code>starts-with('abctest', 'abc')</code>	<code>true</code>					
<code>Boolean contains( string, string )</code>	Returns true if the first argument contains the second argument					
	<table border="1"> <thead> <tr> <th>Example</th> <th>Result</th> </tr> </thead> <tbody> <tr> <td><code>contains('abcde', 'bc')</code></td> <td><code>true</code></td> </tr> </tbody> </table>	Example	Result	<code>contains('abcde', 'bc')</code>	<code>true</code>	
Example	Result					
<code>contains('abcde', 'bc')</code>	<code>true</code>					
<code>string substring-before( string, string )</code>	Returns the substring of the first argument <i>string</i> that precedes the first occurrence of the second argument <i>string</i> with the first argument <i>string</i> , or the empty string if the first argument <i>string</i> does not contain the second argument <i>string</i>					
	<table border="1"> <thead> <tr> <th>Example</th> <th>Result</th> </tr> </thead> <tbody> <tr> <td><code>substring-before( '1999/04/01', '/' )</code></td> <td><code>1999</code></td> </tr> </tbody> </table>	Example	Result	<code>substring-before( '1999/04/01', '/' )</code>	<code>1999</code>	
Example	Result					
<code>substring-before( '1999/04/01', '/' )</code>	<code>1999</code>					
<code>string substring-after(string, string)</code>	Returns the <i>substring</i> of the first argument <i>string</i> that follows the first occurrence of the second argument <i>string</i> in the first argument <i>string</i> , or the empty string if the first argument <i>string</i> does not contain the second argument <i>string</i>					
	<table border="1"> <thead> <tr> <th>Example</th> <th>Result</th> </tr> </thead> <tbody> <tr> <td><code>substring-after('1999/04/01', '/')</code></td> <td><code>04/01</code></td> </tr> </tbody> </table>	Example	Result	<code>substring-after('1999/04/01', '/')</code>	<code>04/01</code>	
Example	Result					
<code>substring-after('1999/04/01', '/')</code>	<code>04/01</code>					

Function	Description							
<code>string substring( string, number, number? )</code>	Returns the <i>substring</i> of the first argument starting at the position specified in the second argument with the length specified in the third argument. If the third argument is not specified, the substring starts at the position specified in the second argument and continues until the end of the <i>string</i>							
	<table border="1"> <thead> <tr> <th>Example</th> <th>Result</th> </tr> </thead> <tbody> <tr> <td><code>substring( '12345', 2, 3)</code></td> <td><code>234</code></td> </tr> <tr> <td><code>substring( '12345', 2)</code></td> <td><code>2345</code></td> </tr> </tbody> </table>	Example	Result	<code>substring( '12345', 2, 3)</code>	<code>234</code>	<code>substring( '12345', 2)</code>	<code>2345</code>	
Example	Result							
<code>substring( '12345', 2, 3)</code>	<code>234</code>							
<code>substring( '12345', 2)</code>	<code>2345</code>							



Function	Description							
	<b>WARNING</b> 	<b>Incorrect results for start position 0</b> The W3C XPath specification does not cover the case if the function <code>substring</code> is called with a start position of 0. The implementation in EB tresos Studio will correct the start position to 1, but due to an implementation error the result in EB tresos Studio is still wrong. In EB tresos Studio the result string will always be shorter than the specified length argument if a start position less or equal than 0 is used. It is recommended not to use 0 as argument at all. In order to avoid incorrect results, a warning message is being logged to the error log every time the function is executed with a start position of 0 and a length argument.						
<code>number string-length( <i>string?</i> )</code>	<p>Returns the <i>number</i> of characters in the <i>string</i>. If the argument is not specified, the length of the value of the context-node will be returned.</p> <table border="1"> <thead> <tr> <th>Example</th><th>Result</th></tr> </thead> <tbody> <tr> <td><code>string-length( '12345' )</code></td><td>5.0</td></tr> </tbody> </table>		Example	Result	<code>string-length( '12345' )</code>	5.0		
Example	Result							
<code>string-length( '12345' )</code>	5.0							
<code>string normalize-space( <i>string?</i> )</code>	<p>Returns the argument <i>string</i> with whitespace normalized by stripping leading and trailing whitespace and replacing sequences of whitespace characters by a single space. If the argument is omitted, it defaults to the <i>string</i>-value of the context-node.</p> <table border="1"> <thead> <tr> <th>Example</th><th>Result</th></tr> </thead> <tbody> <tr> <td><code>normalize-space( ' 12 345 ' )</code></td><td>12 345</td></tr> </tbody> </table>		Example	Result	<code>normalize-space( ' 12 345 ' )</code>	12 345		
Example	Result							
<code>normalize-space( ' 12 345 ' )</code>	12 345							
<code>string translate( <i>string</i>, <i>string</i>, <i>string</i> )</code>	<p>Returns the first argument <i>string</i> with occurrences of characters in the second argument <i>string</i> replaced by the character at the corresponding position in the third argument <i>string</i></p> <table border="1"> <thead> <tr> <th>Example</th><th>Result</th></tr> </thead> <tbody> <tr> <td><code>translate( 'bar', 'abc', 'ABC' )</code></td><td>BAr</td></tr> <tr> <td><code>translate( '-aaa---', 'abc-', 'ABC' )</code></td><td>AAA</td></tr> </tbody> </table>		Example	Result	<code>translate( 'bar', 'abc', 'ABC' )</code>	BAr	<code>translate( '-aaa---', 'abc-', 'ABC' )</code>	AAA
Example	Result							
<code>translate( 'bar', 'abc', 'ABC' )</code>	BAr							
<code>translate( '-aaa---', 'abc-', 'ABC' )</code>	AAA							



### 6.3.4.5.3. Boolean-functions

Function	Description								
<code>Boolean Boolean(object)</code>	<p>Converts it's argument to a <i>Boolean</i> as follows:</p> <ul style="list-style-type: none"> <li>▶ a <i>number</i> is true if it is neither positive or negative zero nor NaN</li> <li>▶ a <i>node-set</i> is true if it is non-empty</li> <li>▶ a <i>string</i> is true if its length is non-zero</li> </ul>								
	<p><b>NOTE</b> <b>Boolean value of reference nodes</b></p>  <p>Since reference nodes have always the path of the reference as value, they are never evaluated to the empty string and therefore, calling the <i>Boolean</i> function on a reference node will never return <code>false</code>.</p>								
	<table border="1"> <thead> <tr> <th>Example</th><th>Result</th></tr> </thead> <tbody> <tr> <td><code>boolean(5)</code></td><td><code>true</code></td></tr> <tr> <td><code>boolean(0)</code></td><td><code>false</code></td></tr> <tr> <td><code>boolean("")</code></td><td><code>false</code></td></tr> </tbody> </table>	Example	Result	<code>boolean(5)</code>	<code>true</code>	<code>boolean(0)</code>	<code>false</code>	<code>boolean("")</code>	<code>false</code>
Example	Result								
<code>boolean(5)</code>	<code>true</code>								
<code>boolean(0)</code>	<code>false</code>								
<code>boolean("")</code>	<code>false</code>								
<code>Boolean not(Boolean)</code>	<p>Returns <code>true</code> if its argument is <code>false</code>, and <code>false</code> otherwise.</p> <table border="1"> <thead> <tr> <th>Example</th><th>Result</th></tr> </thead> <tbody> <tr> <td><code>not(true())</code></td><td><code>false</code></td></tr> </tbody> </table>	Example	Result	<code>not(true())</code>	<code>false</code>				
Example	Result								
<code>not(true())</code>	<code>false</code>								
	<p><b>NOTE</b> <b>Nodes with string value "false"</b></p>  <p>The values of nodes within the data model are all interpreted as string values. This means that the expression <code>not( x )</code> will yield <code>false</code> if the configuration value of node <code>x</code> is <code>"false"</code>. The reason for this are the type conversion rules of the XPath engine, where a string with value <code>"false"</code> will be converted to Boolean <code>true</code> because it is not the empty string.</p>								
<code>Boolean true()</code>	<p>Returns <code>true</code></p> <table border="1"> <thead> <tr> <th>Example</th><th>Result</th></tr> </thead> <tbody> <tr> <td><code>true()</code></td><td><code>true</code></td></tr> </tbody> </table>	Example	Result	<code>true()</code>	<code>true</code>				
Example	Result								
<code>true()</code>	<code>true</code>								
<code>Boolean false()</code>	<p>Returns <code>false</code></p>								



Function	Description	
	Example	Result
	false()	false
Boolean lang(string)	Not supported	

#### 6.3.4.5.4. Number-functions

Function	Description					
number number( object? )	<p>Converts its argument to a <i>number</i> as follows:</p> <ul style="list-style-type: none"> <li>▶ a <i>string</i> that consists of optional whitespace followed by an optional minus sign, followed by a <i>number</i>, followed by whitespace is converted into a number that is nearest to the mathematical value represented by the string;</li> <li>▶ <i>Boolean</i> true is converted to 1</li> <li>▶ <i>Boolean</i> false is converted to 0</li> <li>▶ a <i>node-set</i> is converted to a <i>string</i> in the same way a call to the string-function would convert it. It is then converted in the same way as a <i>string</i> argument</li> </ul>					
	<table border="1"> <thead> <tr> <th>Example</th> <th>Result</th> </tr> </thead> <tbody> <tr> <td>number('5')</td> <td>5 . 0</td> </tr> </tbody> </table>		Example	Result	number('5')	5 . 0
Example	Result					
number('5')	5 . 0					
number sum(node-set)	Returns the sum of each <i>node</i> in the argument <i>node-set</i> .					
	<table border="1"> <thead> <tr> <th>Example</th> <th>Result</th> </tr> </thead> <tbody> <tr> <td>sum( /*[position() &gt; 0] )</td> <td>9 . 0</td> </tr> </tbody> </table>		Example	Result	sum( /*[position() > 0] )	9 . 0
Example	Result					
sum( /*[position() > 0] )	9 . 0					
number floor(number)	Returns the largest (closest to positive infinity) <i>number</i> that is not greater than the argument and that is an <i>integer</i> .					
	<table border="1"> <thead> <tr> <th>Example</th> <th>Result</th> </tr> </thead> <tbody> <tr> <td>floor(5.3)</td> <td>5 . 0</td> </tr> </tbody> </table>		Example	Result	floor(5.3)	5 . 0
Example	Result					
floor(5.3)	5 . 0					
number ceiling( number )	Returns the smallest (closest to negative infinity) <i>number</i> that is not less than the argument and that is an <i>integer</i> .					
	<table border="1"> <thead> <tr> <th>Example</th> <th>Result</th> </tr> </thead> <tbody> <tr> <td>ceiling(5.3)</td> <td>6 . 0</td> </tr> </tbody> </table>		Example	Result	ceiling(5.3)	6 . 0
Example	Result					
ceiling(5.3)	6 . 0					



Function	Description	
<i>number</i> round( <i>number</i> )	Returns the <i>number</i> that is closest to the argument and that is an <i>integer</i> . If the argument is <i>Nan</i> , the result is <i>0</i> .	
Example	Result	
round(1.1)	1.0	
round(1.5)	2.0	
round(1.9)	2.0	

#### 6.3.4.5.5. Additional functions provided by the XPath Engine

The following functions are added to provide additional functionality especially for accessing the data-structure within the DataModel. Hint: The default prefix for the function namespace is fn: and is therefore omitted in the previous tables. Actually it does not matter, whether you write 'fn:round(3.5)' or 'round(3.5)'. In the following table, other function-namespaces are used and thus cannot be omitted.

##### 6.3.4.5.5.1. Node functions

Function	Description	
<i>node</i> node:current()	Returns the current <i>node</i> .	
Example	Result	
node:current()	current node	
<i>node</i> node:islast()	Returns true if the context-node is the last element in its surrounding list.  Note: This does NOT mean, that it will return true for the last node of a LOOP or SELECT!	
Example	Result	
node:islast()	true	
<i>string</i> node:name( <i>node</i> )	Returns the name of the given <i>node</i> . Note that this is not necessarily the SHORT-NAME of the node that you can retrieve when you use <i>as:name</i>	
Example	Result	
node:name( //Var2)	Var2	



Function	Description							
<code>node node:ref(refnode) node</code> <code>node:ref(xpath)</code>	Returns the <i>node</i> which is referenced by the given <i>node</i> (or target- <i>node</i> ) of the <i>xpath-path</i> .							
	<table border="1"> <thead> <tr> <th>Example</th> <th>Result</th> </tr> </thead> <tbody> <tr> <td><code>node:ref('/myNode')</code></td> <td>The node <i>myNode</i>.</td> </tr> <tr> <td><code>node:ref(/myReference)</code></td> <td>The target-node of node <i>myReference</i>.</td> </tr> </tbody> </table>	Example	Result	<code>node:ref('/myNode')</code>	The node <i>myNode</i> .	<code>node:ref(/myReference)</code>	The target-node of node <i>myReference</i> .	
Example	Result							
<code>node:ref('/myNode')</code>	The node <i>myNode</i> .							
<code>node:ref(/myReference)</code>	The target-node of node <i>myReference</i> .							
<code>nodeset node:refs(nodeset)</code> <code>nodeset node:refs(xpath)</code>	Returns the <i>nodes</i> which are referenced by the given <i>nodes</i> (or target- <i>nodes</i> of the <i>xpath-path</i> ). Note that an <i>xpath-path</i> (parameter surrounded by apostrophes) never returns multiple nodes. To give in multiple nodes, an <i>xpath-expression</i> must be used (not surrounded by apostrophes).							
	<table border="1"> <thead> <tr> <th>Example</th> <th>Result</th> </tr> </thead> <tbody> <tr> <td><code>node:refs(/top/Module2/c2/*)</code></td> <td>First example returns <i>Var1</i> and <i>Var2</i> (because its an <i>Xpath-expression</i>).</td> </tr> <tr> <td><code>node:refs('/top/Module2/c2/*')</code></td> <td>Second example only returns <i>Var1</i> (because its an <i>Xpath-path</i>).</td> </tr> </tbody> </table>	Example	Result	<code>node:refs(/top/Module2/c2/*)</code>	First example returns <i>Var1</i> and <i>Var2</i> (because its an <i>Xpath-expression</i> ).	<code>node:refs('/top/Module2/c2/*')</code>	Second example only returns <i>Var1</i> (because its an <i>Xpath-path</i> ).	
Example	Result							
<code>node:refs(/top/Module2/c2/*)</code>	First example returns <i>Var1</i> and <i>Var2</i> (because its an <i>Xpath-expression</i> ).							
<code>node:refs('/top/Module2/c2/*')</code>	Second example only returns <i>Var1</i> (because its an <i>Xpath-path</i> ).							
<code>node:value(node)</code>	Returns the value of the <i>node</i>							
	<table border="1"> <thead> <tr> <th>Example</th> <th>Result</th> </tr> </thead> <tbody> <tr> <td><code>node:value( //Var2)</code></td> <td>5</td> </tr> </tbody> </table>	Example	Result	<code>node:value( //Var2)</code>	5			
Example	Result							
<code>node:value( //Var2)</code>	5							
<code>node:value(String)</code>	Returns the value of the <i>node</i> represented by its path.							
	<table border="1"> <thead> <tr> <th>Example</th> <th>Result</th> </tr> </thead> <tbody> <tr> <td><code>node:value( concat("/top/Module2", "/c2/Var2") )</code></td> <td>5</td> </tr> </tbody> </table>	Example	Result	<code>node:value( concat("/top/Module2", "/c2/Var2") )</code>	5			
Example	Result							
<code>node:value( concat("/top/Module2", "/c2/Var2") )</code>	5							
<code>node node:dtos(dataPath)</code> <code>node node:dtos(dataNode)</code>	Returns the schema- <i>node</i> which is referred to by the given data- <i>node</i> or the data- <i>node</i> described by the given path. "dtos" stands for Data-TOSchema.							
	<table border="1"> <thead> <tr> <th>Example</th> <th>Result</th> </tr> </thead> <tbody> <tr> <td><code>as:dtos( '/data1' )</code></td> <td>The corresponding schema-node.</td> </tr> <tr> <td><code>as:dtos( /data1 )</code></td> <td>The corresponding schema-node.</td> </tr> </tbody> </table>	Example	Result	<code>as:dtos( '/data1' )</code>	The corresponding schema-node.	<code>as:dtos( /data1 )</code>	The corresponding schema-node.	
Example	Result							
<code>as:dtos( '/data1' )</code>	The corresponding schema-node.							
<code>as:dtos( /data1 )</code>	The corresponding schema-node.							



Function	Description							
<code>nodeset as:stod(schemaPath)</code> <code>nodeset</code> <code>as:stod(schemaNode)</code>	Returns the data-nodes which are referencing the given schema-node or the schema-node described by the given path. "stod" stands for SchemaTOData.							
	<table border="1"> <thead> <tr> <th data-bbox="573 512 1029 550">Example</th><th data-bbox="1029 512 1432 550">Result</th></tr> </thead> <tbody> <tr> <td data-bbox="573 550 1029 653"><code>as:stod( '/schema1' )</code></td><td data-bbox="1029 550 1432 653">All data-nodes for the given schema-node.</td></tr> <tr> <td data-bbox="573 653 1029 743"><code>as:stod( /schema1 )</code></td><td data-bbox="1029 653 1432 743">All data-nodes for the given schema-node.</td></tr> </tbody> </table>		Example	Result	<code>as:stod( '/schema1' )</code>	All data-nodes for the given schema-node.	<code>as:stod( /schema1 )</code>	All data-nodes for the given schema-node.
Example	Result							
<code>as:stod( '/schema1' )</code>	All data-nodes for the given schema-node.							
<code>as:stod( /schema1 )</code>	All data-nodes for the given schema-node.							
<code>string node:path(node)</code>	Returns the XPath of the <i>node</i> .							
	<table border="1"> <thead> <tr> <th data-bbox="573 819 1029 857">Example</th><th data-bbox="1029 819 1432 857">Result</th></tr> </thead> <tbody> <tr> <td data-bbox="573 857 1029 938"><code>node:path( /Var )</code></td><td data-bbox="1029 857 1432 938">/Var</td></tr> </tbody> </table>		Example	Result	<code>node:path( /Var )</code>	/Var		
Example	Result							
<code>node:path( /Var )</code>	/Var							
<code>strlist node:paths(nodeset)</code>	Returns the XPaths of the <i>node(s)</i>							
	<table border="1"> <thead> <tr> <th data-bbox="573 1001 1029 1039">Example</th><th data-bbox="1029 1001 1432 1039">Result</th></tr> </thead> <tbody> <tr> <td data-bbox="573 1039 1029 1163"><code>node:paths( //Var )</code></td><td data-bbox="1029 1039 1432 1163">All XPaths for all nodes named Var.</td></tr> </tbody> </table>		Example	Result	<code>node:paths( //Var )</code>	All XPaths for all nodes named Var.		
Example	Result							
<code>node:paths( //Var )</code>	All XPaths for all nodes named Var.							
<code>nodeSet node:order(nodeSet[, condition1[, condition2]])</code>	Orders the given nodeSet. The conditions (which must be Xpath-expressions) will be used to order the nodes. Within the condition, you can access the current node of the nodeSet. If no condition is given, the nodes are ordered by their name. If the nodeSet contains only one element the conditions will not be evaluated.							
	<table border="1"> <thead> <tr> <th data-bbox="573 1388 1029 1426">Example</th><th data-bbox="1029 1388 1432 1426">Result</th></tr> </thead> <tbody> <tr> <td data-bbox="573 1426 1029 1574"><code>node:order(//*, 'node:value(.)', 'node:name(.)')</code></td><td data-bbox="1029 1426 1432 1574">Orders all nodes by their values. Nodes with equal values will be ordered by their name.</td></tr> </tbody> </table>		Example	Result	<code>node:order(//*, 'node:value(.)', 'node:name(.)')</code>	Orders all nodes by their values. Nodes with equal values will be ordered by their name.		
Example	Result							
<code>node:order(//*, 'node:value(.)', 'node:name(.)')</code>	Orders all nodes by their values. Nodes with equal values will be ordered by their name.							
<code>Boolean node:exists(node)</code>	Checks if the given node exists (without generating an error). For disabled optional nodes, this function always returns <code>false</code> .							
	<table border="1"> <thead> <tr> <th data-bbox="573 1695 1029 1733">Example</th><th data-bbox="1029 1695 1432 1733">Result</th></tr> </thead> <tbody> <tr> <td data-bbox="573 1733 1029 1859"><code>node:exists(nonode)</code></td><td data-bbox="1029 1733 1432 1859"><code>false</code></td></tr> <tr> <td data-bbox="573 1859 1029 1861"><code>not(node:exists(nonode))</code></td><td data-bbox="1029 1859 1432 1861"><code>true</code></td></tr> </tbody> </table>		Example	Result	<code>node:exists(nonode)</code>	<code>false</code>	<code>not(node:exists(nonode))</code>	<code>true</code>
Example	Result							
<code>node:exists(nonode)</code>	<code>false</code>							
<code>not(node:exists(nonode))</code>	<code>true</code>							
<code>Boolean node:accessible(node)</code>	Checks if the given node is accessible (without generating an error). This function is similar to <code>node:exists()</code> . However, in contrary to <code>node:exists()</code> , <code>node:accessible</code> returns <code>true</code> for disabled op-							



Function	Description							
	<p>tional nodes if you use <code>node:accessible</code> in the DCtxt API and if <code>DCtxt.opt.getExistingNodesOnly()</code> returns <code>false</code>.</p> <table border="1" data-bbox="573 467 1429 617"> <thead> <tr> <th data-bbox="573 467 1002 512">Example</th><th data-bbox="1002 467 1429 512">Result</th></tr> </thead> <tbody> <tr> <td data-bbox="573 512 1002 557"><code>node:accessible(nonode)</code></td><td data-bbox="1002 512 1429 557"><code>false</code></td></tr> <tr> <td data-bbox="573 557 1002 617"><code>not(node:accessible(nonode))</code></td><td data-bbox="1002 557 1429 617"><code>true</code></td></tr> </tbody> </table>		Example	Result	<code>node:accessible(nonode)</code>	<code>false</code>	<code>not(node:accessible(nonode))</code>	<code>true</code>
Example	Result							
<code>node:accessible(nonode)</code>	<code>false</code>							
<code>not(node:accessible(nonode))</code>	<code>true</code>							
<code>Boolean node:refexists(node)</code>	<p>Returns <code>true</code> if the given reference-node and its referenced node exists (without generating an error).</p> <table border="1" data-bbox="573 729 1429 880"> <thead> <tr> <th data-bbox="573 729 1002 774">Example</th><th data-bbox="1002 729 1429 774">Result</th></tr> </thead> <tbody> <tr> <td data-bbox="573 774 1002 880"><code>node:refexists(refnode)</code></td><td data-bbox="1002 774 1429 880"><code>true</code> if the given node references an existing node.</td></tr> </tbody> </table>		Example	Result	<code>node:refexists(refnode)</code>	<code>true</code> if the given node references an existing node.		
Example	Result							
<code>node:refexists(refnode)</code>	<code>true</code> if the given node references an existing node.							
<code>Boolean node:refvalid(node)</code>	<p>Returns <code>true</code> if the given reference-node and its referenced node exists and if the referenced node is allowed to be referenced according to the schema path.</p> <p><b>NOTE</b> <b>Restrictions are not checked</b></p>  <p>You can further restrict the allowed values of a node, e.g. by adding a <code>RANGE</code> attribute to the reference node. However, these additional restrictions to a node are not checked by the <code>node:refvalid()</code> function. If you want to make sure that such additional restrictions are satisfied, you need to check for them explicitly.</p> <table border="1" data-bbox="573 1403 1429 1554"> <thead> <tr> <th data-bbox="573 1403 1002 1448">Example</th><th data-bbox="1002 1403 1429 1448">Result</th></tr> </thead> <tbody> <tr> <td data-bbox="573 1448 1002 1554"><code>node:refvalid(refnode)</code></td><td data-bbox="1002 1448 1429 1554"><code>true</code> if the given node references an existing valid node.</td></tr> </tbody> </table>		Example	Result	<code>node:refvalid(refnode)</code>	<code>true</code> if the given node references an existing valid node.		
Example	Result							
<code>node:refvalid(refnode)</code>	<code>true</code> if the given node references an existing valid node.							
<code>Boolean node:empty(node)</code>	<p>Returns <code>true</code> if the given node does not exist or if its value is empty (without generating an error) - <code>false</code> otherwise.</p> <table border="1" data-bbox="573 1666 1429 1819"> <thead> <tr> <th data-bbox="573 1666 1002 1711">Example</th><th data-bbox="1002 1666 1429 1711">Result</th></tr> </thead> <tbody> <tr> <td data-bbox="573 1711 1002 1819"><code>node:empty(node)</code></td><td data-bbox="1002 1711 1429 1819"><code>true</code> if the given node does not exist or its value is empty.</td></tr> </tbody> </table>		Example	Result	<code>node:empty(node)</code>	<code>true</code> if the given node does not exist or its value is empty.		
Example	Result							
<code>node:empty(node)</code>	<code>true</code> if the given node does not exist or its value is empty.							
<code>node-set node:difference(node-set1, node-set2)</code>	<p>Removes the "remaining quantity" of the node-set1 (returns the node-set1 not containing any nodes of the node-set2).</p>							



Function	Description	
	Example	Result
	node:difference(//Var, /Os/Var)	All Var-nodes but not /Os/Var.
<i>Boolean node:isconsecutive(node-set, base)</i>	Returns <code>true</code> if the given node-set is a consecutive list of integer-nodes based on the given base. For this, the nodes must not be ordered (e.g. 0, 2, 1 is accepted as zero-based consecutive list).	
	Example	Result
	node:isconsecutive(MyList/*, 0)	<code>true</code> if all given nodes are zero-based consecutive.
<i>Boolean node:contains(node-set, node)</i>	Returns <code>true</code> if the given node-set contains the given node.	
	Example	Result
	node:contains(node:refs(/x/y), .)	<code>true</code> if the reference /x/y points to the current node.
<i>Boolean node:containsValue(node-set, node)</i>	Returns <code>true</code> if at least one of the given node have the given value.	
	Example	Result
	node:containsValue(./*, 'myValue')	<code>true</code> if one node within the current list have the value <code>myValue</code> .
<i>int node:pos( node )</i>	Returns the position of the given node within its parent (starting with 0). Returns -1 if the given node is invalid.	
	Example	Result
	node:pos(.)	0 if the current node is the first node within its parent.
<i>node node:when( expression, [trueValue[, falseValue]] )</i>	Depending on the Boolean result of the given expression, the given <code>trueValue</code> (default is Boolean <code>true</code> ) or <code>falseValue</code> (default is Boolean <code>false</code> ) is returned.  Note, that if ' <code>trueValue</code> ' and/or ' <code>falseValue</code> ' is an expression, they are always evaluated, even if their result is not returned.	
	Example	Result
	node:when(. mod 2 = 0, ../int2 mod 2 = 0, ../int2 mod 2 = 1)	<code>true</code> if the current node and <code>int2</code> are both odd or even - <code>false</code> otherwise.



Function	Description	
	Example	Result
	<code>node:when( . = 'myValue', ../ int1, ../int2 )</code>	value of int1 if the value of the current node is myValue, value of int2 otherwise.
<code>strlist node:foreach( node-set, var- Name, expression )</code>	Executes the given expression once for each element of the given node-set where the value of the current node is stored in the variable with the given name. A list containing the result of the expression for each loop is returned.  Note, that the given expression must be a string.	
	<code>node:foreach( ecu:list('key'), 'ecu', '\$ecu * node:value(.)')</code>	A list containing each value of ecu:list multiplied with the value of the current node.
<code>node-set node:filter( node-set, ex- pression )</code>	Executes the given expression once in the context of each element of the given node-set. Returns all elements for which the expression evaluates to true.  Note, that the given expression must be a string.	
	<code>node:filter (node:refs ('ASTyped:PredefinedVariant'), 'starts-with(as:path(.), "/Some/ Package/")')</code>	A list of PredefinedVariant elements whose AUTOSAR SHORT-NAME path begins with / Some/ Package/.
<code>strlist node:range( node )</code>	Returns the possible values of the given node if this is a fixed list (as e.g. for enumerations).	
	<code>count( node:range(.) )</code>	The number of possible values for the current node.
<code>result node:fallback( expression, expression )</code>	If you convert XDM files with a parameter definition to AUTOSAR format files (BMD, EPD) and if the XDM files contain XPath expressions, often not all of these XPath expressions can be converted. For example, XPath expressions, which access the configuration, cannot be converted.	



Function	Description						
	<p>To work around this problem the <code>node:fallback()</code> function has been introduced. The method takes two XPath expressions as parameters. If the first XPath expression can be evaluated, the function returns the value of this expression. If the first expression cannot be evaluated due to a missing configuration referenced in the expression, the second XPath expression is used as a fallback.</p> <p>The <code>node:fallback()</code> function is not restricted to only handle simple path expressions. You can also use more complex expressions within <code>node:fallback()</code>, which may also include e.g. calls to nested functions. In this case, you must provide the first expression as a string, which needs to be prefixed with <code>-&gt;</code>.</p>						
	<table border="1"> <thead> <tr> <th>Example</th><th>Result</th></tr> </thead> <tbody> <tr> <td><code>node:fallback( ., 0 )</code></td><td>If the value of the current node can be evaluated, the function returns the current node. If the configuration cannot be accessed <code>0</code> is returned</td></tr> <tr> <td><code>node:fallback( as:name(.), "noName" ) // fails</code>   <code>node:fallback( "-&gt;as:name(.)", "noName" )</code></td><td>If the first line is part of an XDM module definition schema and you try to convert it to an AUTOSAR parameter definition ARXML file, then the conversion fails because Xpath first tries to evaluate the inner call to <code>as:name( . )</code>. But this behavior is not possible as it accesses the configuration. The second line shows how to solve this by providing the test expression as a string prefixed with <code>-&gt;</code>.</td></tr> </tbody> </table>	Example	Result	<code>node:fallback( ., 0 )</code>	If the value of the current node can be evaluated, the function returns the current node. If the configuration cannot be accessed <code>0</code> is returned	<code>node:fallback( as:name(.), "noName" ) // fails</code>  <code>node:fallback( "-&gt;as:name(.)", "noName" )</code>	If the first line is part of an XDM module definition schema and you try to convert it to an AUTOSAR parameter definition ARXML file, then the conversion fails because Xpath first tries to evaluate the inner call to <code>as:name( . )</code> . But this behavior is not possible as it accesses the configuration. The second line shows how to solve this by providing the test expression as a string prefixed with <code>-&gt;</code> .
Example	Result						
<code>node:fallback( ., 0 )</code>	If the value of the current node can be evaluated, the function returns the current node. If the configuration cannot be accessed <code>0</code> is returned						
<code>node:fallback( as:name(.), "noName" ) // fails</code>  <code>node:fallback( "-&gt;as:name(.)", "noName" )</code>	If the first line is part of an XDM module definition schema and you try to convert it to an AUTOSAR parameter definition ARXML file, then the conversion fails because Xpath first tries to evaluate the inner call to <code>as:name( . )</code> . But this behavior is not possible as it accesses the configuration. The second line shows how to solve this by providing the test expression as a string prefixed with <code>-&gt;</code> .						
<code>Boolean node:setautovalue( node )</code>  <code>Boolean node:setautovalue( )</code>	<p>Sets the value of the current context node or of the given node to the auto-calculated value without marking the node as manually edited. This only works if <code>node:isAuto()</code> returns <code>true</code>, i.e. if the node is a variable and marked to be auto-calculated.</p> <table border="1"> <thead> <tr> <th>Example</th><th>Result</th></tr> </thead> <tbody> <tr> <td><code>node:setautovalue( '/aNode' )</code></td><td><code>true or false</code></td></tr> <tr> <td><code>node:setautovalue( )</code></td><td><code>true or false</code></td></tr> </tbody> </table>	Example	Result	<code>node:setautovalue( '/aNode' )</code>	<code>true or false</code>	<code>node:setautovalue( )</code>	<code>true or false</code>
Example	Result						
<code>node:setautovalue( '/aNode' )</code>	<code>true or false</code>						
<code>node:setautovalue( )</code>	<code>true or false</code>						



Function	Description						
<code>string node:configclass( node )</code> <code>string node:configclass( )</code>	<p>Returns the configuration class of the current node or of the given node. If no configuration class can be retrieved, it returns an empty string.</p>						
	<table border="1" data-bbox="573 512 1429 907"> <thead> <tr> <th data-bbox="573 512 1002 563">Example</th><th data-bbox="1002 512 1429 563">Result</th></tr> </thead> <tbody> <tr> <td data-bbox="573 563 1002 736">node:configclass( '/aNode' )</td><td data-bbox="1002 563 1429 736">One of 'PublishedInformation', 'PreCompile', 'Link', 'PostBuild' or 'PostBuildSelectable'.</td></tr> <tr> <td data-bbox="573 736 1002 907">node:configclass( )</td><td data-bbox="1002 736 1429 907">One of 'PublishedInformation', 'PreCompile', 'Link', 'PostBuild' or 'PostBuildSelectable'.</td></tr> </tbody> </table>	Example	Result	node:configclass( '/aNode' )	One of 'PublishedInformation', 'PreCompile', 'Link', 'PostBuild' or 'PostBuildSelectable'.	node:configclass( )	One of 'PublishedInformation', 'PreCompile', 'Link', 'PostBuild' or 'PostBuildSelectable'.
Example	Result						
node:configclass( '/aNode' )	One of 'PublishedInformation', 'PreCompile', 'Link', 'PostBuild' or 'PostBuildSelectable'.						
node:configclass( )	One of 'PublishedInformation', 'PreCompile', 'Link', 'PostBuild' or 'PostBuildSelectable'.						
<code>ConfigClasses</code> <code>node:editableconfigclasses( )</code>	<p>Returns the currently editable configuration classes as string array. If no configuration classes are configured as editable, it returns <code>null</code>.</p>						
	<table border="1" data-bbox="573 1028 1429 1334"> <thead> <tr> <th data-bbox="573 1028 1002 1080">Example</th><th data-bbox="1002 1028 1429 1080">Result</th></tr> </thead> <tbody> <tr> <td data-bbox="573 1080 1002 1334">node:editableconfigclasses()</td><td data-bbox="1002 1080 1429 1334">An array of <code>Strings</code> with some or all of the following elements: 'PublishedInformation', 'PreCompile', 'Link', 'PostBuild', 'PostBuildSelectable'.</td></tr> </tbody> </table>	Example	Result	node:editableconfigclasses()	An array of <code>Strings</code> with some or all of the following elements: 'PublishedInformation', 'PreCompile', 'Link', 'PostBuild', 'PostBuildSelectable'.		
Example	Result						
node:editableconfigclasses()	An array of <code>Strings</code> with some or all of the following elements: 'PublishedInformation', 'PreCompile', 'Link', 'PostBuild', 'PostBuildSelectable'.						
<code>Boolean node:isreadpermitted( node )</code> <code>Boolean node:isreadpermitted( )</code>	<p>Checks whether the value of a node or the current node may be read from the current context. This depends on the configuration class of the node and on the editable configuration classes. If the value of the node is editable, because its configuration class is part of the editable configuration classes or if the value of the node was editable earlier in the process, the function returns <code>true</code>.</p>						
	<table border="1" data-bbox="573 1612 1429 1792"> <thead> <tr> <th data-bbox="573 1612 1002 1664">Example</th><th data-bbox="1002 1612 1429 1664">Result</th></tr> </thead> <tbody> <tr> <td data-bbox="573 1664 1002 1747">node:isreadpermitted( '/aNode' )</td><td data-bbox="1002 1664 1429 1747"><code>true</code> if or <code>false</code></td></tr> <tr> <td data-bbox="573 1747 1002 1792">node:isreadpermitted()</td><td data-bbox="1002 1747 1429 1792"><code>true</code> or <code>false</code></td></tr> </tbody> </table>	Example	Result	node:isreadpermitted( '/aNode' )	<code>true</code> if or <code>false</code>	node:isreadpermitted()	<code>true</code> or <code>false</code>
Example	Result						
node:isreadpermitted( '/aNode' )	<code>true</code> if or <code>false</code>						
node:isreadpermitted()	<code>true</code> or <code>false</code>						
<code>String as:formula(String )</code>	<p>Executes the given AUTOSAR formula and returns the result as a string value. The given AUTOSAR formula may also refer to valid ecuc query expressions.</p>						



Function	Description	
	Example	Result
	as:formula( '1 != 0' )	1
	as:formula( '1 == 0' )	0
<code>String as:ecucQuery(String )</code>	Executes the given AUTOSAR query expression and returns the result as a string value.	
	Example	Result
	value(refvalue(localRef("ECUC-BOOLEAN-PARAM-DEF:/TS_-TxDxM.../ComMRteUsage")))	1 if the ComMRteUsage parameter is enabled, otherwise 0

#### 6.3.4.5.5.2. Text functions

Function	Description				
<code>numlist text:range( number, number )</code>	<p>Creates a <i>list of numbers</i> over which an XPath expression can iterate, for example. The next <i>number</i> is generated by adding or subtracting 1.</p> <ul style="list-style-type: none"> <li>▶ param1: The first <i>number</i> to generate</li> <li>▶ param2: The last <i>number</i> to generate</li> </ul> <table border="1"> <thead> <tr> <th>Example</th> <th>Result</th> </tr> </thead> <tbody> <tr> <td><code>text:range( 2, 4 )</code></td> <td>(2.0, 3.0, 4.0)</td> </tr> </tbody> </table>	Example	Result	<code>text:range( 2, 4 )</code>	(2.0, 3.0, 4.0)
Example	Result				
<code>text:range( 2, 4 )</code>	(2.0, 3.0, 4.0)				
<code>numlist text:range( number, number, number )</code>	<p>Creates a <i>list of number</i> over which an XPath expression can iterate, for example. The next <i>number</i> is generated by adding or subtracting step.</p> <ul style="list-style-type: none"> <li>▶ param1: The first <i>number</i> to generate</li> <li>▶ param2: The last <i>number</i> generated is not higher (or smaller if end &lt; start) than end</li> <li>▶ param3: The incremental steps</li> </ul> <table border="1"> <thead> <tr> <th>Example</th> <th>Result</th> </tr> </thead> <tbody> <tr> <td><code>text:range(2, 8, 2)</code></td> <td>(2.0, 4.0, 6.0, 8.0)</td> </tr> </tbody> </table>	Example	Result	<code>text:range(2, 8, 2)</code>	(2.0, 4.0, 6.0, 8.0)
Example	Result				
<code>text:range(2, 8, 2)</code>	(2.0, 4.0, 6.0, 8.0)				
<code>strlist text:split(object)</code>	<p>Splits the <i>string</i>-representation of the given <i>object(s)</i> into a <i>set of nodes</i> at space boundaries.</p> <ul style="list-style-type: none"> <li>▶ param: The data to split up</li> </ul>				



Function	Description	
	Example	Result
	text:split('1 2 3')	(1, 2, 3)
<i>strlist</i> text:split( <i>object</i> , <i>string</i> )	<p>Splits a <i>string</i> into a set of <i>nodes</i> with a given delimiter.</p> <ul style="list-style-type: none"> <li>▶ param1: The data to split up</li> <li>▶ param2: The element delimiter</li> </ul>	
	text:split('1,2,3', ',')	(1, 2, 3)
<i>string</i> text:join( <i>node-set</i> )	<p>Joins the <i>string</i> representation of a list of <i>nodes</i> with spaces to a single <i>string</i>.</p> <ul style="list-style-type: none"> <li>▶ returns a <i>string</i> with the <i>node</i> value delimited by a space</li> </ul>	
	text:join(//Var1)	3 1
<i>string</i> text:join( <i>node-set</i> , <i>string</i> )	<p>Joins the <i>string</i> representation of a list of <i>nodes</i> with a given delimiter to a single <i>string</i>.</p> <ul style="list-style-type: none"> <li>▶ param1: the <i>nodes</i> to join</li> <li>▶ param2: the delimiter that should be put between the <i>node</i> values</li> <li>▶ returns a <i>string</i> with the <i>node</i> value delimited by a space</li> </ul>	
	text:join(//Var1, ',')	3,1
<i>Boolean</i> text:match( <i>string</i> , <i>string</i> )	<p>Performs a regular expression pattern match on a <i>string</i>.</p> <ul style="list-style-type: none"> <li>▶ param1: The text to search</li> <li>▶ param2: The regexp pattern to use</li> <li>▶ returns true if the pattern matches</li> </ul>	
	text:match( '123', ' ^ [0-9] * ' )	true
<i>strlist</i> text:grep( <i>node-set</i> , <i>string</i> )	<p>Performs a regular expression pattern-match on the <i>string</i>-representation of a list of nodes returning the string-representations of all <i>nodes</i> that match.</p>	



Function	Description				
	<ul style="list-style-type: none"> <li>▶ param1: The <i>nodes</i> to match</li> <li>▶ param2: The regexp pattern to use</li> <li>▶ returns the <i>string</i>-representation of all nodes that did match the pattern</li> </ul> <table border="1" style="margin-top: 10px;"> <thead> <tr> <th>Example</th><th>Result</th></tr> </thead> <tbody> <tr> <td><code>text:grep( text:split('1 2 3'), ' ^ [2-9] * ' )</code></td><td>(2, 3)</td></tr> </tbody> </table>	Example	Result	<code>text:grep( text:split('1 2 3'), ' ^ [2-9] * ' )</code>	(2, 3)
Example	Result				
<code>text:grep( text:split('1 2 3'), ' ^ [2-9] * ' )</code>	(2, 3)				
<code>string text:replace( string, string, string )</code>	<p>Performs a regular expression replace operation. The first occurrence matching pattern is replaced by the replacement string</p> <ul style="list-style-type: none"> <li>▶ param1: The input text</li> <li>▶ param2: The search pattern. The part of the text that matches is replaced</li> <li>▶ param3: The replacement <i>string</i></li> <li>▶ returns the replaced text</li> </ul> <table border="1" style="margin-top: 10px;"> <thead> <tr> <th>Example</th><th>Result</th></tr> </thead> <tbody> <tr> <td><code>text:replace( '12345', '3', '0' )</code></td><td>12045</td></tr> </tbody> </table>	Example	Result	<code>text:replace( '12345', '3', '0' )</code>	12045
Example	Result				
<code>text:replace( '12345', '3', '0' )</code>	12045				
<code>string text:replaceAll( string, string, string )</code>	<p>Performs a regular expression replace operation. All occurrences matching pattern are replaced by the replacement string</p> <ul style="list-style-type: none"> <li>▶ param1: The input text</li> <li>▶ param2: The search pattern. The part of the text that matches is replaced</li> <li>▶ param3: The replacement <i>string</i></li> <li>▶ returns the replaced text</li> </ul> <table border="1" style="margin-top: 10px;"> <thead> <tr> <th>Example</th><th>Result</th></tr> </thead> <tbody> <tr> <td><code>text:replaceAll( '12321', '1', '0' )</code></td><td>02320</td></tr> </tbody> </table>	Example	Result	<code>text:replaceAll( '12321', '1', '0' )</code>	02320
Example	Result				
<code>text:replaceAll( '12321', '1', '0' )</code>	02320				
<code>Boolean text:uniq(node-set,string)</code>	<p>Tests if the value is unique inside the <i>node-set</i>.</p> <ul style="list-style-type: none"> <li>▶ param1: The list to search</li> <li>▶ param2: The value to search inside the list.</li> <li>▶ returns true if the value occurs at most once inside the list.</li> </ul>				



Function	Description	
	Example	Result
	text:uniq( text:split('1 2 1'), '1' )	false
<i>node-set</i> text:difference( <i>node-set1</i> , <i>node-set2</i> )	Returns the <i>remaining quantity</i> of <i>node-set1</i> (returns the <i>node-set1</i> not containing any string of the <i>node-set2</i> ).	
	Example	Result
	text:difference( text:split('1 2 3 4'), text:split('1 3') )	(2, 4)
<i>strlist</i> text:concat( <i>node-set</i> , <i>node-set</i> )	Concatenates the <i>string</i> -representation of the given <i>objects</i> . The first element of the returned <i>node-set</i> is the concatenation of the first elements of both given <i>node-sets</i> . If one of the two <i>node-sets</i> is shorter than the other, it's last element is used for the open calculations.	
	Example	Result
	text:concat( text:split('1 2 3'), text:split('4 5') )	(14, 25, 35)
<i>string</i> text:toupper( <i>string</i> )	Converts all of the characters in the given <i>string</i> to upper case	
	Example	Result
	text:toupper( 'small' )	SMALL
<i>string</i> text:tolower( <i>string</i> )	Converts all of the characters in the given <i>string</i> to lower case	
	Example	Result
	text:tolower( 'BIG' )	big
<i>string</i> text:order( <i>node-set</i> )	Sorts the <i>string</i> -representation of the given <i>objects</i> in ascending order and returns the sorted list as <i>node-set</i> .	
	Example	Result
	text:order( text:split('zoo foo boo') )	(boo,foo,zoo)
<i>string</i> text:order( <i>node-set</i> , <i>direction</i> )	Sorts the <i>string</i> -representation of the given <i>objects</i> in the order given by <i>direction</i> and returns the sorted list as <i>node-set</i> . If <i>direction</i> is < 0, the order is descending, >= 0 means ascending order	
	Example	Result
	text:order( text:split('zoo foo boo', -1) )	(zoo,foo,boo)



Function	Description				
<code>string text:order(node-set, string, number)</code>	<p>Sorts the <i>string</i>-representation of the given <i>objects</i> in ascending order and returns the sorted list as <i>node-set</i>. A pattern and a group index are used to compute the portion of the strings which should be considered for the sort condition.</p> <ul style="list-style-type: none"> <li>▶ param1: The list to sort</li> <li>▶ param2: A regular expression which matches a substring of the strings to sort. The regexp may contain groupings (round brackets)</li> <li>▶ param3: The group index of the regular expression which should be used for the sort condition; index 0 is the whole regular expression match, 1 is the first group, ...</li> <li>▶ returns the sorted list</li> </ul>				
	<table border="1"> <thead> <tr> <th data-bbox="581 911 1002 961">Example</th><th data-bbox="1002 911 1432 961">Result</th></tr> </thead> <tbody> <tr> <td data-bbox="581 961 1002 1080"><code>text:order( text:split('zoo-b-b goo-zz-a foo-a-c boo-z-d'), '.*?-([^-]+)', 1 )</code></td><td data-bbox="1002 961 1432 1080">(foo-a-c, zoo-b-b, boo-z-d, goo-zz-a)</td></tr> </tbody> </table>	Example	Result	<code>text:order( text:split('zoo-b-b goo-zz-a foo-a-c boo-z-d'), '.*?-([^-]+)', 1 )</code>	(foo-a-c, zoo-b-b, boo-z-d, goo-zz-a)
Example	Result				
<code>text:order( text:split('zoo-b-b goo-zz-a foo-a-c boo-z-d'), '.*?-([^-]+)', 1 )</code>	(foo-a-c, zoo-b-b, boo-z-d, goo-zz-a)				
<code>string text:order(node-set, string, number, direction)</code>	<p>Sorts the <i>string</i>-representation of the given <i>objects</i> in the given order and returns the sorted list as <i>node-set</i>. A pattern and a group index are used to compute the portion of the strings which should be considered for the sort condition.</p> <ul style="list-style-type: none"> <li>▶ param1: The list to sort</li> <li>▶ param2: A regular expression which matches a substring of the strings to sort. The regexp may contain groupings (round brackets)</li> <li>▶ param3: The group index of the regular expression which should be used for the sort condition; index 0 is the whole regular expression match, 1 is the first group, ...</li> <li>▶ param4: If the value is &lt; 0, the sort order is descending if it is &gt;= 0, the order is ascending</li> <li>▶ returns the sorted list</li> </ul>				
	<table border="1"> <thead> <tr> <th data-bbox="581 1697 1002 1747">Example</th><th data-bbox="1002 1697 1432 1747">Result</th></tr> </thead> <tbody> <tr> <td data-bbox="581 1747 1002 1888"><code>text:order( text:split('zoo-b-b goo-zz-a foo-a-c boo-z-d'), '.*?-([^-]+)', 1, -1 )</code></td><td data-bbox="1002 1747 1432 1888">(goo-zz-a, boo-z-d, zoo-b-b, foo-a-c)</td></tr> </tbody> </table>	Example	Result	<code>text:order( text:split('zoo-b-b goo-zz-a foo-a-c boo-z-d'), '.*?-([^-]+)', 1, -1 )</code>	(goo-zz-a, boo-z-d, zoo-b-b, foo-a-c)
Example	Result				
<code>text:order( text:split('zoo-b-b goo-zz-a foo-a-c boo-z-d'), '.*?-([^-]+)', 1, -1 )</code>	(goo-zz-a, boo-z-d, zoo-b-b, foo-a-c)				
<code>Boolean text:contains(node-set, value)</code>	Returns true if the given list of elements contains the given value				



Function	Description	
	Example	Result
	text:contains( text:split('aa bb cc'), 'aa' )	true
<i>Boolean</i> text:contains( <i>string</i> , value)	Returns true if the given string contains the given value	
	Example	Result
	text:contains( 'testString', 'stSt' )	true
<i>int</i> text:indexOf( <i>string</i> , value)	Returns the position (starting with 0) of the given value within the given string. If the given value is not contained, -1 is returned.	
	Example	Result
	text:indexOf( 'testString', 'stSt' )	2
<i>int</i> text:lastIndexOf( <i>string</i> , value)	Returns the position (starting with 0) of the last occurrence of the given value within the given string. If the given value is not contained, -1 is returned.	
	Example	Result
	text:lastIndexOf( 'testString', 't' )	5

#### 6.3.4.5.5.3. Number functions

**NOTE**
**Data types**


The XPath engine always calculates with Double values. Therefore be aware that the result can be a Double value!

Function	Description	
	Example	Result
<i>numlist</i> num:add( <i>numlist</i> , <i>numlist</i> )	Adds the values of the two lists. If one of the two lists is shorter, then its last element is used for the open calculations.	
	num:add( text:split('1.1 2.2 3.4'), text:split('4 5.3') )	(5.1, 7.5, 8.7)
<i>numlist</i> num:div( <i>numlist</i> , <i>numlist</i> )	Divides the values of the two lists. If one of the two lists is shorter, then its last element is used for the open calculations.	



Function	Description	
	Example	Result
	num:div( 8, text:split('2 4') )	(4.0, 2.0)
<i>Boolean</i> num:isnumber( <i>object</i> )	Returns true if the given <i>object</i> is a or can be converted to a <i>number</i>	
	Example	Result
	num:isnumber( '3' )	true
<i>numlist</i> num:mod( <i>numlist</i> , <i>numlist</i> )	Computes the modulus for the values of the two lists. If one of the two lists is shorter, then it's last element is used for the open calculations.	
	Example	Result
	num:mod( text:split('5 6 7'), text:split('2 3 4') )	(1, 0, 3)
<i>numlist</i> num:mul( <i>numlist</i> , <i>numlist</i> )	Multiplies the values of the two lists. If one of the two lists is shorter, then it's last element is used for the open calculations.	
	Example	Result
	num:mul( 5, text:split('1 2 3') )	(5.0, 10.0, 15.0)
<i>numlist</i> num:negate( <i>numlist</i> )	Negates all values of the list	
	Example	Result
	num:negate( text:split('-1 0 3') )	(1, 0, -3)
<i>intlist</i> num:sub( <i>numlist</i> )	Subtracts the values of the two lists. If one of the two lists is shorter, then it's last element is used for the open calculations.	
	Example	Result
	num:sub( text:split('4 5') text:split('3 2 1') )	(1, 3, 4)
<i>int</i> num:i( <i>numlist</i> )	Returns an <i>integer</i> value from any <i>number</i> (does a floor).	
	Example	Result
	num:i( '5.3' )	5
<i>strlist</i> num:f( <i>numlist</i> )	Returns a <i>string</i> value for every number argument. This function converts any number into a string that holds the same number in floating-point format. In contrast to other conversion functions, this function never represents the result in scientific notation. If a value is passed as an argument, which cannot be converted into a double value, the function fails with an error.	



Function	Description	
	<b>Example</b>	<b>Result</b>
	num:f( '5.3' )	5.3
	num:f( '5.3E3' )	5300.0
	num:f( 'NaN' )	NaN
	num:f( 'Infinity' )	Infinity
	num:f( 'foobar' )	error
<i>intlist</i> num:order( <i>numlist</i> )	Sorts the given numbers in ascending order and returns the result.	
	<b>Example</b>	<b>Result</b>
	num:order( text:split('-5.5 3 6 1 5 -6 0 3.5 -5.3') )	(-6, -5.5, -5.3, 0, 1, 3, 3.5, 5, 6)
<i>intlist</i> num:order( <i>numlist, direction</i> )	Sorts the given numbers in the given order and returns the result. If <i>direction</i> is < 0, the order is descending if <i>direction</i> is >= 0, the order is ascending.	
	<b>Example</b>	<b>Result</b>
	num:order( text:split('-5.5 3 6 1 5 -6 0 3.5 -5.3', -1) )	(6, 5, 3.5, 3, 1, 0, -5.3, -5.5, -6)
<i>number</i> num:min( <i>node-set</i> )	Returns the smallest value of the given nodes	
	<b>Example</b>	<b>Result</b>
	num:min( /* )	The smallest value of the given nodes
<i>number</i> num:max( <i>node-set</i> )	Returns the biggest value of the given nodes	
	<b>Example</b>	<b>Result</b>
	num:max( /* )	The biggest value of the given nodes

#### 6.3.4.5.5.3.1. Number conversions

Function	Description
<i>int</i> num:hextoint( <i>string</i> )	Converts the given <i>string</i> into an <i>integer</i> . The string represents a hexa-decimal number ('0x<hex-number>' or '<hex-number>').



Function	Description						
	<ul style="list-style-type: none"> <li>▶ param1: String representing a hexa-decimal number</li> <li>▶ returns an <i>integer</i> representation of the number</li> </ul> <table border="1" data-bbox="794 518 1432 680"> <thead> <tr> <th data-bbox="794 518 1111 570">Example</th><th data-bbox="1111 518 1432 570">Result</th></tr> </thead> <tbody> <tr> <td data-bbox="794 570 1111 622">num:hextoint( '0xA' )</td><td data-bbox="1111 570 1432 622">10</td></tr> <tr> <td data-bbox="794 622 1111 673">num:hextoint( '0xfb' )</td><td data-bbox="1111 622 1432 673">251</td></tr> </tbody> </table>	Example	Result	num:hextoint( '0xA' )	10	num:hextoint( '0xfb' )	251
Example	Result						
num:hextoint( '0xA' )	10						
num:hextoint( '0xfb' )	251						
<i>int num:octtoint(string)</i>	<p>Converts the given <i>string</i> into an <i>integer</i>. The string represents an octal number ('0&lt;oct-number&gt;' or '&lt;oct-number&gt;').</p> <ul style="list-style-type: none"> <li>▶ param1: String representing an octal number</li> <li>▶ returns an <i>integer</i> representation of the number</li> </ul> <table border="1" data-bbox="794 945 1432 1057"> <thead> <tr> <th data-bbox="794 945 1111 997">Example</th><th data-bbox="1111 945 1432 997">Result</th></tr> </thead> <tbody> <tr> <td data-bbox="794 997 1111 1048">num:octtoint( '010' )</td><td data-bbox="1111 997 1432 1048">8</td></tr> </tbody> </table>	Example	Result	num:octtoint( '010' )	8		
Example	Result						
num:octtoint( '010' )	8						
<i>int num:bintoint(string)</i>	<p>Converts the given <i>string</i> into an <i>integer</i>. The string represents a binary number ('b&lt;bin-number&gt;' or '&lt;bin-number&gt;').</p> <ul style="list-style-type: none"> <li>▶ param1: String representing a binary number</li> <li>▶ returns an <i>integer</i> representation of the number</li> </ul> <table border="1" data-bbox="794 1327 1432 1444"> <thead> <tr> <th data-bbox="794 1327 1111 1379">Example</th><th data-bbox="1111 1327 1432 1379">Result</th></tr> </thead> <tbody> <tr> <td data-bbox="794 1379 1111 1430">num:bintoint( 'b11' )</td><td data-bbox="1111 1379 1432 1430">3</td></tr> </tbody> </table>	Example	Result	num:bintoint( 'b11' )	3		
Example	Result						
num:bintoint( 'b11' )	3						
<i>int num:dectoint(string)</i>	<p>Converts the given <i>string</i> into an <i>integer</i>. The string represents a decimal number.</p> <ul style="list-style-type: none"> <li>▶ param1: String representing a decimal number</li> <li>▶ returns an <i>integer</i> representation of the number</li> </ul> <table border="1" data-bbox="794 1664 1432 1792"> <thead> <tr> <th data-bbox="794 1664 1111 1715">Example</th><th data-bbox="1111 1664 1432 1715">Result</th></tr> </thead> <tbody> <tr> <td data-bbox="794 1715 1111 1767">num:dectoint( '10' )</td><td data-bbox="1111 1715 1432 1767">10</td></tr> </tbody> </table>	Example	Result	num:dectoint( '10' )	10		
Example	Result						
num:dectoint( '10' )	10						
<i>int num:radixtoint(string)</i>	<p>Converts the given <i>string</i> representing a number with arbitrary radix into an <i>integer</i> (uses the corresponding prefix '0x&lt;hex-number&gt;', '0&lt;oct-number&gt;', 'b&lt;bin-number&gt;' as the value type indication).</p>						



Function	Description						
	<ul style="list-style-type: none"> <li>▶ param1: String representing a number to convert</li> <li>▶ returns an <i>integer</i> representation of the <i>number</i></li> </ul> <table border="1" data-bbox="794 482 1429 635"> <thead> <tr> <th data-bbox="794 482 1111 534">Example</th><th data-bbox="1111 482 1429 534">Result</th></tr> </thead> <tbody> <tr> <td data-bbox="794 534 1111 586">num:radixtoint( '0xA' )</td><td data-bbox="1111 534 1429 586">10</td></tr> <tr> <td data-bbox="794 586 1111 635">num:radixtoint( 'b11' )</td><td data-bbox="1111 586 1429 635">3</td></tr> </tbody> </table>	Example	Result	num:radixtoint( '0xA' )	10	num:radixtoint( 'b11' )	3
Example	Result						
num:radixtoint( '0xA' )	10						
num:radixtoint( 'b11' )	3						
<i>string</i> num:inttobin( <i>int</i> )	<p>Starting with <i>b</i>, num:inttobin returns a string representation of the given integer in the binary system.</p> <ul style="list-style-type: none"> <li>▶ param1: Integer to convert</li> <li>▶ returns a string representing the given integer in binary system</li> </ul> <table border="1" data-bbox="794 909 1429 1017"> <thead> <tr> <th data-bbox="794 909 1111 961">Example</th><th data-bbox="1111 909 1429 961">Result</th></tr> </thead> <tbody> <tr> <td data-bbox="794 961 1111 1010">num:inttobin( 3 )</td><td data-bbox="1111 961 1429 1010">b11</td></tr> </tbody> </table>	Example	Result	num:inttobin( 3 )	b11		
Example	Result						
num:inttobin( 3 )	b11						
<i>string</i> num:inttobin( <i>int, int</i> )	<p>Starting with <i>b</i>, num:inttobin returns a string representation of the given integer in the binary system. The second <i>integer</i> argument specifies a fix digit length of the resulting string (without <i>b</i>). Either leading zeros are inserted or the result may be trimmed to fit this length.</p> <ul style="list-style-type: none"> <li>▶ param1: Integer to convert</li> <li>▶ param2: Digit length of the resulting string (without <i>b</i>)</li> <li>▶ returns a string representing the given integer in binary system.</li> </ul> <table border="1" data-bbox="794 1560 1429 1646"> <thead> <tr> <th data-bbox="794 1560 1111 1612">Example</th><th data-bbox="1111 1560 1429 1612">Result</th></tr> </thead> <tbody> <tr> <td data-bbox="794 1612 1111 1639">num:inttobin( 3, 4 )</td><td data-bbox="1111 1612 1429 1639">b0011</td></tr> </tbody> </table>	Example	Result	num:inttobin( 3, 4 )	b0011		
Example	Result						
num:inttobin( 3, 4 )	b0011						
<i>string</i> num:inttohex( <i>int</i> )	<p>Starting with <i>0x</i>, num:inttohex returns a string representation of the given integer in the hexa-decimal system.</p> <ul style="list-style-type: none"> <li>▶ param1: Integer to convert</li> <li>▶ returns a string representing the given integer in hexa-decimal system</li> </ul>						



Function	Description	
	Example	Result
	num:inttohex( 15 )	0xf
	num:inttohex( 254 )	0xfe
<p><i>string num:inttohex(int, int)</i></p>	<p>Starting with 0x, num:inttohex returns a string representation of the given integer in the hexa-decimal system. The second <i>integer</i> argument specifies a fix digit length of the resulting string (without 0x). Either leading zeros are inserted or the result may be trimmed to fit this length.</p> <ul style="list-style-type: none"> <li>▶ param1: Integer to convert</li> <li>▶ param2: Digit length of the resulting string (without 0x)</li> <li>▶ returns a string representing the given integer in hexa-decimal system.</li> </ul>	
	Example	Result
	num:inttohex( 15, 2 )	0x0f
	num:inttohex( 254, 1 )	0xe
<p><i>string num:inttooct(int)</i></p>	<p>Starting with \0, num:inttooct returns a string representation of the given integer in the octal system.</p> <ul style="list-style-type: none"> <li>▶ param1: Integer to convert</li> <li>▶ returns a string representing the given integer in octal system</li> </ul>	
	Example	Result
	num:inttooct( 15 )	\017
<p><i>string num:inttooct(int, int)</i></p>	<p>Starting with \0, num:inttooct returns a string representation of the given integer in the octal system. The second <i>integer</i> argument specifies a fix digit length of the resulting string (without \0). Either leading zeros are inserted or the result may be trimmed to fit this length.</p> <ul style="list-style-type: none"> <li>▶ param1: Integer to convert</li> <li>▶ param2: Digit length of the resulting string (without \0)</li> </ul>	



Function	Description				
	<ul style="list-style-type: none"> <li>▶ returns a string representing the given integer in octal system.</li> </ul> <table border="1" style="margin-top: 10px;"> <thead> <tr> <th>Example</th><th>Result</th></tr> </thead> <tbody> <tr> <td>num:inttooct( 15, 4 )</td><td>\00017</td></tr> </tbody> </table>	Example	Result	num:inttooct( 15, 4 )	\00017
Example	Result				
num:inttooct( 15, 4 )	\00017				

#### 6.3.4.5.5.4. Bit functions

Function	Description				
<i>int</i> bit:and( <i>int,int</i> )	<p>Does a bitwise and of two <i>integer</i> values.</p> <ul style="list-style-type: none"> <li>▶ var1: First <i>integer</i> value</li> <li>▶ var2: Second <i>integer</i> value</li> <li>▶ returns anded value of the two <i>integers</i></li> </ul> <table border="1" style="margin-top: 10px;"> <thead> <tr> <th>Example</th><th>Result</th></tr> </thead> <tbody> <tr> <td>bit:and( 5, 3 )</td><td>1</td></tr> </tbody> </table>	Example	Result	bit:and( 5, 3 )	1
Example	Result				
bit:and( 5, 3 )	1				
<i>int</i> bit:or( <i>int,int</i> )	<p>Does a bitwise or of two <i>integer</i> values.</p> <ul style="list-style-type: none"> <li>▶ var1: First <i>integer</i> value</li> <li>▶ var2: Second <i>integer</i> value</li> <li>▶ returns or-ed value of the two <i>integers</i></li> </ul> <table border="1" style="margin-top: 10px;"> <thead> <tr> <th>Example</th><th>Result</th></tr> </thead> <tbody> <tr> <td>bit:or( 5, 3 )</td><td>7</td></tr> </tbody> </table>	Example	Result	bit:or( 5, 3 )	7
Example	Result				
bit:or( 5, 3 )	7				
<i>int</i> bit:xor( <i>int,int</i> )	<p>Does a bitwise xor of two <i>integer</i> values.</p> <ul style="list-style-type: none"> <li>▶ var1: First <i>integer</i> value</li> <li>▶ var2: Second <i>integer</i> value</li> <li>▶ returns xor-ed value of the two <i>integers</i></li> </ul> <table border="1" style="margin-top: 10px;"> <thead> <tr> <th>Example</th><th>Result</th></tr> </thead> <tbody> <tr> <td>bit:xor(5,3)</td><td>6</td></tr> </tbody> </table>	Example	Result	bit:xor(5,3)	6
Example	Result				
bit:xor(5,3)	6				
<i>int</i> bit:not( <i>int</i> )	<p>Does a bitwise not on an <i>integer</i> value.</p> <ul style="list-style-type: none"> <li>▶ var1: <i>integer</i> value</li> <li>▶ returns inverted value of the <i>integer</i></li> </ul>				



Function	Description	
	Example	Result
	bit:and( bit:not(3), 7)	4
<i>int</i> bit:shl( <i>int,int</i> )	<p>Does a bitwise shift left on an <i>integer</i> value.</p> <ul style="list-style-type: none"> <li>▶ param1: <i>integer</i> value</li> <li>▶ param2: <i>number</i> of bits to shift left</li> <li>▶ returns the shifted value</li> </ul>	
	bit:shl(5,1)	10
<i>int</i> bit:shr( <i>int,int</i> )	<p>Does a bitwise shift right on an <i>integer</i> value.</p> <ul style="list-style-type: none"> <li>▶ param1: <i>integer</i> value</li> <li>▶ param2: <i>number</i> of bits to shift right</li> <li>▶ returns the shifted value</li> </ul>	
	bit:shr(10,1)	5
<i>int</i> bit:bitclear( <i>int, int</i> )	Clears the <param2>th bit of the given param1. The first bit is bit nr. 0.	
	bit:bitclear(7,2)	3
<i>int</i> bit:bitset( <i>int, int</i> )	Sets the <param2>th bit of the given param1. The first bit is bit nr. 0.	
	bit:bitset(3,2)	7
<i>Boolean</i> bit:getbit( <i>int, int</i> )	Returns true if the <param2>th bit is set within param1. Otherwise false is returned. The first bit is bit nr. 0.	
	bit:getbit(8,1)	false



#### 6.3.4.5.5.5. Variable functions

Function	Description	
<code>Boolean var:defined('varName')</code>	Checks if the variable with the given name is defined (assigned to a value) without generating an error.	
	Example	Result
	<code>var:defined('myVar')</code>	true if the variable is defined, false otherwise.
<code>Boolean var:set('varName', 'value')</code>	Defines an xpath variable and sets its value. If the variable already exists, its value is overwritten.	
	Example	Result
	<code>var:set('myVar','myValue')</code>	true if the variable could be set, false otherwise

#### 6.3.4.5.5.6. License functions

Function	Description	
<code>Boolean lic:feature('feature')</code>	Returns true if the requested feature is covered by a license.	
	Example	Result
	<code>lic:feature('YourFeature')</code>	true if the feature YourFeature is covered by a license.

**NOTE**

***Boolean lic:feature('moduleId', 'feature') is deprecated***

This function is not supported any longer. Use `boolean lic:feature('feature')` instead.



#### 6.3.4.5.5.7. Util functions

Function	Description
<code>String util:generationDir(node)</code>	Returns the generation path for the context module. The argument is any node inside the module configuration.



Function	Description	
	Example	Result
	util:generationDir(.)	The generation path for the context module
<i>String util:generationDir(string)</i>	Returns the generation path for the context module. The argument is the name of a module configuration.	
	Example	Result
	util:generationDir('ComConfig')	The generation path for the context module

#### 6.3.4.5.5.8. Documentation functions

This section provides methods for querying information of the parameter comments.

Function	Description				
<i>Boolean doc:has(node, string)</i>	<p>Use this method to query whether a documentation attribute <code>string</code> is set for <code>node</code>:</p> <ul style="list-style-type: none"> <li>▶ param1: Represents the <code>node</code> to query.</li> <li>▶ param2: Represents the attribute name for the documentation, e.g. <code>COMMENTS</code>.</li> </ul> <p>If the node does not exist or is not a node, this method returns <code>false</code>.</p> <p>If the attribute is not a documentation attribute, this method returns <code>false</code>.</p> <table border="1"> <thead> <tr> <th>Example</th> <th>Result</th> </tr> </thead> <tbody> <tr> <td><code>doc:has(., 'COMMENTS')</code></td> <td><code>true</code> if the current node contains documentation of the type <code>COMMENTS</code>.</td> </tr> </tbody> </table>	Example	Result	<code>doc:has(., 'COMMENTS')</code>	<code>true</code> if the current node contains documentation of the type <code>COMMENTS</code> .
Example	Result				
<code>doc:has(., 'COMMENTS')</code>	<code>true</code> if the current node contains documentation of the type <code>COMMENTS</code> .				
<i>int doc:count(node, string)</i>	<p>Use this method to query the number of entries for the given documentation attribute.</p> <ul style="list-style-type: none"> <li>▶ param1: Represents the <code>node</code> to query.</li> <li>▶ param2: Represents the attribute name for the documentation, e.g. <code>COMMENTS</code>.</li> </ul> <p>Returns <code>0</code> if:</p>				



Function	Description				
	<ul style="list-style-type: none"> <li>▶ <code>node</code> does not exist or is not a node.</li> <li>▶ the attribute is not a documentation attribute.</li> <li>▶ the attribute is not set.</li> </ul> <table border="1" data-bbox="573 534 1429 676"> <thead> <tr> <th data-bbox="573 534 1013 586">Example</th><th data-bbox="1013 534 1429 586">Result</th></tr> </thead> <tbody> <tr> <td data-bbox="573 586 1013 676"><code>doc:count(., 'COMMENTS')</code></td><td data-bbox="1013 586 1429 676">count, the number of entries for the <code>COMMENTS</code>-type.</td></tr> </tbody> </table>	Example	Result	<code>doc:count(., 'COMMENTS')</code>	count, the number of entries for the <code>COMMENTS</code> -type.
Example	Result				
<code>doc:count(., 'COMMENTS')</code>	count, the number of entries for the <code>COMMENTS</code> -type.				
<code>strlist doc:names(node, string, number)</code>	<p>Use this method to query the names of the fields for the attribute with the given index of the node.</p> <ul style="list-style-type: none"> <li>▶ param1: Represents the <code>node</code> to query.</li> <li>▶ param2: Represents the attribute name for the documentation, e.g. <code>COMMENTS</code>.</li> <li>▶ param3: Represents the entry index to query.</li> </ul> <p>Returns an empty list if:</p> <ul style="list-style-type: none"> <li>▶ <code>node</code> does not exist or is not a node.</li> <li>▶ the attribute does not exist or is not a documentation attribute</li> <li>▶ the entry with the given index does not exist.</li> </ul>				
	<table border="1" data-bbox="573 1244 1429 1388"> <thead> <tr> <th data-bbox="573 1244 1013 1295">Example</th><th data-bbox="1013 1244 1429 1295">Result</th></tr> </thead> <tbody> <tr> <td data-bbox="573 1295 1013 1388"><code>"doc:names(., 'COMMENTS', 0)</code></td><td data-bbox="1013 1295 1429 1388">strlist of field names added to this node with the index 0.</td></tr> </tbody> </table>	Example	Result	<code>"doc:names(., 'COMMENTS', 0)</code>	strlist of field names added to this node with the index 0.
Example	Result				
<code>"doc:names(., 'COMMENTS', 0)</code>	strlist of field names added to this node with the index 0.				
<code>strlist doc:languages(node, string, number)</code>	<p>Use this method to query the languages of the fields for the attribute with the given index of the node.</p> <ul style="list-style-type: none"> <li>▶ param1: Represents the <code>node</code> to query.</li> <li>▶ param2: Represents the attribute name for the documentation, e.g. <code>COMMENTS</code>.</li> <li>▶ param3: Represents the entry index to query.</li> </ul> <p>This method returns a list of used languages for this node or an empty list if:</p> <ul style="list-style-type: none"> <li>▶ <code>node</code> does not exist or is not a node.</li> <li>▶ the attribute does not exist or is not a documentation attribute.</li> <li>▶ the entry with the given index does not exist.</li> </ul>				



Function	Description	
	Example	Result
	doc:languages(., 'COMMENTS', 1)	strlist, list of language codes used for the parameter COMMENTS for the entry with the index 1.
<code>str doc:field(node, string, number, string, string, string)</code>	<p>Use this method to query a field value for the attribute with the given index of the node and return it in the requested format.</p> <ul style="list-style-type: none"> <li>▶ param1: Represents the node to query.</li> <li>▶ param2: Represents the attribute name for the documentation, e.g. COMMENTS.</li> <li>▶ param3: Represents the entry index to query.</li> <li>▶ param4: Represents the field name.</li> <li>▶ param5: Represents the requested format if it is supported for the given field. The format <code>text/plain</code> is valid.</li> <li>▶ param6: Represents the language to query. If it is not set, the default language <code>en</code> is used.</li> </ul> <p>Returns an empty string if:</p> <ul style="list-style-type: none"> <li>▶ node does not exist or is not a node.</li> <li>▶ the attribute does not exist or is not a documentation attribute.</li> <li>▶ the entry with the given index does not exist.</li> <li>▶ the requested format is not supported.</li> </ul>	
	Example	Result
	doc:field(., 'COMMENTS', 0, 'status', 'text/plain', 'en')	str, returns the value in the given format.

#### 6.3.4.5.5.9. Variant functions

This section provides methods for querying information about post-build selectable variants.

Function	Description
<code>Boolean variant:mayHave(node)</code>	<p>Use this method to query whether a given node is able to have post-build selectable variants:</p> <p>If the given node does not exist in the currently active post-build selectable variant or is not a node, this function returns <code>false</code>.</p>



Function	Description	
	Example	Result
	variant:mayHave(.)	true if the current node may have post-build selectable variants.
<i>Boolean</i> variant:isActive( )	Use this method to query whether there is one currently active post-build selectable variant configured in the project.	
	Example	Result
	variant:isActive( )	true if there is one currently active post-build selectable variant configured in the project.
<i>string</i> variant:name( )	<p>Use this method to query the short name of the currently active post-build selectable variant.</p> <p>Returns an empty string if no currently active post-build selectable variant exists.</p>	
	Example	Result
	variant:name( )	<p>PredefinedVariantOne</p> <p>Represents the short name of the predefined post-build selectable variant, which is currently active in the project.</p>
<i>strlist</i> variant:all( )	<p>Use this method to get the short names of all configured post-build selectable variants.</p> <p>Returns an empty list if there are no post-build selectable variants configured.</p>	
	Example	Result
	variant:all( )	<p>(PredefinedVariantOne, PredefinedVariantTwo)</p> <p>The short name list of all configured predefined post-build selectable variants.</p>
<i>int</i> variant:size( )	Use this method to get the number of all configured post-build selectable variants.	
	Example	Result
	variant:size( )	2



Function	Description				
<code>Boolean variant:hasCondition(node)</code>	<p>Use this method to query if the given node has one or more post-build selectable variant conditions configured.</p> <p>If the node does not exist in the currently active post-build selectable variant or is not a node, this method returns <code>false</code>.</p>				
	<table border="1"> <thead> <tr> <th>Example</th><th>Result</th></tr> </thead> <tbody> <tr> <td><code>variant:hasCondition(.)</code></td><td><code>true</code> if the node has one or more post-build selectable variant conditions configured.</td></tr> </tbody> </table>	Example	Result	<code>variant:hasCondition(.)</code>	<code>true</code> if the node has one or more post-build selectable variant conditions configured.
Example	Result				
<code>variant:hasCondition(.)</code>	<code>true</code> if the node has one or more post-build selectable variant conditions configured.				

#### 6.3.4.5.5.10. Custom data attribute functions

This section provides methods for retrieving values of custom attributes of a data node. For more details about custom data attributes, see [Section 5.1.4.2.4, “Custom data attribute”](#).

Function	Description						
<code>strlist custAttr:getValues(node, String)</code>	Use this method to query the list of values for a given custom attribute id for a given node or the context node.						
	<table border="1"> <thead> <tr> <th>Example</th><th>Result</th></tr> </thead> <tbody> <tr> <td><code>custAttr:getValues(/Var, 'label')</code></td><td>List of strings, if the <code>Var</code> node has a custom attribute with the ID set to <code>label</code>.</td></tr> <tr> <td><code>custAttr:getValues('label')</code></td><td>List of strings, if the context node has a custom attribute with the ID <code>label</code>.</td></tr> </tbody> </table>	Example	Result	<code>custAttr:getValues(/Var, 'label')</code>	List of strings, if the <code>Var</code> node has a custom attribute with the ID set to <code>label</code> .	<code>custAttr:getValues('label')</code>	List of strings, if the context node has a custom attribute with the ID <code>label</code> .
Example	Result						
<code>custAttr:getValues(/Var, 'label')</code>	List of strings, if the <code>Var</code> node has a custom attribute with the ID set to <code>label</code> .						
<code>custAttr:getValues('label')</code>	List of strings, if the context node has a custom attribute with the ID <code>label</code> .						

#### 6.3.4.5.6. Additional XPath-functions for AUTOSAR

Each path-expression prefixed with `ASPath:` will be interpreted SHORT-NAME-path.

So, for example if `/MyPackage/Can` is a valid SHORT-NAME-path:

```
as:ref ('/MyPackage/CAN')
  is equivalent to
ASPath:/MyPackage/Can
  is equivalent to
/AUTOSAR/TOP-LEVEL-PACKAGES/MyPackage/ELEMENTS/Can
```



ASPath:/MyPackage/Can can be used anywhere as xpath-expression and will always be interpreted as SHORT-NAME-path. The conversion from an XPath-path to a SHORT-NAME-path can be done with the function as:path as described in the table below.

Nodes without SHORT-NAME (e.g. lists, choices and variable- nodes) cannot be addressed when using the SHORT-NAME-path.

To provide convenient access to the AUTOSAR node-structure, additional functions have been introduced to XPath:

Function	Description						
<code>node as:ref(<i>refnode</i>)</code>	Returns the <i>node</i> , which is referred to by the given <i>refnode</i> <i>refnode</i> : may be a 'Reference', 'Choice Reference' or a 'Symbolic Name Reference'						
<code>node as:ref(<i>string</i>)</code>	Returns the <i>node</i> , which is referred to by the given SHORT-NAME-path <table border="1" data-bbox="635 1033 1429 1179"> <thead> <tr> <th>Example</th><th>Result</th></tr> </thead> <tbody> <tr> <td><code>as:ref( '/myPkg/ myModule' )</code></td><td>the module <i>myModule</i> within package <i>myPkg</i></td></tr> </tbody> </table>	Example	Result	<code>as:ref( '/myPkg/ myModule' )</code>	the module <i>myModule</i> within package <i>myPkg</i>		
Example	Result						
<code>as:ref( '/myPkg/ myModule' )</code>	the module <i>myModule</i> within package <i>myPkg</i>						
<code>node as:dtos(<i>dataPath</i>)</code> <code>node as:dtos(<i>dataNode</i>)</code>	Returns the schema-node which is referred to by the given data-node or the data-node described by the given path (which must be a valid SHORT-NAME-path). "dtos" stands for DataTOSchema. <table border="1" data-bbox="635 1336 1429 1560"> <thead> <tr> <th>Example</th><th>Result</th></tr> </thead> <tbody> <tr> <td><code>as:dtos( '/AUTOSAR/ Module-Conf' )</code></td><td>The corresponding module-definition.</td></tr> <tr> <td><code>as:dtos( /path/to/ <i>dataNode</i> )</code></td><td>The corresponding module-definition.</td></tr> </tbody> </table>	Example	Result	<code>as:dtos( '/AUTOSAR/ Module-Conf' )</code>	The corresponding module-definition.	<code>as:dtos( /path/to/ <i>dataNode</i> )</code>	The corresponding module-definition.
Example	Result						
<code>as:dtos( '/AUTOSAR/ Module-Conf' )</code>	The corresponding module-definition.						
<code>as:dtos( /path/to/ <i>dataNode</i> )</code>	The corresponding module-definition.						
<code>nodeset as:stod(<i>schemaPath</i>)</code> <code>nodeset as:stod(<i>schemaNode</i>)</code>	Returns the data-nodes which are referencing the given schema-node or the schema-node described by the given path (which must be a valid SHORT-NAME-path). "stod" stands for SchemaTODData. <table border="1" data-bbox="635 1718 1429 1897"> <thead> <tr> <th>Example</th><th>Result</th></tr> </thead> <tbody> <tr> <td><code>as:stod( '/AUTOSAR/ ModuleDef' )</code></td><td>All module-descriptions whose definition-path are pointing to the current schema-node.</td></tr> </tbody> </table>	Example	Result	<code>as:stod( '/AUTOSAR/ ModuleDef' )</code>	All module-descriptions whose definition-path are pointing to the current schema-node.		
Example	Result						
<code>as:stod( '/AUTOSAR/ ModuleDef' )</code>	All module-descriptions whose definition-path are pointing to the current schema-node.						



Function	Description	
	Example	Result
	as:stod( /path/to/ schemaNode )	All module-descriptions whose definition-path are pointing to the current schema-node.
<code>string as:name(node)</code>	The SHORT-NAME of the given node. This might be different from the result of <code>node:name</code> . This might happen when the user manually changes the SHORT-NAME or the node is imported with another SHORT-NAME. Also note that there are some nodes that do not have a SHORT-NAME at all (e.g. lists and variable data nodes).	
Example	Result	
as:name( /top/.../.../MyContainer )	The SHORT-NAME for the specified container (might be different from <i>MyContainer</i> ).	
<code>string as:path(node [, boolean])</code>	The SHORT-NAME-path for the given node. If the second parameter evaluates to true, the resulting string includes the prefix <code>AS-Path:</code> prefix.	
Example	Result	
as:path( /top/Module2 )	The SHORT-NAME-path for Module2.	
as:path( /top/Module2 , true() )	The SHORT-NAME-path for Module2, prefixed with <code>AS-Path:</code> .	
<code>strlist as:paths(nodeset [, boolean])</code>	The SHORT-NAME paths for the given nodeset. If the second parameter evaluates to true, the resulting strings include the prefix <code>ASPath:</code> prefix.	
Example	Result	
as:paths( //Var )	The SHORT-NAME paths for all nodes named <code>Var</code> , without prefix.	
as:paths( //Var , true() )	The SHORT-NAME paths for all nodes named <code>Var</code> , prefixed with <code>ASPath:</code> .	
<code>nodeset as:modconf( modDefName )</code>	Returns the module-configurations which referred to a module-definition with the given name.	



Function	Description	
	Example	Result
	as:modconf( 'Rte' )	All module configurations for the module definition Rte.
<i>string as:uuid()</i>	Returns a new random uuid.	
	Example	Result
	as:uuid()	A random uuid e.g.: 05563-1234-343434-34322
<i>Boolean as:ispostbuildchangeable(dataPath)</i>  <i>Boolean as:ispostbuildchangeable(dataNode)</i>	Returns true if the input points to a container data node which has the <i>postBuildChangeable</i> flag set to true in its schema.	
<i>Boolean as:ismcc(dataPath)</i>  <i>Boolean as:ismcc(dataNode)</i>  <i>Boolean as:ismcc()</i>	Returns true if the input (or the context-node respectively) points to a container data node which has the <i>multipleConfigurationContainer</i> flag set to true in its schema.	
<i>nodeset as:fromcurrentmcc(dataPath)</i>  <i>nodeset as:fromcurrentmcc(dataNode)</i>	<p>Returns the equivalent nodes from the <i>current</i> Multiple Configuration Container (MCC). Those are the nodes which have the same SHORT-NAME-path as the given ones, except the SHORT-NAME-path of the MCC. If there is no <i>current</i> MCC configured, the given nodes are returned instead.</p> <p>For an explanation of the term current MCC, see <a href="#">Section 5.8, “Post-build loadable support via Multiple Configuration Containers”</a>.</p>	

## 6.4. Custom XPath Functions API

To register custom XPath functions, two steps have to be done.

- ▶ Register the extension point `dreisoft.tresos.datamodel2.api.plugin.xpathregistration`
- ▶ Implement the Java class that inherits the XPath functions.

How this can be done will be explained in [Section 6.4.1, “Registering custom XPath functions”](#). See also the demo plugin `PublicApiTest_TS_T00D0M0I0R0`.

### 6.4.1. Registering custom XPath functions



- ▶ Open the plugin.xml of your plug-in and switch to the *Extensions*-tab
- ▶ Click *Add* and select the *dreisoft.tresos.datamodel2.api.plugin.xpathregistration* extension-point
- ▶ On the right-hand side enter a namespace under which the xpath functions shall be called.
- ▶ Create a class by clicking on *class\**: or *Browse...* to find an already implemented XPath class. How to create an XPath class will be explained in [Section 6.4.2, “Implementing custom XPath functions”](#).

**TIP**

Reserved namespaces are: as, bit, ecu, lic, node, num, text, var. These namespaces *must NOT* be entered, as custom namespaces with one of these values will be ignored.



The screenshot shows the 'Extensions' tab in the EB tresos Studio interface. The left pane displays a tree view of extension points under 'All Extensions'. One specific extension point, 'xpathregistration', is selected and highlighted with a red box. To its right, the 'Extension Element Details' pane shows configuration fields for this point. The 'class\*' field is set to 'test.tresos.plugin.xpath.MyXPathFunctionClass' with a 'Browse...' button. The 'namespace\*' field is set to 'demo'. The 'plugin:' field is empty. The 'description:' field contains the text 'This is a demo xpath function class. Usable in Code Templates'. At the bottom of the window, the tabs 'Overview', 'Dependencies', 'Runtime', 'Extensions' (which is highlighted), 'Extension Points', 'Build', 'MANIFEST.MF', 'plugin.xml', and 'build.properties' are visible.

## 6.4.2. Implementing custom XPath functions

A class that inherits custom XPath functions must extend the `dreisoft.tresos.datamodel2.api.model.xpath.AbstractXPathFunctions`. This class provides helper method for the conversion of Objects or to get return-values that are suitable for the JXPath engine.

Each public static method in the registered class is accessible in an XPath expression by calling `<namespace>:FunctionName`. Each parameter or return-value must be of the type `Object!` Each XPath function must throw an `APIInvalidOperationException`.

The following example is taken from the `test.tresos.plugin.xpath.MyXPathFunctionClass`.

```
public class MyXPathFunctionClass extends AbstractXPathFunctions {
```



```

public MyXPathFunctionClass() {
}

/*
 * This function is used in the MyTestModule.xdm to execute an enable check
of a node.
 * See the Examples/Float node
*/
public static Object MyTestXPathFunction( CurrentContext context, Object o )
throws APIInvalidOperationException {
DCtxt ctxt = context.getDCtxt();
DCtxt node = toDCtxt( o );
if( node.var.getBool() ) {
    return getReturnValue( context, true );
} else {
    return getReturnValue( context, false );
}
}
}

```

The example's method signature means, that there is one function registered for the namespace demo, accessible through demo:MyTestXPathFunction( String xpath). As you might have noticed there is a CurrentContext passed to the function as well. This context is created by the XPath engine itself and does not mean that it has to be passed by the caller. So MyTestXPathFunction only expects only *one* parameter. As the implementer knows which type the parameters should have, they can be converted to the correct Object type using the helper methods:

- ▶ toDCtxt(Object param)
- ▶ toInt(Object param)
- ▶ toFloat(Object param)
- ▶ toString(Object param)
- ▶ etc. ...

The DCtxt on which the function was initially called can be acquired by calling `context.getDCtxt()`.

To return a value, the method

```
getReturnValue( CurrentContext context, Object returnValue )
```

should be used.



---

**NOTE****Thread-safe cache**

If there are often called expensive calculations inside your XPath function which you want to cache, take a look at class `dreisoft.tresos.datamodel2.api.model.Cache<T>`. It provides an easy way to create a thread-safe cache which is automatically re-calculated on model change.

---

## 6.5. ECU Resource Manager API

### 6.5.1. Purpose

EB tresos Studio provides a feature called ECU Resource Manager. The ECU Resource Manager allows to register multiple values for ECU resources like RAM, ROM, etc. Each resource value is bound to an ECU type. The type is primarily identified by target and derivate. Additionally, it may depend on a configured module or arbitrary configuration values, e.g. a subderivate identifier.

The ECU Resource Manager provides XPath functions to query these resource values. So it can be used by code templates or XDMs that apply to several derivates or subderivates of the same target but require specific configuration values in detail.

---

**TIP**

A complete example of how to use the ECU Resource Manager can be found in the demo plug-in `PublicApiTest_TS_T00D0M0I0R0` which is located in `<tresos-install-dir>/demos/Studio`. This demo plug-in can be imported to your eclipse installation by right-clicking in the Package Explorer and select *Import... -> Import Existing Project Into Workspace*

---

### 6.5.2. Registering ECU resource properties

ECU resources are described in the form of Java properties files. These files store key/value pairs in the following form:

```
<key>:<value>
```

The key must be unique. To create multi-line values, quote the line end with a trailing backslash. Leading white space of continued lines is ignored. Comments can be introduced by placing a '#' at the beginning of a line.



A snippet from a resource properties file could look like this:

```
# ECU resources for V850EGP1
Ram:512
Flash:2048
Rom:128
Port1.Func:Can,Dio
```

An ECU resource properties file must be placed inside an Eclipse plug-in (usually a module plug-in) that is installed into the `plugins` directory of EB tresos Studio.

To enable the ECU Resource Manager for a plug-in the `dreisoft.tresos.autosar2.api.plugin` must be added to the list of required plug-ins. Therefore change the `META-INF/MANIFEST.MF` of the plug-in as follows:

```
Manifest-Version: 1.0
...
Require-Bundle: dreisoft.tresos.autosar2.api.plugin,
...
```

The ECU resource properties file must be registered at the extension point `dreisoft.tresos.autosar2.api.plugin.ecuresource` in the `plugin.xml`. The `target` and `derivate` attributes are mandatory and used to filter the list of registered properties files to the current project respectively the module configurations. The mandatory `file` attribute defines path to the properties file relative to the plug-in's root folder.

`Target` and `derivate` may be set to `"*"` which means, that the resource properties file is registered for all targets and derivates. In this case, the `moduleId` is mandatory.

The optional attribute `moduleId` defines the module for which these properties are mainly registered for. Normally this will be the module defined by the same plug-in.

The `moduleId` attribute was first introduced in EB tresos Studio 2009.a and is recommended to use from that version on. When a module is defined it doesn't mean that these ECU resource properties are only available for that module. But these properties are only available if the current project respectively the module configurations contain at least one module configuration of the module defined by this attribute.

**NOTE**

Before EB tresos Studio 2009.a it was recommended to register exactly one ECU resource properties file that contains information for any module of the same target and derivate because the first relevant properties file was returned on a request and any other registered properties file was not available. Therefore, it was recommended to deliver the properties file within a single *resource* plug-in that was unique for one target/derivate.

With introducing the `moduleId` attribute in EB tresos Studio 2009.a also a merge strategy for ECU resource properties files has been implemented. Any properties that are registered using the new `moduleId` attribute are merged on any request. It is therefore recommended to deliver an ECU resource properties file, which contains only module-specific data, for each module that requires ECU resources. Make sure that the key of an ECU resource property is still unique after merging, e.g. by prefixing the keys with the module's ID.

To stay downward-compatible, older registrations that do not define a module are not merged and behave in the old way. Nevertheless if both registrations with and without attribute `moduleId` are found, then the registrations with `moduleId` are prioritized.

---

The following example shows a typical ECU resource properties registration for target V850 and derivate V850EGP1 for module Can in the `plugin.xml` of the module's plug-in:

```
<extension point="dreisoft.tresos.autosar2.api.plugin.ecuresource">
<ecuresource
    target="V850"
    derivate="V850EGP1"
    moduleId="Can_TS_T16D4M2I1R0"
    file="resources/V850_V850EGP1.properties">
</ecuresource>
</extension>
```

### 6.5.2.1. Advanced parameters

The extension point `dreisoft.tresos.autosar2.api.plugin.ecuresource` supports optional parameter tags to add more search criteria to an ECU resource properties file than the mandatory `target` and `derivate`. These parameters must correspond to parameters of registered ECU Resource Finders. Read more about ECU Resource Finders below.

```
<extension point="dreisoft.tresos.autosar2.api.plugin.ecuresource">
<ecuresource
    target="V850"
    derivate="V850EGP1"
    moduleId="Can_TS_T16D4M2I1R0"
    file="resources/V850_V850EGP1.properties">
```



```

<parameter name="subderivate" value="ERAY" />
</ecuresource>
</extension>

```

### 6.5.3. ECU Resource Finder

It is not intended to access an ECU resource properties file directly. The ECU Resource Manager uses so called ECU Resource Finders to resolve the relevant resource properties files for the configuration that wants to access a resource property. An ECU Resource Finder has to implement interface `dreisoft.tresos.autosar2.api.ecuresource.IEcuResourceFinder`. It neither depends on a special resource properties file nor on the plugin that registered it.

It is recommended that each plugin that registers an ECU resource properties file also registers an ECU Resource Finder at extension point `dreisoft.tresos.autosar2.api.plugin.ecuresourcefinder`. You may write your own implementation of interface `IEcuResourceFinder`, but the functionality provided by the default implementation `dreisoft.tresos.autosar2.api.ecuresource.DefaultEcuResourceFinder` works for most cases. The following snippet registers a `DefaultEcuResourceFinder` at the `plugin.xml` of the plugin that wants to use the ECU Resource Manager:

```

<extension point="dreisoft.tresos.autosar2.api.plugin.ecuresourcefinder">
  <ecuresourcefinder>
    <finder
      class="dreisoft.tresos.autosar2.api.ecuresource.DefaultEcuResourceFinder"/>
  </ecuresourcefinder>
</extension>

```

#### 6.5.3.1. Advanced parameters

An ECU Resource Finder neither depends on an ECU resource properties file nor on a specific plugin or target (see [Section 6.5.3, “ECU Resource Finder”](#) for detailed information). Therefore any registered ECU Resource Finder is able to find any registered ECU resource properties file. Thus it is essential to the implementation of the interface `IEcuResourceFinder` to restrict the internal logic as much as possible to stay out of the way of other finders.

For this reason the extension point `dreisoft.tresos.autosar2.api.plugin.ecuresourcefinder` supports optional parameter tags, similar to the ECU resource properties extension. These parameters help to define further details for retrieving the relevant properties files than just the mandatory ones. Mandatory details are: `target` and `derivate`. If a `moduleId` is additionally defined then the properties are only relevant for configurations that contain a module configuration of that module.



The `DefaultEcuResourceFinder` only accepts parameter names starting with `name` (the name-parameter) and `path` (the path-parameter); both end with an arbitrary suffix. For any name-parameter one path-parameter must exist, so that one name-parameter and one path-parameter make a pair. You must not define a parameter twice within the same finder registration.

The value of the path-parameter must be a schema path. This schema path will be evaluated by the current configuration upon any request to the finder.

### 6.5.3.2. Default ECU Resource Finder Implementation

The `DefaultEcuResourceFinder` implementation is very restrictive. For a better understanding, which resource is returned on a particular request, it is good to know about the internal logic. This internal logic is explained in the following:

First of all the registered ECU resource properties files are filtered by `target`, `derivate` and `moduleId`. The requested values are automatically taken from the current project respectively the module configurations that form the context of the caller.

The number of name- and path-parameter pairs of a `DefaultEcuResourceFinder` registration must match the number of optional parameters of the registered ECU resource properties extension. If the `DefaultEcuResourceFinder` is registered without parameters, it will only consider ECU resource properties files registered without parameters.

Then the `DefaultEcuResourceFinder` checks each name-/path-parameter pair.

For each name- and path-parameter pair the `DefaultEcuResourceFinder` checks if a parameter exists for the registered ECU resource properties file with the value of the name-parameter as name.

If a corresponding parameter exists in the ECU resource properties extension for all name- and path-parameter pairs, then the values for these parameters are verified. The `DefaultEcuResourceFinder` retrieves the configuration value for the specified schema path of the path-parameter. The `DefaultEcuResourceFinder` compares this value with the value of the corresponding parameter value of the ECU resource properties extension. If all values are equal to the values of the current configuration, the ECU resource properties file matches the requested criteria.

#### NOTE



The properties are returned to the caller, only if the number and values of the parameters associated with an ECU resource properties file match the number and values of the name- and path-parameter pairs defined in the `DefaultEcuResourceFinder` registration.

---

```
<extension point="dreisoft.tresos.autosar2.api.plugin.ecuresource">
<ecuresource
```



```

target="MB91"
derivate="MB91460"
moduleId="Com_TS_T19D1M2I1R0"
file="resources/MB91_MB91461.properties">

<parameter name="subderivate" value="MB91461" />
</ecuresource>
</extension>

<extension point="dreisoft.tresos.autosar2.api.plugin.ecuresourcefinder">
<ecuresourcefinder>
<finder
class="dreisoft.tresos.autosar2.api.ecuresource.DefaultEcuResourceFinder">
<parameter name="name" value="subderivate" />
<parameter name="path"
value="/AUTOSAR/TOP-LEVEL-PACKAGES/TS_T/ELEMENTS/Base/SubDerivate" />
</finder>
</ecuresourcefinder>
</extension>

```

The above registration fetches the subderivate from the configuration parameter SubDerivate of the module Base and compares it to the subderivate parameter of the ECU resource properties parameter subderivate shown above.

There are no limitations on the number of parameters in the ECU resource properties and finder extensions.

```

<extension point="dreisoft.tresos.autosar2.api.plugin.ecuresource">
<ecuresource
target="MB91"
derivate="MB91460"
moduleId="Det_TS_T19D2M3I0R0"
file="resources/MB91_MB91461.properties">

<parameter name="subderivate" value="MB91462" />
<parameter name="version" value="3.0" />
</ecuresource>
</extension>

<extension
point="dreisoft.tresos.autosar2.api.plugin.ecuresourcefinder">
<ecuresourcefinder>
<finder
class="dreisoft.tresos.autosar2.api.ecuresource.DefaultEcuResourceFinder">
<parameter name="nameDerivate" value="subderivate" />
<parameter name="pathDerivate"
value="/AUTOSAR/TOP-LEVEL-PACKAGES/TS_T/ELEMENTS/Base/SubDerivate" />

```



```

<parameter name="nameVersion" value="version" />
<parameter name="pathVersion"
value="/AUTOSAR/TOP-LEVEL-PACKAGES/TS_T/ELEMENTS/Base/Version/SpecVersion" />
</finder>
</ecuresourcefinder>
</extension>

```

## 6.5.4. ECU Resource XPath Functions

To access the content of ECU resource properties via XPath, use the following functions:

- ▶ `ecu:has (<key>)`: Checks if the found ECU resource properties contain a key with the given name, e.g. calling `ecu:has ('Rom')` on the resource properties snippet above would return `true`.
- ▶ `ecu:get (<key>)`: Returns the value of the given key name in the found ECU resource properties or `null` if no value has been found, e.g. calling `ecu:get ('Rom')` on the resource properties snippet above would return `128`.
- ▶ `ecu:list(<key>)`: Splits the value of the given key name at commas in the value and returns a node-set. Calling `ecu:list('Port1.Func')` on the ECU resource properties snippet above would return a node-set containing the 2 values `Can` and `Dio` (unlike `ecu:get ('Port1.Func')` which would return the pure value `Can, Dio`).

### NOTE



If there is no matching ECU resource properties file available (i.e. target and derivate as well as the `<enablement>` must match), using the functions above will produce an error. You should make sure that there always is a matching ECU resource file if you are using these functions.

If this problem occurs within a xdm schema file, the errors will be listed in the **Error Log** view and the corresponding automatic attributes will not be calculated. If this problem occurs within a code-generator template, the errors will also be listed in the **Error Log** view and code generation will fail.

The XPath function `ecu:parameters(String)` reads the values of the given parameter name between all registered ECU resource properties that match the target, derivate and optionally moduleId of the project (but not the optional parameters themselves). A common use-case for this function would be to use the return-values to fill a combo box with possible subderivates in the editor. To define further subderivates only requires you to register further ECU resource properties files using this approach.

```
ecu:parameters(<parameter-name>)
```

In the above example the following call



```
ecu:parameters('subderivate')
```

would return a list containing MB91461 and MB91462.

## 6.5.5. Characteristics when using the legacy commandline

The EB tresos Studio legacy commands often require you to specify additional parameters when working with module configuration files, which use ECU Resource properties. For information on the legacy commands see the EB tresos Studio user's guide, chapter [Working with the commandline](#).

### 6.5.5.1. Defining parameter values for the DefaultEcuResourceFinder

If you are using the `DefaultEcuResourceFinder` with advanced parameters (see [Section 6.5.3, “ECU Resource Finder”](#)) you must specify the values of these parameters explicitly because there is no module configuration to evaluate when converting the schema file. The parameter value can be specified by a Java system property with the same name.

---

**TIP**


Specify any necessary parameter value of the `DefaultEcuResourceFinder` as it would evaluate in a possible configuration for the derivate you want to convert. A Java system property is defined on the EB tresos Studio command line with `-D<parameter-name>=<parameter-value>`, e.g.

`-Dsubderivate=ERAY.`

Note, that this only works when you convert a schema file to the AUTOSAR format via `legacy convert`. If you execute another command (e.g. `legacy generate`) and the parameter value is specified as a path to a configuration parameter, you need to ensure that this configuration parameter is available in one of the given files.

---

### 6.5.5.2. Defining configured modules for the DefaultEcuResourceFinder

If the ECU Resource properties are qualified with the attribute `moduleId` at the ECU Resources extension (see [Section 6.5.2, “Registering ECU resource properties”](#)), the `DefaultEcuResourceFinder` needs to know which modules are configured to respect these ECU Resources in its finding process. Normally these modules are defined by a configuration project. Since the legacy commandline does not handle configuration projects, you need to specify the relevant modules explicitly using the Java system property `EcuResourceModuleIds`. Specify a comma- or semicolon-separated list of moduleIds to be considered as configured modules.



If the moduleIds are not explicitly specified by the Java system property then the `DefaultEcuResourceFinder` will consider any installed module to be a configured module when executing the legacy command.

An empty value for the system property means that no modules are configured. In this case ECU Resources with attribute `moduleId` are ignored by the API.

**TIP**


Specify any necessary module ids to be considered as configured modules for converting your schema file. A Java system property is defined on the EB tresos Studio command line with `-D<parameter-name>=<parameter-value>`, e.g.

```
-DECuResourceModuleIds=Com_TS_T99D01M01I05R01;Rte_TS_T99D01M01I04R02.
```

```
-DECuResourceModuleIds=.
```

#### 6.5.5.3. Converting a module schema with ECU Resources to AUTOSAR

In general, a module schema written in EB tresos Studio format (normally `.xdm`) that uses ECU Resources XPath functions is not convertible into an AUTOSAR format (e.g. `.epd`) without loss of information. The AUTOSAR format does not provide functionality to keep information for multiple derivates. Nevertheless, it is possible to extract a derivate-specific AUTOSAR file; so the source schema file may result in multiple AUTOSAR schema files when converting it for each derivate or sub-derivate.

## 6.6. Resources API

The Resources API allows to load, store and convert configuration data persistently from and to files and to merge data. It provides functionality similar to the **legacy convert** and **import** commands of the command line interface as a Java API and consists of the following classes:

- ▶ `dreisoft.tresos.datamodel2.api.model.DCTxt`

This class is the central access point to configuration data models. It can also merge data models.

- ▶ `dreisoft.tresos.datamodel2.api.modelRCTxt`

This class is an extension of a `DCTxt` that adds loading and storing functionalities.

- ▶ `dreisoft.tresos.datamodel2.api.model.Location`

Use this class to specify files and their content types in an `RCTxt`.

- ▶ `dreisoft.tresos.autosar2.api.model.importer.ImporterConstants`



This class contains constants you may use to define options where applicable.

For sample code, see the demo plugin `ResourceAPIDemo`. To start the demos, open the Sidebar wizard **Demos/ResourceAPIDemo Dialog** in EB tresos Studio.

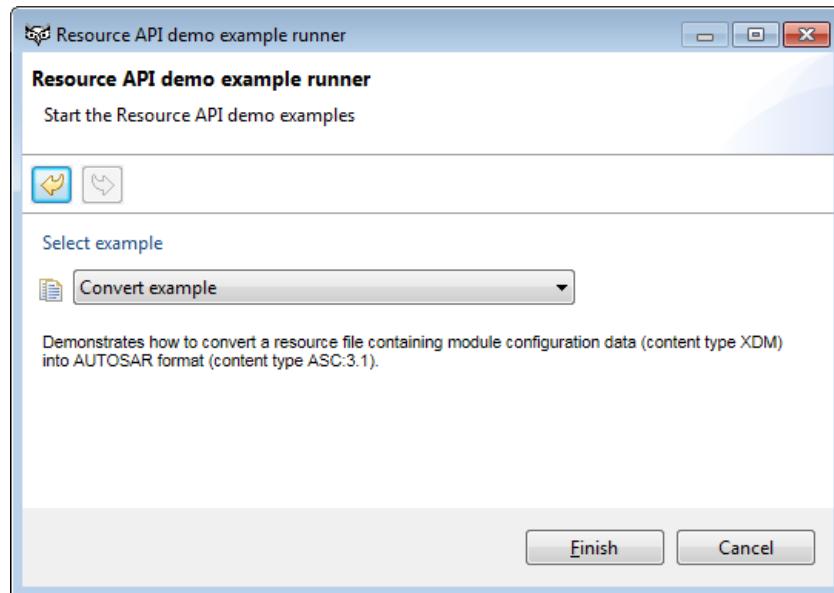


Figure 6.62. Running the `ResourceAPIDemo` examples

## 6.7. System Model Access API

The System Model Access API provides methods to access the system model of a configuration project. It allows to retrieve or modify its entities. The System Model Access API is also known as the tresosDB API. The following classes are the main entry points of this API:

- ▶ `dreisoft.tresos.tresosdb.api.model.DataModelAccess`  
Provides methods to access the tresosDB from within a `DataModel`.
- ▶ `dreisoft.tresos.tresosdb.api.model.ProjectAccess`  
Provides methods to access the integration of TresosDB models into tresos Studio projects.
- ▶ `dreisoft.tresos.tresosdb.api.model.v2.factory.TresosDBFactory`  
Provides methods to create instances of the interfaces defined in the subpackages under `dreisoft.-tresos.tresosdb.api.model.v2`.

See the Java documentation of the plugin `dreisoft.tresos.tresosDB.api.plugin` for more information.



## 6.7.1. System Model Access Demo

For sample code, see the demo plug-in `SystemModelAccessDemo`, which is located in the folder `<tresos-install-loc>/demos/Studio/`. To import the demo into your Eclipse workspace, follow the instructions in [Section 4.9, “How to install a Public API demo plugin”](#). The Java code which demonstrates the System Model Access API is located in the class `dreisoft.tresos.tresosdbapi.demo.SystemModelAccessDemoBackend`.

To run the demo code, make sure that the demo plug-in is part of your run configuration, as described in [Section 4.7, “How to set up a run configuration”](#). The demo requires an arbitrary configuration project with a system model to run. You can import the project from `<tresos-install-loc>/demos/Studio/SystemModelAccessDemo/project` into your EB tresos Studio workspace for that purpose.

Configure and run the unattended wizard **SystemModelAccessDemo** to execute selected parts of the demo code. For more information about how to configure and run an unattended wizard, see the EB tresos Studio user's guide, chapter *Autoconfiguring routine jobs*.

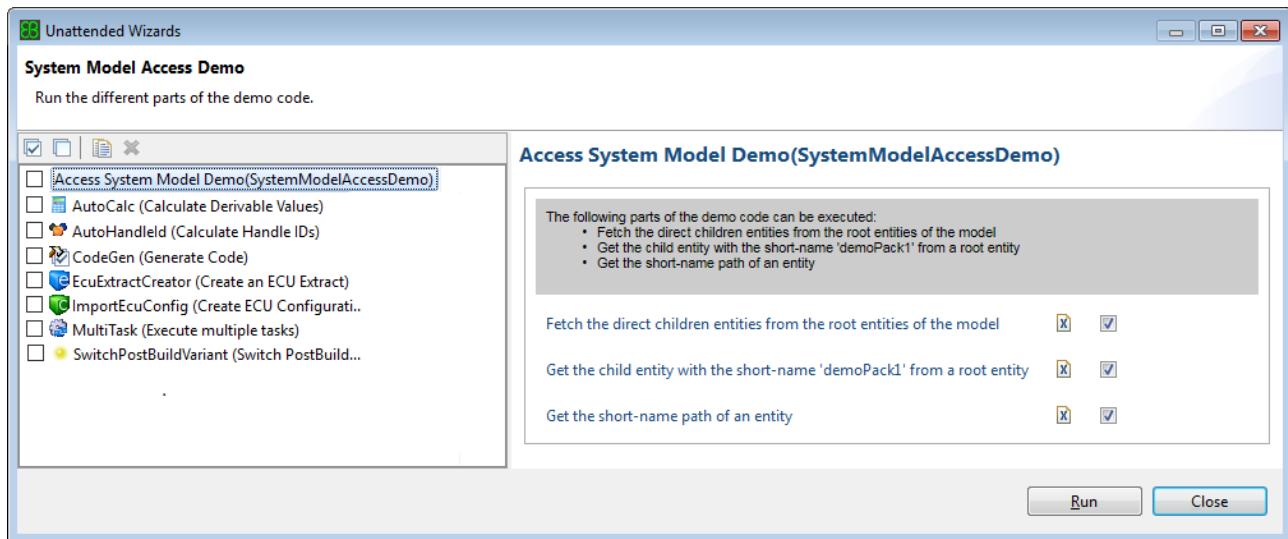


Figure 6.63. `SystemModelAccessDemo` unattended wizard

## 6.8. Custom Attribute API

To register your own custom attribute:

- ▶ Register a contribution to the extension point `dreisoft.tresos.datamodel2.api.plugin.customattribute`
- ▶ Optional: implement your own handler class.

How this can be done will be explained in [Section 6.8.1, “Registering a custom attribute”](#).



## 6.8.1. Registering a custom attribute

- ▶ Open the plugin.xml of your plug-in and switch to the *Extensions*-tab
- ▶ Click *Add* and select the `dreisoft.tresos.datamodel2.api.plugin.customattribute` extension-point
- ▶ On the right-hand side enter a unique ID and provide a user-readable label of the attribute.
- ▶ For more options, see the description of the extension point. See also [Section 4.5, “How to open extension point descriptions”](#).
- ▶ Optional: customize the behavior of the default handler class implementation by providing parameters, or provide a custom handler class implementation which needs to extend `dreisoft.tresos.datamodel2.api.customdata.DefaultHandler`.

With a custom handler class, you can e.g. provide specific cell editors to be used for editing attribute values within the **Properties** view. For more information about parameters of the default handler class and how to extend the handler class, see the **Public API Java Documentation** of the class `dreisoft.-tresos.datamodel2.api.customdata.DefaultHandler`.

The screenshot shows the 'Extensions' tab in the EB tresos Studio interface. The left panel displays a tree view of extensions under 'All Extensions'. A selected node is 'Tags (attribute)' under 'dreisoft.tresos.datamodel2.api.plugin.customattribute'. The right panel shows 'Extension Element Details' for this node. The configuration includes:

Property	Value
<b>id*</b> :	tags
<b>label:</b>	Tags
<b>description:</b>	For supplying specific tags for nodes.
<b>allowEmptyValues:</b>	disabled
<b>allowMultipleValues:</b>	true
<b>inherited:</b>	disabled

Below the main panels, a navigation bar includes links for Overview, Dependencies, Runtime, Extensions, Extension Points, Build, MANIFEST.MF, plugin.xml, and build.properties.



# 7. Generating code

Some generators execute external commands or batch files. For those generators it might be interesting to see the console output. The **Console** view is not visible by default. To display the view, access **Window -> Show View -> Other -> General** and select **Console**.

To obtain more detailed information you can additionally start EB tresos Studio with the `-debug` option. For further information on the debug option, see the [Eclipse help](#).

## 7.1. General concept

### 7.1.1. Generating module-independent code

You can define a code generator to be module-independent because it does not generate code for a specific module. Such a generator always runs no matter which modules you select for code generation.

You can use module-independent generators only for pre-generation and post-generation steps. In the pre-generation step they run before all module-bound generators run. And in the post-generation step they run after all module-bound generators run.

You specify a module-independent generator by omitting the attribute `moduleId` at the generator extension point:

```
<%@ jet package="xyz.jet.generated"
    class="TheNumbersGame" %>
<%
    for( int i = 1; i <= 10; i++ ) {
%
    This is round <%=Integer.toString( i )%>.

<%
    }
%
<generator <!-- no "moduleId" attribute here-->
    id="Lin_TS_T17D12M1I1R0GeneratorId"
    step="pre"
    class="dreisoft.tresos.autosar2.generator.EPCFileGenerator">
    ...
</generator>
```



## 7.1.2. Specifying generation steps

You can influence the order in which code generators are called by specifying a step. The steps are `pre`, `default` and `post` generation where `default` is used if you do not define a step. For details about the execution order of code generators, See [Section 7.1.4, “Order of execution”](#).

If you want to specify the step for a code generator, you must define the attribute `step` at the generator extension point. Each generator runs in exactly one step.

All generators scheduled to the `pre` generation step run before all generators scheduled to the `default` generation step. You can use pre-generation to calculate data that is used by all code-generators or to prepare the output directory structure.

The `default` step is the main generation phase when code generators shall run. Thus, if you do not explicitly specify the `default` step, all existing generators are assumed to run in the `default` step.

All generators scheduled to the `post` generation step run after all generators scheduled to the `default` generation step. You can use post-generation to generate code based on data that is gathered during the normal generator run like post-build structures (structure sizes) etc.

The following example shows a module dependent pre-generator, which generates an `.epc` file for the `Lin` module only.

```
<extension point="dreisoft.tresos.launcher2.plugin.generator"
          id="EPCGenerator"
          name="EPC Generator">
<generator moduleId="Lin_TS_T17D12M1I1R0"
          id="Lin_TS_T17D12M1I1R0GeneratorId"
          step="pre"
          class="dreisoft.tresos.autosar2.generator.EPCFileGenerator">
    <parameter name="cfgFilePath" value="output"/>
    <parameter name="generateAllModules" value="false"/>
    <parameter name="contentType" value="asc:2.1"/>
</generator>
</extension>
```

This attribute also influences the order in which the generator is called. For details, see [Section 7.1.4, “Order of execution”](#).

## 7.1.3. Defining variant-aware code generators

If your code generator shall run once for each variant, you must add the parameter `variantAware` set to `true` to the generator registration like this:



```
<extension point="dreisoft.tresos.launcher2.plugin.generator"
    id="EcuC_VariantsDemo_Gen_Template">
<generator moduleId="EcuC_VariantsDemo"
    id="EcUC_VariantsDemo_Gen_TemplateId"
    class="dreisoft.tresos.launcher2.generator.TemplateBasedCodeGenerator">
<parameter name="templates" value="templates"/>

    <!-- this generator will be called once for each variant! -->
    <parameter name="variantAware" value="true"/>
</generator>
</extension>
```

You can configure several generators for different steps (pre, default, post). For more information, see [Section 7.1.2, “Specifying generation steps”](#). As for one variant, always the whole block of pre, default and post generators is executed. For more information, see [Section 7.1.4, “Order of execution”](#). You may configure several generators for different steps. For example to rename a generated template file in a post step. There are multiple examples for these use cases shown in the provided demo. For more information, see [Section 5.7.9, “Variant handling demo plug-in”](#).

## 7.1.4. Order of execution

The order in which the code generators are executed depends on several criteria:

- ▶ The module dependence
- ▶ The generation step
- ▶ The variant awareness

The execution order is depicted in the following picture:

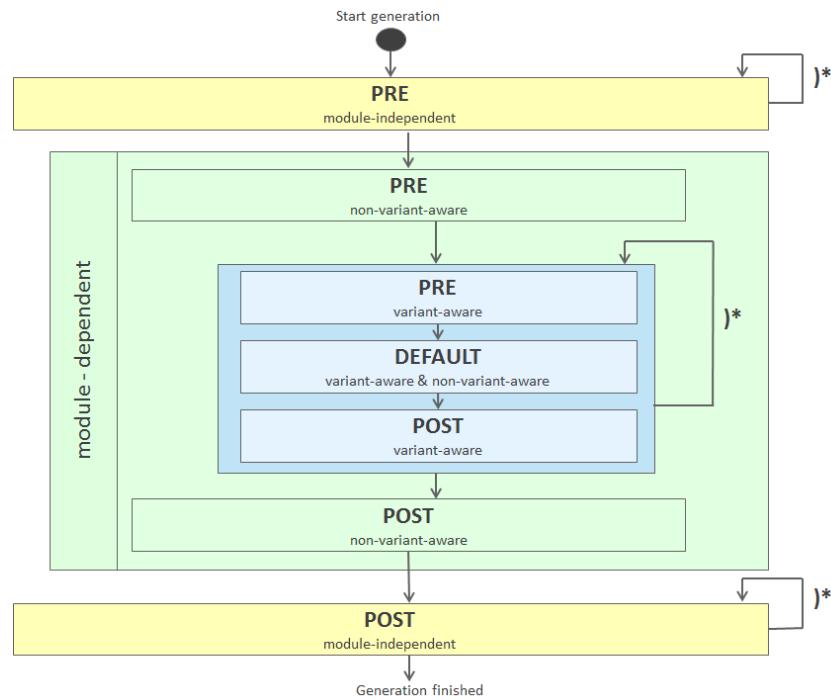


Figure 7.1. Order of execution

)\* All variant-aware generators are called once for each configured selectable variant. Non-variant-aware generators are called only once. As shown in the picture above, the code generators are called in the following order:

- ▶ Step 1:  
`pre, module-independent generators`
- ▶ Step 2:  
`pre, module-dependent, non-variant-aware generators`
- ▶ Step 3:  
`pre module-dependent, variant-aware generators`
- ▶ Step 4:  
`default, module-dependent, variant-aware and non-variant-aware generators`
- ▶ Step 5:  
`post, module-dependent, variant-aware generators`
- ▶ Step 6:  
`post, module-dependent, non-variant-aware generators`
- ▶ Step 7:



post, module-independent, generators

If variants are defined for one or multiple generators, the steps are executed in the following order:

▶ Step 1:

All non-variant-aware generators are called once. All variant-aware generators are called once for each configured selectable variant.

▶ Step 2:

Each generator is called exactly once.

▶ Step 3-5:

This block of steps are called once for each configured selectable variant. Thereby step 4 is executed only once for all non-variant-aware generators.

▶ Step 6:

Each generator is called exactly once.

▶ Step 7:

All non-variant-aware generators are called once. All variant-aware generators are called once for each configured selectable variant.

The provided demo also contains an example of code generators using different steps. For more information, see [Section 5.7.9, “Variant handling demo plug-in”](#).

### 7.1.5. Running generators in parallel

If there are multiple generators configured for one project, EB tresos Studio runs them in parallel if appropriate. Only generators that have the same module dependence and the same generation step may run in parallel. The default whether different generators run in parallel to each other depends on the specific generator API that you use. Unless stated otherwise, you can adjust the defaults for the different generator APIs.

- ▶ Template-based code generators and EPC file generators always run in parallel with other generators.
- ▶ External generators run in parallel to generators of other APIs, but no two external generators are executed in parallel.
- ▶ NG generators run in parallel to generators of other APIs, but no two NG generators are executed in parallel.

You cannot adjust this behavior.
- ▶ Generators that you implemented with the public generator API are not executed in parallel by default. You can however enable your generators in the extension point.



### 7.1.5.1. Prevent generator classes from running in parallel

The file `$TRESOS_BASE/configuration/defaultPreferences.properties` contains preferences to prevent all generators of a certain class to run in parallel. For example you can define that all EPC file generators shall run sequentially.

### 7.1.5.2. Prevent single generators from running in parallel

At each generator registration within the `plugin.xml` file, you can specify the tag `canRunParallel` to define whether the generator may run at the same time as other generators or not.

```
<generator id="MyExternalGenerator"
    moduleId="MyModule"
    class="dreisoft.tresos.autosar2.generator.ExternalGenerator"
    canRunParallel="true">
```

## 7.2. Template-based Code Generator API

The DataModel provides a sophisticated *Code Generator* which takes a code template, enhanced with special macros which refer to variables in the Configuration Data, and which generates code.

The following chapter explains the macro language with which code can be generated dynamically.

### 7.2.1. Concepts

The code generator takes an input stream, performs macro expansion and writes the result in an output stream. All macros used by the code generator are parameterized by the configuration data. The macros can access the data with an XPath expression which allows them to do mathematical evaluations, list and string manipulation, Boolean evaluations and list operations.

The code template read by the code generator is a simple ASCII-file. Macros can be introduced anywhere in the template. Macro-expressions are enclosed by `[!` and `!]` (if not stated otherwise). For example:

```
[ !INCLUDE "test.m"! ]
```



includes the file `test.m`. All macro-expressions from `[! till !]` are eliminated from the generated code.

All code that should be generated must be between the two tags:

```
[ !CODE! ]  
...  
[ !ENDCODE! ]
```

All text before `[!CODE!]` and all text after `[!ENDCODE!]` will not be written in the output stream.

The need for the `[!CODE!]` can be disabled via the Generator-API.

In addition to the `[!CODE!]`-tag the Generator also supports the tags `[!NOCODE!]` and `[!ENDNOCODE!]` which can be used inside the `[!CODE!]`-tag to temporarily disable code printout. Only variables and macros are evaluated inside these tags.

The code generator supports a special line quoting which allows to join lines before macros are evaluated. A line end can be quoted with `!//` at the end of the line.

### 7.2.1.1. Addressing nodes using XPath

Several generator-commands take xpath-expressions which can be used to address configuration-nodes. For this, the correct XPath pointing to the node must be entered. As described in chapter [Section 5.2.3.1, “Path in Template”](#), the EB tresos Studio configurator provides a view showing the complete XPath to the selected configuration-node within the current editor.

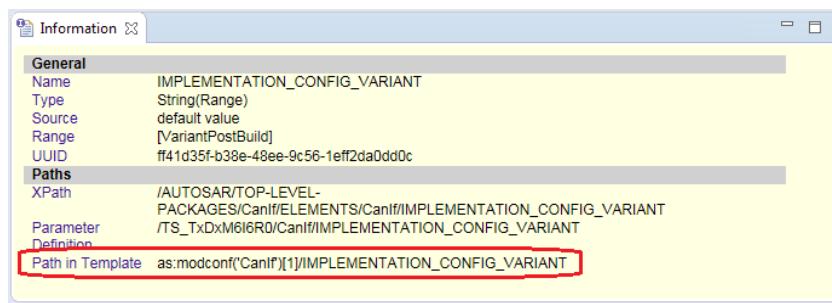


Figure 7.2. Path in template

### 7.2.1.2. Identical generated files

The generator will never overwrite an existing file if the contents do not change. For this, a comparison on a per-line basis is done before writing to the target-file. To disable the comparison for single lines only, the line must contain the following keyword (with exclamation marks and capital letters):



!!!IGNORE-LINE!!!

There may be any other text before and after this keyword.

## 7.2.2. Codetemplate console

To use the **Codetemplate console**, select the preference **Show actions 'XPath console' and 'Codetemplate console'** in **Outline view**. For instructions about how to change preferences, see the EB tresos Studio user's guide, chapter **Setting Developer features preferences**.

You can open the **Codetemplate console** from the context-menu of a node within the **Outline view** by choosing "Codetemplate console". The selected node will be used as context-node for all executed templates.

The path of the context-node is displayed above the check boxes. The part of the path that points to the MODULE-CONFIGURATION is displayed as a hyperlink. To set the MODULE-CONFIGURATION as context-node, click on the hyperlink.



Figure 7.3. Codetemplate console



Type or paste any code template into the upper text area and click "Run" to evaluate it. The result will be shown in the lower text area.

### 7.2.3. Comments

The simplest macro expression is the comment. There are two forms of comments:

- ▶ Line Comments
- ▶ Multi-line comments

Line comments start with `!//` and expand to the end of the line. All text starting with `!//` is not written in the output file.

If a line ends with the sequence `!//` the line termination (linefeed/ctrl-linefeed) is quoted.

A multi-line comment starts with `!/*` and ends with `*/!`. A Multi-line comment can span multiple lines. All text from `!/*` to `*/!` is removed from the output file.

```
[ !/*
multiline-comment
*/!]
... [!// line-comment
```

### 7.2.4. Evaluating variables and expressions

Variables from the configuration data and XPath expressions can be evaluated with:

```
[ ! "<Expression>" ! ]
```

The Expression is an XPath expression evaluated on the configuration so that all variables in the configuration data tree can be accessed.

### 7.2.5. Including other files

Other files can be included with the include statement:



```
[ !INCLUDE "<Filename>"! ]
```

The include expression is substituted by the contents of the included file. Macro expansion is also performed on the included file.

If the filename starts with a question mark, the rest of the filename is interpreted as an XPath expression which results in the filename to be loaded.

If the include expression is inside an if-expression the file is only included if the if-expression evaluates to true. Likewise an include expression in a loop-expression includes the file each time the loop runs.

## 7.2.6. Conditions

With conditions parts of the code can be excluded from the output stream. There are two types of conditions:

- ▶ if-constructs
- ▶ end-conditions

If-constructs are conditional expressions as known in most programming languages. The syntax is as follows:

```
[ !IF "<Expression>"! ]
...
[ !ELSEIF "<Expression>"! ]
...
[ !ELSE! ]
...
[ !ENDIF! ]
```

The **[!ELSEIF!]**-clause and the **[!ELSE!]**-clause are optional. There can be more than one **[!ELSEIF!]**-clause. The expression used in the **[!IF!]**- and the **[!ELSEIF!]**-clause is an XPath-expression which must result in a Boolean value.

The second form of a condition, the end-condition, is a short form that is used to exclude a single line from the output stream. The end- condition is always at the end of the line:

```
... [ !WHEN "<Expression>"! ]
```

If the XPath expression evaluates to false, the complete line is omitted from the output stream.

---

**WARNING** **If the XPath Expression evaluates to a string value, only 'true' (case-insensitive) is interpreted as true**



This is contrary to other uses of XPath, where any string other than the empty string is true (see [Section 6.3.4.1.2, "Type conversion with Boolean expressions"](#)).

---

## 7.2.7. Loops and selects

The XPath language allows to select a set of nodes, not only one single value.

The code generator supports to iterate over such node-sets. While iterating, the current node pointer is set once to each node of the selected node-set. The syntax of such an iteration is:

```
[!LOOP "<Expression>"!]  
...  
[!ENDLOOP!]
```

For each node in the node-set, the code block (eventually instrumented with macros) between **[!LOOP!]** and **[!ENDLOOP!]** is written to the output stream. For each run through the loop, the current node pointer is set to a new node in the node set. Thus inside the loop relative XPath expressions (in relation to the selected node) can be used.

**NOTE****LOOP over a variable is not possible**

Since variables cannot store node-sets, it is not possible to use a variable as argument for the **[!LOOP!]** command.

**NOTE****No parameter-access within a LOOP over strings**

When you iterate over strings, it is not possible to access any configuration parameter, e.g.:

```
[!LOOP "text:split('1 2 3 4')"]!
[!"."!]          [!// e.g. returns '1'
[!"node:current()"]  [!// e.g. returns '1'
[!"/"!]          [!// e.g. returns '1'
[!"as:modconf('Can')"] [!// not possible
[!ENDLOOP!]
```

Instead, the current node is the part of the splitted string. An alternative is to use the FOR or SELECT loop:

```
[!FOR "i" = "1" TO "4"]!
[!// String a to d
[!"text:split('a b c d', ' ')[position() = $i"]]

[!// accesses the parameter from outside the loop
[!"."!]

[!// accesses the parameter from outside the loop
[!"node:current()"]

[!// works here
[!"as:modconf('Can')"]
[!ENDFOR!]
```

```
[!SELECT "<Expression>"!]
...
[!ENDSELECT!]
```

The select-expression can be used to execute contained XPath-expressions relative to a single (the currently selected) node. Hint: The select-expression behaves identical to the loop.

In addition to looping over a group of nodes the code generator also allows looping over numbers with the for-statement of other languages. The code generator for statement has the following syntax:



```
[ !FOR "<Variable-Name>" = "<From-Value>"  
TO "<To-Value>"! ]  
...  
[ !ENDFOR! ]
```

```
[ !FOR "<Variable-Name>" = "<From-Value>"  
TO "<To-Value>" STEP "<Step-Value>"! ]  
...  
[ !ENDFOR! ]
```

The for-statement steps the values given by the XPath expression **<From-Value>** to the value of the XPath expression given by the **<To-Value>** setting the variable **<Variable-Name>** in each iteration. The value of this variable increases (or decreases) by the **<Step-Value>** in each iteration. The **<Step-Value>** defaults to 1.

The variable **<Variable-Name>** is a local variable bound to the scope of the for-statement.

## 7.2.8. Leaving loops

The following loops can be left by using the BREAK-statement:

- ▶ FOR
- ▶ LOOP
- ▶ SELECT

Example:

```
[ !FOR "x" = "0" TO "10"! ]  
[ !IF "num:i($x) > 3"! ] [ !BREAK! ] [ !ENDIF! ]  
[ !".."! ]  
[ !ENDFOR! ]
```

In the example above, the FOR-loop will be left if the variable "x" is greater than 3.

## 7.2.9. Variables and macros

The code generator allows to define variables inside the code template which can be used in XPath expressions. The variables are always stored and interpreted as string values.

Variables are defined in the following way:



```
[ !VAR "<Variable-Name>" = "<Expression>"! ]
[ !VAR "<Variable-Name>"! ]
<Expression>
[ !ENDVAR! ]
```

The first variable-definition maps a variable name to an XPath expression. The XPath expression is evaluated at the point where the variable is defined and the result is stored as string value to the variable.

In the second variable-definition, the variable value can be set in a much more complex way. The expression in this form can contain other macro expressions. Also in this form the expression is evaluated once at the point where the variable is defined and the result is stored as string value to the variable.

Variables defined in this form can be used in any XPath expression as a variable in the form **\$<Variable-Name>**.

---

**NOTE**
**Variables can only be of type *string***


Variables can only store string values. If you attempt to assign a node-set to a variable, only the first item is stored in the variable.

---

Variables are always defined globally which means they are defined from their first occurrence until the generator finishes the code generation.

Besides the user defined variables in the code template, there exists some variables which are always set and can be used without defining them.

**datetime**

Generator variable storing the current timestamp in format YYYY-MM-DD hh:mm:ss.

**date**

Generator variable storing the current date in format YYYY-MM-DD.

**time**

Generator variable storing the current time in format HH:MM:SS.

**target**

Generator variable storing the target, set by the preferences or the commandline. In legacy mode the target can be set with `-Dttarget=<target>`.

**derivate**

Generator variable storing the derivate, set by the preferences or the commandline. In legacy mode the derivate can be set with `-Dderivate=<derivate>`.

**ecuid**

Generator variable storing the ecuid, set by the preferences or the command line. In legacy mode the ecuid can be set with `-Decuid=<ECU-Id>`.

**variant**

Generator variable that stores the variant name of EB tresos Studio. E.g. 2007.b, 2008.a...

**buildnr**

Generator variable that stores the build nr of EB tresos Studio.

**relversion**

Generator variable that stores the release version of EB tresos Studio.

**tresosBase**

Generator variable storing the directory in which EB tresos Studio is installed.

**pluginPath**

Generator variable storing the absolute path to where the Eclipse plug-in is installed that registers the generator.

**outputDir**

Generator variable storing the generation path.

**generationDir**

Generator variable storing the generation path as stored in the project settings. In legacy mode it is the same as the variable `outputDir`.

**generationPaths**

Generator variable storing the generation path of each module configuration in the same order as in variable `generateConfigs`. Only module configurations that are enabled for code generation are considered.

**generationPathsDistinct**

Generator variable storing the generation paths without any duplicate entries. Module configurations which are not enabled for code generation are not considered.

**project**

Generator variable that stores the name of the project currently processed. In legacy mode this variable can be set with `-Dproject=<Project Name>`.

**projectDir**

Generator variable that stores the absolute path to the project currently processed. May be empty in legacy mode if the variable `project` is not set, or if no such project exists within the workspace.

**module**

Generator variable that stores the module-id of the module currently processed.

**modules**

Generator variable that stores a space separated list of all module-ids processed in this generator or verify run.

**generateModules**

Generator variable that stores a space separated list of all module-ids for which code gets generated in this generator run.

**enabledModules**

Generator variable that stores a space separated list of all module-ids which are enabled in the processed project.

**defaultconfig**

Generator variable that stores the name of the module configuration that should be considered as the default configuration when there are more than one module configurations for one module generated. This is by default the first found module configuration.

**config**

Generator variable that stores the name of the currently generated module configuration when there are more than one module configurations for one module generated.

**configs**

Generator variable that stores the name of all module configurations of the module currently processed.

**generateConfigs**

Generator variable that stores the name of all module configurations of the module currently processed for which code generation is enabled.

**enabledConfigs**

Generator variable that stores the name of all module configurations of the module currently processed that are enabled.

**configClass**

Generator variable that stores the AUTOSAR configuration class of the generator.

**postBuildVariant**

Generator variable that stores the short-name of the currently built post-build variant.

**moduleSpecVer**

Generator variable that stores the specification version of the module currently processed in the form <major>.<minor>.<patch>.

**moduleSoftwareVer**

Generator variable that stores the software version of the module currently processed in the form <major>.<minor>.<patch>.

**moduleReleaseVer**

Generator variable that stores the release version of the module currently processed in the form <major>.<minor>.<patch>.

**moduleType**

Generator variable that stores the module type of the module currently processed.

**moduleLayer**

Generator variable that stores the module layer of the module currently processed.

**moduleCategory**

Generator variable that stores the module category of the module currently processed.

**mode**

Generator variable that stores the generator mode.

**step**

Generator variable that stores the current generation step.



### **template**

Generator variable that stores the template name.

### **session**

Generator variable that stores the session ID of the current generator session.

Besides variables the code generator supports macros. Macros are defined with:

```
[ !MACRO "<Macro-Name>",
  "<Parameter-Name>" = "<Default-Value>", . . . ! ]
[ !ENDMACRO ! ]
```

Macros can have parameters which can be used like variables in XPath expressions inside the macro. All parameters have a name and can have a default value which is used when the parameter is not set in the macro-call. The default-value is always interpreted as an XPath expression. The parameter-name is a normal text string. The macro name is also a normal text string if it does not start with a question mark, in which case the rest of the macro name is interpreted as an XPath expression.

A macro can be called with the following command:

```
[ !CALL "<Macro-Name>",
  "<Parameter-Name>" = "<Parameter-Value>", . . . ! ]
```

The content of the macro is evaluated when it is called and added to the output stream at the position of the call.

Each parameter value is interpreted as an XPath expression. Each parameter-name is a plain text string. Like in the macro definition the macro-name normally is a plain text string if it does not start with a question mark. In this case it is also interpreted as an XPath expression.

All parameters of a macro are interpreted as local variables which means they shadow global variables during the macro call and disappear after the macro finishes.

## 7.2.10. Spaces, newlines and indentation

### 7.2.10.1. Tab stops

Indentation can be controlled with the special variable **\$tabs**. This variable can be used as either a parameter to a macro or as a normal variable. When the Code Generator encounters a percent sign at the beginning of a line, followed by three spaces, this character combination is substituted by a number of spaces equal to **\$tabs**.



```
% intended text
[!"$tabs"]!intended text
```

### 7.2.10.2. Whitespaces

To explicitly write whitespaces, you can use the **WS**-command which optionally gets the number of spaces as parameter:

```
[!WS!]one whitespace
[!WS "2"]!two spaces
[!WS "$tabs"]!spaces as defined in variable tabs
```

---

**NOTE****WS command with variables as argument**

The **WS** command does not re-evaluate its argument if the same **WS** command is executed multiple times. A command can be executed multiple times if it is for instance used in a loop or in a macro. The argument of the **WS** command will only be evaluated the first time when the command is executed. If you used a expression that returns a different value the second time, the **WS** command will not recognize this and use the previous value again.

Although this behavior is counter-intuitive, it is not changed due to compatibility requirements with previous versions of EB tresos Studio. As a workaround, you can achieve a similar behavior with the **INDENT** command.

---

### 7.2.10.3. Newlines

To explicitly write newlines, you can use the CR-command (carriage return):

```
[!CR!]
```

### 7.2.10.4. Auto-spacing

Autospacing allows you to indent your generator-commands within your code-templates without influencing your generated code with the spaces of the indentation and the newlines.

Autospacing can be switched on for each template-file by a single command:



```
[ !AUTOSPACING! ]
```

If switched on it does the following for commands which do not produce output:

- ▶ if there are only spaces in front of the command, they are removed
- ▶ if there are only spaces and a newline after the command, the spaces and the newline are removed

Suppose the following code-template once with and once without autospacing:

```
[ !CODE! ]
[ !LOOP "text:split('1 2')"! ]
    myoutput
[ !ENDLOOP! ]
[ !ENCODE! ]
```

Without autospacing, the following code would be generated (to make spaces and newlines visible, they are printed as "." and "<CR>"):

```
..<CR>
....myoutput<CR>
..<CR>
....myoutput<CR>
..<CR>
<CR>
```

With activated auto spacing, the output would look like this:

```
....myoutput<CR>
....myoutput<CR>
```

**NOTE**

The AUTOSPACING has a number of limitations. Existing code might rely on the currently implemented behavior. Due to backwards-compatibility requirements, fixing these limitations is currently not planned. The known limitations are listed below.

**WARNING**

AUTOSPACING removes intermediate spaces

If autospacing is enabled, spaces in front of a command will be removed also if non-white-space text precedes the whitespace. As a result, important whitespace can be missing in the generated code.

For example, the code `#define [!IF "true()"]one[!ENDIF!]` will yield the result `#defineone`.

As workaround, you can use the command `[!'"' "'!]` or `[!WS!]` instead of spaces. In the example above another workaround is to include the desired space character inside the if, directly before one: `#define [!IF "true()"] one[!ENDIF!]`. This will produce the desired result: `#define one`.

**WARNING**

AUTOSPACING does not work correctly in comments

If you use the command `[!AUTOSPACING!]` in the template-based code generator, code lines that have just a `[!/* comment */!]` are generated as empty lines. As workaround you can quote the line ending with `[!//]`.

**WARNING**

The AUTOSPACING command does not remove carriage return characters (CRs) at the end of lines which contain `[!ELSE!]` or `[!ELSEIF ""!]` commands. As a workaround you can quote the line ending with `[!//]`.

**WARNING**

The AUTOSPACING command does not remove leading spaces before `[!IF ""!]` expressions if the condition argument spans over several lines.

As a workaround you can explicitly specify whitespace by using the `[!'"' "'!]` or `[!WS!]` commands.

**NOTE**

The AUTOSPACING command does not remove carriage return characters if there is a whitespace behind a command.

### 7.2.10.5. Indentation

If you need the same indentation for several lines, you can use the INDENT-command which gets the amount of spaces used to indent as parameter. All output within an INDENT-block will be indented.

```
[!AUTOSPACING!]
zero
[!INDENT "4"!]
[!LOOP "text:split('1 2')"]!
[!IF ".='1'"!]    one
[!ELSE!]          two
[!ENDIF!]
[!ENDLOOP!]
three
[!ENDINDENT!]
```

The template-code above produces the following output (to make spaces visible, they are printed as "."):

```
zero
....one
....two
....three
```

---

**WARNING** `INDENT` may not work for the first line



If you use the command `[!INDENT!]` together with `[!AUTOSPACING!]` before the very first line which produces output, the first line will not be indented. As workaround you can e.g. add an extra line break before the `[!INDENT!]` block. Due to backwards-compatibility requirements, it is currently not planned to fix this limitation.

---

### 7.2.11. Excluding files from the generation

A file can be excluded from generation with a special tag. When the code generator encounters the `[!SKIPFILE!]` tag it marks the file. When the application comes across this marked file, it is not generated at all (although it is parsed completely issuing errors and warnings). The `[!SKIPFILE!]` tag comes in two shapes:

```
[!SKIPFILE!]

[!SKIPFILE "<XPath-Expression>"!]
```

In the first form the file is not generated in any case. In the second form the file is only generated if the XPath-Expression evaluates to false.

---

**WARNING** **If the XPath Expression evaluates to a string value, only 'true' (case-insensitive) is interpreted as true**



This is contrary to other uses of XPath, where any string other than the empty string is true (see [Section 6.3.4.1.2, "Type conversion with Boolean expressions"](#)).

---

A code template can have multiple **[!SKIPFILE!]**-statements in which case the last one dominates all other statements.

## 7.2.12. Asserts, errors and warnings

To check the configuration during generation and issue errors and warnings to the user, the code generator supports four constructs: asserts, errors, warnings and info messages. Messages submitted via these constructs are handed to the application via a callback interface.

Errors and assertions stop the generation process immediately while warnings and info messages only submit a message to the application.

Errors, warnings and info messages all use the same syntax:

```
[ !ERROR "<Message>" ! ]
```

```
[ !ERROR! ]
<Message>
[ !ENDERROR! ]
```

```
[ !WARNING "<Message>" ! ]
```

```
[ !WARNING! ]
<Message>
[ !ENDWARNING! ]
```

```
[ !INFO "<Message>" ! ]
```

```
[ !INFO! ]
<Message>
[ !ENDINFO! ]
```

Each construct comes in two flavors.

The first one directly includes the message in the starting tag. No end tag is needed (e.g. **[!ENDERROR!]** etc.). In this case the message is interpreted as plain text if it does not start with a question mark. If it starts with a question mark the rest of the message is evaluated as an XPath expression.



The second form of the construct is more powerful than the first one. In the second form the message can contain all construct described so far (**[!IF!]**, **[!LOOP!]**...).

The assert statement is a special form of the error statement. It includes an XPath expression. Only if this expression evaluates to false the message is printed and the generator run ends. Like the other constructs the assert comes in two shapes:

```
[ !ASSERT "<XPath expression>", "<Message>"! ]
[ !ASSERT "<XPath expression>"! ]
<Message>
[ !ENDASSERT! ]
```

The two shapes work the same way as the other constructs.

## 7.3. Public Generator API

To register a Public API Generator for a module, two steps have to be done.

- ▶ Register the extension point `dreisoft.tresos.generator.api.plugin.generator`
- ▶ Implement the Public API Generator Java class.

How this can be done will be explained in [Section 7.3.1, “Registering A Public API Generator”](#). See also the demo plugin `PublicApiTest_TS_T00D0M0I0R0`.

### 7.3.1. Registering A Public API Generator

- ▶ Open the plugin.xml of your plug-in and switch to the *Extensions*-tab
- ▶ Click *Add* and select the `dreisoft.tresos.generator.api.plugin.generator` extension-point
- ▶ On the right-hand side enter the following values:
  - ▶ Generator-ID: The identifier of this generator.
  - ▶ Module-ID: The module to which this generator shall be registered.



#### **Leave the module-ID empty for module-independent generators**

The module ID can also be left empty. In this case the generator will be module-independent and will run for each module. Note that a module-independent generator is only allowed for steps `pre` and `post`.

- ▶ Step: The step this generator shall run. Steps can be `pre`, `post`, or `default` (leave blank for default).



- ▶ ConfigClass: The configuration class of this generator. Generators must only access parameters of its configuration class or configuration classes earlier in the configuration process. The configuration class must be editable to run the generator.
- ▶ Create a class by clicking on *class\**: or *Browse...* to find an already implemented generator. How to create a generator is explained in [Section 7.3.2, “Implementing a Public API Generator”](#).

The screenshot shows the 'Extensions' tab in the EB tresos Studio interface. On the left, there is a tree view of available extensions under 'All Extensions'. One item, 'generatorID (generator)', is highlighted with a red box. To the right, the 'Extension Element Details' panel is displayed, showing configuration fields for this selected extension. The 'class\*' field is set to 'plugin.generator.api.test.testGenerator'. The 'id\*' field is 'generatorID'. The 'moduleId' field is 'MyTestModule'. The 'step' field is empty. The 'modes' field contains 'configClassTest'. The 'configClass' dropdown menu is open, showing options: 'PreCompile', 'Link', and 'PostBuild', with 'Link' currently selected. At the bottom of the window, the tabs 'Overview', 'Dependencies', 'Runtime', 'Extensions' (which is highlighted), 'Extension Points', 'Build', 'MANIFEST.MF', 'plugin.xml', and 'build.properties' are visible.

For further information, see the extension point description. See also[Section 4.5, “How to open extension point descriptions”](#).

## 7.3.2. Implementing a Public API Generator

A Public API Generator must implement the following Interface `dreisoft.tresos.generator.api.generator.ICodeGenerator` and the three methods:

- ▶ `public void generate( CodeGeneratorContext context )`
- ▶ `public String getDefaultMode()`
- ▶ `public String[] getAvailableModes( ParameterSet parameters )`

Examples:

```
/*
 * returns for which modes, the generator can be called
 */
public String[] getAvailableModes( ParameterSet parameters ) {
    return new String[]{MODE_GENERATE};
    // or new String[]{MODE_GENERATE, MODE_VERIFY} if you also want to verify
    // with this generator
}
```



```
}

/*
 * returns the default mode of the generator - In this case "generate"
 */
public String getDefaultMode() {
    return MODE_GENERATE;
}
```

An example for `public void generate( CodeGeneratorContext context )` can be found in the class `TestGenerator` in the `test.tresos.plugin` in the package `test.tresos.plugin.generator`.

### 7.3.2.1. API

For detailed information on the Public API Generator consult the Public API Java documentation. If you installed the `eclipse_tools.zip` into your eclipse you can call it with *Help -> Help Contents -> Public API Java documentation*

## 7.4. NG Generator API

### 7.4.1. Purpose

Ever since version 2008.b of EB tresos Studio, a new generator framework is available. The so called new generation (NG) generator API brings a lot new features while supporting the classic template based and external code generators as well. Of course, the existing generator framework still works and there is no need to switch to the new API.

This chapter gives only a short overview of the new generator framework. For detailed information and examples of the NG Generator API, see the API documentation of the plugin `dreisoft.tresos.generator.ng.api.plugin`.

You can find a complete demonstration in the plugin `PublicApiTest_TS_T00D0M0I0R0` which is located in `<tresos-install-dir>/demos/Studio`. You can import this demo plug-in into your Eclipse installation by right-clicking in the Package Explorer and selecting **Import... → Import Existing Project Into Workspace**.

### 7.4.2. JET Support



The NG Generator API includes support for generators based on JET (Java Emitter Templates) templates. The JET language is a scripting language similar to jsp that uses scriptlets with Java code to generate dynamic output between fix text. Before a JET template can be executed it must be processed by the JET builder that generates a Java source file. This Java source file must first be compiled to a Java class before it can be executed.

The output of a JET generator normally results in one file. There are no restrictions on the Java expressions within a scriptlet so a JET generator can contain more powerful logic than a classic template based code generator could ever have. The following code snippet shows a simple JET file:

```
<%@ jet package="xyz.jet.generated"
   class="TheNumbersGame" %>
<%
  for( int i = 1; i <= 10; i++ ) {
%>
This is round <%=Integer.toString( i )%>.

<%
}
%>
```

### 7.4.3. Standalone JET compiler

As said above a JET template must be generated to a Java source file first before using it. For that reason EB tresos Studio provides a standalone JET compiler application. You can run it via the `compile_jet.bat` file located in the `bin` directory of your EB tresos Studio installation.

You can specify several space-separated JET template files as arguments to that command. The template files can either be specified absolute or relative to the current execution directory (the directory from where you start the application, normally the `bin` directory).

With the optional parameter `-o` it is possible to define the output directory for the resulting Java files. If the output directory is not defined then the files are generated to the current execution directory. Note that the resulting Java file is described by the `jet` directive within each JET template. The application takes respect to the package requirements of the Java class and creates an appropriate directory structure under the output directory.

```
compile_jet -o C:\work\generated ..\src\templates\MyJetFile.jet
```

### 7.4.4. C Data Structures Generator (CDS)



The C Data Structures (CDS) Generator is a completely new code generation framework. It provides functionality to generate type and variable definitions from Java objects into the C language. The CDS generator is basically intended to generate configuration values into C-structs that are required by the runtime code of a module. By separating the configuration values from the logic it is possible to replace the configuration values after compiling the runtime code. So this feature targets to be used for link-time and post-build configurations.

The following features are supported:

- ▶ Primitive types, pointers, structs and arrays
- ▶ Typedefs
- ▶ Variable initializations of built-in and self-defined types
- ▶ Null pointer and function pointer support
- ▶ Compiler abstraction
- ▶ Memory abstraction
- ▶ Comments for each variable and member
- ▶ Volatile modifier
- ▶ Conditional members within structs

The CDS Generator contains Java classes that represent primitive types, structs, pointers and arrays of the C language. These classes can either be used directly to generate primitive variables or can be subclassed to create own types defined by a `typedef`.

The output is mainly controlled by so called backends. Currently the only backend provided by EB tresos Studio is the `CBackend` that writes pure C code. It is possible to extend the generator by providing various backend implementations, e.g. a binary backend.

#### 7.4.5. Apache Ant support

One of the main goals of the NG Generator API is the integration of the popular build tool Apache Ant. In fact, the NG Generator API always requires an Ant build file to execute. As mentioned above the NG Generator API uses the existing generator extension point. So the registered generator class must always be `dreisoft.-tresos.generator.ng.api.NGGenerator`. The build file(s) are specified with nested `parameter` tags to the `NGGenerator` class at the generator extension point.

```
<extension point="dreisoft.tresos.generator.api.plugin.generator"
          id="NGGeneratorAPITest_TS_genreator_id"
          name="NGGeneratorAPITest_TS Generator">

<generator id="NGGeneratorAPITestGenerator"
           moduleId="NGGeneratorAPITest_TS"
           class="dreisoft.tresos.generator.ng.api.NGGenerator">
```



```

<parameter name="buildfile"
           value="resources/generator_build.xml" />
</generator>
</extension>
```

There are Ant tasks provided for each type of generator. Together with the standard Ant tasks it is possible to create powerful build scripts. The following list gives a short overview of the delivered tasks:

NG Generator Ant tasks	Description
ng.property	<p>Allows to set an Ant property via a query to the DataModel, for example:</p> <pre> &lt;ng.property   name="propertyName"   ctxt="variableName"   xpath="XPath expression"   override="false" &gt; &lt;/ng.property&gt;</pre>
ng.setgeneratorvar	<p>Allows to set context variables from within the build.xml. See the Java class <code>dreisoft.tresos.generator.api.generator.CodeGeneratorContext</code> for the predefined variables. For example:</p> <pre> &lt;ng.setgeneratorvar   name="propertyName"   ctxt="variableName"   xpath="XPath expression"   override="true" &gt; &lt;/ng.setgeneratorvar&gt;</pre>
ng.getgeneratorvar	<p>Allows to get context variables and set it to ant properties. See the Java class <code>dreisoft.tresos.generator.api.generator.CodeGeneratorContext</code> for the predefined variables. For example:</p> <pre> &lt;ng.getgeneratorvar   name="variableName"   property="propertyName"   override="true" &gt; &lt;/ng.getgeneratorvar&gt;</pre>
ng.tmplgen	<p>Runs the template based code generator, for example:</p> <pre> &lt;ng.tmplgen   template="templates/CanIf_Cfg.h"   file="CanIf_Cfg.h"</pre>



NG Generator Ant tasks	Description
	<pre> dctxt="variableName" failonerror="false" needCodeTags="true" &gt;  &lt;include path="generate" /&gt;  &lt;variable name="var1" value="foo" /&gt; &lt;variable name="var2" value="bar" /&gt; &lt;/ng.tmplgen&gt;</pre>
ng.extgen	<p>Runs an external code generator, for example:</p> <pre> &lt;ng.extgen   commandline="\$\$\{pluginPath\}/bin/gen.exe     -f \$\$\{outputDir\}/epcFiles/project.epc     -o \$\$\{outputDir\}/src/CanIf_Cfg.c"   cfgFilePath="epcFiles"   contentType="asc:2.0"   generateAllModules="true"   generateIntoOneFile="true"   workingDir="/TEMP"   shell="false"   timeout="5"   error="^\s*Error\[\$\$f,\$\$l\]:\s*\\$\\$m\s*\\$\\$"   warning="^\s*Warning\[\$\$f,\$\$l\]:\s*\\$\\$m\s*\\$\\$"   info="^\s*Info\[\$\$f,\$\$l\]:\s*\\$\\$m\s*\\$\\$"   failonerror="true" &gt;  &lt;variable name="var1" value="foo" /&gt; &lt;variable name="var2" value="bar" /&gt; &lt;/ng.extgen&gt;</pre>
ng.jetgen	<p>Runs a jet template that is already compiled to a Java class that implements interface <code>dreisoft.tresos.generator.ng.api.ant.IJavaGenTemplate</code>, for example:</p> <pre> &lt;ng.jetgen   class="some.package.SomeJetGenerator"   file="JetOutput.txt"   failonerror="false" &gt;  &lt;argument name="useDet" value="true" /&gt; &lt;argument name="someArg" value="123" /&gt; &lt;/ng.jetgen&gt;</pre>



NG Generator Ant tasks	Description
ng.jetgen.src	<p>Compiles and runs a jet template that is delivered as source code, for example:</p> <pre data-bbox="477 473 1240 923">&lt;ng.jetgen.src     class="some.package.SomeJetGenerator"     file="JetOutput.txt"     failonerror="true" &gt;      &lt;jetIncludes         dir="../templates"         includes="**/*Generator.jet" /&gt;     &lt;javaIncludes dir="../src" includes="**/*.java" /&gt;      &lt;argument name="someArg" value="123" /&gt;     &lt;argument name="arg1" value="\${someAntProperty}" /&gt; &lt;/ng.jetgen.src&gt;</pre>
ng.javagen	<p>Runs a java generator. It must implement dreisoft.tresos.generator.ng.api.ant.IJavaGenTemplate, for example:</p> <pre data-bbox="477 1084 1033 1219">&lt;ng.javagen     class="some.package.JavaGenGenerator"     file="generated.txt" &gt; &lt;/ng.javagen&gt;</pre>
ng.javagen.src	<p>Compiles and runs a java generator that is delivered as source code, for example:</p> <pre data-bbox="477 1376 1240 1736">&lt;ng.javagen.src     class="some.package.JavaGenGenerator"     file="xyz.txt"     failonerror="false" &gt;      &lt;includes dir="../src" includes="**/*.java" /&gt;      &lt;argument name="someArg" value="abc_-" /&gt;     &lt;argument name="arg1" value="\${someAntProperty}" /&gt; &lt;/ng.javagen.src&gt;</pre>

## 7.4.6. Source delivery

As shown in the Ant tasks overview above there are the tasks `ng.javagen.src` and `ng.jetgen.src` that allow to deliver the generator as source code and not already compiled to a class. In case of the Java-based



generator it is the Java source file that implements the `dreisoft.tresos.generator.ng.api.ant.-IJavaGenTemplate` interface, in case of the JET-based code generator it is the JET template.

Both tasks first compile the specified source files before executing the specified generator class. The required Java compiler is delivered with the NG Generator API and it is not possible to use another one. So the generator source code must be compatible with the relevant compiler (current version Java JDK 1.8.0). The class path is based on the plugin's class path. It is possible to add further class path locations. The source files are compiled to a temporary directory within the configuration area of EB tresos Studio.

## 7.5. EPC File Generator API

### 7.5.1. Purpose

The EPC File Generator is useful for all developers using external generators. To invoke external generators usually `.epc` files are necessary.

EB tresos Studio uses `.xdm` files to store the configuration. To be able to use the external generators in combination with EB tresos Studio, an additional generator is provided: The EPC File Generator.

### 7.5.2. Generating EPC files

The EPC File Generator is easy to use:

- ▶ Use the EPC File Generator in the generator extension-point for all modules that provide an external generator.
- ▶ Run the generators in generate mode from the GUI or commandline (project-based mode or legacy mode) to create required `.epc` files in the output directory.

The EPC File Generator can be registered in the `plugin.xml` of the respective module via the generator extension-point:

```
<extension point="dreisoft.tresos.launcher2.plugin.generator"
          id="EPCGenerator"
          name="EPC Generator">
<generator moduleId="Lin_TS_T17D12M1I1R0"
          id="Lin_TS_T17D12M1I1R0GeneratorId"
          class="dreisoft.tresos.autosar2.generator.EPCFileGenerator">
  <parameter name="cfgFilePath" value="output"/>
  <parameter name="generateAllModules" value="false"/>
```



```

<parameter name="generateIntoOneFile" value="false"/>
<parameter name="generateModuleTypes" value="Rte, Os, EcuM" />
<parameter name="contentType" value="asc:2.1"/>
</generator>
</extension>

```

**NOTE**

To be able to access the EPCFileGenerator the `MANIFEST.MF` of the module must define the plug-in `dreisoft.tresos.autosar2.plugin` as a required bundle:



`Require-Bundle: dreisoft.tresos.autosar2.plugin`

Available parameters for the ECU File Generator are:

**cfgFilePath**

path relative to the output directory in which to create the `.epc` files.

**generateAllModules**

This is a Boolean parameter that tells the EPC File Generator if all module configurations of the processed project must be exported or only the module configuration of the module for which the generator is registered. The default value is "true".

**generateModuleTypes**

Specifies which types of module configurations of the processed project must be exported. Module types are specified as a comma-separated list, for example "`Rte,Os,EcuM`". May only be used if "generateAllModules" is set to "false".

**generateIntoOneFile**

Specifies if the generated module configurations are to be stored into one single file or if each module configuration is to be stored into a separate file which is named `<Module-Type>.epc`. The default value is "false".

If the parameter is set to "true", the filename depends on the parameters "generateAllModules" and "generateModuleTypes". If "generateAllModules" is set, the filename is `project.epc`. If module types are specified via "generateModuleTypes", the filename consists of the alphabetically sorted module types plus the extension `.epc`. For example if "generateModuleTypes" is set to "`Rte,Os,EcuM`", the filename is `EcuMOsRte.epc`.

If each module is exported into a single file only the first module configuration of a given type is exported.

**contentType**

Specifies in which content type the files should be generated. The default value is `asc:2.0..`

**allVariants**

Specifies whether to export all variants including variant information or not. The default value is `false`.

If the parameter is set to "true" and the project has a currently selected Selectable variant, then the complete configuration including all variants and variation point information will be exported. If the parameter is set



to "false" or not set at all, then only the configuration for the currently selected variant will be exported without variation point information.

If the parameter is set to "true" and the project has a currently selected Selectable variant, then also all PostBuild Selectable variants incl. Criterions and CompuMethod definitions will be exported to a dedicated file called 'PostBuildSelectableVariants.arxml' located beneath the written .epc file.

Example: A project with 3 module configurations:

- ▶ Com (with generator extension point for EPCFileGenerator; generateAllModules="true"; generateIntoOneFile="true"; cfgFilePath="output")
- ▶ EcuC
- ▶ Lin (with generator extension point for EPCFileGenerator; generateAllModules="true"; generateIntoOneFile="false"; cfgFilePath="output")

If the user runs "generate" from the Commandline or the GUI several files will be generated into the folder "output" relative to the output directory. First all module configurations will be generated into one file: `project.-epc`. And additionally 3 single files will be generated in addition: `Com.epc`, `EcuC.epc` and `Lin.epc`.

## 7.6. External Generator API

### 7.6.1. Purpose

You can use the External Generator API to embed modules with existing code generators in EB tresos Studio.

Typical use cases are:

- ▶ You want to provide the advanced EB tresos Studio configuration features to your users without having to rewrite your existing generator.
- ▶ You want to integrate some third-party module(s) into an EB tresos Studio project and keep the configuration tooling consistent.

### 7.6.2. Functionality

The External Generator Engine first exports the ECU configuration of the project to one or several AUTOSAR files of any of the supported AUTOSAR versions. Depending on the generator configuration, the whole configuration or only parts of it are exported. For details, see [Section 7.6.5, "Registering an External Generator"](#).



Subsequently, the external generator is executed using the configured command line, see [Section 7.6.6, “Generator Commandline”](#). The output of the generator is parsed for messages which are logged within EB tresos Studio as described in [Section 7.6.7, “Parsing the output of the generator”](#).

### 7.6.3. Prerequisites

External generators are bound to a module like the normal template based code generators.

Therefore a normal module plug-in that will host the generator must be available. The module must include:

- ▶ module registration in the `plugin.xml`
- ▶ XDM schema

### 7.6.4. Adding the Generator to the plug-in

The generator executable itself must be copied to the module plug-in for which it generates code. To do that a directory `bin/` should be added to the module plug-in and the executable including necessary companion files should be added to this directory resulting in a file structure comparable to:

```
Can_TS_T16D4M2I0R0/META-INF/MANIFEST.MF
Can_TS_T16D4M2I0R0/plugin.xml
Can_TS_T16D4M2I0R0/config/Can.xdm
Can_TS_T16D4M2I0R0/bin/Cangen.exe
Can_TS_T16D4M2I0R0/src/...
Can_TS_T16D4M2I0R0/include/...
```

If the generator needs `.dll` that are not installed to the Windows system directory these `.dll` files could be copied to `lib/` beneath the tresos base installation directory.

### 7.6.5. Registering an External Generator

The generator must be registered via the generator Eclipse extension-point in the `plugin.xml` of the corresponding plug-in.

```
<extension point="dreisoft.tresos.launcher2.plugin.generator"
    id="Generator_Det_TS_T16D4M2I0R0"
    name="Det TemplateGenerator">
<generator id="ExtGenDet"
    moduleId="Det_TS_T16D4M2I0R0"
```



```

        modes="myMode"
        class="dreisoft.tresos.autosar2.generator.ExternalGenerator">
<parameter name="cfgFilePath" value="epc"/>
<parameter name="generateAllModules" value="false"/>
<parameter name="generateModuleTypes" value="EcuC, Dem, Det"/>
<parameter name="generateIntoOneFile" value="true"/>
<parameter name="contentType" value="asc:2.1"/>
<parameter name="shell" value="false"/>
<parameter name="cwd" value="${pluginPath}/generator"/>
<parameter name="commandline"
           value="det.exe -f ${configFiles} -o ${outputDir}"/>
<parameter name="error" value="^Error:\s*\$m\s*\$/>
<parameter name="warning" value="^Warning:\s*\$m\s*\$/>
<parameter name="info" value="^Info:\s*\$m\s*\$/>
<parameter name="evalExitCode" value="false"/>
</generator>
</extension>
```

The External Generator interface that runs the generator is implemented in the class `dreisoft.tresos.autosar2.generator.ExternalGenerator`.

With the optional `xml` attribute `modes`, you may define generation modes. If they are not defined, "generate,verify" is used as default instead. You may trigger the mode(s) via the GUI or from the command line. For instructions how to do that, see the EB tresos Studio user's guide, chapter [Working with the commandline/Commands for legacy code generation/Running generation modes](#).

How the generator is run is defined by the parameters to the registration. The following parameters are available:

Parameter	Default	Description
<code>cfgFilePath</code>		The parameter <code>cfgFilePath</code> selects the path in which configuration file are generated. This path is interpreted relative to the generation path.
<code>generateAllModules</code>	true	If the parameter <code>generateAllModules</code> is set to <code>false</code> , then only the configurations of a module in the project are generated, for which the generator is registered. Per default all enabled module configurations in the project are generated.
<code>generateModuleTypes</code>		The parameter <code>generateModuleTypes</code> defines which types of enabled module configurations are generated. May only be used if <code>generateAllModules</code> is set to <code>false</code> .
<code>generateIntoOneFile</code>	false	If the parameter <code>generateIntoOneFile</code> is not set, then each module configuration is to be stored into a separate file which is named <Module-Type>.epc.



Parameter	Default	Description
		<p>If the parameter <code>generateIntoOneFile</code> is set to <code>true</code>, then one single configuration file is generated, which contains all generated module configurations. The filename depends on the parameters <code>generateAllModules</code> and <code>generateModuleTypes</code>. If <code>generateAllModules</code> is set, the filename is <code>project.epc</code>. If module types are specified via <code>generateModuleTypes</code>, the filename consists of the alphabetically sorted module types plus the extension <code>.epc</code>. For example if <code>generateModuleTypes</code> is set to "EcuC, Dem, Det", the filename is <code>DemDetEcuC.epc</code>.</p> <p>If each module is exported into a single file only the first module configuration of a given type is generated.</p>
<code>contentType</code>	<code>asc</code>	The parameter <code>contentType</code> sets the AUTOSAR ECU Configuration version with which the configuration is generated to disk. All available version ( <code>asc</code> , <code>asc:2.0</code> , <code>asc:2.1</code> , <code>asc:3.0</code> , <code>asc:3.1...</code> ) are usable here.
<code>allVariants</code>	<code>false</code>	<p>Specifies whether to export all variants including variant information or not. The default value is <code>false</code>.</p> <p>If the parameter <code>allVariants</code> is set to "true" and the project has a currently selected Selectable variant, then the complete configuration including all variants and variation point information will be exported. If the parameter is set to "false" or not set at all, then only the configuration for the currently selected variant will be exported without variation point information.</p> <p>If the parameter is set to "true" and the project has a currently selected Selectable variant, then also all PostBuild Selectable variants incl. Criterions and CompuMethod definitions will be exported to a dedicated file called <code>PostBuildSelectableVariants.arxml</code> located beneath the written <code>.epc</code> file.</p>
<code>commandline</code>		The parameter <code>commandline</code> is mandatory and stores the command-line that calls the generator.
<code>error</code>		The parameter <code>error</code> contains the grammar via which to parse errors out of the output of the generator.
<code>warning</code>		The parameter <code>warning</code> contains the grammar via which to parse warnings out of the output of the generator.
<code>info</code>		The parameter <code>info</code> contains the grammar via which to parse info messages out of the output of the generator.



Parameter	Default	Description
shell	false	The parameter <code>shell</code> must be set to <code>true</code> if the generator is implemented as a batch-file.
cwd		The parameter <code>cwd</code> can be used to specify the working directory from which to run the generator.
timeout		The parameter <code>timeout</code> , when given, gives a number of seconds after which the generator will be killed if it did not finish yet.
evalExitCode	false	The optional parameter <code>evalExitCode</code> must be set to <code>true</code> if the exit code of the executed commandline shall be evaluated. An exit code different from 0 (zero) will be interpreted as an error in that case. If set to false, the exit code will be ignored.

## 7.6.6. Generator Commandline

The commandline defined by the parameter `commandline` and the working directory, with which the generator is started, defined by `cwd` may contain variables which are expanded right before the parameter is used by the External Generator Engine. Variables always have the form:

```
 ${variable-name}
```

As an example the commandline of the generator call could have been set with:

```
<parameter name="commandline"
    value="${pluginPath}/bin/Cangen.exe -f ${configFiles} -o
    ${outputDir}/can"/>
```

The following variables are expanded:

Variable	Description
<code>outputDir</code>	The variable <code>outputDir</code> contains the generation path.
<code>generationDir</code>	The variable <code>generationDir</code> contains the generation path as stored in the project settings. In legacy mode it is the same as the variable <code>outputDir</code> .
<code>generationPaths</code>	The variable <code>generationPaths</code> contains the generation path of each module configuration in the same order as in variable <code>generateConfigs</code> . Only module configurations that are enabled for code generation are considered.



Variable	Description
generationPathsDistinct	The variable <code>generationPathsDistinct</code> contains the generation paths for each module configuration that is enabled for code generation without any duplicate entries.
tresosBase	The variable <code>tresosBase</code> stores the absolute path to where EB tresos Studio is installed.
pluginPath	The variable <code>pluginPath</code> stores the absolute path to where the Eclipse plug-in is installed that registers the generator.
variant	The variable <code>variant</code> stores the EB tresos Studio variant string (e.g. 2008.a).
buildnr	The variable <code>buildnr</code> stores the EB tresos Studio buildnr.
relversion	The variable <code>relversion</code> stores the EB tresos Studio release version.
project	The variable <code>project</code> stores the name of the project that will be generated. In legacy mode the project name is given with the system property <code>project</code> .
projectDir	The variable <code>projectDir</code> stores the absolute path to the project that will be generated. May be empty in legacy mode if the variable <code>project</code> is not set, or if no such project exists within the workspace.
module	The variable <code>module</code> stores the module-id of the module that will be generated.
modules	The variable <code>modules</code> stores a space separated list of module-ids of all modules in the project that will be generated. In legacy mode these are the names of all module-ids that have been loaded.
generateModules	The variable <code>generateModules</code> stores a space separated list of module-ids of all modules in the project that are generating code.
enabledModules	The variable <code>enabledModules</code> stores a space separated list of module-ids of all modules in the project that are enabled.
configFiles	The variable <code>configFiles</code> stores the a space separated list of the config files that were generated as absolute paths.
defaultconfig	The variable <code>defaultconfig</code> stores the name of the module configuration that should be considered as the default configuration when there are more than one module configurations for one module generated. This is by default the first found module configuration.
config	The variable <code>config</code> stores the name of the currently generated module configuration when there are more than one module configurations for one module generated.
configs	The variable <code>configs</code> stores a space separated list of the names of all configurations for the module that is been generated.
generateConfigs	The variable <code>generateConfigs</code> stores a space separated list of the names of all configurations that should be generated for the module that will be generated.



Variable	Description
enabledConfigs	The variable <code>enabledConfigs</code> stores a space separated list of names of all configurations that are enabled for the module that will be generated.
postBuildVariant	The variable <code>postBuildVariant</code> stores the short-name of the currently built post-build variant.
moduleSpecVer	The variable <code>moduleSpecVer</code> stores the full spec version of the module that will be generated as extractable from the module registration of the module.
moduleSoftwareVer	The variable <code>moduleSoftwareVer</code> stores the full software version of the module that will be generated as extractable from the module registration of the module.
moduleReleaseVer	The variable <code>moduleReleaseVer</code> stores the full release version of the module that will be generated as extractable from the module registration of the module.
moduleType	The variable <code>moduleType</code> stores the type of the module that will be generated as extractable from the module registration of the module.
moduleCategory	The variable <code>moduleCategory</code> stores the category of the module that will be generated as extractable from the module registration of the module.
moduleLayer	The variable <code>moduleLayer</code> stores the layer of the module that will be generated as extractable from the module registration of the module.
target	The variable <code>target</code> stores the target of the project that will be generated. In legacy mode the target can be set with the system property <code>target</code> .
derivate	The variable <code>derivate</code> stores the derivate of the project that will be generated. In legacy mode the derivate can be set with the system property <code>derivate</code> .
ecuid	The variable <code>ecuid</code> stores the ecuid of the project that will be generated. In legacy mode the ecuid can be set with the system property <code>ecuid</code> .
mode	The variable <code>mode</code> stores the mode of the generator that is running.

## 7.6.7. Parsing the output of the generator

To parse back the output of the generator the grammar of the output must be specified. The External Generator Engine then parses standard output and standard error output of the generator and creates log entries for the error logging facility of EB tresos Studio. If any errors are logged, the generation is assumed to be failed.

The grammar for parsing back the output is specified with the parameters:

- ▶ `error`
- ▶ `warning`
- ▶ `info`



All three parameters are optional. The grammar for error messages is specified with the parameter `error`, the one for warnings with the `warning` parameter and the one for info messages with the `info` parameter.

The grammar has the following form (EBNF):

```
( variable | regexp ) *
```

The grammar is composed of (perl) regular expressions and variables. Variables have the form:

```
"$" variableName
```

where `variable name` is a single character. Variables match the part between the regular expressions. If a variable is given as the first token in the grammar it matches the text between the start of the yet unparsed output and the first regular expression. If a variable is given as the last token of a grammar it matches the end of the output.

The grammar allows to parse multi-line messages by using the special variable `$L`. The `$L` variable is actually not a real variable but serves as a marker to mark the beginning of a continued line pattern. Everything before the first `$L` must match the first line of a message printed by the generator. Afterwards the parser tries to match the continued line patterns. A continued line pattern is the grammar between a `$L` variable and another `$L` variable or a `$L` variable and the `$N` variable or a `$L` variable and the end of the grammar.

The grammar also allows to define multiple alternative patterns for one message type (`error`, `warning`, `info`). An alternative pattern is introduced by the marker variable `$N`. Everything from a `$N` to the next `$N` or the end of the pattern is considered as an alternative grammar pattern. Each such pattern can contain continued line patterns whereas the first pattern is always the first line pattern.

Consider the following example:

```
<parameter name="error"
           value="ERROR:$m$LFILE:$f$LLINE:$NFATAL:$m$LFILE:$f$LLINE:$l"/>
```

This pattern matches error messages of the following forms:

```
ERROR: error message
FILE: test.c
LINE: 10
```

```
FATAL: fatal error message
LINE: 5
FILE: test.c
```

```
ERROR: error message
FILE: test.c
```

```
FATAL: fatal error
LINE: 7
```

---

**NOTE****Separate handling of output streams**

The standard output stream and the standard error output stream are handled separately from each other. Thus, multi-line patterns must only refer to output of one of the streams. If multi-line patterns refer to both output streams within the same pattern, it will not work, e.g. if an error message is reported on standard error output followed by some details reported on standard output.

---

Any line that does not match a pattern is ignored.

All other variables are used to fill in parts of the messages returned to . The following variables can be used:

Variable	Description
\$f	the variable \$f matches the filename printed in the error message.
\$l	the variable \$l matches the line printed in the error message.
\$m	the variable \$m provides the error message.

The resulting message printed by looks either the following way if at least file name message variables are used:

```
"[" filename ":" line "] " message
```

otherwise only the message is printed.



## 8. Extending the user interface

### 8.1. How to use the National Language Support (NLS) / Operation Status (OS)

---

**NOTE**

*Precondition:* You must set up Eclipse as described here [Section 4.3, “How to install the EB tresos Studio Eclipse tooling”](#)



#### 8.1.1. Localizing the language of the user interface

##### 8.1.1.1. Concepts: National Language Support (NLS)

National Language Support is used to localize messages or labels of a plug-in.

For eclipse plugins, there are two places where localized messages are most likely used: in the plugin manifest file and inside the java code. In the java code, NLS can easily be used by calling a method that returns the localized version of a string. Parameterized messages are easily achievable through method parameters. The EB tresos Studio NLS framework therefore generates classes that provide such methods.

However, in the plugin manifest files (`plugin.xml` and `Manifest.mf`) no generated java methods for translation can be used. EB tresos Studio therefore adopts the eclipse approach, which is explained in the next paragraph, and extends it a bit.

The eclipse approach works as follows: inside the plugin manifest properties can be used which are automatically replaced by localized strings at runtime. The properties have the following form: `%propertyName`. The values of the properties are looked up at runtime from `*.properties` files. If a locale is set, the lookup takes place in the file `plugin_<locale>.properties`. If no such file exists, or if a single message cannot be found inside that file, default file called `plugin.properties` is used as fallback. The syntax of the property files follows the usual eclipse property file format.



### 8.1.1.2. Workflow for NLS-support in Java code

The EB tresos Studio NLS framework uses classes extending the `dreisoft.tresos.lib2.nls.NLS` class for the retrieval of localized messages. Those classes are automatically generated from textfiles by the NLS/OS builder (see [Section 8.1.3, “The NLS/OS builder”](#)).

Using the NLS framework in Java code roughly involves the following steps:

- ▶ Define an `nls-<classname>.txt` config file (with `<classname>` being the name of the generated class in the following step). A description of the config file format and an example of how a config file should look like, can be found in [Section 8.1.1.3, “Config file format”](#)
- ▶ After creating a config file, open the properties of your plugin (right-click -> Properties) and go to the NLS Preferences tab. There you need to select the Java source directory and then "Add" the config file to the table below. If the Type, shown in the table, is not set to "nls", do that manually.

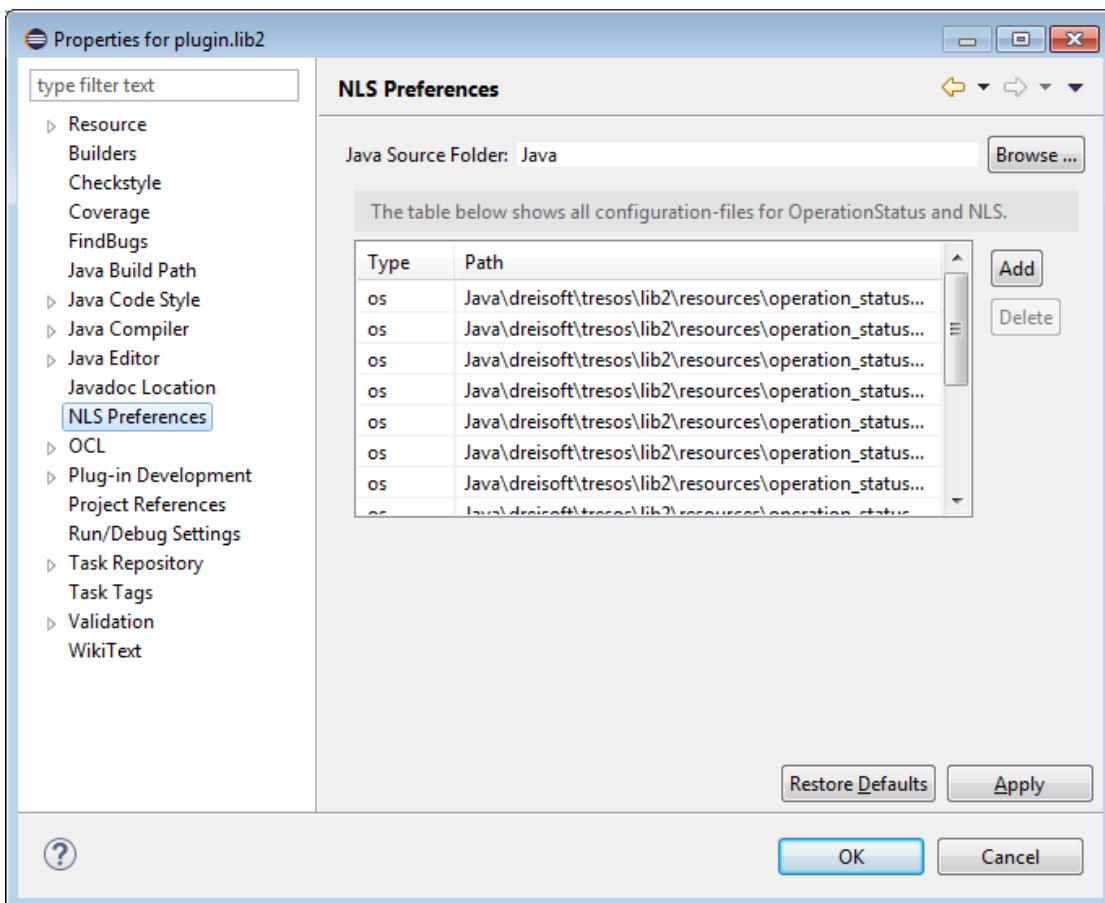


Figure 8.1. NLS/OS Preference Page

- ▶ The property files and classes are generated by the NLS/OS builder during the next project build. For further information, see [Section 8.1.3, “The NLS/OS builder”](#).
- ▶ use the static methods of the generated class `<classname>` in your code to generate the localized strings



### 8.1.1.3. Config file format

The config file consists of two parts: a header section which provides general info necessary for class generation, followed by a list of message blocks which begin with the message id and contain the messages and additional data. Each line inside the config file begins with a directive. If a # character is found at the beginning of a line, the line is regarded as comment and ignored. Whitespace is filtered out automatically.

In the following the syntax of a config file is explained.

```

# The NLS factory class to be generated
TARGET-CLASS: test.tresos.plugin.generated.nls.MyNLSClass
# Note, that this class must be derived from APINLS
EXTENDS      : APINLS
# Note, that this class must import dreisoft.tresos.lib2.api.log.APINLS
IMPORT       : dreisoft.tresos.lib2.api.log.APINLS
# Optional: if the generated class should implement one or several
# Interfaces, they can be added here separated by a semicolon
IMPLEMENTES: ISomeClass
# The imports are needed for the different parameters
# of the factory methods (see PARAMs below)
IMPORT : dreisoft.tresos.datamodel2.api.DCtxt
# Optional: The javadoc-description of the generated class can be edited
# here
DESCRIPTION: This class contains several status for application-specific
errors and warnings
# specifies the registration-name under which the resource-bundle
# is registered at the ResourceHandler and the full-qualified
# properties-file-name (to which all messages will be written)
# <Bundle-Name>=<File-Name>
BUNDLE-NAME: MyEclipsePluginNLS=test.tresos.plugin.resources.
               MyEclipsePluginNLS
# Additional code can be added (only) at the end of the file
# using the following tag. All lines up to the end of this
# tag will be added as code to the generated class

```

Example:

```

TARGET-CLASS: test.tresos.plugin.generated.nls.MyNLSClass
BUNDLE-NAME : MyEclipsePluginNLS=test.tresos.plugin.resources.
               MyEclipsePluginNLS
IMPORT       : dreisoft.tresos.lib2.api.log.APINLS
EXTENDS      : APINLS

```

The header can be followed by multiple message blocks. Each block contains the unique message id, the non-localized message, the parameter definitions of the message and possibly multiple translations of the message.



```

ID      : HELLO_WORLD
PARAM   : String arg0=an adjective
PARAM   : String arg1=another adjective
L-en    : Hello {0} {1} England!
L-de    : Hallo {0} {1} Deutschland!

```

## ID

The ID is used as the name of the factory method and also used to generate a constant variable with the name `ID_<Id>` that stores the id of the message. The ID must be unique within one file.

## PARAM

A PARAM defines a parameter with which the message can be parameterized. The parameter has the following form:

```
"PARAM" ":" [ [ <Class> ] <parameterName> "=" ]
<description>
```

The Class must be added as IMPORT within the file-header. Primitive types (e.g. int, float) can be used too. The default-Case is String. If no parameterName is given, a name of the form `<param><number>` will be generated.

Example:

```

PARAM : INode node=the node to describe
PARAM : value=a string-value
PARAM : a description for a string-parameter

```

## L-\*

The L-\* are the localized messages where \* is the name of the language (e.g. "en" for English, which is mandatory). The value can contain parameter references to the parameters defined with the PARAM keyword in the form defined by the `java.text.MessageFormat`.



### 8.1.1.4. NLS examples

#### 8.1.1.4.1. Example for the usage in a Plugin.xml

In a plugin.xml no Java methods can be called to localize your plugin. So some special settings must be made. At first the plugin must be told where to find the localization file. This is done within the `MANIFEST.MF` with the following entry:

```
Bundle-Localization: plugin
```

Then a file named `nls-plugin.txt` must be created somewhere in your plugin with similar entries like the following:

```
#  
# NLS Messages for the plugin.xml file  
  
#  
  
MODE: MODE_PROPERTIES  
  
ID : DescriptionPage  
L-en: englisch description  
L-de: deutsche Beschreibung
```

The `nls-plugin.txt` gets a special handling from the NLS/OS class generator ([Section 8.1.3, “The NLS/OS builder”](#)) and only generates \*.properties files in the root dir of the plugin.

At last the NLS messages can be used in the plugin.xml. Eclipse gets the NLS information from the generated .-properties files and replaces the `%NLS` placeholder with the localized values.

```
<extension  
    id="DescriptionPage"  
    name="DescriptionPage"  
    point="org.eclipse.ui.views">  
  
    <view  
        id="package.name.of.this.DescriptionPage"  
        name="%NLS.DescriptionPage"  
        icon="icons/description.gif"  
        class="package.name.of.this.DescriptionPage"  
        category="yourapp"  
        allowMultiple="false"/>  
</extension>
```



#### 8.1.1.4.2. NLS source code usage example

To use NLS in source code, no special entry is needed. Take the following example of the creation of a button, where you want to set a localized text. At first you need the NLS config file nls-MyEclipseNLS.txt from which the Java class is generated by the NLS/OS class generator ([Section 8.1.3, “The NLS/OS builder”](#)).

```
TARGET-CLASS: test.tresos.plugin.generated.nls.MyEclipseNLS
BUNDLE-NAME : MyEclipse=test.tresos.plugin.generated.resources.MyEclipse
IMPORT       : dreisoft.tresos.lib2.api.log.APINLS
EXTENDS      : APINLS

#####
#
# Section Comment
#
#####

ID      : BUTTON_BROWSE
L-en    : Browse
L-de    : Durchsuchen

m_button = new Button( parent, SWT.PUSH );
m_button.addSelectionListener( new SelectionListener() {
    public void widgetSelected( SelectionEvent e ) {
        // do something
    }

    public void widgetDefaultSelected( SelectionEvent e ) {
    }
} );
// localized message
m_button.setText( MyEclipseNLS.BUTTON_BROWSE() );
// non-localized message
// m_button.setText( MyEclipseNLS.NONLS_BUTTON_BROWSE() );
```

## 8.1.2. Defining APIOperationStatus subclasses



### 8.1.2.1. Concepts: APIOperationStatus

There are a lot of complex and nested operations possible within the DataModel. Each of these operations can return an error, a warning or an info-message, so that the result of an operation may be a complex structure of information.

Depending on the situation, the system must be able to provide the right information at the right time. For example if the user starts the generation and there is an error within the loaded files, only one error-message-line is needed including the information, where the error occurred. On the other side - if the user cannot solve this problem - there must be more information available for the programmer, e.g. the call stack of the occurred exception within a log file. Maybe information about previously successful operations are needed too. To handle these requirements, the Lib2.API provides a sophisticated error reporting system.

In EB tresos Studio all error handling is done with APIOperationStatus. Status are handed to methods, and methods just hang in their own status if an error occurred. The calling method then checks the status after the called method returned. An example on how to use an APIOperationStatus can be found in section [Section 8.1.2.4, "Operation Status Examples".](#)

#### 8.1.2.1.1. Glossary

Term	Description
APIOperationStatus	All status information is gathered in an operation status. An operation status is a container that stores a tree of status messages.
Status Message	One entry in an operation status consisting of all data of the message like, id, parameters, severity.
Status Code	A status message is uniquely identified by a status code which is a 32 Bit Integer or a String, depending on the PREFIX in the Configuration File.
Configuration File	All messages available in the system are defined in configuration files which are simple ASCII-files defining all static data of a status message.

#### 8.1.2.1.2. Functionality

- ▶ There are four severity-levels for messages:

Error:

A real problem occurred. The operation cannot be completed successfully.

**Warning:**

A possible problem occurred. The operation can be completed, but the result may have problems.

**Info:**

An informative message.

**Ok:**

This is not a real status but is used to describe the running operation to provide a context for nested messages.

- ▶ Because operations may be nested, the messages may reflect this structure. E.g. if there are 5 nested operations and the last operation generates an error-message, the message must include context information from all 5 nesting levels.
- ▶ A message may provide a localized version for the user (e.g. in Japanese language, French etc.).
- ▶ A message provides a non-localized representation for log files.
- ▶ An error-message includes a stack trace for debugging purposes.
- ▶ As a status is created very often the status handling is time performance optimized.
- ▶ Messages are configured with ID, severity, non-localized version, localized versions, parameter specification via a textfile.
- ▶ Messages may be parameterized to provide context information provided by the creator of the message.
- ▶ Messages may contain a node reference so an error can be navigated to in a GUI based application.
- ▶ The API provides a typesafe interface for reporting messages by using a code-generator to generate message classes.

### 8.1.2.2. Workflow for OS-support in Java code

The EB tresos Studio OS framework uses classes extending the `dreisoft.tresos.lib2.api.log.API-OperationStatus` class for creation of Operation Status. Those classes are automatically generated from textfiles by the NLS/OS builder (see [Section 8.1.3, “The NLS/OS builder”](#)).

Using the Operation Status framework in Java code roughly involves the following steps:

- ▶ Define an `operation_status-<classname>.txt` config file (with `<classname>` being the name of the generated class in the following step). A description of the config file format and an example of how a config file should look like, can be found in [Section 8.1.2.3, “Config file format”](#)
- ▶ After creating a config file, open the properties of your plugin (right-click -> Properties) and go to the NLS Preferences tab. There you need to select the Java source directory and then “Add” the config file to the table below. If the Type, shown in the table, is not set to “os” do that manually.

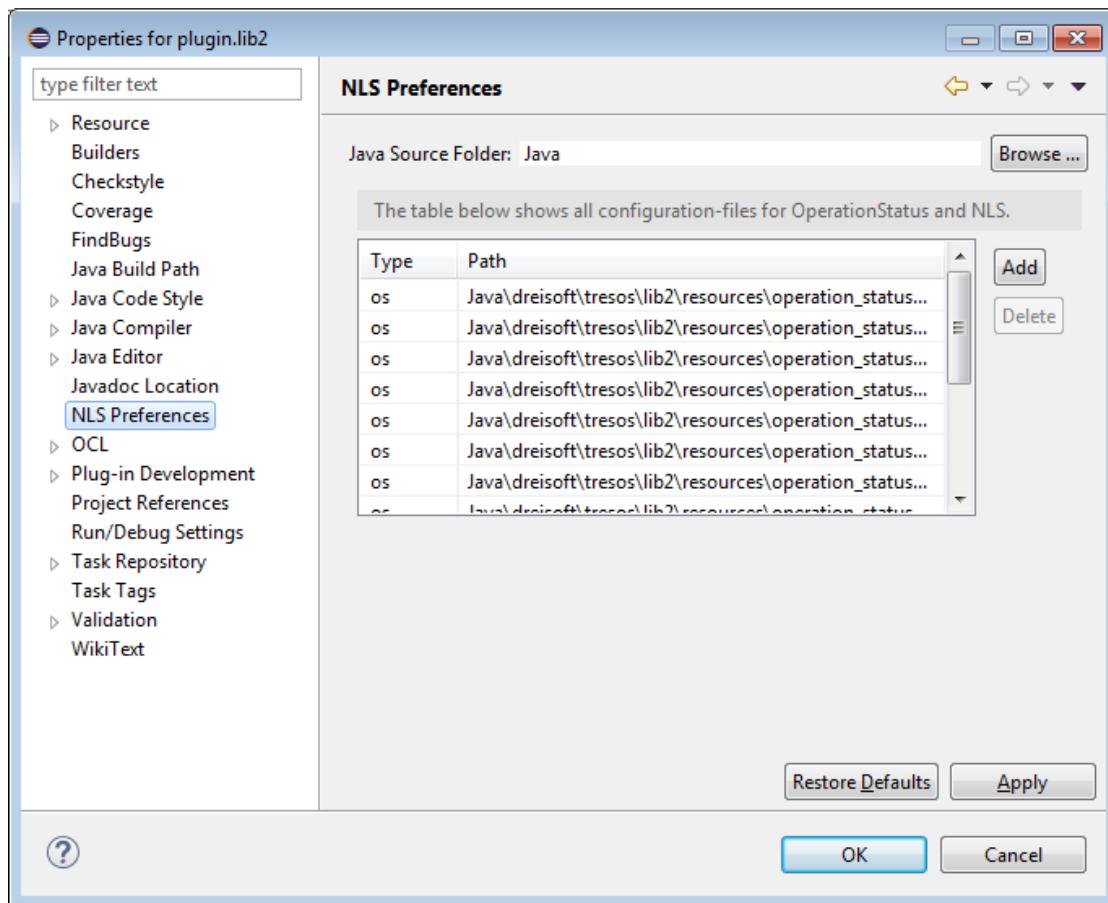


Figure 8.2. NLS/OS Preference Page

- ▶ The NLS/OS builder generates the property files and classes. For further information see [Section 8.1.3, "The NLS/OS builder"](#).
  - ▶ Use the static methods of the generated class <classname> in your code to generate the localized strings

### 8.1.2.3. Config file format

Each configuration file will be used to create one java-class derived from the OperationStatus-class.

Configuration files are written in the syntax of Java property files which contain tag/value pairs with a ":" as delimiter. A # at the line beginning can be used for comments. Empty lines are ignored. A configuration file starts with a header followed by status message definitions.

The header of a configuration file must contain the following configuration-parameters:

```
# this integer will be added to all codes - the combination  
# PREFIX+CODE must be unique for all operation stati in the system
```



```

# This PREFIX can be an Integer or a String. In case it is an Integer
# the error Code of an Operation Status will be added to the prefix.
# In case it is a String, the ERROR Code will be appended to the prefix
# with an underscore
PREFIX : 1000
# PREFIX : ASC

# The operation status factory class to be generated
TARGET-CLASS: test.tresos.plugin.generated.operationstatus.MyOperationStatus

# Optional: if the generated class should extend another class
# as OperationStatus, this class can be given here
# Note, that this class must be derived from OperationStatus
EXTENDS: MyAbstractOperationStatus

# Optional: if the generated class should implement one or several
# Interfaces, they can be added here separated by a semicolon
IMPLEMENTES : ISomeClass

# The imports are needed for the different node types and
# parameters of the factory methods (see PARAMs below)
IMPORT : dreisoft.tresos.datamodel2.api.DCtxt

# Optional: The javadoc-description of the generated class can be edited here
DESCRIPTION : This class contains several status for
application-specific errors and warnings

# specifies the registration-name under which the resource-bundle
# is registered at the ResourceHandler and the full-qualified
# properties-file-name (to which all messages will be written)
# <Bundle-Name>=<File-Name>
BUNDLE-NAME : status=test.tresos.plugin.resources.OperationStatus

Example:
PREFIX      : 1000
TARGET-CLASS: dreisoft.generated.operationstatus.MyOperationStatus
BUNDLE-NAME : status=dreisoft.generated.resources.OperationStatus

```

The header is followed by the definition of the status messages. A complete entry for one message looks like this:

```

ID : EXAMPLE_STATUS
CODE : 1
SEVERITY: ERROR
PARAM : String value=description of parameter 0

```



```
PARAM : DCtxt ctxt=description of parameter 1
L-en : the english message with two parameters: {0} {1}
L-de : die deutsche Meldung mit zwei Parametern: {0} {1}
```

## ID

The ID is used as the name of the factory method and also used to generate a constant variable with the name `CODE_<Id>` that stores the code of the message. The ID must be unique within one file.

## CODE

The CODE is the status code of the status message. The code is a 32bit integer. The prefix defined in the header is added to the CODE to form the status code. The status code must be unique in the system.

## SEVERITY

The severity level of the message: ERROR, USER\_ERROR, WARNING, USER\_WARNING, INFO. If the SEVERITY start with USER\_ the message is flagged as error type user error or user warning.

## PARAM

A PARAM defines a parameter with which the message of an operation status can be parameterized. The parameter has the following form:

```
"PARAM" ":" [ [ <Class> ] <parameterName> "=" ]
<description>
```

The Class must be added as IMPORT within the file-header. Primitive types (e.g. int, float) can be used too. The default-Class is String. If no parameterName is given, a name name of the form <param><number> will be generated.

Example:

```
PARAM : INode node=the node to describe
PARAM : value=a string-value
PARAM : a description for a string-parameter
```

## L-\*

The L-\* are the localized messages for the status message where \* is the name of the language (e.g. "en" for English, which is mandatory). The value can contain parameter references to the parameters defined with the PARAM keyword in the form defined by the `java.text.MessageFormat`.

### 8.1.2.4. Operation Status Examples

The following example of an ExternalGenerator shows how an APIOperationStatus is used normally.



```

public void generate( CodeGeneratorContext context ) {

    int number = context.getDCtxtVariable().var.getInt( "path/to/a/number" );
    APIOperationStatus status = APIOperationStatus.getOkStatus();
    checkNumber( number, status );

    if(!status.isOk()) {
        // do something else
    }
}

public void checkNumber( int number, APIOperationStatus status ) {

    APIOperationStatus s = APIOperationStatus.getOkStatus();

    if( number > 100 ) {
        s = MyEclipseOperationStatus.NUMBER_ERROR_STATUS( number );
    }
    // do something else

    // add the s to status at the end
    if(!s.isOk()) {
        status.addStatus(s);
    }
}

```

The corresponding APIOperationStatus config-file should look like the following.

```

# Copyright (C) EB Automotive - All rights reserved
#
# OperationStatus Messages for Autosar plugin handling.
# The structure of the file is described in the corresponding design-document
#

PREFIX      : EXAMPLE
TARGET-CLASS : test.tresos.plugin.generated.operationstatus.
               MyEclipseOperationStatus
BUNDLE-NAME : MyEclipseOperationStatus=test.tresos.plugin.generated.
               resources.APIOperationStatus
EXTENDS     : dreisoft.tresos.lib2.api.log.APIOperationStatus

#
# ConfigManager
#

```



```

ID      : NUMBER_ERROR_STATUS
CODE    : 1
SEV     : ERROR
PARAM   : int number=the number which is wrong
L-en    : A wrong number was detected: {0}

```

## 8.1.3. The NLS/OS builder

After you have once configured the NLS/OS files in the project **Properties** dialog on page **NLS Preferences** (see e.g. [Section 8.1.1.2, “Workflow for NLS-support in Java code”](#)), the NLS/OS property files and classes are generated automatically while the project is building. To enable that the generated property files and classes are updated immediately after you apply any changes to the configured NLS/OS files, enable the **Build Automatically** option in the **Project** menu.

### 8.1.3.1. Commandline support

You can also generate the NLS/OS property files and classes for single projects with help of the command line:

```
eclipsec -nosplash -application dreisoft.tresos.sdk.api.plugin.NLSToolCommandline
<path_to_the_plugin_project>
```

This will invoke the NLS/OS generator for the specified project.

## 8.2. Guided Configuration API

### 8.2.1. Purpose

The Guided Configuration API provides the possibility to enhance EB tresos Studio with your own dialogs or editors. With the help of the Guided Configuration API, you can create:

- ▶ automatic value calculation wizards, e.g. to calculate the Handle IDs or to recalculate specific values,
- ▶ custom module editors to simplify the configuration of a module,
- ▶ wizards to simplify often conducted configuration steps, also e.g. along several modules.



## 8.2.2. Prerequisites

### 8.2.2.1. Knowledge required

In order to work with the instructions in this chapter, you need knowledge of the following topics:

#### XML

The data of the guided configuration wizards is stored persistently in XML files.

#### DCtxt

The configuration data of a module is stored in the generic `DataModel` which you can access via DCtxt Public API ([Section 5.6.2, “API”](#)). If you want to exchange data between the generic `DataModel` and a wizard, you need knowledge of how to use the DCtxt Public API.

### 8.2.2.2. Plug-ins required

After you set up Eclipse as described in [Section 4.2, “How to install Eclipse”](#), create a plug-in project and then add the following plug-ins as dependencies to the `MANIFEST.MF`:

- ▶ `dreisoft.tresos.guidedconfig.api.plugin`
- ▶ `dreisoft.tresos.lib2.api.plugin`
- ▶ `org.eclipse.swt`

Optional plug-ins:

- ▶ `dreisoft.tresos.datamodel2.api.plugin` if you want to use the DCtxt Public API

To add these plug-ins as dependencies:

- ▶ Open the `plugin.xml` or `MANIFEST.MF` within Eclipse.  
The **Plug-in Manifest Editor** opens up.
- ▶ Select the tab **Dependencies**.
- ▶ Add all plug-ins with the **Add...** button.

## 8.2.3. Design overview

This chapter gives a short overview of the design of the guided configuration framework. The following text introduces new terms to you. All new terms are written in italics. You can find a table of these new terms with an explanation in [Table 8.1, “Glossary”](#).



Each guided configuration wizard is built on the model-view-controller pattern.

#### Model

The model part is stored in a memento tree.

The wizards store data persistently in XML files, which are easily accessible through the `Memento` and `MementoOperationHandler` class. For details about the data representation, see [Section 8.2.3.4, “Data representation”](#).

#### View

The view part is represented by the *Wizard Page* that consists of one or more pages.

#### Controller

The controller part is represented by the `Backend` class that manages the lifecycle of the wizard. Find more information about the lifecycle below and about the methods that control the lifecycle in chapter [Section 8.2.3.3, “Lifecycle”](#)

There exist four different kinds of wizards. Each wizard type has one specific spot in EB tresos Studio from where it can be opened. These spots are called *docking points*. For further information about the different wizard types, see [Section 8.2.3.1, “Wizard type overview”](#).

A wizard always depends on a project. To restrict whether a wizard is to be available for a specific project, the *Selection context* can be used. For information on how to restrict the visibility of a wizard, see [Section 8.2.6.3, “Restricting the visibility of a wizard”](#).

When you open the guided configuration wizard, EB tresos Studio displays the GUI with all the *widgets*. The data, stored persistently in XML files, is loaded and displayed in the GUI. Furthermore the state of the widgets (e.g. the column width of a table) is loaded and the widgets are adapted.

The Guided Configuration API provides a collection of basic GUI widgets you can create via the *WidgetFactory* class. If these widgets do not meet your requirements, you can create and use your own custom widgets.

The GUI validates the data after each data change and displays markers for errors and warnings. You can navigate to the erroneous parameter via a click on the error message.

When you close the wizard, the `Backend` class validates the data. If no errors are reported, the wizard is triggered to run.

Then the data is written to an XML file. Optionally, you can store data in the `DataModel` by using the *push service*. For details about the *push service*, see [Section 8.2.3.6, “The push service pattern”](#). You can access to the generic configuration `DataModel` through a `DataModel` API, which writes data directly to the configuration tree.

You can add your own *result widget* to display users the results of the wizard run, e.g. which data has been changed.

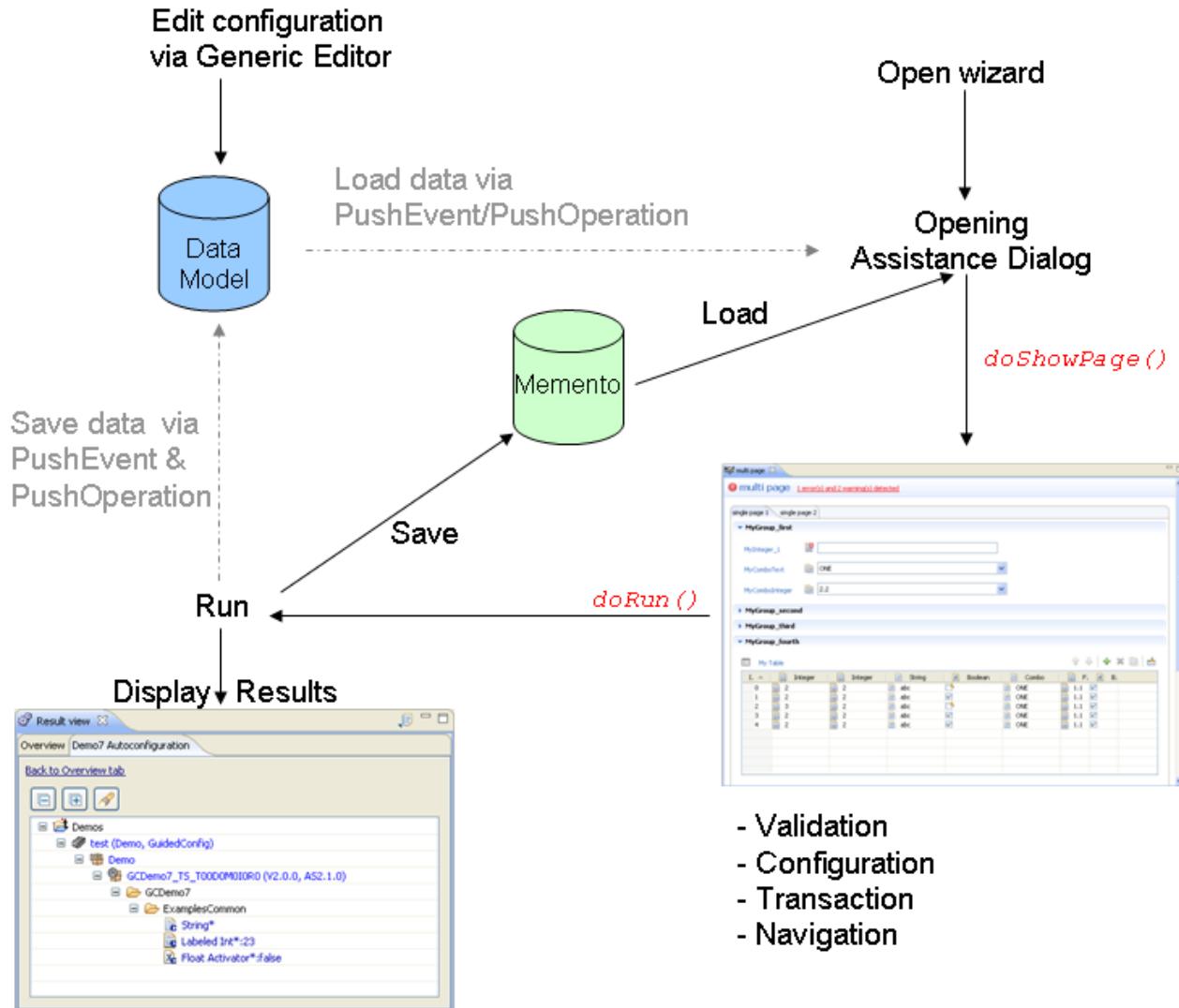


Figure 8.3. Guided configuration workflow

**TIP****The wise owl is the icon of the guided configuration**

The characteristic icon of the guided configuration wizards is the wise owl .

The following terms are important for the understanding of the further text:

Term	Description
Backend	The <i>backend</i> extends the <code>AbstractBackend</code> class and handles the background functionality of a wizard. As in the model-view-controller concept it corresponds to the controller. It manages the lifecycle of the guided configuration wizard. For more information about the lifecycle of a wizard, see <a href="#">Section 8.2.3.3, “Lifecycle”</a> .



Term	Description
Selection context	<p>There exist several classes (<code>ECUConfigContext</code>, <code>ModuleConfigContext</code>, and <code>SystemConfigContext</code>) that provide information about the currently selected project and module. You can restrict the visibility of docking points in the GUI or execution of <i>push operations</i> with the help of the <i>selection context</i>. You find more details about the <i>selection context</i> in <a href="#">Section 8.2.3.2, “Selection context”</a>.</p>
Memento	<p>The <code>Memento</code> class corresponds to the model in the model-view-controller concept as it is the data representation of the guided configuration wizard.</p> <p>The data is stored in a memento tree with the root memento as top node.</p> <p>Two classes provide access to the guided configuration data layer, <code>Memento</code> and <code>MementoOperationHandler</code>. Refer to <a href="#">Section 8.2.3.4, “Data representation”</a> for information about when to use which class.</p>
Wizard page	<p>A wizard page is the viewable part of a wizard and can consist of one or more pages. You can open a wizard page in an editor or a dialog.</p> <p>You can either:</p> <ul style="list-style-type: none"> <li>▶ create wizard pages coding in Java by extending the classes <code>AbstractPage</code> or <code>AbstractMultiPage</code>. Instructions are available at <a href="#">Section 8.2.6.5, “Creating the page”</a>.</li> <li>▶ or define the wizard pages by using the XForms page description. To do that, register the <code>XFormPage</code> or the <code>XFormMultiPage</code>. For more information see <a href="#">Section 8.2.6.4, “Using XForms as GUI description”</a>.</li> </ul>
Widget	<p>A widget is a GUI element for displaying values, that has its own data representation. The Guided Configuration API provides a set of basic GUI widgets. For more information see <a href="#">Section 8.2.3.5.5, “Predefined widgets”</a>.</p>
WidgetFactory	<p>The <code>WidgetFactory</code> class contains methods for the creation of GUI widgets and also a method to register custom widgets.</p>
Push service	<p>The Guided Configuration API provides a pattern called <i>push service</i> to encapsulate the data exchange between the data of the guided configuration wizard (memento) and the <code>DataModel</code> configuration (DCtxt).</p> <p>The <i>push service</i> consists of <i>push events</i> and <i>push operations</i>. The backend implementation sends a <i>push event</i> and the corresponding <i>push operations</i> are executed. The operations are used to exchange data between the generic <code>DataModel</code> and the mementos. For details about <i>push events</i> and <i>push operations</i>, refer to <a href="#">Section 8.2.3.6, “The push service pattern”</a>.</p>
Push event	<p>You send a <i>push event</i> to start a specific <i>push operation</i>. The push event is bidirectional and can return the result from the push operation to the <code>Backend</code> class.</p>



Term	Description
Push operation	The <i>push operation</i> is responsible for the data transfer from the wizard to the data model and back. A push operation can be reused for more than one wizard by passing a specific event to which it listens.
Result widget	The Guided Configuration API provides the possibility to register a <i>result widget</i> , which displays the result of a wizard in a formatted way. ( <a href="#">Section 8.2.3.5.7, “Results view”</a> )

Table 8.1. Glossary

### 8.2.3.1. Wizard type overview

You can open a wizard from various places in EB tresos Studio. These places are called *docking points*. The following docking points are available:

- ▶ Project Explorer
- ▶ Unattended wizard
- ▶ Sidebar view

To be able to use a wizard, define and register the following two parts:

- ▶ [Section 8.2.6.1, “Registering a wizard”](#) with its GUI representation and lifecycle.
- ▶ [Section 8.2.6.2, “Registering the docking point”](#), in which the wizard appears in the EB tresos Studio GUI.

Therefore it is possible that several docking points share the same wizard registration.

Based on the docking points, the Guided Configuration API distinguishes different kinds of wizards, namely:

- ▶ Sidebar wizard, which can appear as
  - ▶ dialog: [Section 8.2.3.1.2, “Dialog”](#)
  - ▶ or editor: [Section 8.2.3.1.1, “Standalone editor”](#)
- ▶ unattended wizard: [Section 8.2.3.1.3, “Unattended wizard”](#)
- ▶ custom module wizard: [Section 8.2.3.1.4, “Custom module editor”](#)

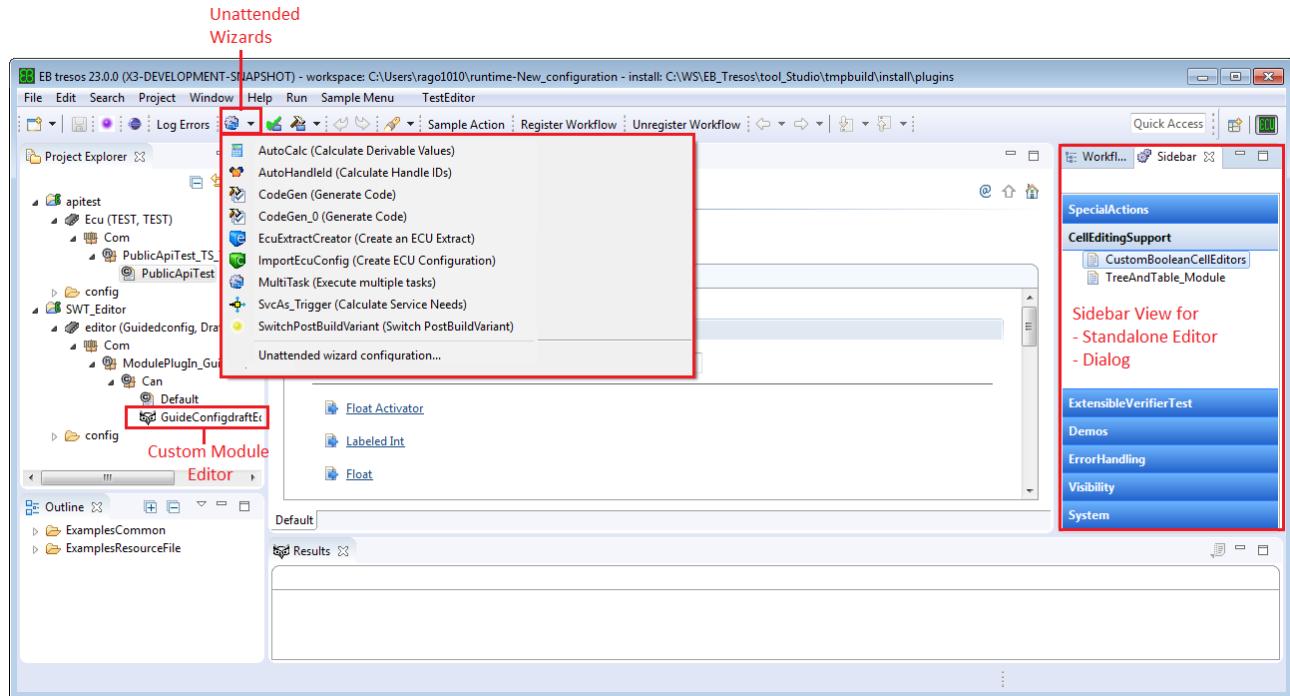


Figure 8.4. Guided configuration wizard type overview

**NOTE****Register the docking points via trigger extension points**

You can only register the docking points in the **Sidebar** view of the EB tresos Studio GUI and the **Unattended Wizards** configuration dialog via the trigger extension point `dreisoft.-tresos.guidedconfig.api.plugin.trigger`. Thus these docking points are also called *trigger*.

### 8.2.3.1.1. Standalone editor

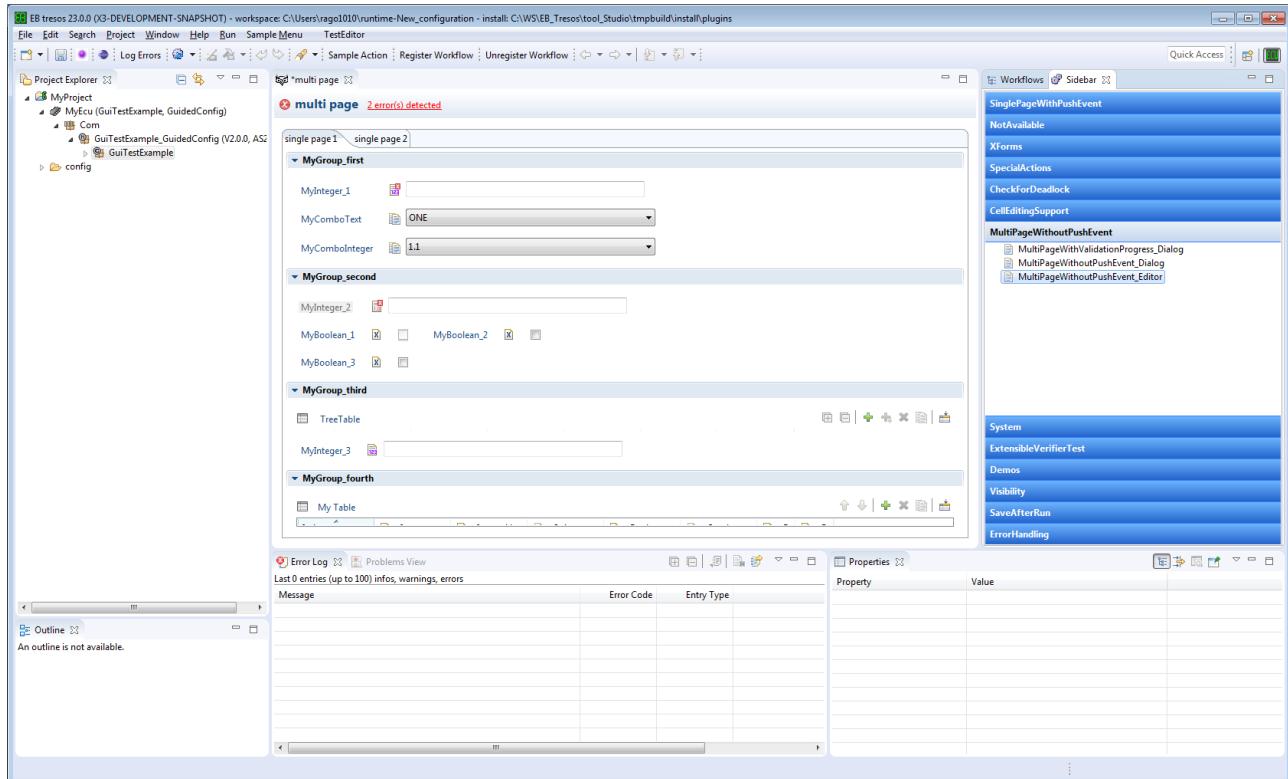


Figure 8.5. Standalone editor

#### Docking point

The standalone editor is only available in the **Sidebar** view.

#### Error handling

Errors and warnings are displayed at the top of the editor as a hyperlink. The hyperlink summarizes how many errors and warnings appear in the wizard. Clicking the hyperlink opens a list of all errors and warnings. It is possible to navigate to each erroneous parameter by selecting an error or warning from the list.

#### Undo/redo support

Users may use the undo/redo functionality.

#### Multi-page support

Users may switch between pages by clicking into another tab page in the respective editor.



## Running the wizard

Closing the editor triggers the wizard to run. If the wizard has errors, the wizard cannot run. In this case an error dialog opens up which asks the users whether they want to close the editor and loose its data, or whether they want to return to the wizard to change the configuration. The results of the wizard run are logged to the EB tresos Studio **Error Log**.

### 8.2.3.1.2. Dialog

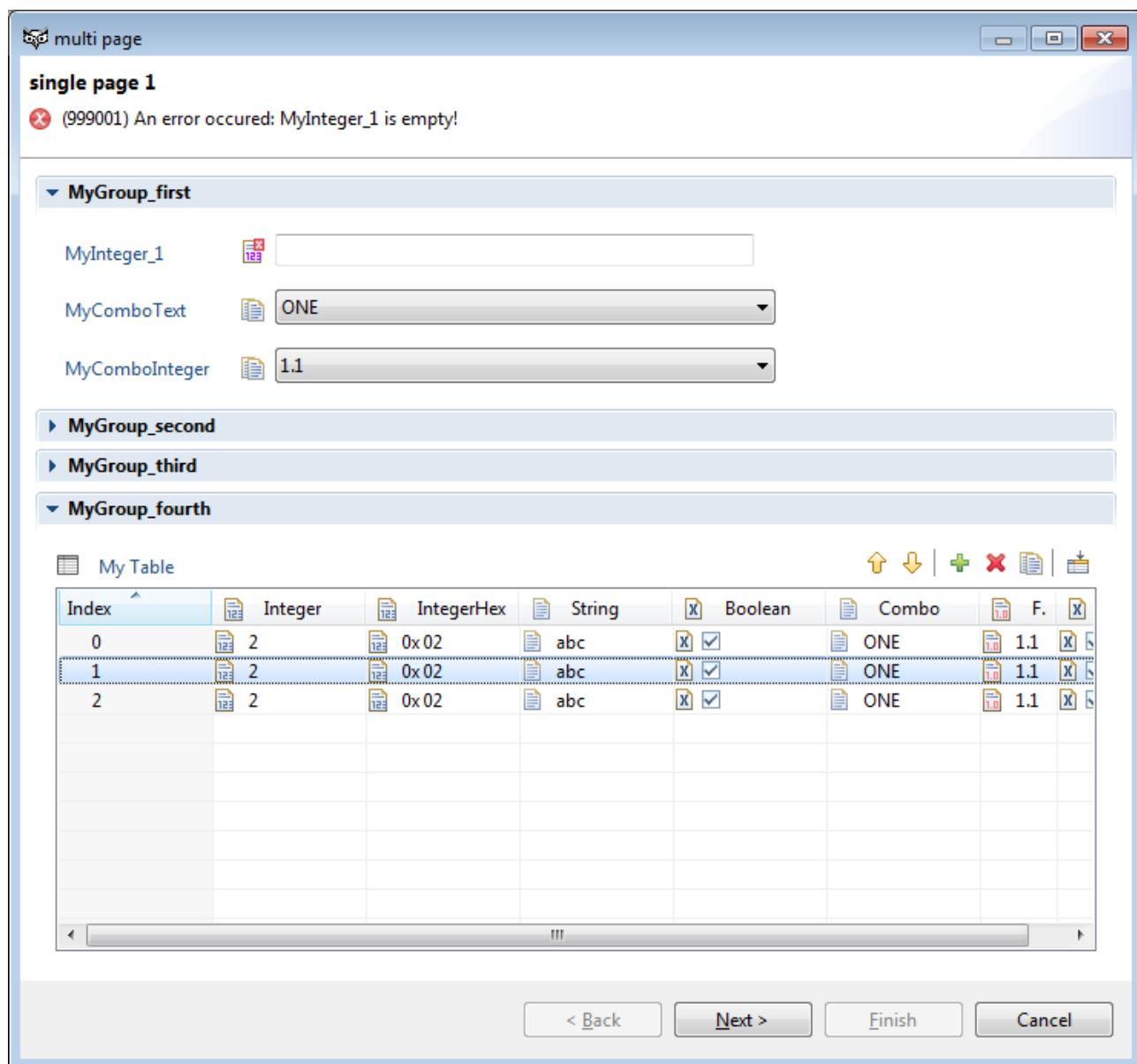


Figure 8.6. Dialog

## Docking point

The modal dialog is only available in the **Sidebar** view.



### Error handling

If the wizard has errors or warnings, only one item is displayed at a time on the top of the page. As soon as the displayed error is fixed, the next error/warning of the list is displayed. First the errors are displayed. Only if no errors exist anymore, the warnings are displayed one by one. It is not possible to navigate to the parameter by clicking the error message.

Changing a parameter triggers an asynchronous validation run. While validation is running, all buttons except the **Cancel** button are disabled. For long running validations a progress bar is shown. Nevertheless it is always possible to change parameters in the GUI, even while a validation is running.

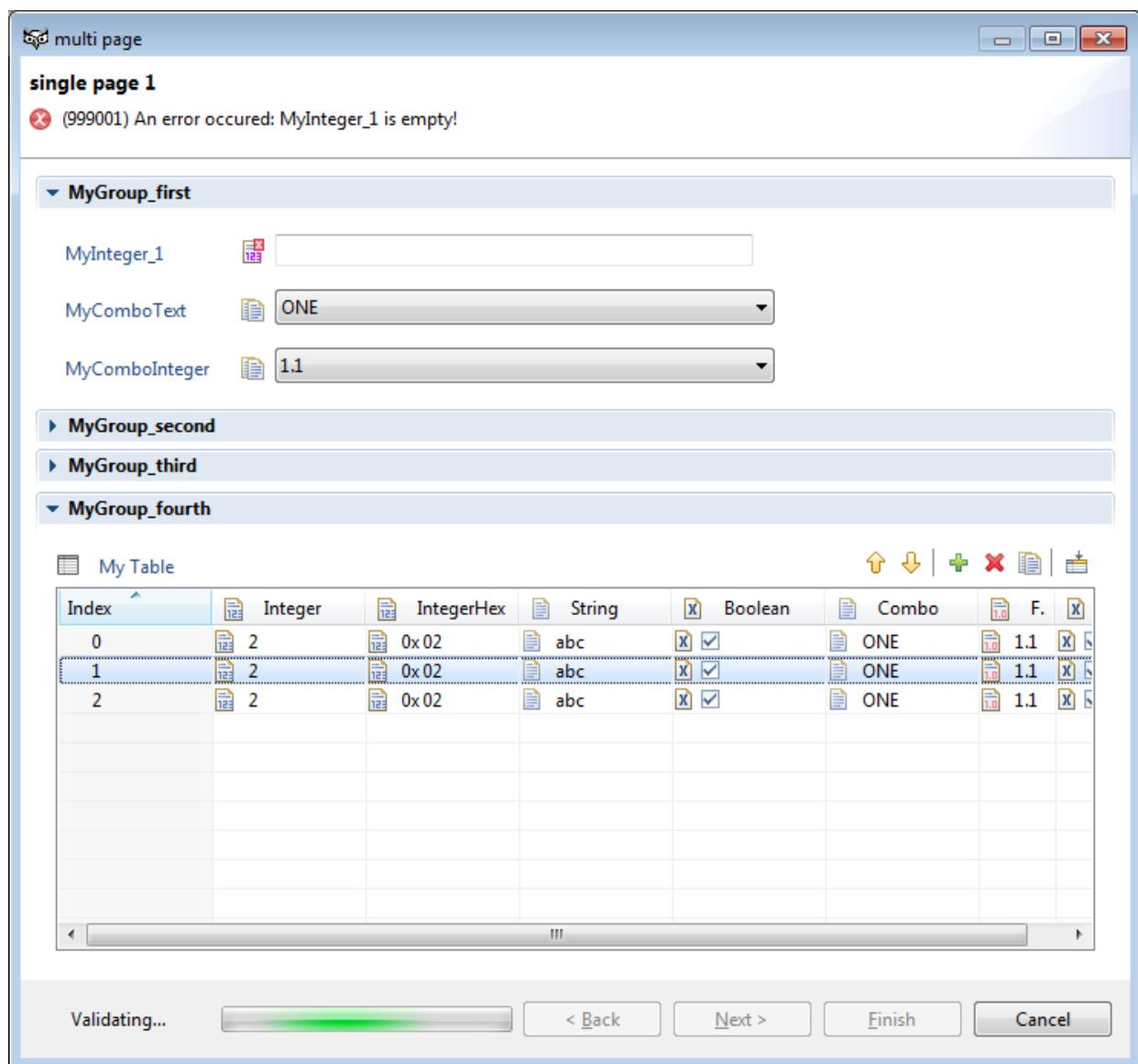


Figure 8.7. Validation is running in dialog



### Undo/redo support

The dialog by default provides undo/redo support via buttons and the keyboard shortcut **Ctrl+Z/Y**. The buttons are located on the top of the dialog.

You can disable the undo/redo support for dialogs in the extension point for special use cases.

### Multi-page support

Users can switch from one page to another by using the **Back** and **Next** buttons.

### Running the wizard

You can close the dialog with the **Finish** button. This button is only enabled if the wizard has no errors.

After clicking the **Cancel** button, a confirmation message opens up, which asks users if they want to exit the dialog or return to the dialog. The data of the dialog is always saved. The results of the wizard run are displayed in the **Results** view.

### Dialog's initial size support

You can specify the dialog's initial size in the extension point through the `width` and `height` attributes.

#### 8.2.3.1.3. Unattended wizard

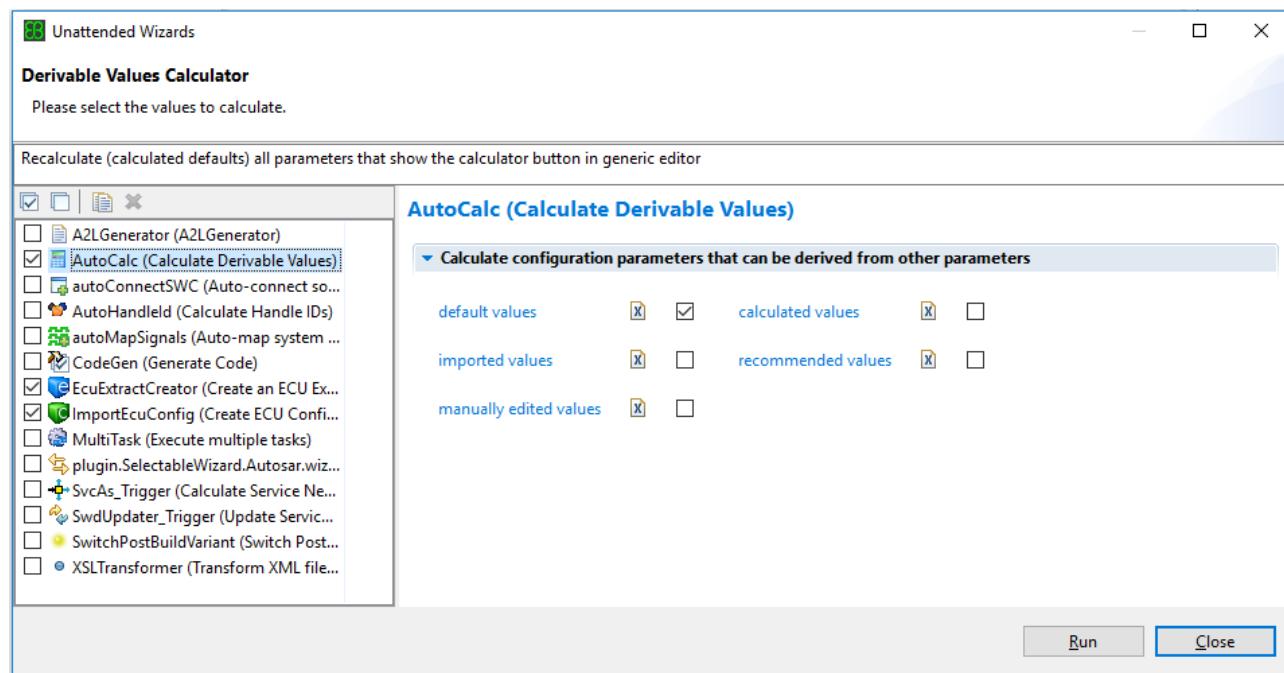


Figure 8.8. The **Unattended Wizards** configuration dialog

### Docking point

The unattended wizard configuration is only available in the **Unattended Wizards** configuration dialog.



## Error handling

If the wizard has errors or warnings, only one error or warning is displayed on the top of the page. As soon as the displayed error is fixed, the next error is displayed. It is not possible to navigate to the parameter via the error.

## Undo/redo support

The dialog does not provide undo/redo support.

## Multi-page support

The user can switch from one page to another by clicking into the tab of another page in the right part of the **Unattended Wizards** configuration dialog.

## Running the wizard

If the unattended wizard has been configured correctly in the GUI, it can be run without user interaction. The user can start the unattended wizard either from the GUI or from the command line. If the user starts the unattended wizard from the GUI, the results of the wizard run are displayed in the **Results** view.

The GUI provides several possibilities to run a wizard:

- ▶ In the **Unattended Wizards** configuration dialog, select one wizard and click the **Run** button.
- ▶ In the **Project** menu, select **Unattended Wizards** and select the wizard you want to run.
- ▶ In the EB tresos Studio tool bar, click the **View** button next to the **Open unattended wizard configuration dialog** button . A menu opens up displaying the previously selected wizards (favorites). Select the wizard of your choice.

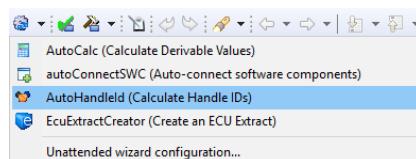


Figure 8.9. Choosing the desired unattended wizard

- ▶ After selecting the wizards in the **Unattended Wizards** configuration dialog, click the **Open unattended wizard configuration dialog** button which will open the unattended wizard dialog.

The unattended wizard trigger can also be executed via the command line in addition to the GUI.

```
tresos_cmd.bat autoconfigure <projectname> [<triggerid>, ...]
```

If only `<projectname>` is specified, all available and selected unattended wizard triggers for the given project are executed. There is also the possibility to restrict the execution by specifying a comma-separated list of triggers. This list is selected one after the other. The trigger is only executed if it exists and if the visibility restrictions for the given project are fulfilled.



### 8.2.3.1.4. Custom module editor

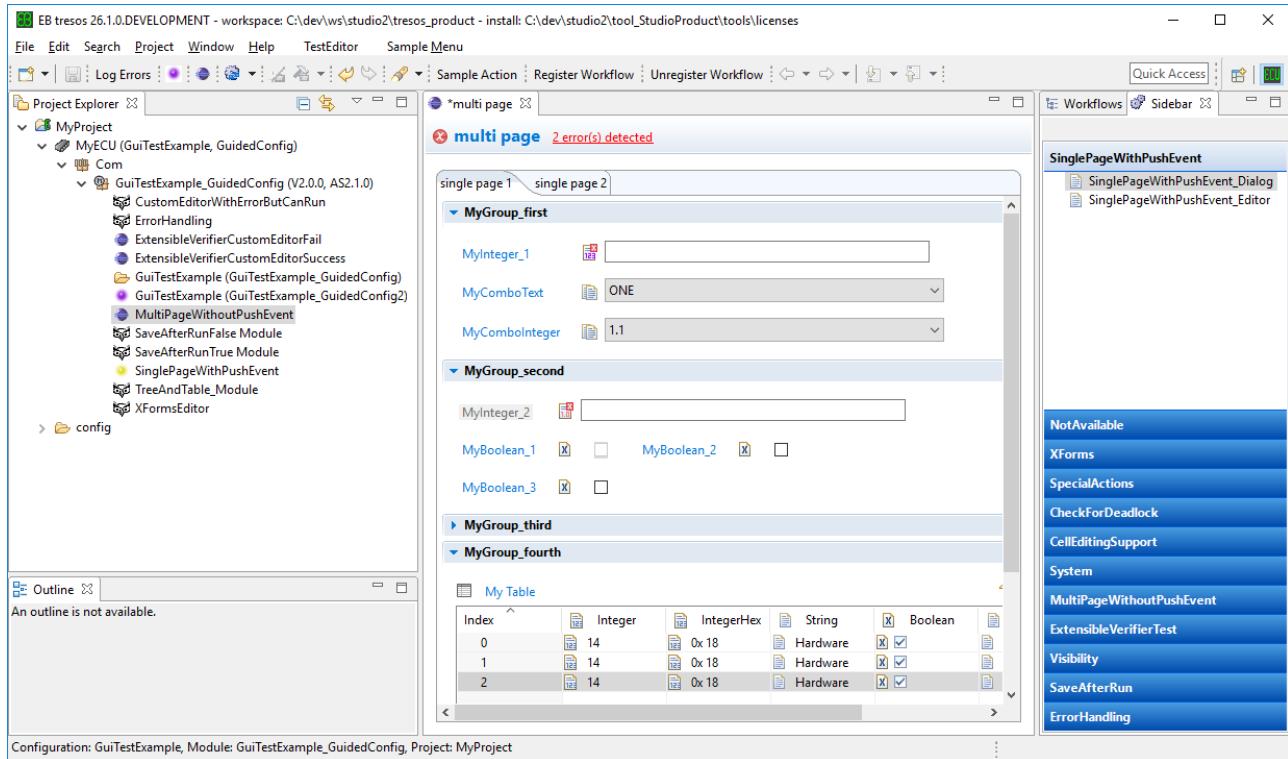


Figure 8.10. Custom module editor

#### Docking point

The custom module editor is displayed in the **Project Explorer** in the same way as the Generic Configuration Editor.

#### Error handling

Errors and warnings are displayed at the top of the editor as a hyperlink. The hyperlink summarizes how many errors and warnings appear in the wizard. Clicking the hyperlink opens a list of all errors and warnings. It is possible to navigate to each erroneous parameter by selecting an error or warning from the list.

#### Undo/redo support

Users can use the undo/redo functionality.

#### Multi-page support

Users can switch between multiple pages by clicking into the tab of another page in the editor part.

#### Running the wizard

Closing the editor triggers the wizard to run. If the wizard has errors, the wizard cannot run. In this case an error message opens up asking the users whether they want to close the editor and loose its data, or whether they rather want to return to the wizard to change the configuration. The results of the wizard run are displayed in the EB tresos Studio **Error Log** view.



### 8.2.3.2. Selection context

You can restrict the visibility of wizards that are registered via the extension point `dreisoft.tresos.guidedconfig.api.plugin.trigger`, aka. `trigger`. This depends on the current *selection context*. To run a wizard, you always need a loaded project.

To find out how to restrict the visibility of a wizard, see [Section 8.2.6.3, “Restricting the visibility of a wizard”](#).

Several classes exist that provide information about the currently selected project and - if applicable - about the selected module.

The following context classes are available:

- ▶ `ECUConfigContext` provides information about the currently selected project. If no project is selected or the selected project is not loaded, this context is disabled.
- ▶ `ModuleConfigContext` provides information about the currently selected module. If no information is available, this context is disabled.
- ▶ `SystemConfigContext` is enabled if the currently selected project contains a system model.

Each context class also provides the possibility to check whether the current context is available at the moment. E.g. for `ECUConfigContext` you can check whether a loaded project is selected.

The following sections provide you with information about the context classes and the data you can use to restrict the visibility of the docking points and the runnability of *push operations*:

- ▶ [Section 8.2.3.2.1, “The `ECUConfigContext` class”](#)
- ▶ [Section 8.2.3.2.2, “The `ModuleConfigContext` class”](#)
- ▶ [Section 8.2.3.2.3, “The `SystemConfigContext` class”](#)

The contexts are available via the `ContextManager` class. Each context has a unique ID and can be queried by using:

```
ContextManager.getInstance().getContext(ECUConfigContext.ID)
```

When opening a wizard, all contexts are cloned and given to the `Backend` class. Like this the information from which project the wizard is opened does not get lost.

You can query the list of the cloned contexts by calling the method `getcontexts()` of the `Backend` class.

#### 8.2.3.2.1. The `ECUConfigContext` class

The `ECUConfigContext` class provides information about the currently selected project.

This context provides the following data:



- ▶ `projectname`: the name of the currently selected project
- ▶ `target`: the target of the project (e.g. TRICORE)
- ▶ `derivate`: the derivative (e.g. TC1766)
- ▶ `relVersion`: the release version (e.g. 2.1.1, 2.1 = 2.1.\* , 2 = 2.\* = 2.\*.\* )
- ▶ `ecuid`: the ECU ID
- ▶ `generationPath`: the generation path set in the project

You can query the data in the following way:

```
ECUConfigContext ecuContext = (ECUConfigContext)getContexts() .
    get(ECUConfigContext.ID);
ecuContext.getAttribute(ECUConfigContext.ATTR_PROJECT_NAME);
```

The `ECUConfigContext` class provides additional information about the modules in the selected project: For each module of the currently selected project the following attributes are provided:

- ▶ `moduleId.<ModuleName>`

If at least one module configuration with this ID is available and enabled, the value is set to TRUE. Example:  
`attribute:moduleId.Can_TS_T16D6M2I1R0;value=TRUE` .

You can query the enabled `moduleIds` in the following way:

```
ECUConfigContext ecuContext = (ECUConfigContext)getContexts() .
    get(ECUConfigContext.ID);
ecuContext.getModuleIds();
```

- ▶ `moduleType.<TypeName>`

If at least one module configuration with this type is available and enabled, the value is set to TRUE.  
Example: `attribute:moduleType.Can;value=TRUE`

You can query the enabled `moduleTypes` in the following way:

```
ECUConfigContext ecuContext = (ECUConfigContext)getContexts() .
    get(ECUConfigContext.ID);
ecuContext.getModuleTypes();
```

The `ECUConfigContext` class also allows you to query the root DCtxt of the currently selected project:

```
ECUConfigContext ecuContext = (ECUConfigContext)getContexts() .
```



```
get(ECUConfigContext.ID);  
DCtxt rootDCtxt = ecuContext.getDCtxt();
```

### 8.2.3.2.2. The ModuleConfigContext class

The `ModuleConfigContext` class provides information about the modules of the currently selected project. The `ModuleConfigContext` class provides the following data:

- ▶ `schemapath`: the schema path of the currently selected node in the editor. If no node is selected in the editor, `schemapath` is empty.
- ▶ `datapath`: the data path of the currently selected node in the editor. If no node is selected in the editor, `datapath` is empty.
- ▶ `editorid`: the ID of the currently selected editor.
- ▶ `moduleid`: the module ID.
- ▶ `configurationid`: the configuration ID.
- ▶ `dctxt`: the root DCtxt of the currently selected configuration.
- ▶ `selectedDCtxt`: the DCtxt of the currently selected node.
- ▶ `type`: the type of the module (e.g. PduR).
- ▶ `category`: the module category (e.g. Com).
- ▶ `layer`: the module layer (e.g. Service).
- ▶ `specVersion`: the module specification version (e.g. 2.1.1, 2.1 = 2.1.\* , 2 = 2.\* = 2.\*.\*).
- ▶ `swVersion`: the software version of the module (e.g. 2.1.1, 2.1 = 2.1.\* , 2 = 2.\* = 2.\*.\*)
- ▶ `relVersion`: the release version of the module (e.g. 2.1.1, 2.1 = 2.1.\* , 2 = 2.\* = 2.\*.\*).

You can query the data in the following way:

```
ModuleConfigContext moduleContext = (ModuleConfigContext)getContexts().  
                                get(ModuleConfigContext.ID);  
moduleContext.getAttribute(ModuleConfigContext.ATTR_MODULE_ID);
```

You can also query the different DCxts of the `ModuleConfigContext` class by calling:

```
ModuleConfigContext moduleContext = (ModuleConfigContext)getContexts().  
                                get(ModuleConfigContext.ID);  
DCtxt configurationRootDCtxt = moduleContext.getDCtxt();  
DCtxt currentlySelectedNodeDCtxt = moduleContext.getSelectedDCtxt();
```

### 8.2.3.2.3. The SystemConfigContext class

The `SystemConfigContext` class provides information if the currently selected project has system model support. The data can be used to restrict the visibility of the triggers and the runnability of push operations. The `SystemConfigContext` class provides the following data:

- ▶ `projectname`: the name of the currently selected project.

### 8.2.3.3. Lifecycle

The lifecycle of a wizard is represented by the `Backend` class. At special times in this lifecycle, you can modify the wizard's behavior by overwriting specific methods in your `Backend` implementation.

The following figure shows the time line of the lifecycle methods that can be overridden in your `Backend` class.

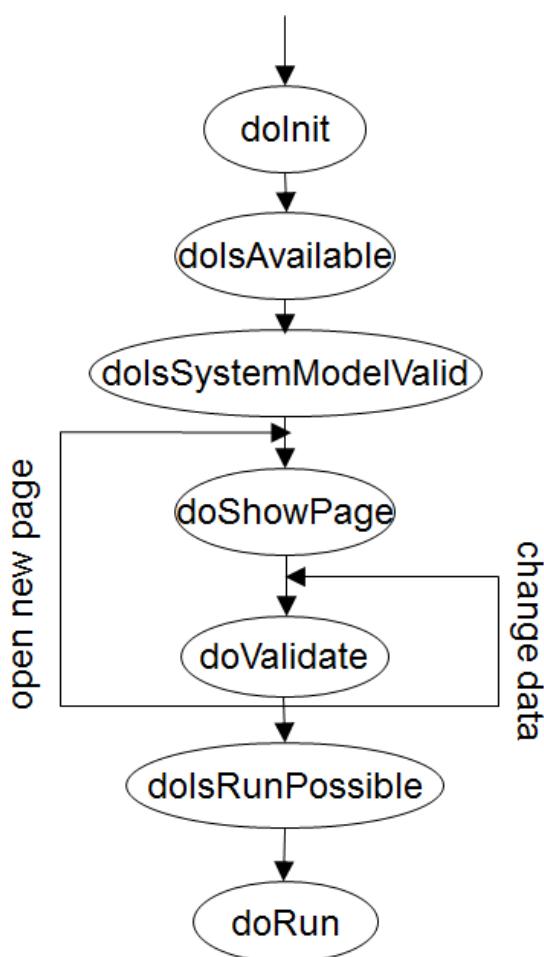


Figure 8.11. Wizard lifecycle



Backend lifecycle method	Description
doInit()	Implement this method to initialize your Backend class. <code>doInit()</code> is called right after instantiation of the backend before anything else is initialized. This means that the root memento has not yet been created and there is no GUI. This method is only called once in the wizard lifecycle.
doIsAvailable()	Implement this method if there are restrictions under which the wizard cannot be displayed. Return the reason why the wizard cannot be opened as <code>APIOperationStatus</code> . The return code is then displayed to the user. This method is called once right before the wizard is opened.
doIsSystemModelValid()	Implement this method if there are restrictions in the system model under which the wizard cannot be displayed. You can use a standard AutosarVerifier which verifies that all mandatory elements are available or you implement your own custom verifier which checks all your requirements to the system model. Return the reason why the wizard cannot be opened as <code>APIOperationStatus</code> . The return code is then displayed to the user. This method is called once right before the wizard is opened.
doShowPage()	This method is called each time a new page is displayed. For multi-pages this method is also called for each page switch. Implement this method to load the data and fill it into the GUI data representation.
doValidate()	Implement this method to validate the data so that the validation results are made visible in the GUI as errors and warnings. This method is called when the wizard is opened and before it is finished and each time a value is changed.
doIsRunPossible()	Implement this method if you want to execute the wizard even if there exist validation errors. The default implementation returns false if there exist any validation errors and therefore by default a wizard with errors does not run at all.
doRun	<p>This method does the real work for this wizard, e.g. calculating values or changing the configuration in the data model. This method is called once when the wizard is run. The wizard is only run if <code>doIsRunPossible()</code> returns true.</p> <p>Two possible implementations exist for the <code>doRun</code> function:</p> <ul style="list-style-type: none"> <li><i>doRun():</i> This method runs in the UI thread and is intended to be used for short operations only.</li> <li><i>doRun(IProgressMonitor):</i> This method runs in the backend and it is intended to use this function for long running operations. It is recommended to show information about the progress via the <code>IProgressMonitor</code>. Note that access to the UI needs to be encapsulated into an <code>UI.run</code> block.</li> </ul>

Detailed information on those API functions are available at in the Studio Public Java API (see [Section 3.1, “What is the Public Java API?”](#)).



#### 8.2.3.4. Data representation

Each wizard has its own set of data, in which each widget is connected to one element. The data is organized as a tree which is stored persistently in XML files, located in the project workspace in the subfolder `.prefs` as `.mem` files. The runtime representation of the data is a tree of memento objects.

Several ways exist to manipulate the data. The recommended way to change the memento tree is to use the `MementoOperationHandler` class. This class encapsulates the direct access via the `Memento` class. The `MementoOperationHandler` class provides undo/redo support and transaction handling. For details about the `MementoOperationHandler` class, see [Section 8.2.3.4.1, “The MementoOperationHandler class”](#).

The content of a `.mem` file looks as follows:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<memento version="1.0" lastmodified="Aug 7, 2009 10:34:13 AM">
<guidedDemo4 comboText="TWO" text="3453" float="3453.0" comboNumber="1.1">
  <__metadata>
    <Group_1_0 __expanded="true"/>
    <text __radix="DEC" __separators="false"/>
    <comboText __radix="DEC" __separators="false"/>
    <table __width="50 100 100 100 100 95" __visible="yyyyyy"/>
    <Tree __width="136 100 100 100 100" __visible="yyyyyy"/>
    ....
  </__metadata>
  <table>
    <__ IntColumnMemento="2" FloatColumnMemento="1.1"
        StringColumnMemento="abc" BooleanColumnMemento="false"
        ComboBoxMemento="ONE"/>
    <__ IntColumnMemento="2" FloatColumnMemento="1.1"
        StringColumnMemento="abc" BooleanColumnMemento="false"
        ComboBoxMemento="ONE"/>
    ....
  </table>
  <Tree>
    <__ DescMemento="1.1" IntColumnMemento="2"
        StringColumnMemento="basic" BooleanColumnMemento="true"
        ComboBoxMemento="ONE">
    <__ DescMemento="1.1" IntColumnMemento="2"
        StringColumnMemento="abc"
        BooleanColumnMemento="false" ComboBoxMemento="ONE">
      <__ DescMemento="1.1" IntColumnMemento="2"
          StringColumnMemento="abc"
          BooleanColumnMemento="true" ComboBoxMemento="ONE"/>
    </__>
  </Tree>
</guidedDemo4>
```



```
</memento>
```

The XML tag `guidedDemo4` represents the root memento and is named after the wizard ID. Attributes represent widgets with single values.

In addition to the user data a node for the meta data exists, such as:

- ▶ expand state of groups
- ▶ radix settings of integer parameters
- ▶ width of table columns

More complex widgets are connected to a sub tag of the root tag. The specific data representation depends on the widget. At the moment `WidgetFactory` provides only two widgets that are connected to a sub tag:

#### table

The subtag for the table contains a list of row entries. Each attribute of the row tag represents a cell value.

#### tree table

The subtag of the tree table contains a tree of nested elements.

##### 8.2.3.4.1. The `MementoOperationHandler` class

The `MementoOperationHandler` class provides access to the data representation. It provides getter- and setter- methods for memento values, as well as methods to manipulate special widget, e.g. to create a new row for a table.

The `MementoOperationHandler` class is the main class to perform memento operations and the recommended way of working with the data. Using this class provides undo/redo support and transaction handling.

---

#### TIP

#### Adapt undo history size



You can adapt the undo history size for operations via `MementoOperationHandler`. Open the EB tresos Studio preferences, navigate to General/Editors/Text Editors and change the value for *Undo history size*.

---

Use transactions to combine several actions into one atomic action, so the user can undo these actions with one click.

The `MementoOperationHandler` class provides methods for:

- ▶ setting values of different types (String, Boolean, int, float)
- ▶ adding and deleting mementos



- ▶ **table operations** (`addRow`, `moveRow`, `removeRows`, `duplicateRows`)
- ▶ **tree table operations** (`addTreeRow`, `removeTreeRows`, `duplicateTreeRows`)

The exact syntax of the methods in the Java documentation is available at [Section 8.2.4, “API”](#).

---

**NOTE**
**Accessing MementoOperationHandler**


You can access the `MementoOperationHandler` class by using the following code, supposing that `backend` is the instance of your `Backend` class:

```
backend.getMementoOperationHandler().
```

---

Implement your own `AbstractMementoOperation` if you want to combine several actions to one atomic action or if you have to keep internal data up to date while performing undo/redo in the GUI.

To call your operation use `dreisoft.tresos.guidedconfig.api.gui.databinding.MementoOperationHandler.execute(AbstractMementoOperation)`.

#### 8.2.3.4.2. The Memento interface

The `Memento` class is the data layer access of the Guided Configuration Public API. It provides getter- and setter-methods for memento values, as well as methods to create new mementos. The underlying data has a hierarchical structure which is persistently stored in XML format. It has a top node (the table itself), a subnode for each row, and at least one attribute for each cell entry.

#### 8.2.3.4.3. MementoFactory

The `MementoFactory` class defines the construction of the memento tree. It is comparable to a Schema file in XML.

The default implementation `DefaultMementoFactory` is only a simple implementation and returns instances of `Mementos` but does not change the data.

Implement a custom factory if you want to:

- ▶ set default values, or
- ▶ calculate specific values, or
- ▶ define which memento is to be persistent or not.

A custom factory must extend `AbstractMementoFactory` and implement the method `getMemento( Memento parent, String name )`.



---

**NOTE**

**The custom `MementoFactory` class must be a singleton.**

The custom `MementoFactory` class must be a singleton.



---

### 8.2.3.5. GUI abstraction

The guided configuration framework provides common functionality for the GUI representation. The following elements are provided:

- ▶ [Section 8.2.3.5.1, “Multi-page support”](#)
- ▶ [Section 8.2.3.5.2, “Layout”](#)
- ▶ [Section 8.2.3.5.3, “Validation”](#)
- ▶ [Section 8.2.3.5.4, “Navigation”](#)
- ▶ [Section 8.2.3.5.5, “Predefined widgets”](#)
- ▶ [Section 8.2.3.5.6, “Widget lifecycle”](#)
- ▶ [Section 8.2.3.5.7, “Results view”](#)

#### 8.2.3.5.1. Multi-page support

A wizard consists of one or more pages. The guided configuration framework provides the functionalities with which the user can navigate from page to page. Depending on the kind of wizard, the user can navigate between multiple pages either via tab pages or via **Back/Next** buttons. For an explanation of which wizard uses which navigation type, see [Section 8.2.3.1, “Wizard type overview”](#).

#### 8.2.3.5.2. Layout

Each page is based on a grid layout. When you want to create a group, you lay out this grid vertically or horizontally, using the following `layoutType`:

- ▶ `WidgetFactory.GroupLayout.VERTICAL` or
- ▶ `WidgetFactory.GroupLayout.HORIZONTAL`

The default layout is the vertical one.

The vertical layout is built on four columns. Parts of the widgets span several columns to provide a nice layout. To find out which widget uses which span width, see [Section 8.2.3.5.5, “Predefined widgets”](#)



The horizontal layout consists of one row and each widget is appended on the right side.

### Group with layout HORIZONTAL

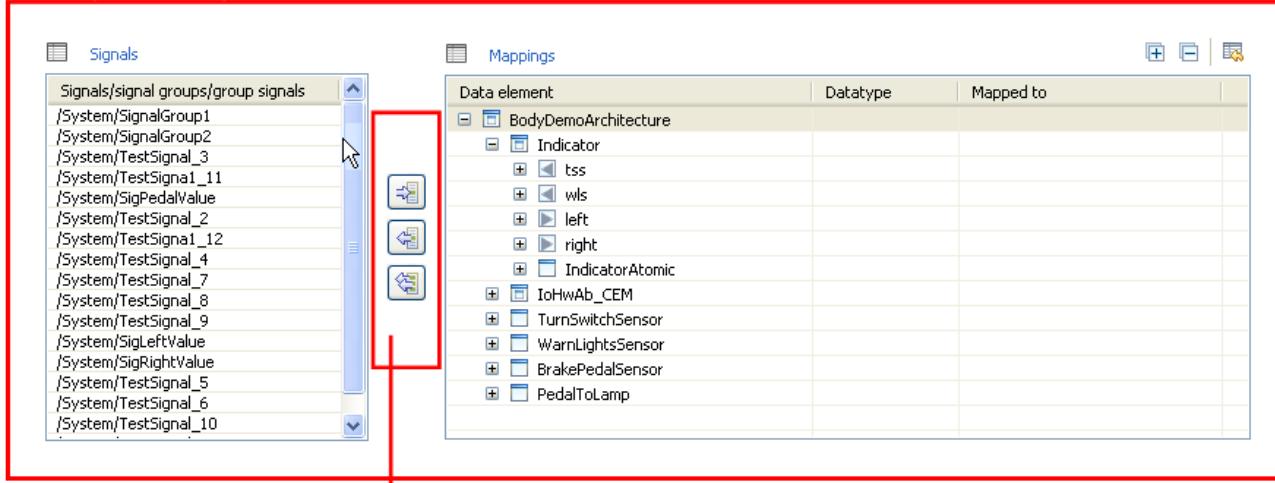


Figure 8.12. Example layout

#### 8.2.3.5.3. Validation

Validate the data in the backend `doValidate()` method. The guided configuration framework updates the error and warning markers in the GUI corresponding to the results of the validation. Each data change triggers a validation and update of the GUI.

The validation of the data is run asynchronously, so as not to disturb the user who edits the page.

The errors and warnings are displayed on top of the page: for dialogs and unattended wizards only one error or warning is displayed at a time. For standalone editors and custom module editors, all errors and warnings are displayed as hyperlinks. The hyperlinks can navigate directly to the erroneous widget, even if the widget is on another page inside the wizard. For an example, see [Section 8.2.5.2, “Demo1”](#)

For dialogs, all buttons except the **Cancel** button are disabled while validation is running. For long running validation operations, a progress bar is shown. The validation is always started asynchronously, so you can change the wizard data while validation is running. The **Finish** button is only enabled if currently no validation is running and no errors occurred during the last validation run.

For dialogs it is also possible to control the workflow by controlling whether the **Back** and **Next** buttons are enabled. This way you can prevent the user from clicking the **Next** button while there is still an error on the page. For an example, see [Section 8.2.5.6, “Demo5”](#)



#### 8.2.3.5.4. Navigation

The GUI abstraction layer also provides the possibility to add your own navigation links to a page. There exists a predefined browser widget that can contain hyperlinks. With these links it is possible to navigate to a specific widget inside a wizard.

The following example code shows how to navigate to a specific widget inside the wizard:

```
<a href="wizard://<wizard-ID>/<widget-ID>">
```

In this example the wizard-ID is the ID defined in the `dreisoft.tresos.guidedconfig.api.plugin.guidedconfigwizard` extension point. And the widget-ID is the unique ID of the widget to navigate to.

#### 8.2.3.5.5. Predefined widgets

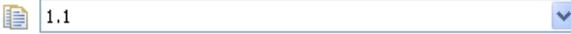
All widgets provided by the Guided Configuration API are available via the `WidgetFactory` class.

The `WidgetFactory` class provides methods for the simple creation of GUI widgets. It connects each widget to the underlying data layer (memento), by registering it with a user-chosen memento-ID. The `WidgetFactory` class also defines the layout for the widgets. The standard layout is based on a grid layout using four columns.

To be able to use the `WidgetFactory` class within the `doCreateControls()` method, call `getWidgetFactory()`. You get then the instance for this page. The `WidgetFactory` class provides the following collection of basic widgets:

Widget type	Description
Group	A grouping composite. If you use the frame type <code>TITLE</code> , the group can be collapsed. It spans all four columns in the layout. 
Label	A text label. It uses one column in the layout.  Figure 8.13. Text label
Check box	A check box that is of the type Boolean. It uses one column in the layout and has the type icon:  .
Button	A button that can either have text or an icon as label. It uses four columns in the layout. The button can have the following styles: ▶ <code>ButtonFlags.PUSH</code> , or ▶ <code>ButtonFlags.TOGGLE</code>



Widget type	Description
	 <p>Figure 8.14. Widget button</p>
Text	<p>A plain text box with a type icon. A text box can use the following types:</p> <ul style="list-style-type: none"> <li>▶ <code>TextFormat.TEXT</code>  , or</li> <li>▶ <code>TextFormat.INTEGER</code>  , or</li> <li>▶ <code>TextFormat.FLOAT</code> </li> </ul> <p>It uses three columns in the layout.</p>  <p>Figure 8.15. Text box</p>
Combo box	<p>A drop-down list box that can use the following types:</p> <ul style="list-style-type: none"> <li>▶ <code>TextFormat.TEXT</code> , or</li> <li>▶ <code>TextFormat.INTEGER</code>, or</li> <li>▶ <code>TextFormat.FLOAT</code></li> </ul> <p>It uses three columns in the layout and has the following type icon: .</p>  <p>Figure 8.16. Drop-down list box</p>
Browser	<p>A browser part that can contain HTML content with hyperlinks. Users can choose how many columns of the GUI it spans.</p>
Spacer	<p>A placeholder widget. It does not display anything, but only uses space within the GUI for layout purposes. The <code>Spacer</code> can span several columns. Thus, for example, users can define that one row displays just one label and a check box instead of two, which is the default layout.</p>
Table	<p>A table with several user-defined columns and one index column. The user-defined columns are configured by the <code>SWTColumnDescription</code> class. The table itself needs an ID for the top node memento. The table spans all four columns of the layout. For each row a submemento is created, which contains the values for each column. The path to a specific column value within the memento would be: <code>/tableId/rowNumber/columnId</code> of which</p>



Widget type	Description
	<p><code>tableId</code> is the memento-ID of the table memento.</p> <p><code>rowNumber</code> is the memento-ID for a table row. <code>rowNumber</code> is the index of the row.</p> <p><code>columnId</code> is the memento-ID for the column, defined via the <code>SWTColumnDescription</code></p>
Tree table	<p>A combination of a tree and a table. The first column displays a tree (tree column) and the other columns display the data for each tree item. The tree table spans four columns of the layout.</p> <p>The tree column is configured by the <code>SWTTreeColumnDescription</code> class and uses the type <code>TextFormat.TREE</code>.</p> <p>The <code>SWTTreeColumnDescription</code> provides the possibility to define:</p> <ul style="list-style-type: none"> <li><code>iconPaths</code> You may specify an icon for each level.</li> <li><code>editableRows</code> You may set each row level to editable or not editable. If a row level is not editable, by default only the data for the tree column is displayed, all other columns remain empty for this level. If the flag <code>SHOW_READONLY_ROW_CONTENT</code> is set, data is shown for all columns.</li> <li><code>structuralEditableRows</code> You may set a row level as not-structurally editable. This means that no child items can be added or deleted. However, the row data can be changed.</li> </ul> <p>The tree column displays the structure of the data. Each tree item consists of an optional icon and a text entry.</p> <p>All other columns are configured by the <code>SWTColumnDescription</code> class.</p> <p>The tree table itself and each column needs a memento-ID. The path to locate a specific column value within the tree table memento would be <code>/treeId/rowPath/columnId</code>, in which <code>rowPath</code> is of the form: <code>rowIndex(/rowIndex)*</code>. E.g.: <code>/treeId/0/2/columnId</code> represents the memento path for the column <code>columnId</code> of the third child of the first top-level item.</p>

Widget type	Description
	<p><b>NOTE</b> The maximum number of tree table columns is 32  You can only add 32 columns to the tree table.</p>

Here is an example of how a table and a tree table are displayed in the GUI:

Index	Integer	IntegerHex	String	Boolean	Combo	Booleanerl
0	2	0x 02	abc	<input checked="" type="checkbox"/>	ONE	<input checked="" type="checkbox"/>
1	2	0x 02	abc	<input checked="" type="checkbox"/>	ONE	<input checked="" type="checkbox"/>
2	2	0x 02	abc	<input checked="" type="checkbox"/>	ONE	<input checked="" type="checkbox"/>
3	2	0x 02	abc	<input checked="" type="checkbox"/>	ONE	<input checked="" type="checkbox"/>

Figure 8.17. Table

Description	Integer	String	Boolean	ComboBox
1.1	2	basic	<input checked="" type="checkbox"/>	ONE
1.1	2	abc	<input type="checkbox"/>	ONE
1.1	2	abc	<input checked="" type="checkbox"/>	ONE
1.1	2	abc	<input type="checkbox"/>	TWO
1.1	2	abc	<input checked="" type="checkbox"/>	ONE
1.1	2	abc	<input type="checkbox"/>	ONE
1.1	2	abc	<input type="checkbox"/>	THREE

Figure 8.18. Tree table




---

**WARNING** **At the end of the `doCreateControls` method, you must call `factory.layout(parent)`**



If you create the widgets using the `WidgetFactory`'s `create` methods, call `factory.layout(parent)` at the end of the `doCreateControls` method of the wizard page to finalize the layout of the page.

---

**TIP****Create your own custom widgets**

If the predefined widgets do not meet your requirements, you can use your own custom widgets. See [Section 8.2.6.6, “Creating a custom widget”](#) for more information on custom widgets.

---

**NOTE****You can write your GUI representation in XML format**

If you prefer to write your GUI representation in XML format instead of writing Java code, see [Section 8.2.6.4, “Using XForms as GUI description”](#) for more information on using XForms.

---

#### 8.2.3.5.6. Widget lifecycle

The Guided Configuration API also provides methods to manipulate a widget during its lifecycle. The following methods can be overridden to change the data of the widget as well as its appearance:

`AbstractBackend.doHandleUpdatedData()`

To change GUI elements that depend on the content of another GUI element, you can override this method. The method `AbstractBackend.doHandleUpdatedData()` is called at the point in time when data in the GUI is changed and after the data was written to the mementos.

`AbstractBackend.doUpdateWidgetEnablement()`

You can use this method to enable or disable a widget. The method `AbstractBackend.doUpdateWidgetEnablement()` is called asynchronously each time after data has been changed.

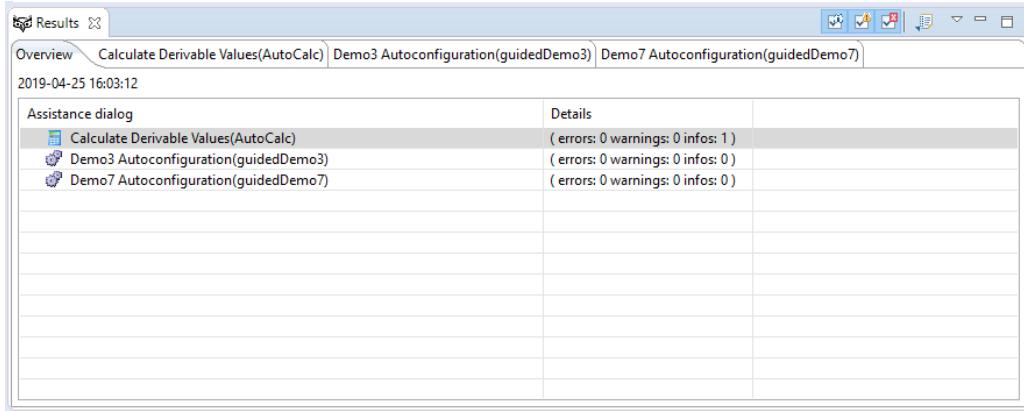
`XFormMultiPage.doAdaptWidget()`

If you use the XForms representation of the GUI, you can use this method to adapt the widget after loading the page. This could be e.g. adding an event listener or table actions.

#### 8.2.3.5.7. Results view

After running an unattended wizard or dialog, the results are displayed in the **Results** view. The results of a custom module editor or standalone editor are written to the **Error Log** view.

The **Results** view contains several tab pages. The first tab, named **Overview** always exists and displays a summary of all wizard runs.

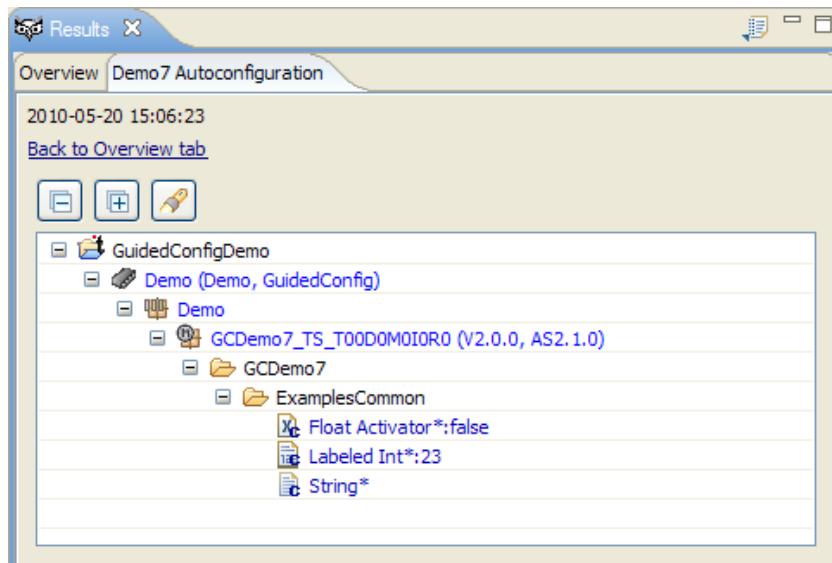
Figure 8.19. **Results** view overview

In addition to the **Overview** tab, there exists one tab for each wizard run. The tab displays either your own custom result page or if you have not specified an own result widget, the `DefaultResultWidget`. If you want to display your own result page, implement a class inherited from `AbstractResultWidget` and register it in the `dreisoft.tresos.guidedconfig.api.plugin.guidedconfigwizard` extension point.

The `DefaultResultWidget` displays the list of `APIOperationStatus` that are returned by the `doRun()` method.

For example, your own *result widget* can display which parameters of the DataModel configuration have been changed by the *push operation* (see [Section 8.2.6.11, “Registering a custom result widget”](#)).

The Guided Configuration API already provides a possibility to display changed DataModel parameters. To use this feature, register the class `ChangedDCxtsResultWidget` as result GUI.

Figure 8.20. `ChangedDCxtsResultWidget`

### 8.2.3.6. The push service pattern

To encapsulate the data exchange between the data of the guided configuration wizard (*memento*) and the data model configuration, the Guided Configuration API provides a pattern called *push service*.

The *push service* consists of a *push event* and a corresponding *push operation* that listens to the event.

A push operation is triggered from within the `Backend` class by sending a push event. The push operation is responsible for the data transfer from the wizard to the data model and back:

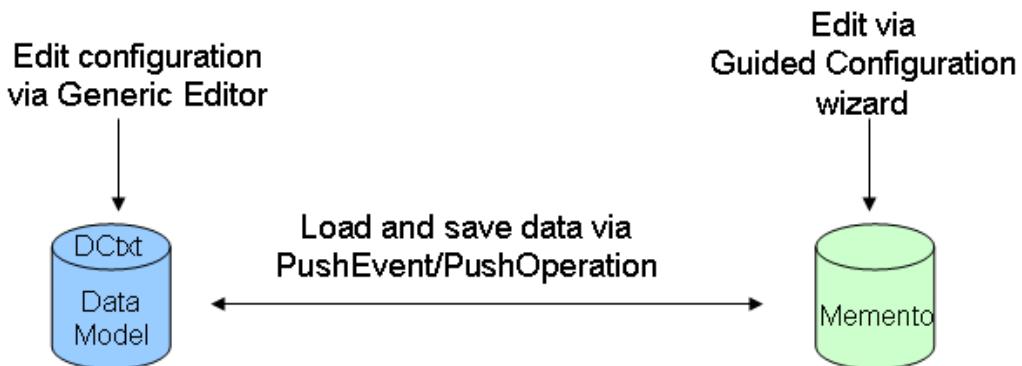


Figure 8.21. *Push service* data flow

A *push operation* can be reused for more than one wizard, by passing a special kind of event.

A *push event* is bidirectional and can return the result from the *push operation* to the `Backend` class.

You can register a *push operation* in two ways:

- ▶ via the extension point `dreisoft.tresos.guidedconfig.api.plugin.pushservice`.

To register *push operation* via this extension point, register a class that implements `AbstractConfigurablePushOperation`. You may also restrict the runnability of the *push operation* by using the `event` tag. If you use the `event` tag, you can either restrict the *push operation* to a specific event class or to a specific *selection context*.

- ▶ via the `PushService` registry.

To register *push operation* via this registry, register an operation that must implement `AbstractPushOperation`. Example:

```
PushService.getInstance().addPushOperation(AbstractPushOperation operation)
```

The extended class can implement your own `doCanHandle( AbstractPushEvent event )` method to query whether the operation can handle the given event.

**NOTE****You do not have to use *push operation***

A *push operation* is optional. The backend can also write to the EB tresos Studio data model directly via the Public API DCtxt class.

### 8.2.3.7. Locking mechanism

If a wizard for a specific project is displayed, all functions that access the datamodel of *this* project are locked.

Some of these functions can try to unlock the project by closing the wizard, but this is only possible if the wizard has no validation errors. EB tresos Studio then opens a confirmation dialog to ask the users whether they want to close the wizard in order to perform the chosen function:

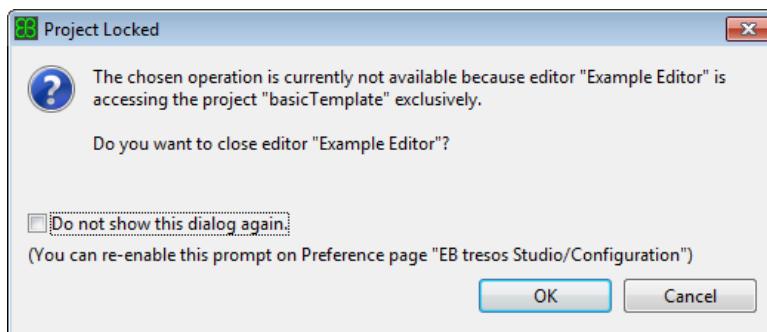


Figure 8.22. Confirmation dialog to automatically close the open wizard

If you check the **Do not show this dialog again** check box and click **OK**, this confirmation dialog does not appear again. Open wizards are always closed automatically until you reset the preference.

**TIP****Turning confirmation dialogs on and off**

In the EB tresos Studio preferences you may change whether this confirmation dialog shall appear or not. To turn off this confirmation dialog, clear the check box **Show confirmation dialog to close a locking part** on the EB tresos Studio preferences page. For further information on the EB tresos Studio preferences, see the EB tresos Studio user's guide, chapter Setting general preferences.

If the wizard has validation errors, it is not possible to close the wizard automatically:

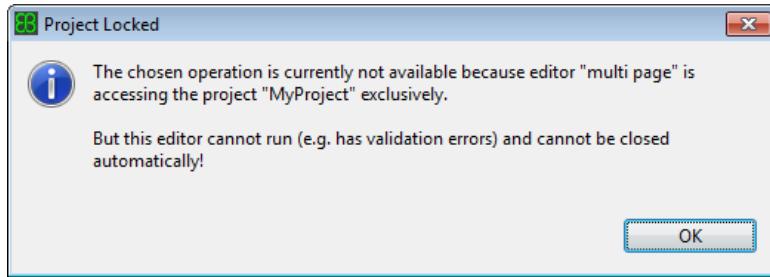


Figure 8.23. Dialog informing users that the wizard cannot be closed

In such a case, the chosen function is not performed.

The project can be unlocked by automatically closing the locking part, when the user performs the following functions:

- ▶ Opening the **New Importers/Exporters** dialog
- ▶ Opening the **Module Configuration** dialog.
- ▶ Trying to open another wizard or trying to run an unattended wizard for the same project.
- ▶ Trying to save the project while the wizard is displayed.

The following functions cannot be performed as they do not unlock the project:

- ▶ The user can still open and browse a configuration editor, but cannot edit them.
- ▶ The user cannot generate or verify code.

## 8.2.4. API

You find the exact syntax of the methods in the Java documentation. For further information see [Section 3.3, “Public Java API documentation”](#).

**TIP****Do not use deprecated methods**

Do not use methods in internal packages or methods marked as *deprecated*.

**TIP****do- methods are overwritten**

Methods with the prefix *do* are meant to be overwritten.

### 8.2.4.1. Extensibility

Each guided configuration wizard is built on the model-view-controller pattern. To develop a wizard, you must extend some of the classes necessary for these parts. In the following chapter you will learn which classes must be extended and which classes can be extended to use additional features of the guided configuration framework.

#### 8.2.4.1.1. Model

For the *model* part of the wizard, you do not have to extend any classes, but only use the classes `Memento` and `MementoOperationHandler` as mentioned in chapter [Section 8.2.3.4, “Data representation”](#).

To manipulate the construction of the memento tree, implement your own custom memento factory that subclasses `AbstractMementoFactory` and implements the method `getMemento( Memento parent, String name)`. For further information on the `MementoFactory` class, see [Section 8.2.3.4.3, “Memento-Factory”](#).

#### 8.2.4.1.2. View

It depends on the type of page description what you need to extend to create your own *View* part.

Java page description (package: `dreisoft.tresos.guidedconfig.api.gui.page`)

##### multi-page

If you want to create a multi-page view, implement an `AbstractMultiPage` and register it at the `dreisoft.tresos.guidedconfig.api.plugin.guidedconfigwizard` extension point. In addition, add one instance of `AbstractPage` for each single-page view.

##### single-page

To create a single-page view, implement an `AbstractPage` class and register it as GUI for the wizard.



### XForm page description (package: dreisoft.tresos.guidedconfig.api.gui.xform)

To use an XForm page description, register either `XFormMultiPage`(for a multi-page wizard) or `XFormPage` (for a single-page wizard).

Only subclass these classes if the GUI of the wizard needs to be adapted directly after opening the wizard. This is the case if, for example, some table actions must be added. In this case override the method `doAdaptWidget()`.

#### 8.2.4.1.3. Controller

To create the *Controller* part, you have to implement one of the following classes:

##### AbstractBackend

`package: dreisoft.tresos.guidedconfig.api.backend`

This class is necessary to manage the lifecycle of the wizard. Register the `backend` class in the `plugin.xml`.

##### AbstractDCtxtBackend \*deprecated\*

`package: dreisoft.tresos.guidedconfig.api.backend`

If the backend changes data model parameters, e.g. using a push service, implement the `AbstractDCtxtBackend` class instead of the `AbstractBackend` class. The `AbstractDCtxtBackend` class provides methods to mark the parameters as changed. The changed parameters are then shown to the user. The guided configuration framework provides the `ChangedDCtxtsResultWidget` to show the changed DCtxt nodes.

This class is marked as deprecated, as it is not necessary anymore to implement this class to be able to show changed DCxts in the **Results** view. All executed DCtxt operations are remembered in the backend by default. To be able to show these DCtxt changes in the **Results** view, just register the `ChangedDCtxtsResultWidget` as `resultGui` parameter in the `plugin.xml`. It is even not necessary anymore to collect the changed DCtxt nodes by yourself.

#### 8.2.4.1.4. Additional extensible classes

The following classes provide additional features for the guided configuration. These classes are not mandatory to use.

##### 8.2.4.1.4.1. Context

If you want to create your own context class or if you want to listen to context changes, extend the following classes.



Class	Description/Use case
IContext	The <code>IContext</code> class is the public interface to set and query a selection context. The selection context stores information about the current environment in a list of tag/value pairs. An abstract implementation of this class exists: <code>AbstractContext</code> . You find a description of the <code>AbstractContext</code> in the following table row.
AbstractContext	EB tresos Studio provides several context classes that make information available about the currently selected project and node. The information is used to restrict the visibility of the guided configuration wizard and to provide the wizard with information. You can implement your own context class and register it via <code>ContextManager</code> . It is not the recommended way to restrict the visibility of a wizard implementing an own context class. For more information about restricting the visibility of wizards, see <a href="#">Section 8.2.6.3, “Restricting the visibility of a wizard”</a>
IContextChangedListener	You can implement an <code>IContextChangedListener</code> and register it at the <code>ContextManager</code> class. Like this you are notified if the current context changes.

#### 8.2.4.1.4.2. Widget

This chapter is important if you want to:

- ▶ use custom widgets
- ▶ listen to selection events
- ▶ add actions for tables, tree tables, and buttons.

Classes to extend when changing widgets:

Class	Description/use case
AbstractWidgetController	<p><code>package: dreisoft.tresos.guidedconfig.api.gui.form</code></p> <p>This class is the abstract implementation of the controller class for a guided configuration widget, which provides methods to communicate with the guided configuration engine. <code>AbstractWidgetController</code> serves as connector between the backend and the GUI. If you create your own custom widget, extend the <code>AbstractWidgetController</code> class. If you also connect to a memento, it is recommended to subclass <code>AbstractMementoWidgetController</code>.</p>
AbstractMementoWidgetController	<code>package: dreisoft.tresos.guidedconfig.api.gui.form</code>



Class	Description/use case
	<p>The <code>AbstractMementoWidgetController</code> is a subclass of <code>AbstractWidgetController</code>. For better access of the underlying data representation, subclass this class instead of the <code>AbstractWidgetController</code>.</p>
WidgetSelectionEvent	<p><code>package: dreisoft.tresos.guidedconfig.api.gui.eventing</code></p> <p>The guided configuration framework provides the possibility to create your own custom widgets. If you want to provide a selection event for the custom widget, subclass the <code>WidgetSelectionEvent</code> class. If the custom widget is selected, this selection event must be sent.</p>
ISelectionEventListener	<p><code>package: dreisoft.tresos.guidedconfig.api.gui.form</code></p> <p>To be able to react on <code>WidgetSelectionEvents</code>, implement the <code>ISelectionEventListener</code> interface. You will then be notified if a selection event occurs.</p>
AbstractTableAction	<p><code>package: dreisoft.tresos.guidedconfig.api.gui.form</code></p> <p>The guided configuration framework allows you to add your own actions to a table. If you want to add a new table action to the <code>SWTTable</code> on your page, implement this interface.</p>
AbstractTreeTableAction	<p><code>package: dreisoft.tresos.guidedconfig.api.gui.form</code></p> <p>The guided configuration framework allows you to add your own actions to a tree table. If you want to add a new tree table action to the <code>SWTTreeTable</code> on your page, implement this interface.</p>
ISWTWidget	<p><code>package: dreisoft.tresos.guidedconfig.api.gui.form</code></p> <p>If you want to implement your own custom widget, implement the <code>ISWTWidget</code> interface in your widget class.</p>
AbstractResultWidget	<p><code>package: dreisoft.tresos.guidedconfig.api.gui.page</code></p> <p>With the guided configuration framework you can add your own custom result widget for unattended wizards and dialog wizards to the <b>Results</b> view. To do so, subclass the <code>AbstractResultWidget</code> class and register your custom result widget in the <code>dreisoft.tresos.guidedconfig.api.plugin.guidedconfigwizard</code> extension point.</p>
WidgetFactory	<p><code>package: dreisoft.tresos.guidedconfig.api.gui</code></p> <p>If you want to use custom widgets via XForms, override the <code>WidgetFactory</code> class. This is necessary to parse the XForms page and to create the widget.</p>



Class	Description/use case
	<p>In your <code>WidgetFactory</code>, create a <code>DefaultWidgetDescription</code> for your custom widget and add it to the list of common widgets calling <code>addWidgetDescription()</code>. So calling <code>doGetWidgetDescriptions()</code> also returns the new custom widget.</p> <p>In order to use the custom widget factory, implement <code>doGetWidgetFactory</code> in your <code>AbstractBackend</code> implementation so it returns the custom widget factory</p>

#### 8.2.4.1.4.3. Push service

Use the push service for data exchange between the wizard data representation and the data model. You can also access the data model without using push service. If you want to use push service, implement the following classes:

Class	Description/use case
<code>AbstractPushEvent</code>	<p><code>package: dreisoft.tresos.guidedconfig.api.pushservice</code></p> <p>To use the push service, implement <code>AbstractPushEvent</code>. One event can trigger several operations to run. The event must contain all necessary information to invoke the operation.</p>
<code>AbstractConfigurablePushOperation</code>	<p><code>package: dreisoft.tresos.guidedconfig.api.pushservice</code></p> <p>The common and recommended way is to register a <i>push operations</i> in the <code>plugin.xml</code>. There, you can restrict the execution of the operation with the <code>event</code> tag. To use this feature, you need a subclass of the <code>AbstractConfigurablePushOperation</code>.</p>
<code>AbstractPushOperation</code>	<p><code>package: dreisoft.tresos.guidedconfig.api.pushservice</code></p> <p>If the execution of a <i>push operation</i> does not have to be restricted, you can register it via the <code>PushService</code> registry calling <code>addPushOperation()</code>. In this case, create a subclass of <code>AbstractPushOperation</code>.</p>
<code>AbstractHandleIdPushOperation</code>	<p><code>package: dreisoft.tresos.guidedconfig.api.pushservice</code></p> <p>If you want to implement a push operation for creating handle IDs, use the <code>AbstractHandleIdPushOperation</code> as a base implementation.</p>

#### 8.2.4.1.4.4. Triggers

The common way to use triggers in the **Sidebar** view is to register them in the `plugin.xml` file. Alternatively, you may also register `AbstractTrigger` via `TriggerRegistry`.



Classes to extend when registering triggers via `TriggerRegistry`:

Class	Description/use case
AbstractTrigger	<p>package: <code>dreisoft.tresos.guidedconfig.api.trigger</code></p> <p>Subclass and register <code>AbstractTrigger</code> at the <code>TriggerRegistry</code> by calling <code>addTrigger()</code>. It is not recommended to register a trigger via <code>TriggerRegistry</code>. Instead rather add a trigger in the <code>plugin.xml</code>, because there you can restrict the visibility of the trigger querying the current context.</p>

## 8.2.5. Demos

### 8.2.5.1. Setting up the demo plug-in

This chapter describes several demos and their specifics. To be able to use the demos, follow these instructions.

The demo plug-ins are available in the folder `<tresos-install-loc>/demos/Studio/`. For each demo one plug-in is available. Install the plug-in of the demo you are interested in, according to the instruction in [Section 4.9, “How to install a Public API demo plugin”](#).

Then return to this chapter and follow the instructions in this section.

After you installed the demo plug-in, create a project with

- ▶ target/derivative `Demo/GuidedConfig`
- ▶ release version 2.1
- ▶ and a module for the demo you are interested in. You find the corresponding module-ID in the respective demo chapter.

If the project is not loaded, load the configuration. If the **Sidebar** view is not visible, follow the instructions in [Section 8.2.5.1.1, “Making the Sidebar view visible”](#).

The data of the wizards is stored persistently in XML files. You find these files in your project in the subfolder `.-prefs` in files that have the extension `.mem`. As default `.* resources` are not shown in the **Project Explorer**. To show the files, follow the instructions in [Section 8.2.5.1.2, “Customizing the view”](#)

You are done setting up the demo plug-in.

Each demo displays different features of the Guided Configuration API. Find an overview of which demo uses which feature in the following list:

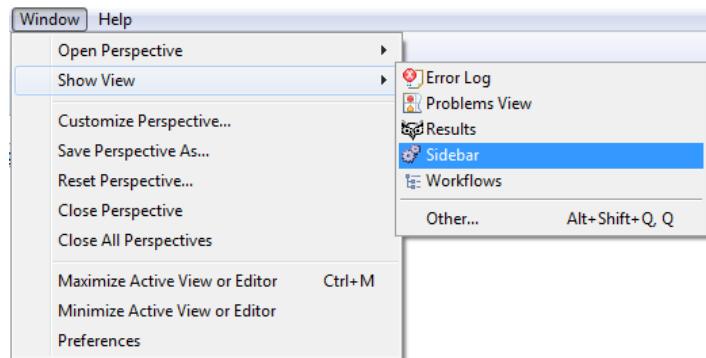


	<b>Visibility</b>	<b>Multipage</b>	<b>Custom widget</b>	<b>Widgets overview</b>	<b>Xforms</b>	<b>Validation</b>	<b>Update widget-enabling</b>	<b>Selection events</b>	<b>Navigation</b>	<b>MementoFactory</b>	<b>PushService</b>	<b>Custom ResultWidget</b>	<b>ChangeIDContexts-ResultWidget</b>	<b>Place widgets on external view</b>
<b>Standalone editor (demo1)</b>	x	x			x	x			x	x				
<b>Standalone editor (demo2)</b>	x		x				x				x			
<b>Unattended wizard dialog (demo3)</b>	x										x	x		
<b>Custom module editor (demo4)</b>	-			x		x		x						
<b>Dialog (demo5)</b>	x	x				x								
<b>Dialog (demo6)</b>	x													
<b>Unattended wizard dialog (demo7)</b>	x										x		x	
<b>Standalone editor (demo8)</b>					x	x				x	x			
<b>Standalone editor (demo9)</b>														x

Figure 8.24. Overview of the features of the demos

### 8.2.5.1.1. Making the Sidebar view visible

If the **Sidebar** view is not visible in EB tresos Studio, open the **Window** menu, point to **Show View** and select **Sidebar**.

Figure 8.25. Making the **Sidebar** view visible

The **Sidebar** view opens up in the upper right corner of the EB tresos Studio main window.

### 8.2.5.1.2. Customizing the view

Customize your **Project Explorer** view so that .\* resources are displayed. This means you can see the persistent data representation as XML files with extension .mem in the .prefs folder of the corresponding project.

To display the .\* resources:

- ▶ Click the **View** button in the **Project Explorer**.



A menu opens up.

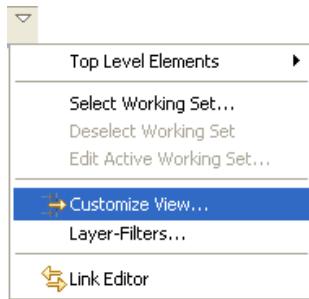


Figure 8.26. Customizing your display options

- ▶ In the drop-down menu, select **Customize View...**.

The **Available Customizations** dialog opens up.

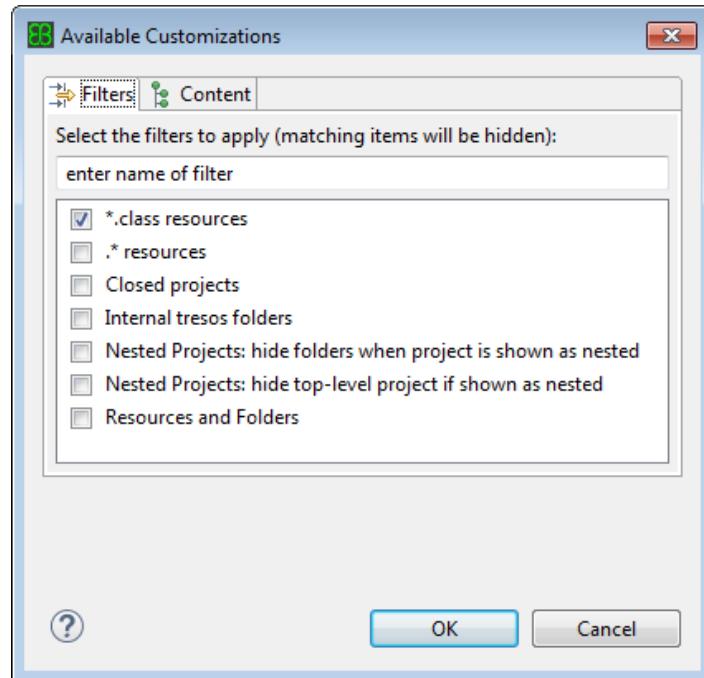


Figure 8.27. Selecting available customizations

- ▶ Clear the **\* resources** check box.
- ▶ Click **OK**.

You have customized the **Project Explorer** to display the persistent data representations as XML files.



### 8.2.5.2. Demo1

The wizard for demo1 is available in the **Sidebar** view and opens the standalone editor **XFormsWizard** in the editor area of EB tresos Studio.

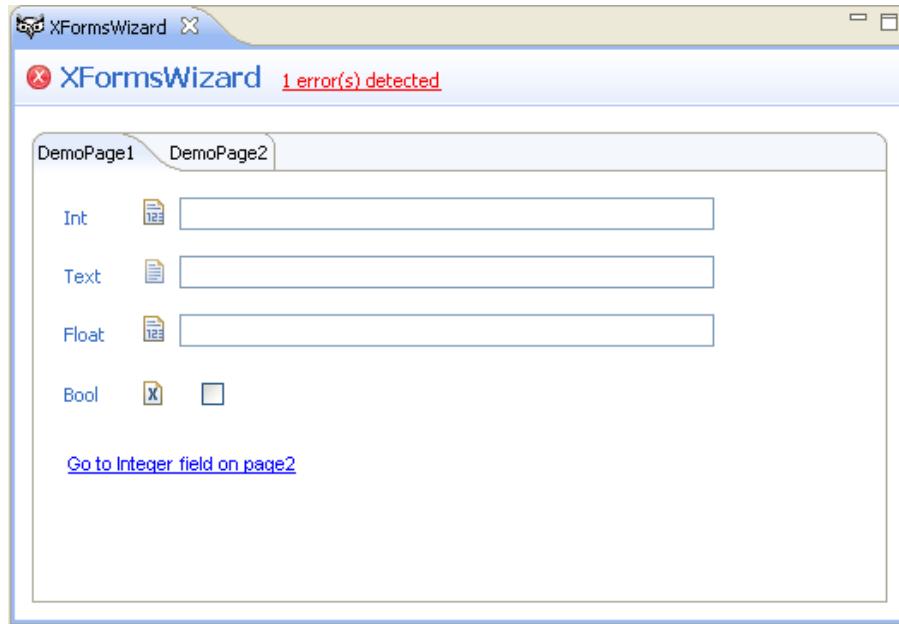


Figure 8.28. Demo1: XForms multi-page

In this example you will learn:

- ▶ how to restrict the visibility of a wizard, so that it is only displayed in the **Sidebar** view, and
- ▶ how multi-pages are displayed in an editor and how to navigate through the wizard, and
- ▶ how to use an XForms page description to describe the GUI representation in an XML format, and
- ▶ how to validate the data and how the validation results are displayed in the GUI, and
- ▶ how to use your own navigation links, and
- ▶ how to control which data to store persistently.

#### 8.2.5.2.1. Restricting the visibility

In this section, you find information on how to restrict the visibility of a wizard so that it is only displayed in the **Sidebar** view if some preconditions are fulfilled. This is helpful e.g. if you want to make the editor only visible for projects of a particular target/derivative.

- ▶ To make demo 1 available in the **Sidebar** view, select a loaded project. This loaded project must contain the target `Demo` and a `GCDemo1` module.



- ▶ To restrict the visibility of the wizard so that it is not visible in the **Sidebar** view, disable or remove the module `GCDemo1` via the module configuration dialog. The trigger with which the user can open the wizard then disappears from the **Sidebar** view.

For more information, see [Section 8.2.6.3, “Restricting the visibility of a wizard”](#).

#### 8.2.5.2.2. Using multi-page support

The wizard has two pages, which are available as tabs in the editor area of EB tresos Studio. If an error is found, it is displayed on the top part of the tab no matter which tab is open in your editor area. You can switch between the pages by clicking the tabs in the editor area of EB tresos Studio.

#### 8.2.5.2.3. Using XForm GUI description

The XForm page description is not written in Java, but is an XML file, that is part of the demo plug-in. You find this XML file in the demo plug-in at `pages/Pages.xml`. For information on registering an XForm page description, see [Section 8.2.6.4, “Using XForms as GUI description”](#).

#### 8.2.5.2.4. Validating

When you open the wizard, an error notification is displayed on top of the page. This error summary is a hyper link and via which you may navigate to the erroneous parameters. In this case, when you click the error message **(GUIDEDDEMO\_1) An error occurred: Integer field on page 2 must not be empty**, the tab page **DemoPage2** is activated and the integer field becomes active.



Figure 8.29. Demo1: Validation summary as a hyperlink on top of the page

#### 8.2.5.2.5. Navigating

With the browser widget, you may add your own hyper links to navigate to other widgets. The following example code shows you how to do this:



<a href="wizard://guidedDemo1/Page2IntWidget">Go to Integer field on page2</a>

The `WidgetFactory` class navigates to the correct page and sets the focus to the specified widget.

#### 8.2.5.2.6. Creating your own `MementoFactory` class

This demo uses its own `MementoFactory` class instead of the `DefaultMementoFactory` class. Via the `MementoFactory` class, you can define which parameters you want to be persistent and which not. To apply your changes, close and reopen the wizard after you have changed the values of the parameters.

You can store only those parameters that are not read by the `DataModel` when the wizard is opened the next time.

Now only the values of the following parameters are stored persistently and are available after reopening the wizard:

- ▶ Text (DemoPage1)
- ▶ Bool (DemoPage1)
- ▶ Integer (DemoPage2)

For information on how to create your own `MementoFactory`, see [Section 8.2.6.9, “Creating a MementoFactory”](#).

#### 8.2.5.3. Demo2

The wizard for demo2 is available in the **Sidebar** view. Opening the wizard **Demo2 Custom Widget** opens a standalone editor in the editor area of EB tresos Studio. This demo shows how to create and use a custom widget, which can be used if the predefined widgets of the Guided Configuration API do not fit your needs.

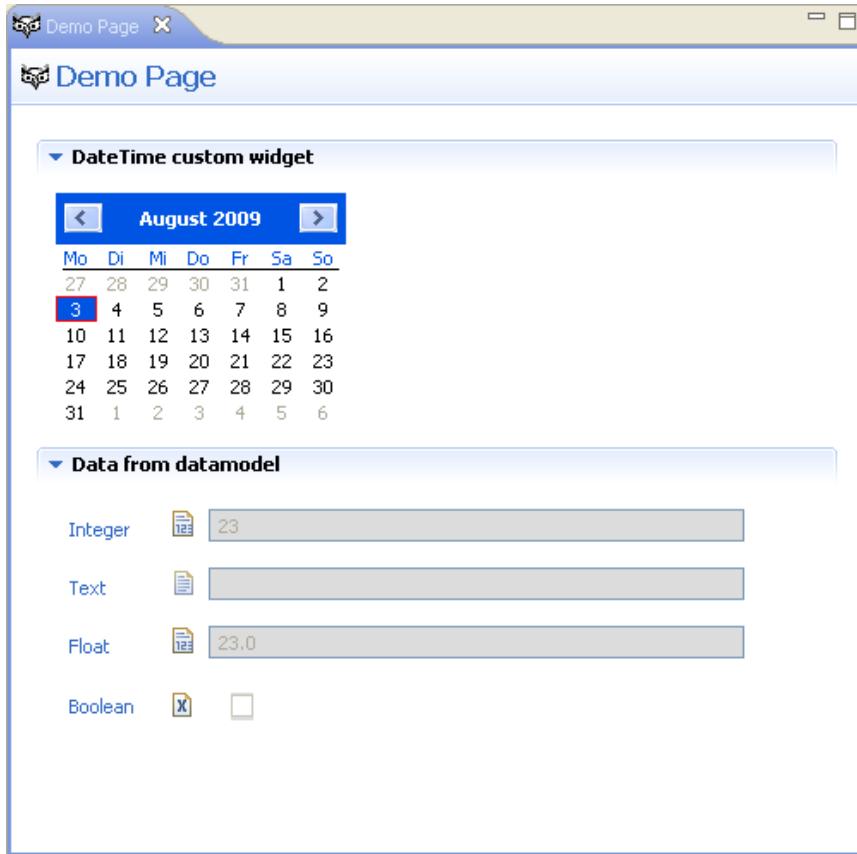


Figure 8.30. Demo2: Standalone editor with custom widget

In this demo you learn:

- ▶ how to create and register your own custom widget, and
- ▶ how to use push services to load data from the data model, and
- ▶ how to update the widget enablement during the lifecycle of the wizard.

#### 8.2.5.3.1. Restricting the visibility

In this section, you find information on how to restrict the visibility of a wizard so that it is only displayed in the **Sidebar** view.

- ▶ To make demo 2 available in the **Sidebar** view, select a loaded project. This loaded project must contain a module with the module ID `GCDemo2_TS_T00D0M0I0R0`.
- ▶ To restrict the visibility of the wizard so that it is not displayed in the **Sidebar** view, disable or remove the module `GCDemo1` via the module configuration dialog. The trigger with which the user can open the wizard for Demo2 then disappears from the **Sidebar** view.



#### 8.2.5.3.2. Creating a custom widget

If the wizard is opened for the first time, the custom **datetime** widget shows the current day. You can change the value, which is then stored persistently. Thus after reopening the wizard, the last date is displayed again.

The values for day/month/year are attributes of a child memento of the root memento of the wizard. But changing the values for day/month/year is done in an atomic transaction. Since the wizard is a standalone editor, users may use the **Undo/Redo** buttons to undo or redo a selection. Then all data is changed with one click.

To add your own custom widget to your page, see [Section 8.2.6.6, “Creating a custom widget”](#).

#### 8.2.5.3.3. Using the push service

In the section **Data from datamodel** of the wizard, you find some values from the data model configuration. These values are loaded via a push operation using the DCtxt API from the data model and stored in the memento tree. As the demo does not use a push operation to store values back into the data model configuration, the parameters of this group are *read only*.

If you want to use the push service, see [Section 8.2.6.10, “Using the push service”](#).

#### 8.2.5.3.4. Updating widget enablement

The parameters of the section **Data from datamodel** in the wizard editor are disabled, as they are not supposed to be changed. They are displayed only to show the values of the data model configuration. Changing the enablement of the widgets can be implemented in the `doUpdateWidgetEnablement` method of your **Backend class**.

For further information about the lifecycle of the widgets, have a look at chapter [Section 8.2.3.5.6, “Widget lifecycle”](#).

#### 8.2.5.4. Demo3

Demo3 provides an unattended wizard, which is available via the **Unattended Wizards** configuration dialog. To open an unattended wizard, click the **View** button next to the **Open unattended wizard configuration dialog** button in the tool bar and select **Unattended wizard configuration....**

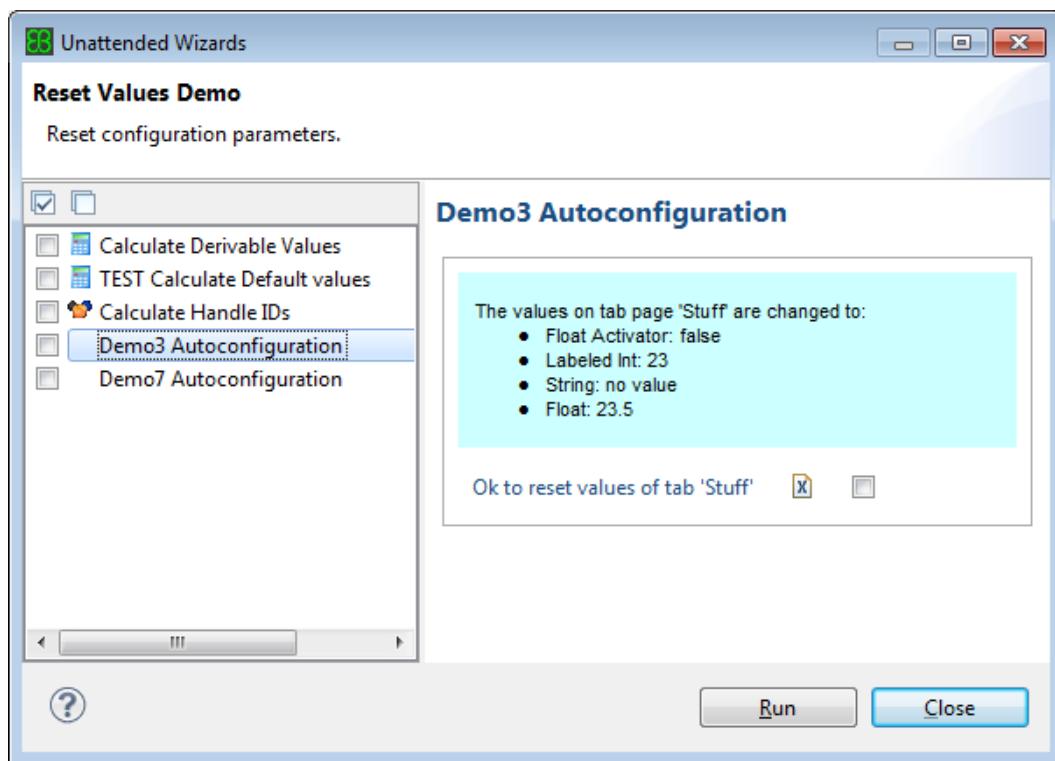


Figure 8.31. Demo3: **Unattended Wizards** configuration dialog

In the left part of the **Unattended Wizards** configuration dialog, you see the list of available wizards for the currently selected project. Each selected wizard is considered as favorites. Using the **Check all for manual run** button  and **Uncheck all for manual run** button , users can select or clear all wizards with one click.

The right part of the **Unattended Wizards** configuration dialog is the configuration area for one wizard. This configuration area is updated when you select another unattended wizard in the left part of the **Unattended Wizards** configuration dialog. If the wizard contains multiple pages, the pages are displayed on different tabs inside the right part of the **Unattended Wizards** configuration dialog.

In this demo you learn:

- ▶ how to use push services to change data of the data model, and
- ▶ how to create your own result widget to display the result of the push operation.

Running this demo helps you to understand what you can do with push operations. Use this demo to reset the values of the data model configuration to predefined values. You find the parameters in the configuration tab **Stuff**. The parameters are reset to the following values:

- ▶ **Float Activator** to **false**
- ▶ **Labeled Int** to **23**
- ▶ **String** is made empty



- ▶ **Float to 23.5**

To work with this demo:

1. Open the Generic Configuration Editor for the module `GCDemo3_TS_T00D0M0I0R0`.
2. In the tab **Stuff**, change the values of the parameters mentioned above and save the configuration.

Remember your values.

3. Open the **Unattended Wizards** configuration dialog.
4. In the list on the left side, select **Demo3 Autoconfiguration**.
5. Select the **OK to reset values of tab 'Stuff'** check box. For further information about which values are reset, see [Figure 8.31, “Demo3: Unattended Wizards configuration dialog”](#).

There are four alternative ways to run the wizard you have recently configured:

- ▶ Select the wizard directly in the **Unattended Wizards** configuration dialog and click **Run**.
- ▶ Or select the check box of the Demo3 wizard on the left side of the **Unattended Wizards** configuration dialog. If there are more than one wizard, make sure only the required wizard is selected. Then click **Close** and click the **Open unattended wizard configuration dialog** button in the tool bar. All selected wizards of the **Unattended Wizards** configuration dialog are executed.
- ▶ Or click the **View** button next to the **Open unattended wizard configuration dialog** button in the tool bar and select the **Demo3 Autoconfiguration** wizard from the context menu that opens up.
- ▶ Or select **Unattended Wizards** from the **Project** menu, and select the **Demo3 Autoconfiguration** wizard.

After running the unattended wizard, the **Results** view appears with a list of which configuration data has been changed. To see which values have been changed, open the Generic Configuration Editor again.

#### 8.2.5.4.1. Using the `PushService` class

If you run the unattended wizard, a *push event* is sent. All registered *push operations* then check whether they are listening to this event. All *push operations* for this event are run. In the case of Demo3, the *push operation* sets the four values and returns the paths that lead to these configuration parameters to the *push event*.

For more information about the *push service*, see [Section 8.2.6.10, “Using the push service”](#).

#### 8.2.5.4.2. Registering a custom *result widget*

The custom result widget is displayed on the second tab of the **Results** view. It displays a table with two columns. The first column shows the path for the changed configuration parameter and the second column shows the new value of this parameter.

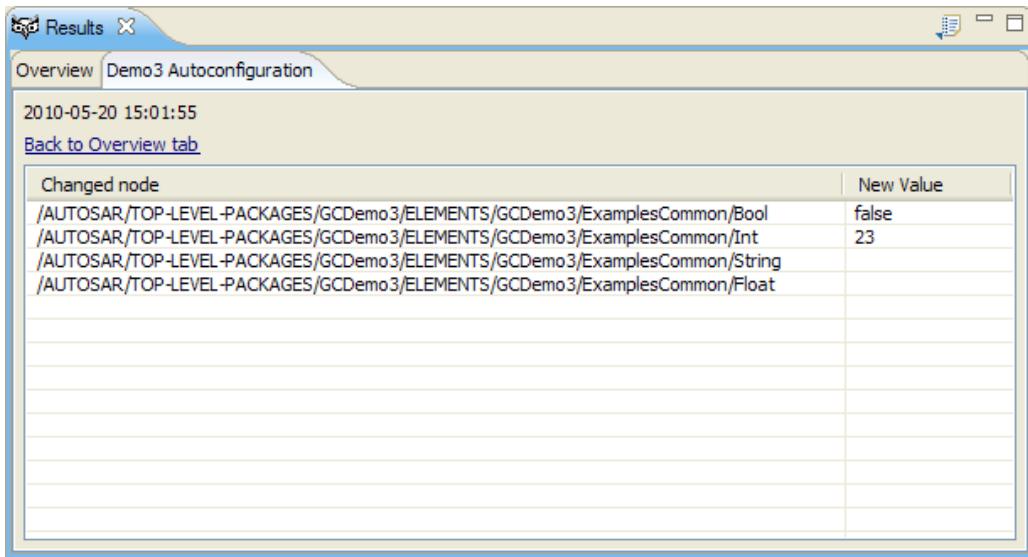


Figure 8.32. Demo3: Custom *result widget*

Find more information on how to register your own *result widget* in [Section 8.2.6.11, “Registering a custom result widget”](#)

#### 8.2.5.5. Demo4

Demo4 shows a custom module editor which is only available via the **Project Explorer** view.

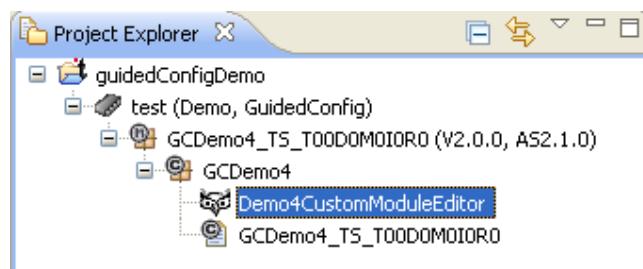


Figure 8.33. Demo4: A custom module editor in the **Project Explorer** view

The demo contains examples of :

- ▶ all predefined widgets, and
- ▶ how to validate parameters (text boxes, tables and tree tables), and
- ▶ selection events for buttons, and
- ▶ filtering the rows of the table, and
- ▶ changing the sorting column of the table.



#### 8.2.5.5.1. Restricting the visibility

For custom module editors it is not possible to restrict the visibility via extension point. If the corresponding module is added to the project, this docking point is always available.

#### 8.2.5.5.2. Displaying predefined widgets

This demo shows an example of all predefined widgets that can be created via the `WidgetFactory` class. A list of which widgets are available via `WidgetFactory` is available in [Section 8.2.3.5.5, “Predefined widgets”](#).

#### 8.2.5.5.3. Validating

If the following parameters are empty, they display an error:

- ▶ Integer
- ▶ Float

If the **Boolean** check box is cleared, the **Boolean** column is validated and a warning is displayed for the table and tree table.

TreeTable				
Description	Integer	String	Boolean	ComboBox
◆ 1.1	2	abc	⚠	ONE
■ ◆ 1.1	2	abc	⚠	ONE
■ ■ ◆ 1.1	2	abc	⚠	ONE
< > 1.1	2	abc	⚠	ONE
< > 1.1	2	abc	⚠	ONE

Figure 8.34. Demo4: Tree table displaying errors

You find an example for validating the data in [Section 8.2.6.7, “Validating the data”](#).



#### 8.2.5.5.4. Using selection events

I.	Integer	Text	Boolean	Combo	Float
0	47	hello	<input type="checkbox"/>	ONE	7.89
1	15	abc	<input checked="" type="checkbox"/>	TWO	1.1
2	77	abc	<input type="checkbox"/>	ONE	3.27
3	2	hello	<input type="checkbox"/>	THREE	1.1

**FilterTable**

**ChangeSortColumn**

Figure 8.35. Demo4: Table with push and toggle button

Demo4 provides an example for selection events by using buttons. There exist two kinds of buttons:

- ▶ the toggle button called **Filter Table**, and
- ▶ the push button called **ChangeSortColumn**.

Per default the table is sorted by the index column (short name: *I.*). The current sort column is always marked with a little arrow icon in the column title.

I.	Integer	Text	Boolean	Combo	Float
0	47	hello	<input type="checkbox"/>	ONE	7.89
1	15	abc	<input checked="" type="checkbox"/>	TWO	1.1
2	77	abc	<input type="checkbox"/>	ONE	3.27
3	2	hello	<input type="checkbox"/>	THREE	1.1

Figure 8.36. Demo4: Table before using the buttons

The button **ChangeSortColumn** changes the sorted column so that the table is sorted by the **Text** column.



Index	Integer	Text	Boolean	Combo	Float
1	15	abc	<input checked="" type="checkbox"/>	TWO	1.1
2	77	abc	<input type="checkbox"/>	ONE	3.27
0	47	hello	<input type="checkbox"/>	ONE	7.89
3	2	hello	<input type="checkbox"/>	THREE	1.1

Figure 8.37. Demo4: Table with changed sort column

The toggle button **Filter Table** filters all rows with the value `hello` in the column **Text** of the table. To reset the row filter, click the toggle button again. All rows are then displayed again.

Index	Integer	Text	Boolean	Combo	Float
1	15	abc	<input checked="" type="checkbox"/>	TWO	1.1
2	77	abc	<input type="checkbox"/>	ONE	3.27

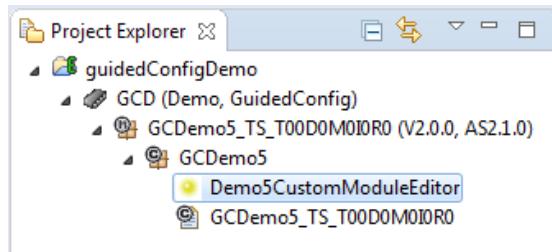
Figure 8.38. Demo4: Table with row filter

If you want to know how to implement a selection event listener, see [Section 8.2.6.8, “Using selection events”](#).

### 8.2.5.6. Demo5

The wizard of demo5 is docked into multiple docking points of the GUI:

- ▶ As custom module editor in the **Project Explorer** view, which opens a multi-page editor in the editor area of EB tresos Studio.

Figure 8.39. Demo5: Custom module editor in the **Project Explorer**



As you can see a custom module editor can define a specific icon in the **Project Explorer** view as does the Generic Configuration Editor. The default icon is the owl.

- ▶ As dialog in the **Sidebar** view, where the multiple pages are navigateable via **Back/Next** buttons.

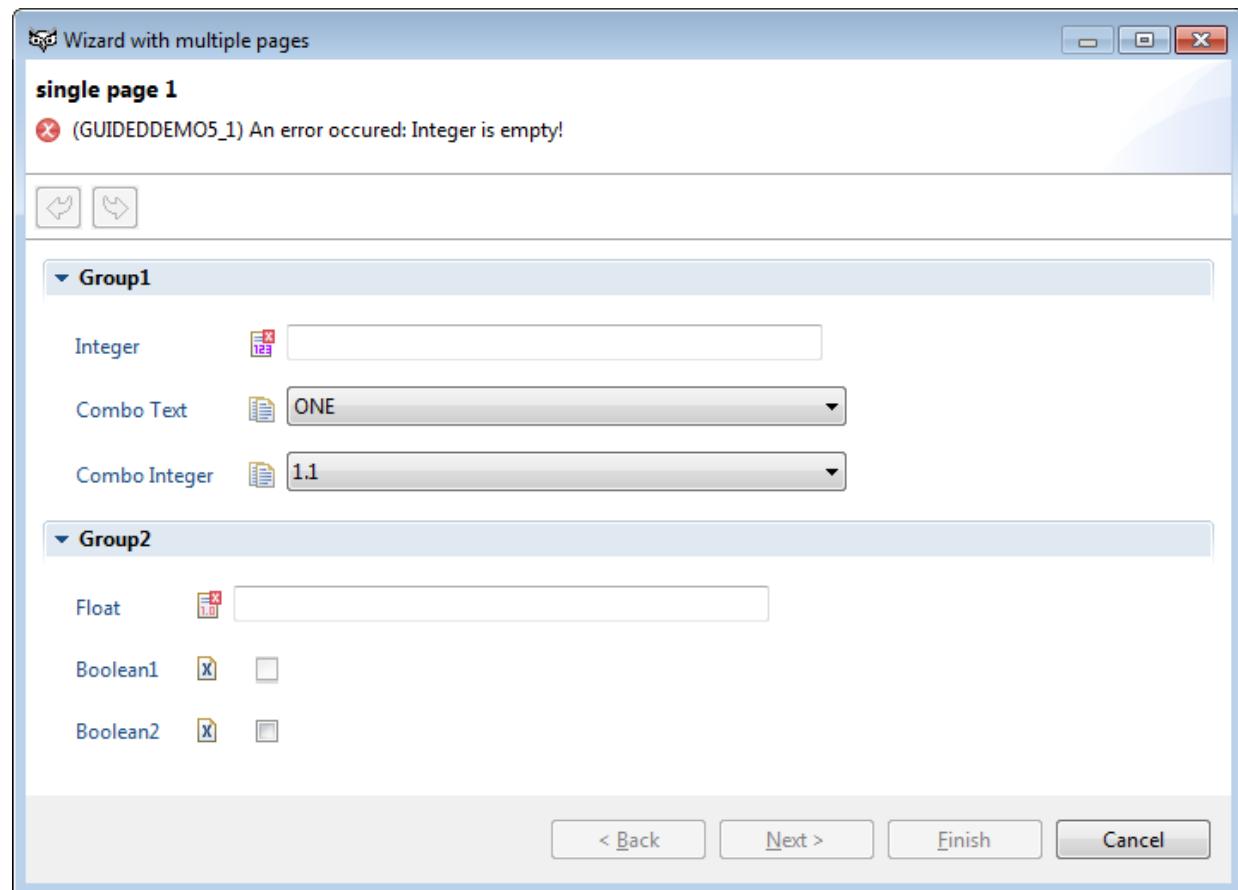


Figure 8.40. Demo5: Dialog available via **Sidebar** view

In this demo you learn the differences between editors and dialogs:

- ▶ The navigation in multi-page wizards differs in editors and dialogs.

In editors you navigate via the tabs in the editor area.

Dialogs provide **Back/Next** buttons to navigate between several pages.

- ▶ In editors all validation results are displayed as hyperlink on top of the page. It is possible to navigate to the erroneous parameter via the error message.

Dialogs display only one error at a time on top of the dialog. It is not possible to navigate to the erroneous parameter.



### 8.2.5.6.1. Using multi-page support

A multi-page is registered as GUI page in the wizard extension point. The multi-page class manages the single pages with the following methods:

`doGetFirstPage()`

Returns the first single page.

`doGetNextPage()`

Depending on the current page, the next page is returned.

`doHasNextPage()`

This method manages the **Next** button in the dialog. You do not need to manage the way back through the pages, as the Guided Configuration API does this by remembering the page history.

### 8.2.5.6.2. Validating

When you open the **Demo5** wizard as dialog via the **Sidebar** view, a single error or warning is displayed at a time on top of the page. Warnings are only displayed if no errors exist anymore. Validation always checks the complete memento that contains all parameters of all pages. Thus the first page displays an error from the last page and vice versa. It is not possible to navigate to the erroneous widget.

The **Finish** button is only enabled if no errors exist anymore. You can finish the wizard, even if warnings are still contained.

For dialogs, all buttons except the **Cancel** button are disabled while validation is running. For long running validation operations a progress bar is shown. The validation is always started asynchronously, so you can change the wizard data while validation is running.

For dialogs you can control the workflow by controlling the enablement of the **Back** and **Next** buttons. In this way you can prevent the user from clicking the **Next** button while there is still an error on the page. To control the enablement of the **Back** and **Next** buttons, override the method `AbstractBackend.doValidationFinished()`:

```
public void doValidationFinished( List<ValidationResult> results ) {
    // if an error occurred, then the next button is disabled
    for( ValidationResult result : results ) {
        if( result.getStatus().getSeverity() == APIOperationStatus.SEVERITY_ERROR )
        {
            getPage().setNextButtonEnabled( false );
            return;
        }
    }
    getPage().setNextButtonEnabled( true );
}
```



When you open the **Demo5** wizard as custom module editor, all errors and warnings are displayed as summary on top of the page. To open a dialog in which you may select the errors and warnings, click the hyperlink. The corresponding widget then becomes the focus no matter whether the error message is available on the current page or on another page.

For more information about validation, see [Section 8.2.6.7, “Validating the data”](#).

### 8.2.5.7. Demo6

The wizard for demo6 is available in the **Sidebar** view. In this demo you learn

- ▶ how to restrict the visibility of a wizard via the backend implementation,
- ▶ how to disable the undo/redo support in dialogs,
- ▶ and the usage of a special kind of label for the **Sidebar** view.

#### 8.2.5.7.1. Restricting the visibility

The `Backend` class overwrites the method `doIsAvailable`. If the current project does not have a configuration of the module `GCDemo6_Activator` or it is disabled, then the method returns an `APIOperationStatus`:

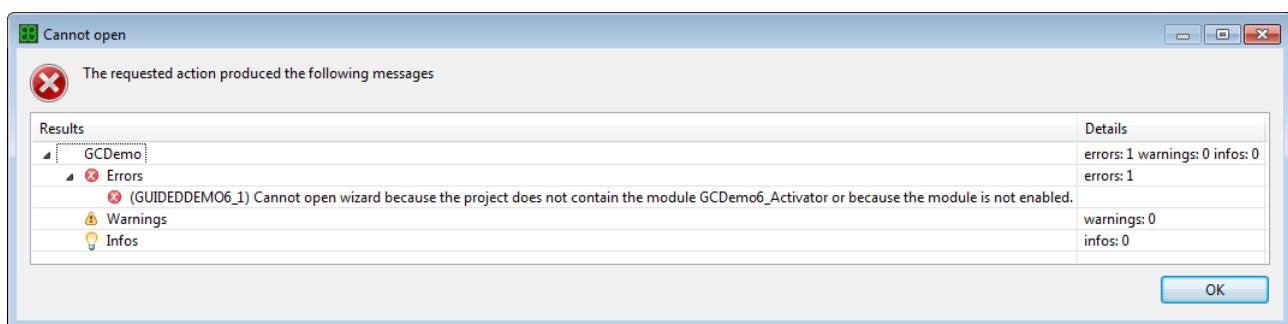


Figure 8.41. Demo6: Wizard cannot be opened

To be able to open the dialog, add the modules `GCDemo6` and `GCDemo6_Activator` to your project and make sure both are enabled. Then you can open the wizard and no error dialog is displayed:

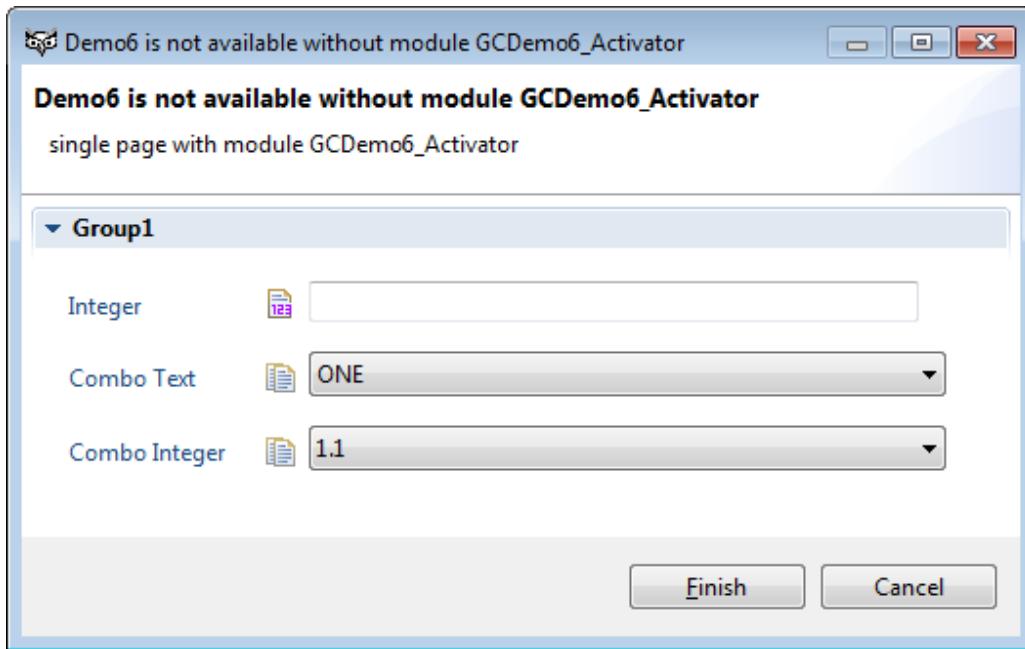


Figure 8.42. Demo6: Dialog if project has the module `GCDemo6_Activator` configured and enabled

#### 8.2.5.7.2. Disabling the Undo/redo support in dialogs

In this demo the undo/redo support in dialogs is disabled. By default, guided configuration dialogs feature undo/redo buttons on the top. For dialogs, you can disable undo/redo support for special use cases.

To disable the undo/redo support in dialogs, set the attribute `undoRedo` of the element `sidebar` to `false` in the extension point.

#### 8.2.5.7.3. Using hierarchical Sidebar labels

This demo uses a specific label in the **Sidebar** view. The docking point is rendered as a tree.

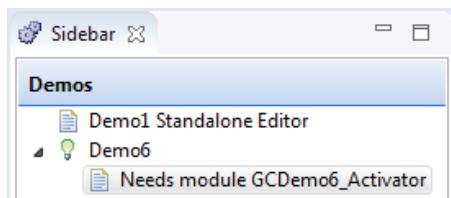


Figure 8.43. Demo6: Specific tree label in the **Sidebar** view

Define this label when you register the wizard in the extension point. The label may contain slashes /, each of which creates a subnode in the tree. If there are several wizards registered that use the same subpath, they are all leaf nodes of the same tree branch.



```

<extension
    point="dreisoft.tresos.guidedconfig.api.plugin.trigger">
    <trigger>
        <sidebar
            categoryId="Demos"
            id="guidedDemo6"
            undoRedo="false"
            wizardId="guidedDemo6"
            wizardType="dialog">
            <visibility>
                <with variable="ECUConfigContext.moduleType.GCDemo6">
                    <equals value="true"/>
                </with>
            </visibility>
            <display>
                label="Demo6/Needs module GCDemo6_Activator"
                tooltip="Demo6 needs module GCDemo6_Activator">
            </display>
        </sidebar>
    </trigger>
</extension>

```

### 8.2.5.8. Demo7

Demo7 provides nearly the same unattended wizard as demo3 with the difference, that it uses the already existing result widget `ChangedDCtxtsResultWidget`.

It shows the changed parameters as a tree in the same way, as the **Search** view does.

To use this widget:

- ▶ Register the `ChangedDCtxtsResultWidget` instead of your own custom *result widget* in the extension point:

```

<resultGui
    class="dreisoft.tresos.guidedconfig.api.gui.page.ChangedDCtxtsResultWidget"
    plugin="dreisoft.tresos.guidedconfig.api.plugin">
</resultGui>

```

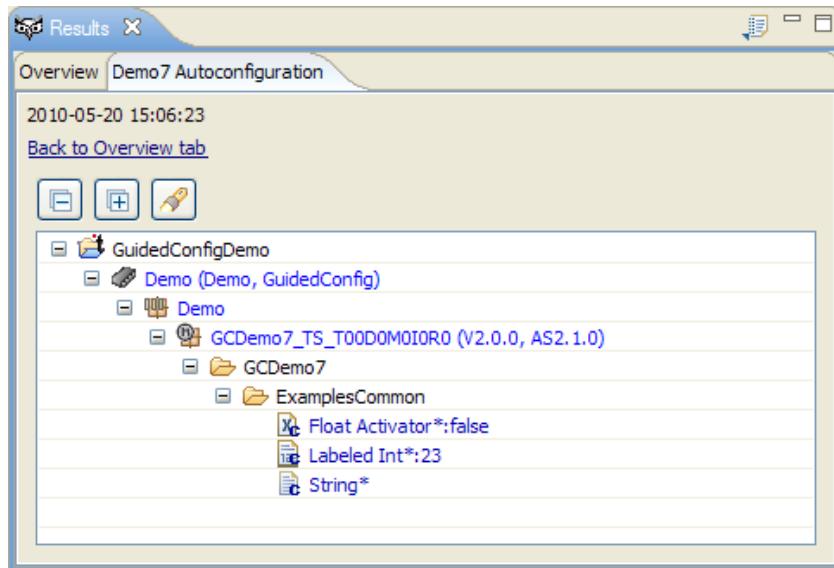


Figure 8.44. Demo7: Easy to use result widget

#### 8.2.5.9. Demo8

The wizard for demo8 is available in the **Sidebar** view and opens a standalone editor in the editor area of EB tresos Studio.

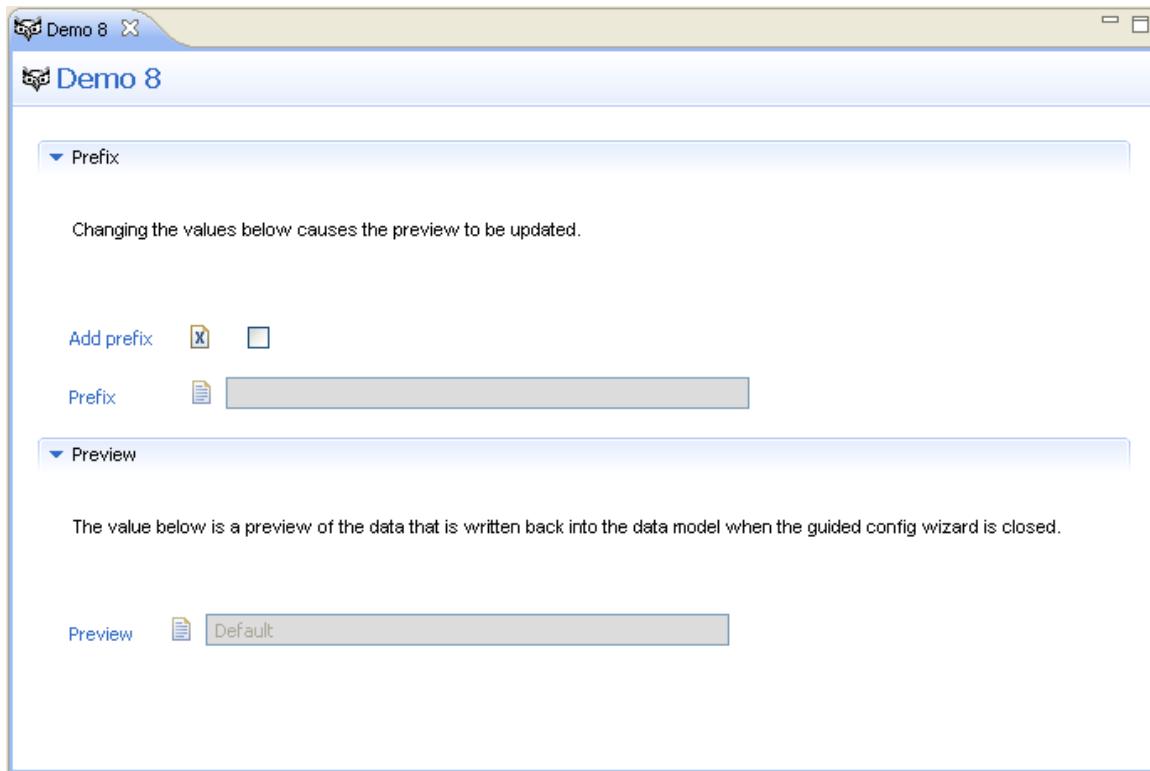


Figure 8.45. Demo8: XForms single-page

In this example you learn:

- ▶ how to use an XForms single-page description in XML format, and
- ▶ how to use a custom memento factory to define default values, and
- ▶ how to validate the wizard data, and
- ▶ how to use push events to query values from the data model and write back values.

#### 8.2.5.9.1. Using an XForm GUI description

The XForm page describes the single-page of the demo8. You can find this XML file in the demo plug-in at `pages/Pages.xml`. To register an XForms page description, see [Section 8.2.6.4, “Using XForms as GUI description”](#).

#### 8.2.5.9.2. Using a custom MementoFactory

This demo uses its own `MementoFactory` class instead of the `DefaultMementoFactory` class. Via the `MementoFactory`, the demo defines default values which are shown the first time the wizard is opened:



Add prefix  
is cleared.

Prefix  
is empty.

Preview  
shows the value *Default* and is disabled.

For instructions on how to create your own `MementoFactory`, see [Section 8.2.6.9, “Creating a MementoFactory”](#)

#### 8.2.5.9.3. Validating

When you change the value of the check box *Add prefix* to true, and the value of the parameter *Prefix* is empty, an error message is displayed on top of the page. The error message informs you that the parameter prefix might not be empty.

#### 8.2.5.9.4. Using the push service

When you open the **Demo8** wizard, a `DataTransferEvent` with condition `EVENT_DIRECTION_PULL` is sent. In the `plugin.xml` a `PushOperation` class with the ID `Demo8PullOperation` is registered, which is executed if a `DataTransferEvent` with condition `EVENT_DIRECTION_PULL` is sent. Then the corresponding class `Demo8PullOperation` is invoked and writes the values from the data model to the memento tree.

Closing the wizard triggers again sending a `DataTransferEvent`, but this time with the condition `EVENT_DIRECTION_PUSH`. Now `Demo8PushOperation` is invoked as registered in `plugin.xml`. The `Demo8PushOperation` writes the value of the wizard parameter *Preview* back to the data model.

If you want to know how to use the push service, see [Section 8.2.6.10, “Using the push service”](#).

#### 8.2.5.10. Demo9

This demo shows you how to place guided configuration widgets outside of a guided configuration page, e.g. on a dialog or view. These widgets still work on the memento structure and are managed by the `WidgetFactory`.

The wizard for demo9 is available in the **Sidebar** view and opens a standalone editor in the **Editor** view of EB tresos Studio.

In this example you will learn:



- ▶ how to place guided configuration widgets outside of a guided configuration page, e.g. into a view or dialog.

#### 8.2.5.10.1. Working with the demo

To open the guided configuration demo9, double-click the item **Demo9 Standalone Editor** in the category **Demos** in the **Sidebar** view. A standalone editor opens up.

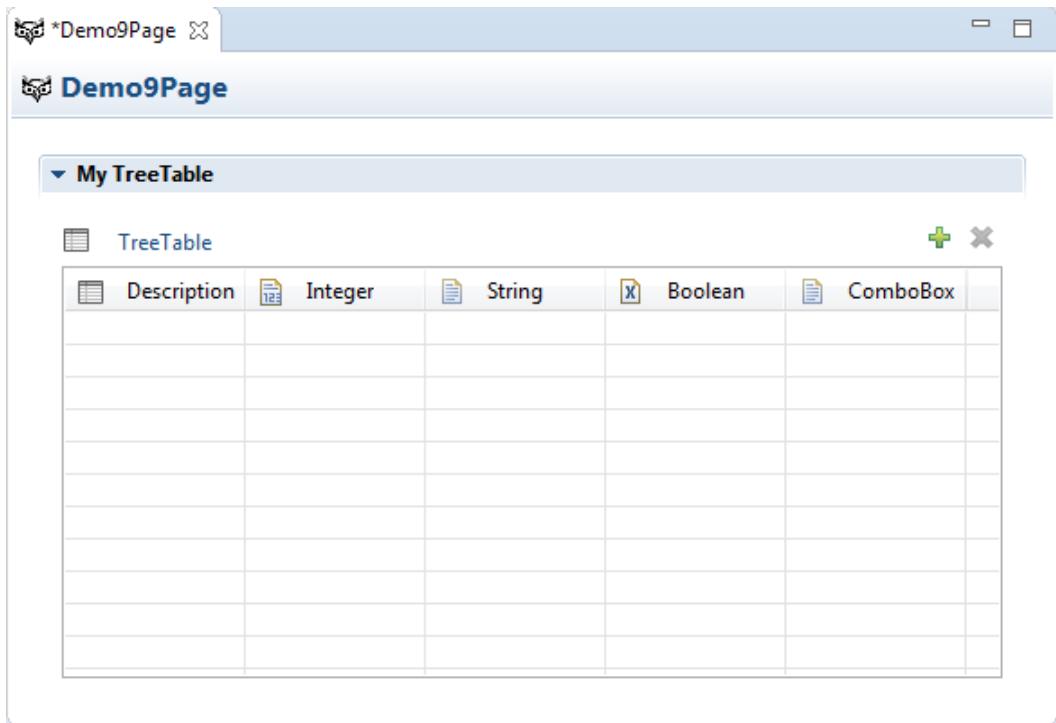


Figure 8.46. Demo9: Standalone editor

The standalone editor **Demo9Page** contains a tree table. In this tree table, you may add rows:

- ▶ To add rows to the tree table, click the **Add element** button .
- ▶ To add subitems to a row:
  - ▶ Select one of the rows in the tree table.

The **TreePropertiesView** view opens up.



Index	Integer	String	Boolean	ComboBox
0	2	abc	X	ONE
1	2	abc	X	ONE
2	2	abc	X	ONE

Commit changes

Figure 8.47. Demo9: Adding subitems to a row of a tree table in the **TreePropertiesView** view

- ▶ To add a new element with default values, click the **Add element** button in the **TreePropertiesView**.
- ▶ Click **Commit changes**.

The new subitems and their values are now written to the tree table and displayed there. When you close the standalone editor, the data is saved persistently.

- ▶ To edit the values of the **TreePropertiesView** view:
  - ▶ Double-click a table entry in the **TreePropertiesView**.

The **Adapt values** dialog opens up.

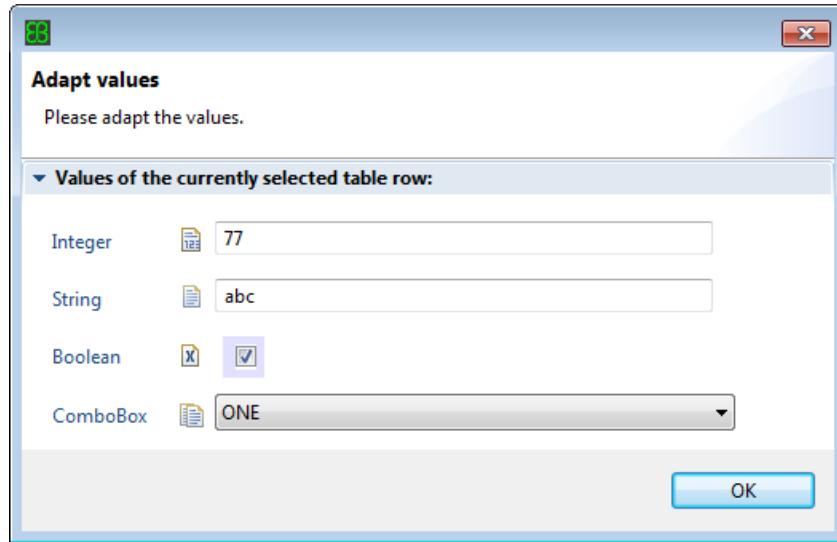


Figure 8.48. Demo9: Adapting values of the tree table

- ▶ Edit the values as you wish.
- ▶ Click **OK**.

The new values are now displayed in the **TreePropertiesView**. If you click **Commit changes**, the recently added or edited values are displayed in the tree table of the standalone editor.

#### 8.2.5.10.2. Placing guided configuration widgets on an external view or dialog

To place guided configuration widgets outside of the normal guided configuration page, i.e. on an external view:

1. Create a class that is extended from a basic dialog or view class.
2. Give this class a reference to your **Backend class** (extended from **AbstractBackend**).
3. Use the **WidgetFactory** to create the widgets on your view/dialog:

```
WidgetFactory factory = m_backend.getWidgetFactory();

SWTGroup outerGroup = factory.createGroup( container,
    "WidgetGroupId_1",
    WidgetFactory.GroupDecoration.TITLE,
    WidgetFactory.GroupLayout.VERTICAL,
    "Values of the currently selected table row:" );

factory.createLabel( outerGroup,
    IConstants.WIDGET_ID_LABEL + "Int",
    IConstants.MEMENTO_KEY_LABEL + "Int",
```



```
MyEclipseNLS.LABEL_COLUMN_INT() );
```

4. If you want to show error and warning markers in the widgets of an external view/dialog, force a validation, so that the validation results are updated:

```
m_backend.validate();
```

5. If you change values programmatically, update the GUI afterwards using the method `updateView()`:

```
m_backend.updateView( tableRoot.getPath() );
```

## 8.2.6. Common use cases

### 8.2.6.1. Registering a wizard

In this chapter you learn how to register and define a wizard.

Open the **Plug-in Manifest Editor** by double-clicking the `MANIFEST.MF` file. To register a wizard, select the tab *Extensions* and add the following extension point:

► `dreisoft.tresos.guidedconfig.api.plugin.guidedconfigwizard`

You find the extension point description as a link in the `plugin.xml` editor.

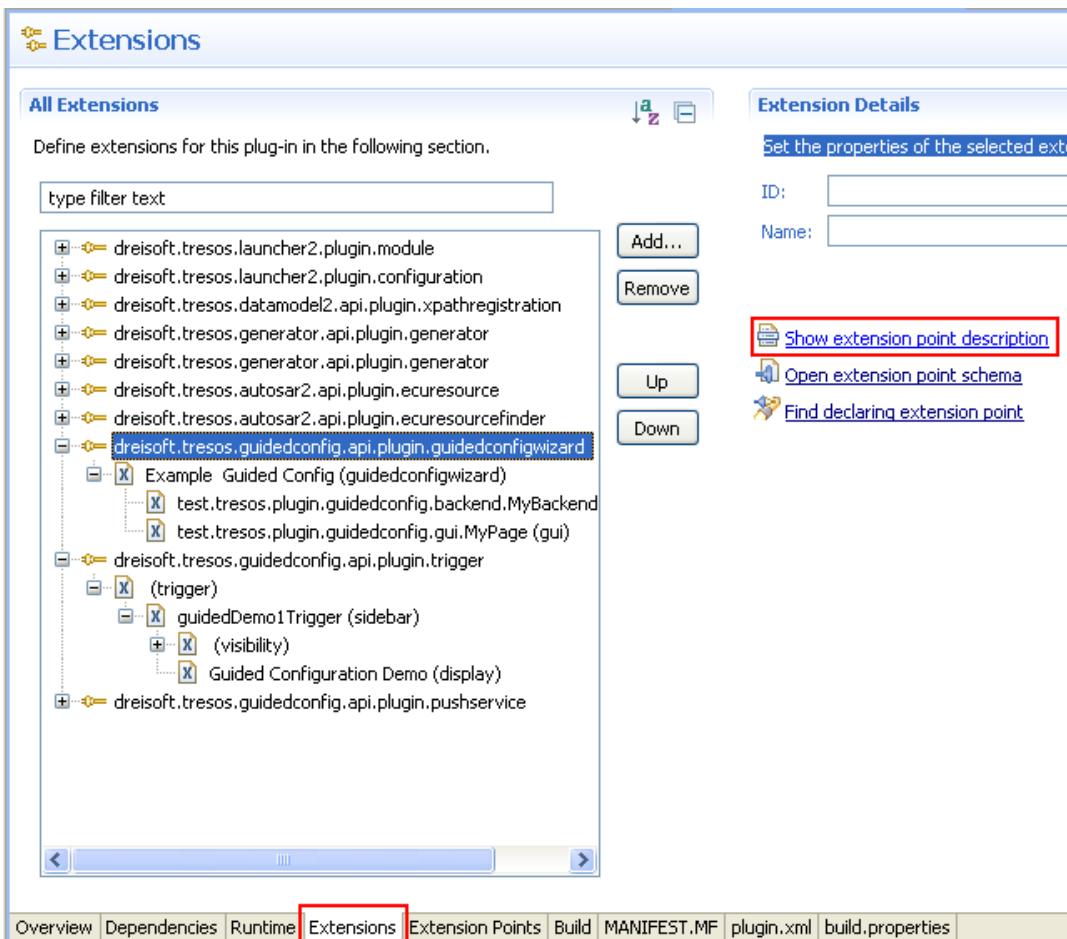


Figure 8.49. Plug-in Manifest Editor: Extensions tab

Add your required parameters and give the guided configuration wizard an ID. The ID is referenced from the corresponding docking point. Multiple docking points may share the same wizard registration. The following parameters are optional:

#### icon

This parameter indicates the path to an icon that is displayed in the GUI. If you leave it blank, the default icon is displayed.

#### helpContextId

The `helpContextId` parameter is a unique identifier that indicates the help context for this action.

#### basePath

This parameter indicates the path to the resources that are necessary for the wizard, e.g. subfolder `icons` for the GUI widget icons.

#### basePathPlugin

If the path given in the `basePath` parameter is not located in the current plug-in, the `basePathPlugin` parameter declares the path to the plug-in which holds the given `basePath`.



Now add the subparameters:

`backend`

The `Backend` class of the wizard.

`gui`

The GUI representation of the wizard: either single-page or multi-page.

`resultGui`

The `result widget` is optional. If you do not add your own `result widget`, the `DefaultResultWidget` is used.

All three parameters (`backend`, `gui`, and `resultGui`) have two entries themselves, the `class` and the `plugin` entry. Whereas only the `class` entry is of interest.

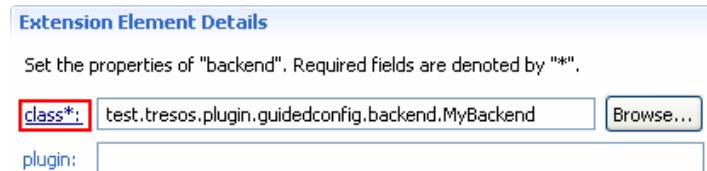


Figure 8.50. Creating the classes

When you open the link `class*`, the class creation dialog opens up. The appropriate abstract classes to extend are already added to the class creation dialog. Give the new class a name and a location and click **Finish**.

### 8.2.6.2. Registering the docking point

This chapter provides an explanation on how to register the different kinds of docking points in the GUI. Find information about the docking points in [Section 8.2.3.1, “Wizard type overview”](#)

#### 8.2.6.2.1. Adding an editor or dialog to the Sidebar view

To add a standalone editor or a dialog to the **Sidebar** view, add the following extension point to your `plugin.xml`:

- ▶ `dreisoft.tresos.guidedconfig.api.plugin.trigger`

To expand the trigger extension point, right-click on **(trigger)**, select **New**, and then choose **sidebar**.

- ▶ Assign a trigger **ID** and enter the **wizard-ID** of your guided configuration wizard. In the drop-down menu, select either **editor** or **dialog** as **wizardType**.
- ▶ The field **categoryID** is for grouping several wizards. If more than one wizard has the same `categoryID`, the wizards are displayed in the same group in the **Sidebar** view.



- ▶ In the **display** element enter a label, which is displayed in the **Sidebar** view. If the label contains slashes/, the trigger is displayed in a tree-element.

Additionally it is possible to add an optional icon and a tooltip for the wizard.

- ▶ To add constraints for the visibility of the trigger, use the **visibility** element. Therefore information provided by the *selection context* is used. For information on how to do this, have a look at the extension point description or see [Section 8.2.6.3.1, “Via extension point”](#).

Create a trigger and choose *sidebar*.

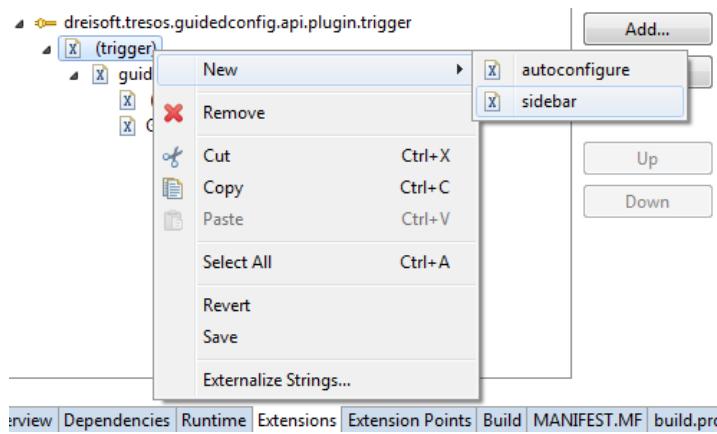


Figure 8.51. Adding a sidebar trigger

Example of the registration for a standalone editor:

```
<extension
    point="dreisoft.tresos.guidedconfig.api.plugin.trigger">
    <trigger>
        <sidebar
            categoryId="Project"
            id="guidedDemoEditorTrigger"
            wizardId="guidedDemo1"
            wizardType="editor">
            <visibility>
                <with variable="ECUConfigContext.projectname">
                    <equals value="test"/>
                </with>
            </visibility>
            <display>
                label="Guided Configuration Demo Editor"
                tooltip="Guided Configuration Demo Editor">
            </display>
        </sidebar>
    </trigger>
</extension>
```



If e.g. the currently selected project is named *test*, the trigger is displayed in the **Sidebar** view with the following label:

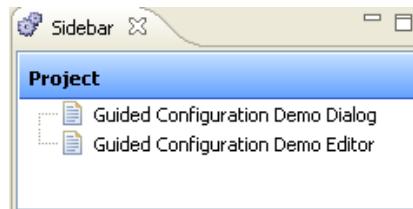


Figure 8.52. **Sidebar** view

You can register a dialog wizard the same way as the standalone editor: Give it a new label and ID, and change the parameter `wizardType` in the extension point:

- ▶ `wizardType="dialog"`

You can set the initial size of a dialog by specifying the `width` and `height` attributes in the extension point:

- ▶ `width="700"`
- ▶ `height="700"`

### 8.2.6.2.2. Creating a custom module editor

If you want to create a custom module editor:

1. Register the following extension points:

- ▶ `dreisoft.tresos.launcher2.plugin.configuration`
- ▶ `dreisoft.tresos.launcher2.plugin.module`

2. Pass `dreisoft.tresos.guidedconfig.api.gui.editor.GuidedConfigEditor` as `editor` class, as well as a parameter `wizardId` with the ID of your wizard as value.

---

**NOTE**

**Register a custom module editor via the module extension point**



Register a custom module editor via the module extension point and not via the trigger extension point.

---

Example for the registration of a custom module editor:



```
<extension point="dreisoft.tresos.launcher2.plugin.configuration"
    id="ModulePlugInConfiguration"
    name="ModulePlugIn Parameter Definition Extension">
    <configuration moduleId="ModulePlugIn_GuidedConfigdraft">
        <editor id="ModulePlugIn_GuidedConfigdraftEditor"
            label="GuidedConfigdraftEditor"
            tooltip="sample module">
            <class
                class="dreisoft.tresos.guidedconfig.api.gui.editor.GuidedConfigEditor"
                plugin="dreisoft.tresos.guidedconfig.api.plugin">
                <parameter name="wizardId" value="guidedDemo"/>
            </class>
        </editor>
    </configuration>
</extension>
```

### 8.2.6.2.3. Registering an unattended wizard

To register an unattended wizard, add the following extension point to your `plugin.xml`:

► `dreisoft.tresos.guidedconfig.api.plugin.trigger`

To expand the trigger extension point, right-click on **(trigger)**, select **New**, and then choose **autoconfigure**. Assign a trigger ID and enter the ID of your guided configuration wizard.

Example for the registration of an unattended wizard:

```
<extension
    point="dreisoft.tresos.guidedconfig.api.plugin.trigger">
    <trigger>
        <autoconfigure
            id="guidedDemoAutoTrigger"
            wizardId="guidedDemoAuto">
            <visibility>
                <with variable="ECUConfigContext.projectname">
                    <equals value="test"/>
                </with>
            </visibility>
            <display>
                label="Guided Configuration Demo Autoconfigure"
                tooltip="Guided Configuration Demo Autoconfigure">
            </display>
        </autoconfigure>
    </trigger>
</extension>
```



```
</trigger>
</extension>
```

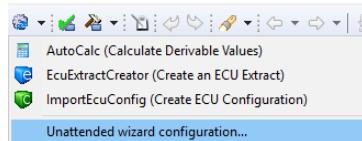


Figure 8.53. The unattended wizards menu

The unattended wizard trigger has more than one run option:

- ▶ Click the **View** button and choose **Unattended wizard configuration...**, which opens the **Unattended Wizards** configuration dialog. In this dialog, configure all runnable unattended wizards. To run a selected unattended wizard, select the required unattended wizard and click the **Run** button.
- ▶ In the **Unattended Wizards** configuration dialog, select a check box for each required unattended wizard and click **Close**.

Click the **Open unattended wizard configuration dialog** button in the tool bar, which opens the **Unattended Wizards** configuration dialog. In this dialog, configure all runnable unattended wizards. To run a selected unattended wizard, select the required unattended wizard and click the **Run** button.

- ▶ You can click the **View** button to open the context menu. Triggers that were earlier used(favorites) are selected in the **Unattended Wizards** configuration dialog. You can run each of them by clicking the trigger entry in the menu. The same applies for the **Unattended Wizards** menu item of the **Project** menu.

### 8.2.6.3. Restricting the visibility of a wizard

The visibility can only be restricted for **Sidebar** wizards (i.e. dialogs and standalone editors) and unattended wizards. If the corresponding module is added to the project, the custom module editor is always available.

There are two ways to restrict the visibility of the wizard:

1. [Section 8.2.6.3.1, “Via extension point”.](#)
2. [Section 8.2.6.3.2, “Via Backend implementation”.](#)

#### 8.2.6.3.1. Via extension point

To restrict the visibility via extension point, use the parameter `visibility` in the `dreisoft.tresos.guidedconfig.api.plugin.trigger` extension point. You may add several conditions in combination with AND/OR operations.



You find the parameters which can be queried in chapter [Section 8.2.3.2, “Selection context”](#).

```
<trigger>
  <sidebar categoryId="Demos" id="Demo1" wizardId="Demo1" wizardType="editor">
    <visibility>
      <and><with variable="ECUConfigContext.target">
        <equals value="TEST"/></with>
      <with variable="ECUConfigContext.moduleType.PublicApiTest">
        <equals value="true"/></with>
      </and>
    </visibility>
    <display label="Demo1Label" tooltip="Demo1Tooltip"/>
  </sidebar>
</trigger>
```

### 8.2.6.3.2. Via Backend implementation

The `Backend` class provides the method `doIsAvailable`. To restrict the visibility, you have to overwrite this method.

- ▶ If you do not want the backend to further restrict the visibility, this method must return null.
- ▶ If you want to restrict the visibility, an `APIOperationStatus` containing the error message has to be returned.

The default implementation returns null.

If the visibility conditions are not met and an `APIOperationStatus` is returned, the message is displayed in a dialog. The dialog provides the possibility to copy the message from the dialog to the clipboard.

```
public APIOperationStatus doIsAvailable() {
  if( (getContexts().get( SystemConfigContext.ID ) == null)
  || (!getContexts().get( SystemConfigContext.ID ).isEnabled()) ) {
    return MyGuidedConfigOperationStatus.ERROR_OPEN_WIZARD();
  }
  return null;
}
```

### 8.2.6.4. Using XForms as GUI description

Instead of writing the GUI representation in Java, you can also register an XML file that defines the GUI.



Example to register an XForms multi-page:

```
<extension
    point="dreisoft.tresos.guidedconfig.api.plugin.guidedconfigwizard">
<guidedconfigwizard id="guidedDemo1">
    <gui
        class="dreisoft.tresos.guidedconfig.api.gui.xform.XFormMultiPage">
            <parameter name="pageDescription" value="pages/Pages.xml"/>
    </gui>
    <backend class="test.tresos.plugin.guidedconfig.demo1.Demo1Backend" />
</guidedconfigwizard>
</extension>
```

To register a single-page GUI representation, register the following class:

```
<gui class="dreisoft.tresos.guidedconfig.api.gui.xform.XFormPage">
```

You can use all widgets provided by the guided configuration framework via XForms description. These widgets are described in [Section 8.2.3.5.5, “Predefined widgets”](#).

You find the widget descriptions in the `WidgetFactory` class. For each widget type there exists an instance of a `DefaultWidgetDescription` and a method named `create<type>`. The `DefaultWidgetDescription` defines the widget type and the default parameters. The type, in the following example `button`, is also the name of the XML tag to create the widget. The type is always written in lower case and the corresponding XML tag starts with an upper case letter.

For example, the type `Button` provides the following parameters and their default values:

```
private final DefaultWidgetDescription DESC_BUTTON = new
DefaultWidgetDescription(
this,
"button",
new DefaultParameterDescription[]{
new DefaultParameterDescription( "widgetId", String.class ),
new DefaultParameterDescription( "mementoId", String.class, "__Button" ),
new DefaultParameterDescription( "text", String.class, "" ),
new DefaultParameterDescription( "icon", String.class, "" ),
new DefaultParameterDescription( "flag", ButtonFlags.class, "PUSH" )} );
```

Creating the button via XForms:



```
<Button widgetId="button" icon="icons/sample.gif" flag="TOGGLE"/>
```

In this example, a button is added with:

- ▶ the widget-ID `button`, and
- ▶ an icon instead of text as label, and
- ▶ with the type `TOGGLE`, which has two states: `ON` and `OFF`.

In this example, not all parameters that are provided by the button widget are set. The parameters you do not define, are filled with their default values.

Some features for the GUI representation via Java code are not provided by the XForms description, e.g. adding selection events. But the Guided Configuration API provides the possibility to change the GUI representation right before the page is displayed.

To be able to change the GUI, register your own page class inherited from `XFormMultiPage`(in case of multi-pages) or `XFormPage`(in case of a single-page). Then overwrite the method `doAdaptWidget( ISWTWidget widget )`, to adapt the GUI representation of the given widget.

### 8.2.6.5. Creating the page

Instead of registering an XForms GUI description, you can write the GUI representation in Java.

You may choose to create:

- ▶ A single-page. Your page class must then extend `AbstractPage`.
- ▶ A multi-page. Your page class must then extend `AbstractMultiPage`.

An editor shows multi-pages in different tabs in the editor. In a dialog the user can navigate to the different pages with the **Back/Next** buttons. You can manage this navigation with the Guided Configuration API.

To fill the page with some widgets, open your page class and add the following to the `doCreateControls(Composite parent)` method:

```
// fetch the widget factory
WidgetFactory factory = getWidgetFactory();

// create a label
factory.createLabel( parent, "LABEL1", "LABEL_MEMENTO1", "LabelText Integer" );

// create a text box of type integer
factory.createText( parent, "TEXT1", "INT_MEMENTO1", TextFormat.INTEGER );
```



```
factory.layout( parent );
```

---

**NOTE**



**Calling `factory.layout( parent )` at the end is mandatory**

`factory.layout( parent )` *MUST* be called at the end. You may change the layout afterwards as you like, but to finish laying out the page, call `factory.layout( parent )` on the parent composite for at least one time, after all GUI elements were added.

---

Implement the following methods:

- ▶ `doGetId()` has to return the page ID. In a multi-page, this ID has to be unique.
- ▶ `doGetDescription()` has to return a description for the page.
- ▶ `doGetTitle()` has to return the page title.

#### 8.2.6.6. Creating a custom widget

To create a custom widget, you have to implement a controller and a view part.

The view part has to implement the interface `ISWTWidget` with the following methods:

`getWidgetId()`  
returns the widget-ID of the custom widget.

`getParent()`  
returns the parent composite of the custom widget

To connect to the wizard lifecycle, extend the controller part `AbstractWidgetController`.

To connect the widget to a memento, register it at the `WidgetFactory` by calling the following method:

```
registerWidget(ISWTWidget widget, String widgetId, String mementoId)
```

To provide undo/redo support, register the operations on a memento with the `MementoOperationHandler` methods. For more information about the `MementoOperationHandler`, see [Section 8.2.3.4.1, “The MementoOperationHandler class”](#).

For an example how to add the custom widget to the page, see [Section 8.2.5.3, “Demo2”](#)

```
// create Custom Datetime widget
MyDatetime datetimeWidget = new MyDatetime( WIDGETID_DATETIME, customGroup );
// create controller for widget to receive events from the wizard engine
MyDatetimeController datetimeController = new MyDatetimeController(
```



```
        datetimeWidget, MEMENTOID_DATETIME );
// register controller at the WidgetFactory
factory.registerWidgetController( datetimeController );
```

If the custom widget is used outside of a guided configuration page (as described in [Section 8.2.6.5, “Creating the page”](#)), the widget needs to ensure that its controller is unregistered at the factory. Therefore, register a dispose listener at one of the internal SWT widgets of the custom widget, i.e. within the constructor code. For information on registering a custom widget, see [Section 8.2.5.3, “Demo2”](#).

```
// Add dispose listener for notifying the controller about the widget being
// disposed.
// Required when widget is used outside of a guided configuration Page.
m_widget.addDisposeListener( new DisposeListener() {
    public void widgetDisposed( DisposeEvent e ) {
        if( m_controller != null ) m_controller.doWidgetIsDisposed();
    }
} );
```

## 8.2.6.7. Validating the data

The guided configuration framework provides the possibility to validate the data of the widgets and show error and warning markers in the widget and on top of the page on which the widget is placed.

To validate the widget data, override the method `doValidate()` of your Backend class.

Example:

```
public List<ValidationResult> doValidate() {
    // Initialize list of validation results
    ArrayList<ValidationResult> result = new ArrayList<ValidationResult>();

    // Get root memento of wizard
    Memento rootMemento = getMemento();

    // show an error if the integer field is empty
    String value = rootMemento.getString(
        ( Demo1MementoFactory.MEMENTOID_PAGE2_INTEGER ) );
    if( value == null || "".equals( value ) ) {
        result.add( new ValidationResult(
            Demo1MementoFactory.MEMENTOID_PAGE2_INTEGER,
            MyGuidedConfigOperationStatus.DRAFT_ERROR(
```



```

        "Integer field on page2 must not be empty!" ) ) );
}
return result;
}

```

As the `ValidationResult` contains a memento-ID and a message, the guided configuration framework can link from the message to the corresponding widget.

The validation is started each time:

- ▶ a wizard is opened, or
- ▶ a parameter in the GUI changes, or
- ▶ another wizard in the **Unattended Wizards** configuration dialog is selected.

If you change parameters one by one, the validation currently running is canceled and a new validation starts. If the `doValidate()` method contains some long running operations, it is recommended to make the validation cancelable. To enable cancellation of the validation, make the `doValidate()` method query whether the validation has been canceled and let it return results:

```
if( isValidationCanceled() ) return result;
```

### 8.2.6.8. Using selection events

Some widgets provide events, so you can react on actions of the user. The following widgets provide selection events:

- ▶ button
- ▶ table
- ▶ tree table

Example for implementing a button selection event

```

protected void doCreateControls( Composite parent ) {
    // retrieve the widget factory
    WidgetFactory factory = getWidgetFactory();
    // add some widgets
    ...
    // create push button
    SWTButton sortButton = factory.createButton( group3,

```



```

        "ChangeSortColumn",
        "ChangeSortColumn",
        "ChangeSortColumn",
        "spy.gif",
        ButtonFlags.PUSH );

// create your own
ISelectionEventListener
SortColumnSelection sorting = new SortColumnSelection();
// add SelectionListener to the button widget
sortButton.getController().addSelectionEventListener( sorting );
// change tooltip text
sortButton.setToolTipText( "Change sort column to column 'Text'." );
factory.layout( parent );
}

private static class SortColumnSelection implements
ISelectionEventListener {
    public void selectionChanged( WidgetSelectionEvent event ) {
        if( !(event instanceof ButtonSelectionEvent) ) return;
        m_table.setSortColumn( MyEclipseNLS.LABEL_TEXT(),
                               SortFlags.COLUMN_ORDER_ASCENDING );
    }
}
}

```

### 8.2.6.9. Creating a MementoFactory

Per default the `DefaultMementoFactory` is used and all mementos are persistent. You may change this behavior by creating your own `MementoFactory`.

A custom `MementoFactory` must extend `AbstractMementoFactory` and implement only one method `getMemento( Memento parent, String name )`. Use this method to:

- ▶ set default values or calculate values for the mementos
- ▶ decide whether a memento is to be persistent or not.

```

public class MyMementoFactory extends AbstractMementoFactory {
    public static final MyMementoFactory INSTANCE = new MyMementoFactory();
    /*
     * Create a new Memento with the given name and
     * set default values for the root memento
     */
    public Memento getMemento( Memento parent, String name ) {

```



```
Memento memento = new Memento( name, this );
if( parent == null ) {
    memento.setString( IConstants.MEMENTO_KEY_TEXT1, "defaulttext1" );
    memento.setString( IConstants.MEMENTO_KEY_TEXT2, "defaulttext2" );
}
return memento;
}
}
```

## 8.2.6.10. Using the push service

In this chapter you learn how to register a *push operation* and how to start this operation by sending a *push event*.

### 8.2.6.10.1. Registering a *push operation*

In the **Plug-in Manifest Editor**, select the tab **Extensions**. Add the following extension point:

► dreisoft.tresos.guidedconfig.api.plugin.pushservice

After adding the extension point, assign an ID, a name, and a description and create the PushOperation class by clicking the **class** link.

You may restrict the the *push operation* to a special event or selection context via the event tag:

```
<extension point="dreisoft.tresos.guidedconfig.api.plugin.pushservice">
<pushoperation
    desc="Example PushOperation"
    id="DemoPushOperation"
    name="Example Push Operation">
    <operationclass
        class=
        "test.tresos.plugin.guidedconfig.pushservice.operation.
        ExamplePushOperation"/>
    <event>
        <with variable="class">
            <equals value=
            "test.tresos.plugin.guidedconfig.pushservice.operation.ExampleEvent"/>
        </with>
    </event>
</pushoperation>
```



```
</extension>
```

#### 8.2.6.10.2. Sending a *push event*

To send a *push event*, extend the `AbstractPushEvent` class of the event in the following way:

```
// Create a push event that is sent to a push operation via the PushService
ExampleEvent pushEvent = new ExampleEvent( ExampleEvent.EVENT_TYPE_AUTOCONFIGURE
);
pushEvent.setBackend( this );
pushEvent.setSourceMemento( rootMemento );
pushEvent.setTargetContext( getTargetConfigurationContext( MODULE_SCHEMA_PATH ) );
);

// Trigger the push operations
PushService.getInstance().callSync( pushEvent, false );

// Get the list of changed nodes
List<DCtxt> changedNodes = pushEvent.getChangedNodes();
```

#### 8.2.6.11. Registering a custom result widget

In this section you learn how to register a custom result widget.

To register a custom result widget:

- ▶ Extend the `AbstractResultWidget` class of the result widget.
- ▶ Register the `AbstractResultWidget` class in the `dreisoft.tresos.guidedconfig.api.plugin.guidedconfigwizard` extension point:

```
<extension
    point="dreisoft.tresos.guidedconfig.api.plugin.guidedconfigwizard">
    <guidedconfigwizard id="guidedDemo3">
        backend class="test.tresos.plugin.guidedconfig.demo3.Demo3Backend"/>
        <gui class="test.tresos.plugin.guidedconfig.demo3.Demo3Page"/>
        <resultGui class="test.tresos.plugin.guidedconfig.demo3.
            Demo3ResultWidget"/>
    </guidedconfigwizard>
</extension>
```



## 8.3. Workflow API

### 8.3.1. Purpose

The Workflow API provides the possibility to enhance EB tresos Studio with workflows, which guide the user through the tool. A workflow consists of a list of steps the user has to perform to accomplish a certain task, e.g. to configure a Com stack from scratch.

### 8.3.2. Prerequisites

#### 8.3.2.1. Knowledge required

In order to work with the instructions in this chapter, you need to have knowledge of the following topics:

XML

The workflow description is written in XML.

#### 8.3.2.2. Plug-ins required

After you set up Eclipse as described in [Section 4.2, “How to install Eclipse”](#), create a plug-in project and then add the following plug-in as dependency to the MANIFEST.MF:

- ▶ dreisoft.tresos.workflow.api.plugin

To add this plug-in as dependency:

- ▶ open the plugin.xml or MANIFEST.MF within Eclipse.

A special editor opens, the **Plug-in Manifest Editor**.

- ▶ Select the tab **Dependencies**.
- ▶ Add the plug-in by using the **Add...** button.



### 8.3.3. Design overview

In this chapter you learn about the basic design of the Workflows API.

A workflow with all its steps is described by a workflow description file written in XML.

This workflow description file must be registered either via an extension point in a plugin.xml or programmatically via the Workflows class.

A separate view, called the **Workflows** view, provides the functionality to process a workflow. This special view shows one workflow at a time. Only registered workflows will be available in the **Workflows** view.

---

**NOTE**

**Workflows view**



See the EB tresos Studio user's guide, chapter [The GUI main window](#) for more information about the view part.

---

#### 8.3.3.1. Workflow description

A workflow description is an XML file that describes one workflow including its steps. The workflow schema can be found at:

[http://www.tresos.de/\\_projects/tresos/workflow\\_1\\_0.xsd](http://www.tresos.de/_projects/tresos/workflow_1_0.xsd)

Example for a workflow description file:

```
<?xml version="1.0" encoding="UTF-8"?>
<workflow xmlns="http://www.tresos.de/_projects/tresos/workflow_1_0.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="
    http://www.tresos.de/_projects/tresos/workflow_1_0.xsd"
  id="CreateDemoProject" version="10.9" label="Create Project"
  description="Create demo project">

  <group id="CreateProject" label="Create the project">
    <action id="NewProject"
      label="Create a configuration project">
        <command description="Create the project"
          serialization="
            org.eclipse.ui.newWizard(newWizardId=newConfigurationProject) "
        />
        <description>Create a new project </description>
    </action>
  </group>
</workflow>
```



...

Each workflow contains a tree of steps with a single root node. A step that contains other steps is called *group* step. The leaf nodes of the tree are called *action* steps. The action steps can contain commands that can be performed when processing the workflow.

A group step can either be of the type *sequence* or of the type *choice*. *Sequence* steps should be used if all sub-steps of this group have to be performed to finish the task. If the user can choose to perform only one sub-step, *choice* steps may be used.

The root element of the workflow description is the *workflow* tag. The workflow tag has the following attributes:

Attributes of the workflow tag	Description
<code>id</code>	The unique ID of the workflow.
<code>version</code>	The workflow version. If several module plug-ins that contain a workflow with the same ID are registered, only the plug-in with the highest version is registered and available in the <b>Select a workflow</b> dialog.
<code>label</code>	The label of the workflow, which is displayed when the user selects the workflow in the <b>Workflows</b> view.
<code>icon</code>	The optional icon of the workflow.
<code>description</code>	The optional description of the workflow. It is recommended to enter a description to make the workflow selection easier for the user. The description is interpreted as plain text.

The workflow tag contains exactly one *group* tag. This group tag can itself contain several group and action steps. The group and action steps have the following attributes in common:

Attributes of the group and action tag	Description
<code>id</code>	The unique ID of the step, to be able to link to each step.
<code>label</code>	The label of the step, which is shown in the workflow step tree.
<code>repeatable</code>	The <code>repeatable</code> attribute is used to mark a step as repeatable if it can be performed more than once. This attribute is optional. The default value is <code>false</code> . By default a step is not repeatable and therefore marked as finished after it has been performed once. If a step that is marked as repeatable is performed, the step is not marked as finished after running this step. A repeatable step can be marked as finished by clicking the <b>Finish</b> button.
<code>next</code>	The <code>next</code> attribute contains the next step to navigate to. This attribute is optional. By default if a step is performed, the next step of the hierarchical order is se-



Attributes of the group and action tag	Description
	<p>lected. If you add the <code>next</code> attribute, this behavior can be changed. Thus the order of the performed steps might differ from the hierarchical tree order.</p> <p>The syntax of the <code>next</code> attribute is similar to hyperlinks in HTML, therefore the content of the <code>next</code> attribute is referred to as <i>workflow hyperlinks</i>. For further information about using workflow hyperlinks, see <a href="#">Section 8.3.3.2, “Workflow hyperlinks”</a>.</p>
needsproject	<p>The optional <code>needsproject</code> attribute marks this step to require a project. By default this parameter is set to false.</p> <p>This option shall be used in workflows which start without having a project. In this case the workflow most probably starts with a step to create or import a project. The attribute <code>needsproject</code> marks the project-specific steps of the workflow. If users have already created a new project before they use the workflow, the workflow view pre-selects the first step with the attribute <code>needsproject=true</code>. Users can immediately start to configure their project then.</p>

Each step can also have a sub-tag *description* with an HTML description for the step, which is shown in the description area of the **Workflows** view. The description can contain workflow hyperlinks, either linking to a step in the same workflow or linking to another workflow. The syntax of the hyperlinks is described in [Section 8.3.3.2, “Workflow hyperlinks”](#).

The *group* step has an additional attribute to specify the type of the group:

Additional type attributes	Description
type	<p>The optional group type can have the following values:</p> <ul style="list-style-type: none"> <li>sequence           <p>Type <i>sequence</i> means, that all sub-steps of a group step must be performed to finish the group step. This is the default value for the group type.</p> </li> <li>choice           <p>For the type <i>choice</i> only one of the sub-steps has to be performed.</p> </li> </ul>

Each action step may contain command tags, which are not displayed as nodes in the workflow tree. The commands define the functions executed if the user performs the action step. The commands are based on the Eclipse command framework. Therefore, in addition to the EB tresos Studio specific commands, you may use every command registered via the extension point `org.eclipse.ui.commands`. For further information on the command extension point, see the [Eclipse help](#).

Commands can be of the type *default* or of the type *configure*. The default value is of the type *default*.



The default command is performed either when the user clicks the **Run** button , double-clicks the step or presses **ENTER** while the action step is selected. After the default command has been performed, the step is marked as finished and the next step is selected.

If the default command runs an unattended wizard, this wizard might need to be configured before. Thus you may add a configuration command to configure the unattended wizard first. The configuration command is performed when the user clicks the **Configure** button . The step state and selection will not be changed.

An action can have up to two commands.

- ▶ If the action has no command, only a description informs the user what to do. When performing this action, the step is marked as finished and the next step is selected.
- ▶ If the action has only one command, this command must be of the type *default*.
- ▶ If the action has two commands, one default command and one configuration command must be provided.

Each command contains the following attributes:

Attribute	Description
type	<p>The optional type of the command:</p> <p><b>default</b>            The default command is performed if the user clicks the tool bar button <b>Run</b>  if the user double-clicks the step, or presses <b>ENTER</b> while the action step is selected. This is the default type.</p> <p><b>configure</b>            The configuration command is performed if the user clicks the tool bar button <b>Configure</b> .</p>
serialization	<p>The command is of the form:</p> <pre>commandId [ "(" paramId "=" paramValue            [ "," paramId "=" paramValue ]* ")" ]</pre> <p>For further information about the available EB tresos Studio commands, see <a href="#">Section 8.3.5, “Commands”</a>.</p>
description	The optional description as plain text.
autoadvance	<p>The optional attribute <i>autoadvance</i> controls whether the next step will be selected automatically after executing the default command. By default, this attribute is set to <code>true</code>.</p> <p>You may use this attribute, for example, when opening guided configuration editors. If you set the autoadvance attribute to <code>false</code>, the next step is not any more</p>



Attribute	Description
	selected automatically after the editor is opened. Therefore the description of the current step stays visible.

### 8.3.3.2. Workflow hyperlinks

The order of the steps in the workflow description file is significant. The workflow tree is built from the XML tree structure. But it is possible to use workflow hyperlinks, so two steps can be performed after each other although they are not siblings in the workflow tree. This can be done using hyperlinks similar as in HTML.

The link to another step is of the form:

```
"workflow://" [ workflowId ] "/" stepId
```

With this link you may locate a step in the current workflow if the `workflowId` is omitted. If a `workflowId` that differs from the currently displayed workflow is provided with the hyperlink, the workflow with the given ID will be displayed and the step which matches `stepId` is selected in this workflow. If the step cannot be found, an error message appears.

It is also possible to use these links inside the HTML step description which is displayed in the description area:

```
<a href="workflow:///CalculateHandleIds">
```

In this example no workflow Id is provided, so this link only refers to a step inside the current workflow.

### 8.3.3.3. Workflow registration

There are two different ways to register a workflow description file:

- ▶ [Section 8.3.3.3.1, “Registration via extension point”](#)
- ▶ [Section 8.3.3.3.2, “Registration via workflows class”](#)
- ▶ [Section 8.3.3.3.3, “Registration via project”](#)

Registered workflows are not remembered persistently. The registration will be lost if EB tresos Studio is being restarted. Workflows registered via extension points are registered automatically at startup. In addition, the workflow registration via extension points does not require you to write Java. However, the initial workflow can only be set programmatically.



### 8.3.3.3.1. Registration via extension point

It is possible to register a workflow via the `dreisoft.tresos.workflow.api.plugin.workflow` extension point in the `plugin.xml` of a module.

```
<extension
    point="dreisoft.tresos.workflow.api.plugin.workflow">
    <workflow
        file="resources/WorkflowDemo1.xml"
        plugin="WorkflowDemo1">
    </workflow>
</extension>
```

The path to the workflow description file is interpreted relatively to the declaring plug-in if the `plugin` attribute is empty. Otherwise it is searched in the given plug-in.

### 8.3.3.2. Registration via workflows class

The `Workflows` class provides the possibility to register a workflow description at runtime:

```
Workflows.getInstance().registerWorkflow(
    new File( "resources/WorkflowDemo1.xml" ),
    APIOperationStatus.getOkStatus()
);
```

It is also possible to register a URL instead of a File.

Additionally, you may set an initial workflow ID. If the tool starts for the very first time, or if the user had not selected a workflow before, the workflow with the given ID will be displayed when the user opens the **Workflows** view or when EB tresos Studio is restarted.

```
Workflows.getInstance().setInitialWorkflowId( "CreateDemoProject" );
```

### 8.3.3.3. Registration via project

It is possible to register a workflow for a special project. To do this, create a folder named `.workflows` in your project right beneath the `config` folder. Then create an `.xml` file with the workflow description. To register the workflow file, it is necessary to either close and reopen your project or the **Workflows** view.



The **Workflow selection** dialog shows the commonly registered workflows, which are available for all projects. If you select an open project in the **Project Explorer** and there are project-specific workflows available in the project, the dialog additionally shows these workflows.

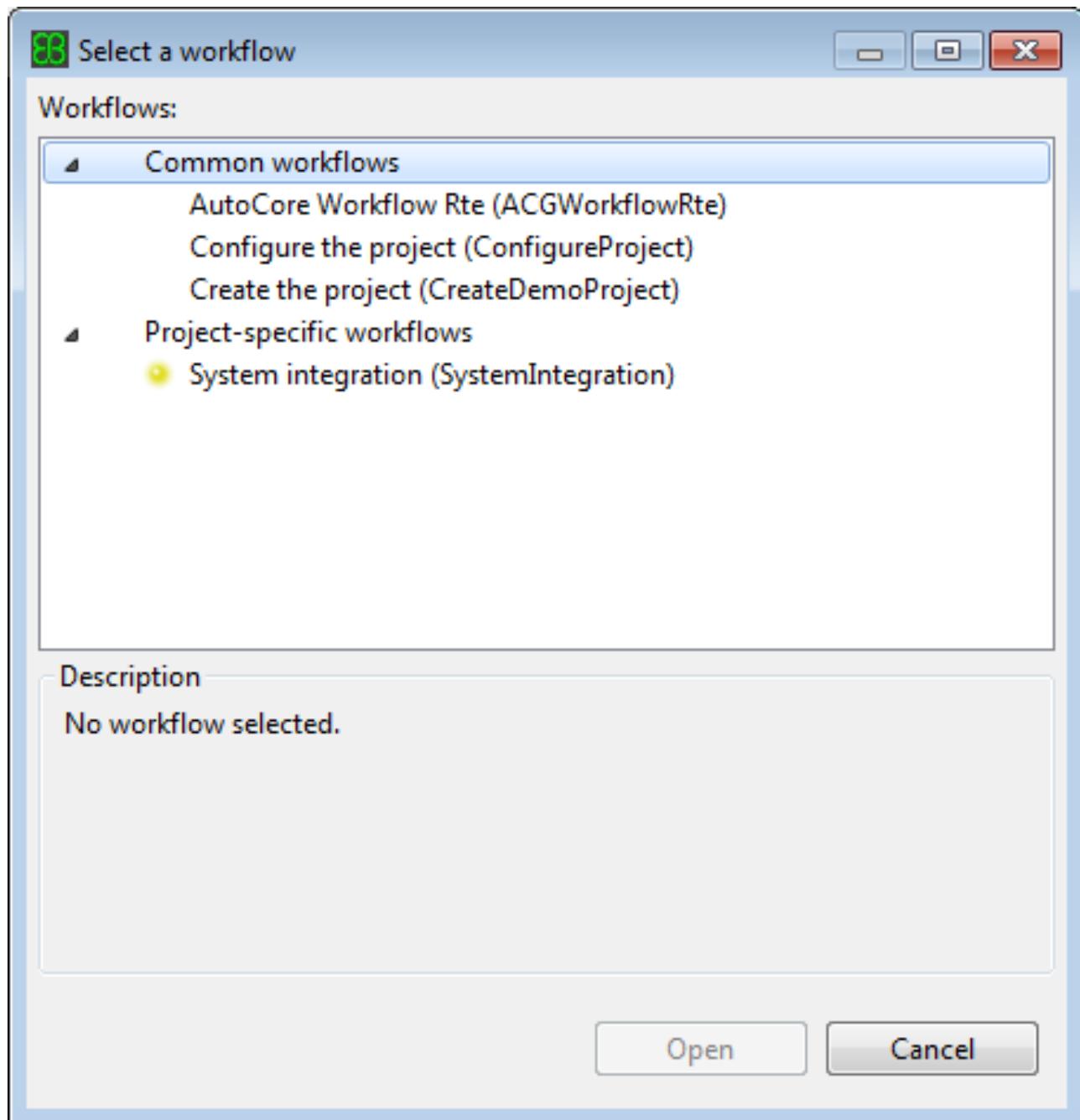


Figure 8.54. **Workflow selection** dialog

**NOTE****Availability of project-specific workflows**

A project-specific workflow is only available if you select the project from where it is registered. Accordingly for all other projects in your workspace the workflow will not be visible. If you need this workflow for another project too, copy the respective workflow description .xml file below a .workflows folder in the other project.

**WARNING****Project-specific workflow IDs may not collide with IDs of common workflows**

The ID of a project-specific workflow may not exist within the commonly registered workflows. Else the registration will fail and an error is logged in the **Error Log** view. However, it is possible to register the same project-specific workflow description in several different projects.

## 8.3.4. API

Find the exact syntax of the methods in the Java documentation. For further information see [Section 3.3, “Public Java API documentation”](#).

## 8.3.5. Commands

Each *action* step can have up to two commands. A command can be added by using the serialized form of the command in the workflow description:

```
commandId [ "(" paramId "=" paramValue
           [ "," paramId "=" paramValue ]* ")" ]
```

As the command framework is part of Eclipse, all commands provided by the Eclipse framework may be used. This means: All commands registered via the extension point `org.eclipse.ui.commands` are available. For further information on the command extension point, see the [Eclipse help](#).

Additionally, EB tresos Studio provides some specific commands. These commands are described in the following sections.

Each EB tresos Studio command depends on the current selection. A loaded project is the base for all EB tresos Studio specific commands. If the command is not available for the current selection, it is marked with the unavailable icon .



### 8.3.5.1. Config Editor Command

You may use the **Config Editor Command** to open configuration editors of the selected project. If the selected and loaded project does not match the given parameters, the action step is marked as unavailable.

commandId

```
dreisoft.tresos.launcher2.api.plugin.EditConfigurationCommand
```

Parameters of the **Config Editor Command**:

ParameterId	Description
moduleId	The moduleId parameter allows the user to supply a module ID. If this parameter is set, the command shows all editors of the module with the given moduleId. The moduleId is supplied by the attribute <code>id</code> of the module extension-point (see <a href="#">Section 5.3.7.1, “dreisoft.tresos.launcher2.plugin.module”</a> ).
moduleType	The moduleType parameter allows the user to supply a module type. If this parameter is set, the command shows all editors for all modules of that type. The type of a module is supplied when the module is registered via the attribute "categoryType" of the module extension-point (see <a href="#">Section 5.3.7.1, “dreisoft.tresos.launcher2.plugin.module”</a> ).

You need to specify at least one parameter.

If several editors are available, performing this command will open a dialog in which the user may select one of the configuration editors.

### 8.3.5.2. Generator Command

You can use the **Generator Command** to run code generators for an ECU configuration project. If the selected and loaded project does not provide a code generator for the given mode, the action step is marked as unavailable.

commandId

```
dreisoft.tresos.launcher2.api.plugin.GeneratorCommand
```

Parameters of Generator Command:

ParameterId	Description
mode	The generator command provides the optional command parameter "mode" that allows to select an alternative generator mode. If the parameter is omitted the standard mode "generate" will be used.

### 8.3.5.3. Importer/Exporter Command

You can use the **Importer/Exporter Command** to open the importer/exporter dialog for a selected project.



commandId

```
dreisoft.tresos.launcher2.api.plugin.ImporterExporterCommand
```

#### 8.3.5.4. Module Wizard Command

You can use the **Module Wizard Command** to open the **Module Configurations** dialog for a selected project.

commandId

```
dreisoft.tresos.launcher2.api.plugin.ModuleConfigurationDialogCommand
```

#### 8.3.5.5. Sidebar Trigger Command

You can use the **Sidebar Trigger Command** to open an assistance dialog that is registered by a sidebar trigger. The assistance dialog will operate on the currently selected project. If the selected and loaded project does not match the given parameters, the action step is marked as unavailable. If an assistance dialog is already open for the selected project, the step is marked as unavailable, too.

commandId

```
dreisoft.tresos.guidedconfig.api.plugin.SidebarTriggerCommand
```

Parameters of the Sidebar trigger command:

ParameterId	Description
triggerId	The Sidebar trigger command provides the optional command parameter <code>triggerId</code> that allows to select the sidebar trigger id of the assistance dialog to run.
triggerType	The Sidebar trigger command provides the optional command parameter <code>triggerType</code> that allows to select the sidebar trigger type of the assistance dialog to run. The sidebar trigger type is a new optional attribute for the trigger registration. Have a look at the description of extension point <code>dreisoft.tresos.guidedconfig.api.plugin.trigger</code> .

It is necessary to specify at least one parameter.

It is possible that more than one trigger matches the `triggerType` parameter for the currently selected project. In this case, performing the **Sidebar Trigger Command** will open a dialog that allows to select one of the registered triggers.

#### 8.3.5.6. Autoconfigure Dialog Command

You can use the **Autoconfigure Dialog Command** to open the **Unattended Wizards** configuration dialog. If the selected and loaded project does not provide assistance dialogs for the given parameters, the action step



is marked as unavailable. If an assistance dialog is already open for the selected project, the step is marked as unavailable, too.

commandId

```
dreisoft.tresos.guidedconfig.api.plugin.AutoConfigureDialogCommand
```

Parameters of the autoconfigure dialog command

ParameterId	Description
triggerId	The <b>Autoconfigure Dialog Command</b> provides the optional command parameter triggerId that allows to select the unattended wizard trigger ID.
triggerType	The <b>Autoconfigure Dialog Command</b> provides the optional command parameter triggerType that allows to select unattended wizard triggers of a given type. The sidebar trigger type is a new optional attribute for the trigger registration. Have a look at the description of extension point <code>dreisoft.tresos.guidedconfig.api.plugin.trigger</code> .

It is mandatory to specify at least one of the parameters triggerId or triggerType.

The **Autoconfigure Dialog Command** opens the **Unattended Wizards** configuration dialog with the triggers for the currently selected configuration project. These triggers are filtered according to the specified parameters.

### 8.3.5.7. Autoconfigure Command

You can use the **Autoconfigure Command** to run an AutoConfigure wizard of the Guided Configuration framework in a way that the user does not have to open the wizard's GUI. If the selected and loaded project does not match the given parameters, the step is marked as unavailable. If an assistance dialog is already open for the selected project, the step is marked as unavailable, too.

commandId

```
dreisoft.tresos.guidedconfig.api.plugin.AutoConfigureTriggerCommand
```

Parameters of the autoconfigure command

ParameterId	Description
triggerId	The <b>Autoconfigure Command</b> provides the optional command parameter triggerId that allows to select the autoconfigure trigger id of the assistance dialog to run.
triggerType	The <b>Autoconfigure Command</b> provides the optional command parameter triggerType with which you may select the trigger type that causes the assistance dialog to run. For further information have a look at the description of extension point <code>dreisoft.tresos.guidedconfig.api.plugin.trigger</code> .

If several triggers are available when you select the triggers with the triggerType parameter, performing this command opens a dialog in which you may select one of the registered triggers.



## 8.3.6. Demos

### 8.3.6.1. Setting up the demo plug-in

This chapter describes the workflow demo. To use the demo, follow the instructions below.

By default no Java code is necessary for using workflows. Therefore the plug-in *WorkflowDemo1* does not contain a single line of code. The plug-in just contains some workflow description files and registers the workflows in the `plugin.xml`.

But to demonstrate the whole functional range of the Workflows API, this demo uses features of the demo plug-in *GuidedConfigDemo2*. The *GuidedConfigDemo2* demo plug-in provides a module configuration, some assistance dialogs and one Generic Configuration Editor.

You may open the workflow without using the plug-in *GuidedConfigDemo2*. In this case you do not need to set up Eclipse to use Java code. The registered workflows of the *WorkflowDemo1* plug-in can be opened, but not all actions of the workflow can be performed: The steps before reaching the step *Inspect the changes* can be performed, but the mentioned example settings (target/derivate, module plugin etc) won't be available to finish the steps.

All demo plug-ins are available at: <tresos-install-loc>/demos/Studio/.

To follow the example described below, you need the following demo plug-ins with their bundle-names:

#### WorkflowDemo1:

The *WorkflowDemo1* plug-in provides the workflow descriptions and the registration.

To use the workflows, copy the folder *WorkflowDemo1* from <tresos-install-loc>/demos/Studio/ to <tresos-install-loc>/plugins/.

#### GuidedConfigDemo2:

The *GuidedConfigDemo2* plug-in provides a module configuration, an assistance dialog and a Generic Configuration Editor.

Follow the instructions in [Section 4.9, “How to install a Public API demo plugin”](#) to install this demo plug-in.

If the **Workflows** view is not visible, follow the instructions in [Section 8.3.6.1.1, “Making the Workflows view visible”](#).

Now you are done setting up the demo plug-in.

If you have installed the plug-in *WorkflowDemo1*, the **Select a workflow** dialog displays at least two entries:

- ▶ Configure the project
- ▶ Create the project



To check if you have installed the plug-in *GuidedConfigDemo2* correctly:

- ▶ Select the menu item **EB tresos Details** from the **Help** menu.
- ▶ Switch to the tab **Module Registry**.
- ▶ Check if the following entry is available in the list:

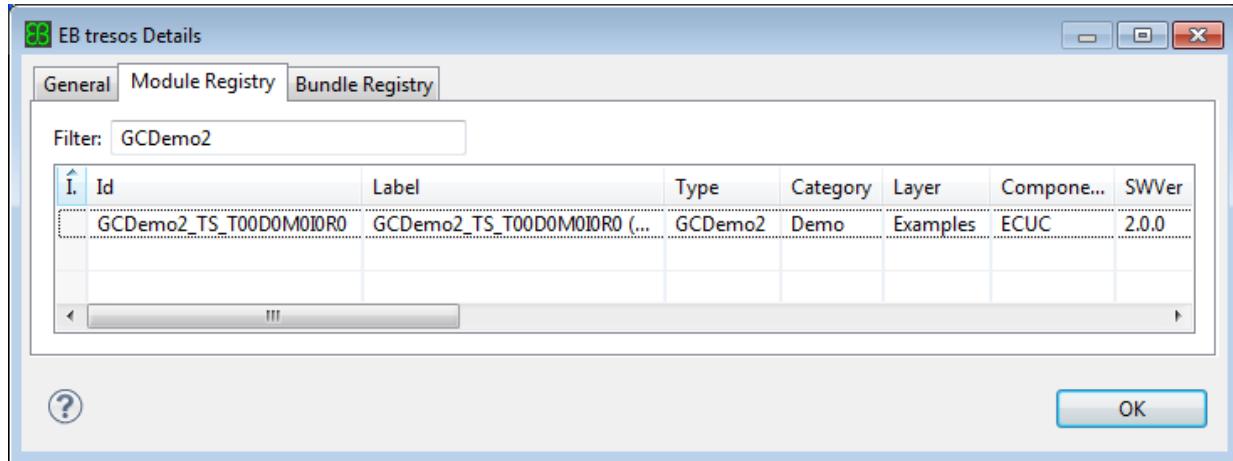


Figure 8.55. Module Registry

### 8.3.6.1.1. Making the Workflows view visible

If the **Workflows** view is not visible in EB tresos Studio, you can make it visible via **Window>Show View/Workflows**.

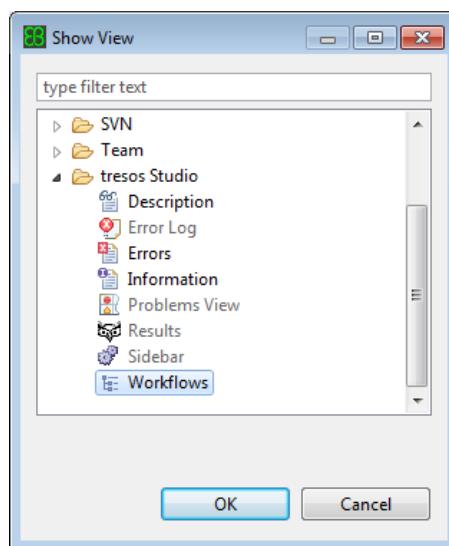


Figure 8.56. Making the **Workflows** view visible



### 8.3.6.2. Demo Overview

The **Workflows** view does not depend on a special target or derivative. Once a plug-in that registers a workflow is installed, this workflow will be available via the **Workflows** view no matter if a project exists or if one is selected.

A workflow action step can contain commands. These commands depend on the current selection and may therefore be marked as unavailable if the command parameters do not match the selection.

In this example you will learn

- ▶ how to write a workflow description file and use workflow hyperlinks
- ▶ how to use the EB tresos Studio specific commands.

### 8.3.6.3. Demo step by step

The workflow demo plug-in registers two workflow files. These workflows are linked together. The first workflow *Create the project* creates the required project, adds a module and runs an importer. The second workflow *Configure the project* configures the project.

In this chapter both workflows are explained step by step.

To start the demo:

- ▶ Click the **View** button  in the tool bar of the **Workflows** view.

The following menu opens up:

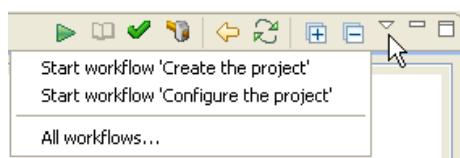


Figure 8.57. WorkflowDemo1: the **workflow selection** menu

- ▶ Select **All workflows....**

The **Select a workflow** dialog opens up.

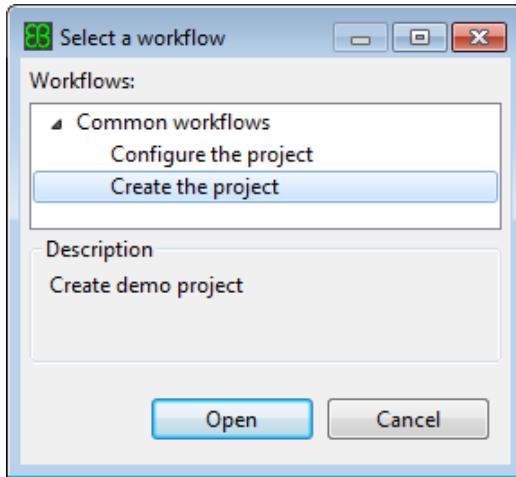


Figure 8.58. WorkflowDemo1: the **Select a workflow** dialog

- Select **Create the project**.

The **Create the project** workflow opens up in the **Workflows** view.

Proceed to [Section 8.3.6.3.1, “Workflow \*Create the project\*”](#) for information on the **Create the project** workflow.

#### 8.3.6.3.1. Workflow *Create the project*

---

**TIP**



**Enable developer features to see the step IDs**

To see the step and workflow IDs in the **Workflows** view additionally to the label, select **Show action 'Reload workflows' and show workflow IDs in Workflows view** in the **Preferences**. For instructions about how to change preferences, see the EB tresos Studio user's guide, chapter [Setting Developer features preferences](#).

---

After expanding the workflow tree, the **Workflows** view looks like this:

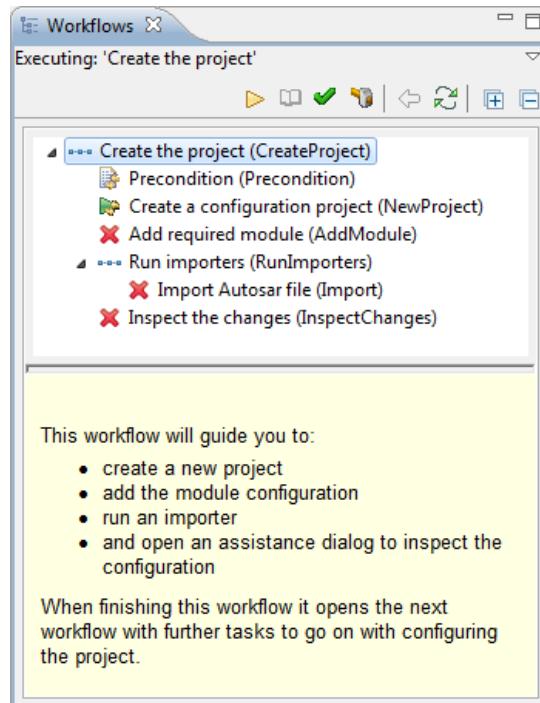


Figure 8.59. WorkflowDemo1: Expanded workflow

To navigate to the first action step, click the **Jump** button in the tool bar.

#### 8.3.6.3.1.1. Precondition

The *Precondition* step provides a textual description of what is necessary to run the workflow demo. The instruction shown in the description area is written in HTML.

This step is marked as repeatable step and is not finished after performing it.

Performing this step by pressing the **Run** button in the tool bar opens a message dialog which asks if you want to navigate to the next step *Create a configuration project*.

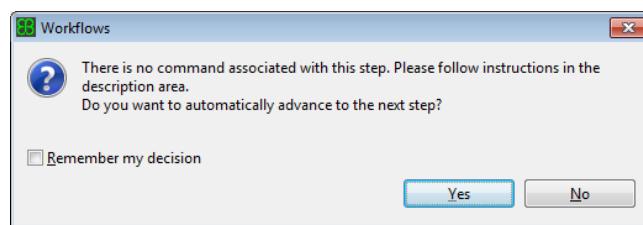


Figure 8.60. Automatically advance?

**TIP****Remember my decision**

If you do not want to get asked again, select the check box **Remember my decision** and answer the question by clicking either Yes or No. The next time this message appears no longer.

To enable the message dialog again, change the setting in the EB tresos Studio preferences below **EB tresos Studio/Assistance dialogs** in the **Workflows** section.

### 8.3.6.3.1.1.1. Create a configuration project

The step *Create a configuration project* has an associated Eclipse command which is currently available, as the icon ➤ shows. All other steps are currently not available ✘, as they are EB tresos Studio specific and no loaded project is selected.

If you double-click the tree item or if you use the **Run** tool bar button ➤, the new project wizard opens up to create the required project. Follow the instructions in the description area and enter the project name, select a target/derivative and release version and click **Finish**.

After project creation all steps except the last one are now available, as the new project is selected and loaded. The last step *Inspect the changes* is still not available, as the underlying command would open an assistance dialog which is currently not available for the selected project.

In the workflow XML file, this command is marked with the attribute *autoadvance* set to 'false'. Therefore the next step is not selected automatically after performing the action **Create a configuration project**.

As the step is marked as repeatable ↗ the step is not set finished automatically. To finish the action step **Create a configuration project**, click the **Finish** button ✓ in the tool bar.

Select the next step *Add required module*.

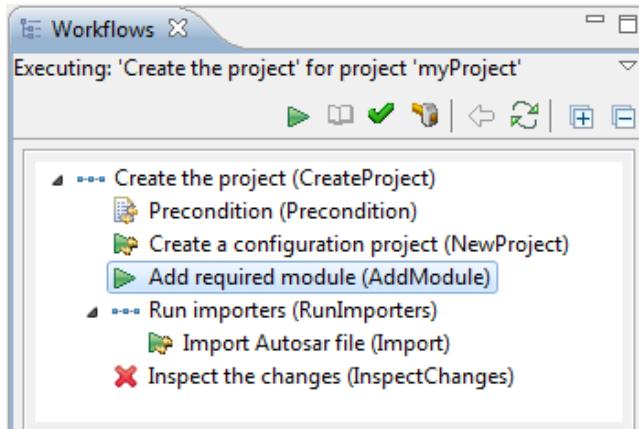


Figure 8.61. WorkflowDemo1: Workflow after project creation



### 8.3.6.3.1.1.2. Add required module

This step opens the **Module Configurations** dialog and allows to select the module `GCDemo2_TS-T00D0M0I0R0`.

The underlying commandId for this action is: `dreisoft.tresos.launcher2.api.plugin.ModuleConfigurationDialogCommand`.

After you have added the required module and left the dialog by clicking **OK**, the last step *Inspect the changes* is now available and the workflow looks like this:

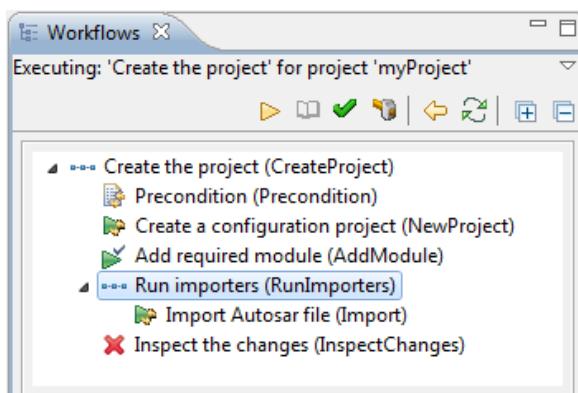


Figure 8.62. WorkflowDemo1: Workflow after adding the module

### 8.3.6.3.1.1.3. Import Autosar file

After you have added the module configuration, the group step *Run importers* is selected. To select the next action step, click the **Jump** tool bar button ➤.

To perform the action step *Import Autosar file*, click the **Run** tool bar button again ➡. This command opens the **Importer/Exporter** dialog. Now you can add and run an AUTOSAR importer as specified in the **description** area. The file `GCDemo2.epc` can be found in the resources folder of the workflow demo plug-in.

The underlying commandId for this action is: `dreisoft.tresos.launcher2.api.plugin.ImporterExporterCommand`.

After running the importer, the project configuration has been changed. The importer run shall finish sucessfull. The step *Import Autosar file* is not marked as finished after importer run, as this step is marked as repeatable.

A repeatable step can be run more than once. It can be marked as finished by using the **Finish** button ✓ in the tool bar. The next step is selected.

### 8.3.6.3.1.1.4. Inspect the changes

Performing this step opens an assistance dialog with the title *Demo Page*.



The underlying commandId for this action is: dreisoft.tresos.guidedconfig.api.plugin SidebarTriggerCommand.

In addition, the **Workflows** view has changed and the workflow *Configure the project* is displayed. A message occurs to inform the user about the change:

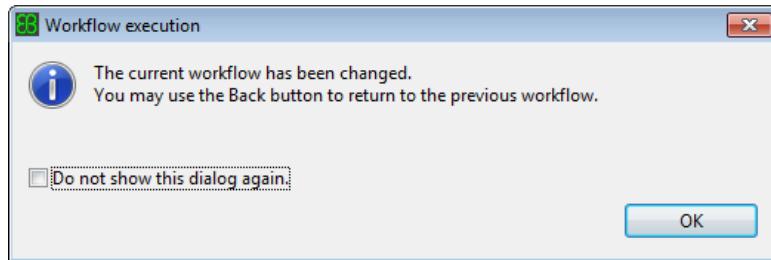


Figure 8.63. WorkflowDemo1: Workflow changed message

The opened assistance dialog shows the configuration data. The parameters are shown as read-only, as they cannot be changed in this assistance dialog. The importer run has changed the value of the *Boolean* parameter.

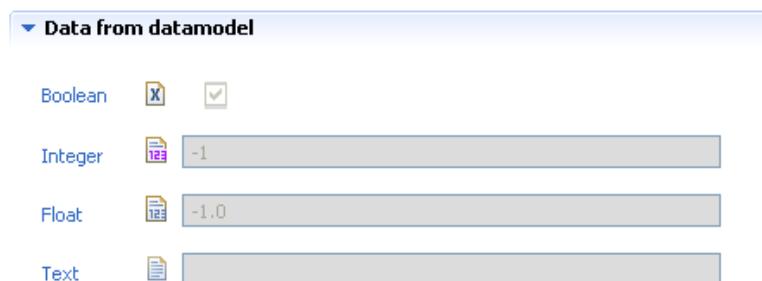


Figure 8.64. WorkflowDemo1: Changed datamodel configuration

### 8.3.6.3.1.2. Workflow *Configure the project*

The second workflow shows how to configure and generate the project.

To navigate to the first action step, click the **Jump** tool bar button ▶.

#### 8.3.6.3.1.2.1. Module configuration

The workflow contains a choice step with the label *Module configuration*. This means that only one sub-step has to be performed to finish the task. If either the step *Calculate Handhelds* or the step *Edit module configuration* is finished, the parent group step is marked as finished, too.



This step shows how to use workflow hyperlinks in the HTML description. When clicking on one of the links, the corresponding step of the workflow is selected and the description area shows the description of the new step.

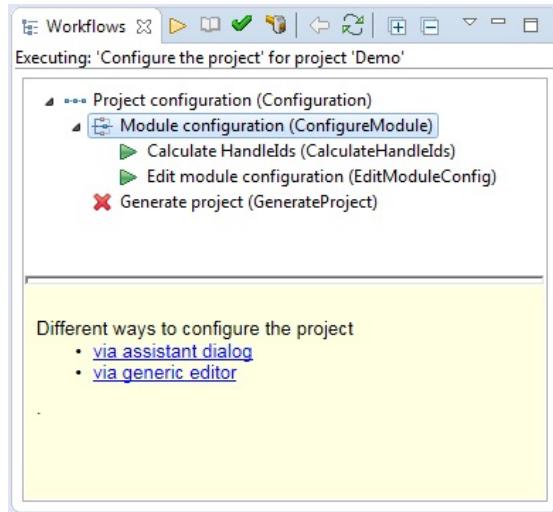


Figure 8.65. WorkflowDemo1: Step with workflow hyperlinks in the HTML description

### 8.3.6.3.1.2.2. Calculate HandleIds

The step *Calculate HandleIds* is not available, as the assistance dialog *Demo Page* is still open and locks the current project. After closing the assistance dialog, the action step is available again.

In the description area an image is shown. The link to the images is resolved relative to the location of the workflow file.

This step has two commands, one configure command and one default command.

```
commandId (configure)
    dreisoft.tresos.guidedconfig.api.plugin.AutoConfigureDialogCommand

commandId (default)
    dreisoft.tresos.guidedconfig.api.plugin.AutoConfigureTriggerCommand
```

The **configure** command opens the **Unattended Wizards** configuration dialog to configure a wizard which then can be run unattendedly.

The default command then runs the configured wizard unattended. The results are shown in the **Results** view afterwards.

### 8.3.6.3.1.2.3. Edit module configuration

This command opens the Generic Module Editor to configure the module configuration.



The underlying commandId for this action is: `dreisoft.tresos.launcher2.api.plugin.EditConfigurationCommand`

Now you can change the configuration of the project.

If you perform either the step *Calculate Handhelds* or the step *Edit module configuration*, the parent group step is marked as finished and the next step *Generate project* is selected.

#### 8.3.6.3.1.2.4. Generate project

The generator command starts the code generation in the given mode. If no mode is given, the default mode `generate` is used.

In this demo the step *Generate project* is not available as the project does not provide a generation mode named `generate_SWC-T`, which is specified in the command parameter.

The underlying commandId for this action is: `dreisoft.tresos.launcher2.api.plugin.GeneratorCommand`.

# Bibliography

- [1] AUTOSAR *Layered Software Architecture*, Issue Version 2.1.0, Revision 0014, Publisher: [www.autosar.org](http://www.autosar.org)
- [2] OSI *Reference Model - The ISO Model of Architecture for Open Systems Interconnection*, Author: Zimmermann, Pages: 425 to 432,
- [3] AUTOSAR - *Specification of ECU Configuration*, Issue Document number 087. Document version 2.0.1, Revision 0002, Part of release 3.0, Publisher: [www.autosar.org](http://www.autosar.org)
- [4] AUTOSAR - *Specification of ECU Configuration*, Issue Document number 087. Document version 3.2.0, Revision 3, Part of release 4.0, Publisher: [www.autosar.org](http://www.autosar.org)
- [5] AUTOSAR - *Specification of the Virtual Functional Bus*, Publisher: [www.autosar.org](http://www.autosar.org)
- [6] AUTOSAR - *Specification of the Software Component Template*, Publisher: [www.autosar.org](http://www.autosar.org)
- [7] AUTOSAR *Model Persistence Rules for XML*, Issue Version 2.1.2, Release 3.0, Revision 0001, Publisher: [www.autosar.org](http://www.autosar.org)
- [8] LIN *Specification Package*, URL: <http://www.lin-subbus.org/>, Publisher: Lin Consortium



# Appendix A. Third party license

This product includes third-party components that require the following notices. For respective license terms refer to the subfolder /licenses

- ▶ This product includes Java(TM) 2 Runtime Environment 1.8.0\_121.

Java(TM) 2 Runtime Environment 1.8.0\_121 is licensed under the Oracle Corporation Binary Code License Agreement. For additional Licenses Terms according to Section G of the Oracle Corporation Binary Code License Agreement please refer to "jre/THIRDPARTYLICENSEREADME.txt".

- ▶ The product is based in part on software developed for eclipse.org licensed under the EPL 1.0. (Eclipse runtime environment 4.6.2)

Source code for these components is available from <http://subclipse.tigris.org/> and [www.eclipse.org](http://www.eclipse.org/), under the Eclipse Public License ("EPL", see <http://www.eclipse.org/org/documents/epl-v10.php>).

As a requirement of the EPL, Elektrobit hereby:

(1) disclaims on behalf of all Eclipse.org Contributors all warranties and conditions, express and implied, including warranties or conditions of title and non-infringement, and implied warranties or conditions of merchantability and fitness for a particular purpose;

(2) excludes on behalf of all Contributors all liability for damages, including direct, indirect, special, incidental and consequential damages, such as lost profits; and

(3) states that any provisions which differ from the Eclipse Public License are offered by Elektrobit alone and not by any other party.

- ▶ The product is based in part on software developed for eclipse.org licensed under the EPL 1.0. (Eclipse Modeling Framework 2.8.0)

Source code for these components is available from <http://subclipse.tigris.org/> and [www.eclipse.org](http://www.eclipse.org/), under the Eclipse Public License ("EPL", see <http://www.eclipse.org/org/documents/epl-v10.php>).

As a requirement of the EPL, Elektrobit hereby:

(1) disclaims on behalf of all Eclipse.org Contributors all warranties and conditions, express and implied, including warranties or conditions of title and non-infringement, and implied warranties or conditions of merchantability and fitness for a particular purpose;

(2) excludes on behalf of all Contributors all liability for damages, including direct, indirect, special, incidental and consequential damages, such as lost profits; and

(3) states that any provisions which differ from the Eclipse Public License are offered by Elektrobit alone and not by any other party.



- ▶ This product includes jaxen 1.1-beta-7.

Copyright 2003-2006 The Werken Company. All Rights Reserved.

- ▶ This product includes commons-jxpath.Apache Commons.

JXPath Copyright 2001-2008 The Apache Software Foundation.

This product includes software developed by The Apache Software Foundation (<http://www.apache.org/>).

- ▶ This product includes commons-jxpath.Apache Commons.

JXPath Copyright 2001-2015 The Apache Software Foundation.

This product includes software developed by The Apache Software Foundation (<http://www.apache.org/>).

- ▶ This product includes java cup 0.10k.

Copyright 1996 by Scott Hudson, Frank Flannery, C. Scott Ananian

- ▶ This product includes log4j 1.2.7.

This product includes software developed by the Apache Software Foundation (<http://www.apache.org/>).

Copyright (C) The Apache Software Foundation. All rights reserved.

- ▶ This product includes software developed by the JDOM Project (<http://www.jdom.org/>).

Copyright (C) 2000-2004 Jason Hunter & Brett McLaughlin. All rights reserved.

- ▶ This product includes poi 2.5.1.

This product includes software developed by The Apache Software Foundation (<http://www.apache.org/>).

- ▶ This product includes sqlite-jdbc 3.6.19.

sqlite-jdbc Copyright 2013 The Apache Software Foundation.

This product includes software developed by The Apache Software Foundation (<http://www.apache.org/>).

- ▶ This product includes Xerces-J 2.8.0.

This product includes software developed by The Apache Software Foundation (<http://www.apache.org/>).

Apache Xerces Java, Copyright 1999-2007 The Apache Software Foundation

Portions of this software were originally based on the following:

- software copyright (c) 1999, IBM Corporation., <http://www.ibm.com>.

- software copyright (c) 1999, Sun Microsystems., <http://www.sun.com>.

- voluntary contributions made by Paul Eng on behalf of the Apache Software Foundation that were originally developed at iClick, Inc., software copyright (c) 1999.

- ▶ This product includes Xerces 3.1.1

This product includes software developed by The Apache Software Foundation (<http://www.apache.org/>).

Portions of this software were originally based on the following:

- software copyright (c) 1999, IBM Corporation., <http://www.ibm.com>.

- ▶ This product includes trang version 20091111.

Copyright (c) 2002, 2003, 2008 Thai Open Source Software Center Ltd

- ▶ The product is based in part on software developed for [eclipse.org](http://eclipse.org) licensed under the EPL 1.0. (subclipse 1.8.18)

Source code for these components is available from <http://subclipse.tigris.org/> and [www.eclipse.org](http://www.eclipse.org), under the Eclipse Public License ("EPL", see <http://www.eclipse.org/org/documents/epl-v10.php>).

As a requirement of the EPL, Elektrobit hereby:

(1) disclaims on behalf of all Eclipse.org Contributors all warranties and conditions, express and implied, including warranties or conditions of title and non-infringement, and implied warranties or conditions of merchantability and fitness for a particular purpose;

(2) excludes on behalf of all Contributors all liability for damages, including direct, indirect, special, incidental and consequential damages, such as lost profits; and

(3) states that any provisions which differ from the Eclipse Public License are offered by Elektrobit alone and not by any other party.

- ▶ The product is based in part on software developed for [eclipse.org](http://eclipse.org) licensed under the EPL 1.0. (collabnet merge 2.1)

Source code for these components is available from <http://subclipse.tigris.org/> and [www.eclipse.org](http://www.eclipse.org), under the Eclipse Public License ("EPL", see <http://www.eclipse.org/org/documents/epl-v10.php>).

As a requirement of the EPL, Elektrobit hereby:

(1) disclaims on behalf of all Eclipse.org Contributors all warranties and conditions, express and implied, including warranties or conditions of title and non-infringement, and implied warranties or conditions of merchantability and fitness for a particular purpose;

(2) excludes on behalf of all Contributors all liability for damages, including direct, indirect, special, incidental and consequential damages, such as lost profits; and

(3) states that any provisions which differ from the Eclipse Public License are offered by Elektrobit alone and not by any other party.

- ▶ This product includes javaHL 1.7.8.



This product includes software developed by CollabNet (<http://www.Collab.Net/>). Copyright (c) 2000-2007 CollabNet. All rights reserved.

- ▶ This product includes software developed by Computing Services at Carnegie Mellon University (<http://www.cmu.edu/computing/>).
- ▶ This product includes jsch 0.1.53.

Copyright (c) 2002-2012 Atsuhiko Yamanaka, JCraft, Inc.

- ▶ This product includes antlr 3.5.2.

Copyright (c) 2013 Terence Parr All rights reserved.

- ▶ This product includes antlr 4.7.

Copyright (c) 2012 Terence Parr and Sam Harwell. All rights reserved.

- ▶ This product includes ICU4J 56.1.0

Copyright (c) 1995-2011 International Business Machines Corporation and others

ICU4J contains further third party products. icu\_56.1.0\_license.txt contains all additional licenses.

- ▶ This product includes Google Guava 20.0

This product includes software developed by Google Inc. (<http://www.google.com/>).

Portions of this software were originally based on the following:

- software copyright (c) 2010-2016, Google Inc., <http://www.google.com>.

- ▶ This product includes Apache Ant.

Apache Ant Copyright 1999-2017 The Apache Software Foundation

This product includes software developed by The Apache Software Foundation (<http://www.apache.org/>).

- ▶ This product includes Apache Batik.

Apache Batik Copyright 1999-2017 The Apache Software Foundation

This product includes software developed by The Apache Software Foundation (<http://www.apache.org/>).

Batik includes the following software:

DOM

DOM is developed by the World Wide Web Consortium. Your use of DOM is subject to the terms and conditions of the license found in the file LICENSE.dom-software.txt which is included with this product and can also be found at <http://www.w3.org/Consortium/Legal/copyright-software-19980720>.



## SAX

SAX is developed by the SAX project (<http://www.saxproject.org>). Your use of SAX is subject to the terms and conditions of the license found in the file LICENSE.sax.txt which is included with this product.

- ▶ This product includes Javax EL API.

Copyright © 1996-2015, Oracle and/or its affiliates. All Rights Reserved. Use is subject to license terms.

This product includes software developed by Sun/Oracle as part of the Glassfish project (<https://javaee.github.io/glassfish/>).

- ▶ This product includes javax.annotation.

Copyright © 1996-2015, Oracle and/or its affiliates. All Rights Reserved. Use is subject to license terms.

This product includes software developed by Sun/Oracle as part of the Glassfish project (<https://javaee.github.io/glassfish/>).

- ▶ This product includes the package "javax.inject" from the atinject project

Copyright © 1996-2015, Oracle and/or its affiliates. All Rights Reserved. Use is subject to license terms.

This product includes software developed by Sun/Oracle as part of the Glassfish project (<https://javaee.github.io/glassfish/>).

- ▶ This product includes the package "xml-apis"

Copyright 1999-2017 The Apache Software Foundation

This product includes software developed by The Apache Software Foundation (<http://www.apache.org/>).

- ▶ This product includes Javax Servlet API.

Copyright © 1996-2015, Oracle and/or its affiliates. All Rights Reserved. Use is subject to license terms.

This product includes software developed by Sun/Oracle as part of the Glassfish project (<https://javaee.github.io/glassfish/>).

- ▶ This product includes Javax JSP API.

Copyright © 1996-2015, Oracle and/or its affiliates. All Rights Reserved. Use is subject to license terms.

This product includes software developed by Sun/Oracle as part of the Glassfish project (<https://javaee.github.io/glassfish/>).

- ▶ This product includes Java Server Pages Engine.

Copyright © 1996-2015, Oracle and/or its affiliates. All Rights Reserved. Use is subject to license terms.



This product includes software developed by Sun/Oracle as part of the Glassfish project (<https://javaee.github.io/glassfish/>).

- ▶ This product includes Apache Lucene.

Copyright 1999-2017 The Apache Software Foundation.

This product includes software developed by The Apache Software Foundation (<http://www.apache.org/>).

- ▶ This product includes SAT4J.

SAT4J includes content that was obtained under licenses that differ from the SAT4J licenses.

is based on code obtained from the Minisat 1.1.4 implementation, the source code for which can be found at [www.minisat.se](http://www.minisat.se), under a permissive license.

MiniSat -- Copyright (c) 2003-2005, Niklas Een, Niklas Sorensson.

- ▶ This product includes CSS SAC 1.3 Java Binding developed by the World Wide Web Consortium ("W3C").

This product includes Batik SVG Toolkit 1.6 (subset) developed by the Apache Software Foundation.

This product includes SAX developed by the SAX project.

# Index

## A

accessing resources  
 RAM/ROM  
   via command line, 232  
   via XPath, 231

Apache Ant  
   code generator support, 263

API  
   Custom Attribute API, 235  
   Custom XPath Functions API, 222  
   DataModel Public API, 126  
   ECU Resource Manager API, 225  
   EPC File Generator API, 267  
   External Generator API, 269  
   Guided Configuration API, 321  
   introduction to all APIs, 25  
   Module Transformer API, 126  
   NG Generator API, 261  
   Public API Java Documentation, 126  
   Public Generator API, 259  
   Public Java API, 25  
   Resources API, 233  
   System Model Access API, 234  
   Template-based Code Generator API, 242  
   Workflow API, 368  
   XPath API, 176

APIInvalidOperationException  
   XPath function, 223

APIOperationStatus  
   changing, 284

application example (see demo)

AUTOSAR

- 2.0
  - StMD, 106
- 2.1
  - data node tag, 85
  - StMD, 106
- 3.0
  - data node tag, 85

StMD, 106  
 3.1
 

- StMD, 106
- converting modules, 233
- file format
  - converting to XDM, 107
- mapping nodes
  - to DataModel nodes, 81
- upgrading to 3.x, 121

## B

bash shell (see command line)  
 Boolean values  
   layout
 

- changing in XDM, 153

 builder  
   NLS
 

- class generator, 290

## C

C Data Structures Generator (CDS)  
   code generator, 262  
 cluster (see module cluster)  
 code generator  
   Apache Ant support, 263  
   C Data Structures Generator (CDS), 262  
   EPC File Generator, 267  
   External Generator API, 269  
   JET, 261  
   NG Generator API, 261  
   Public Generator API, 259  
   Template-based Code Generator API, 242  
 Codetemplate console, 244  
 command line  
   legacy command line
 

- accessing ECU resources, 232

 comment  
   line comment, 245  
   multi-line comment, 245  
 configuration data  
   accessing with XPath, 96  
   representation

- DataModel**, 142
  - configuration extension point, 44
  - console, 173
    - (see also command line)
    - Codetemplate console, 244
    - XPath console, 173
  - container
    - data node tag, 85
  - crypto command
    - singing modules, 124
  - cryptographic key
    - of modules, 124
  - Custom Attribute API, 235
  - Custom XPath Functions API, 222
  - customizing
    - default preferences
      - in Eclipse, 48
- D**
- data node
    - tag for containers, 85
  - data tree
    - of configuration data, 142
  - data types
    - in XPath, 174
  - DataModel
    - accessing and merging, 233
    - configuration data representation, 142
    - DataModel Public API, 126
    - mapping nodes, 81
    - navigating
      - with XPath, 172
    - storage format, 50
    - System Model Access API, 234
  - DCtxt
    - DataModel Public API, 126
    - extending, 233
    - specifying, 233
  - default settings
    - customizing
      - in Eclipse, 48
  - demo
  - code generator, 242
  - GUI extension, 327
  - installing in Eclipse, 49
  - new modules, 107
  - ResourceAPIDemo, 233
  - SystemModelAccessDemo, 235
  - Workflow API, 380
- E**
- Description editor
    - Eclipse, 34
  - docking point
    - registering, 354
  - EB tresos Details dialog, 117
  - EB tresos Studio
    - creating modules, 106
    - installing new modules, 120
    - opening from Eclipse, 31
  - EB tresos Studio GUI (see GUI)
  - Eclipse, 29
    - demo
      - installing, 49
    - Description editor, 34
    - errors and warnings, 41
    - extension point descriptions
      - opening, 34
    - installing, 29
    - opening EB tresos Studio, 31
    - PDE Editor, 34
    - preferences
      - defining, 48
    - prerequisites for EB tresos Studio, 31
    - prerequisites for installing, 29
    - project
      - creating, 37
    - target platform
      - setting up, 31
  - ECU Resource Finder
    - accessing resources, 228
  - ECU Resource Manager
    - registering ECU resources, 225
  - ECU resources

- accessing via command line, 232
- editor
  - Description editor, 34
  - Generic Configuration Editor
    - changing layout, 148
  - PDE Editor, 34
  - registering, 115
- enhancement
  - of the GUI, 290
- enumeration
  - layout
    - changing in XDM, 156
- EPC File Generator API, 267
- errors
  - in Eclipse, 41
- example (see demo)
- exception
  - XPath function
    - APIInvalidOperationException, 223
- extension (see enhancement)
- extension point
  - configuration, 44
  - descriptions
    - opening, 34
  - generator, 44
  - module, 43
- external generator
  - EPC File Generator API, 267
  - External Generator API, 269
    - command line, 273
- F**
  - finding resources
    - RAM/ROM, 228
  - float values
    - layout
      - changing in XDM, 156
  - functional safety
    - suitability, 24
  - functions
    - custom XPath functions
      - implementing, 223
- G**
  - registering, 223
  - XPath functions
    - naming conventions, 174
- I**
  - installing
    - demo, 49
    - EB tresos Studio prerequisites for Eclipse, 31
  - Eclipse, 29

integer values

layout

    changing in XDM, 156

## J

Java Emitter Templates (JET)

    code generator, 261

## K

key

    cryptographic key, 124

    signature key, 124

## L

languages

    adapting GUI, 278

layout

    of GUI elements

        changing, 148

legacy command line

    accessing ECU resources, 232

line comment, 245

localization

    of the GUI, 278

## M

MANIFEST.MF

    adapting, 108

mapping

    AUTOSAR nodes

        to DataModel nodes, 81

MCC

    multiple configuration container, 139

messages

    message status

        changing language, 284

    severity level, 284

model-view-control pattern

    what is this?, 291

module

    creating in Eclipse, 37

    demo, 107

EB tresos Studio module

    creating, 106

    installing, 120

extension point, 43

layout

    changing in plugin.xml, 151

module cluster

    assigning, 111

module transformer

    registering, 126

Module Transformer API, 126

registering, 108

signing, 124

StMD, 106

VsMD, 106

module transformer

    chaining, 129

    implementing, 128

multi-line comment, 245

## N

naming conventions

    XPath functions, 174

National Language Support (NLS) (see NLS/OS)

NG Generator API, 261

NLS/OS builder

    commandline

        class generator, 290

node, 142

    (see also tree)

    addressing via XPath, 243

## O

Operation Status (OS), 283

OS

    properties file

        generating, 290

Outline view

    opening XPath Console, 173

## P

parameters



- layout
  - changing in XDM, 156
- plugin
  - creating in Eclipse , 37
- Plugin Development Environment (PDE)
  - Eclipse, 29
  - errors and warnings, 41
  - PDE Editor, 34
- plugin.xml
  - adapting, 108
  - changing module layout, 151
  - custom XPath functions
    - implementing, 223
    - registering, 223
  - finding resources, 228
  - registering resources, 225
- plugins directory
  - of EB tresos Studio, 120
- post-build
  - post-build loadable configuration support, 139
- postBuildChangeable
  - post-build changeable containers, 139
- preferences
  - defining in Eclipse, 48
- project
  - creating in Eclipse , 37
- properties file
  - default preferences
    - defining, 48
- ECU resources
  - finding, 228
  - registering, 225
- NLS
  - generating, 290
- Public API demo plugin
  - installing in Eclipse, 49
- Public API Java Documentation, 126
- Public Generator API, 259
  - implementing, 260
  - registering, 259
- Public Java API
  - what is this?, 25
- public static methods
  - XPath expression
    - accessing, 223
- R**
- RAM
  - registering resources, 225
- RCtxt
  - specifying, 233
- reference
  - layout
    - changing in XDM, 156
- release version
  - adding to plugin.xml, 117
- resources
  - RAM/ROM
    - finding, 228
    - registering, 225
  - Resource API
    - demo, 233
  - resource properties file
    - accessing via command line , 232
    - accessing via XPath , 231
  - Resources API, 233
- ROM
  - registering resources, 225
- S**
- safe use of
  - EB tresos Studio, 24
- schema tree
  - node mapping
    - in XDM, 81
  - of configuration data, 142
- Search dialog
  - XPath Console results, 174
- severity levels
  - warnings and messages, 284
- shell (see command line)
- signature key
  - what to do with it?, 124
- software version

adding to plugin.xml, 117  
specification version  
    adding to plugin.xml, 117  
standardized module definition (StMD)  
    AUTOSAR 2.0, 2.1, 3.0, 3.1, 106  
string values  
    layout  
        changing in XDM, 156  
System Model Access API, 234

## T

Tab  
    split assign, 172  
target platform  
    setting up in Eclipse, 31  
Template-based Code Generator API, 242  
Third-party licenses, 391  
transformer (see module transformer)  
tree  
    data tree  
        configuration data, 142  
    schema tree  
        configuration data, 142  
types of data (see data types)

## U

update (see upgrade)  
upgrade  
    AUTOSAR modules, 121

## V

variant handling, 129  
vendor-specific module definition (VsMD)  
    creating, 107  
    what to do with it?, 42  
version  
    release version, 117  
    software version, 117  
    specification version, 117

## VSMD

vsmdcheck commandline, 140

## W

warnings  
    in Eclipse, 41  
    severity levels, 284  
wizard  
    registering, 352  
Workflow API, 368  
    demos, 380  
Workflows view  
    adding content, 368

## X

XDM  
    changing Boolean values layout, 153  
    changing parameter layout, 156  
    changing the EB tresos Studio GUI, 148  
    converting to, 107  
    creating new modules, 42  
    mapping nodes, 81  
    what is this?, 50  
XPath  
    accessing  
        resources, 231  
    addressing  
        configuration data, 96  
        nodes, 243  
    API, 176  
    custom XPath functions  
        implementing, 223  
        registering, 222  
    data types, 174  
    XPath Console  
        opening, 173  
        results, 174  
    XPath functions  
        naming conventions, 174