



Elektrobit

# EB GUIDE Studio

User guide

Version 6.10.0.200602120856



Elektrobit Automotive GmbH  
Am Wolfsmantel 46  
D-91058 Erlangen  
GERMANY

Phone: +49 9131 7701-0  
Fax: +49 9131 7701-6333  
<http://www.elektrobit.com>

## Legal notice

Confidential and proprietary 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 2020, Elektrobit Automotive GmbH.

# Table of Contents

1. About this documentation .....	18
1.1. Target audiences of the user documentation .....	18
1.1.1. Target audience: Modelers .....	18
1.1.2. Target audience: Extension developers .....	19
1.2. Structure of user documentation .....	19
1.3. Typography and style conventions .....	20
1.4. Naming conventions .....	21
1.5. Path conventions .....	22
2. Safe and correct use .....	23
2.1. Intended use .....	23
2.2. Possible misuse .....	23
3. Support .....	24
4. Introduction to EB GUIDE .....	25
4.1. The EB GUIDE product line .....	25
4.2. EB GUIDE Studio .....	25
4.2.1. Modeling HMI behavior .....	25
4.2.2. Modeling HMI appearance .....	26
4.2.3. Handling data .....	26
4.2.4. Simulating the EB GUIDE model .....	26
4.2.5. Exporting the EB GUIDE model .....	27
4.3. EB GUIDE TF .....	27
4.3.1. Modularization .....	27
4.4. EB GUIDE aware .....	28
5. Getting started .....	29
5.1. Starting EB GUIDE .....	29
5.2. Creating a project .....	30
5.3. Modeling HMI behavior .....	31
5.4. Modeling HMI appearance .....	34
5.5. Starting the simulation .....	36
6. Background information .....	38
6.1. 3D graphics .....	38
6.1.1. Supported 3D graphic formats .....	38
6.1.2. Settings for 3D graphic files .....	38
6.1.3. Import of a 3D graphic file .....	39
6.2. Animations .....	41
6.2.1. Execution of animations .....	41
6.2.2. Widget animation .....	42
6.2.3. Datapool item animation .....	42
6.2.4. View transition animation .....	42

6.2.5. Color animation .....	44
6.2.6. Script curve .....	44
6.3. Anti-aliasing .....	44
6.4. Application programming interface between application and model .....	45
6.5. Communication context .....	45
6.6. Components of the graphical user interface .....	45
6.6.1. Graphical user interface of EB GUIDE Studio .....	45
6.6.1.1. Project center .....	46
6.6.1.1.1. Navigation area .....	46
6.6.1.1.2. Content area .....	47
6.6.1.2. Project editor .....	47
6.6.1.2.1. Navigation component .....	48
6.6.1.2.2. <b>Outline</b> component .....	49
6.6.1.2.3. Toolbox component .....	50
6.6.1.2.4. Properties component .....	50
6.6.1.2.5. Content area .....	51
6.6.1.2.6. <b>Events</b> component .....	53
6.6.1.2.7. <b>Datapool</b> component .....	53
6.6.1.2.8. <b>Assets</b> component .....	54
6.6.1.2.9. <b>Namespaces</b> component .....	54
6.6.1.2.10. Command area .....	54
6.6.1.2.11. Problems component .....	56
6.6.1.2.12. VTA component .....	56
6.6.1.2.13. Templates component .....	56
6.6.2. Graphical user interface of EB GUIDE Monitor .....	56
6.6.3. Dockable components .....	58
6.7. Datapool .....	59
6.7.1. Concept .....	59
6.7.2. Datapool items .....	59
6.7.3. Windowed lists .....	60
6.8. EB GUIDE model and EB GUIDE project .....	60
6.8.1. Storage format .....	61
6.8.2. Export format .....	63
6.8.3. Naming of model elements .....	64
6.8.4. Validation criteria for EB GUIDE project .....	64
6.8.4.1. Validation while opening an EB GUIDE project .....	65
6.8.4.2. Validation using the <b>Problems</b> component .....	65
6.9. EB GUIDE Monitor .....	65
6.10. Event handling .....	66
6.10.1. Event system .....	66
6.10.2. Events .....	66
6.11. Extensions .....	68

6.11.1. EB GUIDE Studio extension .....	68
6.11.2. EB GUIDE GTF extension .....	69
6.11.3. EB GUIDE Monitor extensions .....	69
6.12. Focus handling .....	70
6.13. Gamma-correct rendering .....	71
6.13.1. Concepts .....	71
6.13.2. Gamma correction in EB GUIDE Studio .....	72
6.14. Image-based lighting .....	73
6.14.1. IBLGenerator, file formats and importing .....	73
6.14.2. Limitations to IBL with an OpenGL renderer .....	74
6.15. Languages .....	74
6.15.1. Display languages in EB GUIDE Studio .....	74
6.15.2. Languages in the EB GUIDE model .....	74
6.15.3. Export and import of language-dependent texts .....	75
6.16. Linking .....	76
6.17. Namespaces .....	78
6.18. Model interfaces .....	80
6.18.1. Import of datapool items .....	80
6.18.2. Import of events .....	80
6.18.3. Import of event groups .....	81
6.18.4. Import of namespaces .....	81
6.19. Photoshop file format support .....	82
6.20. Resource management .....	82
6.20.1. Fonts .....	83
6.20.1.1. Bitmap fonts .....	83
6.20.1.2. Multifont support .....	84
6.20.2. Image-based lighting for 3D graphics .....	86
6.20.3. Images .....	86
6.20.3.1. 9-patch images .....	86
6.20.4. Meshes for 3D graphics .....	87
6.21. Scripting language EB GUIDE Script .....	88
6.21.1. Capabilities and areas of application .....	88
6.21.2. Prefixes and identifiers .....	88
6.21.3. Comments .....	89
6.21.4. Types .....	89
6.21.5. Expressions .....	90
6.21.6. Constants and references .....	90
6.21.7. Arithmetic and logic expressions .....	91
6.21.8. L-values and r-values .....	92
6.21.9. Local variables .....	92
6.21.10. While loops .....	93
6.21.11. If-then-else .....	94

6.21.12. Foreign function calls .....	95
6.21.13. Datapool access .....	96
6.21.14. Widget properties .....	97
6.21.15. Lists .....	98
6.21.16. Events .....	99
6.21.17. String formatting .....	101
6.21.18. The standard library .....	101
6.22. Scripted values .....	101
6.23. Skins .....	103
6.24. State machines and states .....	104
6.24.1. Dynamic state machines .....	105
6.24.2. States .....	106
6.24.2.1. Initial state .....	107
6.24.2.2. View state .....	107
6.24.2.3. Compound state .....	107
6.24.2.4. Choice state .....	108
6.24.2.5. History states .....	109
6.24.2.6. Final state .....	112
6.24.3. Transitions .....	113
6.24.4. Execution of a state machine .....	116
6.24.5. EB GUIDE notation in comparison to UML notation .....	119
6.24.5.1. Supported elements .....	119
6.24.5.2. Not supported elements .....	120
6.24.5.3. Deviations from UML .....	120
6.25. Touch input .....	121
6.25.1. Non-path gestures .....	121
6.25.2. Path gestures .....	122
6.25.3. Input processing and gestures .....	122
6.25.4. Multi-touch input .....	122
6.26. Widgets .....	123
6.26.1. View .....	123
6.26.2. Widget categories .....	125
6.26.3. Widget properties .....	126
6.26.4. Widget templates .....	127
6.26.5. Widget features .....	129
6.26.5.1. List management widget feature category .....	130
7. Modeling HMI behavior .....	131
7.1. Modeling a state machine .....	131
7.1.1. Adding a state machine .....	131
7.1.2. Defining an entry action for a state machine .....	132
7.1.3. Defining an exit action for a state machine .....	132
7.1.4. Deleting a state machine .....	133

7.2. Modeling a dynamic state machine .....	133
7.2.1. Enabling a dynamic state machine list .....	133
7.2.2. Adding a dynamic state machine .....	134
7.2.3. Controlling a dynamic state machine .....	134
7.3. Modeling states .....	135
7.3.1. Adding a state .....	135
7.3.2. Adding a state to a Compound state .....	135
7.3.3. Adding a Choice state .....	136
7.3.4. Defining an entry action for a state .....	137
7.3.5. Defining an exit action for a state .....	137
7.3.6. Deleting a model element from a state machine .....	138
7.4. Connecting states through transitions .....	138
7.4.1. Adding a transition between two states .....	139
7.4.2. Moving a transition .....	139
7.4.3. Moving transition labels .....	140
7.4.4. Defining a trigger for a transition .....	141
7.4.5. Adding a condition to a transition .....	141
7.4.6. Adding an action to a transition .....	142
7.4.7. Adding an internal transition to a state .....	143
7.4.8. Changing the priority of choice transitions .....	143
8. Modeling HMI appearance .....	145
8.1. Changing the background color of states and state machines .....	145
8.2. Working with widgets .....	145
8.2.1. Adding a View .....	145
8.2.2. Adding a basic widget to a View .....	146
8.2.2.1. Adding a Rectangle .....	146
8.2.2.2. Adding an Ellipse .....	146
8.2.2.2.1. Editing an Ellipse .....	147
8.2.2.3. Adding an Image .....	148
8.2.2.4. Adding a Label .....	149
8.2.2.5. Adding a container .....	150
8.2.2.6. Adding an Instantiator .....	151
8.2.2.7. Adding an Animation .....	153
8.2.2.7.1. Animating a widget .....	153
8.2.2.7.2. Animating a view transition .....	154
8.2.2.7.3. Animating with a script curve .....	157
8.2.2.7.4. Animating a datapool item .....	158
8.2.2.7.5. Animating colors .....	160
8.2.2.8. Adding an Alpha mask .....	161
8.2.3. Adding a 3D widget to a View .....	162
8.2.3.1. Adding a Scene graph to a View .....	162
8.2.4. Importing a .psd file to a View .....	163



8.2.5. Extracting images from a .psd file .....	164
8.2.6. Importing IBL files .....	164
8.2.7. Deleting a widget from a View .....	166
8.3. Working with widget properties .....	166
8.3.1. Positioning a widget .....	166
8.3.2. Resizing a widget .....	167
8.3.3. Linking between widget properties .....	168
8.3.4. Linking a widget property to a datapool item .....	171
8.3.5. Linking to a list element .....	172
8.3.6. Adding a user-defined property to a widget .....	175
8.3.6.1. Adding a user-defined property of type <code>Function () : bool</code> .....	176
8.3.7. Renaming a user-defined property .....	177
8.3.8. Editing a property of type list .....	177
8.3.9. Managing order and visibility of widgets .....	178
8.4. Extending a widget by widget features .....	180
8.4.1. Adding a widget feature .....	180
8.4.2. Removing a widget feature .....	182
8.5. Changing the Label settings .....	183
8.5.1. Changing the font size of a Label .....	184
8.5.2. Changing the font of a Label .....	184
8.5.3. Changing the line spacing .....	185
8.5.3.1. Changing the default line spacing .....	186
8.5.3.2. Changing the line spacing for multiple lines .....	187
8.5.4. Changing the text position .....	187
8.5.5. Managing multifont support .....	190
8.5.6. Placing text along a circular path .....	193
8.6. Working with language support .....	194
8.6.1. Adding a language to the EB GUIDE model .....	194
8.6.2. Adding language support to a datapool item .....	195
8.6.3. Deleting a language .....	195
8.6.4. Creating an export set for languages .....	196
8.7. Working with skin support .....	197
8.7.1. Adding a skin to the EB GUIDE model .....	197
8.7.2. Adding skin support to a datapool item .....	198
8.7.3. Switching between skins .....	199
8.7.4. Deleting a skin .....	199
8.7.5. Creating an export set for skins .....	200
8.8. Working with templates .....	200
8.8.1. Adding a template .....	201
8.8.2. Creating a template from widget tree .....	201
8.8.3. Defining the template interface .....	202
8.8.4. Using a template .....	203



8.8.5. Deleting a template .....	203
8.9. Enabling anti-aliasing .....	204
8.9.1. Enabling anti-aliasing globally .....	204
8.9.2. Enabling anti-aliasing for scene graphs .....	205
9. Handling data .....	206
9.1. Working with namespaces .....	206
9.1.1. Adding a namespace .....	206
9.1.2. Adding model elements to a namespace .....	207
9.1.3. Moving model elements between namespaces .....	207
9.1.4. Deleting a namespace .....	208
9.2. Working with events .....	209
9.2.1. Adding an event .....	209
9.2.2. Adding a parameter to an event .....	210
9.2.3. Addressing an event .....	211
9.2.4. Mapping a key to an event .....	211
9.2.5. Adding events to a model interface .....	213
9.2.6. Deleting an event .....	213
9.3. Working with datapool items .....	214
9.3.1. Adding a datapool item .....	214
9.3.2. Editing datapool items of type list .....	215
9.3.3. Converting a property to a scripted value .....	216
9.3.4. Establishing external communication .....	217
9.3.5. Linking between datapool items .....	218
9.3.6. Adding datapool items to a model interface .....	220
9.3.7. Deleting a datapool item .....	220
10. Handling a project .....	222
10.1. Creating a project .....	222
10.2. Opening a project .....	222
10.2.1. Opening a project from the file explorer .....	223
10.2.2. Opening a project within EB GUIDE Studio .....	223
10.3. Renaming model elements .....	224
10.4. Validating and simulating an EB GUIDE model .....	224
10.4.1. Validating an EB GUIDE model .....	225
10.4.1.1. Validating an EB GUIDE model in EB GUIDE Studio .....	225
10.4.1.2. Validating an EB GUIDE model using command line .....	226
10.4.2. Starting and stopping the simulation .....	226
10.5. Exporting an EB GUIDE model .....	226
10.5.1. Exporting an EB GUIDE model using EB GUIDE Studio .....	227
10.5.2. Exporting an EB GUIDE model using command line .....	228
10.6. Changing the display language of EB GUIDE Studio .....	228
10.7. Configuring profiles .....	228
10.7.1. Adding a profile .....	229

10.7.2. Adding a library .....	229
10.7.3. Configuring a scene .....	232
10.8. Exporting and importing language-dependent texts .....	233
10.8.1. Exporting language-dependent texts .....	233
10.8.2. Importing language-dependent texts .....	234
10.8.2.1. Importing language-dependent texts using EB GUIDE Studio .....	235
10.8.2.2. Importing language-dependent texts using command line .....	235
10.9. Working with model interfaces .....	236
10.9.1. Adding a model interface .....	236
10.9.2. Exporting a model interface .....	237
10.9.3. Importing a model interface .....	238
10.9.4. Updating an imported model interface .....	239
10.9.5. Deleting a model interface .....	239
11. Working with EB GUIDE Monitor .....	241
11.1. Starting EB GUIDE Monitor as a stand-alone application .....	241
11.2. Configuring EB GUIDE Monitor .....	242
11.3. Loading configurations into EB GUIDE Monitor .....	244
11.4. Firing an event in EB GUIDE Monitor .....	245
11.5. Changing the value of the datapool item with EB GUIDE Monitor .....	246
11.6. Using scripts in EB GUIDE Monitor .....	247
11.6.1. Writing script files for EB GUIDE Monitor .....	247
11.6.2. Starting scripts in EB GUIDE Monitor .....	251
11.7. Exporting and importing watch lists .....	252
12. Extending EB GUIDE Studio .....	255
12.1. Concepts .....	255
12.1.1. Dependency injection .....	255
12.1.2. EB GUIDE model extensions .....	256
12.1.3. EB GUIDE Studio UI extensions .....	258
12.2. Creating an extension project .....	260
12.3. Disabling copying of the assemblies .....	261
12.4. Running an extension .....	262
13. Best practices .....	263
13.1. Best practice: Handling scripted values .....	263
14. Tutorials .....	264
14.1. Tutorial: Adding a dynamic state machine .....	264
14.2. Tutorial: Modeling button behavior with EB GUIDE Script .....	272
14.3. Tutorial: Modeling a path gesture .....	279
14.4. Tutorial: Creating a list with dynamic content .....	281
14.5. Tutorial: Making an ellipse move across the screen .....	289
14.6. Tutorial: Adding a language-dependent text to a datapool item .....	291
14.7. Tutorial: Working with a 3D graphic .....	295
14.8. Tutorial: Rendering gamma correctly .....	300

14.9. Tutorial: Using view transition animations .....	303
14.10. Tutorial: Using script curves for animations .....	312
14.11. Tutorial: Creating a horizontal progress bar .....	316
15. References .....	321
15.1. Command line options .....	321
15.1.1. Command line options for <code>Studio.Console.exe</code> .....	321
15.1.2. Command line options for <code>Monitor.Console.exe</code> .....	322
15.2. Datapool items .....	323
15.3. Data types .....	323
15.3.1. Boolean .....	323
15.3.2. Color .....	323
15.3.3. Conditional script .....	324
15.3.4. Float .....	324
15.3.5. Font .....	325
15.3.6. Function () : bool .....	325
15.3.7. lbl .....	325
15.3.8. Image .....	326
15.3.9. Integer .....	326
15.3.10. Mesh .....	327
15.3.11. String .....	327
15.3.12. List .....	327
15.4. EB GUIDE Script .....	328
15.4.1. EB GUIDE Script keywords .....	328
15.4.2. EB GUIDE Script operator precedence .....	329
15.4.3. EB GUIDE Script standard library .....	330
15.4.3.1. EB GUIDE Script functions A - B .....	330
15.4.3.1.1. <code>abs</code> .....	330
15.4.3.1.2. <code>absf</code> .....	330
15.4.3.1.3. <code>acosf</code> .....	331
15.4.3.1.4. <code>animation_before</code> .....	331
15.4.3.1.5. <code>animation_beyond</code> .....	331
15.4.3.1.6. <code>animation_cancel</code> .....	331
15.4.3.1.7. <code>animation_cancel_end</code> .....	332
15.4.3.1.8. <code>animation_cancel_reset</code> .....	332
15.4.3.1.9. <code>animation_pause</code> .....	332
15.4.3.1.10. <code>animation_play</code> .....	332
15.4.3.1.11. <code>animation_reverse</code> .....	333
15.4.3.1.12. <code>animation_running</code> .....	333
15.4.3.1.13. <code>animation_set_time</code> .....	333
15.4.3.1.14. <code>asinf</code> .....	333
15.4.3.1.15. <code>atan2f</code> .....	334
15.4.3.1.16. <code>atan2i</code> .....	334

15.4.3.1.17. atanf .....	334
15.4.3.1.18. bool2string .....	335
15.4.3.2. EB GUIDE Script functions C - H .....	335
15.4.3.2.1. ceil .....	335
15.4.3.2.2. changeDynamicStateMachinePriority .....	335
15.4.3.2.3. character2unicode .....	335
15.4.3.2.4. clampf .....	336
15.4.3.2.5. clampi .....	336
15.4.3.2.6. clearAllDynamicStateMachines .....	336
15.4.3.2.7. color2string .....	337
15.4.3.2.8. cosf .....	337
15.4.3.2.9. deg2rad .....	337
15.4.3.2.10. expf .....	337
15.4.3.2.11. float2string .....	338
15.4.3.2.12. floor .....	338
15.4.3.2.13. fmod .....	338
15.4.3.2.14. focusMoveTo .....	338
15.4.3.2.15. focusNext .....	339
15.4.3.2.16. focusPrevious .....	339
15.4.3.2.17. format_float .....	339
15.4.3.2.18. format_int .....	340
15.4.3.2.19. frac .....	341
15.4.3.2.20. getAllLanguages .....	341
15.4.3.2.21. getAllSkins .....	341
15.4.3.2.22. getConfigItem .....	342
15.4.3.2.23. getFontAscender .....	342
15.4.3.2.24. getFontDescender .....	342
15.4.3.2.25. getFontLineGap .....	343
15.4.3.2.26. getImageHeight .....	343
15.4.3.2.27. getImageWidth .....	343
15.4.3.2.28. getLabelTextHeight .....	344
15.4.3.2.29. getLabelTextWidth .....	344
15.4.3.2.30. getLanguage .....	344
15.4.3.2.31. getLanguageName .....	344
15.4.3.2.32. getLanguageTag .....	345
15.4.3.2.33. getLineCount .....	345
15.4.3.2.34. getLineHeight .....	345
15.4.3.2.35. getProductString .....	346
15.4.3.2.36. getSkin .....	346
15.4.3.2.37. getSkinName .....	346
15.4.3.2.38. getTextHeight .....	346
15.4.3.2.39. getTextLength .....	347

15.4.3.2.40. getTextWidth .....	347
15.4.3.2.41. getVersionString .....	348
15.4.3.2.42. has_list_window .....	348
15.4.3.2.43. hsba2color .....	348
<b>15.4.3.3. EB GUIDE Script functions I - R .....</b>	<b>349</b>
15.4.3.3.1. int2float .....	349
15.4.3.3.2. int2string .....	349
15.4.3.3.3. isDynamicStateMachineActive .....	349
15.4.3.3.4. isWidgetOnActiveStatemachine .....	349
15.4.3.3.5. language .....	350
15.4.3.3.6. lerp .....	350
15.4.3.3.7. localtime_day .....	350
15.4.3.3.8. localtime_hour .....	350
15.4.3.3.9. localtime_minute .....	351
15.4.3.3.10. localtime_month .....	351
15.4.3.3.11. localtime_second .....	351
15.4.3.3.12. localtime_weekday .....	351
15.4.3.3.13. localtime_year .....	352
15.4.3.3.14. log10f .....	352
15.4.3.3.15. logf .....	352
15.4.3.3.16. maxf .....	352
15.4.3.3.17. maxi .....	353
15.4.3.3.18. minf .....	353
15.4.3.3.19. mini .....	353
15.4.3.3.20. nearbyint .....	354
15.4.3.3.21. popDynamicStateMachine .....	354
15.4.3.3.22. powf .....	354
15.4.3.3.23. pushDynamicStateMachine .....	354
15.4.3.3.24. rad2deg .....	355
15.4.3.3.25. rand .....	355
15.4.3.3.26. rgba2color .....	355
15.4.3.3.27. round .....	355
<b>15.4.3.4. EB GUIDE Script functions S - W .....</b>	<b>356</b>
15.4.3.4.1. saturate .....	356
15.4.3.4.2. seed_rand .....	356
15.4.3.4.3. setLanguage .....	356
15.4.3.4.4. setSkin .....	357
15.4.3.4.5. shutdown .....	357
15.4.3.4.6. sinf .....	357
15.4.3.4.7. skin .....	357
15.4.3.4.8. smoothstep .....	358
15.4.3.4.9. sqrtf .....	358

15.4.3.4.10. string2float .....	358
15.4.3.4.11. string2int .....	359
15.4.3.4.12. string2string .....	359
15.4.3.4.13. substring .....	359
15.4.3.4.14. system_time .....	360
15.4.3.4.15. system_time_ms .....	360
15.4.3.4.16. tanf .....	360
15.4.3.4.17. trace_dp .....	360
15.4.3.4.18. trace_string .....	361
15.4.3.4.19. transformToScreenX .....	361
15.4.3.4.20. transformToScreenY .....	361
15.4.3.4.21. transformToWidgetX .....	362
15.4.3.4.22. transformToWidgetY .....	362
15.4.3.4.23. trunc .....	362
15.4.3.4.24. widgetGetChildCount .....	363
15.5. Events .....	363
15.5.1. Decimal codes for key events .....	363
15.6. Buttons and icons .....	365
15.7. Scenes .....	371
15.8. Shortcuts .....	372
15.9. State machines .....	374
15.9.1. Haptic state machine .....	374
15.9.2. Logic state machine .....	375
15.9.3. States .....	375
15.9.3.1. Initial state .....	375
15.9.3.2. Compound state .....	375
15.9.3.3. View state .....	376
15.9.3.4. Choice state .....	376
15.9.3.5. Shallow history state .....	376
15.9.3.6. Deep history state .....	376
15.9.3.7. Final state .....	377
15.9.4. Transitions .....	377
15.9.4.1. Default transition .....	377
15.9.4.2. Choice transition .....	377
15.9.4.3. Else transition .....	377
15.9.4.4. Internal transition .....	378
15.9.4.5. Self transition .....	378
15.10. Widgets .....	378
15.10.1. View .....	378
15.10.2. Basic widgets .....	379
15.10.2.1. Alpha mask .....	380
15.10.2.2. Animation .....	381

15.10.2.2.1. Constant curve .....	382
15.10.2.2.2. Fast start curve .....	382
15.10.2.2.3. Slow start curves .....	383
15.10.2.2.4. Quadratic curve .....	384
15.10.2.2.5. Sinus curve .....	385
15.10.2.2.6. Script curve .....	385
15.10.2.2.7. Linear curve .....	386
15.10.2.2.8. Linear interpolation curve .....	387
15.10.2.3. Container .....	387
15.10.2.4. Ellipse .....	388
15.10.2.5. Image .....	388
15.10.2.6. Instantiator .....	389
15.10.2.7. Label .....	389
15.10.2.8. Rectangle .....	390
15.10.3. 3D widgets .....	391
15.10.3.1. Ambient light .....	391
15.10.3.2. Camera .....	391
15.10.3.3. Directional light .....	391
15.10.3.4. Image-based light .....	392
15.10.3.5. Material .....	392
15.10.3.6. Mesh .....	393
15.10.3.7. PBR GGX material .....	393
15.10.3.8. PBR Phong material .....	394
15.10.3.9. Point light .....	395
15.10.3.10. Scene graph .....	396
15.10.3.11. Scene graph node .....	396
15.10.3.12. Spot light .....	397
15.11. Widget features .....	397
15.11.1. Common .....	398
15.11.1.1. Child visibility selection .....	398
15.11.1.2. Enabled .....	398
15.11.1.3. Focused .....	398
15.11.1.4. Font metrics .....	399
15.11.1.5. Multiple lines .....	399
15.11.1.6. Pressed .....	400
15.11.1.7. Selected .....	401
15.11.1.8. Selection group .....	401
15.11.1.9. Spinning .....	402
15.11.1.10. Text truncation .....	402
15.11.1.11. Touched .....	403
15.11.2. Effect .....	404
15.11.2.1. Border .....	404



15.11.2.2. Coloration .....	405
15.11.2.3. Circular text .....	405
15.11.2.4. Stroke .....	406
15.11.3. Focus .....	407
15.11.3.1. Auto focus .....	407
15.11.3.2. User-defined focus .....	407
15.11.4. Gestures .....	408
15.11.4.1. Flick gesture .....	408
15.11.4.2. Hold gesture .....	409
15.11.4.3. Long hold gesture .....	410
15.11.4.4. Path gestures .....	410
15.11.4.4.1. Gesture IDs .....	411
15.11.4.5. Pinch gesture .....	412
15.11.4.6. Rotate gesture .....	413
15.11.5. Input handling .....	414
15.11.5.1. Gestures .....	414
15.11.5.2. Key pressed .....	414
15.11.5.3. Key released .....	414
15.11.5.4. Key status changed .....	415
15.11.5.5. Key unicode .....	415
15.11.5.6. Move in .....	416
15.11.5.7. Move out .....	416
15.11.5.8. Move over .....	417
15.11.5.9. Moveable .....	417
15.11.5.10. Rotary .....	418
15.11.5.11. Touch lost .....	418
15.11.5.12. Touch move .....	419
15.11.5.13. Touch pressed .....	420
15.11.5.14. Touch released .....	420
15.11.5.15. Touch status changed .....	421
15.11.6. Layout .....	422
15.11.6.1. Absolute layout .....	422
15.11.6.2. Box layout .....	422
15.11.6.3. Flow layout .....	423
15.11.6.4. Grid layout .....	424
15.11.6.5. Layout margins .....	424
15.11.6.6. List layout .....	425
15.11.6.7. Scale mode .....	426
15.11.7. List management .....	427
15.11.7.1. Line index .....	427
15.11.7.2. List index .....	427
15.11.7.3. Template index .....	428

15.11.7.4. Viewport .....	428
15.11.8. 3D .....	429
15.11.8.1. Anti-aliasing mode .....	429
15.11.8.2. Camera bloom .....	429
15.11.8.3. Camera depth of field .....	430
15.11.8.4. Camera viewport .....	431
15.11.8.5. Clear coat .....	431
15.11.8.6. Ambient texture .....	432
15.11.8.7. Anisotropy .....	433
15.11.8.8. Diffuse texture .....	433
15.11.8.9. Base color texture .....	434
15.11.8.10. Emissive texture .....	436
15.11.8.11. Light map texture .....	437
15.11.8.12. Metallic texture .....	438
15.11.8.13. Normal map texture .....	439
15.11.8.14. Opaque texture .....	440
15.11.8.15. Reflection texture .....	441
15.11.8.16. Roughness texture .....	442
15.11.8.17. Shininess texture .....	443
15.11.8.18. Specular texture .....	444
15.11.8.19. Reflectance texture .....	445
15.11.8.20. Texture coordinate transformation .....	447
15.11.8.21. Tone mapping .....	447
15.11.8.22. Screen space ambient occlusion .....	448
15.11.9. Transformation .....	449
15.11.9.1. Pivot .....	450
15.11.9.2. Rotation .....	450
15.11.9.3. Scaling .....	451
15.11.9.4. Shearing .....	451
15.11.9.5. Translation .....	451
16. Installation of EB GUIDE Studio .....	453
16.1. Background information .....	453
16.1.1. Restrictions .....	453
16.1.2. System requirements .....	453
16.2. Downloading EB GUIDE .....	454
16.3. Installing EB GUIDE .....	454
16.4. Uninstalling EB GUIDE .....	455
Glossary .....	456
Index .....	461

# 1. About this documentation

## 1.1. Target audiences of the user documentation

This chapter informs you about target audiences involved in an EB GUIDE project and the tasks they usually perform.

You can categorize your tasks and find the documentation relevant to you.

The following roles exist:

- ▶ [section 1.1.1, “Target audience: Modelers”](#)
- ▶ [section 1.1.2, “Target audience: Extension developers”](#)

### 1.1.1. Target audience: Modelers

Modelers use EB GUIDE Studio to create a human machine interface (HMI). In EB GUIDE the HMI is called EB GUIDE model. Communication with applications is carried out through determined events using the event mechanism, through datapool items using the datapool and through user-specific EB GUIDE Script functions.

Modelers perform the following tasks:

- ▶ Use an architecture of widgets and views to specify graphical elements on the displays
- ▶ Communicate with designers and usability experts to optimize user interfaces
- ▶ Use state machine functionality to specify when graphical elements are displayed
- ▶ Define how elements react to input from devices such as control panels or touch screens
- ▶ Define how elements receive information from hardware or software applications that offer services like a navigation unit
- ▶ Define interfaces between model elements as well as input and output devices

Modelers have profound knowledge of the following:

- ▶ EB GUIDE Studio features
- ▶ The UML state machine concept
- ▶ The specifications and requirements of the domain
- ▶ The interchanged data and the EB GUIDE GTF communication mechanism

- ▶ The specifications of 3D graphics, if 3D graphics are used in the project

### 1.1.2. Target audience: Extension developers

There may be missing features that cannot be provided through simply modeling an EB GUIDE model or adding customer-specific applications. This is when new widgets or a specific renderer may be required.

Extension developers perform the following tasks:

- ▶ Communicate with members of the EB GUIDE development team through [chapter 3, “Support”](#) to find out if there are already solutions to problems
- ▶ Work on the framework and develop new features, EB GUIDE Studio extensions or EB GUIDE GTF extensions
- ▶ Write code for additional modules for the following items:
  - ▶ Existing EB GUIDE GTF modules such as widgets or the shaders
  - ▶ Existing EB GUIDE Studio extensions such as additional toolbar buttons

Extension developers have the profound knowledge of the following:

- ▶ EB GUIDE interfaces
- ▶ Interaction between the central modules
- ▶ Structure of the framework's data

## 1.2. Structure of user documentation

The information is structured as follows:

- ▶ Background information

Background information introduce you to a specific topic and important facts. With this information you are able to carry out the related instructions.

- ▶ How-to-instruction

The instructions guide you step-by-step through a specific task and show you how to use EB GUIDE. Instructions are recognized by the present participle in the title (*ing*), for example, *Starting EB GUIDE Studio*.

- ▶ Tutorial

A tutorial is an extended version of a how-to-instruction. It guides you through a complex task. The headline starts with *Tutorial:*, for example *Tutorial: Creating a button*.

► Reference

References provide detailed technical parameters and tables.

► Demonstration

Demonstrations give you insight into how an application is written and the sequence of interactions. The demonstrations are part of the EB GUIDE GTF SDK.

## 1.3. Typography and style conventions

The following pictographs and signal words are used in this documentation to indicate important information.

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

---

**WARNING**



**Source and kind of 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 will find words and phrases that are displayed in bold or in italic or mono-spaced font.

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

All default text is written in Arial Regular font.

Font	Description	Example
Arial italics	to emphasize new or important terms	The <i>basic building blocks</i> of a configuration are module configurations.
Arial boldface	for GUI elements and keyboard keys	1. In the <b>Project</b> drop-down list box, select Project_A. 2. Press the <b>Enter</b> key.
Monospaced font (Courier)	for file names, directory names and chapter names	Put your script in the <code>function_name/abcdi-rectory</code> .
Monospaced font (Courier)	for user input, code, and file directories	<pre>CC_FILES_TO_BUILD =(PROJECT_PATH) / source/network/can_node.c CC_ FILES_TO_BUILD += \$(PROJECT_PATH) / source/network/can_config.c</pre> <p>The module calls the <code>BswM_Dcm_Re-</code> <code>questSessionMode()</code> function.</p> <p>For the project name, enter <code>Project_Test</code>.</p>



This is a step-by-step instruction

Whenever you see the bar with step traces, you are looking at step-by-step instructions or how-tos.

Prerequisite:

- This line lists the prerequisites to the instructions.

Step 1

An instruction to complete the task.

Step 2

An instruction to complete the task.

Step 3

An instruction to complete the task.

## 1.4. Naming conventions

In EB GUIDE Studio user guide the following folder names are used:

- The folder to which you installed EB GUIDE is referred to as `$GUIDE_INSTALL_PATH`.

For example:

```
C:/Program Files/Elektrobit/EB GUIDE Studio 6.10
```

- ▶ The folder for your EB GUIDE SDK platform is referred to as `$GTF_INSTALL_PATH`. The name pattern is `$GTF_INSTALL_PATH/platform/<platform name>`.

For example:

```
C:/Program Files/Elektrobit/EB GUIDE Studio 6.10/platform/win64
```

- ▶ The folder to which you save EB GUIDE projects is referred to as `$GUIDE_PROJECT_PATH`.

For example:

```
C:/Users/[user name]/Documents/EB GUIDE 6.10/projects/
```

- ▶ The folder to which you export your EB GUIDE model is referred to as `$EXPORT_PATH`.

For example:

```
C:/Documents/Projects/My_exported_model
```

## 1.5. Path conventions

EB GUIDE Studio supports handling of path names with more than 260 characters in Windows 10. A full path name can have more than 260 characters, however, single file names or directory names in the path still have a limit of 248 characters.



## 2. Safe and correct use

### 2.1. Intended use

- ▶ EB GUIDE Studio and EB GUIDE GTF are intended to be used in user interface projects for infotainment head units, cluster instruments and selected industry applications.
- ▶ Main use cases are mass production, specification and prototyping usage depending on the scope of the license.

### 2.2. Possible misuse

---

**WARNING****Possible misuse and liability**

You may use the software only as in accordance with the intended usage and as permitted in the applicable license terms and agreements. Elektrobit Automotive GmbH assumes no liability and cannot be held responsible for any use of the software that is not in compliance with the applicable license terms and agreements.

- 
- ▶ Do not use the EB GUIDE product line as provided by Elektrobit Automotive GmbH to implement human machine interfaces in safety-relevant systems as defined in ISO 26262/A-SIL.
  - ▶ EB GUIDE product line is not intended to be used in safety-relevant systems that require specific certification such as DO-178B, SIL or A-SIL.

Usage of EB GUIDE GTF in such environments is not allowed. If you are unsure about your specific application, contact Elektrobit Automotive GmbH for clarification at [chapter 3, “Support”](#).



## 3. Support

EB GUIDE support is available in the following ways:

- ▶ For the community edition, find comprehensive information in our articles, blogs, and user documentation.
- ▶ For the enterprise edition, contact us according to your support contract.

When you look for support, prepare the version number of your EB GUIDE Studio installation. To find the version number, open EB GUIDE Studio, go to the project center, and click **Help**. The version number is located in the lower right corner of the dialog.

## 4. Introduction to EB GUIDE

EB GUIDE assists users in development process of the human machine interface (HMI). The EB GUIDE product line provides tooling and platform for graphical user interfaces. The EB GUIDE product line is intended to be used in projects for infotainment head units, cluster instruments, and selected industry applications. Main use cases are mass production, specification and prototyping.

### 4.1. The EB GUIDE product line

The EB GUIDE product line comprises the following software parts:

- ▶ EB GUIDE Studio
- ▶ EB GUIDE TF
- ▶ EB GUIDE arware

EB GUIDE Studio is the modeling tool on your PC. With EB GUIDE Studio you model the whole HMI functionality as a central control element that provides the user access to functions.

EB GUIDE TF executes EB GUIDE models created in EB GUIDE Studio. EB GUIDE TF is available for development PCs and for different embedded platforms.

EB GUIDE arware is a software framework that enables the creation of augmented reality solutions to enhance the driving experience.

### 4.2. EB GUIDE Studio

#### 4.2.1. Modeling HMI behavior

The dynamic behavior of the EB GUIDE model is specified by placing states and by combining multiple states in state machines.

##### State machines

A state machine is a deterministic finite automaton and describes the dynamic behavior of the system. In EB GUIDE Studio different types of state machines are available, for example a haptic state machine. Haptic state machines allow the specification of graphical user interfaces.

##### States

States are linked by transitions. Transitions are the connection between states and trigger state changes.

## 4.2.2. Modeling HMI appearance

In EB GUIDE Studio you define the graphical user interface of the EB GUIDE model.

To create a graphical user interface EB GUIDE Studio offers widgets. Widgets are model elements that define the look. They are mainly used to display information, for example text labels or images. Widgets also allow users to control system behavior, for example buttons or sliders. Multiple widgets are assembled to a structure, which is called view.

## 4.2.3. Handling data

The communication between the HMI and the application is implemented with the datapool and the event system.

### Datapool

The datapool is an embedded database that holds all data to be displayed and further internal information. Datapool items store and exchange data.

### Event system

Events are temporary triggers. Events can be sent to both HMI and application to signal that something specific happens.

Application software can access events and the datapool through the API.

## 4.2.4. Simulating the EB GUIDE model

With EB GUIDE Studio you can test the functionality of your EB GUIDE model during simulation. You start the simulation with a mouse-click and can immediately experience the look and feel of your EB GUIDE model.

You interact with simulation using input devices like mouse, keyboard, or touch screen.

You can also control your EB GUIDE model with EB GUIDE Monitor and do the following:

- ▶ Change the displayed data by changing values of datapool items
- ▶ Simulate user input by firing events
- ▶ Track all changes in the log
- ▶ Start scripts

You can also use EB GUIDE Monitor as a stand-alone application.

## 4.2.5. Exporting the EB GUIDE model

To use the EB GUIDE model on the target device, you need to export the EB GUIDE model from EB GUIDE Studio and to convert it into a format that the target device understands. During the export, all relevant data is exported as a set of binary, configuration, and resource files.

## 4.3. EB GUIDE TF

EB GUIDE TF consists of the `GtfStartup` executable file and a set of libraries, which are required to execute an EB GUIDE model.

EB GUIDE GTF (EB GUIDE Graphics Target Framework) is the run-time environment executing a graphical HMI.

Most of the program code of EB GUIDE TF is platform-independent. The code can be ported to a new system very easily.

It is possible to exchange the complete HMI, simply by exchanging the EB GUIDE model files. It is not necessary to recompile EB GUIDE TF. The changed EB GUIDE model just needs to be re-exported from EB GUIDE Studio.

EB GUIDE TF uses the following platform abstractions:

- ▶ OS abstraction

Platform dependencies of the operating system (OS) are encapsulated by the Operating System Abstraction Layer (`GtfOSAL`). Functionalities that EB GUIDE TF uses from the operating system are for example the file system or TCP sockets.

- ▶ GL abstraction

Platform dependencies of the graphics subsystem are encapsulated by the renderer. EB GUIDE models contain element properties such as geometry and lighting. The data contained in exported EB GUIDE models is passed to the renderer for processing and output to a digital image. The renderer is the abstraction to the real graphic system on your hardware. EB GUIDE TF supports various renderers for different platforms.

### 4.3.1. Modularization

EB GUIDE provides several features that support a modular architecture of your HMI. You can create an HMI that consists of multiple EB GUIDE models. EB GUIDE models can run independent of each other. Multiple

EB GUIDE GTF instances can run on a single target. Data can be exchanged between models running on the same EB GUIDE GTF instance or different models running on different processes, using for example IPC.

## 4.4. EB GUIDE aware

EB GUIDE aware is a software framework that enables the creation of augmented reality solutions to make driving safer, build driver trust in the vehicle and enhance the driving experience. EB GUIDE aware uses the car's GPS (global positioning system) and sensors to acquire and identify objects in the car's environment. With EB GUIDE aware you can call out these objects to the attention of the driver within the field of view on the head-up display in real time.

EB GUIDE aware can process the following information:

- ▶ Information provided by vehicle sensors e.g. object data from an ADAS ECU
- ▶ Information provided by a map and guidance information provider e.g. ADASISv3 data
- ▶ Information computed by EB GUIDE aware from above sources e.g. by fusing map and sensor data
- ▶ Information provided by an in-vehicle infotainment system through an EB GUIDE Studio interface

EB GUIDE aware consists of two main software parts:

- ▶ Data fusion that creates a virtual model of the vehicle's environment, correlates measurements from different sensors, and extrapolates to compensate latencies.
- ▶ Visualization subsystem that decides which user interface elements are shown, calculates where these elements are shown, and renders them using the host OS graphics subsystem.

You can extend and customize the visualization subsystem of EB GUIDE aware by using EB GUIDE Studio and EB GUIDE GTF.

EB GUIDE aware has to be integrated into a vehicle-specific application. The application is responsible for converting the incoming data messages into a standardized format which can be processed by EB GUIDE aware.

## 5. Getting started

### TIP



### Default window layout

All instructions and screenshots use the default window layout. If you want to follow the instructions, we recommend to set the EB GUIDE Studio or EB GUIDE Monitor window to default layout by selecting **Layout > Reset to default layout**.

The following section gives you a short overview on HMI modeling with EB GUIDE Studio. It explains you how to start EB GUIDE Studio, how to create a project, how to model the behavior and appearance of an EB GUIDE model, and how to simulate an EB GUIDE model.

Approximate duration: 20 minutes.

## 5.1. Starting EB GUIDE



### Starting EB GUIDE

Prerequisite:

- EB GUIDE is installed.

#### Step 1

In the Windows **Start** menu, select the EB GUIDE Studio version you want to start.

EB GUIDE Studio starts. The project center is displayed.



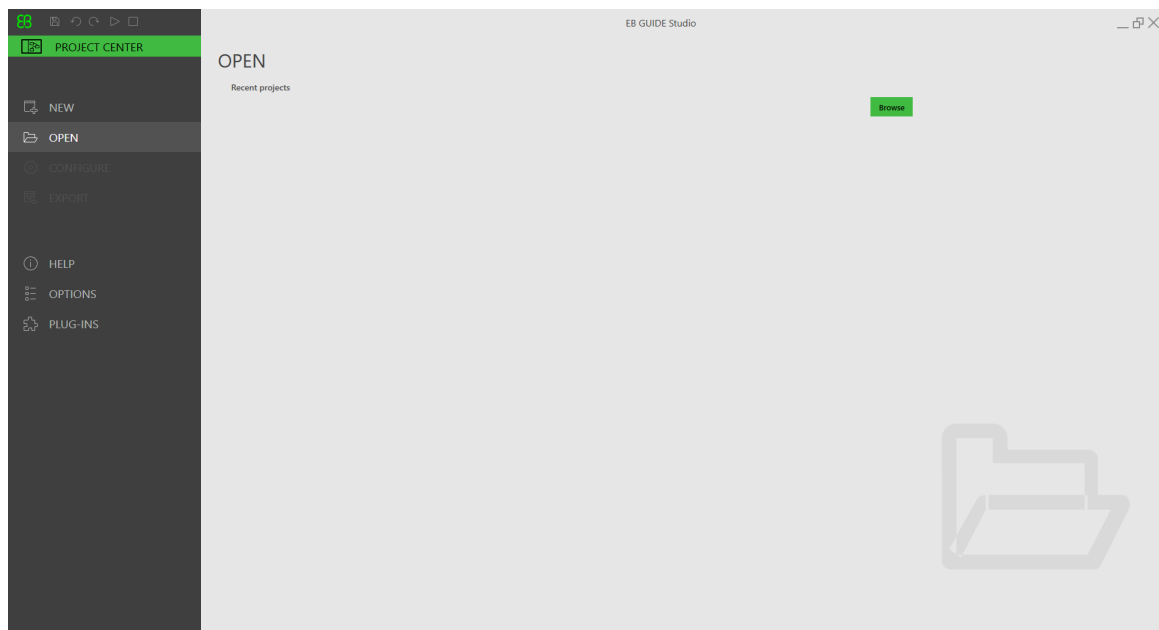


Figure 5.1. Project center

## 5.2. Creating a project



### Creating a project

Prerequisite:

- EB GUIDE Studio is started.
- A folder `C : /temp` is created.

#### Step 1

In the navigation area of the project center, click **New**.

#### Step 2

In the content area, select the `C : /temp` folder as **Location**.

#### Step 3

Enter the project name `MyProject`.

#### Step 4

Click **Create**.

The project is created. The project editor opens and displays the empty project.

The **Main** state machine is added by default and displayed in the content area.

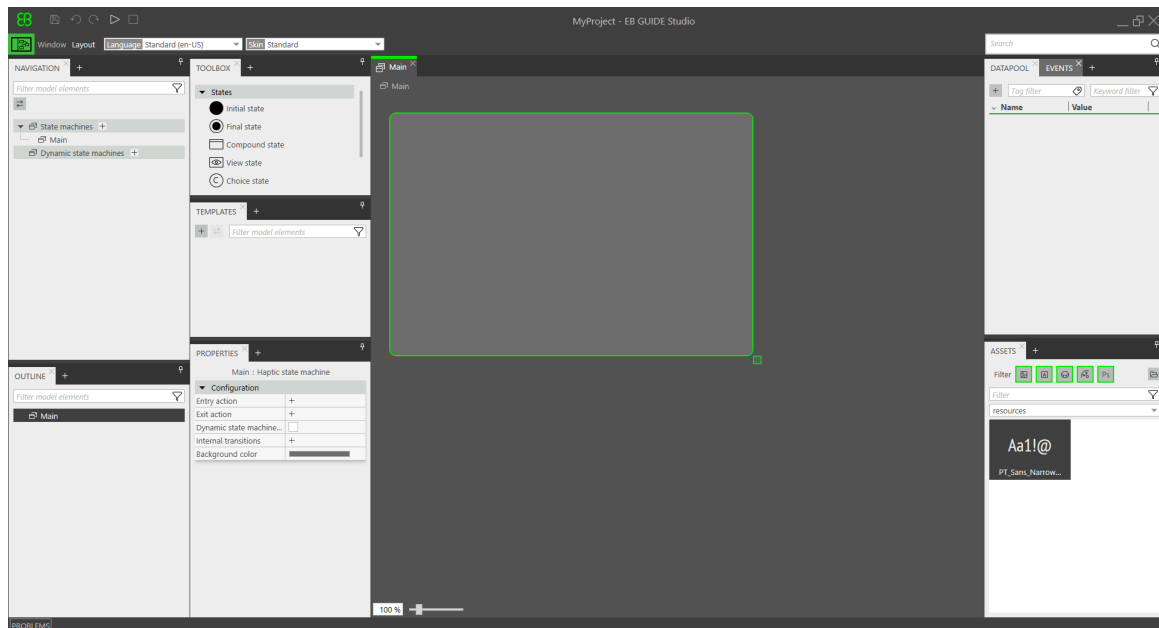


Figure 5.2. Project editor with **Main** state machine

## 5.3. Modeling HMI behavior

The behavior of your EB GUIDE model is defined by state machines. EB GUIDE uses a syntax similar to UML to do that.

In the following section, you learn how to model a state machine that displays a defined view on start-up and changes to a different view when a button is pressed.



### Adding states to the state machine

EB GUIDE offers a variety of states. The following section shows three different states. An initial state defines the starting point of the state machine. A view state displays a view by default. And the final state of the state machine terminates the state machine.

Prerequisite:

- The project `MyProject` is created.
- The content area displays the **Main** state machine.

#### Step 1

Drag a View state from the **Toolbox** into the State machine.

Along with `View state 1`, a View is added to the EB GUIDE model.

### Step 2

Repeat step 1.

View state 2 is added.

### Step 3

Drag an Initial state from the **Toolbox** into the State machine.

### Step 4

Drag a Final state from the **Toolbox** into the State machine.

The four states you added to the **Main** state machine are displayed both in the content area as a state chart and in the **Navigation** component as a hierarchical tree view when you expand the widget tree.

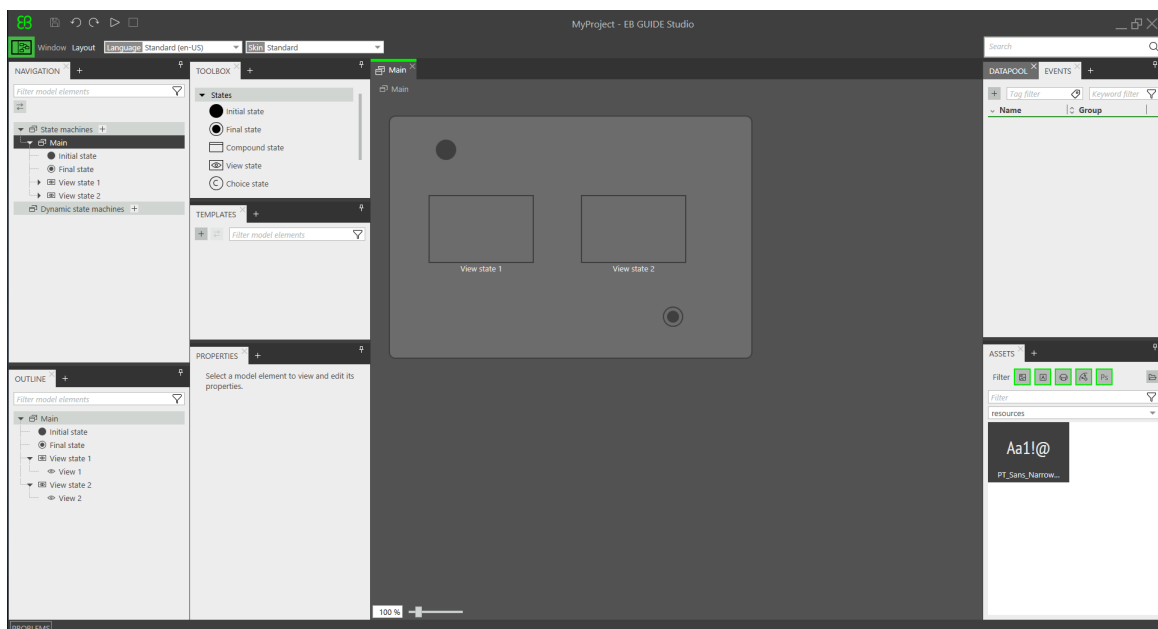


Figure 5.3. Project editor with states



## Adding transitions

Transitions are the connection between states and trigger state changes. There are different transition types. The following section shows a default transition and an event-triggered transition.

Prerequisite:

- The content area displays the **Main** state machine.
- The **Main** state machine contains an initial state, two view states, and a final state.

### Step 1

Select the Initial state. The Initial state is the source state for the transition.

A green drag point is displayed.

Step 2

Drag the drag point into the target state, `View state 1`.

Step 3

When the target state is highlighted green, release the mouse button.

The transition is displayed as a green arrow.

Step 4

Add a transition between `View state 1` and `View state 2`.

Select `View state 1`.

The state border is highlighted green.

Step 5

Hover over the highlighted state border.

A drag point is displayed.

Step 6

Drag the drag point into `View state 2` as the target state.

The transition is displayed as a green arrow.

Step 7

Select the transition between `View state 1` and `View state 2`.

As a next step, you associate the transition to an event.

Step 8

Go to the **Properties** component, enter `Event 1` in the **Trigger** combo box and click **Add event**.

An event called `Event 1` is created and added as a transition trigger. Whenever `Event 1` is fired, the transition is executed.

Step 9

Add a transition between `View state 2` and the Final state.

Select `View state 2` and repeat steps 4-6 for the final state as the target state.

Add a new event `Event 2` as a trigger.

At this point, your state machine resembles the following figure:

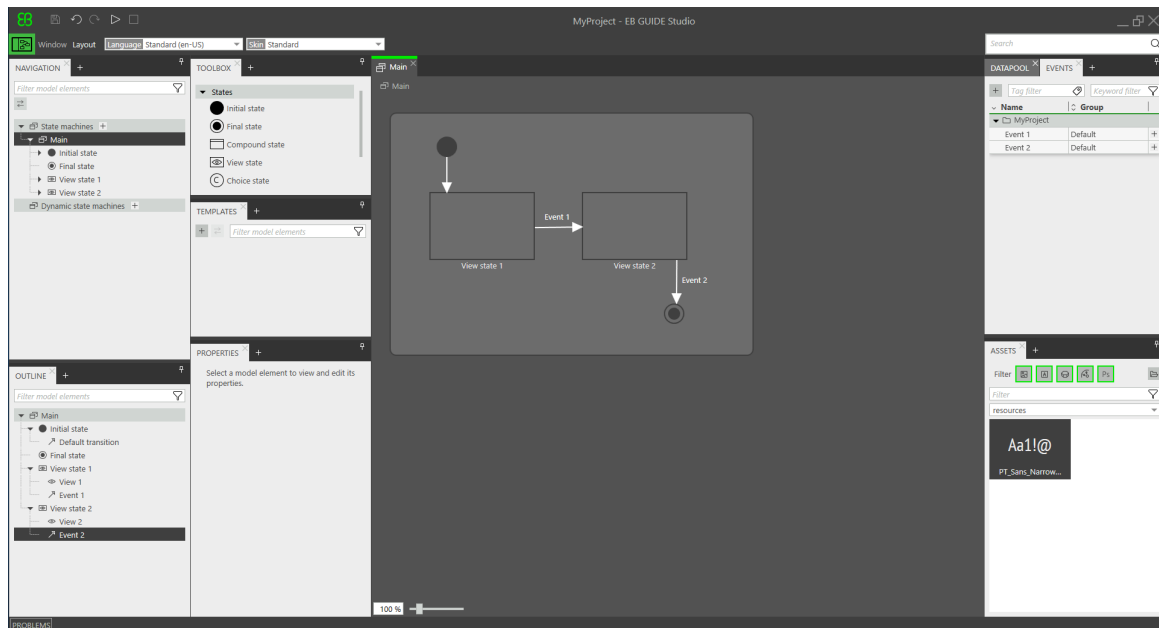


Figure 5.4. States linked by transitions with events

You have defined the behavior of a basic state machine.

## 5.4. Modeling HMI appearance

The state machine you created in the section above contains two view states. In the following section, you learn how to model a view.



### Opening a view

Prerequisite:

- View state 1 is added to the model.

#### Step 1

Double-click View state 1.

The content area displays View 1.



### Adding a button to a view

With EB GUIDE Studio you have a variety of options to model the appearance of a view.

To give you one example, the next section shows you how to add a rectangle to a view. The rectangle reacts on user input and thus functions as a button.

Prerequisite:

- The content area displays `View 1`.

#### Step 1

Drag a Rectangle from the **Toolbox** into the View.

#### Step 2

In the **Properties** component, go to the **Widget feature properties** category, and click **Add/Remove**.

The **Widget features** dialog is displayed.

#### Step 3

Under **Available widget features**, expand the **Input handling** category, and select **Touch released**.

Click **Accept**.

The related widget feature properties are added to the **Properties** component.

The default `touchPolicy` that is selected is `press then react`.

The rectangle reacts on touch input in the simulation mode.

#### Step 4

Go to the `touchShortReleased` property, and click `{ }`.

An EB GUIDE Script editor opens.

#### Step 5

Enter the following EB GUIDE Script:

```
function(v:touchId::int, v:x::int, v:y::int, v:fingerId::int)
{
  fire_delayed 500, ev:"Event 1"()
  true
}
```

If the rectangle is touched in the simulation mode, `Event 1` is fired after 500 milliseconds.

#### Step 6

Click **Accept**.

#### Step 7

In the **Properties** component, for the `fillColor` property select red.

#### Step 8

In the **Navigation** component, double-click `View 2`.

The content area displays `View 2`.

Step 9

Repeat steps 1-5.

Step 10

Enter the following EB GUIDE Script:

```
function(v:touchId::int, v:x::int, v:y::int, v:fingerId::int)
{
  fire_delayed 500, ev:"Event 2"()
  true
}
```



Figure 5.5. Widget property with an EB GUIDE Script

Step 11

Click **Accept**.

If the rectangle is touched in the simulation mode, `Event 2` is fired after 500 milliseconds.

Step 12

In the **Properties** component, for the `fillColor` property select blue.

## 5.5. Starting the simulation

EB GUIDE allows you to simulate your model on the PC before exporting it to the target device.






## Starting the simulation

### Step 1

To save the project, click  in the command area.

### Step 2

In the command area, click .

The EB GUIDE model starts and shows the behavior and appearance you modeled.

First, `View 1` is displayed. A click on the red rectangle changes the screen to `View 2`. This is because the click fires `Event 1` and `Event 1` executes the transition from `View state 1` to `View state 2`.

Then, `View 2` is displayed. A click on the blue rectangle in `View 2` terminates the state machine. This is because the click fires `Event 2` and `Event 2` executes the transition from `View state 2` to the final state.

The simulation window remains open. To stop the simulation, click .

## 6. Background information

The topics in this chapter are sorted alphabetically.

---

### TIP



#### Default window layout

All instructions and screenshots use the default window layout. If you want to follow the instructions, we recommend to set the EB GUIDE Studio or EB GUIDE Monitor window to default layout by selecting **Layout > Reset to default layout**.

---

## 6.1. 3D graphics

EB GUIDE Studio offers the possibility to use 3D graphics in your EB GUIDE project.

### 6.1.1. Supported 3D graphic formats

Only the renderers for OpenGL ES 2.0 or higher can display 3D graphics. The supported 3D graphic formats are COLLADA (.dae) and Filmbox (.fbx). For best results, use the Filmbox format.

### 6.1.2. Settings for 3D graphic files

To make 3D objects appear in a view in EB GUIDE Studio, you need to create the 3D graphic file with the following options:

- ▶ A perspective camera
- ▶ At least one object containing a mesh and at least one material
- ▶ At least one light source

---

### TIP



#### Gamma correction for scene graphs

The `gamma` property allows to adjust the luminance output of the scene graph to match the luminance response of your monitor or display device for best visual results. The value must be higher than 0.0 and is set to 2.2 per default, which is suitable for most displays.

---

To create a 3D graphic file, use third-party 3D modeling software.

3D graphic files support a wide variety of additional content, which is listed below:

- ▶ 3D objects with positions, normals, binormals, tangents, and one texture channel
- ▶ Directional light sources
- ▶ Image-based light sources
- ▶ Ambient light sources
- ▶ Point light sources with constant, linear, quadratic, and cubic attenuation
- ▶ Spot light sources with cone angles, constant, linear, quadratic, and cubic attenuation
- ▶ Perspective camera support for fields of view, near plane, and far plane
- ▶ Textures: Emissive, diffuse, specular, normal map, opacity, reflection cube, and light map

**NOTE**



**Opacity maps**

Opacity maps need a valid alpha channel.

### 6.1.3. Import of a 3D graphic file

To add a 3D graphic to a view, you need to import a 3D graphic file using a scene graph. During import EB GUIDE Studio converts the 3D graphic file into a widget tree with scene graph as a parent node. For the content of the 3D graphic file, for example camera, material, meshes, EB GUIDE Studio creates the respective widgets. If the 3D scene of the imported 3D graphic file contains animations, EB GUIDE Studio imports these animations using the linear key value interpolation curve. This curve is applied to animated properties of type float, integer, or color. It is not possible to apply the linear key value interpolation curve in the same way the other animation curves are applied. This curve is only used to import animations of 3D graphics.

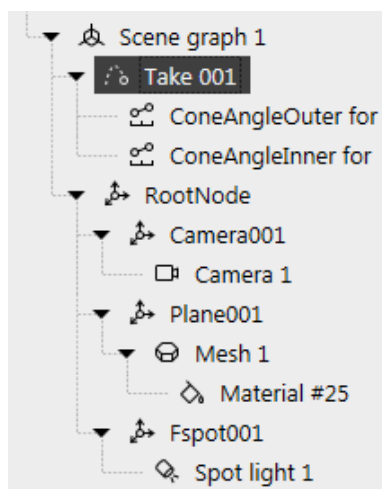


Figure 6.1. Example of a scene graph as displayed in the **Navigation** component

## NOTE



## Restrictions

Note the following:

- ▶ If your 3D graphic has more than one material per mesh, during import EB GUIDE Studio creates additional mesh for each additional material.
- ▶ During the import of an `.fbx` file only a default material widget is created. If your 3D model has other types of materials, EB GUIDE Studio adds only a default material and its properties are set to default values. In EB GUIDE Studio, you can add other types of materials using PBR Phong material and PBR GGX material widgets.
- ▶ In EB GUIDE Studio only one material per mesh is rendered. If a mesh has several materials added, only the topmost material widget is rendered.
- ▶ When you import an `.fbx` file that was exported from Blender, the emissive color is set to (0, 0, 0) and all lights in the scene are changed to a directional light with an intensity of 1.0. This is done because Blender does not export the necessary material and light information.

After importing a 3D graphic file, a subfolder is created in the folder `$GUIDE_PROJECT_PATH/<project name>/resources`. The subfolder is named after the imported `.fbx` file. Additionally date and time of creation are added to the name of the subfolder.



### Example 6.1. Naming of the import folder

The 3D graphic file is called `car.fbx`. After importing a 3D graphic file in EB GUIDE Studio, in `$GUIDE_PROJECT_PATH/<project name>/resources` you find a subfolder named `car_20160102_103029`.

The subfolder contains the following:

- ▶ Meshes as `.ebmesh` files
- ▶ Textures as `.png` or `.jpg` files

To use additional textures for your 3D graphics, copy a texture into `$GUIDE_PROJECT_PATH/<project name>/resources`. As texture use `.png` or `.jpg` images.

Import of multiple 3D graphics within one scene graph is possible.

After import, you can add, modify or delete 3D widgets.

For details, see [section 6.26, “Widgets”](#), [section 15.10.3, “3D widgets”](#), and [section 15.11.8, “3D”](#).

For instructions, see [section 8.2.3.1, “Adding a Scene graph to a View”](#), and [section 14.7, “Tutorial: Working with a 3D graphic”](#).

## 6.2. Animations

You can create different types of animations in EB GUIDE Studio, such as, loading screens, toggle buttons, transitions, fade-ins, or scaling. The component that is used to create and to configure animations is called **Animation editor**.

Animation in the common sense means that something that is displayed appears to be moving. In reality objects do not move with constant speed. Opening a drawer happens quickly at first and then it slows down. A car accelerates slowly and then the movement is steady. In HMLs this type of movement is simulated with easing. Easing is defined with animation curves.

Any value that is represented through numbers can be animated. Even colors can be animated because colors are defined in hexadecimal numbers. Every numerical value can be animated, whether it is a datapool item or a widget property.

In more general terms, animation means changing a value gradually into a different value. Gradually means that the element of time is involved in animation. Animation means changing a value over a defined time.

Animation curves bring all of this together. They define which value is the target of the animation, towards which value it changes, the duration of the animation, and the easing.

Animations are available for datapool items, widget properties, user-defined properties, and widget feature properties of the following data types:

- ▶ Boolean
- ▶ Color
- ▶ Float
- ▶ Integer
- ▶ For more information about the animation curves, see [section 15.10.2.2, "Animation"](#).
- ▶ To create a custom animation curve, see [section 6.2.6, "Script curve"](#).
- ▶ When you add an animation to a view, the view has additional properties for animations. For more information, see [section 15.10.1, "View"](#).
- ▶ To find out how animations in imported 3D graphics are handled, see [section 6.1.3, "Import of a 3D graphic file"](#).

### 6.2.1. Execution of animations

You can control animations for example with the following EB GUIDE Script functions:

- ▶ `animation_play`

- ▶ `animation_pause`
- ▶ `animation_cancel`

---

**NOTE****Simultaneous animations**

In EB GUIDE, animations can be executed simultaneously. That means, an animation target can be the target of several animation curves that are executed at the same time. In this case, the curves overwrite the target's value at the same time.

---

## 6.2.2. Widget animation

Animating widget means applying animation curves to the values of widget properties. You can move widgets, change their size, or fade them in and out. Animation is available for widget properties, user-defined properties, and widget feature properties of the following data types:

- ▶ Boolean
- ▶ Color
- ▶ Float
- ▶ Integer

For instructions, see [section 8.2.2.7.1, “Animating a widget”](#).

## 6.2.3. Datapool item animation

Animating datapool items means applying animation curves to the values of datapool items. Animation is available for datapool items of the following data types:

- ▶ Boolean
- ▶ Color
- ▶ Float
- ▶ Integer

For instructions, see [section 8.2.2.7.4, “Animating a datapool item”](#).

## 6.2.4. View transition animation

If you want to execute an animation when a view is entered or exited, use view transition animation (VTA). VTA is triggered by a view change. You can define view transition animations for view states and view templates. Every time you re-use the view template, the instance inherits the view transition animations of the template. The following types of VTA are available.

Table 6.1. Animation types

Animation type	Description
Entry animation	The animation is played when the view state with the animation is entered. The animation can only manipulate the widget properties and the widget feature properties of the added view.
Exit animation	Animation is played when the view state with the animation is exited. The animation can only manipulate the widget properties and the widget feature properties of the added view.
Change animation	The animation is played on view state change. The animation can manipulate the properties of source and destination view. The source view is the view state the animation is added to. The destination view can be another view state or view template. When you use the VTA change animation, be aware that during the lifetime of the VTA, both views are visible and drawn in the same location. The destination view is drawn on top of the start view.
Pop-up on animation	Only available for view templates and dynamic state machine views. The animation is played when the respective dynamic state machine is activated (pushed dynamic state machine). The animation can manipulate the properties and the widget properties of the added view.
Pop-up off animation	Only available for view templates and dynamic state machine views. The animation is played when the respective dynamic state machine is exited (popped dynamic state machine). The animation can manipulate the properties and the widget properties of the added view.

VTA is available for datapool items, widget properties, user-defined properties, and widget feature properties of the following data types:

- ▶ Boolean
- ▶ Color
- ▶ Float
- ▶ Integer

For instructions, see [section 8.2.2.7.2, “Animating a view transition”](#).

## 6.2.5. Color animation

Animating colors means fading colors into different colors, into shades of the same color, or fading something in or out. To create this effect in EB GUIDE you can apply animation curves to properties or datapool items of the color data type. The color format used in EB GUIDE is RGBA8888. The values are in hexadecimal digits. Color animation changes the hexadecimal digits according to the properties of the animation curve. This means, that during color animation all of the colors that are between the start and end colors are displayed. For instructions, see [section 8.2.2.7.5, “Animating colors”](#).

For more information about color animation in 3D graphics, see [section 6.1.3, “Import of a 3D graphic file”](#).

## 6.2.6. Script curve

For cases where the default animation curves are not suitable to handle your animation use case, you can define your own animation curve. This feature is called script curve. The script curve is an animation curve that provides a scripted `curve` property where you can enter your curve equation in EB GUIDE Script. This curve's script provides two time values that are updated while the animation is executed:

- ▶ `v:diff`: The time in ms since the last execution.
- ▶ `v:t_anim`: The time in ms since the start of the animation.

For instructions, see [section 8.2.2.7.3, “Animating with a script curve”](#). For a tutorial, see [section 14.10, “Tutorial: Using script curves for animations”](#).

## 6.3. Anti-aliasing

In EB GUIDE Studio you can enable anti-aliasing for an entire scene or for each scene graph separately. So, you can enable or disable anti-aliasing globally, and at the same time you can enable and configure it for scene graphs separately to override the global configuration.

Settings for anti-aliasing are hardware-dependent. If the required settings are not possible from hardware side, the console log displays an error message and information about what is not supported.

Consider that the higher the resolution for anti-aliasing is the better the quality of the rendering result. However, be aware that anti-aliasing decreases the rendering performance, especially on a target device. So start with no anti-aliasing and, if the performance is good, try the settings 2x or 4x anti-aliasing. If there is no visible difference in quality with higher anti-aliasing, use a lower setting. Also consider, the improvements anti-aliasing brings, will have only small significance on small displays with high resolution.

For instructions, see [section 8.9, “Enabling anti-aliasing”](#).



## 6.4. Application programming interface between application and model

EB GUIDE abstracts all communication data between an application and EB GUIDE TF in an application programming interface (API). An application is for example a media player or a navigation.

The API is defined by datapool items and events. Events are sent between HMI and application.



### **Example 6.2.** **Contents of an API**

- ▶ Event `START_TRACK` that is sent to the application and that contains the parameter `track` for the number of the track that should be played
- ▶ Event `TRACK_STOPPED` that is sent from the application to the HMI when the played track has ended
- ▶ The dynamic datapool item `MEDIA_CURRENT_TRACK` that is written by the application
- ▶ The dynamic datapool item `MEDIA_PLAY_SPEED` that defines the speed for playing and is set by the user in the HMI

## 6.5. Communication context

The communication context describes the environment in which communication occurs. An example for a communication context is a media or a navigation application which communicates with an HMI model. Changes made by one communication context are invisible to other communication contexts until the changes are published by the writer application and updated by the reader application.

A communication context is identified by a unique name in the project configuration.

For instructions, see [section 9.3.4, “Establishing external communication”](#).

## 6.6. Components of the graphical user interface

### 6.6.1. Graphical user interface of EB GUIDE Studio

The graphical user interface of EB GUIDE Studio is divided into two components: the project center and the project editor. In the project center, you administer your EB GUIDE projects, configure options, and export EB GUIDE models for copying to the target device. In the project editor, you model HMI appearance and behavior.

### 6.6.1.1. Project center

The project center is the first screen that is displayed after starting EB GUIDE Studio. All project-related functions are located in the project center. The project center consists of two parts: the navigation area and the content area.



Figure 6.2. Project center with navigation area (1) and content area (2)

#### 6.6.1.1.1. Navigation area

The navigation area of the project center consists of function tabs. You click a tab in the navigation area and the content area displays the corresponding functions and settings.

Find the following functions and settings in the tabs:

##### New

In the **New** tab, you can create a new project.

##### Open

In the **Open** tab, you can open an existing project.

##### Configure

In the **Configure** tab, you can configure settings for e.g. **Profiles**, **Skins**, etc.

##### Export

In the **Export** tab, you can export an EB GUIDE model.

## Help

In the **Help** tab, you find links to user documentation.

## Options

In the **Options** tab, you can switch the user interface language of EB GUIDE Studio.

## Plug-ins

In the **Plug-ins** tab, all loaded plug-ins are listed.

### 6.6.1.1.2. Content area

The content area of the project center is where project management and configuration takes place. For example, you select a folder to save a project or define the start-up behavior for your EB GUIDE model. The appearance of the content area depends on the tab selected in the navigation area.

### 6.6.1.2. Project editor

After creating a project, the project editor is displayed. In the project editor you model the behavior and the appearance of the HMI: you model state machines, create views, and manage events and datapool items. All components of the project editor can either be docked or floating and placed at any position of the project editor except the content area. The project editor consists of the following areas and components.

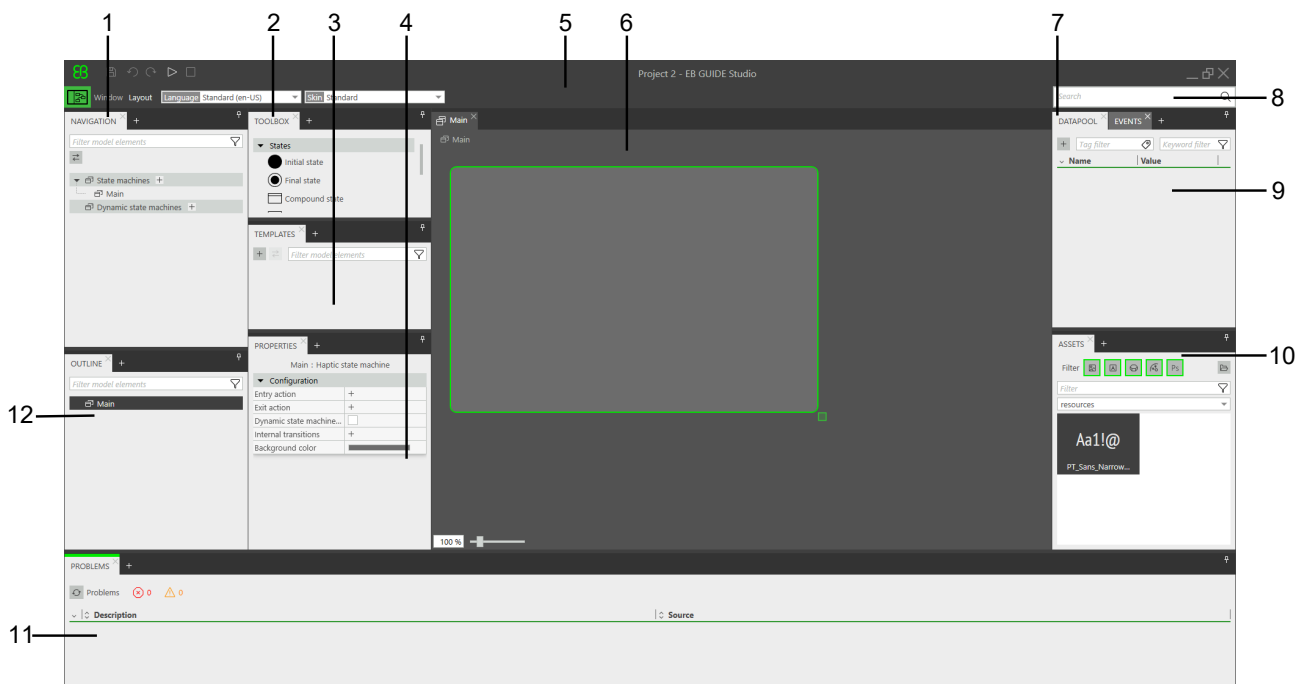


Figure 6.3. Project editor with its areas and components

- (1) **Navigation** component
- (2) **Toolbox** component
- (3) **Templates** component
- (4) **Properties** component
- (5) Command area
- (6) Content area
- (7) **Datapool** component
- (8) Search box
- (9) **Events** component
- (10) **Assets** component
- (11) **Problems** component
- (12) **Outline** component

#### 6.6.1.2.1. Navigation component

The **Navigation** component displays the model elements such as states, views, animations and transitions of your EB GUIDE model as a hierarchical structure and allows you to navigate to any element. Double-clicking a model element displays the model element in the content area.

The **Navigation** component gives you an overview of all graphical and non-graphical elements of the EB GUIDE model and reflects the state machine hierarchy.

It is also where you add elements to your EB GUIDE model, such as state machines, and dynamic state machines. You can add elements from the **Toolbox** such as widgets and animations using a drag-and-drop operation.

### TIP



### Filtering and searching

At the top of the component you find a filter box to search for any element within the component.

To start a reference search, select an element in the component and press **F3**. The search results window opens and lists all occurrences of the selected element in the EB GUIDE model.

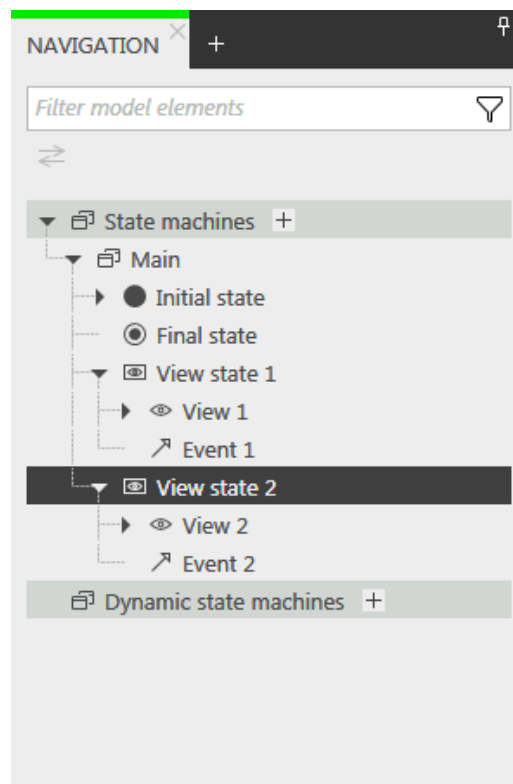


Figure 6.4. **Navigation** component in project editor

#### 6.6.1.2.2. Outline component

The **Outline** component lists the structure and model elements that are currently displayed in the content area.

#### TIP



#### Filtering and searching

At the top of the component you find a filter box to search for any element within the component.

To start a reference search, select an element in the component and press **F3**. The search results window opens and lists all occurrences of the selected element in the EB GUIDE model.

#### 6.6.1.2.3. Toolbox component

All tools you need for modeling are available in the **Toolbox** component, also referred to as **Toolbox**. Depending on the element that is displayed in the content area, the **Toolbox** offers a different set of tools, which can be dragged into the content area or the **Navigation** component. The **Toolbox** can for example contain the following:

- ▶ If the content area displays a state machine, the **Toolbox** contains states you can add to the state machine.
- ▶ If the content area displays a view, the **Toolbox** contains widgets you can arrange in the view.
- ▶ If the content area displays a scripted value property, the **Toolbox** contains EB GUIDE Script functions you can insert.

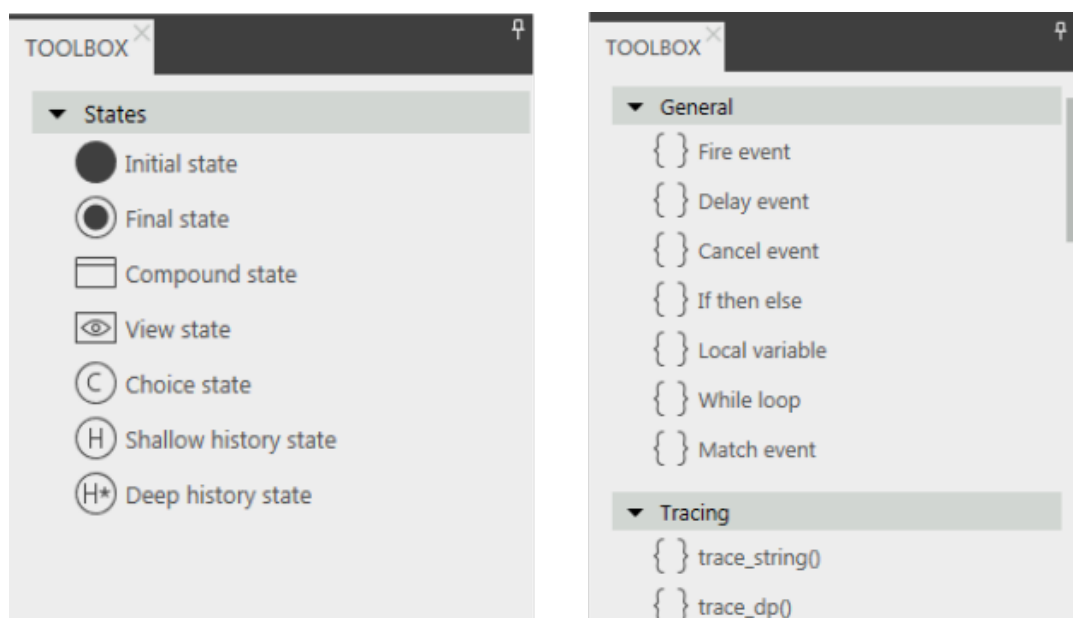


Figure 6.5. Toolbox in project editor

#### 6.6.1.2.4. Properties component

The **Properties** component displays the properties of the selected model element, for example of a widget or a state. The properties are grouped by categories and can be edited in the **Properties** component.

#### TIP



#### Searching for an element

To start a reference search, select an element in the component and press **F3**. The search results window opens and lists all occurrences of the selected element in the EB GUIDE model.

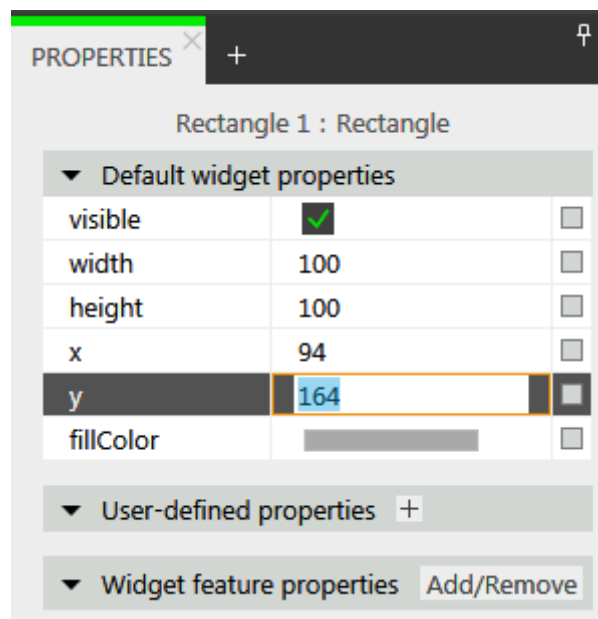


Figure 6.6. **Properties** component displaying properties of a widget

#### 6.6.1.2.5. Content area

What is displayed in the content area depends on the selection in the **Navigation** component. To edit a model element, you double-click the model element in the **Navigation** component and the content area displays it. For example, you model the states of a state machine, you arrange widgets in a view, or you edit an EB GUIDE Script in the content area.

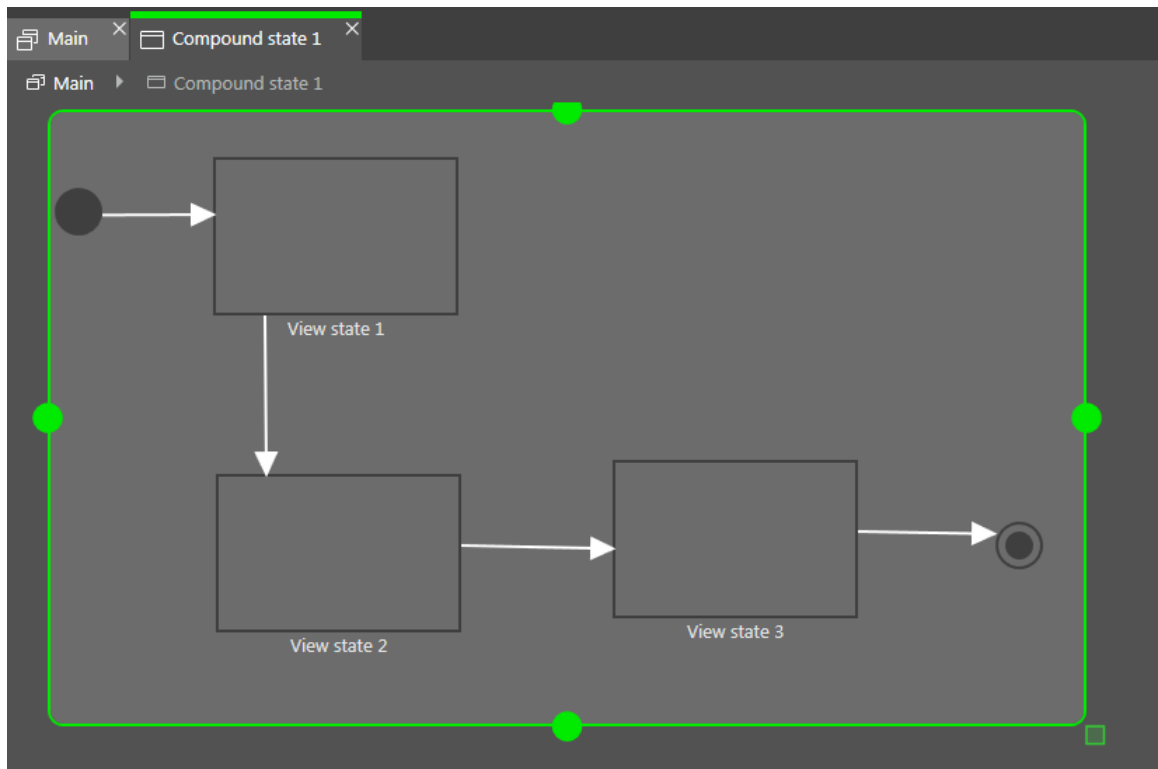


Figure 6.7. Content area in project editor

If in the content area you have an open view and the view contains an animation, the **Animation** editor is opened. In the **Animation** editor you can add curves to widget properties. You can also edit the `delay` and `duration` properties of the curves by moving the handles in the preview.

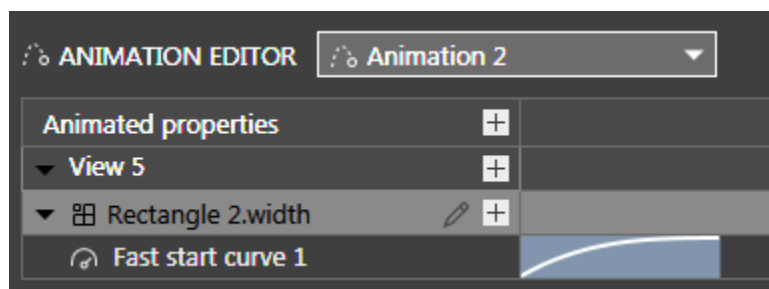


Figure 6.8. Animation editor



---

**TIP**



**Searching for an element**

To start a reference search, select an element in the component and press **F3**. The search results window opens and lists all occurrences of the selected element in the EB GUIDE model.

---

#### 6.6.1.2.6. Events component

Here you can add events to a selected namespace, move events between namespaces, and edit the properties such as **Name**, **Group**, **Parameter name** and **Type** in the event table.

Root namespace and user-defined namespaces are visible in the **Events** component only if they have at least one event added to them.

---

**TIP**



**Filtering and searching**

At the top of the component you find a filter box to search for any element within the component.

To start a reference search, select an element in the component and press **F3**. The search results window opens and lists all occurrences of the selected element in the EB GUIDE model.

---

---

**TIP**



**Multiple selection of model elements**

To select multiple elements, hold down **Ctrl** while clicking the elements you want to select. You can also hold down **Shift** while clicking the respective elements or use the **Up arrow** or **Down arrow** keys.

---

#### 6.6.1.2.7. Datapool component

Here you can add datapool items to a selected namespace, move datapool items between namespaces, and edit the properties such as **Name** and **Value**. You can also add a link to a datapool item, convert a value to a script, and add language and skin support.

Root namespace and user-defined namespaces are visible in the **Datapool** component only if they have at least one datapool item added to them.

At the top of the component you find a tag filter box and a keyword filter box.

Using the tag filter box, you can filter the datapool items according to the categories: data types, namespaces, and model interfaces. Search for a tag and select it under the corresponding category, for example search for the `Color` tag and select it under the **Data types** category. If you enter several tags from one category, the filter will display all elements that have at least one of the tags (OR disjunction). If you select several tags from

different categories, the filter will display all elements that have all of the tags (**AND** conjunction). To reset the filter, click **X** next to the tags.

Using the keyword filter, you can filter for the names of the datapool items. To filter for several keywords, use space between them. To exclude a keyword, use the **!** operator.

---

**TIP**



**Searching for an element**

To start a reference search, select an element in the component and press **F3**. The search results window opens and lists all occurrences of the selected element in the EB GUIDE model.

---

**TIP**



**Multiple selection of model elements**

To select multiple elements, hold down **Ctrl** while clicking the elements you want to select. You can also hold down **Shift** while clicking the respective elements or use the **Up arrow** or **Down arrow** keys.

---

#### 6.6.1.2.8. Assets component

Here you can add resources such as images, fonts, `.ebmesh`, `.psd`, and `.ebibl` files. All resource files are located in `$GUIDE_PROJECT_PATH/<project name>/resources` and its subfolders are displayed in the preview area of the component.

---

**TIP**



**Filtering**

At the top of the component you find a filter box to search for any element within the component.

---

#### 6.6.1.2.9. Namespaces component

Here you can create, move and delete namespaces. A root namespace is added by default.

---

**TIP**




**Default layout**

The **Namespaces** component is not in the default layout. To open it, select menu **Layout > Namespaces**.

---

#### 6.6.1.2.10. Command area

In the command area, you find:

- ▶ The  button that opens the project center
- ▶ Search box to search for elements of the model and jump to them
- ▶ Further menus

#### Search box


Model elements can be found with the help of the search box. Use the search box as follows:

- ▶ Click the search box or use the **Ctrl+F** shortcut to jump into the search box. Enter the name of the model element to be searched.

Alternatively, you can select a model element and press **F3**. Search results window opens and search results are shown.

- ▶ Jump to a model element by double-clicking it in the hit list.

The left part of the search results window lists the model elements that are found grouped by categories. Use the filter buttons above to show or hide categories. Select a model element to get a preview or to see the properties of the model element in read-only mode.

When closing the search results window the last search term, filter settings and corresponding hit list are saved and shown when the search results window is opened again. When model elements were changed in between, the search needs to be executed again. To refresh the search results, click .

The search is not case sensitive.

When using the asterisk \* for wildcard search the following rules apply:

- ▶ Search entry *t* returns all element names containing a t.
- ▶ Search entry *\*t* returns all element names ending with t.
- ▶ Search entry *t\** returns all element names starting with t.

You can search for the following model element categories.

Table 6.2. Categories in search box

Category	Description
States	The hit list also shows the parent states of the states found.
Views	The hit list also shows the parent states of the views found.
Templates	The hit list also shows the parent states and parent widgets of the templates found.
Events	The preview shows the properties of the event.
Datapool items	The preview shows the properties of the datapool item.

Category	Description
Scripts	The preview shows the content of the scripts containing the text. The found text is highlighted.
Properties	The preview shows the widget to which the property belongs.
View transition animations	The preview shows the view to which the view transition animation belongs.

#### 6.6.1.2.11. Problems component

In the **Problems** component you can check if your model is valid. It displays possible errors and warnings of the currently opened EB GUIDE model. To jump directly to the part where the problems occur, double-click the description.

#### 6.6.1.2.12. VTA component

In the **VTA** (view transition animations) component you can edit view transition animations of a view state or a view template. You can select different animation types. All relevant animation types such as the first one of each animation type with condition true start at the same time.

The **VTA** component is not shown in the default layout. To open the **VTA** component, select **VTA (view transition animations)** in the **Layout** menu.

#### 6.6.1.2.13. Templates component

In the **Templates** component you can create widget templates. Templates are useful when you want to reuse a widget in your EB GUIDE model.

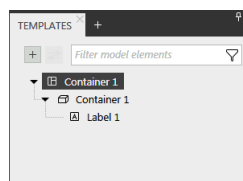


Figure 6.9. **Templates** component in project editor

## 6.6.2. Graphical user interface of EB GUIDE Monitor

In EB GUIDE Monitor, you can rearrange components and add new components according to your project's needs. You can also dock and undock components within the EB GUIDE Monitor window.

### TIP



### Default window layout

All instructions and screenshots use the default window layout. If you want to follow the instructions, we recommend to set the EB GUIDE Studio or EB GUIDE Monitor window to default layout by selecting **Layout > Reset to default layout**.

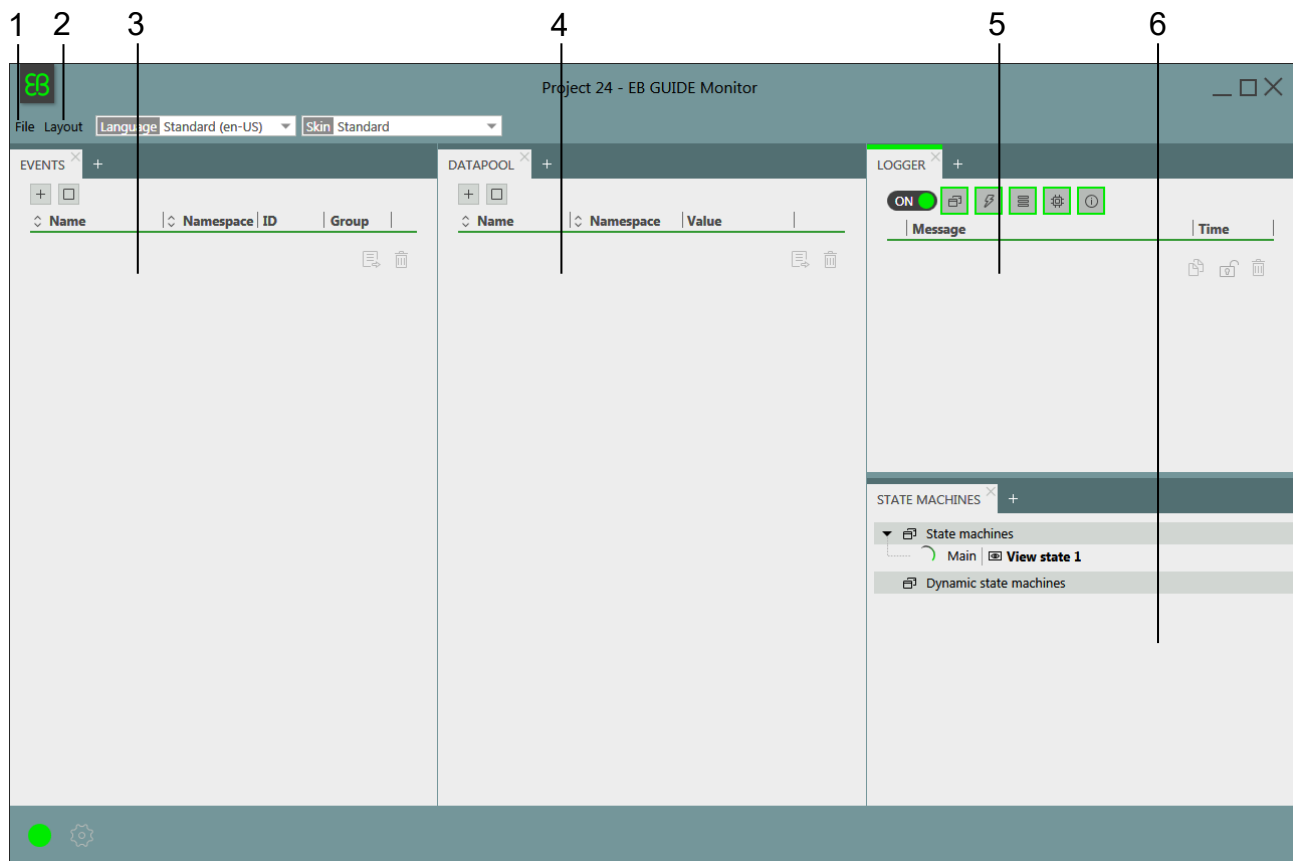




Figure 6.10. EB GUIDE Monitor with default layout

- (1) **File** menu
- (2) **Layout** menu
- (3) **Events** component
- (4) **Datapool** component
- (5) **Logger** component
- (6) **State machines** component

EB GUIDE Monitor contains the following components:

- In the **Events** component you can add and fire events. If an event has parameters, you can change the parameters and then fire this event.

- ▶ In the **Datapool** component you can add datapool items and change their values.
- ▶ In the **Logger** component all changes, information messages, errors, and warnings are tracked. At the top of the component you find filter buttons to filter entries within the component. To change the auto-scrolling functionality, click  or .
- ▶ In the **State machines** component the currently active state and state machine are shown.
- ▶ In the **Scripting** component you can start scripts and see the output script messages. Note that the **Scripting** component is not in the default layout. To add the component, click **Layout > Scripting**.

It is also possible to change the language and the skin using the drop-down boxes in the command area.

For more information on EB GUIDE Monitor, see [section 6.9, “EB GUIDE Monitor”](#).

For instructions, see [chapter 11, “Working with EB GUIDE Monitor”](#).

### 6.6.3. Dockable components

You can dock all components of the project editor in EB GUIDE Studio and of EB GUIDE Monitor as tabs or undock as floating components. You can drag a component as floating component to any part of the project editor except the content area.

The arrows of the docking control help you to select a docking location and the live preview shows you how the layout is going to look like.

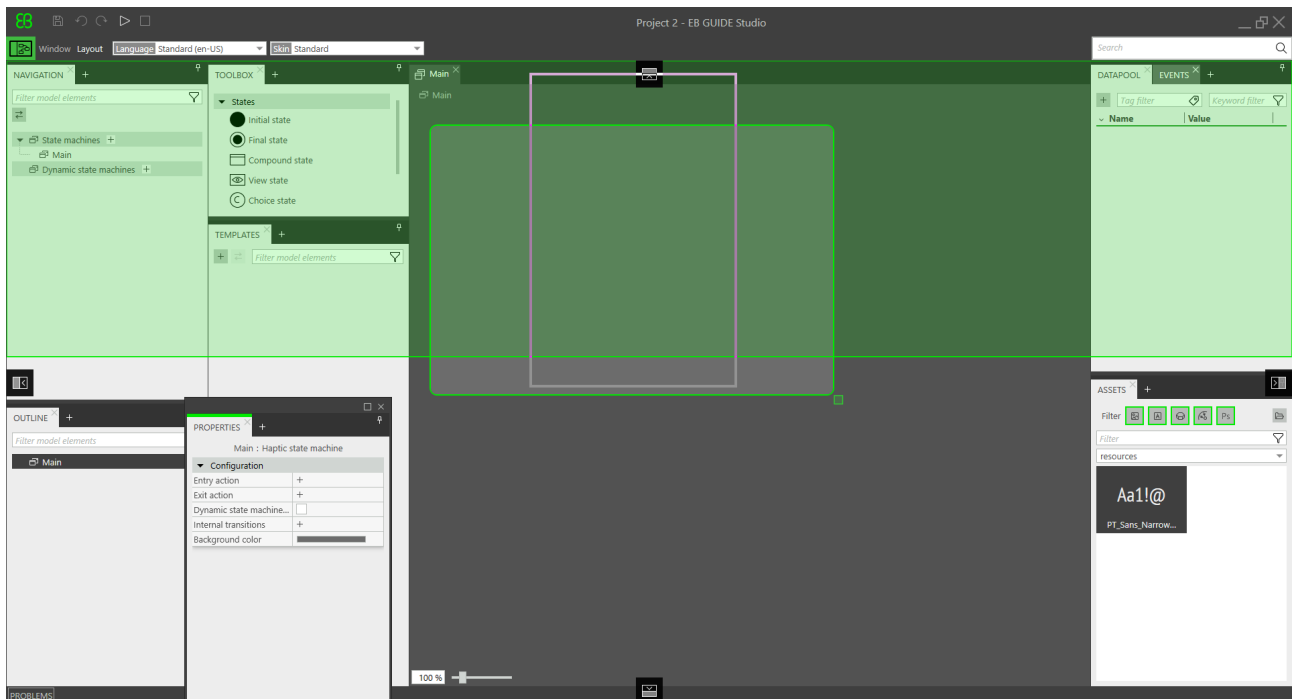


Figure 6.11. Docking control and live preview

---

**TIP**



**Default layout**

To restore the default layout, go to the command area and select **Layout > Reset to default layout**.

---

**TIP**



**Auto-hide**

To gain more space in the project editor, you can hide components.

- ▶ To hide a component or a component group, click the pin symbol.
  - ▶ To display a hidden component, hover over the tab with the mouse and click the pin symbol again.
- 

## 6.7. Datapool

### 6.7.1. Concept

During the execution, a model communicates with different applications. To enable the communication, your EB GUIDE model has to provide an interface. The datapool is an interface which allows access to datapool items to exchange data. Datapool items store values and communicate between HMI and applications. Datapool items are defined in the EB GUIDE model.

### 6.7.2. Datapool items

Datapool items are model elements that are used to do the following:

- ▶ Sending data from the applications to the HMI.
- ▶ Sending data from the HMI to the applications.
- ▶ Storing data that is only used in either HMI or applications.
- ▶ Storing data persistently.
- ▶ Storing data that is skin or language dependent.
- ▶ Storing data that needs to be exchanged between EB GUIDE model and application.
- ▶ Storing data that needs to be exchanged between EB GUIDE models.
- ▶ Storing data that needs to be exchanged between EB GUIDE GTF instances over IPC.

For instructions, see [section 9.3.1, “Adding a datapool item”](#).

To channel communication, you use writer and reader applications.

Internal communication is used to store data. Using two different applications establishes external communication.

For instructions, see [section 9.3.4, “Establishing external communication”](#).

For more information, see [section 6.16, “Linking”](#).

### 6.7.3. Windowed lists

The EB GUIDE product line supports the concept of windowed lists. The windowed list operating mode is often used to reduce memory consumption for the display of large lists, for example all MP3 titles in a folder. Those lists are typically provided by one application, for example media application, and are only partially displayed by another application, for example HMI.

The writer application defines a virtual list length and a number of windows, which possibly contain only parts of the list. The reader application reads data only from locations that are covered by windows. Reading from other locations fails. In such a use case, the reader application has to inform the writer application about the currently required parts of the list. For example, HMI can make application calls that provide the current cursor position within the complete list.



#### **Example 6.3. Windowed list**

The MP3 title list of an audio player device has 1,000,000 elements. The HMI has to display this list on three different displays in parallel: head unit display, cluster instrument display, and head-up display.

Each display is controlled separately, has a different number of display lines and has a different cursor position within the complete list.

Whenever one of the three cursors moves, the HMI sends the new position asynchronously to the media application through an event. The media application provides a list with three windows. Each of the three windows is associated to one of the three displays. Window updates delay a little bit after the cursor moves. Therefore it is advisable to use window positions and window sizes which cover an extended range around the lines that are shown by the specific display.

## 6.8. EB GUIDE model and EB GUIDE project

An EB GUIDE model is the sum of all elements that describe the look and behavior of an HMI. It is built entirely in EB GUIDE Studio. You can simulate the EB GUIDE model on your PC.





To execute an EB GUIDE model on a target device, you export the EB GUIDE model and copy the resulting binary files to the target device.

An EB GUIDE project consists of an EB GUIDE model and settings that are needed for running the EB GUIDE model on the target device. Exported EB GUIDE models can communicate with each other on the target device through defined model elements. You can define these model elements in the model interface.

An EB GUIDE project contains objects that are configured and linked within an EB GUIDE model. These objects are called EB GUIDE model elements. Examples for EB GUIDE model elements are as follows:

- ▶ Datapool item
- ▶ Event
- ▶ State
- ▶ State machine
- ▶ Widget
- ▶ Resource
- ▶ Language

### 6.8.1. Storage format

The EB GUIDE project is stored in multiple files of the EB GUIDE-proprietary file format. The file format is represented by two file extensions:

- ▶ `.ebguide` for the EB GUIDE project file
- ▶ `.gdata` for all other project files

The EB GUIDE Studio storage format is defined by the following pseudo-EBNF syntax:

```
INT = [0-9]+ ;

HEXINT = '0' ( 'x' | 'X' ) [a-fA-F0-9]+

FLOAT = <as represented in the C# specification> ;

STRING = " " ; //escape characters are supported as specified in MSDN

SUFFIX = [a-zA-Z_-][a-zA-Z0-9_-]* ;

COLOR = [a-fA-F0-9]{8} ;

IDENTIFIER = ( '_' | [a-zA-Z] ) ( [a-zA-Z] | [0-9] | '_' | '$' | '.' ) * ;

file = header object ;
```

```
header = 'EBGUIDE' INT '.' INT '.' INT '.' INT SUFFIX ';' ;

object = type '(' objectId ')' '{' propertyList '}' ;

type = identifier [ '<' type { ',' type } '>' ] ;

property = identifier ':' value ;

value = bool
      | int
      | float
      | string
      | color
      | object
      | externalObject
      | nullObject
      | objectReference
      | propertyReference
      | list ;

string : STRING { '\\' STRING } ;

int = [ '+' | '-' ] INT
     | HEXINT ;

color = '#' COLOR ;

float = [ '+' | '-' ] FLOAT ;

bool = 'true' | 'false' ;

externalObject = '(' objectId ')' ;

nullObject = type '(' 'none' ')' ;

objectReference = '@' objectId '(' type ')' ;

propertyReference = identifier '@' objectId '(' type ')' ;

list = type '[' [ value { ',' value } ] ']' ;

identifier = IDENTIFIER | STRING ;

objectId = GUID ; //encoded as hex digits in
                //the XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX format
```

The EB GUIDE project folder `$GUIDE_PROJECT_PATH/<project name>/` contains the following:

- ▶ The `.ebguide` folder that contains `.json` files with the project- and user-relevant settings for both EB GUIDE Studio and EB GUIDE Monitor. For example what filter are activated, how tables are sorted. This folder is created once you create an EB GUIDE project. If you delete this folder and re-start EB GUIDE Studio, the settings will be reset and a new `.ebguide` folder is created.

---

**NOTE**



**.ebguide folder**

Exclude the `$GUIDE_PROJECT_PATH/<project name>/` `.ebguide` folder from the revision control system because it contains user-relevant settings.

---

- ▶ Folders with files that are relevant for the following:
  - ▶ Configuration
  - ▶ Model interfaces
  - ▶ Namespaces
  - ▶ State machines
  - ▶ Templates
  - ▶ View transition animations
- ▶ The `resource` folder with the project-specific resources. For more information, see [section 6.20, “Resource management”](#).
- ▶ The `.gdata` files that contain data for contexts, datapool, event system, languages, and skins.
- ▶ The `.txt` files that contain information about the loading errors, migration or import messages for `.xliif` files, `.psd` files, and `.fbx` files.

## 6.8.2. Export format

After exporting an EB GUIDE project, the exported files are created at the user-defined location. For instructions on how to export a project, see [section 10.5, “Exporting an EB GUIDE model”](#).

At `$EXPORT_PATH` you find the following:

- ▶ `.json` configuration files
- ▶ `.gtf` files
- ▶ Resource files. For more information, see [section 6.20, “Resource management”](#).
- ▶ Global and local header files for datapool items and events
- ▶ Header files for aspects, i.e. languages and skins

### 6.8.3. Naming of model elements

For the names of the model elements, state machines, etc. EB GUIDE Studio supports all possible characters, but the following characters are recommended for use: a-z, A-Z, 0-9, space, and underscore.

During the export of EB GUIDE model, the export files are created. In this files the model element names are transferred to variable names. If for the names you used characters outside the recommended range, the export files contain auto-generated names.



**Example 6.4.**  
**Naming of model elements**

Model elements in EB GUIDE Studio	Entries in the export files												
Datapool items <table><tr><td>Int_A</td><td>0</td><td><input checked="" type="checkbox"/></td></tr><tr><td>Int_A1</td><td>0</td><td><input type="checkbox"/></td></tr><tr><td>Int_üöä</td><td>0</td><td><input type="checkbox"/></td></tr><tr><td>üöä</td><td>0</td><td><input type="checkbox"/></td></tr></table>	Int_A	0	<input checked="" type="checkbox"/>	Int_A1	0	<input type="checkbox"/>	Int_üöä	0	<input type="checkbox"/>	üöä	0	<input type="checkbox"/>	<pre>#define LOCAL_DP_ID_INT_A ((uint32_t) 0x00000002)  #define LOCAL_DP_ID_INT_A1 ((uint32_t) 0x00000000)  #define LOCAL_DP_ID_INT_ ((uint32_t) 0x00000003)  #define LOCAL_DP_ID_GENERATED_DPITEM_NAME_ ((uint32_t) 0x00000001)</pre>
Int_A	0	<input checked="" type="checkbox"/>											
Int_A1	0	<input type="checkbox"/>											
Int_üöä	0	<input type="checkbox"/>											
üöä	0	<input type="checkbox"/>											
Events <table><tr><td>Event_A</td><td>Default</td><td><input checked="" type="checkbox"/></td></tr><tr><td>Event_A1</td><td>Default</td><td><input checked="" type="checkbox"/></td></tr><tr><td>Event_üäö</td><td>Default</td><td><input checked="" type="checkbox"/></td></tr><tr><td>üäö</td><td>Default</td><td><input checked="" type="checkbox"/></td></tr></table>	Event_A	Default	<input checked="" type="checkbox"/>	Event_A1	Default	<input checked="" type="checkbox"/>	Event_üäö	Default	<input checked="" type="checkbox"/>	üäö	Default	<input checked="" type="checkbox"/>	<pre>#define LOCAL_EVENT_MSG_ID_EVENT_A ((uint32_t) 0x00000000)  #define LOCAL_EVENT_MSG_ID_EVENT_A1 ((uint32_t) 0x00000001)  #define LOCAL_EVENT_MSG_ID_EVENT_ ((uint32_t) 0x00000002)  #define LOCAL_EVENT_MSG_ID_GENERATED_EVENT-NAME_ ((uint32_t) 0x00000003)</pre>
Event_A	Default	<input checked="" type="checkbox"/>											
Event_A1	Default	<input checked="" type="checkbox"/>											
Event_üäö	Default	<input checked="" type="checkbox"/>											
üäö	Default	<input checked="" type="checkbox"/>											

For instructions on how to export a project, see [section 10.5, “Exporting an EB GUIDE model”](#).

### 6.8.4. Validation criteria for EB GUIDE project

EB GUIDE Studio performs two types of validation check for an EB GUIDE project.

#### 6.8.4.1. Validation while opening an EB GUIDE project

When you open your EB GUIDE project, EB GUIDE Studio performs several structural verifications, for example as follows:

- ▶ If the `.ebguide` project file does not exist, or several `.ebguide` files are located in the same folder
- ▶ If the object IDs are duplicated
- ▶ If child objects are missing within the EB GUIDE project
- ▶ If there are duplicate property names
- ▶ If values of list items are inconsistent
- ▶ If the EB GUIDE Studio version number in the `.gdata` files does not correspond with the EB GUIDE Studio version number in the `.ebguide` file
- ▶ If an unknown type is referenced

If one of these criteria is met, the EB GUIDE project cannot be opened and a log file with the type of error and the position of this error inside the project files is created in `$GUIDE_PROJECT_PATH/<project name>/<project name>_LoadingErrorLog.txt`.

For instructions on how to open an EB GUIDE project, see [section 10.2, “Opening a project”](#).

#### 6.8.4.2. Validation using the Problems component

When the EB GUIDE project is already opened, you can validate the EB GUIDE model in the **Problems** component. The errors are, for example, as follows:

- ▶ If the EB GUIDE Script usage is not valid
- ▶ If a default transition is missing
- ▶ If a target of a linked item is missing

If any errors are found, you cannot simulate and export the EB GUIDE model.

For instructions on how to validate an EB GUIDE model, see [section 10.4, “Validating and simulating an EB GUIDE model”](#).

## 6.9. EB GUIDE Monitor

EB GUIDE provides the tool EB GUIDE Monitor to observe and control an EB GUIDE model during the simulation. EB GUIDE Monitor includes mechanisms for the communication with datapool, the event system, and the state machines of the EB GUIDE model.

EB GUIDE Monitor is started automatically in EB GUIDE Studio during the EB GUIDE model simulation. You can also use EB GUIDE Monitor as a stand-alone application, if you want to control your exported EB GUIDE model.

For more information on EB GUIDE Monitor GUI, see [section 6.6.2, “Graphical user interface of EB GUIDE Monitor”](#).

For instructions, see [chapter 11, “Working with EB GUIDE Monitor”](#).

For the EB GUIDE Monitor API, see `$GUIDE_INSTALL_PATH/doc/monitor/monitor_api.chm`.

You can enhance EB GUIDE Monitor with additional functionalities by creating a customized extension. The following ready-to-use EB GUIDE extension examples show you how to create an EB GUIDE Monitor extension:

- ▶ `MonitorRemoteViewPlugin`
- ▶ `MonitorUiExtension`
- ▶ `MonitorUiExtensionEvents`
- ▶ `MonitorUiExtensionDatapool`
- ▶ `MonitorUiExtensionTargetViewer`

Download the EB GUIDE extension examples from <https://github.com/Elektrobit/eb-guide-examples>. For instructions, see the `readme.md` files enclosed.

## 6.10. Event handling

### 6.10.1. Event system

The event system is an asynchronous mechanism for communication within or between applications.

The EB GUIDE event system delivers all events exactly in the order they were sent. There is no pre-defined order for delivering an event to different subscribers.

### 6.10.2. Events

An event in EB GUIDE is a model element that has a unique numerical event ID and belongs to an event group. The event ID is used by EB GUIDE GTF to send and receive the event.

---

**NOTE**



**Duplicate event IDs**

Event IDs in an event group must be unique. When you import several model interfaces, validation errors occur in case there are duplicate IDs of events that belong to different model interfaces but are within the same event group.

Since you cannot change the event IDs of the imported events in EB GUIDE Studio, revert the import, change the event IDs in the source model, export and import again. It is advisable to define event ID ranges for all EB GUIDE models beforehand.



**Example 6.5.**  
**Usage of events**

In an HMI that has a rotary button or a set of hard keys, for example left, right, up, down, enter, it may not be clear with which element a user should interact next. Therefore, these systems typically highlight the currently active display element. For example, a **YES** button is marked as active with a colored border. In EB GUIDE Studio you model this highlighting feature using the **Focused** widget feature. The element that is currently focused, i.e. the `focused` property is set to `true`, is the element that is also active. Also the parents of this element, which form the focus path, are also active. If the focused element cannot handle a key or rotary input, the input is processed along the focus path in backwards direction, i.e. towards the root element. If one of the elements of the focus path handles the input, the focus is considered as processed.

In an HMI with touch input, the interaction is done with an element at a certain position. For example, when you press a **YES** button on a touch screen, the input is not ambiguous. The reason for this is that based on the location on the display which has been pressed, the system knows that the interaction was done with the **YES** button.

Event group IDs are numerical IDs. The event group IDs between 0 and 65535 are reserved for the internal use within the EB GUIDE product line. The event group IDs greater than 65535 are free for the external use and can be used for custom event groups.

The following figure shows how you can model the touch, key, and rotary events in EB GUIDE Studio.

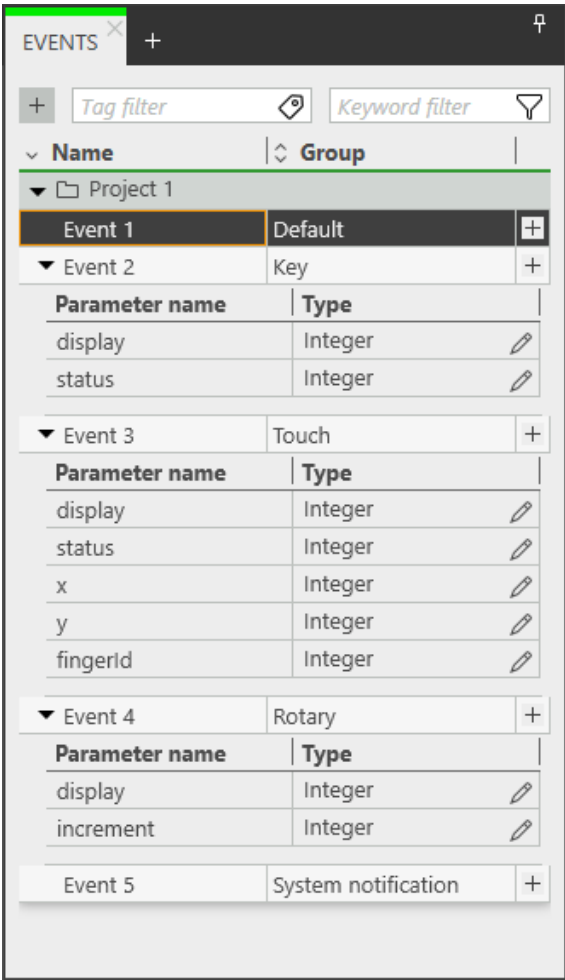


Figure 6.12. Example of event groups and event IDs

The remaining range of group IDs is available for customer-specific applications.

For instructions, see the following:

- ▶ [section 9.2.1, “Adding an event”](#)
- ▶ [section 9.2.3, “Addressing an event”](#)

For references, see [section 15.5, “Events”](#).

## 6.11. Extensions

### 6.11.1. EB GUIDE Studio extension



An EB GUIDE Studio extension is a supplement to EB GUIDE Studio and is valid for all EB GUIDE models. An EB GUIDE Studio extension does not concern EB GUIDE GTF. Typical EB GUIDE Studio extensions are custom EB GUIDE model elements or custom UI elements. For instructions and more information, see [chapter 12, “Extending EB GUIDE Studio”](#).

### 6.11.2. EB GUIDE GTF extension

An EB GUIDE GTF extension is a supplement to EB GUIDE GTF which provides additional features in EB GUIDE Studio, but is only valid for one EB GUIDE model. The EB GUIDE GTF extension is based on the EB GUIDE GTF.

Typical EB GUIDE GTF extensions are:

- ▶ New widget features
- ▶ New EB GUIDE Script functions

EB GUIDE GTF extensions are dynamic link library (.dll) or shared object (.so) files.

Place the EB GUIDE GTF extension, including their third party libraries in: `$GUIDE_PROJECT_PATH/<project name>/resources/target`

For more information and instructions, see EB GUIDE GTF user guide.

You can customize the visual appearance and behavior of your EB GUIDE model by creating a customized EB GUIDE GTF extension. A collection of ready-to use EB GUIDE examples show you how to create your own EB GUIDE GTF extensions. Download the EB GUIDE extension examples from <https://github.com/Elektrobit/eb-guide-examples>. For instructions, see the `readme.md` files enclosed.

For more information on classes and interfaces, see EB GUIDE GTF API documentation.

### 6.11.3. EB GUIDE Monitor extensions

A EB GUIDE Monitor extension provides additional functionalities to EB GUIDE Monitor.

Typical EB GUIDE Monitor extensions are:

- ▶ Additional components for EB GUIDE Monitor
- ▶ Extensions to create screenshots during the simulation

You can create your own customized extension. A collection of ready-to use EB GUIDE examples show you how to create your own EB GUIDE Monitor extensions. Download the EB GUIDE extension examples from <https://github.com/Elektrobit/eb-guide-examples>. For instructions, see the `readme.md` files enclosed.

For more information on classes and interfaces, see EB GUIDE Monitor API documentation.

## 6.12. Focus handling

In EB GUIDE Studio you model the focus management of the widgets using the **Focus** widget features: **Auto focus** and **User-defined focus**.

For more information about widget features, see [section 6.26.5, “Widget features”](#). For instructions, see [section 8.4, “Extending a widget by widget features”](#). For more information about the focus-related widget features, see [section 15.11.3, “Focus”](#).

The following two focus directions are available:

1. Forward direction: The next focusable widget is focused.
2. Backward direction: The previous focusable widget is focused.

The **Auto focus** and **User-defined focus** widget features provide a configuration for how the focus is handled for the forward direction. For the backward directions, the same focus order is used but only in reverse direction.

The **Focus** widget features have the following characteristics:

### Auto focus

In this policy the focus is distributed between the focusable widgets from left to right starting with the top row. The order is defined through the structure of the widget tree.

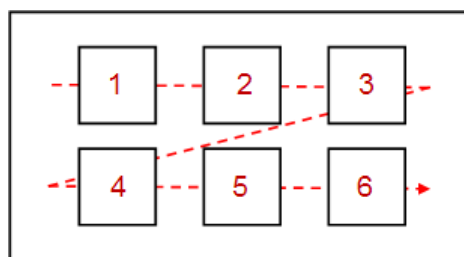


Figure 6.13. The policy of the **Auto focus** widget feature

Focusable child widgets cannot be skipped. Invisible widgets, widgets with disabled `focused` property, and widgets without the **Focused** widget feature are not recognized as valid focusable widgets. Thus they are skipped over when the currently focused widget is determined.

### User-defined focus

Due to view complexity the focus sequencing through the auto focus policy may be quite difficult. In this case it is useful to determine a user-defined focus order.

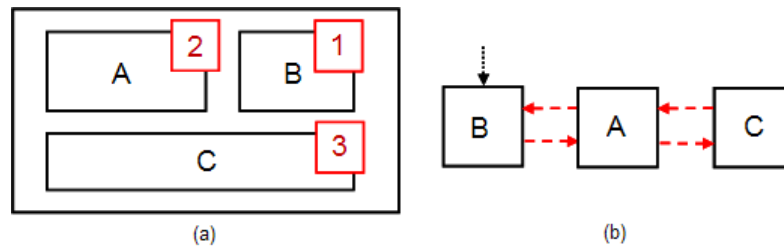


Figure 6.14. The policy of the **User-defined focus** widget feature

In [figure 6.14, “The policy of the User-defined focus widget feature”](#), (a) shows the view, while (b) shows the focus order. The order, in which the focus changes are processed, may differ from the widget tree structure.

When widgets within a widget hierarchy are marked as focusable, they are part of a focus hierarchy. This focus hierarchy consists of focusable widgets and a focus policy, the **Auto focus** widget feature or the **User-defined focus** widget feature, that defines how the focus is handled within the hierarchy. It is possible to nest focus hierarchies.

## 6.13. Gamma-correct rendering

### 6.13.1. Concepts

Gamma correction plays an important role in the rendering pipeline. Gamma correction affects color reproduction on the screen as well as image color storage. Gamma expresses the relationship between color values and the perceived intensities on the screen, for an example of gamma see [figure 6.15, “Example of gamma”](#).

The human visual system (HVS) exhibits a similar behavior. It is more sensitive to luminance differences in dark image regions than to luminance differences in bright image regions. Common 8-bit image formats (JPEG, PNG) exploit this fact and store colors in the sRGB color space that uses a non-linear transfer function in order to increase the precision in image dark regions. This affects 3D lighting computations as well as textured alpha-blending, because they rely on linear texture color input. For this reason EB GUIDE employs gamma-correct rendering to counter these effects as shown in [figure 6.16, “Example of an sRGB textures”](#).

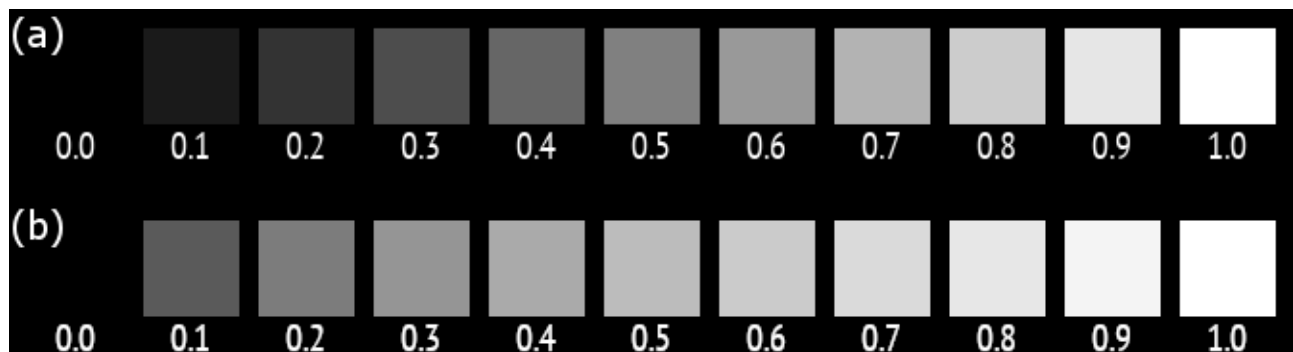


Figure 6.15. Example of gamma

**(a)** Values below the colored squares denote gray levels. Due to the non-linear relationship between color values and displayed luminance, 50% brightness is approximately reached at a 0.7 gray level for a correctly calibrated monitor. Note that despite the non-linear luminance response of the screen, the relative differences are perceptually uniform.

**(b)** Color values that have been gamma encoded prior to display. Because the encoding gamma cancels out the display's gamma, 50% brightness is reached at a 0.5 gray level.

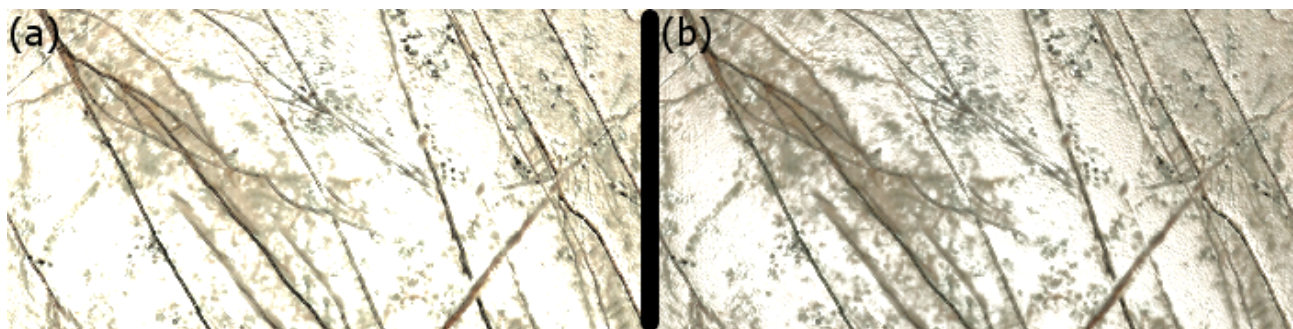


Figure 6.16. Example of an sRGB textures

**(a)** sRGB texture remains uncorrected in lighting computations and output is not gamma corrected. The lighting is oversaturated and details are washed out.

**(b)** sRGB textures are linearized before lighting and the result is gamma corrected. Details and surface structure become visible.

## 6.13.2. Gamma correction in EB GUIDE Studio

In EB GUIDE Studio gamma-correct rendering requires from you to configure the rendering pipeline as follows:

- ▶ The output configuration controls gamma encoding for the display itself.
- ▶ The input configuration tells EB GUIDE Studio which image and texture resources shall be treated as sRGB images in order for the rendering pipeline to properly linearize them for rendering operations.

To configure the input encoding, it has to be configured for each image or texture that is used. Note that the image format itself does not provide the information about sRGB encoding. You need to acquire this information beforehand. For instructions, see [section 14.8, “Tutorial: Rendering gamma correctly”](#)

## 6.14. Image-based lighting

Image-based lighting (IBL) is a technique that makes it possible to use an image as light for 3D objects. In EB GUIDE Studio, IBL is applied with the image-based light widget. You can apply this widget to scene graph nodes. It is not possible to have more than one image-based light per scene graph. If you add more than one, only the first image-based light in the hierarchy is used in the scene.



Figure 6.17. Example of image based lighting. Left: A teapot with a ceramic PBR GGX material lit by three-point-light sources. Center: Using image-based lighting (IBL), the teapot is lit by a virtual environment and the ceramic PBR GGX material appears realistic. Right: Additionally, textures are used to spatially vary the material parameters.

For more information on the ibl datatype, see [section 15.3.7, “ibl”](#). For more information on the Image-based light widget, see [section 15.10.3.4, “Image-based light”](#).

### 6.14.1. IBLGenerator, file formats and importing

Storing lighting information requires an image format that supports high dynamic range image data. EB GUIDE Studio supports two IBL formats:

- ▶ Portable float map (.pfm)
- ▶ RGBE (.hdr)

For the RGBE format, EB GUIDE does not support the XYZ color space. Only the -Y +X orientation is supported.

To use these IBL files in EB GUIDE Studio, you need to transform them into the `.ebibl` format. Do this with IBLGenerator. IBLGenerator is controlled through the command line and it is included in your installation in `$GUIDE_INSTALL_PATH\tools`. For instructions, see [section 8.2.6, “Importing IBL files”](#).

IBL files can contain images in either cube, sphere, or latitude-longitude parametrization. In IBLGenerator you can choose the parametrization type. To see all of the options, that IBLGenerator provides, type the following in the command line: `IBLGenerator.exe -h`

## 6.14.2. Limitations to IBL with an OpenGL renderer

An OpenGL 3 renderer always supports IBL. But if you use the OpenGL renderer, your OpenGL ES 2.0 driver must support the following OpenGL extensions. If one of the following extensions is not supported, the image-based light widget is ignored:

- ▶ `GL_EXT_shader_texture_lod`
- ▶ `GL_EXT_texture_rg`
- ▶ `GL_OES_texture_float`
- ▶ `GL_OES_texture_half_float`

## 6.15. Languages

### 6.15.1. Display languages in EB GUIDE Studio

EB GUIDE Studio offers different display languages for the graphical user interface. You select the display language in the project center, in the tab **Options**.

For instructions, see [section 10.6, “Changing the display language of EB GUIDE Studio”](#).

### 6.15.2. Languages in the EB GUIDE model

Most human machine interfaces offer the possibility to display texts in the user's preferred language. Such language management is also provided by EB GUIDE Studio.

In the project configuration you add a language for your EB GUIDE model. You can then export the texts, send them to your localization service provider and import the translated texts back into your EB GUIDE model.

It is possible to add language support to all datapool item types and thus enable your EB GUIDE model to display texts in different languages. A datapool item defines a value for each language. The language of the exported EB GUIDE model can be changed during run-time.

#### NOTE



#### No skin support available

When you have defined a language support for a datapool item, it is not possible to add a skin support to the same item.

For more information, see [section 8.6.1, “Adding a language to the EB GUIDE model”](#) and [section 6.15.3, “Export and import of language-dependent texts”](#).



#### Example 6.6.

#### Language-dependent texts for a multilingual user interface

In the project configuration three languages are added: English, German, and French. You can now model a label that changes, when the language of the user interface is changed. For this, link the label's `text` property to a datapool item with the value `Welcome` in English and the values `Willkommen` in German and `Bienvenue` in French.

For instructions, see [section 14.6, “Tutorial: Adding a language-dependent text to a datapool item”](#).

When you export an EB GUIDE model, the export also contains the languages. They are listed in file `languages.h`. Usually, all of the languages in the default set are exported. The default set lists all of the languages in the EB GUIDE model. If you want to export only specific languages instead of all of them, you can define an export set. An export set defines which sub-set of languages are exported. For instructions, see [section 8.6.4, “Creating an export set for languages”](#).

## 6.15.3. Export and import of language-dependent texts

Use the export and import functionality in EB GUIDE Studio to export, edit, translate, and import all language-dependent texts. The texts are exported to an `.xliff` file. `.xliff` (XML Localization Interchange File Format) is an XML-based format to store extracted text and carry the data from one step to another in the localization process. The `.xliff` can be sent to your localization service provider and understood by any translation tool.

After translation, you import the translated `.xliff` file back into your EB GUIDE model.

For instructions, see [section 10.8, “Exporting and importing language-dependent texts”](#).

The `.xliff` file is structured as follows:

- ▶ The header contains metadata about the source and target language:
  - ▶ The `source-language` and the `target-language` tag consist of a language code that follows the ISO 639 standards for representing language names and a language region that follows ISO 3166-1 standards for representing country codes. For example, the `en-US` language tag means `English` language in region `United States`.

- ▶ A unique alphanumeric `sourcelanguageid` and `targetlanguageid` is created for every project and language pair. These IDs prevent unintentional import of an `.xliff` file from another project or target language.
- ▶ The `trans-unit` elements contain the localizable data. Each `trans-unit` element holds a `source` element to store the source text, and a `target` element to store the translated text. When a new language is added to the EB GUIDE model, the `target` elements are filled with the source language. Therefore, when exporting an `.xliff` file, all the `target` elements that were not translated yet show the source language.

## 6.16. Linking

In many cases elements of an EB GUIDE model depend on other elements. For example, you want to make sure that some elements have exactly the same height, or when elements change in a scene you want other elements to change in the same way. For this purpose, values can be linked. This section explains what linking means and which limitations you need to consider.

For more information, see [section 6.7, “Datapool”](#) and [section 6.26.3, “Widget properties”](#).

The following applies:

- ▶ Model elements that can share data are datapool items and widgets using their properties.
- ▶ Model elements that you want to link must have the same data type. For example, you can link a datapool item of type integer only to another datapool item of type integer. Data type can be a simple data type, such as integer or boolean, or of a list type, such as integer list or boolean list. For more information, see [section 15.3, “Data types”](#).
- ▶ You can link to a list element. List elements that you want to link must have the same data type. For example, you can link a datapool item of type integer to a list element of another datapool item of type integer list.
- ▶ You cannot link model elements of type Conditional script.
- ▶ You cannot link scripted values.
- ▶ You cannot link a widget property to a widget property in a different view.
- ▶ You cannot link to the direct child widgets of an instantiator. It cannot be assured that the children of an instantiator exist at all times.



### Example 6.7. Linking Instantiator

This example shows you which possibilities there are to link to or from the Instantiator or its child widgets.



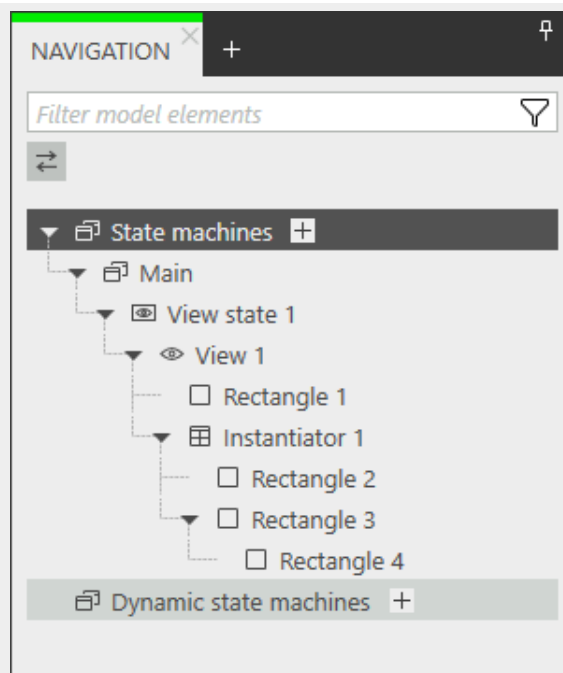


Figure 6.18. Widget tree

You can link as follows:

- ▶ A property of `Rectangle 1` can be linked to a property of `Instantiator 1`.
- ▶ A property of `Rectangle 2` can be linked to a property of `Rectangle 1`.
- ▶ A property of `Rectangle 4` can be linked to a property of `Rectangle 3`.

You cannot link the following properties:

- ▶ A property of `Rectangle 1` cannot be linked to a property of `Rectangle 2`, `Rectangle 3`, or `Rectangle 4`.
- ▶ A property of `Rectangle 2` cannot be linked to a property of `Rectangle 3`.
- ▶ A property of `Rectangle 4` cannot be linked to a property of `Rectangle 2`.

EB GUIDE Studio supports bidirectional linking, this means data can be changed both from the source element and from the target element. When one of the elements changes, so does the other element. But this behavior depends on the data type of the linked elements. Consider the following:

- ▶ If the linked model element are both either of a simple data type or of a list type, you can change their value directly in the EB GUIDE Studio UI. For instructions, see [section 8.3.3, “Linking between widget properties”](#) and [section 8.3.4, “Linking a widget property to a datapool item”](#).

- ▶ If the linked model elements are of a simple data type and of a list type, you can change their value in the EB GUIDE Studio UI only from the side of the list type element. To change their value from the side of the simple type element, use EB GUIDE Script. For instructions, see [section 8.3.5, “Linking to a list element”](#).




#### NOTE

#### Storage format



If a widget property is linked to a datapool item, the value is stored in the `$GUIDE_PROJECT_PATH/datapool.gdata` file. In the `$GUIDE_PROJECT_PATH/statemachines` folder, the `views/*.gdata` files contain only references to the datapool item. For this `objectref<type>` is used.

The following applies to the linking direction:

- ▶ An element can only be the source of one link, but the target of many links. The source widget property in the **Properties** component has either  or  next to it. The source datapool item in the **Datapool** component has  next to it.
- ▶ A datapool item can link to a widget property and to another datapool item at the same time. In this case, the datapool item is the target of two links.
- ▶ You cannot link one widget property to another widget property and a datapool item at the same time. Adding a link to a datapool item deletes the link to a widget property and vice versa.

The following image shows an example overview for linked model elements.

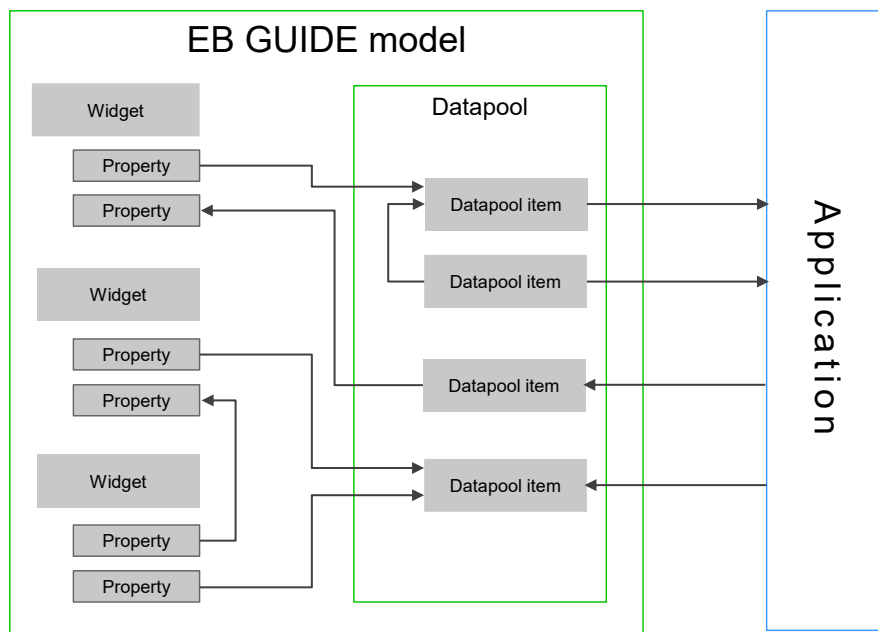


Figure 6.19. Example: Linked elements overview within a model and between a model and an application

## 6.17. Namespaces

In EB GUIDE Studio, with namespaces you create groups of model elements like datapool items and events. These groups have usually a defined functionality. Each namespace creates a naming scope for model elements so that model elements in different namespaces can have the same name.

Each model element belongs exactly to one namespace.

The root namespace is the default namespace and can neither be deleted nor renamed. The root namespace has the same name as the EB GUIDE project. All other namespaces are derived from this namespace. Model elements are always added to the default namespace in the following cases:

- ▶ If from the context menu you select **Add link to a datapool item** and create a new datapool item
- ▶ If you create an event in the **Trigger** combo box for a transition or an internal transition

You can move model elements between the namespaces.

#### NOTE



#### Moving model elements

If you move model elements from one namespace to another and the target namespace already contains an element with the same name, the move operation is not successful and an error message is shown.



#### Example 6.8. Namespace tree

In [figure 6.20, “Example of a namespace tree”](#), an example for a namespace tree is shown. The `myProject` namespace is the default namespace and also the name of the EB GUIDE project. Some namespaces are nested inside other namespaces.

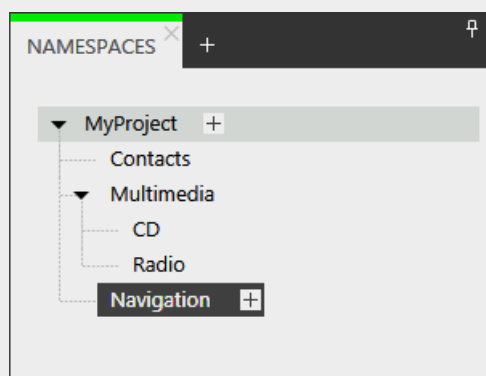


Figure 6.20. Example of a namespace tree

For more information, see [section 6.6, “Components of the graphical user interface”](#).

For instructions, see [section 9.1, “Working with namespaces”](#).

## 6.18. Model interfaces

With EB GUIDE it is possible to have an HMI that consists of multiple EB GUIDE models that can be developed, tested, maintained, and run separately. To make this possible, you can export one or more interfaces of an EB GUIDE model. These interfaces can then be imported into other EB GUIDE models. You can import several model interfaces from different EB GUIDE models.

Fundamentally, these interfaces consist of events and datapool items. Events and datapool items are what makes the communication between models possible. You can define the events and datapool items that the interface consists of.

Every model has an empty default model interface. But you can create and define a model interface yourself. Export and import is accomplished using `.json` files. Consider that an EB GUIDE model that imports an interface cannot change the imported interface.

---

### NOTE

#### Restrictions for scripted values




It is not possible to add scripted values to model or template interfaces.

---

For instructions on how to create an interface, how to add events and datapool items to a model interface, and how to import and export a model interface, see [section 10.9, “Working with model interfaces”](#).


### 6.18.1. Import of datapool items

An EB GUIDE model can read or write the value of the datapool items of an imported model interface, but it cannot rename or remove these datapool items. Imported datapool items do not have the  button.

If a datapool item does not belong to any namespace in the source model, during import a namespace named after the corresponding model interface is created and this datapool item is added to it.

For instructions on how to add a datapool item to a model interface, see [section 9.3.6, “Adding datapool items to a model interface”](#).

### 6.18.2. Import of events

An EB GUIDE model can trigger events of an imported model interface, but it cannot modify, rename, or remove these events. Imported events do not have the  button.

---

**NOTE**



**Duplicate event IDs**

Event IDs in an event group must be unique. When you import several model interfaces, validation errors occur in case there are duplicate IDs of events that belong to different model interfaces but are within the same event group.

Since you cannot change the event IDs of the imported events in EB GUIDE Studio, revert the import, change the event IDs in the source model, export and import again. It is advisable to define event ID ranges for all EB GUIDE models beforehand.

---

If an event does not belong to any namespace in the source model, during import a namespace named after the corresponding model interface is created and this event is added to it.

For instructions on how to add an event to a model interface, see [section 9.2.5, “Adding events to a model interface”](#).

### 6.18.3. Import of event groups

When you import a model interface with event groups, ownership of event groups is shared with the model where the interface is imported. So event groups have a special handling in connection with model interfaces:

- ▶ It is not possible to change an event group for an event that is part of an imported model interface.
- ▶ It is not possible to delete an event group that is used by at least one event that is part of an imported model interface.
- ▶ When you delete a model interface, event groups that were imported with this interface are not deleted.
- ▶ When you update and reimport a model interface with renamed event groups, the event groups are renamed in the model where you are importing the interface. Also the event group IDs are updated.
- ▶ When you import an event group with an ID that matches an already existing event group, these event groups are not combined. The reason for this is that the UUIDs (universally unique identifiers) of these event groups in the source files are different from the event group IDs that are used in the EB GUIDE Studio UI. UUIDs are used as the main distinguishing ID in the model.

### 6.18.4. Import of namespaces

If the imported model interface has events or datapool items that belong to specific namespace, these namespaces are also imported. These namespaces are read-only. This means that the following restrictions apply:

- ▶ You cannot change the names and the content, i.e. datapool items or events, of the imported namespaces.
- ▶ You cannot delete the imported namespaces.
- ▶ You cannot add sub-namespaces to the imported namespaces.

- ▶ You cannot move any datapool item or an event to the imported namespaces.

## 6.19. Photoshop file format support

EB GUIDE Studio supports all common `.psd` file formats. The supported color spaces are 8-bit, 16-bit, and 32-bit RGB as well as CMYK. You can import a `.psd` file directly or you can extract the images from the `-psd` file. `.psb` files are not supported.

### Importing

Elements from the `.psd` file are put directly into your model and a widget tree is created. The widget tree consists of containers, images, and labels derived from the layers of the `.psd` file. For instructions, see [section 8.2.4, “Importing a `.psd` file to a View”](#). Note the following:

- ▶ If a layer in the `.psd` file is set to invisible, the check box next to the `visible` property of the corresponding container or image is cleared.
- ▶ Text layers in `.psd` files are imported as Label.
- ▶ Image layers in `.psd` files are imported as Image.
- ▶ Group layers in `.psd` files are imported as Container. Containers are named after the group layers. Containers can contain images, labels, or other containers.

### Extracting

A subfolder is created that contains the images from the `.psd` file but the EB GUIDE model that you are working on is not changed. For instructions, see [section 8.2.5, “Extracting images from a `.psd` file”](#).

### Limitations

EB GUIDE Studio does not support the following features of the Photoshop file format:

- ▶ Layer effects, filters, and textures
- ▶ Color models other than RGB or CMYK
- ▶ Masks
- ▶ Multiple masks applied to a layer (layer mask and vector mask)
- ▶ Text styling and fonts
- ▶ Only color channels are used

## 6.20. Resource management

Resources are content that is not created within EB GUIDE but is required by your projects. Locate all resources of an EB GUIDE project in the resources folder.

The resources folder is located at `$GUIDE_PROJECT_PATH/<project name>/resources`.

EB GUIDE supports the following types of resource files:

1. Fonts
2. .ebibl file format for 3D graphics
3. Images
4. Meshes for 3D graphics
5. .psd file format

To use resources in the project, add the resource files to `$GUIDE_PROJECT_PATH/<project name>/resources`.

## 6.20.1. Fonts

To use a font in the project, add the font to `$GUIDE_PROJECT_PATH/<project name>/resources`.

Supported font types are TrueType fonts (\*.ttf, \*.ttc), OpenType fonts (\*.otf), and bitmap fonts (\*.fnt).

### Replacement character

The replacement character is used in case the dedicated font character is not found in the current font. The default replacement character is the Unicode character `OFFFD`. Note that different fonts have different representation of this character, e.g. a rectangle, a question mark, a space.

If you want to define the replacement character, define the value of the `gtf.model.textengine.replace-mentGlyph` configuration item in the `model.json` configuration file. For more information on configuration files, see the EB GUIDE TF user documentation.

For instructions on how to work with fonts, see [section 8.5, “Changing the Label settings”](#).

#### 6.20.1.1. Bitmap fonts

EB GUIDE Studio supports the \*.fnt bitmap fonts from Angelcode in version 3.0. To create a bitmap font, use a third-party font generator, for example Angelcode Bitmap Font Generator. For more information, see <http://www.angelcode.com>.

Make sure that the generated font has the following settings:

- ▶ The desired font size is defined.
- ▶ The character set is Unicode.
- ▶ The font descriptor is binary.

- ▶ The textures are provided as 8-bit .png files.

Note the following:

- ▶ In EB GUIDE Studio you are not able to change the font size of a bitmap font using the `font` property of a label. That means that you need to define the size when you generate your .fnt font.
- ▶ The **Stroke** widget feature does not apply to bitmap fonts. If you need a specific outline for your font, define it when you generate your .fnt font.
- ▶ In the `$GUIDE_PROJECT_PATH/resources` folder, create a subfolder for your .fnt bitmap font and .png texture files that you generated with a third-party tool. EB GUIDE Studio expects to find the .png files in same folder as the .fnt file.

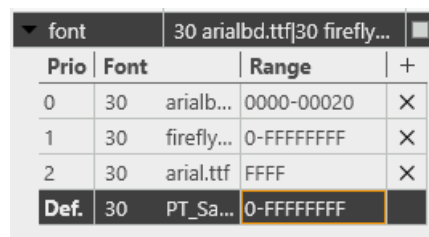
If you have several bitmap fonts, create a subfolder for each of these fonts.

### 6.20.1.2. Multifont support

In EB GUIDE Studio you can create your own font combinations using the multifont support. This feature is useful, for example, if the font that you selected does not provide all necessary characters. In this case you can replace missing characters with characters from a different font.

The multifont support can be added to the following model elements:

- ▶ Properties of type font and entries of font list
- ▶ User-defined properties of type font and entries of font list
- ▶ Datapool items of type font or entries of font list



font		30 arialbd.ttf30 firefly...	
Prio	Font	Range	+
0	30 arialb...	0000-00020	×
1	30 firefly...	0-FFFFFFFF	×
2	30 arial.ttf	FFFF	×
Def.	30 PT_Sa...	0-FFFFFFFF	

Figure 6.21. Example of a property of type font with multifont support

You define the following:

#### Priority

Priority defines which font entry should be used in case the Unicode character fits into more than one range. The lowest number has the highest priority for the evaluation. In case the character does not fit in any range, the default entry is used.

#### Font

Select the font that is available in `$GUIDE_PROJECT_PATH/resources` and define the size of the selected font.



## Range

You can define the Unicode character range, for which you want to use the specified font, as follows:

- ▶ With a single Unicode character, for example 0000.
- ▶ With several Unicode characters separated by comma, for example 0000, 0001.
- ▶ With a range of Unicode characters, for example 0000-FFFF.
- ▶ With several ranges separated by comma, for example 0000-0022, 0045-0055.

The characters are specified using the hexadecimal number format.

## Default font

When you add multifont support, automatically a default value is added. You cannot delete the default value nor edit its priority and range. However, you can edit size and font for the default value.

As the baseline of the label, the baseline of the default font is used.

## Performance

If a label has a multifont support added, the text is split internally into smaller parts depending on how many fonts are used. Each part is shaped separately using the defined font and size, then combined to one label and rendered together. This means that the number of the character ranges and the number and type of fonts affects the performance of your EB GUIDE model:

- ▶ If you define many ranges, it takes longer to find out to which font the current character belongs. The search is performed top-down according to the priority.
- ▶ Mixing many fonts and alternating between them leads to an increase of complexity.

Note the following:

- ▶ For better performance, instead of defining a font for a single Unicode character use character ranges or combination of character ranges. For example use 0000, 0001 or 0000-FFFF.
- ▶ For the font that is used for the most characters, define the highest priority.

---

### NOTE



### Valid font

EB GUIDE Studio does not validate font files. In case the font definition is faulty, the character may not be rendered at all.

---

For instructions on how to use the multifont support, see [section 8.5.5, “Managing multifont support”](#).

## 6.20.2. Image-based lighting for 3D graphics

It is possible to use image-based lighting in EB GUIDE Studio. The external command line tool IBLGenerator takes a `.pfm` or `.hdr` file as input data and creates an `.ebibl` file which represents an IBL resource. The IBL resource is used by the `ibl` property of the image-based light widget.

For instruction, about how to get an `.ebibl` file, see [section 8.2.6, “Importing IBL files”](#).

For background information, see [section 6.14, “Image-based lighting”](#).

## 6.20.3. Images

To use an image in the project, add the image to `$GUIDE_PROJECT_PATH/<project name>/resources`. If you select an image from a different folder, the image is copied to the project folder.

The supported image formats are Portable Network Graphic (`*.png`), JPEG (`*.jpg`) and 9-patch images (`*.9.png`).

For instructions, see [section 8.2.2.3, “Adding an Image”](#).

### 6.20.3.1. 9-patch images

EB GUIDE Studio supports images with additional meta information according to the 9-patch image approach. 9-patch images are stretchable `.png` images. 9-patch images contain two black markers, one at the top and one at the left side of the image. Areas that are not marked are not scaled. Marked areas are scaled. Markers are not displayed in EB GUIDE Studio.

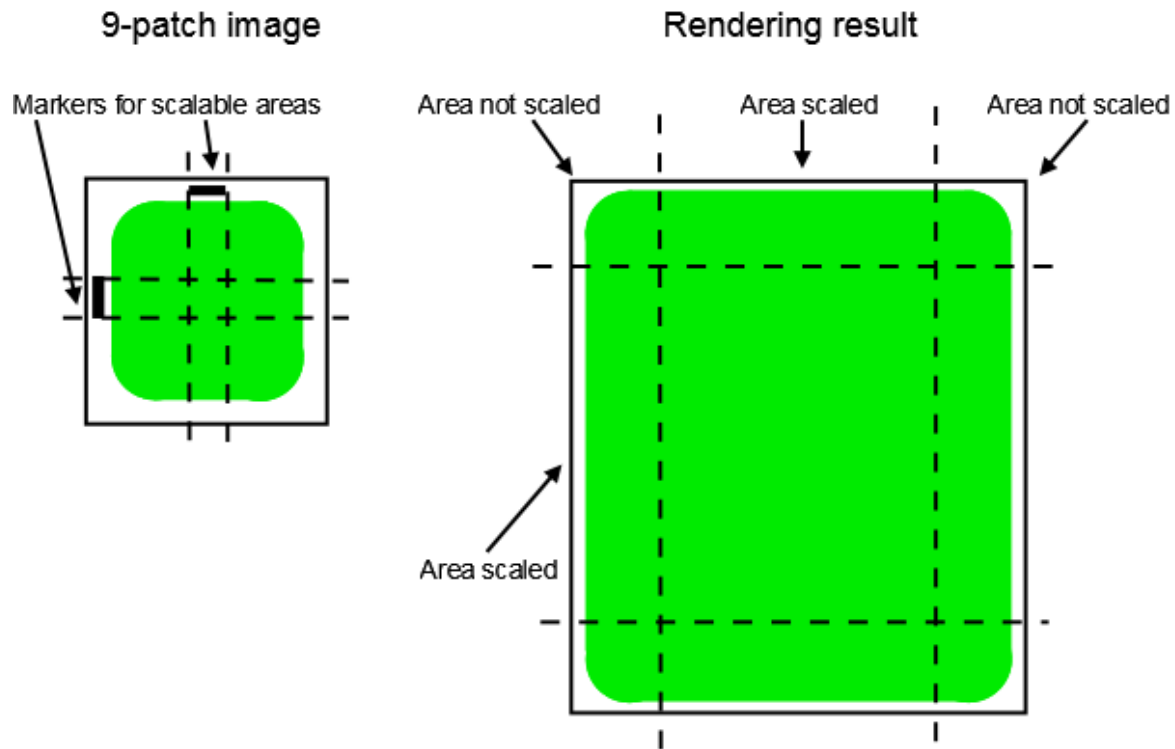


Figure 6.22. 9-patch example

When you work with 9-patch images, consider the following:

- ▶ 9-patch processing works only with the renderers for OpenGL ES 2.0 or higher.
- ▶ 9-patch processing works with .png images only.
- ▶ The \*.9.png extension is mandatory for 9-patch images.
- ▶ It is possible to specify none, one, or more than one marker at the top and the left side. The 9-patch definition also includes markers for text areas at the right side and at the bottom of the image. These markers are not evaluated in EB GUIDE Studio.

For instructions, see [section 8.2.2.3, “Adding an Image”](#).

## 6.20.4. Meshes for 3D graphics

It is possible to import 3D graphic files in EB GUIDE Studio. After you have imported a 3D graphic file in EB GUIDE Studio, in `$GUIDE_PROJECT_PATH/<project name>/resources`, you find a subfolder. Meshes as defined in the 3D graphic file are imported as .ebmesh files. For details, see [section 6.1.3, “Import of a 3D graphic file”](#).

For instructions, see [section 8.2.3.1, “Adding a Scene graph to a View”](#).

## 6.21. Scripting language EB GUIDE Script

EB GUIDE Script is the built-in scripting language of EB GUIDE. This chapter describes EB GUIDE Script language features, syntax, and usage.

### 6.21.1. Capabilities and areas of application

You can use EB GUIDE Script in a variety of places in a project, for example:

- ▶ In a widget property
- ▶ In the state machine as part of a transition or state
- ▶ In a datapool item

Not all features of EB GUIDE Script are available in all cases. For example access to local widget properties is only allowed when the script is part of a widget. Access to the datapool, on the other hand, is always allowed.

With EB GUIDE Script you can directly manipulate model elements, for example to do the following:

- ▶ Fire events
- ▶ Write datapool items
- ▶ Modify widget properties

### 6.21.2. Prefixes and identifiers

In EB GUIDE, it is possible to give identical names to different kinds of objects. For example, you can name both an event and a datapool item `Napoleon`. To make this possible, every identifier, i.e. name of an object, in EB GUIDE Script must have a prefix. The prefix defines the type of an object, followed by a colon.

The set of prefixes is fixed in EB GUIDE Script, you cannot introduce new prefixes. The following prefixes exist:

- ▶ `ev:` events
- ▶ `dp:` datapool items
- ▶ `f:` user-defined actions (foreign functions)
- ▶ `v:` local variables

For example, `ev:Napoleon` specifies the event named `Napoleon` while `dp:Napoleon` specifies the datapool item named `Napoleon`.

Identifiers without a prefix are string constants.

Identifiers in EB GUIDE contain many characters including spaces and punctuation. Thus it can be necessary to quote identifiers in EB GUIDE Script. If an identifier does not contain special characters, for example a valid C identifier consisting only of letters, numbers and underscores, it does not have to be quoted.



#### Example 6.9. Identifiers in EB GUIDE Script

```
dp:some_text = foo; // foo is a string here
dp:some_text = "foo"; // this statement is identical to the one above
dp:some_text = v:foo; // foo is the name of a local variable
// of course you can quote identifiers, even if it is not strictly necessary
dp:some_text = v:"foo";
// again, a string constant
dp:some_text = "string with spaces, and -- punctuation!";
// identifiers can also contain special characters, but you have to quote them
dp:some_text = v:"identifier % $ with spaces @ and punctuation!";
```

### 6.21.3. Comments

EB GUIDE Script has two kinds of comment: C style block comments and C++ style line comments. Block comments must not be nested.



#### Example 6.10. Comments in EB GUIDE Script

```
/* this is a C style block comment */
// this is a C++ style line comment
```

For every EB GUIDE Script comment that contains a string `todo`, EB GUIDE Studio shows a warning in the **Problems** component when you validate a project. Use this feature to mark all your open tasks and display them at a glance.

#### NOTE



#### Default comment for conditional scripts

By default, a datapool item or a property of type `Conditional script` contains a comment `// todo: auto generated return value, please adapt`. To eliminate the warning, delete the `todo` string from the comment once you entered the required EB GUIDE Script code.

### 6.21.4. Types

EB GUIDE Script is a strongly-typed and statically-typed programming language. Every expression has a well defined type. Supplying an unexpected type results in an error.

EB GUIDE Script supports the following types:

- ▶ Integer
- ▶ Unicode strings (string)
- ▶ Objects with reference counting
- ▶ Type definitions to the above listed types and to the following:
  - ▶ Color (integer for 32-bit RGBA value)
  - ▶ Boolean
  - ▶ IDs of different model elements: datapool items, views, state machines, pop-ups (all of type integer)
- ▶ Void, also known as the unit type. This type has a role as in functional programming, for example Haskell.
- ▶ Widget and event references. These are record types, the fields of which you may access by using the `dot` notation, as known in C or Java. You cannot directly create new objects of these kinds, they are created automatically where appropriate.

All types and type definitions are incompatible with each other and there are no typecasts. This feature ensures type safety once a script is successfully compiled.

## 6.21.5. Expressions

EB GUIDE Script is expression-based. Every language construct is an expression. You form larger expressions by combining smaller expressions with operators.

To evaluate an expression means to replace it by its value.



### Example 6.11. Evaluation of an integer value

```
1 + 2 // when this expression is evaluated, it yields the integer 3
```

## 6.21.6. Constants and references

The basic expressions are integer, color, boolean, and string constants and references to model elements.

The void type also has a value constant that can be written in two different but semantically equivalent ways:

- ▶ With the opening curly brace followed by the closing curly brace `{ }`
- ▶ With the keyword `unit`



### Example 6.12.

### Usage of constants

```
"hello world" // a string constant
true          // one of the two boolean constants
ev:back       // the event named "back" of type event_id
dp:scrollIndex // the datapool item named "scrollIndex",
               // the type is whichever type the dp item has
5             // integer constants have a dummy type "integer constant"
5::int        // typecast your constants to a concrete type!
color:255,255,255,255 // the color constant for white in RGBA format

// the following are two ways to express the same
                if( true )
{
}
else
{
}

if( true )
    unit
else
    unit
```

## 6.21.7. Arithmetic and logic expressions

EB GUIDE Script supports the following arithmetic expressions:

- ▶ Addition (+), subtraction (-), multiplication (\*), division (/), and modulo (%) can be applied to expressions of type integer.
- ▶ The logical operators or (||), and (&&), not (!) can be applied to expressions of type boolean.
- ▶ Integers and strings can be compared with the comparison operators greater-than (>), less-than (<), greater-than-or-equal (>=), less-than-or-equal (<=).
- ▶ Data types can be compared with the equality operators: equal to (==) and not equal to (!=).

Strings can be compared without case sensitivity with the equality operator (=Aa=).

#### NOTE



#### Availability of equality operators

Events and resource data types, for example 3D graphics, fonts and images, do not support the equality operators (==) and (!=).

- ▶ Strings can be concatenated with the (+) operator.



### Example 6.13. Arithmetic and logic expressions

```
10::int + 15::int // arithmetic expression of type int
dp:scrollIndex % 2 // arithmetic expression of type int,
                  // the concrete type depends on the type
                  // of dp:scrollIndex
"Morning Star" == "Evening Star" // type bool and value false (wait, what?)
"name" =Aa= "NAME" // type bool and value true
!true // type bool, value false
!(0 == 1) // type bool, value true
// as usual, parenthesis can be used to group expressions
((10 + dp:scrollIndex) >= 50) && (!dp:buttonClicked)
// string concatenation
"Napoleon thinks that " + "the moon is made of green cheese"
f:int2string(dp:speed) + " km/h" // another string concatenation
```

## 6.21.8. L-values and r-values

There are two kinds of expressions in EB GUIDE Script: *l-values* and *r-values*. L-values have an address and can occur on the left hand side of an assignment. R-values do not have an address and may never occur on the left hand side of an assignment.

- ▶ L-values are datapool references, local widget properties, and local variables.
- ▶ R-values are event parameters and constant expressions such as string or integer constants.

## 6.21.9. Local variables

The `let` expression introduces local variables. It consists of a list of variable declarations and the `in` expression, in which the variables are visible. Variables are l-values, you can use them on the left hand side of assignments. Variables have the prefix `v:.`. The syntax of the `let` expression is as follows:

```
let v:<identifier> = <expression> ;
    [ v:<identifier> = <expression> ; ]...
in
    <expression>
```

The type and value of the `let` expression are equal to the type and value of the `in` expression.

`let` expressions may be nested, variables of the outer `let` expressions are also visible in the inner expressions.





#### Example 6.14. Usage of the `let` expression

```
// assign 5 to the datapool item "Napoleon"
let v:x = 5 in dp:Napoleon = v:x;

// define several variables at once
let v:morning_star = "Venus";
    v:evening_star = "Venus";
in
    v:morning_star == v:evening_star; // Aha!

let v:x = 5;
    v:y = 20 * dp:foo;
in
{
    // Of course you may have a sequence as the in expression,
    // but parenthesis or braces are required then.
    v:x = v:y * 10;
    dp:foo = v:x;
}

// Because let expression also have types and values, we can have them
// at the right hand side of assignments.
dp:x = let v:sum = dp:x + dp:y + dp:z
        in v:sum; // this is the result
                // of the let expression

// A nested let expression
let v:x = dp:x + dp:y;
v:a = 5;
in
{
    let v:z = v:x + v:a;
    in
    {
        dp:x = v:z;
    }
}
```

## 6.21.10. While loops

`while` loops in EB GUIDE Script have a syntax similar to that in C or Java, they consist of a condition expression and a `do` expression. The syntax is as follows:

```
while (<condition expression> ) <do expression>
```

The `do` expression is evaluated repeatedly until the condition expression yields `false`. The condition expression must be of type `boolean`, the `do` expression must be of type `void`. The `while` expression is of type `void` and must not occur at the left or right hand side of an assignment.



**Example 6.15.**  
**Usage of the `while` loop**

```
// Assume dp:whaleInSight is of type bool
while( ! dp:whaleInSight )
{
    dp:whaleInSight = f:lookAtHorizon();
}
```

## 6.21.11. If-then-else

`if-then-else` in EB GUIDE Script behaves like the ternary conditional operator (`?:`) in C and Java.

The `if-then-else` expression consists of the following sub-expressions:

- ▶ condition expression
- ▶ then expression
- ▶ else expression

The syntax is as follows:

```
if ( < condition expression> ) <then expression> else <else expression>
```

`if-then-else` is processed as follows:

1. First, the condition expression is evaluated. It must be of type `boolean`.
2. If the condition is true, the then expression is evaluated.
3. If the condition is false, the `else` expression is evaluated.

`if-then-else` itself is an expression. The type of the whole expression is the type of the then expression and the `else` expression, which must be identical. The value of `if-then-else` expressions is either the value of the then expression, or the value of the `else` expression, in accordance with the rules above.

#### NOTE



#### Short-circuit evaluation

EB GUIDE Script does not support short-circuit evaluation (minimal evaluation).

If the `if` expression consists of several sub-conditions that are concatenated by `&&` or `||`, EB GUIDE Script, unlike some other programming languages, evaluates all sub-conditions. This means that, if a sub-condition is false and hence the whole condition is false, all sub-conditions will still be evaluated.

There is a special form of `if-then-else`, in which you may omit the `else` branch. This special form is of type `void` and cannot be used to return values from scripts.



#### Example 6.16. Usage of `if-then-else`

```
// Assume dp:whaleInSight is of type bool
// and dp:user is of type string.
if( dp:whaleInSight && dp:user == "Captain Ahab" )
{
    dp:mode = "insane";
}
else
{
    dp:mode = "normal";
}

// Because if-then-else is also an expression,
// we may simplify the previous example:
dp:mode = if( dp:whaleInSight && dp:user == "Captain Ahab" )
    "insane"
    else
    "normal"

if ( <expression> ) <expression> // This is the reduced way of
    writing if-then-else
//It is an alternative to the following
if( <expression> ) { <expression> ; {} } else {}
```

## 6.21.12. Foreign function calls

You can extend EB GUIDE Script with functions written in C, so-called foreign functions.

An identifier prefixed by `f:` is the name of a foreign function. Foreign functions have an argument list and a return value, as they do in C. The syntax of foreign function calls is as follows:

```
f:<identifier> ( <expression> [ , <expression> ] ... )
```



#### **Example 6.17.** **Calling foreign functions**

```
// write some text to the connection log
f:trace_string("hello world");
// display dp:some_index as the text of a label
v:this.text = f:int2string(dp:some_index);

// passing different parameters of matching type
f:int2string(v:this.x)
f:int2string(4)
f:int2string(dp:myInt)
f:int2string(v:myVar)

//passing parameters of different types
// starts an animation (parameter type GtfTypeRecord) from a script
// located in its parent widget
f:animation_play(v:this->Animation);

// checks the number of child widgets of a widget (parameter type widget)
f:widgetGetChildCount(v:this);

// traces debugging information about a datapool item (parameter type dp_id)
// to the connection log; uses the address of the datapool item as parameter
f:trace_dp(&dp:myFlag);
```

## 6.21.13. Datapool access

Scripts written in EB GUIDE Script can read and write datapool items. An identifier with the prefix `dp:` is called datapool item expression. Its type is datapool item of type `X`, where `X` is the type of the datapool entry it refers to. Identifier may include only the name of the datapool item, or, if the datapool item is not in the default namespace, the name of the namespace followed by the name of the datapool item.

If a datapool item of type `X` occurs on the left hand side of an assignment, and an expression of type `X` occurs on the right hand side of the assignment, the value of the datapool item is written.

If a datapool item occurs somewhere in a program but not on the left hand side of an assignment, the value of the datapool item is read.



#### **Example 6.18.** **Assignment of datapool values**

```
// Assume intA to be of type int. Assign 10 to it.
```

```
dp:intA = 10;
// Assume strA to be of type string. Assign the string "blah" to it.
dp:strA = blah; // Yes, we can omit the quotes, remember?
dp:strA = 42; // Error: integer cannot be assigned to string

// Assign the value of the datapool item intB to intA.
// Both datapool items must have the same type.
dp:intA = dp:intB;
// Multiply the value of intB by two and assign it to intA.
dp:intA = 2 * dp:intB;
// Use the value of a datapool item in an if-clause.
if( dp:speed > 100 )
{
    // ...
}
```

The following operators can be applied to the datapool items:

- ▶ The reference operator (&) can be applied to datapool items. It refers to the address of a datapool item rather than to its value. The reference operator is used in foreign function calls to pass parameters of type `dp_id`.
- ▶ The redirect-link operator (`=>`) changes the link target of a datapool item. Link source can only be a datapool item that was already linked.

## 6.21.14. Widget properties

If a script is part of a widget, it can access the properties of that widget. EB GUIDE Script creates a variable called `v:this` to access the properties using the dot notation.

A script is part of a widget if it is attached to a widget property, for example as an input reaction such as click or button press.



### Example 6.19. Setting widget properties

```
// assume this script is part of a widget
v:this.x = 10; // if the widget has an x-coordinate

v:this.text = "hello world"; // if the widget is a label and has a text property
// assume testEvent has one integer parameter
fire ev:testEvent(v:this.x);
```

If a script is part of a widget, it can also access properties of other widgets in the widget tree.

The go-to operator ( $\rightarrow$ ) is used to refer to other widgets within the widget tree. The syntax is as follows:

```
<expression> -> <expression>
```

The expression on the left hand side must refer to a widget and the expression on the right hand side must be a string, the name of a child widget. To navigate to the parent widget, use the symbol  $\wedge$  on the right hand side. The whole go-to expression refers to a widget.

Navigating the widget tree might affect run-time performance. Widgets are assigned to variables for the efficient manipulation of multiple properties.



#### **Example 6.20.** **Accessing widget properties**

```
v:this.x          // access the properties of the current widget
v:this->^.x       // access the x property of the parent widget
v:this->^->caption.text // access the text property of a label called caption,
                      // read: "go-to parent, go-to caption, text"

// Modify several properties of the caption.
// This way, the navigation to the caption is only performed once.
let v:cap = v:this->^->caption
in
{
  v:cap.textColor = color:0,0,0,255;
  v:cap.x += 1;
  v:cap.y += 1;
}
```

## 6.21.15. Lists

Datapool items and widget properties can hold lists. The subscript operator ( $[]$ ) accesses list elements. The syntax is as follows:

```
<expression> [ <expression> ]
```

The first expression must evaluate to a list type, the second expression must evaluate to an integer value. If the list is of type `list A`, the whole list subscript expression must be of type `A`.

If the list subscript expression occurs at the left hand side of an assignment, the value of the referred list element is written.

The `length` keyword returns the number of elements of a list. If it is put in front of a list expression, the whole expression must be of type integer.



#### **Example 6.21.**

### Lists

```
// Assume this widget is a label and dp:textList is a list of strings
v:this.text = dp:textList[3];

dp:textList[1] = v:this.text; // writing the value of the list element

v:this.width = length dp:textList; // checking the length of the list
dp:textList[length dp:textList - 1] = "the end is here";
```

Adding elements to and removing elements from lists is currently not supported in EB GUIDE Script.

Trying to access list elements beyond the end of a list stops the execution of the script immediately. Make sure that all your list accesses are in range.

## 6.21.16. Events

EB GUIDE Script offers the following expressions to handle events:

- ▶ The `fire` expression sends events. The syntax is as follows:

```
fire ev:<identifier> ( <parameter list> )
```

Events can, but do not need to have parameters. The parameter list of the `fire` expression must match the parameters of the fired event. If an event has no parameters, the parentheses must be empty.



### Example 6.22. Using the `fire` expression

```
fire ev:toggleView(); // the event "toggleView" has no parameters
fire ev:mouseClick(10, 20); // "mouseClick" has two integer parameters
fire ev:userNameEntered("Ishmael"); // string event parameter
```

- ▶ The `fire_delayed` expression sends events after a specified time delay. The syntax is as follows:

```
fire_delayed <time> , ev:<identifier> ( <parameter list> )
```

The `time` parameter is an integer value that specifies the delay in milliseconds.



### Example 6.23. Using the `fire_delayed` expression

```
fire_delayed 3000, ev:mouseClick(10, 20); // send the event "mouseClick"
//in 3 seconds.
```

- ▶ The `cancel_fire` expression cancels the delayed event. The syntax is as follows:

```
cancel_fire ev:<identifier>
```

- ▶ The `match_event` expression checks whether the execution of a script has been triggered by an event. The syntax is as follows:

```
match_event v:<identifier> = ev:<identifier>
in
    <expression>
else
    <expression>
```

The type of the `match_event` expression is the type of the `in` expression and the `else` expression, which must be identical.

There is a special form of the `match_event` expression, in which you can omit the `else` branch. This special form is of type `void` and cannot be used to return values from scripts.



#### **Example 6.24.** **Using the `match_event` expression**

```
match_event v:theEvent = ev:toggleView in
{
    // this code will be executed when the "toggleView" event
    // has triggered the script
    dp:infoText = "the view has been changed";
}
else {}

match_event ( <expression> ) in <expression> //special form
        //without an else branch
        //The special form is an alternative way to express the following
        match_event ( <expression> ) in { <expression> ; {} } else {}
```

Identifier may include only the name of the event, or, if the event is not in the default namespace, the name of the namespace followed by the name of the event.

If an EB GUIDE Script has been triggered by an event with parameters, the parameters are accessible in the `in` expression of a `match_event` expression. Read parameters using the dot notation, as you would access fields of a structure in C. Event parameters are not available in the `else` expression.



#### **Example 6.25.** **Event parameters**

```
// assume that "mouseClick" has two parameters: x and y
match_event v:event = ev:mouseClick in
{
    dp:rectX = v:event.x;
    dp:rectY = v:event.y;
```



```
}
```

## 6.21.17. String formatting

String formatting in EB GUIDE Script is done using the concatenation operator (+) on strings in combination with various data-to-string conversion functions. The EB GUIDE Script standard library comes with the `int2string` function for simple integer-to-string conversion.



### Example 6.26. String formatting

```
// Assume this widget is a label and has a text property.  
// Further assume that the datapool item dp:time_hour and  
// dp:time_minute hold the current time.  
v:this.text = "the current time is: " + f:int2string(dp:time_hour)  
            + ":" + f:int2string(dp:time_minute);
```

## 6.21.18. The standard library

EB GUIDE Script comes with a standard library that consists of a set of foreign functions for example as follows:

- ▶ String formatting
- ▶ Language management
- ▶ Tracing
- ▶ Time and date
- ▶ Random number generation

For details, see [section 15.4.3, “EB GUIDE Script standard library”](#).

## 6.22. Scripted values

A scripted value is an alternative notation for the value of a widget property or a datapool item. Such properties of widgets or datapool items use other model elements to evaluate their own value or to react on events or property updates. Scripted values are written in the EB GUIDE Script scripting language.

A property in EB GUIDE can be converted to a scripted value and back to its plain value.

For instructions, see [section 9.3.3, “Converting a property to a scripted value”](#).

## NOTE



### Restrictions for scripted values

It is not possible to add scripted values to model or template interfaces.

For editing a scripted value, EB GUIDE Studio contains an EB GUIDE Script editor which is divided into different categories.

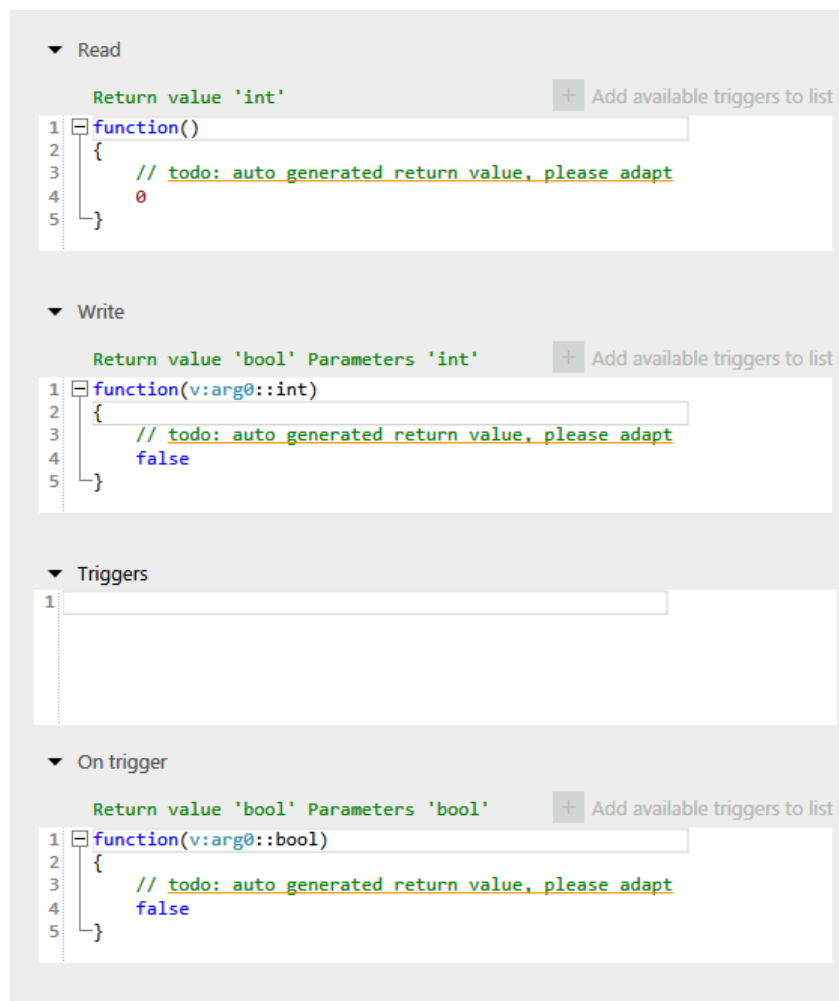


Figure 6.23. EB GUIDE Script editor in EB GUIDE Studio

- ▶ The **Read** script is called when the scripted value property is read. If the property is of type list, the parameters include the list index.

The return value of the **Read** script represents the current value of the property.

- ▶ The **Write** script is called when the scripted value property is written.

The new property value is a parameter of the **Write** script. If the property is of type list, the parameters includes the list index.

The return value of the **Write** script controls change notifications for the property.

- ▶ true: trigger a change notification
- ▶ false: do not trigger a change notification
- ▶ The **Triggers** script contains a list of events, datapool items and widget properties that trigger the execution of the **On trigger** script.

Clicking on **Add available triggers to list** will add all triggers highlighted in the corresponding script to the trigger script.

- ▶ The **On trigger** script is called on initialization, after an event trigger or after a property update.

The parameter of the **On trigger** script indicates the cause for the execution of the script. Execution can be caused by initialization or by one of the triggers in the **Triggers** list.

The return value of the **On trigger** script controls change notifications for the property.

- ▶ true: trigger a change notification
- ▶ false: do not trigger a change notification
- ▶ The **Length** script is only available for properties of type list.

The return value of the **Length** script represents the current length of the list.

## 6.23. Skins

Skins allow you to define different user interfaces by defining different datapool values for the same EB GUIDE model. This way you can define various looks for the same HMI as for example skins for night and day mode.

You can switch between the skins during run-time to see the effect of the different datapool values.

Skin support is only available for plain datapool values and cannot be used for scripted values or linked datapool items.

### NOTE



#### No language support available

When you have defined a skin support for a datapool item, it is not possible to add a language support to the same item.

For instructions see [section 8.7, “Working with skin support”](#).

When you export an EB GUIDE model, the export also contains the skins. They are listed in file `skins.h`. Usually, all of the skins in the default set are exported. The default set lists all of the skins in the EB GUIDE model. If you want to export only specific skins instead of all of them, you can define an export set. An export

set defines which sub-set of skins are exported. For instructions, see [section 8.7.5, “Creating an export set for skins”](#).

## 6.24. State machines and states

A state machine is a deterministic finite automaton and describes the dynamic behavior of a system. In EB GUIDE, a state machine consists of an arbitrary number of hierarchically ordered states and of transitions between these states. State machines and states are fundamental features of modeling with EB GUIDE. Applying them correctly makes it easier to handle EB GUIDE models of large complexity.

To structure your EB GUIDE model, separate the behavior that you want to implement into different state machines. Structuring your model reduces the complexity of your code, makes your model easier to understand for modelers, improves the handling of the widget tree, and therefore improves the performance of the EB GUIDE model.

The dynamic nature of state-of-the-art HMIs, their large number of widgets, images, modules, 3D-objects, and input options can be handled with dynamic state machines. For more information about dynamic state machines, see the sections below.

For instructions, see [section 7.1, “Modeling a state machine”](#).

The following types of state machines exist:

### Haptic state machines

Describe elements of EB GUIDE models that are visible in the GUI, such as labels, images, or rectangles.

### Logic state machines

Describe elements of EB GUIDE models that are not visible in the GUI, such as processes that run independently from the GUI. For example, you could run an incremental search in a logic state machine, while the user is typing letters in an entry field in the GUI.

Logic state machines cannot contain view states. View states are disabled for logic state machines.

In an EB GUIDE model, logic and haptic state machines can be implemented as main state machines or dynamic state machines.

### Main state machines

Are created automatically when you create a new project. They drive the behavior of a scene. Without a main state machine there is no behavior and therefore no scene.

### Dynamic state machines

Are used to model pop-ups that are displayed on top of all other layers. They can be activated as a child of a main state machine of a fitting type. For more information, see [section 6.24.1, “Dynamic state machines”](#). For instructions, see [section 7.2.2, “Adding a dynamic state machine”](#).

The following combinations exist for state machines:

Table 6.3. Types of state machines

	Main	Dynamic
<b>Haptic</b>	Main-haptic  Main state machine with haptic and logic behavior.	Dynamic-haptic  Dynamic state machine with haptic and logic behavior.
<b>Logic</b>	Main-logic  Main state machine with logic behavior.	Dynamic-logic  Dynamic state machine with logic behavior.

## 6.24.1. Dynamic state machines

Dynamic state machines are used to create pop-ups and are also useful to give an HMI a modern, dynamic look and feel. For example, you can use them to create overlapping, moving views.

Consider the following facts:

- ▶ Dynamic state machines are displayed on top of other state machines.
- ▶ Dynamic state machines are started and stopped by other state machines.
- ▶ Dynamic state machines run in parallel to other state machines.
- ▶ Dynamic state machines separate the widgets and the behavior that belongs to pop-ups.

The following EB GUIDE Script functions are used to control state machines:

- ▶ `pushDynamicStateMachine`
- ▶ `popDynamicStateMachine`
- ▶ `changeDynamicStateMachinePriority`
- ▶ `isDynamicStateMachineActive`



### Example 6.27. Dynamic state machines

This example shows the work flow of a dynamic state machine. There are three dynamic state machines and they are supposed to be displayed in the following order:

- ▶ `PopUp_error`
- ▶ `PopUp_repeat`
- ▶ `PopUp_return`

All three state machines are started. `pushDynamicStateMachine` is called three times with the following parameters:

1. `stack 1, sm PopUp_error, priority 0`
2. `stack 1, sm PopUp_repeat, priority 1`
3. `stack 1, sm PopUp_return, priority 2`

This results in the following order of state machines:

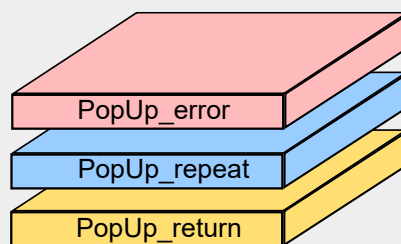


Figure 6.24. Resulting order of dynamic state machines

To change the order of state machines `changeDynamicStateMachinePriority` is called with the following parameters:

1. `stack 1, sm PopUp_error, priority 0`
2. `stack 1, sm PopUp_repeat, priority 2`
3. `stack 1, sm PopUp_return, priority 1`

This results in the following order of state machines:

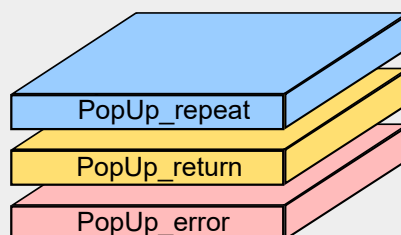


Figure 6.25. New order of dynamic state machines

## 6.24.2. States

States determine the status and behavior of a state machine. Different types of states exist. Transitions link states. They are the connection between states and define a state change from a source state to a target state. In EB GUIDE Studio transitions are indicated by arrows. For instructions, see [section 7.4, “Connecting states through transitions”](#).

The types of states and transitions that are available in EB GUIDE Studio are explained in the following sections.

#### 6.24.2.1. Initial state

Initial states define the state in a state machine that is entered first. They also define which state is entered first in a compound state. Initial states have only one outgoing transition that is called default transition. The default transition triggers the first state in the state machine. Initial states have no incoming transitions.

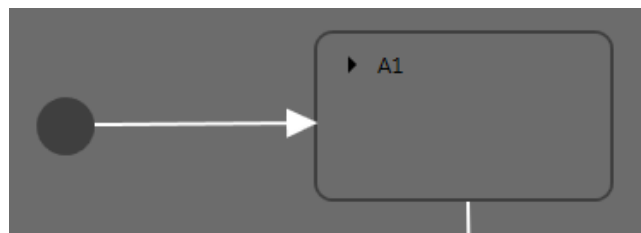


Figure 6.26. Example Initial state

#### 6.24.2.2. View state

View states contain views. A view represents a project specific HMI screen. The view is displayed while the corresponding view state is active. The view consists of widgets which are the interface between user and system.

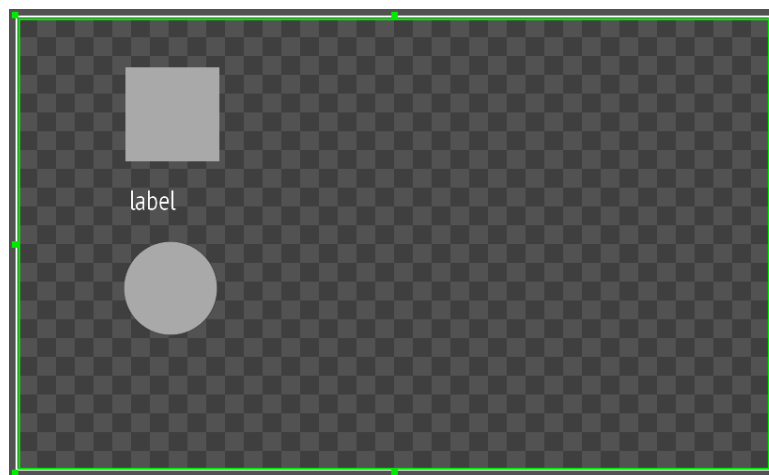


Figure 6.27. Example View state

#### 6.24.2.3. Compound state

A compound state can have other states within it as child states. The compound state structure is hierarchical and the number of possible child states is arbitrary. Any type of state can be nested in a compound state.

A compound state can have an arbitrary number of incoming and outgoing transitions, and of internal transitions. Child states inherit the transitions of parent states.

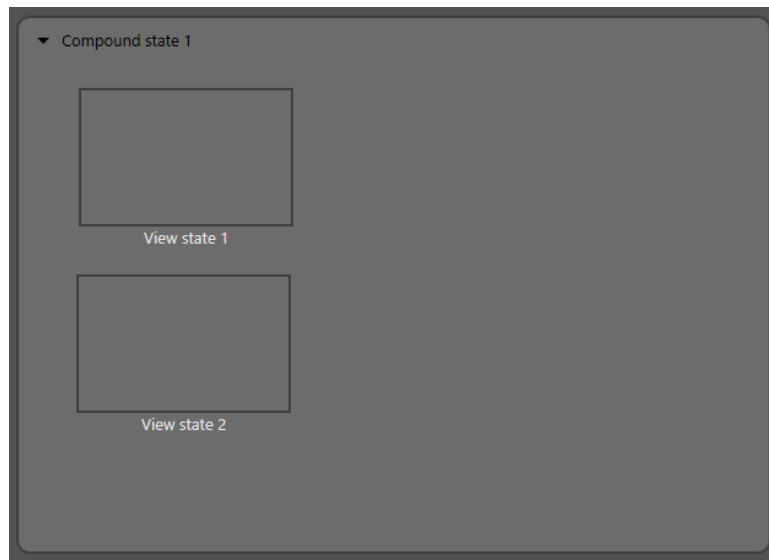


Figure 6.28. Example Compound state

In the **Navigation** component, the state hierarchy is shown as a tree structure.

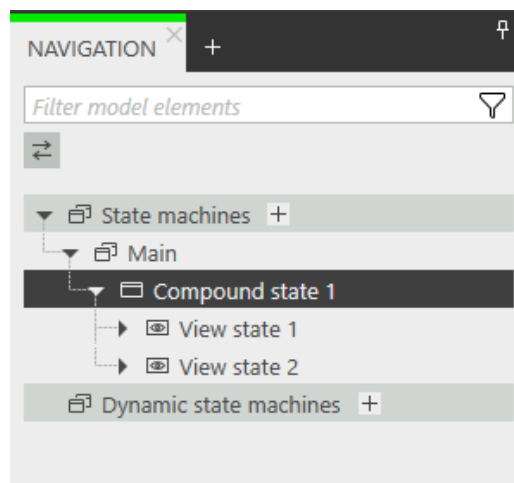


Figure 6.29. State hierarchy in the **Navigation** component

#### 6.24.2.4. Choice state

Choice states realize a dynamic conditional branch. They are used when firing an event depends on conditions. A choice state is the connection between a source state and a target state. A choice state can have several



incoming and outgoing transitions. Every outgoing transition is assigned a condition and is only executed if the condition evaluates to `true`. One outgoing transition is the `else` transition. It is executed if all other conditions evaluate to `false`. This `else` transition is mandatory.

In case more than one of the outgoing transitions are true, the priority of the transitions determines which transition is executed. The priority is defined by the order of the transitions in the widget tree. You can change the priority by changing the order.

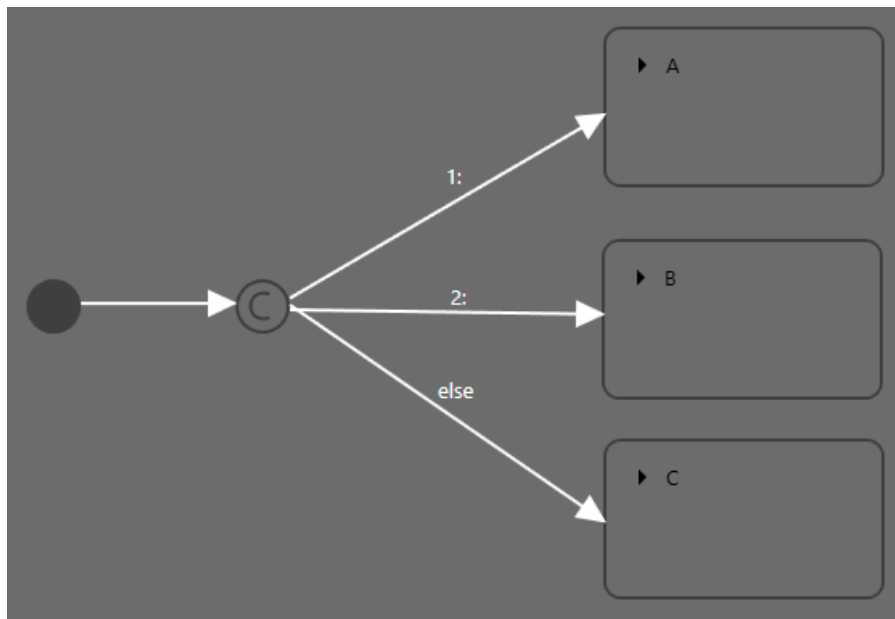


Figure 6.30. Example Choice state with transitions

### 6.24.2.5. History states

History states restore states that were active when their parent state was left. It is possible to have a history state in a parent state and another history state in a child state. When a state with a history state is entered for the first time, the history state is empty. For this case history states have a default transition. When an empty history state is entered, the default transition is executed. EB GUIDE supports the following types of history states:

#### Shallow history states

- Restore the last child-state that was active before a compound state was exited.

- Do not store hierarchies.

- Have an outgoing default transition without conditions.

#### Deep history states

- Restore a state and its complete sub-hierarchy before the state was exited.

Have an outgoing default transition without conditions.



### Example 6.28. Shallow history state

A shallow history state can be used as follows.

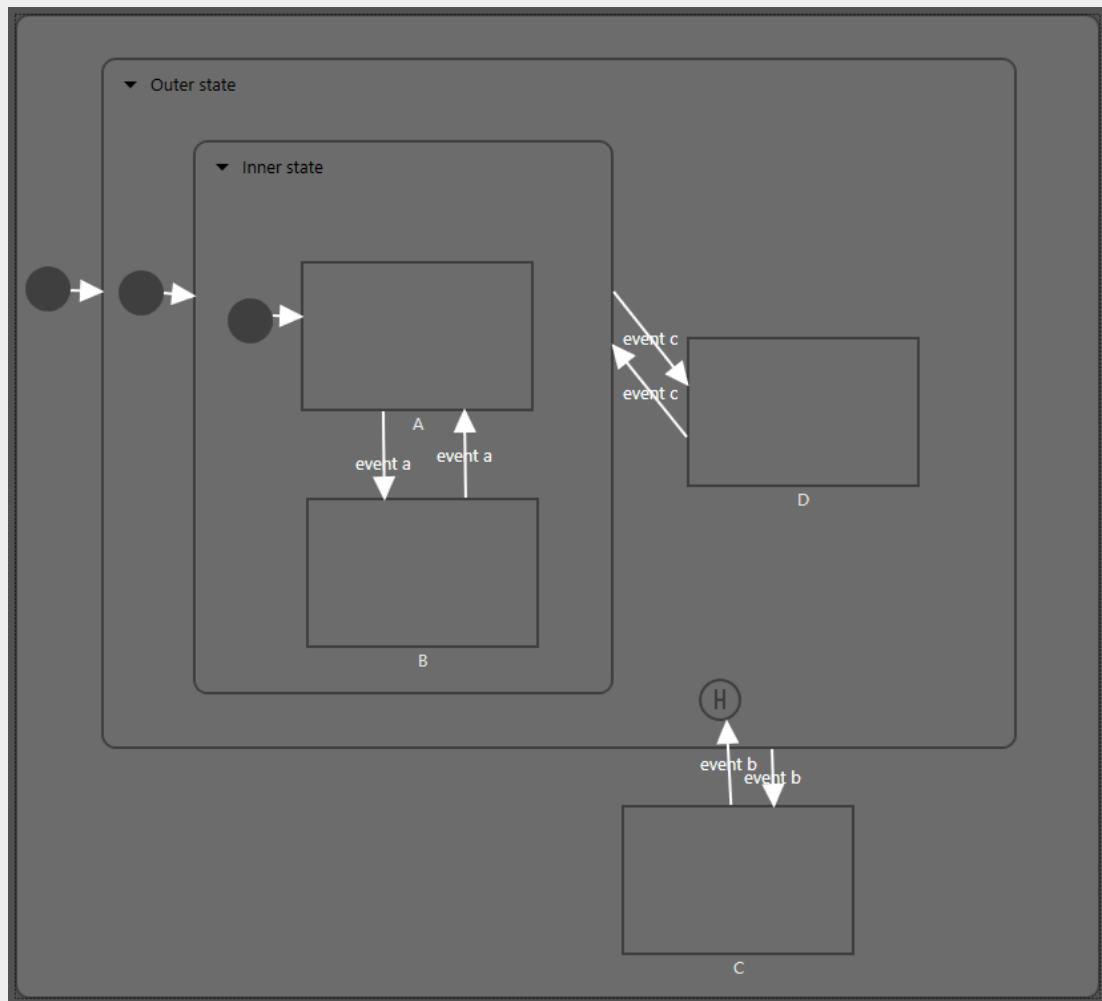


Figure 6.31. Example Shallow history state

- Case 1: The active state is D.
  1. event b is fired and state C is entered.
  2. event b is fired again and the shallow history state is entered.
  3. From the shallow history state, the state machine enters state D because state D was the last active state in Outer state.
- Case 2: The active state is B.
  1. event b is fired and state C is entered.

2. `event b` is fired again the shallow history state is entered.
3. From the shallow history state, the state machine enters `Inner state` because shallow history states remember the last active state but cannot remember hierarchies.
4. Entering `Inner state` leads to state A.

A deep history state can save hierarchical histories.



### Example 6.29. Deep history state

A deep history state can be used as follows.

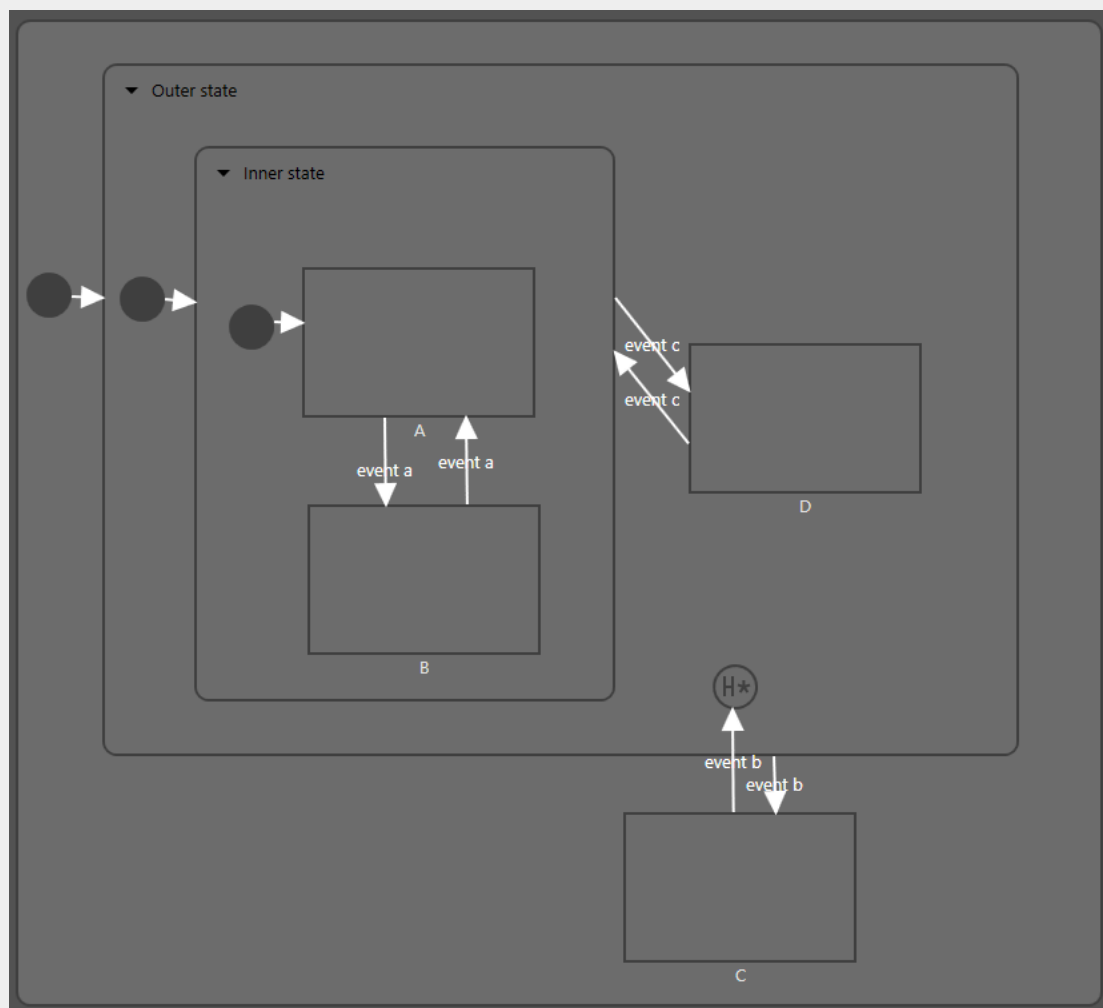


Figure 6.32. Example Deep history state

► Case 1: The active state is D.

1. `event b` is fired and state C is entered.

2. `event b` is fired again and the deep history state is entered.
3. From the deep history state, the state machine enters state `D` because state `D` was the last active state in `Outer state`.

► Case 2: The active state is `B`.

1. `event b` is fired and state `C` is entered.
2. `event b` is fired again and the deep history state is entered.
3. From the deep history state, the state machine enters state `B` because state `B` was the last active state and deep history state remembers state hierarchies.

#### 6.24.2.6. Final state

Final states are used to exit a compound state or to terminate state machines. Compound states and state machines can have only one final state. A final state does not have any outgoing transitions. If a compound state contains a final state, the compound state must have an outgoing transition. History states within the compound state are reset.

A final state is triggered by the following actions:

- A transition from a child state to the outside of the compound state (the transition with event `z`)
- An outgoing transition from the compound state (the transition with event `y`)
- A transition to the final state in a compound state (the transition with event `x`)

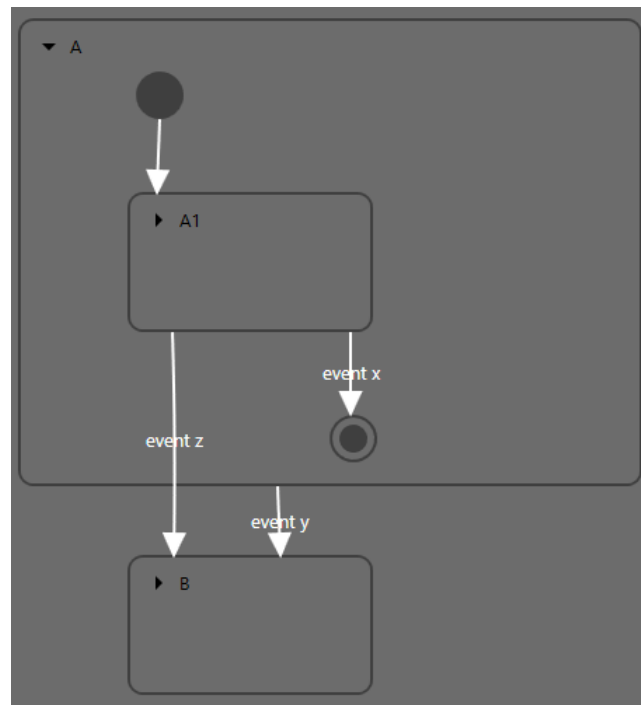


Figure 6.33. Example Final state usage in a Compound state

### 6.24.3. Transitions

A transition is a directed relationship between a source state and a target state. It takes the state machine from one state to another. Transitions have properties that you can use to make it unambiguous which transitions should be executed or to trigger actions with a transition. For instructions, see [section 7.4, “Connecting states through transitions”](#).

A transition has the following properties:

- ▶ A trigger to execute the transition. A trigger can either be an event or the change of a datapool item.
- ▶ A condition that must be evaluated as `true` to execute the transition
- ▶ An action that is executed along with the transition

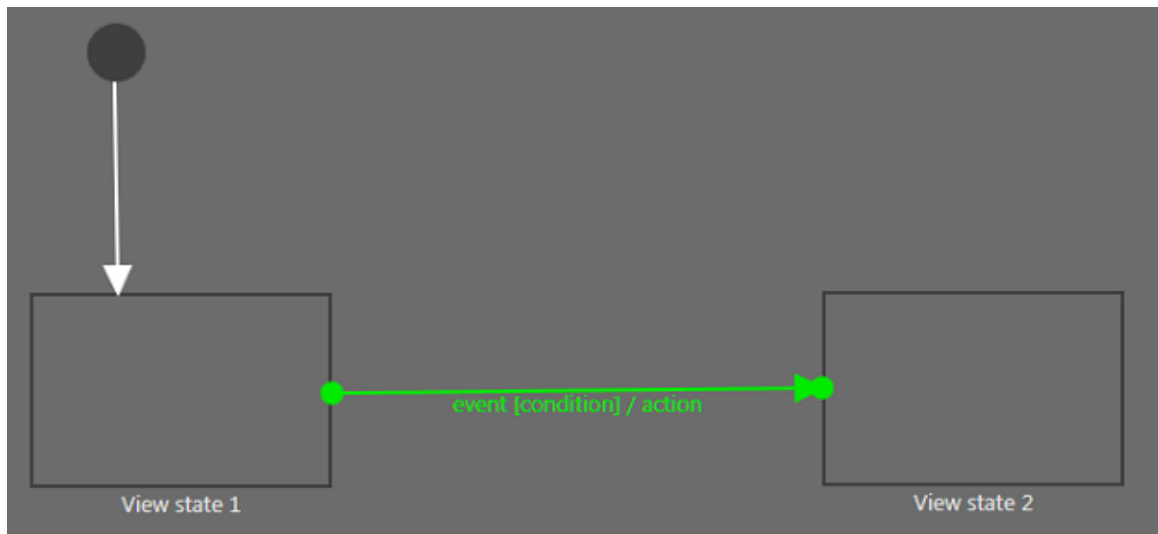


Figure 6.34. Example Transition

There are different types of transitions. Which one you need depends on your use case:

#### Default transition

A default transition is triggered automatically and not by any event or datapool item update. It has no condition, but can have an action. It is used with initial state, final state, choice state, and history states.

#### Choice transition

An outgoing transition with a condition assigned to it. Choice transitions are available when you use a choice state. Its source state is a choice state. Choice transitions are triggered by the evaluation of their condition. They result in an action. The first choice transition that has condition `true` is executed.

#### Else transition

The mandatory counterpart of a choice transition. Every choice state needs to have one else transition that is executed if the conditions of all its choice transitions evaluate to `false`.

#### Internal transition

Has no target state and thus does not change the active state. The purpose of an internal transition is to react to an event without leaving the present state. It can have a condition and it results in an action. It is possible to have several internal transitions for the same event in a state. The order of execution is defined.

#### Self transition

Has the same state as source state and target state. Unlike an internal transition, a self transition leaves and re-enters the state and thus executes its entry and exit actions.

A state inherits all transitions from its parent states. If a number of states share the same transitions to another state, an enclosing compound state can be used to bundle the transitions to reduce the number of conditions.



#### Example 6.30.

### Transition inheritance

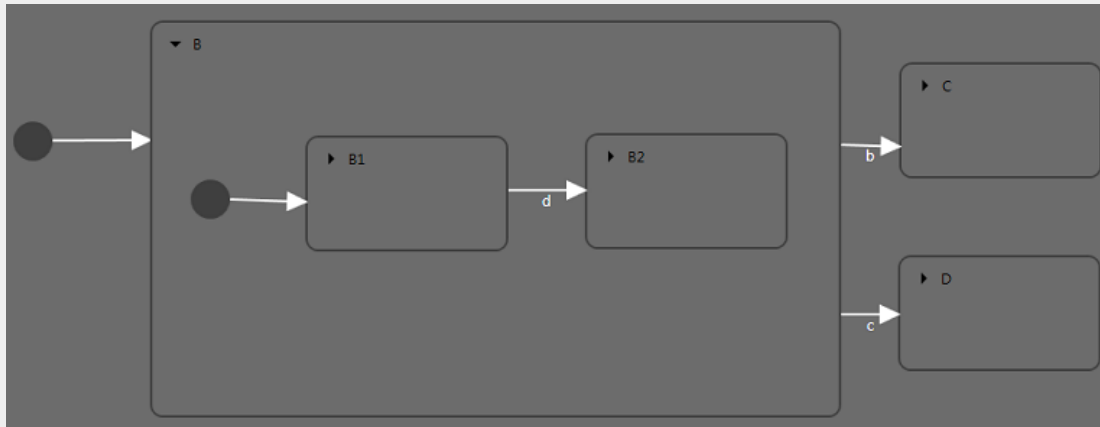


Figure 6.35. Transition inheritance

If the event **b** is fired while the state machine is in State **B1**, the transition to State **C** is executed because the child states State **B1** and State **B2** inherit the transitions of state State **B**.

If an internal transition from the child state uses the same event as the transition from the parent state, transition inheritance is overridden.



### Example 6.31. Transition override

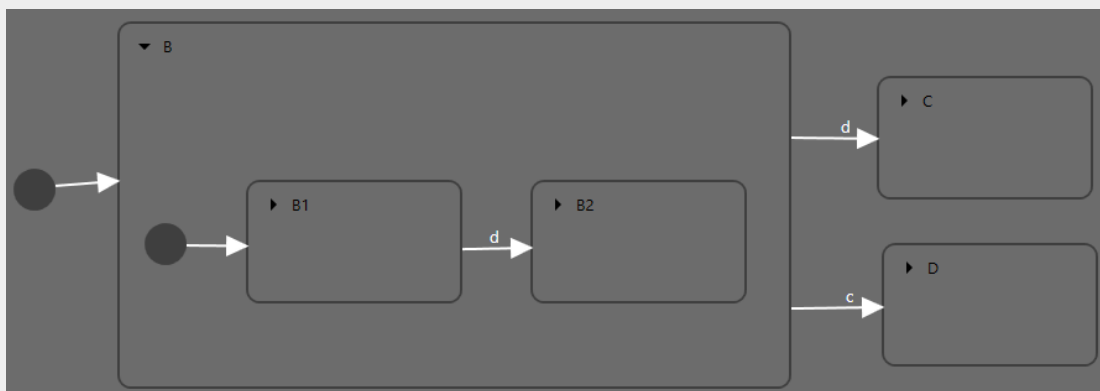


Figure 6.36. Transition override

If event **d** is fired while the state machine is in state State **B**, the transition to State **C** is executed.

If event **d** is fired while the state machine is in state State **B1**, the transition to State **B2** is executed instead of the transition to State **C**. Because the two transitions have the same name, the inner transition overrides the outer one.

## 6.24.4. Execution of a state machine

When a state machine is executed, at any moment in time it has exactly one active state. A state machine is event-driven. That means whether a state is entered or exited depends on events.

When a state is executed, all following actions such as their default transitions are also evaluated. To prevent endless cycles this evaluation is stopped when the number of state changes for a single trigger exceeds 1000.

In a state machine the hierarchy for the execution of transitions that use the same event is always from the inside out. This means internal transitions are preferred.

The state machine life cycle is as follows:

1. The state machine is started by entering its initial state.

The default transition is triggered immediately. If the default transition points to a history state and the history state contains no history information, the history state's default transition is triggered immediately.

2. The state machine waits for incoming events. When an event is received, the following process is started to find a transition that matches the event and conditions:

Starting at the current state, transitions are searched. The following search results and actions are possible:

- a. Search result: An internal transition in the current state matches the received event and the condition.

Action: The matching internal transition is executed. Actions of this transition are executed and default or initial transitions are executed.

- b. Search result: No internal transition in the current state matches the event and condition.

Action: The current state is searched for matching transitions.

- i. Search result: A transition in the current state matches the received event and the condition.

Action: The matching transition is executed. The source state is exited and the target state is entered. Between the exit and the entry the transition's action is executed. Several compound states in the state hierarchy can be exited and entered with one transition.

- ii. Search result: No transition in the current state matches the event and condition.

Action: The current state is left. The search continues in the parent state. Again, first matching internal transitions are searched. If none are found, transitions are searched. If none are found, again the parent state is searched. The search continues this way until a matching transition is found or until the root is reached.

3. The state machine stops when the final state of the state machine is reached.

If a transition crosses several states in the state hierarchy, a cascade of exit and entry actions is executed. See the following examples.





### Example 6.32. Executing a transition

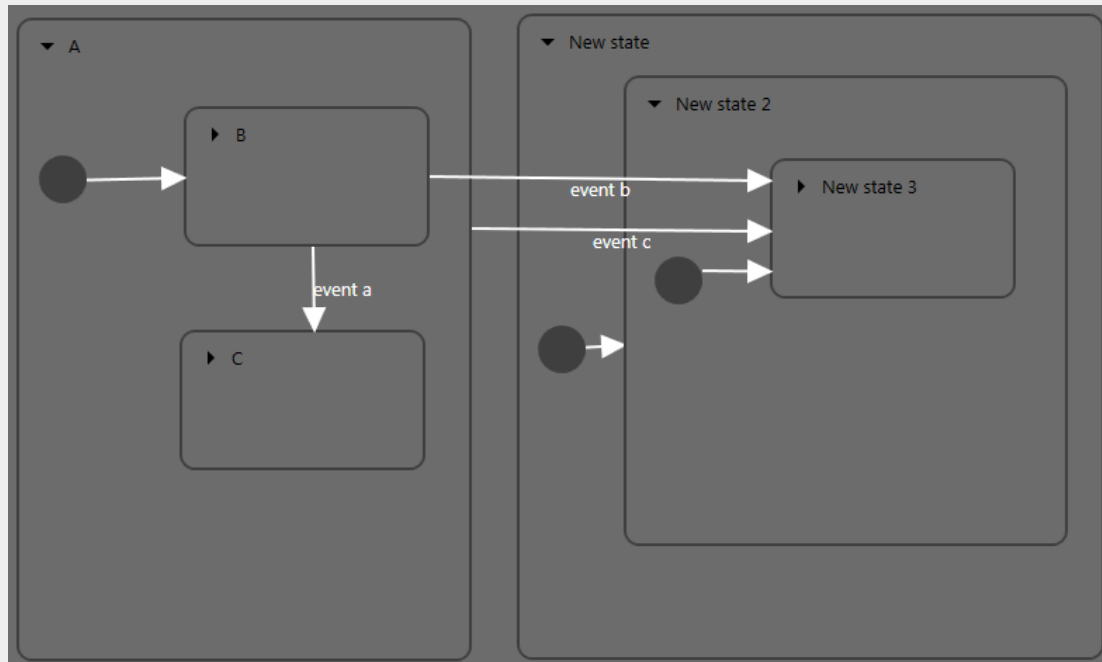


Figure 6.37. Executing a transition

When `event a` is fired, the following happens:

1. State `B` is exited.
2. State `C` is entered.

When `event b` is fired, the following happens:

1. State `B` is exited.
2. State `A` is exited.
3. State `New state` is entered.
4. State `New state 2` is entered.
5. State `New state 3` is entered.

When `event c` is fired, the following happens:

1. If state `B` or state `C` is active, state `B` or state `C` is exited.
2. State `A` is exited.
3. State `New state` is entered.
4. State `New state 2` is entered.

5. State `New state 3` is entered.



**Example 6.33.**  
**Executing a transition**

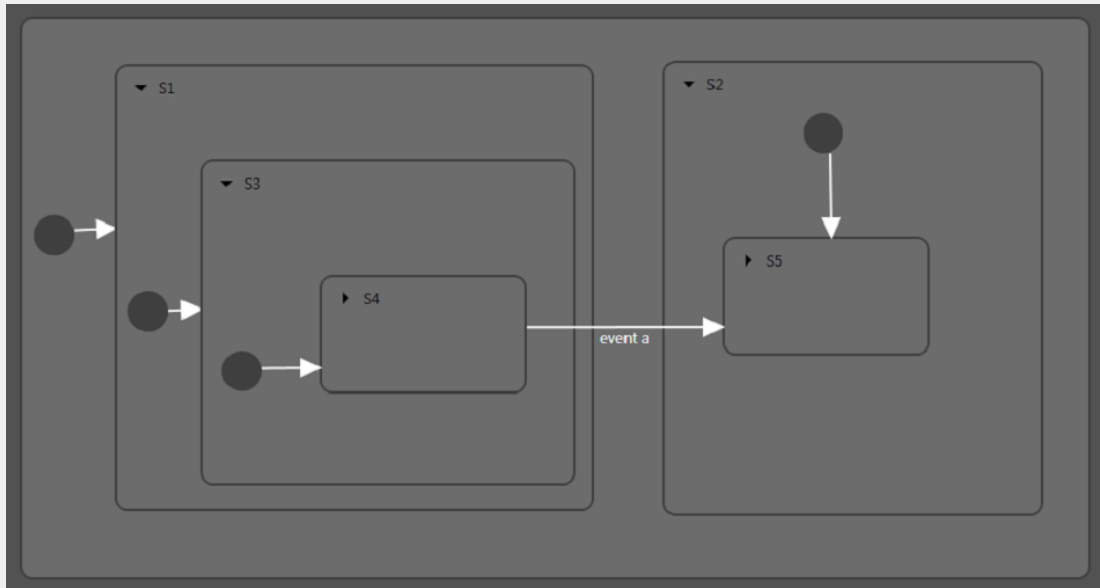


Figure 6.38. Executing a transition

When `event a` triggers the transition, the following happens:

1. State `S4` is exited.
2. State `S3` is exited.
3. State `S1` is exited.
4. State `S2` is entered.
5. State `S5` is entered.



**Example 6.34.**

### Executing a transition

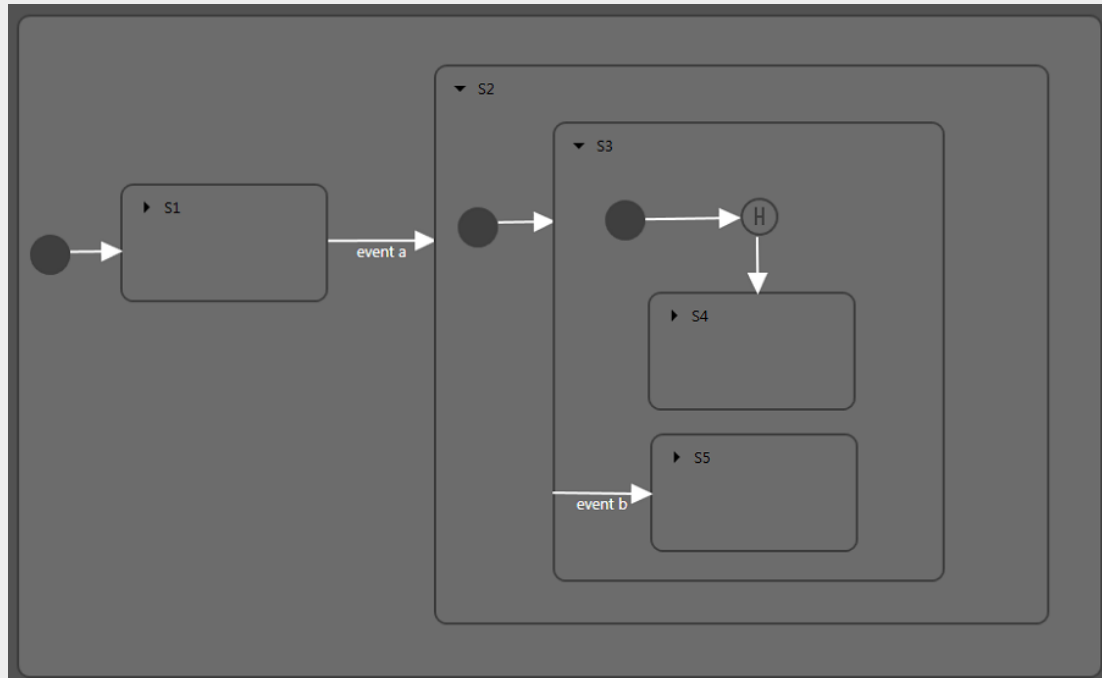


Figure 6.39. Executing a transition

The transition that is triggered by `event a` causes the following transition sequence:

1. The state machine goes to state `S2`.
2. The default transition leads to state `S3`.
3. The next default transition enters the shallow history state.
4. Shallow history state restores the last active state of state `S3`, either state `S4` or state `S5`.

For each step the entry-exit-cascade is executed separately.

## 6.24.5. EB GUIDE notation in comparison to UML notation

In this section the EB GUIDE notation is compared to the Unified Modeling Language (UML) 2.5 notation.

### 6.24.5.1. Supported elements

The following table shows all UML 2.5 elements that are supported by EB GUIDE. The names of some elements deviate from the naming convention in UML 2.5, but the functionality behind these elements remains the same:

Table 6.4. EB GUIDE notation in comparison to UML notation

Name in EB GUIDE	Name in UML 2.5
Initial state	Initial (pseudostate)
Final state	Final state
Compound state	State
Choice state	Choice (pseudostate)
Deep history state	DeepHistory (pseudostate)
Shallow history state	ShallowHistory (pseudostate)
Internal transition	Internal transition
Transition	External/local/self transition.  EB GUIDE does not differentiate between external, local, and self transitions.

#### 6.24.5.2. Not supported elements

The following UML 2.5 elements are not supported in EB GUIDE:

- ▶ Join
- ▶ Fork
- ▶ Junction
- ▶ Entry point
- ▶ Exit point
- ▶ Terminate

#### 6.24.5.3. Deviations from UML

Some elements of the UML 2.5 notation are not implemented in EB GUIDE. But the functionality of these elements can be modeled with EB GUIDE.

Concept in UML 2.5	Workaround with EB GUIDE
Parallel states	To keep track of independent states or partly independent states, it is recommended to use dynamic state machines. For real parallelism you need additional scenes configured on own contexts with own threads. For more information about dynamic state machines, see <a href="#">section 6.24.1, “Dynamic state machines”</a> .

Concept in UML 2.5	Workaround with EB GUIDE
Number of triggers per transition	In EB GUIDE Studio it is possible to have multiple transitions triggered with one trigger. But it must be defined unambiguously which transition is to be chosen under which condition. This is done with EB GUIDE Script. For instructions, see <a href="#">section 7.4.5, “Adding a condition to a transition”</a> .
Time triggers at transitions	Concept is implemented through expression <code>fire_delayed</code> in EB GUIDE Script. For more information, see <a href="#">section 15.4, “EB GUIDE Script”</a> .

## 6.25. Touch input

EB GUIDE supports two types of touch input: Touch gestures and multi-touch input.

Each touch gesture is represented in EB GUIDE Studio as a widget feature. Enabling the widget feature adds a set of properties to a widget.

The gestures are divided into two basic types:

- ▶ Non-path gestures
- ▶ Path gestures

### 6.25.1. Non-path gestures

EB GUIDE implements the following non-path gestures:

- ▶ Flick
- ▶ Pinch
- ▶ Rotate
- ▶ Hold
- ▶ Long hold

Non-path gestures include multi-touch and single-touch gestures. Multi-touch gestures require an input device that supports multi-touch input. Single-touch gestures work with any supported input device.

Each gesture reacts independently of the others. If several gestures are enabled, the modeler is responsible to make sure that the EB GUIDE model behaves consistently.

## 6.25.2. Path gestures

Path gestures are shapes drawn by a finger on a touch screen or entered by some other input device. When a widget has the widget feature enabled, the user can enter a shape starting on the widget. The shape has to exceed a configurable minimal bounding box to be considered by the path gesture recognizer. The shape is matched against a set of known shapes and, if a match is found, a gesture is recognized.

For instructions, see [section 14.3, “Tutorial: Modeling a path gesture”](#).

## 6.25.3. Input processing and gestures

Gesture recognition runs in parallel to ordinary input processing. Each gesture can request that the contact involved in the gesture is removed from ordinary input processing. The moment at which a gesture requests contact removal depends on the actual gesture and for some gestures this can be configured.

Contact removal is only relevant for fingers involved in a gesture. Once a contact is removed, it is ignored by ordinary input handling until a release event is received for the contact. On a touch screen without proximity support this implies that a contact, once removed, does not trigger any further touch reactions.

---

### NOTE



#### Removing a contact from ordinary input processing

Consider a window with a button and a widget feature for gestures. When a contact is involved in a gesture it should not cause the action associated with the button to be triggered, even if the contact is released while on the button.

---

## 6.25.4. Multi-touch input

EB GUIDE is able to handle multi-touch input, if a compatible multi-touch input device is used.

Multi-touch is the ability of a surface to recognize and track more than one point of contact on an input device. The typical scenario are multiple fingers touching a touch screen.

### ► Multi-touch event handling

Multi-touch events are dispatched using the mechanism for touch events, in the same way events from the mouse and from single-touch touch screens are dispatched. The only difference is that each contact triggers touch reactions independently of all others. To be able to distinguish individual contacts, each touch reaction is supplied with a parameter called `fingerId`.

### ► Finger ID

Each contact tracked by an input device is assigned a number that identifies it. This identifier is called `fingerId` and is unique per input device. However, the same value can be assigned to another contact at a later time when it is no longer in use.

Consider the extra touch interaction sequences the end user is allowed to make when multi-touch input is enabled. They include the following:

- ▶ The end user can interact with multiple elements of the interface at the same time, for example press a button while scrolling in a list.
- ▶ The end user can place multiple fingers on a single widget.

Two typical situations where this manifests are scrolling and dragging. They can be handled correctly by employing `fingerId`. Depending on the required behavior, possible solutions include the following:

- ▶ Allow only the first finger that pressed a widget to do scrolling and/or dragging.
- ▶ Always use the last finger to land on a widget to do scrolling and/or dragging. This is easily achieved by a slight modification of the previous approach.

## 6.26. Widgets

Widgets are the basic graphical elements an EB GUIDE model is composed of.

It is possible to customize widgets. Editing the properties of a widget adapts the widget to individual needs. The following elements are the example properties when being touched or moved:

- ▶ Size
- ▶ Color
- ▶ Layout
- ▶ Behavior

It is possible to combine widgets. Out of small building blocks, complex structures are created. For example, it is possible to make up a button of the following elements:

- ▶ Ellipse
- ▶ Image
- ▶ Label
- ▶ Rectangle

It is possible to nest widgets. These subordinate widgets are referred to as child widgets and the superordinate widgets are referred to as parent widgets in a widget hierarchy.

### 6.26.1. View

A View is the topmost widget of each scene. While modeling, the following elements are placed into views:

- ▶ Basic widgets
- ▶ 3D widgets
- ▶ Animations
- ▶ Widget templates

Every view is associated to exactly one view state. A view cannot exist without a view state.

#### TIP




#### Changing the size of a view

In EB GUIDE Studio, to get a close-up view or to see more, you have the possibility to increase or decrease the size of a view. To zoom in and zoom out, use the slider or click the text box at the bottom of the View. The default zoom level is 100%. Alternatively, use the **Ctrl++** to zoom in, **Ctrl+-** to zoom out and **Ctrl+0** to reset the zoom level to 100%.

#### TIP



#### Aligning elements on a master image

In EB GUIDE Studio, to align elements evenly such as basic widgets and 3D widgets, you have the possibility to add a master image to a view. To add a master image to a view, click  at the bottom of the view. To hide the master image, check or clear the check box. If you close the view, you need to add the master image again.

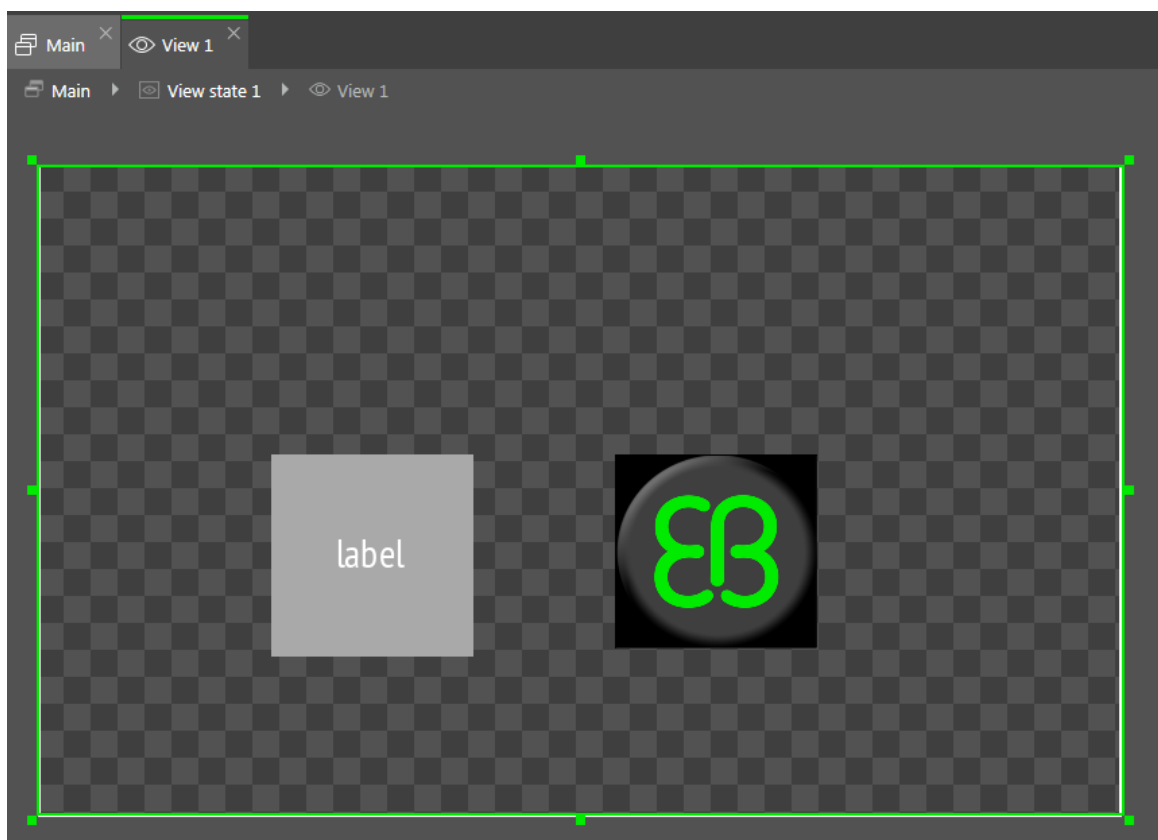


Figure 6.40. View that contains a Rectangle, a Label, and an Image



## 6.26.2. Widget categories

In the **Toolbox**, widgets are grouped by categories. The following categories are available.

- ▶ Basic widgets

The following elements are the basic widgets:

- ▶ Alpha mask
- ▶ Animation
- ▶ Container
- ▶ Ellipse
- ▶ Image
- ▶ Instantiator
- ▶ Label
- ▶ Rectangle

- ▶ 3D widgets

To display a 3D graphic, use widgets that are contained in the **3D widgets** category. The **3D widgets** are the following elements:

- ▶ Ambient light
- ▶ Camera
- ▶ Directional light
- ▶ Image-based light
- ▶ Material
- ▶ Mesh
- ▶ PBR GGX material
- ▶ PBR Phong material
- ▶ Point light
- ▶ Scene graph
- ▶ Scene graph node
- ▶ Spot light

---

**NOTE****Supported renderers**

To display 3D graphics, OpenGL ES 2.0 or higher is required. Make sure that your graphics driver is compatible to the version of the renderer.

---

► **Templates**

The **Templates** category contains widget templates. It is only visible if widget templates are defined.

► **Custom widgets**

The **Custom widgets** category contains customized widgets and is therefore only visible when customized widgets are added to the project. For more information about extensions, see the EB GUIDE Studio user guide chapter "Extending EB GUIDE Studio". For extension and application examples, see <https://github.com/Elektrobit/eb-guide-examples>.

For instructions, see [section 8.2, "Working with widgets"](#).

### 6.26.3. Widget properties

A widget is defined by a set of properties which specify the appearance and behavior of the widget. The **Properties** component displays the properties of the currently focused widget and allows editing the properties.

Each widget property has a data type. For a list of the data types that are supported by EB GUIDE Studio, see [section 15.3, "Data types"](#).

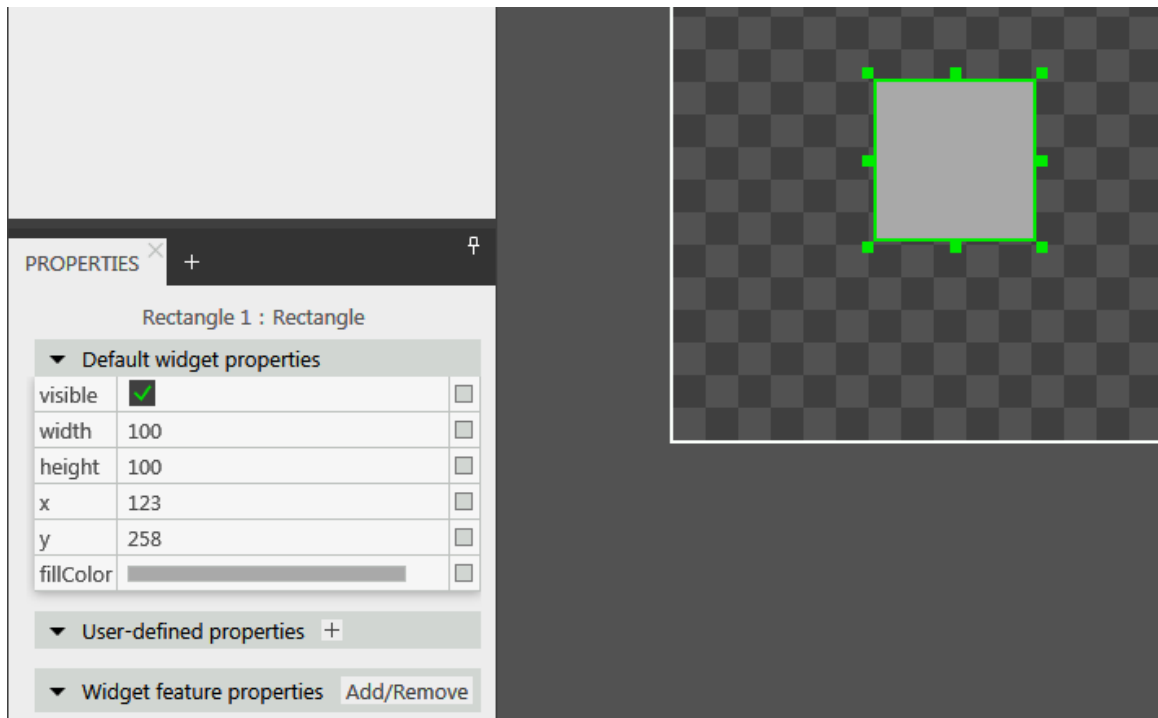


Figure 6.41. Rectangle and its properties

There are three types of widget properties:

- ▶ Default widget properties are created along with each widget instance. For a list of default properties for all widgets, see [section 15.10, “Widgets”](#).
- ▶ User-defined widget properties are created by the modeler in addition to the default ones.
- ▶ Widget feature properties are created by EB GUIDE Studio when the modeler adds a widget feature to a widget. Widget feature properties are grouped by categories. Widget features add more functionality for the appearance and behavior of widgets.




#### Example 6.35. Touched widget feature


The **Touched** widget feature defines if and how a widget reacts to being touched. It adds four properties. The boolean property `touchable` determines if the widget reacts on touch input. The boolean property `touched` is set during run-time by EB GUIDE if the widget is currently touched. The `touchPolicy` property defines how to handle touch and the `touchBehavior` property determines the touch area.

For more information, see [section 6.16, “Linking”](#).

## 6.26.4. Widget templates

A widget template allows the definition of a customized widget that can be used multiple times in an EB GUIDE model. You have the possibility to define templates on the basis of existing widgets or derive a new template from an existing one. After creating, you modify the template according to your needs, for example, by adding properties or widget features. Widget templates thus allow you to build a library of complex widgets.

A widget template has a template interface. The template interface contains the properties of the template which are visible and accessible in widget instances. A widget instance thus inherits the properties of its template's interface. Inherited properties are called template properties. Template properties are marked with the  button.

When you change the value of a template property, the property is turned into a local property. Local properties are marked with the  button.

---

**NOTE**

**Parent widgets for a template**



It is not possible to use the animation widget as a parent widget for a template.

---

**NOTE**

**Restrictions for scripted values**



It is not possible to add scripted values to model or template interfaces.



**Example 6.36.**

**Relation of the properties of a widget template and its instances**

You add a widget template `Square` to the EB GUIDE model. Let `Square` have a property `color`. `color` is added to the template interface. Let the value of `color` be `red`.

You add an instance of the widget template `Square` to a view. The instance is named `BlueSquare`.

- ▶ `BlueSquare` inherits `color` with the value `red`.
- ▶ Change the value of `color` in the `Square` template to `green`.  
=> The value of `color` in `BlueSquare` changes to `green`, too.
- ▶ Change the value of `color` in `BlueSquare` to `blue`.

Change the value of `color` in the `Square` template to `yellow`.

=> The value of `color` in `BlueSquare` remains `blue`.

For instructions, see [section 8.8, “Working with templates”](#).

## 6.26.5. Widget features

It is possible to extend widgets and widget templates in their functionality using widget features. Widget features have predefined widget properties. Widget features are grouped into categories.

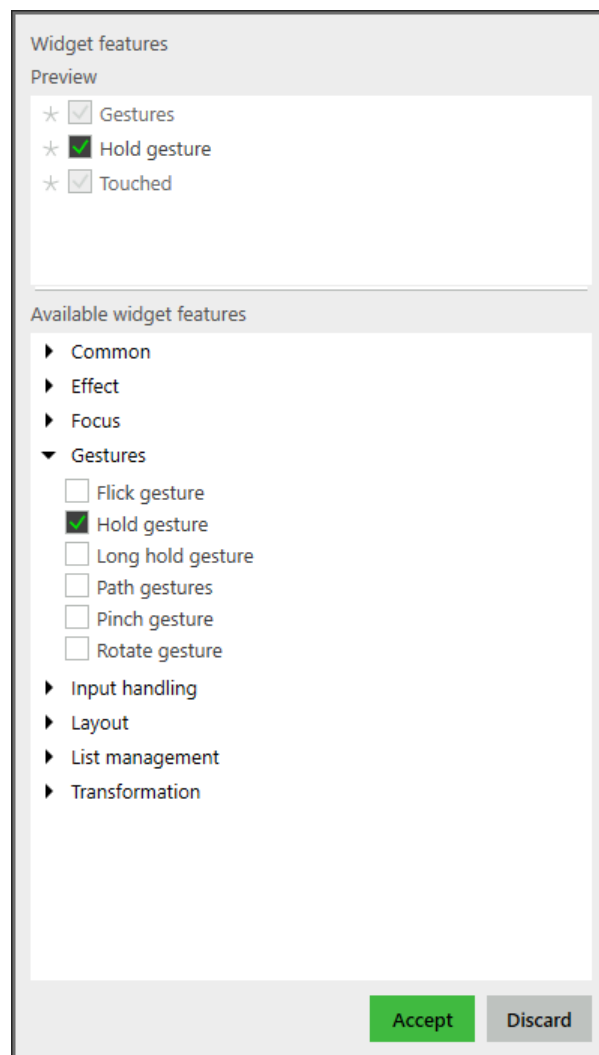


Figure 6.42. Widget features

If you add a widget feature to a widget template, any created widget template instance inherits the added widget feature. Note that you cannot add widget features to a widget template instance or to a template that was created from a template.

Restrictions for usage of widget features are as follows:

- ▶ Widget features do not have an inheritance hierarchy.
- ▶ It is not possible to add a widget feature more than once per widget.

- ▶ Some widget features are interdependent. That means, to add one widget feature, you have to add another, or widget features may exclude each other.
- ▶ It is possible to restrict widget features to a particular type of widgets.
- ▶ It is not possible to activate or deactivate widget features during run-time.

By default all widget features are disabled. If you need a specific widget feature, you must add it to a widget.

For instructions, see [section 8.4, “Extending a widget by widget features”](#). For a list of all widget features, see [section 15.11, “Widget features”](#).

### 6.26.5.1. List management widget feature category

The **Line index** and **Template index** widget features allow you to connect data, for example images, song titles, to the corresponding dynamically created line templates of an instantiator.

#### Line index

The **Line index** widget feature is used to customize the line templates of the instantiator widget. The **Line index** widget feature defines the unique position for each line of your list or table.



#### Example 6.37. Line index widget feature

If you want to model a list, you would expect that each entry of the list has a specific value that reflects the entry in a list property. To access a certain entry in a list, the instance of the line template needs to know which of the instantiator's child it is. The **Line index** widget feature adds the `lineIndex` property. While the instantiator creates the instances of line templates, it fills `lineIndex` with values: The index starts with zero for the first instance. If you have two elements in the instantiator, the second element receives the `lineIndex` value 1.

For instructions, see [section 14.4, “Tutorial: Creating a list with dynamic content”](#).

#### Template index

The **Template index** widget feature allows complex data abstraction. For very complex lists or tables, to visualize an entry or a set of entries, you require more than one data list. For example, a table with mixed image and text content requires a list of images and a list of strings. To cover such complex cases, the **Template index** widget feature provides the property `lineTemplateIndex`.



#### Example 6.38. Template index widget feature

If you model a list using an instantiator with the property `lineMapping` set to `0|1` and the property `numItems` set to 5, the `lineTemplateIndex` results in `0|0|1|1|2`.

## 7. Modeling HMI behavior

### TIP



### Default window layout

All instructions and screenshots use the default window layout. If you want to follow the instructions, we recommend to set the EB GUIDE Studio or EB GUIDE Monitor window to default layout by selecting **Layout > Reset to default layout**.

## 7.1. Modeling a state machine

### 7.1.1. Adding a state machine



#### Adding a state machine

#### Step 1

In the **Navigation** component, go to **State machines**, and click **+**.

A menu expands.

#### Step 2

Select a type for the state machine.

A new state machine of the selected type is added.

#### Step 3

Rename the state machine.

#### Step 4

Go to the **Project center**.

#### Step 5

In the navigation area, click **Configure > Profiles**.

A menu opens.

#### Step 6

In the **Name** category, select the profile where you want to enable the state machine.

#### Step 7

In the **Scenes** category, click **Add**.

A new row is added to the table.

#### Step 8

Next to the state machine that you want to add, select the **Enable** check box.

Step 9

Select the communication context for the state machine in the drop-down list.

Now the state machine is enabled.

## 7.1.2. Defining an entry action for a state machine



### Defining an entry action for a state machine

Step 1

Select a state machine.

Step 2

In the **Properties** component, go to the **Entry action** property, and click **+**.

A script editor opens.

Step 3

Enter an action using EB GUIDE Script.

For background information, see [section 6.21, “Scripting language EB GUIDE Script”](#).

Step 4

Click **Accept**.

You defined an entry action for a state machine.

## 7.1.3. Defining an exit action for a state machine



### Defining an exit action for a state machine

Step 1

Select a state machine.

Step 2

In the **Properties** component, go to the **Exit action** property, and click **+**.

A script editor opens.

Step 3

Enter an action using EB GUIDE Script.

For background information, see [section 6.21, “Scripting language EB GUIDE Script”](#).

Step 4

Click **Accept**.



You defined an exit action for a state machine.

## 7.1.4. Deleting a state machine



### Deleting a state machine

#### Step 1

In the **Navigation** component, right-click the state machine.

#### Step 2

In the context menu, click **Delete**.

The state machine is deleted.

## 7.2. Modeling a dynamic state machine

In general, there are three steps to model a dynamic state machine. They are described in detail in the sections below.

1. Adding a dynamic state machine list
2. Adding a dynamic state machine
3. Controlling the dynamic state machine with EB GUIDE Script

For background information, see [section 6.24.1, “Dynamic state machines”](#). For a tutorial, see [section 14.1, “Tutorial: Adding a dynamic state machine”](#).

### 7.2.1. Enabling a dynamic state machine list

Before you can use dynamic state machines, you need to add a dynamic state machine list to the state where you want to use the dynamic state machine. This list is also called stack. When you enable the dynamic state machine list, it is enabled for all sub states as well.



### Enabling a dynamic state machine list

Prerequisite:

- A state machine, View state, or Compound state where you want to add the dynamic state machine is added to the model.

#### Step 1

In the **Navigation** component, click the state machine, View state, or Compound state where you want to add the dynamic state machine.

#### Step 2

In the **Properties** component, select the **Dynamic state machine list** check box.

## 7.2.2. Adding a dynamic state machine



### Adding a dynamic state machine

Prerequisite:

- A state machine, View state, or Compound state is added to the EB GUIDE model.

#### Step 1

In the **Navigation** component, go to **Dynamic state machines**, and click **+**.

A menu expands.

#### Step 2

Select a type for the dynamic state machine.

A new dynamic state machine of the selected type is added.

## 7.2.3. Controlling a dynamic state machine

Dynamic state machines run in parallel to other state machines and can be started (pushed) and stopped (popped) during run-time. This is done using EB GUIDE Script functions. The following script functions apply to dynamic state machines:

Starting (pushing)

```
pushDynamicStateMachine
```

Stopping (popping)

```
popDynamicStateMachine
```

Changing the order, the priority

```
changeDynamicStateMachinePriority
```

Checking the status

```
isDynamicStateMachineActive
```

Checking whether a widget belongs to a dynamic state machine

```
isWidgetOnActiveStatemachine
```

Removing all state machines from the dynamic state machine list

```
clearAllDynamicStateMachines
```

## 7.3. Modeling states

### 7.3.1. Adding a state



#### Adding a state

Prerequisite:

- The content area displays a state machine.

#### Step 1

Drag a state from the **Toolbox** into the state machine.

A state is added to the state machine.

#### NOTE



#### Initial state, Final state, and history states are unique

You can insert Initial state, Final state, and history states only once per compound state.

#### TIP



#### Copying and finding states

Alternatively, you can copy and paste an existing state using the context menu or **Ctrl+C** and **Ctrl+V**.

To find a specific state within your EB GUIDE model, enter the name of the state in the search box or use **Ctrl+F**. To jump to a state, double-click it in the hit list.

### 7.3.2. Adding a state to a Compound state



#### Adding a state to a Compound state

To create a state hierarchy, you create a state as a child to another state. You do so by adding a state to a Compound state.

Prerequisite:

- The content area displays a state machine.
- The state machine contains a Compound state.

#### Step 1

In the **Navigation** component, double-click the Compound state.

The Compound state expands in the content area.

#### Step 2

Drag a state from the **Toolbox** into the Compound state.

The state is added as a child state to the Compound state.

### 7.3.3. Adding a Choice state



#### Adding a Choice state

Prerequisite:

- The content area displays a state machine.
- The state machine contains at least two states.

#### Step 1

Drag a Choice state from the **Toolbox** into the state machine.

#### Step 2

Add a condition to the outgoing transition. For instructions, see [section 7.4.5, "Adding a condition to a transition"](#).

The condition is assigned priority one. When the state machine enters the Choice state, the condition with priority one is evaluated first.

#### Step 3

To add more choice transitions, repeat the two previous steps.

A new choice transition is assigned a lower priority than the transition that was created before.

#### Step 4

Add an outgoing transition from the Choice state.

#### Step 5

In the **Navigation** component, right-click the transition. In the context menu, click **Convert to else**.

You added an else transition. The else transition is executed when all conditions which are assigned to outgoing choice transitions evaluate to `false`.

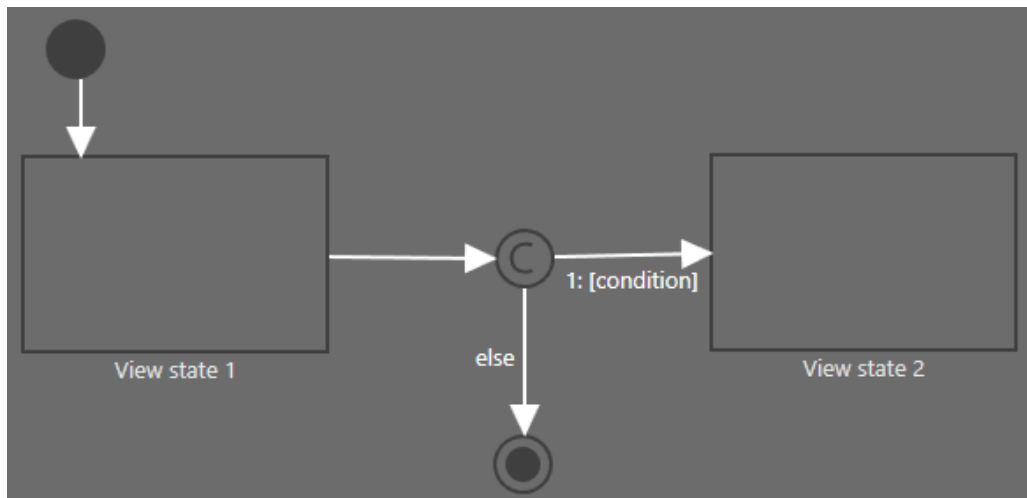


Figure 7.1. Choice state with its choice transitions

### 7.3.4. Defining an entry action for a state



#### Defining an entry action for a state

For View states and Compound states you can define an entry action. The entry action is executed every time the state is entered.

Prerequisite:

- A state machine contains a View state or a Compound state.

#### Step 1

Select a state.

#### Step 2

In the **Properties** component, go to the `Entry action` property, and click **+**.

A script editor opens.

#### Step 3

Enter an action using EB GUIDE Script.

#### Step 4

Click **Accept**.

### 7.3.5. Defining an exit action for a state



#### Defining an exit action for a state

For View states and Compound states you can define an exit action. The exit action is executed every time the state is exited.

Prerequisite:

- A state machine contains a View state or a Compound state.

##### Step 1

Select a state.

##### Step 2

In the **Properties** component, go to the `Exit action` property, and click **+**.

A script editor opens.

##### Step 3

Enter an action using EB GUIDE Script.

##### Step 4

Click **Accept**.

## 7.3.6. Deleting a model element from a state machine



#### Deleting a model element from a state machine

Prerequisite:

- A state machine contains at least one model element.

##### Step 1

In the **Navigation** component, right-click a model element.

##### Step 2

In the context menu, click **Delete**.

The model element is deleted.

## 7.4. Connecting states through transitions

## 7.4.1. Adding a transition between two states



### Adding a transition between two states

With a transition, you connect a source state to a target state.

Prerequisite:

- The content area displays a state machine.
- The state machine contains at least two states.

#### Step 1

Select a state as a source state for the transition.

The state border is highlighted green.

#### Step 2

Hover over the highlighted state border.

A drag point is displayed.

#### **TIP**



#### **Drag point**

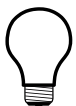
You can place a drag point on any part of the border.

#### Step 3

Drag the drag point into the target state.

The transition is displayed as a green arrow.

#### **TIP**



#### **Connect transitions to the state machine**

The state machine is the top-most compound state. Therefore, you can create transitions to and from the border of the state machine. All states in the state machine inherit such a transition.

## 7.4.2. Moving a transition



### Moving a transition

You move a transition by moving one of its end points.

Prerequisite:

- The content area displays a state machine.
- The state machine contains at least two states.
- The states are connected by a transition.

#### Step 1

In the content area, click a transition.

Two green drag points are displayed.

#### Step 2

Select and hold the drag point that you want to move.

#### Step 3

Drag the mouse into a different state.

#### Step 4

When the state is highlighted green, release the mouse button.

The transition is moved.

## 7.4.3. Moving transition labels



### Moving transition labels

Prerequisite:

- The content area displays a state machine.
- The state machine contains at least two states.
- The states are connected by a transition.

#### Step 1

To change the shape of a transition, select and drag the transition.

#### Step 2

To move a transition label, press **Shift** and drag the label.

#### Step 3

To rotate a transition label, press **Ctrl** and drag the label.



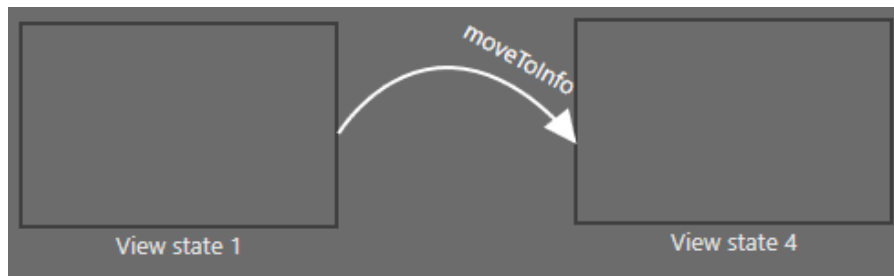


Figure 7.2. Moved and rotated transition label

## 7.4.4. Defining a trigger for a transition



### Defining a trigger for a transition

For a transition, you can define an event that triggers it.

Prerequisite:

- A state machine contains at least two states.
- The states are connected by a transition.

#### Step 1

Select a transition.

#### Step 2

In the **Properties** component, click in the box next to **Trigger**.

#### Step 3

Select an event.

To create a new event, enter a name and click **Add event**.

The event is added as a transition trigger.

## 7.4.5. Adding a condition to a transition



### Adding a condition to a transition

For every transition, you can define a condition that needs to be fulfilled to execute the transition.

Prerequisite:

- A state machine contains at least two states.
- The states are connected by a transition.

Step 1

Select a transition.

Step 2

To add a condition to the transition, go to the **Properties** component. Next to the `Condition` property, click **+**.

A script editor opens.

Step 3

Enter a condition using EB GUIDE Script.

Step 4

Click **Accept**.

The condition is added to the transition.

## 7.4.6. Adding an action to a transition



### Adding an action to a transition

For every transition, you can define an action that is executed along with the transition.

Prerequisite:

- A state machine contains at least two states.
- The states are connected by a transition.

Step 1

Select a transition.

Step 2

To add an action to the transition, go to the **Properties** component. Next to the `Action` property, click **+**.

A script editor opens.

Step 3

Enter an action using EB GUIDE Script.

Step 4

Click **Accept**.

The action is added to the transition.

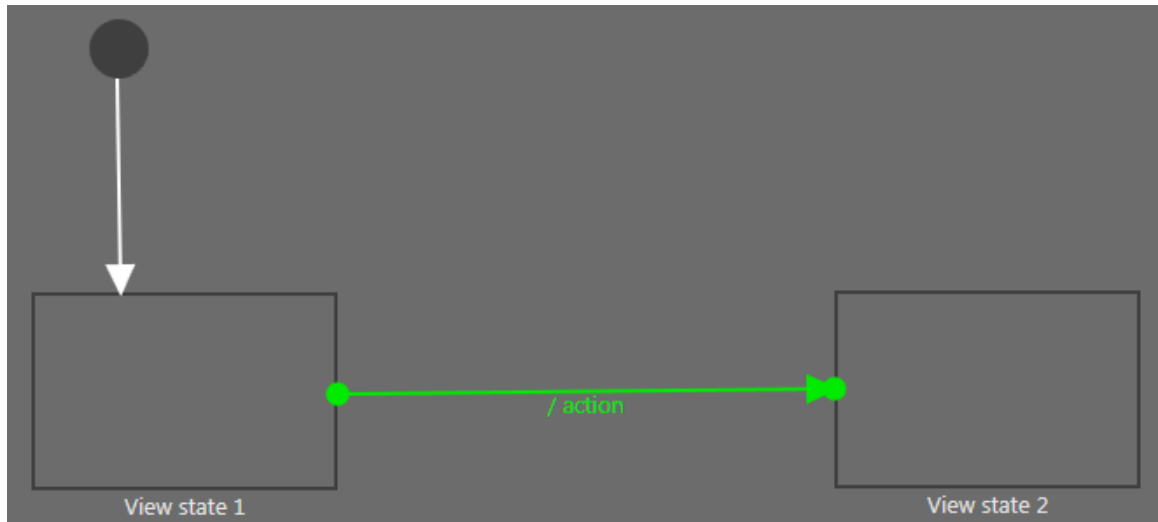


Figure 7.3. A transition with an action

## 7.4.7. Adding an internal transition to a state



### Adding an internal transition to a state

Prerequisite:

- A state machine contains a view state or a compound state.

#### Step 1

Select a state.

#### Step 2

In the **Properties** component, go to **Internal transitions**, and click **+**.

An internal transition is added to the state. The internal transition is visible in the **Navigation** component.

## 7.4.8. Changing the priority of choice transitions

The priority of choice transitions depends on their position in the widget tree. The priority of choice transitions is indicated by the number that is displayed in the widget tree and at the transition arrows. To change the priority, change the position in the widget tree.



## Changing the priority of choice transitions

Prerequisite:

- A state machine contains a choice state with at least two choice transitions.

### Step 1

Go to the **Navigation** component.

### Step 2

Drag the choice transition on top or below the other choice transitions.

You changed the priority of the choice transition. The numbers at the transition arrows change accordingly.

#### **NOTE**



#### **Else transitions**

If you change a choice transition into an else transition, the priority of the other choice transitions is adjusted automatically. For instructions, see [section 7.3.3, “Adding a Choice state”](#).

## 8. Modeling HMI appearance

### TIP



#### Default window layout

All instructions and screenshots use the default window layout. If you want to follow the instructions, we recommend to set the EB GUIDE Studio or EB GUIDE Monitor window to default layout by selecting **Layout > Reset to default layout**.

## 8.1. Changing the background color of states and state machines

You can change the background color that is displayed in EB GUIDE Studio. This does not affect the background color on the target. You can change the background color of haptic state machines, view states, compound states, and dynamic haptic state machines.



### Changing the background color of states and state machines

#### Step 1

In the **Navigation** component select the state or state machine.

#### Step 2

In the **Properties** component select a color from the `Background color` drop-down list box. The color is changed accordingly.

## 8.2. Working with widgets

### TIP



#### Copying and finding views and widgets

You can copy and paste an existing View or widget using the context menu or **Ctrl+C** and **Ctrl+V**.

To find a specific View or widget within your EB GUIDE model, enter the name of the View or widget in the search box or use **Ctrl+F**. To jump to a View or widget, double-click it in the hit list.

### 8.2.1. Adding a View



### Adding a View

Prerequisite:

- The content area displays a state machine.

#### Step 1

Drag a View state from the **Toolbox** into the state machine.

Along with the View state, a View is added to the EB GUIDE model.

#### Step 2

In the **Navigation** component, expand the widget tree and click the View.

#### Step 3

Press the **F2** key, and rename the View.

#### Step 4

Double-click the View state in the content area.

The content area displays the new View.

## 8.2.2. Adding a basic widget to a View

For details on basic widgets, see [section 15.10.2, “Basic widgets”](#).

### 8.2.2.1. Adding a Rectangle



### Adding a Rectangle

Prerequisite:

- The content area displays a View.

#### Step 1

Drag a Rectangle from the **Toolbox** into the View.

The Rectangle is added to the View.

### 8.2.2.2. Adding an Ellipse



### Adding an Ellipse

Prerequisite:

- The content area displays a View.

#### Step 1

Drag an Ellipse from the **Toolbox** into the View.

The widget is added to the View.

### 8.2.2.2.1. Editing an Ellipse

You can draw just a sector of an Ellipse and you can change the arc of an Ellipse.



### Creating a circular sector

Prerequisite:

- The View contains an Ellipse.

#### Step 1

Click the Ellipse and go to the **Properties** component.

#### Step 2

Enter the angle of the sector in the `centralAngle` text box.

#### Step 3

Enter the orientation of the sector in the `sectorRotation` text box.

You created a circular sector.



### Creating a circular arc

Prerequisite:

- The View contains an Ellipse.

#### Step 1

Click the Ellipse and go to the **Properties** component.

#### Step 2

Enter a width between 0 and 50 in the `arcWidth` text box.

You created a circular arc.

### 8.2.2.3. Adding an Image



#### Adding an Image using **Toolbox**

Prerequisite:

- An image file is located in `$GUIDE_PROJECT_PATH/<project name>/resources`. For supported file types, see [section 6.20.3, “Images”](#).
- The content area displays a View.

#### Step 1

Drag an Image from the **Toolbox** into the View.

#### Step 2

In the **Properties** component, select an image from the `image` combo box. Alternatively, drag another image from the **Assets** component into the `image` drop-down list box.

The View displays the image.



#### Adding an Image using **Assets** component

Prerequisite:

- An image file is located in the `$GUIDE_PROJECT_PATH/<project name>/resources` directory. For supported file types, see [section 6.20.3, “Images”](#).
- The content area displays a View.

#### Step 1

Drag an image file from the **Assets** component into the View.

The View displays the image.

#### Step 2

To change the image file, go to the **Properties** component and select an image from the `image` combo box. Alternatively, drag another image from the **Assets** component into the `image` combo box.

The View displays the image.





## Adding 9-patch images

Prerequisite:

- A 9-patch image file is located in the `$GUIDE_PROJECT_PATH/<project name>/resources` directory. For background information on 9-patch images, see [section 6.20.3.1, “9-patch images”](#).
- The content area displays a View.
- An Image is added to the EB GUIDE model.

### Step 1

Select the Image, and go to the **Properties** component.

### Step 2

From the `image` combo box, select a 9-patch image.

### Step 3

Go to the **Widget features properties** and click **Add/Remove**.

The **Widget features** dialog is displayed.

### Step 4

Under **Available widget features**, expand the **Layout** category, and select **Scale mode**.

### Step 5

Click **Accept**.

The related widget properties are added to the Image and displayed in the **Properties** component.

### Step 6

In the **Properties** component, for the `scaleMode` property select `fit to Size (1)`.

#### NOTE



#### Adding 9-patch images

If you do not add the **Scale mode** widget feature or if for the `scaleMode` property you select `original Size (0)` or `keep aspect ratio (2)`, the 9-patch image is scaled like a normal `.png` image.

## 8.2.2.4. Adding a Label

#### NOTE



#### Character replacement

When you enter a text to the `text` property of a label, the following characters are replaced:

- ▶ The sequence `\\\\` is replaced by `\\`.
- ▶ The sequence `\\n` is replaced by `\n`.
- ▶ In case the text is displayed in one line, `\n` is replaced by a space character.



#### Adding a Label using **Toolbox**

Prerequisite:

- The content area displays a View.

##### Step 1

Drag a Label from the **Toolbox** into the View.

The Label is added to the View. The Label has the default font `PT_Sans_Narrow.ttf`.

For more information, see [section 8.5, “Changing the Label settings”](#).



#### Adding a Label using **Assets** component

Prerequisite:

- A font file is located in the `$GUIDE_PROJECT_PATH/<project name>/resources` directory. For supported file types, see [section 6.20.1, “Fonts”](#).
- The content area displays a View.

##### Step 1

Drag a font file from the **Assets** component into the View.

The View displays the label with the selected font.

For more information, see [section 8.5, “Changing the Label settings”](#).

### 8.2.2.5. Adding a container



### Adding a Container

A Container allows grouping widgets.

Prerequisite:

- The content area displays a View.

#### Step 1

Drag a Container from the **Toolbox** into the View.

#### Step 2

In the content area, enlarge the Container by dragging one of its corners.

#### Step 3

Drag two or more widgets from the **Toolbox** into the Container.

The widgets are modeled as child widgets of the Container. Moving the Container moves its child widgets along with it.

## 8.2.2.6. Adding an Instantiator



### Adding an Instantiator

Prerequisite:

- The content area displays a View.

#### Step 1

Drag an Instantiator from the **Toolbox** into the View.

#### Step 2

Drag a widget from the **Toolbox** into the Instantiator.

The widget serves as a line template.

#### Step 3

Select the Instantiator, and go to the **Properties** component.

##### Step 3.1

For the `numItems` property enter a value that is greater than one.

##### Step 3.2

Add one of the following widget features to the Instantiator:

- ▶ **Box layout**

- ▶ **Flow layout**
- ▶ **Grid layout**
- ▶ **List layout**

For details, see [section 8.4.1, “Adding a widget feature”](#).

In the View, the child widget is displayed as many times as specified by the `numItems` property and in the layout specified by widget features for the Instantiator.

#### Step 4

Drag a widget from the **Toolbox** into the Instantiator.

You added the second child widget that serves as the second line template.

#### Step 5

Select the Instantiator, and go to the **Properties** component.

##### Step 5.1

Select the `lineMapping` and click .

##### Step 5.2

Click the **Add** button.

A new entry is added to the table.

##### Step 5.3

In the `Value` text box enter 0.

##### Step 5.4

Click the **Add** button.

The new entry is added to the table.

##### Step 5.5

In the `Value` text box enter 1.

You defined the order in which the line templates are instantiated.



### Example 8.1. Instantiation order

The `lineMapping` property defines the order of instantiation. For example, if you enter the values 1 | 0, the Instantiator instantiates the line template 1 as the first child widget and the line template 0 as the second child widget.

The `lineMapping` property is applied iteratively. This means that if for the `numItems` property you enter 10, the result is the order 1|0|1|0|1|0|1|0|1|0.

For a detailed example of how to use Instantiator, see [section 14.4, “Tutorial: Creating a list with dynamic content”](#).

#### NOTE



#### Linking of properties of the line templates

The following are the rules for linking:

- ▶ You cannot link properties between line templates.
- ▶ You cannot link from the outside of the Instantiator to its line templates.
- ▶ You can link from a line template to the corresponding Instantiator.

### 8.2.2.7. Adding an Animation

#### 8.2.2.7.1. Animating a widget



#### Animating a widget

Prerequisite:

- The content area displays a View.

##### Step 1

Drag one of the basic widgets from the **Toolbox** into the View.

##### Step 2

Drag an Animation from the **Toolbox** into the widget.

The **Animation editor** is displayed below the content area.

##### Step 3

Add a user-defined property of type `Conditional script`. For instructions, see [section 8.3.6, “Adding a user-defined property to a widget”](#).

##### Step 4

Next to the conditional script, click `{ }`.

An EB GUIDE Script editor opens.

##### Step 5

Enter the following EB GUIDE Script in the **On trigger** section:

```
function(v:arg0::bool)
{
    f:animation_play(v:this)
}
```

##### Step 6

Go to the View.

#### Step 7

Go to the **Animation editor** and next to **Animated properties** click **+**.

#### Step 8

Select the View.

A menu expands.

#### Step 9

Under **Animation properties** select the property that you want to animate and under **Animation curves** select a curve.

#### Step 10

Click **Accept**.

A new animation is added to the **Animated properties** list.

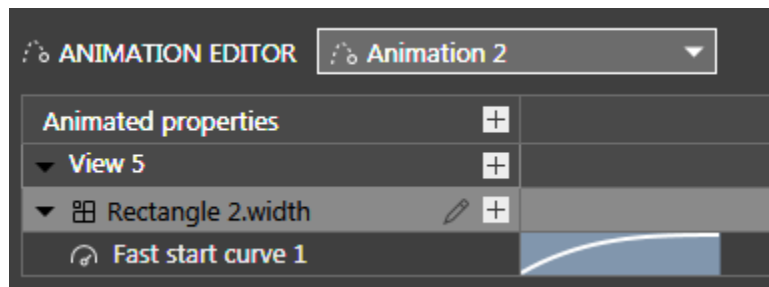


Figure 8.1. **Animation editor** with an animation curve

The **Properties** component displays the properties of the animation curve.

#### Step 11

In the **Properties** component, enter the properties for the animation curve.

Depending on the animation curve that you select, you need to change the default values for properties **end**, **constant**, or **value**. Otherwise there is no change to the animated value. For example, you can enter the following values: 50, 200, -400.

#### Step 12

Start the simulation.

The value is changed as specified by the animation curve.

### 8.2.2.7.2. Animating a view transition

#### Adding an entry animation



### Adding an entry animation

The following instruction guides you through the process of adding an entry animation to a View state. The instruction also applies to exit animations, pop-up on animations and pop-up off animations. For more information, see [section 6.2.4, “View transition animation”](#) and [section 15.10.1, “View”](#).

Prerequisite:

- A View state and a View are added to the EB GUIDE model.
- The **VTA** component is opened.

#### Step 1

In the **VTA** component, click .

#### Step 2

In the context menu, select **Entry animation**.

The **Animation** editor opens.

The **Properties** component displays the properties of the added entry animation.

#### Step 3

Animate all available widget properties in the dependent view.

Define the entry animation in the **Animation** editor.

## Adding a change animation



### Adding a change animation

The following instruction guides you through the process of adding a change animation to a View state or view template.

Prerequisite:

- **VTA** component is opened.
- The **Main** state machine contains two view states.

#### Step 1

Select `View state 1` in the **Main** state machine.

#### Step 2

In the **VTA** component, click .

#### Step 3

In the context menu, select **Change animation**.

A dialog opens.

#### Step 4

Select **View 2**.

#### Step 5

Click **Accept**.


The name of the destination view is shown next to the change animation name.

The **Animation** editor opens.

The **Properties** component displays the properties of the added change animation.

The start view is shown in the project editor.

#### Step 6

To edit the destination view, in the **VTA** component, click .

#### Step 7

To add animation properties, click  in the **Animation** editor to add the respective categories.

Animate all available widget properties in the dependent view.

### Rearranging animations



#### Rearranging animations

The following instruction guides you through the process of rearranging animations of a View state or view template. This instruction also applies to exit animations, change animations, pop-up on animations and pop-off animations.

Prerequisite:

- **VTA** component is opened.
- You added multiple entry animations.

#### Step 1

In the **VTA** component, select the prioritization text box next to the entry animation that you want to start first.

#### Step 2

Change the value of the entry animation to 0.

The order of the entry animations is changed. The edited entry animation starts first and all following entries are incremented by one.



The value 0 means that the animation is evaluated to be played first. Subsequent values mean that the animations are evaluated to be played in a subsequent order. Only the entry animation that has the highest priority and the fulfilled condition is played.

### 8.2.2.7.3. Animating with a script curve



#### Getting the output of the script curve

Prerequisite:

- The content area displays the **Main** state machine.
- The **Main** state machine contains an initial state and a View state.
- The initial state has a transition to the View state.

#### Step 1

Rename the View to `FirstView` and open it.

#### Step 2

Drag an Animation from the **Toolbox** into `FirstView`.

#### Step 3

Add a user-defined property of type `Conditional script`. For instructions, see [section 8.3.6, “Adding a user-defined property to a widget”](#).

#### Step 4

Next to the conditional script, click `{ }`.

An EB GUIDE Script editor opens.

#### Step 5

Enter the following EB GUIDE Script in the **On trigger** section:

```
function(v:arg0::bool)
{
    f:animation_play(v:this)
}
```

#### Step 6

Go to the `FirstView`.

#### Step 7

In the **Animation editor**, next to **Animated properties** click **+** and then click `FirstView`.

A menu expands.

#### Step 8

Under `FirstView` select the `y` property and then **Script curve**.

#### Step 9

Click **Accept**.

An Animation is added to the **Animation editor**.

#### Step 10

Rename the new Animation to `ScriptCurveMonitoring`.

#### Step 11

In the **Properties** component, go to **Default widget properties** and next to the `curve` property click `{}`. EB GUIDE Script editor is displayed.

Enter the following EB GUIDE Script:

```
function(v:diff::int, v:t_anim::int)
{
f:trace_string("Diff : "+ f:int2string(v:diff) + " t_anim: " + f:int2string(v:t_anim))
    0::int
}
```

Click **Accept**.



### Saving and testing the EB GUIDE model

Prerequisite:

- You completed the previous instruction.

#### Step 1

To save the project, click  in the command area.

#### Step 2

To start the simulation, click  in the command area.

In EB GUIDE Monitor, observe the **Logger** component. `v:diff` shows the animation is executed every 16 milliseconds. `v:t_anim` shows how long the animation ran at this point since the start. See the figure below.


 traceString 'Diff : 0 t_anim: 0'	12:21:07.779
 traceString 'Diff : 16 t_anim: 16'	12:21:07.779
 traceString 'Diff : 16 t_anim: 32'	12:21:07.779
 traceString 'Diff : 17 t_anim: 49'	12:21:07.779
 traceString 'Diff : 19 t_anim: 68'	12:21:07.779
 traceString 'Diff : 16 t_anim: 84'	12:21:07.779
 traceString 'Diff : 16 t_anim: 100'	12:21:07.779

Figure 8.2. EB GUIDE Monitor messages

#### 8.2.2.7.4. Animating a datapool item



## Animating a datapool item

Prerequisite:

- A datapool item of one of the following types is added: Boolean, color, float, integer.
- The content area displays a View.

### Step 1

Drag an Animation from the **Toolbox** into the View.

The **Animation editor** is displayed below the content area.

### Step 2

Add a user-defined property of type `Conditional script`. For instructions, see [section 8.3.6, “Adding a user-defined property to a widget”](#).

### Step 3

Next to the conditional script, click `{ }`.

An EB GUIDE Script editor opens.

### Step 4

Enter the following EB GUIDE Script in the **On trigger** section:

```
function(v:arg0::bool)
{
    f:animation_play(v:this)
}
```

### Step 5

Go to the View.

### Step 6

Go to the **Animation editor** and next to **Animated properties** click **+**.

### Step 7

Select **Datapool item**.

A menu expands.

### Step 8

Under **Animation properties** select the datapool item that you want to animate and under **Animation curves** select a curve.

### Step 9

Click **Accept**.

A new animation is added to the **Animated properties** list.

The **Properties** component displays the properties of the animation curve.

#### Step 10

In the **Properties** component, enter the properties for the animation curve.

Depending on the animation curve that you select, you need to change the default values for properties `end`, `constant`, or `value`. Otherwise there is no change to the animated value. For example, you can enter the following values: 50, 200, -400.

#### Step 11

Start the simulation.

The value is changed as specified by the animation curve.

### 8.2.2.7.5. Animating colors



#### Animating colors

Prerequisite:

- The content area displays a View.

#### Step 1

Drag a Rectangle from the **Toolbox** into the View.

#### Step 2

Drag an Animation from the **Toolbox** into the Rectangle.

The **Animation editor** is displayed below the content area.

#### Step 3

Add a user-defined property of type `Conditional script`. For instructions, see [section 8.3.6, “Adding a user-defined property to a widget”](#).

#### Step 4

Next to the conditional script, click `{ }`.

An EB GUIDE Script editor opens.

#### Step 5

Enter the following EB GUIDE Script in the **On trigger** section:

```
function(v:arg0::bool)
{
    f:animation_play(v:this)
}
```

#### Step 6

Go to the View.

#### Step 7

Go to the **Animation editor**. Next to **Animated properties** click **+**.

#### Step 8

Select the View.

A menu expands.

#### Step 9

Under **Animation properties** select the `fillColor` of the Rectangle.

#### Step 10

Under **Animation curves** select the Linear interpolation curve.

#### Step 11

Click **Accept**.

A new animation is added to the **Animated properties** list.

The **Properties** component displays the properties of the animation curve.

#### Step 12

In the **Properties** component, set the `start` property to green.

#### Step 13

Set the `end` property to red.

#### Step 14

Start the simulation.

The color is changed from green to red. All of the colors that lie in between red and green according to their RGBA8888 value are displayed during the animation.

### 8.2.2.8. Adding an Alpha mask



#### Adding an Alpha mask

For details on Alpha mask, see [section 15.10.2.1, "Alpha mask"](#).

Prerequisite:

- The `$GUIDE_PROJECT_PATH/<project name>/resources` folder contains an image.
- The content area displays a View.

#### Step 1

Drag the Alpha mask from the **Toolbox** into the View.

#### Step 2

Go to the **Properties** component and select an image from the `image` drop-down list box.

#### NOTE



#### Supported image file types for alpha mask

The available image formats depend on the implementation of the renderer. The renderers for OpenGL ES 2.0 or higher support .png files and .jpg files. RGB images are converted to grayscale images before being used as alpha masks. Grayscale images are used as is. The alpha channel in the image is ignored.

Alpha mask functionality is not applied to 9-patch images. 9-patch images are handled the same way the PNG and JPEG file formats are.

#### Step 3

Add one of the basic widgets from the **Toolbox** as a child widget to the Alpha mask.

The alpha channel, i.e. the opacity of the child widget is controlled with the Alpha mask.

## 8.2.3. Adding a 3D widget to a View

### 8.2.3.1. Adding a Scene graph to a View



#### Adding a Scene graph to a View

For restrictions and recommendations, see [section 6.1.2, “Settings for 3D graphic files”](#).

Prerequisite:

- A 3D graphic file is available. The file contains a camera, a light source, and one object containing a mesh and at least one material. For supported 3D graphic file formats, see [section 6.1.1, “Supported 3D graphic formats”](#).
- The content area displays a View.

#### Step 1

Drag a Scene graph from the **Toolbox** into the View.

The View displays the empty bounding box.

#### Step 2

In the **Properties** component, click **Import file**.

A dialog opens.

#### Step 3

Navigate to the folder where the 3D graphic file is stored.

#### Step 4

Select the 3D graphic file.

#### Step 5

Click **Open**.

The import starts. A dialog opens.

#### Step 6

Click **OK**.

The View displays the 3D graphic. The **Navigation** component displays the imported widget tree with the Scene graph as a parent node. If the imported 3D scene has animations, the Linear key value interpolation integer curve or Linear key value interpolation float curve are added. Note that you cannot modify the underlying key-value pairs of these curves in EB GUIDE Studio.

#### TIP



#### Multiple import

Import of multiple 3D graphics within one Scene graph is possible.

After importing, multiple 3D graphics are rendered on top of each other. To display 3D objects separately, use the `visible` property of `RootNode`.

## 8.2.4. Importing a .psd file to a View



### Adding a .psd file into a View

For background information, see [section 6.19, "Photoshop file format support"](#).

Prerequisite:

- A .psd file is available in `$GUIDE_PROJECT_PATH/<project name>/resources` or a subfolder.
- The content area displays a View.

#### Step 1

In the **Assets** component, select the corresponding folder.

#### Step 2

From the preview area, drag the .psd file into the content area.

An import status message appears.

#### Step 3

Click **OK**.

If the import was successful, the **Navigation** component displays a new widget tree. The top element is a container named after the .psd file.

In the `$GUIDE_PROJECT_PATH/<project name>/resources` folder, a subfolder is created, that contains all of the extracted images.

**NOTE**



**Multiple lines**

If a text layer contains text for more than one line, you need to add widget feature **Multiple lines** and edit the properties accordingly. For more information, see [section 8.4.1, “Adding a widget feature”](#) and [section 15.11.1.5, “Multiple lines”](#).

## 8.2.5. Extracting images from a .psd file



### Extracting images from a .psd file

When you extract images from a .psd file instead of importing them, no widget tree is created. For background information, see [section 6.19, “Photoshop file format support”](#).

Prerequisite:

- A .psd file is available in `$GUIDE_PROJECT_PATH/<project name>/resources` or in subfolders.

Step 1

In the **Assets** component, select the corresponding folder.

Step 2

Right-click the .psd file and select **Extract images from .psd file**.

An import status message appears.

Step 3

Click **OK**.

In the `$GUIDE_PROJECT_PATH/<project name>/resources` folder, a subfolder is created, that contains all the extracted images. The subfolder is named after the .psd file.

## 8.2.6. Importing IBL files

For background information, see [section 6.14, “Image-based lighting”](#).



### Importing IBL files

To import IBL files they must be transformed into the .ebibl format first. This is done using IBLGenerator



Prerequisite:

- An EB GUIDE project is created.
- Your IBL files are in either `.pfm` or `.hdr` format.
- You have admin rights.
- A Scene graph widget is added to the EB GUIDE model.
- A 3D file is imported.

Step 1

Open the command line prompt as administrator.

Step 2

Navigate to the IBLGenerator installation path. It is in your EB GUIDE Studio installation directory in the `tools` folder, `$GUIDE_INSTALL_PATH\tools\IBLGenerator`.

Step 3

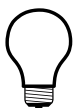
Type the command to transform your file into to `.ebibl` format. It could look like this:

```
IBLGenerator.exe -i yourfile.hdr -o yourfile.ebibl -p latlong -q 1
```

- ▶ `-i`: The input file name
- ▶ `-o`: The output file name
- ▶ `-p`: The parametrization type. Other types are cube and sphere.
- ▶ `-q`: The quality level. The quality level with 1 as the low quality and 10 as the highest quality. Higher levels of quality need significantly more processing time.

The `.ebibl` file is placed in the folder that you provided.

**TIP**



**IBLGenerator help**

To see a list of options for IBLGenerator, run it with parameters `-h: IBLGenerator.exe -h`

Step 4

Copy the `.ebibl` file into the resources folder of your EB GUIDE model. Now you can use the `.ebibl` file in a Scene graph node.

Step 5

In EB GUIDE Studio, in the **Toolbox** component, from the **3D widgets** drag the Image-based light into a Scene graph node.

Step 6

In the **Properties** component, next to the `ibl` property, select your `.ebibl` file.

The import of the IBL file is finished.

**TIP**



**Best results for IBL**

For best results, adapt the properties of the Image-based light and use PBR GGX material or PBR Phong material.

To only illuminate the scene with image-based lighting, disable all other light sources.

## 8.2.7. Deleting a widget from a View



### Deleting a widget from a View

Prerequisite:

- The EB GUIDE model contains a widget.

Step 1

In the **Navigation** component, right-click a widget.

Step 2

In the context menu, click **Delete**.

The widget is deleted.

**TIP**



**Deleting widgets from the content area**

It is also possible to delete a widget by selecting it in the content area and pressing the **Delete** key.

## 8.3. Working with widget properties

### 8.3.1. Positioning a widget



### Positioning a widget

Positioning a widget means adjusting the widget's  $x$  and  $y$  properties. The point of origin where both  $x$  and  $y$  have the value 0 is the top left corner of the parent widget.

Prerequisite:

- The content area displays a View.
- The View contains a widget.

Step 1

Select a widget.

The **Properties** component displays the properties of the selected widget.

Step 2

To define the x-coordinate of the widget enter a value in the *x* text box.

Step 3

To define the y-coordinate of the widget enter a value in the *y* text box.

Step 4

Click outside the text box.

The content area displays the widget at the entered position.

**TIP**



**Alternative approach**

To position a widget by visual judgment, select the widget in the content area and move it with the mouse.

## 8.3.2. Resizing a widget



### Resizing a widget

Prerequisite:

- The content area displays a View.
- The View contains a widget.

Step 1

Select a widget.

The **Properties** component displays the properties of the selected widget.

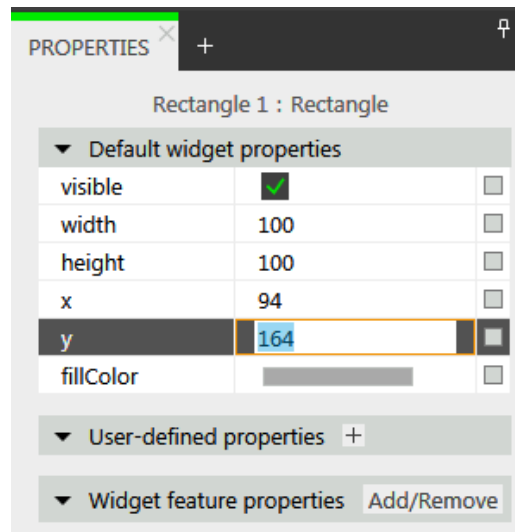


Figure 8.3. Properties of a rectangle

#### Step 2

To define the height of the widget enter a value in the `height` text box.

#### Step 3

To define the width of the widget enter a value in the `width` text box.

#### Step 4

Click outside the text box.

The content area displays the widget with the entered size.

#### NOTE



#### Negative values

Do not use negative values for `height` and `width` properties. EB GUIDE Studio treats negative values as 0, this means the respective widget will not be depicted.

#### TIP



#### Alternative approach

To resize a widget by visual judgment, select the widget in the content area and drag one of its corners with the mouse.

### 8.3.3. Linking between widget properties



## Linking between widget properties

In order to make sure that two widget properties have the same value at all times, you can link two widget properties. As an example, the following instructions show you how to link the `width` property of a rectangle to the `width` property of a view.

You can only link the properties of widgets within the same View.

You cannot link to properties of child widgets of an Instantiator.

For more information, see [section 6.16, “Linking”](#).

Prerequisite:

- The EB GUIDE model contains a View state.
- The View contains a Rectangle.
- The `width` property of the Rectangle is not a scripted value.

### Step 1

Select the Rectangle.

The **Properties** component displays the properties of the Rectangle.

### Step 2

In the **Properties** component, go to the `width` property, and click the  button next to the property.

A menu expands.

### Step 3

In the menu, click **Add link to widget property**.

A dialog opens.

### Step 4

In the dialog, go to the view, and select its `width` property.

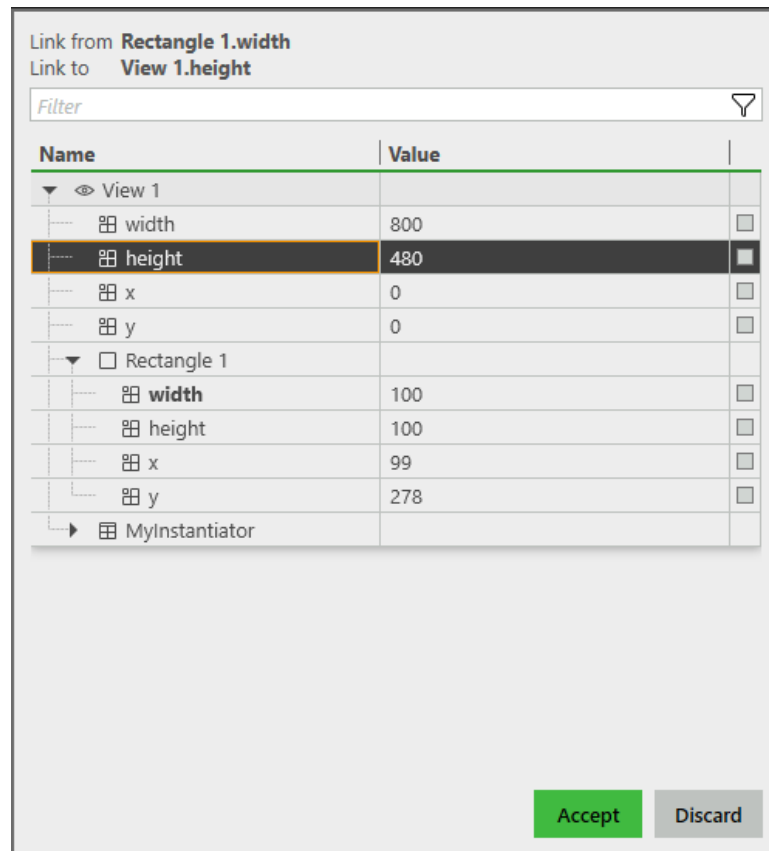



Figure 8.4. Linking between widget properties


#### Step 5

Click **Accept**.

The dialog closes. The  button is displayed next to the `width` property. It indicates that the `width` property of the rectangle is now linked to the `width` property of the view. Whenever you change the width of the View, the width of the Rectangle changes and vice versa.


---

**NOTE****Link source and link target**

The  button is only displayed next the link source. It is not displayed for the link target.

---

**TIP****Removing the link**

To remove the link, click the  button again. In the menu that opens click **Remove link**.

---

### 8.3.4. Linking a widget property to a datapool item



#### Linking a widget property to a datapool item

In order to make sure that a widget property and a datapool item have the same value at all times, you can link a widget property to a datapool item. As an example, the following instructions show you how to link the `image` property of an image to a new datapool item.

For more information, see [section 6.16, "Linking"](#).

Prerequisite:

- The EB GUIDE model contains a View state.
- The View contains an Image.
- The `image` property of the Image is not a scripted value.

#### Step 1

Select the Image.

The **Properties** component displays the properties of the Image.

#### Step 2

In the **Properties** component, go to the `image` property, and click the  button next to the property.

A menu expands.

#### Step 3

In the menu, click **Add link to datapool item**.

A dialog opens.

#### Step 4

To add a new datapool item, enter a name in the text box.


Step 5

Click **Add datapool item**.

Step 6

Click **Accept**.


A new datapool item is added.

The dialog closes. The  button is displayed next to the `image` property. It indicates that the `image` property is now linked to a datapool item. The created link is bidirectional. Whenever you change the Image, the datapool item changes and vice versa.

**NOTE**

**Link source and link target**




The  button is only displayed next the link source. It is not displayed for the link target.

**TIP**

**Removing the link**



To remove the link, click the  button again. In the menu that opens, click **Remove link**.

## 8.3.5. Linking to a list element

You can link a widget property to a list element of a datapool item or a widget property of type list. You can link list elements directly or indirectly. Directly means that it is not possible to change the list index at run-time. Indirectly means that it is possible to change the list index at run-time.



### Linking to a list element of a datapool item

Linking is only possible if the types of the widget property and of the datapool item match. As an example, the following instructions show you how to link the `fillColor` property of a rectangle to a datapool item of type color list.

For more information, see [section 6.16, “Linking”](#). For instructions on how to edit datapool items of type list, see [section 9.3.2, “Editing datapool items of type list”](#). For a tutorial on how to create a dynamic list, see [section 14.4, “Tutorial: Creating a list with dynamic content”](#).

Prerequisite:

- A datapool item of type color list is added and has several list elements.




- A Rectangle is added.
- The `fillColor` property of the Rectangle is not a scripted value.
- For indirect linking, a datapool item of type integer or a user-defined property of type integer is added.

#### Step 1

Select the Rectangle.

The **Properties** component displays the properties of the Rectangle.

#### Step 2

In the **Properties** component, next to the `fillColor` property, click  and select **Add link to a datapool item**.

An editor opens.

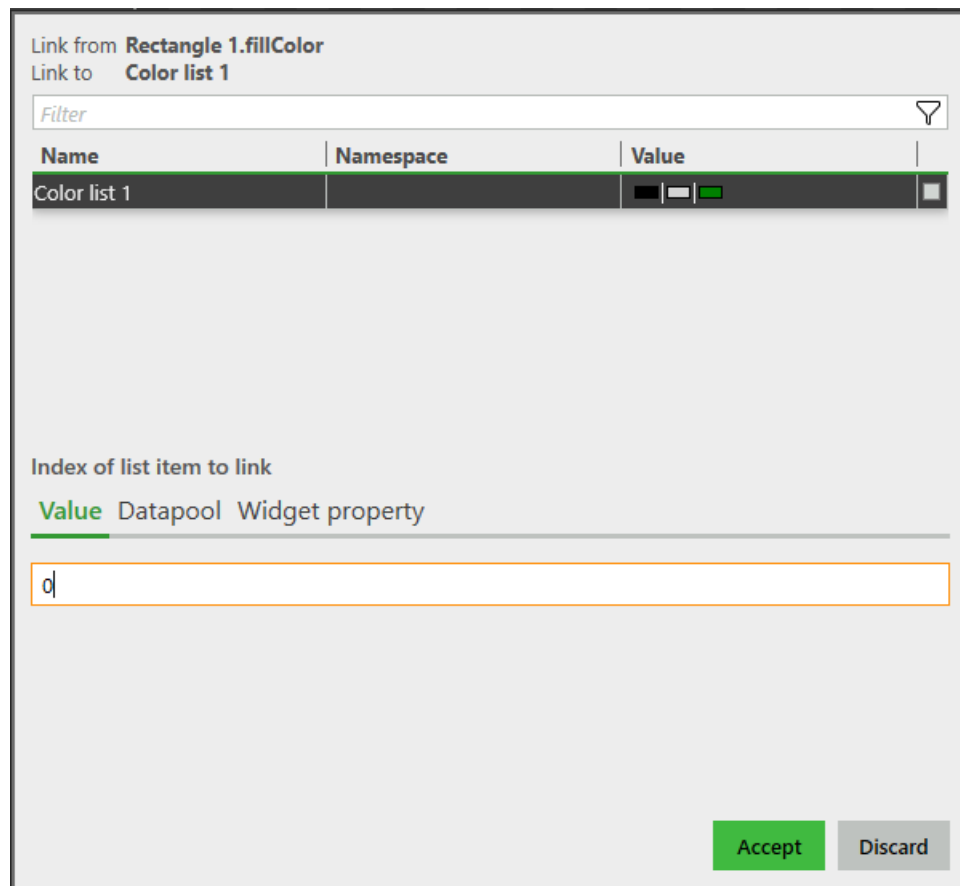


Figure 8.5. Linking to a datapool item

#### Step 3

Select the datapool item of type color list.

#### Step 4


Define the index of the list item that you want to link to:

- ▶ If you want to link directly, go to the **Value** tab and enter the index of the list element.
- ▶ If you want to link indirectly, do either of the following:
  - ▶ If you want to use a datapool item, go to the **Datapool** tab and select the datapool item of type integer.
  - ▶ If you want to use a widget property, go to the **Widget property** tab and select the widget property of type integer.

You can now use this datapool item or widget property in e.g. EB GUIDE Script and change the linked list element dynamically.

#### Step 5

Click **Accept**.

The dialog closes. The  button is displayed next to the `fillColor` property. It indicates that the `fillColor` property is now linked to a datapool item. The created link is bidirectional. Whenever you change the list elements, the property changes.



#### Linking to a list element of a widget property

Linking is only possible if the types of the widget properties match. As an example, the following instructions show you how to link the `fillColor` property of a Rectangle to a widget property of type color list.

For a tutorial on how to create a dynamic list, see [section 14.4, "Tutorial: Creating a list with dynamic content"](#).

Prerequisite:

- A user-defined property of type color list is added and has several list elements.
- A Rectangle is added.
- The `fillColor` property of the Rectangle is not a scripted value.
- For indirect linking, a datapool item of type integer or a user-defined property of type integer is added.

#### Step 1

Select the Rectangle.

The **Properties** component displays the properties of the Rectangle.

#### Step 2

In the **Properties** component, next to the `fillColor` property, click  and select **Add link to a widget property**.

An editor opens.

#### Step 3

Select the widget property of type color list.

#### Step 4


Define the index of the list item that you want to link to:

- ▶ If you want to link directly, go to the **Value** tab and enter the index of the list element.
- ▶ If you want to link indirectly, do either of the following:
  - ▶ If you want to use a datapool item, go to the **Datapool** tab and select the datapool item of type integer.
  - ▶ If you want to use a widget property, go to the **Widget property** tab and select the widget property of type integer.

You can now use this datapool item or widget property in e.g. EB GUIDE Script and change the linked list element dynamically.

#### Step 5


Click **Accept**.

The dialog closes. The  button is displayed next to the `fillColor` property. It indicates that the `fillColor` property is now linked to a widget property. The created link is bidirectional. Whenever you change the list elements, the property changes.

#### TIP

#### Removing the link



To remove the link, click the  button again. In the menu that opens, click **Remove link**.

## 8.3.6. Adding a user-defined property to a widget



### Adding a user-defined property to a widget

Prerequisite:

- The EB GUIDE model contains a View state.
- The View contains a widget.

#### Step 1

Select a widget.

The **Properties** component displays the properties of the selected widget.

#### Step 2

In the **Properties** component, go to the **User-defined properties** category, and click **+**.

A menu expands.

Step 3

In the menu, click a type for the user-defined property.

A new widget property of the selected type is added to the widget.

Step 4

Rename the property.

### 8.3.6.1. Adding a user-defined property of type `Function () : bool`



#### Adding a user-defined property of type `Function () : bool`

A property of type `Function () : bool` is a function that has no parameters and returns a boolean value. You call the function in EB GUIDE Script in the way you address widget properties followed by the arguments list.

Prerequisite:

- The EB GUIDE model contains a View state.
- The View contains a widget.

Step 1

Select a widget.

The **Properties** component displays the properties of the selected widget.

Step 2

In the **Properties** component, go to the **User-defined properties** category, and click **+**.

A menu expands.

Step 3

In the menu, click `Function () : bool`.

A new widget property of type `Function () : bool` is added to the widget.

Step 4

Rename the property.

Step 5

Next to the datapool item, select the **Value** column and click **{ }**.

An EB GUIDE Script editor opens.

Step 6

Define the behavior of the new function using EB GUIDE Script.

Step 7

Click **Accept**.



**Example 8.2.**  
**Calling a property of type `Function () : bool`**

In your EB GUIDE model, there is a rectangle called `Background color`. You added a property of type `Function () : bool` to it. The property is called `change`.

How you call the property in an EB GUIDE Script depends on the position of the widget in the widget tree:

▶ `v:this->"Background color".change()`

or

▶ `v:this->^->"Background color".change()`

### 8.3.7. Renaming a user-defined property



**Renaming a user-defined property**

Prerequisite:

- The EB GUIDE model contains a widget with a user-defined property.

Step 1

In the **Navigation** component, select the widget with the user-defined property.

Step 2

In the **Properties** component, right-click the property name.

A menu expands.

Step 3

In the menu click **Rename**.

Step 4

Enter a name for the property.

Step 5

Press the **Enter** key.

### 8.3.8. Editing a property of type list



### Editing a property of type list

For more information on properties of type list, see [section 15.3.12, “List”](#).

For more information on resource management, see [section 6.20, “Resource management”](#) and [section 6.6, “Components of the graphical user interface”](#).

Prerequisite:

- The EB GUIDE model contains a View state.
- The View contains a widget.
- The widget has a property of type list.

#### Step 1

In the **Properties** component, select the property of type list, and next to it, click the  button.

The **Edit** dialog opens.

#### Step 2

To add a new entry to the list, click the **Add** button.

A new row is added.

#### Step 3

To edit a value, in the table, click the corresponding row in the **Value** column.

#### TIP



#### Assets component

If your property is of type font list, ibl list, image list, or mesh list, the **Edit** dialog has an additional **Assets** component. You can drag and drop the asset from the **Assets** component to the corresponding row in the **Value** column.

#### Step 4

When you are finished editing the list, click the **Accept** button.

The dialog closes.

## 8.3.9. Managing order and visibility of widgets

In EB GUIDE Studio you have several possibilities, to define the order, layers, or visibility of widgets. The following possibilities are available and whether you can apply them depends on your use case.

When you just want to change which widget is on top in relation to other widgets in a view, use the **Navigation** tree. In the **Navigation** tree, the order of the widgets determines which widget is going to be displayed on top.

If the widgets are on the same branch, the widget with the higher position in the tree is displayed below. If widgets are on different branches, the widget on the higher level is displayed on top.

When you want to have a pop-up that is displayed on top of everything else, use dynamic state machines. For instructions, see [section 14.1, “Tutorial: Adding a dynamic state machine”](#).

When you have complex menus that change according to what users select, use the **Child visibility selection** widget feature. With this widget feature you can control the visibility of the child widgets of a widget. It overwrites the `Visibility` property of the affected widget. This feature has two usage possibilities:

- ▶ Making a single child widget visible. For this use case you only need the index of the widget in the widget tree. For instructions, see [“Making a single child widget visible”](#).
- ▶ Making multiple child widgets visible. For this use case you need to define groups of child widgets and provide one group's identifying `Value`. For instructions, see [“Making multiple child widgets visible”](#).



#### Making a single child widget visible

For more information on widget features, see [section 8.4, “Extending a widget by widget features”](#).

Prerequisite:

- An EB GUIDE project is opened in EB GUIDE Studio.
- The EB GUIDE model contains widgets.

##### Step 1

Add the **Child visibility selection** widget feature to the parent widget.

##### Step 2

In the **Navigation** component, check in which position the child widget is. 0 is the first position.

##### Step 3

In the **Properties** component, in the `containerIndex` text box enter the position of the child widget.

This child widget is now the only child widget that is visible.



#### Making multiple child widgets visible

For more information on widget features, see [section 8.4, “Extending a widget by widget features”](#).

Prerequisite:

- An EB GUIDE project is opened in EB GUIDE Studio.
- The EB GUIDE model contains more than three widgets.

#### Step 1

Add the **Child visibility selection** widget feature to the parent widget.

#### Step 2

In the **Properties** component, select `containerMapping` and then click .

An editor opens.

#### Step 3

Define a mapping between the group of widgets and child widgets.

The **Index** column contains the child widget index. In the **Value** column, enter the group to which you want to map the child widgets.

#### Step 4

Click **Accept**.

#### Step 5

In the `containerIndex` text box, enter the group's `Value` that should be visible.

This group of widgets is now visible. Widgets that are not mapped to this group are invisible.

## 8.4. Extending a widget by widget features

Widget features add more functionality for the appearance and behavior of widgets. Adding a widget feature to a widget means adding one or more widget properties. The offered widget features depend on the type of the widget.

### 8.4.1. Adding a widget feature



#### Adding a widget feature

Prerequisite:

- The EB GUIDE model contains a widget.

#### Step 1

In the **Navigation** component, click a widget.

The **Properties** component displays the properties of the selected widget.

#### Step 2

In the **Properties** component, go to the **Widget feature properties** category, and click **Add/Remove**.



The **Widget features** dialog is displayed.

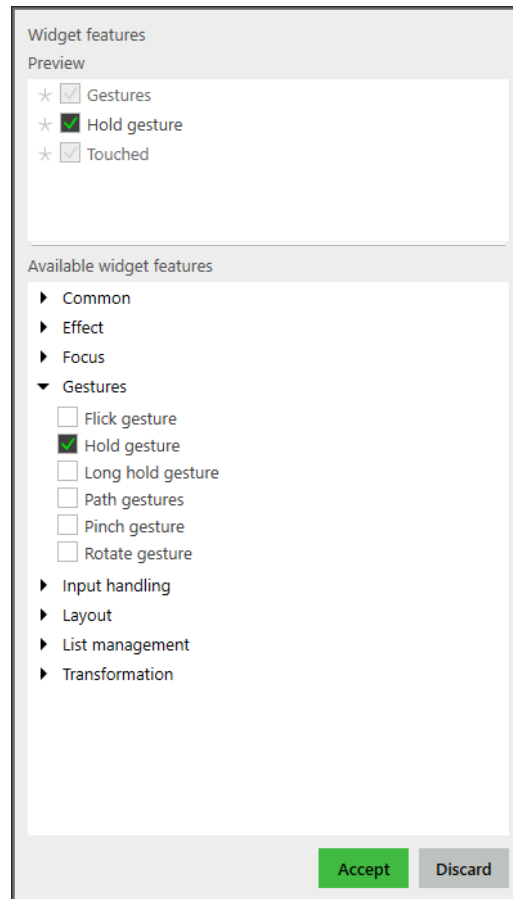


Figure 8.6. Widget features dialog

### Step 3

Under **Available widget features**, expand a category, and select the widget feature you want to add.

The selected widget feature as well as dependent widget features that are activated automatically along with it, are listed under **Preview**.

Click **Accept**.

### NOTE



#### Dependencies between widget features

Some widget features require other widget features. Therefore, in some cases, if you select a widget feature, other widget features are selected automatically.

For example, you want to add the widget feature **Moveable**. In addition the widget features **Touched** and **Touch Move** are added automatically.

For a list of widget features grouped by categories, see [section 15.11, “Widget features”](#).

For tutorials, see the following:

- ▶ [section 14.3, “Tutorial: Modeling a path gesture”](#)
- ▶ [section 14.4, “Tutorial: Creating a list with dynamic content”](#)
- ▶ [section 14.2, “Tutorial: Modeling button behavior with EB GUIDE Script”](#)

## 8.4.2. Removing a widget feature



### Removing a widget feature

Prerequisite:

- The EB GUIDE model contains a widget.
- At least one widget feature is added to the widget.

#### Step 1

In the **Navigation** component, click a widget.

The **Properties** component displays the properties of the selected widget.

#### Step 2

In the **Properties** component, go to the **Widget feature properties** category and click **Add/Remove**.

The **Widget features** dialog is displayed.

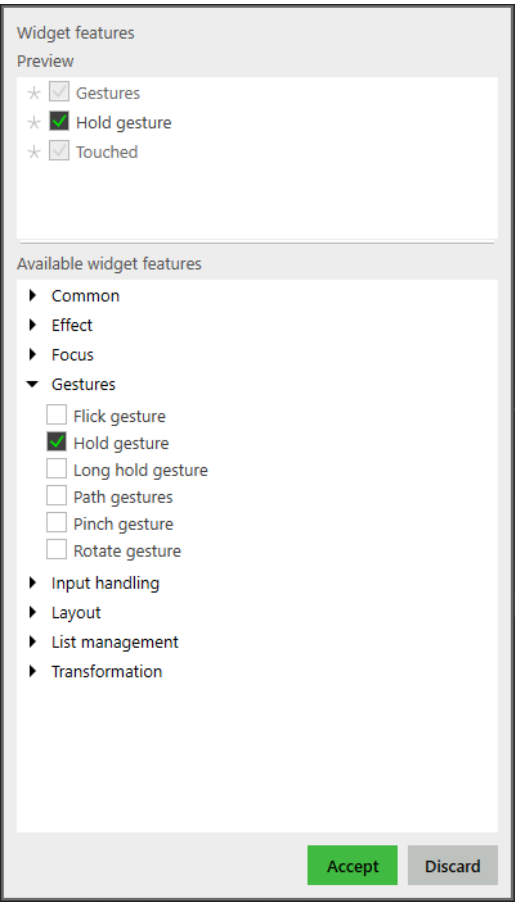


Figure 8.7. Widget features dialog

**Step 3**  
Under **Preview** clear the widget feature you want to remove.

Click **Accept**.

The related widget feature properties are removed from the **Properties** component.

**NOTE**



**Removing widget features with dependencies**

Widget features which were added automatically due to dependencies are not deleted automatically. They cannot be removed directly. Clear the parent widget feature before you clear the child widget feature.

## 8.5. Changing the Label settings

## 8.5.1. Changing the font size of a Label

### NOTE



### Resizing a Label

If you change the `width` and `height` properties of the Label or drag the corners of the bounding box, only the bounding box will change its size. The size of the text in the Label will stay the same.

For more instructions on resizing a widget, see [section 8.3.2, “Resizing a widget”](#).

The truncation behavior is as follows:

- ▶ When you scale down the size of the bounding box upright, the text is truncated line-by-line if the Label has several lines, or pixel-by-pixel if the Label has only one line.
- ▶ When you scale down the size of the bounding box across, the text is truncated character-by-character if the label has several characters, or pixel-by-pixel if the Label has only one character.

To visualize that the text is truncated, you can add a character or a set of characters as a truncation symbol, for example `. . .`. To define the truncation symbol, use the **Text truncation** widget feature. For more information, see [section 15.11.1.10, “Text truncation”](#).



### Changing the font size of a Label

Prerequisite:

- The content area displays a View.
- The view contains a Label.

#### Step 1

Select the Label.

The **Properties** component displays the properties of the selected Label.

#### Step 2

To define the size of the text, enter a value in the `font` text box.

#### Step 3

Click outside the text box.

The content area displays the text in the Label with the entered size.

## 8.5.2. Changing the font of a Label



## Changing the font of a Label

### Prerequisite:

- A font file is located in `$GUIDE_PROJECT_PATH/<project name>/resources`. For supported file types, see [section 6.20.1, “Fonts”](#).
- The EB GUIDE model contains a View state.
- The View contains a Label.

### Step 1

Select the Label in the View.

### Step 2

In the **Properties** component, select a font from the `font` combo box.

Alternatively, drag a font file from the **Assets** component into the `font` combo box.

The view displays the Label with the new font. Note that if you select an `.fnt` bitmap font, the size of the font is fixed and you cannot change it in the `font` property of the Label.

For instructions on how to use multifont support, see [section 8.5.5, “Managing multifont support”](#).

## 8.5.3. Changing the line spacing

Each font has a line spacing defined by default. You can change this spacing for each label with the `lineGap` property in the **Font metrics** widget feature. When the **Multiple lines** widget feature is also added to this Label, you can additionally define the line spacing with the `lineOffset` property. It is possible to set both properties for the same Label. Take into account that in this case the settings of both properties can offset each other.

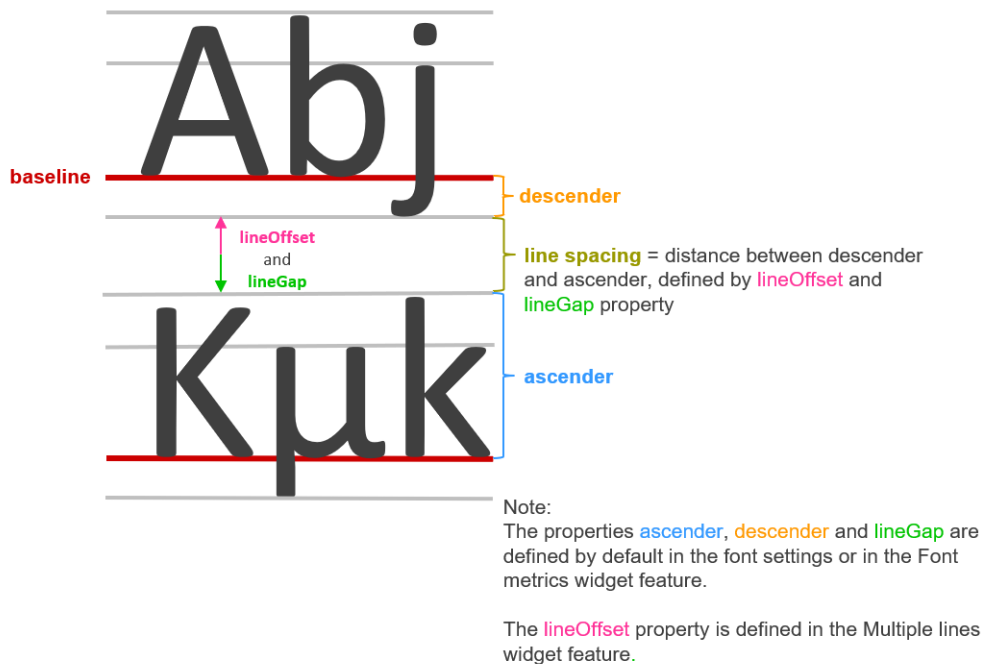


Figure 8.8. Line spacing defined by `lineOffset` and `lineGap`

### 8.5.3.1. Changing the default line spacing

Each font has a line spacing defined by default. You can change the line spacing for each Label with the `lineGap` property in the **Font metrics** widget feature.

#### NOTE



#### Interaction with `lineOffset` property

Take into account that when the **Multiple lines** widget feature is also added to this Label, the line spacing can also be changed there through the `lineOffset` property and that both properties can offset each other.



#### Changing the `lineGap` property in the **Font metrics** widget feature

The following instruction shows you how to change the default line spacing of a font.

Prerequisite:

- The View contains a Label.
- A property or a datapool item of type font exists.

#### Step 1

Select the Label.

#### Step 2

In the **Properties** component, go to the **Widget feature properties** and click **Add/Remove**.

The **Widget feature** dialog is displayed.

#### Step 3

Under **Available widget features**, expand the **Common** category, and select **Font metrics**.

#### Step 4

Click **Accept**.

The related widget feature properties are added to the font and displayed in the **Properties** component.

#### Step 5

In the **Properties** component, select the `lineGap` property and change the value.

### 8.5.3.2. Changing the line spacing for multiple lines



Changing the `lineOffset` property in the **Multiple lines** widget feature

The **Multiple lines** widget feature enables line breaks in a Label. The following instruction shows you how to change the line spacing for multiple lines.

#### NOTE



#### Interaction with `lineGap` property

Take into account that changing the `lineOffset` property has an impact on the line spacing defined in the `lineGap` property of the **Font metrics** widget feature and that both properties can offset each other.

Prerequisite:

- The View contains a Label and the **Multiple lines** widget feature is added to it.
- A property or a datapool item of type font exists.

#### Step 1

Select the Label.

#### Step 2

In the **Properties** component go to the **Multiple lines** widget feature.

#### Step 3

Change the value of the `lineOffset` property.

### 8.5.4. Changing the text position

To change the position of the text within a label, you can use the following widget properties:

- ▶ `horizontalAlign` default widget property
- ▶ `verticalAlign` default widget property
- ▶ `lineOffset` property from the **Multiple lines** widget feature
- ▶ `ascender`, `descender` and `lineGap` property from the **Font metrics** widget feature

It is possible to use all the above mentioned properties for the same label.



Changing the text position with `ascender` and `descender`

The following instruction shows you how to change the position of the characters and thus of the text with the `ascender` and `descender` properties from the **Font metrics** widget feature.

#### NOTE



#### Text size is not changed

Changing the `ascender` and `descender` properties only affects the position, not the size of the text. The text size can be changed through the `font` widget property. For more information, see [section 8.5.1, “Changing the font size of a Label”](#).

Prerequisite:

- The EB GUIDE model contains a view state.
- The view contains a label.

#### Step 1

In the **Properties** component, go to the **Widget feature properties** and click **Add/Remove**.

The **Widget feature** dialog is displayed.

#### Step 2

Under **Available widget features**, expand the **Common** category, and select **Font metrics**.

#### Step 3

Click **Accept**.

The related widget feature properties are added to the font and displayed in the **Properties** component.

#### Step 4

In the **Properties** component, in the **Font metrics** category, change the values of the `ascender` and `descender` property.

The position of the text changes accordingly.



#### Example 8.3.



### Changing the descender property

The following example showcases how the position of the text within a label changes when the `descender` property is changed.

A label is added with the default widget property `horizontalAlign` set to `leading (0)` and the `verticalAlign` set to `center (0)`.

The text is positioned in the label as follows:

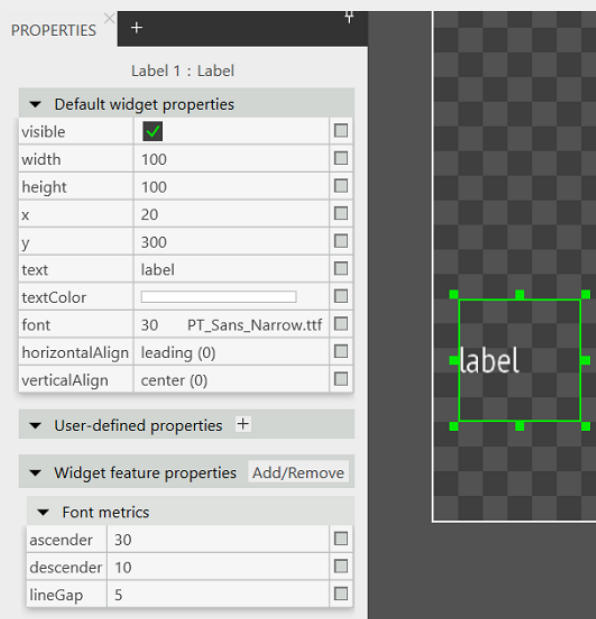
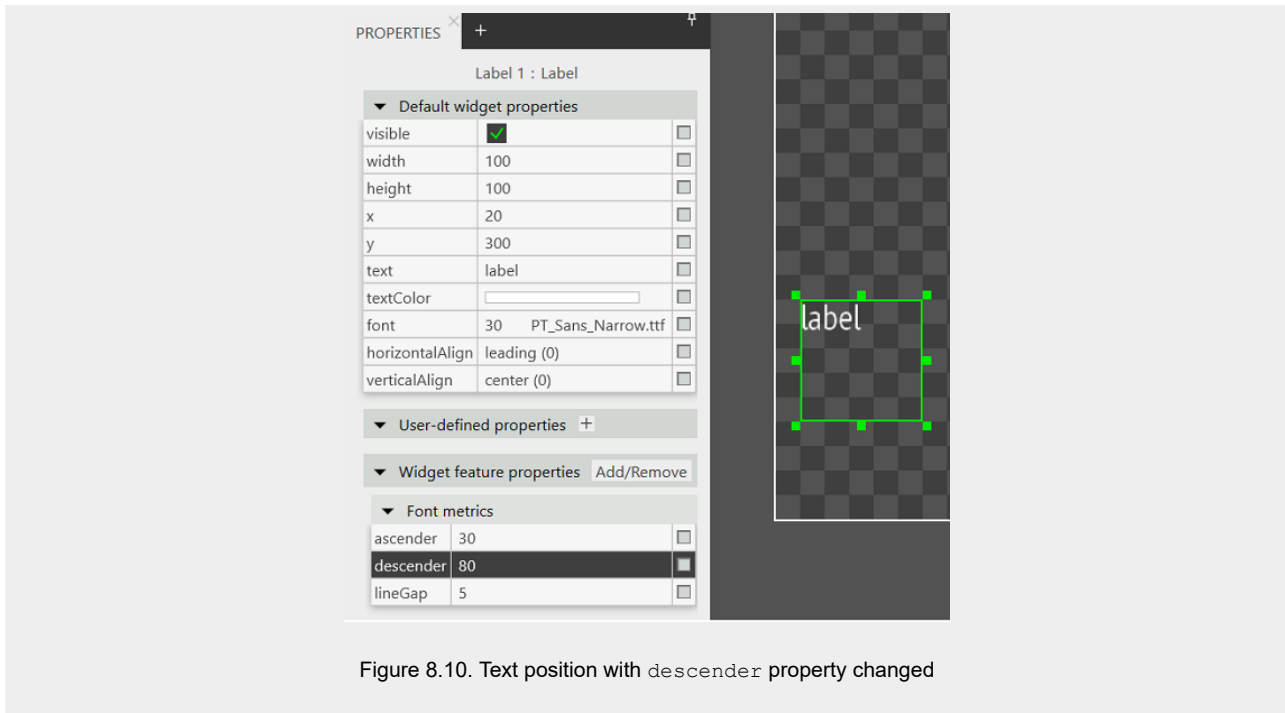


Figure 8.9. Text position with default values

Now the `descender` property is set to 80.

The position of the text changes as follows:



## 8.5.5. Managing multifont support

For more information, see [section 6.20.1.2, “Multifont support”](#).



### Adding multifont support for type font

Prerequisite:

- More than one font is available in `$GUIDE_PROJECT_PATH/resources`.
- A property or a datapool item of type font exists. This property is not a scripted value.

#### Step 1

To add multifont support, do the following:

- ▶ If you want to add multifont support to a widget property, go to the **Properties** component.
- ▶ If you want to add multifont support to a datapool item, go to the **Datapool** component.

#### Step 2

Next to the property or datapool item, click the  button.

A menu expands.

#### Step 3

In the menu, click **Add multifont support**.

A table is displayed below the property or the datapool item. The table has one default multifont value.

#### Step 4

To add a new multifont value, click the **+** button.

A new row is added and is filled with values based on the default font.

#### Step 5

In the row you can edit and define the following:

- ▶ In the column **Prio**, define the evaluation priority of the font entry.
- ▶ In the column **Font**, define the font size and the font.
- ▶ In the column **Range**, define the Unicode characters that are affected by the selected font.

#### NOTE



#### Performance

For better performance, instead of defining a font for a single Unicode character use character ranges or combination of character ranges. For example use 0000, 0001 or 0000-FFFF.

#### Step 6

Repeat the steps 2 to 5 until all required font values are added.

#### TIP



#### Removing the multifont support

To remove the multifont support, click the  button again. In the menu that opens, click **Remove multifont support**.



#### Adding multifont support for type font list

For instructions on how to work with properties of type list, see [section 8.3.8, "Editing a property of type list"](#).

Prerequisite:

- More than one font is available in `$GUIDE_PROJECT_PATH/resources`.
- A property or a datapool item of type font list exists and has at least one font list entry.


#### Step 1

To add multifont support, do the following:

- ▶ If you want to add multifont support to a widget property, go to the **Properties** component.


- ▶ If you want to add multifont support to a datapool item, go to the **Datapool** component.

#### Step 2

Select the property or datapool item, and next to it, click the  button.

The **Edit** dialog opens.

#### Step 3

Next to the entry of type font, click the  button.

A menu expands.

#### Step 4

In the menu, click **Add multifont support**.

A table is added below the property or the datapool item. The table has one default multifont value.

#### Step 5

To add a new multifont value, click the  button.

A new row is added and filled with values based on the default font.

#### Step 6

In the row you can edit and define the following:

- ▶ In the column **Prio**, define the evaluation priority of the font entry.
- ▶ In the column **Font**, define the font size and the font.
- ▶ In the column **Range**, define the Unicode characters that are affected by the selected font.

#### NOTE



#### Performance

For better performance, instead of defining a font for a single Unicode character use character ranges or combination of character ranges. For example use 0000, 0001 or 0000-FFFF.

#### Step 7

Repeat the steps 3 to 6 until all required font values are added.

#### TIP



#### Removing the multifont support

To remove the multifont support, click the  button again. In the menu that opens, click **Remove multifont support**.

#### Step 8

When you are finished editing entries in the list, click the **Accept** button.

The dialog closes.

## 8.5.6. Placing text along a circular path

With the **Circular text** widget feature you can place text along a circular path. This gives you more opportunities for the design of your EB GUIDE model.

If you want your text to follow a path of any arbitrary shaped curve, you can develop your own widget feature with the EB GUIDE SDK.

### NOTE



### Limitations

If you have added the **Circular text** widget feature to a label, the following limitations apply:

- ▶ Clipping of the text on widget boundaries is disabled when widget features **Circular text** and **Stroke** are both enabled. Use widget feature **Viewport** as a workaround.
- ▶ Only Latin script fonts are supported.
- ▶ **Circular text** cannot be used in combination with `verticalAlign`.
- ▶ **Circular text** cannot be used in combination with **Multiple lines**.
- ▶ **Circular text** cannot be used in combination with **Font metrics**.



### Adding **Circular text** to a label

Prerequisite:

- The content area displays a View.
- The View contains a Label.

#### Step 1

Select the Label.

The **Properties** component displays the properties of the selected Label.

#### Step 2

Enter your text in the `Text` text box.

#### Step 3

In the **Widget feature** properties category, click **Add/Remove**.

The **Widget features** dialog is displayed.

##### Step 3.1

Under **Available widget features**, expand the **Effect** category, and select the **Circular text** widget feature.

##### Step 3.2

Click **Accept**.

The text is now placed along a circle path.

Step 4

Adjust the **Widget feature properties**. For information on each property, see [section 15.11.2.3, “Circular text”](#).

## 8.6. Working with language support

To change the language of your EB GUIDE model during run-time, you add language support and language-dependent texts.

For more information on languages in an EB GUIDE model, see [section 6.15.2, “Languages in the EB GUIDE model”](#)

### 8.6.1. Adding a language to the EB GUIDE model

**NOTE**



**No skin support available**

When you have defined a language support for a datapool item, it is not possible to add a skin support to the same item.



#### Adding a language

The following instruction shows you how to add a language to your EB GUIDE model.

Step 1

Click .


The project center opens.

Step 2

In the navigation area, click **Configure > Languages**.

A default set and available languages are displayed. The first language in the list is always the default language.

Step 3

In the content area, click .

A language is added to the table. The language uses the standard language settings as initial values.

Step 4

Enter a name for the language.

#### Step 5

Select a language from the **Language** drop-down list box.

#### Step 6

Select a country from the **Country** drop-down list box.

The language is added to the EB GUIDE model. In the project editor, in the **Language** drop-down list of the command area, you can select the new language.

You can switch between the languages during run-time, to see the effect of the different datapool values. For more information, see [section 14.6, “Tutorial: Adding a language-dependent text to a datapool item”](#).

## 8.6.2. Adding language support to a datapool item



### Adding language support to a datapool item

The following instruction shows you how to add a language support to a datapool item of your EB GUIDE model.


Prerequisite:

- The EB GUIDE model contains datapool items.
- At minimum two languages are added to the model.

#### Step 1

In the project editor go to the **Datapool** component.


#### Step 2

Next to the **Value** property of a datapool item, click the  button.

A menu expands.

#### Step 3

In the menu, click **Add language support**.

The dialog closes. Next to the **Value** property, the  button is displayed. It indicates that a language support is added to this datapool item and now different values for each language can be defined.

You added language support to a datapool item. Now you can define language-dependent values for this datapool item. For more information, see [section 14.6, “Tutorial: Adding a language-dependent text to a datapool item”](#).

## 8.6.3. Deleting a language

#### NOTE



#### Default language

The first language in the list is always the default language and cannot be deleted.



#### Deleting a language

Prerequisite:

- At minimum two languages are added to the EB GUIDE model.

##### Step 1

Click .

The project center opens.

##### Step 2

In the navigation area, click **Configure > Languages**.

The available languages are displayed.

##### Step 3

In the content area, right-click a language.

A menu opens.

##### Step 4

Click **Delete**.

The language is deleted from the table.

## 8.6.4. Creating an export set for languages

An export set defines which languages are exported. This is useful in cases where you do not want to export all of the languages in an EB GUIDE model. If you do not define an export set, all of the languages in the EB GUIDE model are exported.



#### Creating an export set for languages

Prerequisite:

- An EB GUIDE project is opened.



#### Step 1


Click .

The project center opens.

#### Step 2

In the navigation area, click **Configure > Languages**.

#### Step 3

In the top-left corner, click .

An empty export set is created.

#### Step 4

Select the default set.

A list of available languages is displayed.

#### Step 5

Drag the languages that you want to export into the export set that you created.

The number of languages in the export set is displayed.

## 8.7. Working with skin support

With skin support you can define different datapool values for your model. This way you can define different looks for the same model, as for example night and day mode.

For more information on skins, see [section 6.23, "Skins"](#).

### 8.7.1. Adding a skin to the EB GUIDE model

#### NOTE



#### No language support available

When you have defined a skin support for a datapool item, it is not possible to add a language support to the same item.



#### Adding a skin to the EB GUIDE model

#### Step 1

Click .

The project center opens.

#### Step 2

In the navigation area, click **Configure > Skins**.

A default set and a standard skin are displayed.

#### Step 3

In the content area, click **+**.

A skin is added to the table.

#### Step 4

Enter a name for the skin.

The new skin is added to the EB GUIDE model. In the project editor, in the **Skin** drop-down list box of the command area you can select the new skin.

## 8.7.2. Adding skin support to a datapool item



### Adding skin support to a datapool item

To define different datapool values and thus define various looks for your EB GUIDE model, you first need to add a skin support to the datapool item.


Prerequisite:

- The EB GUIDE model contains datapool items.
- A skin is added to the model.

#### Step 1

In the project editor, go to the **Datapool** component.


#### Step 2

Next to the **Value** property of a datapool item, click the  button.

A menu expands.

#### Step 3

In the menu, click **Add skin support**.

The dialog closes. Next to the **Value** property, the button  is displayed. It indicates that a skin support is added to this datapool item and now different values for each skin can be defined.

#### Step 4

To define different values for the datapool item, select the datapool in the **Datapool** component.

The **Properties** component displays a table with all skins available in the EB GUIDE model.

#### Step 5

Define a value for each skin in the table.

### 8.7.3. Switching between skins



#### Switching between skins

Prerequisite:

- The EB GUIDE model contains datapool items.
- A skin is added to the model.

#### Step 1

In the project editor go to the command area.

#### Step 2

Select a skin in the drop-down list box.

The content area displays the model with the datapool values valid for this skin. Also the simulation mode will display the model with the specific skin values.

### 8.7.4. Deleting a skin



#### Deleting a skin

Prerequisite:

- A skin is added to the model.

#### Step 1

Click .

The project center opens.

#### Step 2

In the navigation area, click **Configure > Skins**.

All skins of the current project are listed.

#### Step 3

In the content area, right-click a skin.

A menu opens.

Step 4

Click **Delete**.

The skin is deleted from the table.

## 8.7.5. Creating an export set for skins

An export set defines which skins are exported. This is useful in cases where you do not want to export all of the skins in an EB GUIDE model. If you do not define an export set, all of the skins in the EB GUIDE model are exported.



### Creating an export set for skins

Prerequisite:

- An EB GUIDE project is opened.

Step 1


Click .

The project center opens.

Step 2

In the navigation area, click **Configure > Skins**.

Step 3

In the top-left corner, click .

An empty export set is created.

Step 4

Select the default set.

A list of available skins is displayed.

Step 5

Drag the skins that you want to export into the export set that you created.

The number of skins in the export set is displayed.

## 8.8. Working with templates

For more information on templates, see [section 6.26.4, “Widget templates”](#).

## 8.8.1. Adding a template



### Adding a template

#### Step 1

In the **Templates** component, click **+**.

A menu expands.

#### Step 2

In the menu, select a type for the template.

#### TIP



#### Templates of templates

A type for the template can be an existing template. EB GUIDE thus allows creating templates from templates.

A new template of the selected type is added. The content area displays the template.

#### Step 3

Rename the template.

#### Step 4

In the **Properties** component, edit the template's properties, and define the template interface.

#### TIP



#### Copying and finding templates

Alternatively, you can copy and paste an existing template using the context menu or **Ctrl+C** and **Ctrl+V**.

To find a specific template within your EB GUIDE model, enter the name of the template in the search box or press **Ctrl+F**. To jump to a template, double-click it in the hit list.

## 8.8.2. Creating a template from widget tree



### Creating a template from widget tree

It is possible to create a widget template from the widget tree.

Prerequisite:

- At least one widget is added.

#### Step 1

In the **Navigation** component, right-click the widget.

A menu expands.

#### Step 2

In the menu, select **Create template**.

A new template is created. The **Templates** component displays the template. If the parent widget had child widgets, properties, and widget features, they are also included in the created template.

#### Step 3

Rename the template.

#### Step 4

In the **Properties** component, edit the template's properties, and define the template interface.

### 8.8.3. Defining the template interface

#### NOTE

#### Restrictions for scripted values



It is not possible to add scripted values to model or template interfaces.



#### Defining the template interface

Prerequisite:

- The EB GUIDE model contains a template.

#### Step 1


Select a template.

#### Step 2

To add a property to the template interface, in the **Properties** component, click the  button next to the property. In the menu, click **Add to template interface**.

The  icon is displayed next to the property.

#### Step 3

To remove a property from the template interface, click the  button next to the property. In the menu, click **Remove from template interface**.

The  icon is no longer displayed next to the property.

---

**NOTE**



**Instantiator templates**

For templates of instantiators, it is not possible to add properties of the instantiator's child widgets to the template interface.

---

## 8.8.4. Using a template



### Using a template

Prerequisite:

- The content area displays a view.
- In the **Toolbox**, a widget template is available.
- There is at least one property in the template interface of the widget template.

#### Step 1

Drag a widget template from the **Toolbox** into the view.

An instance of the template is added to the view. The **Properties** component displays the properties which belong to the template interface.

---

**TIP**



**Define the template interface**

If the **Properties** component does not display any properties for a template instance, no properties have been added to the template interface. Define the template interface to change that.


---

#### Step 2

In the **Properties** component, edit the properties of the template instance.

After editing a property, the  button changes to the  button.

#### Step 3

To reset a property value to the value of the template, click the  button next to the property. In the menu, click **Reset to template value**.

## 8.8.5. Deleting a template



#### Deleting a template

##### Step 1

In the **Templates** component, right-click a template.

##### Step 2

In the context menu, click **Delete**.

The template is deleted.

## 8.9. Enabling anti-aliasing

For background information, see [section 6.3, “Anti-aliasing”](#).

### 8.9.1. Enabling anti-aliasing globally



#### Enabling anti-aliasing globally

Prerequisite:

- There is an EB GUIDE model.

##### Step 1

In the **Project center** select **Configure > Profiles**.

The **Profiles** menu opens.

##### Step 2

In the **Scenes** tab, from the `antiAliasing` drop-down list box, select the anti-aliasing mode that you want to set for the scene.

The anti-aliasing mode is now configured for the whole EB GUIDE model.



## 8.9.2. Enabling anti-aliasing for scene graphs



### Enabling anti-aliasing globally

Prerequisite:

- The EB GUIDE model contains a Scene graph.

#### Step 1

Select a Scene graph.

#### Step 2

In the **Properties** component, go to the **Widget feature properties** category, and click **Add/Remove**.

The **Widget features** dialog is displayed.

#### Step 3

Expand the **3D** category, select the **Anti-aliasing mode** widget feature, and click **Accept**.

In the **Properties** component the **Anti-aliasing mode** widget feature is displayed.

#### Step 4

From the `antiAliasing` drop-down list box, select the mode that you want to have for this Scene graph.

The anti-aliasing mode for this Scene graph is configured.

## 9. Handling data

### TIP



#### Default window layout

All instructions and screenshots use the default window layout. If you want to follow the instructions, we recommend to set the EB GUIDE Studio or EB GUIDE Monitor window to default layout by selecting **Layout > Reset to default layout**.

## 9.1. Working with namespaces

For more information on namespaces, see [section 6.6, “Components of the graphical user interface”](#) and [section 6.17, “Namespaces”](#).

### 9.1.1. Adding a namespace



#### Adding a namespace

You can add a new namespace as a child to the root namespace, or to any existing namespace.

#### Step 1

As the **Namespaces** component is not available in the default window layout of EB GUIDE Studio, do the following:

##### Step 1.1

In the command area click **Layout** and select **Namespaces**.

The **Namespaces** component is displayed.

##### Step 1.2

In the **Namespaces** component, click **+**.

A namespace is added to the tree.

#### Step 2

Rename the namespace

---

**TIP**



**Moving a namespace**

To move a namespace, drag it to the root namespace or to another namespace. Take care of the naming conventions for namespaces to avoid naming conflicts.

---

## 9.1.2. Adding model elements to a namespace

---

**NOTE**



**Imported namespace**

You cannot add or move a model element to a namespace that is part of an imported model interface.

---

For instructions on how to add an event to a namespace, see [section 9.2.1, “Adding an event”](#).

For instructions on how to add a datapool item to a namespace, see [section 9.3.1, “Adding a datapool item”](#).

## 9.1.3. Moving model elements between namespaces

---

**NOTE**



**Imported namespace**

You cannot add or move a model element to a namespace that is part of an imported model interface.

---



### Moving model elements between namespaces

Prerequisite:

- An event or a datapool item is added to a namespace.
- At least two namespaces exist.

Step 1

To move an event, go to the **Events** component.

To move a datapool item, go to the **Datapool** component.

Step 2

Select the corresponding namespace and right-click the model element.

#### Step 3

From the context menu, select **Move to namespace....**

A dialog opens.

#### Step 4

Select the target namespace and click **Accept**.

The model element is moved to the target namespace.

#### TIP



#### Moving model elements

Alternatively, you can drag the model element to another namespace.

### 9.1.4. Deleting a namespace

#### WARNING



#### Deleting a namespace

When you delete a namespace, you also delete all model elements that this namespace contains.

#### NOTE



#### Root namespace

You cannot delete the root namespace.



#### Deleting a namespace

Prerequisite:

- A namespace is added to the EB GUIDE model.

#### Step 1

In the **Namespace** component, right-click the namespace.

#### Step 2

In the context menu, click **Delete**.

The namespace is deleted.

## 9.2. Working with events

### TIP



#### Default window layout

All instructions and screenshots use the default window layout. If you want to follow the instructions, we recommend to set the EB GUIDE Studio or EB GUIDE Monitor window to default layout by selecting **Layout > Reset to default layout**.

For more information, see [section 6.10, “Event handling”](#).

### 9.2.1. Adding an event

### NOTE



#### Imported namespace

You cannot add or move a model element to a namespace that is part of an imported model interface.



#### Adding an event

Root namespace and user-defined namespaces are visible in the **Events** component only if they have at least one event added to them.

#### Step 1

Go to the **Events** component.

#### Step 2

Click **+**.

An event is added to the table.

- ▶ If you have not selected a namespace, the event will be added to the first visible namespace that is not read-only (not imported). In case that only imported namespaces are visible, the event will be added to the root namespace.
- ▶ In case you have selected several namespaces, the event is added to the first selected namespace if it is not read-only (not imported). If this namespace is read-only (imported), the event will be added to the root namespace.
- ▶ If filters were set and a newly added event does not match the filter settings, filters are reset.

#### Step 3

Rename the event.

#### NOTE



#### Naming of model elements

For the names of the model elements use the following characters: a-z, A-Z, 0-9, space, and underscore.

For more information, see [section 6.8.3, "Naming of model elements"](#).

#### Step 4

To change the event ID, go to the **Properties** component, and in the `Event ID` text box enter an ID.

#### TIP



#### Copying and finding events

Alternatively, you can copy and paste an existing event using the context menu or **Ctrl+C** and **Ctrl+V**. To prevent duplicates, the pasted event has a different event ID than the copied event.

To find a specific event within your EB GUIDE model, enter the name of the event in the search box or use **Ctrl+F**. To jump to an event, double-click it in the hit list.

## 9.2.2. Adding a parameter to an event



### Adding a parameter to an event

Prerequisite:

- An event is added to the EB GUIDE model.

#### Step 1

In the **Events** component, click an event.

#### Step 2

In the events table click **+** next to the event.

#### Step 3

From the drop-down list box select a type for the parameter.

A parameter of the selected type is added to the event.

#### Step 4

Rename the parameter.

### 9.2.3. Addressing an event

Event IDs and event group IDs are used to address events. EB GUIDE TF uses the IDs to send and receive the events at run-time.



#### Adding an event group

##### Step 1

Click .

The project center opens.

##### Step 2

In the navigation area, click **Configure** > **Event groups**.

##### Step 3

In the content area, click **Add**.

An event group is added to the table.

##### Step 4

Rename the event group.

##### Step 5

To change an event group ID, double-click the **ID**, and type a number.



#### Addressing an event for EB GUIDE TF

Prerequisite:

- An event group is added.
- An event is added to the EB GUIDE model.

##### Step 1

In the **Events** component, click an event.

The **Properties** component displays the properties of the selected event.

##### Step 2

Insert an ID in the `Event ID` text box.

##### Step 3

Go to the **Events** component and select an event group from the `Group` drop-down list box.

### 9.2.4. Mapping a key to an event

To fire an event, you can press a key on your keyboard, or, for example, you can press a key or use a rotary button on your target device.

In order to react on keyboard key presses, the model needs to define a mapping for these key events.

For each key, EB GUIDE GTF defines a numeric code in the C++ header file. For code numbers, see `$INSTALL_PATH$/platform/win64/include/gtf/displayfactory/inputmapper/KeyConstants.h`.



### Mapping a key to an event

For more information on event handling, see [section 6.10, “Event handling”](#) and [section 15.5, “Events”](#).

Prerequisite:

- The event group **Key** (ID 10) is added in **Project Center > Configure > Event groups**.
- An event is added.

#### Step 1

Look up the hex code number of the key you want to map in the file `KeyConstant.h`.

#### Step 2

Calculate the decimal code number.

#### Step 3

In the **Events** component, in the **Group** column, select **Key** (ID 10).

#### Step 4

In the **Properties** component, into the `Event ID` text box enter the calculated decimal code number.

The selected key is now mapped to an event.



### Example 9.1. Mapping the key F1 to an event

The internal hex code number of **F1** is 12.

The decimal code number of 12 is 18.

Go to the **Properties** component and enter 18 into the `Event ID` text box.

The key **F1** is now mapped to the event.



**NOTE**



**Invisible event in EB GUIDE Monitor**

The event itself is not shown in EB GUIDE Monitor, but an EB GUIDE Script, which was triggered by the event, reacts.

For more information on decimal code numbers, see [section 15.5.1, “Decimal codes for key events”](#).

## 9.2.5. Adding events to a model interface



### Adding events to a model interface

For more information on model interfaces, see [section 6.18, “Model interfaces”](#).

For instructions on how to work with model interfaces, see [section 10.9, “Working with model interfaces”](#).

Prerequisite:

- The EB GUIDE model contains an event.

Step 1

Go to the **Events** component.

Step 2

If there is only one default model interface, right-click the event and select **Add to model interface**.

If there are multiple model interfaces, right-click the event, select **Model interfaces**, and then select the model interface where this event is supposed to be included.

The event is now contained in the model interface. A colored bar on the left side of the event indicates that it was added to the model interface.

## 9.2.6. Deleting an event



### Deleting an event

Prerequisite:

- An event is added to the EB GUIDE model.

#### Step 1

In the **Events** component, select the corresponding namespace.

#### Step 2

Right-click the event and from the context menu select **Delete**.

The event is deleted.

## 9.3. Working with datapool items

### TIP



### Default window layout

All instructions and screenshots use the default window layout. If you want to follow the instructions, we recommend to set the EB GUIDE Studio or EB GUIDE Monitor window to default layout by selecting **Layout > Reset to default layout**.

For more information, see [section 6.7, "Datapool"](#).

### 9.3.1. Adding a datapool item

### NOTE



### Imported namespace

You cannot add or move a model element to a namespace that is part of an imported model interface.



### Adding a datapool item

Root namespace and user-defined namespaces are visible in the **Datapool** component only if they have at least one datapool item added to them.

#### Step 1

Go to the **Datapool** component.

#### Step 2

Click **+**.

A menu expands.

#### Step 3

In the menu, click a type for the datapool item.

A new datapool item of the selected type is added. The datapool item is ready for internal use.

- ▶ If you have not selected a namespace, the datapool item will be added to the first visible namespace that is not read-only (not imported). In case that only imported namespaces are visible, the datapool item will be added to the root namespace.
- ▶ In case you have selected several namespaces, the datapool item is added to the first selected namespace if it is not read-only (not imported). If this namespace is read-only (imported), the datapool item will be added to the root namespace.
- ▶ If filters were set and a newly added datapool item does not match the filter settings, filters are reset.

#### Step 4

Rename the datapool item.

#### NOTE



#### Naming of model elements

For the names of the model elements use the following characters: a–z, A–Z, 0–9, space, and underscore.

For more information, see [section 6.8.3, “Naming of model elements”](#).

#### TIP



#### Copying and finding datapool items

Alternatively, you can copy and paste an existing datapool item using the context menu or **Ctrl+C** and **Ctrl+V**.

To find a specific datapool item within your EB GUIDE model, enter the name of the datapool item in the search box or use **Ctrl+F**. To jump to a datapool item, double-click it in the hit list.

## 9.3.2. Editing datapool items of type list



### Editing datapool items of type list

Prerequisite:

- A datapool item of type list is added.

#### Step 1

In the **Datapool** component, click a datapool item of a list type.

#### Step 2

Select the **Value** column and click .

An editor opens.

#### Step 3

To add an item to the list datapool item, click **Add**.

A new entry is added to the table.

#### Step 4

Enter a value for the new entry in the `Value` text box or select a value from the combo box.

#### Step 5

Repeat steps three and four to add more items to the list.

#### Step 6

Click **Accept**.

The content of the list is displayed in the `Value` column.

### 9.3.3. Converting a property to a scripted value



#### Converting a property to a scripted value

Properties of datapool items and widgets can be converted to a scripted value and back to their plain value. The following instruction shows the procedure with a datapool item value. With a widget property, the procedure is the same.

#### NOTE




#### Restrictions for scripted values

It is not possible to add scripted values to model or template interfaces.

Prerequisite:

- A datapool item is added.
- The datapool item is not language-dependent.
- The datapool item is not skin-dependent.
- The datapool item is not linked.
- The datapool item does not have multifont support.

#### Step 1

In the **Datapool** component, click a datapool item and click the  button.


A menu expands.

#### Step 2

In the menu, click **Convert to script**.

The datapool item is converted to a scripted value.

#### Step 3


Next to the datapool item, select the **Value** column and click .

An EB GUIDE Script editor opens.

#### Step 4

Edit the EB GUIDE Script.

#### Step 5

To convert the datapool item back to its plain value, click the  button.

A menu expands.

#### Step 6

In the menu, click **Convert to plain value**.

The datapool item is converted to its plain value.

## 9.3.4. Establishing external communication

To establish external communication for example between the EB GUIDE model and an application, you add communication contexts to the EB GUIDE model.



### Adding a communication context

With communication contexts you are able to channel communication.

#### Step 1

Click .

The project center opens.

#### Step 2

In the navigation area, click **Configure > Communication contexts**.

#### Step 3

In the content area, click **Add**.

A communication context is added to the table.

#### Step 4

Rename the communication context, for example to `Media`.

#### Step 5

To run the communication context in an own thread, select **Use own thread**.

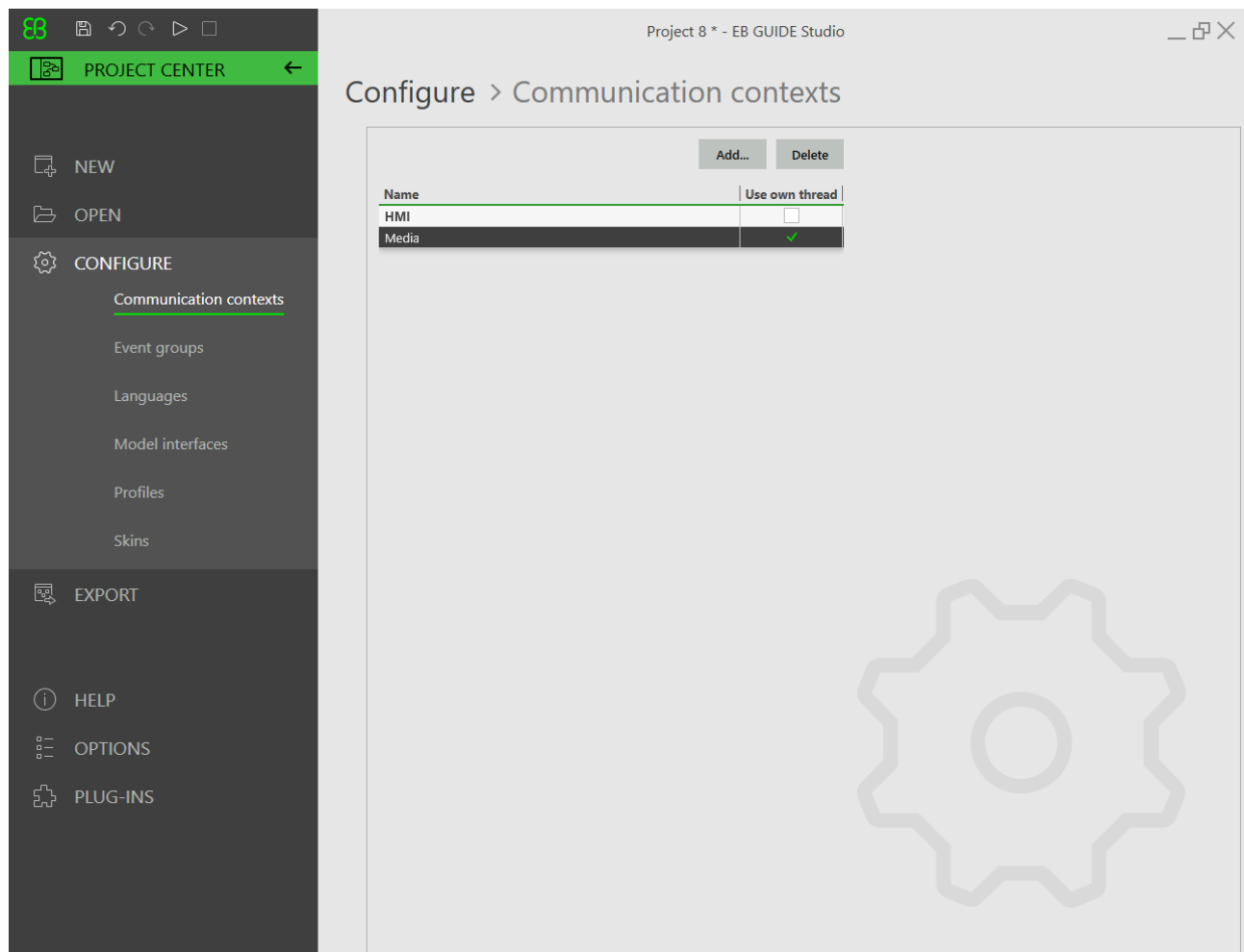


Figure 9.1. Communication context *Media*.

### 9.3.5. Linking between datapool items



#### Linking between datapool items

Prerequisite:

- A datapool item is added.
- The datapool item is not language-dependent.
- The datapool item is not skin-dependent.
- The datapool item is not a scripted value.

Step 1

In the **Datapool** component, click a datapool item.

Step 2

Click the  button.

A menu expands.

Step 3

In the menu, click **Add link to datapool item**.

A dialog opens.

Step 4

To add a new datapool item, enter a name in the text box.

Step 5

Click **Add datapool item**.

Step 6

Click **Accept**.

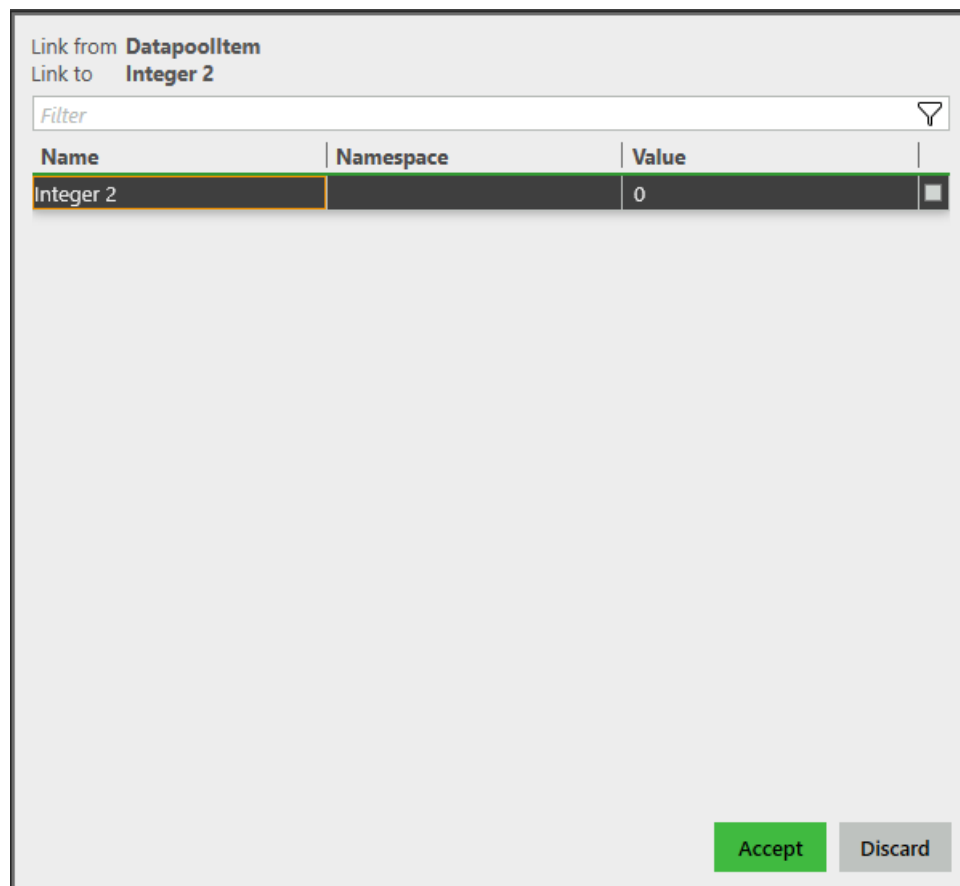



Figure 9.2. Linking between datapool items

The dialog closes. Next to the `Value` property, the  button is displayed. It indicates that the `Value` property is linked to a datapool item. Whenever one of the datapool items changes its value, the value of the other datapool item changes as well.

### 9.3.6. Adding datapool items to a model interface

#### NOTE

#### Restrictions for scripted values



It is not possible to add scripted values to model or template interfaces.



#### Adding datapool items to a model interface

For more information on model interfaces, see [section 6.18, “Model interfaces”](#).

For instructions on how to work with model interfaces, see [section 10.9, “Working with model interfaces”](#).

Prerequisite:

- The EB GUIDE model contains a datapool item.

#### Step 1

Go to the **Datapool** component.

#### Step 2

If there is only one default model interface, right-click the datapool item and select **Add to model interface**.

If there are multiple model interfaces, right-click the item, select **Model interfaces**, and then select the model interface where this item is supposed to be included.

The datapool item is now contained in the model interface. A colored bar on the left side of the datapool item indicates that it was added to the model interface.

### 9.3.7. Deleting a datapool item



#### Deleting a datapool item

Prerequisite:

- A datapool item is added.





Step 1

In the **Datapool** component, select the corresponding namespace.

Step 2

Right-click the datapool item and from the context menu select **Delete**.

The datapool item is deleted.

## 10. Handling a project

### TIP



### Default window layout

All instructions and screenshots use the default window layout. If you want to follow the instructions, we recommend to set the EB GUIDE Studio or EB GUIDE Monitor window to default layout by selecting **Layout > Reset to default layout**.

## 10.1. Creating a project



### Creating a project

#### Step 1

Click .

The project center opens.

#### Step 2

In the navigation area, click **New**.

#### Step 3

Enter a project name, and select a location.

#### Step 4

Click **Create**.

The project is created. The project editor opens and displays the new project.

## 10.2. Opening a project

#### NOTE



#### Invalid project

If the EB GUIDE project is not valid, EB GUIDE Studio cannot open it. An error message is shown and a log file with the error description is created in `$GUIDE_PROJECT_PATH/<project name>/<project name>_LoadingErrorLog.txt`.

For more information, see [section 6.8.4, “Validation criteria for EB GUIDE project”](#).

## 10.2.1. Opening a project from the file explorer



### Opening a project from the file explorer

Prerequisite:

- An EB GUIDE project is created.

#### Step 1

Open the file explorer, and select the EB GUIDE project file you would like to open. EB GUIDE project files have the file extension `.ebguide`.

#### Step 2

Double-click the EB GUIDE project file.

The project opens in EB GUIDE Studio.

## 10.2.2. Opening a project within EB GUIDE Studio



### Opening a project within EB GUIDE Studio

Prerequisite:

- An EB GUIDE project is created.

#### Step 1

Click .

The project center opens.

#### Step 2

In the navigation area, click the **Open** tab.

#### Step 3

Select a project that is listed under **Recent projects**, or click **Browse** and select the EB GUIDE project file that you would like to open. EB GUIDE project files have the file extension `.ebguide`.

The project opens in EB GUIDE Studio.

## 10.3. Renaming model elements



### Renaming model elements

The following instruction guides you through the process of renaming a model element.

Prerequisite:

- A model element is added to the EB GUIDE model.

#### Step 1

To rename a model element, right-click the element.

The context menu opens.

#### Step 2

In the context menu, select either of the following:

- ▶ To rename only the selected model element, select **Rename**.
- ▶ To rename the selected model element, and also its occurrences in the EB GUIDE model, for example in EB GUIDE Script, select **Rename global**.

#### Step 3

Rename the element and confirm with **Enter** or click outside the text box.

The element name is changed.

## 10.4. Validating and simulating an EB GUIDE model

Before exporting an EB GUIDE model to the target device, you resolve errors and simulate the model on your PC.



## 10.4.1. Validating an EB GUIDE model

### 10.4.1.1. Validating an EB GUIDE model in EB GUIDE Studio



#### Validating an EB GUIDE model in EB GUIDE Studio

In the **Problems** component, EB GUIDE displays the following:


- ▶  errors
- ▶  warnings

For more information, see [section 6.8.4, “Validation criteria for EB GUIDE project”](#).

#### Step 1

To expand the **Problems** component, click **Problems**.

#### Step 2

In the **Problems** component, click .

A list of errors and warnings is displayed.

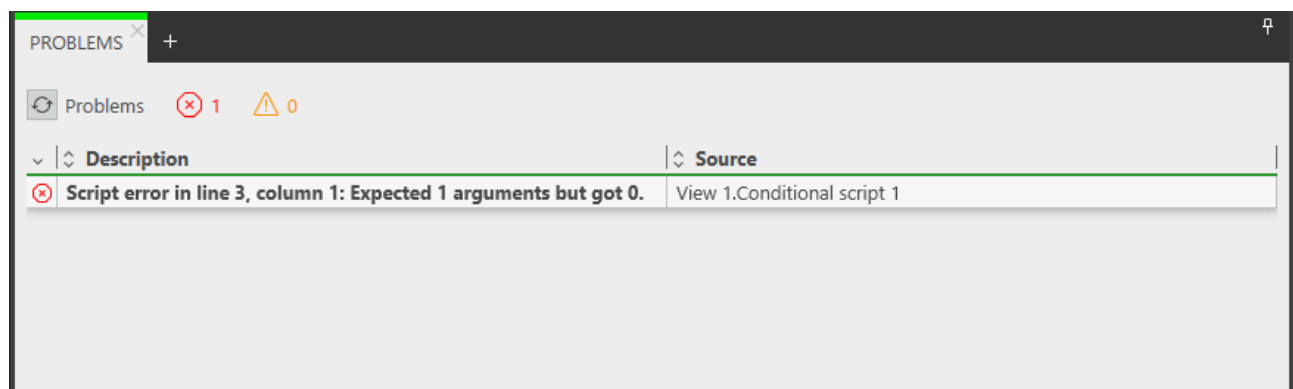


Figure 10.1. Problems component

#### Step 3

To navigate to the source of a problem, double-click the corresponding line.

The element that causes the problem is highlighted.

#### Step 4

Solve the problem.

#### Step 5

Click .

The problem you solved is no longer listed in the **Problems** component.

If there are no errors, the EB GUIDE model is valid. The EB GUIDE model is also valid if there are some warnings.

### 10.4.1.2. Validating an EB GUIDE model using command line



#### Validating an EB GUIDE model using command line

##### Step 1

With command line navigate to `$GUIDE_INSTALL_PATH/Studio`.

##### Step 2

Enter `Studio.Console.exe -c "<logfile dir>/log.txt" -o "$GUIDE_PROJECT_PATH/project_name.ebguide"`.

The EB GUIDE model is validated and the result is saved to a logfile at the specified location `<logfile dir>`.

### 10.4.2. Starting and stopping the simulation



#### Starting and stopping the simulation

##### Step 1

To start the simulation, click  in the command area.

The simulation and EB GUIDE Monitor start. The simulation starts with its own configuration.

To change the configuration, go to the project center, and click **Configure > Profiles**.

##### Step 2

To stop the simulation, click  in the command area.

The simulation and EB GUIDE Monitor stop.

## 10.5. Exporting an EB GUIDE model

## 10.5.1. Exporting an EB GUIDE model using EB GUIDE Studio



### Exporting an EB GUIDE model using EB GUIDE Studio

To copy the EB GUIDE model to the target device, you need to export it using EB GUIDE Studio.

For every export of an EB GUIDE model you select a profile.

Prerequisite:

- There is an export set for languages. For instructions, see [section 8.6.4, “Creating an export set for languages”](#).
- There is an export set for skins. For instructions, see [section 8.7.5, “Creating an export set for skins”](#).
- If there are no export sets, all of the languages and skins in the EB GUIDE model are exported.

#### Step 1

Click .

The project center opens.

#### Step 2

In the navigation area, click the **Export** tab.

#### Step 3

From the `Profile` drop-down list box select a profile.

#### Step 4

From the `Skin set` drop-down list box select an export set for skins.

If you do not select an export set, all of the skins in the EB GUIDE model are exported.

#### Step 5

From the `Language set` drop-down list box select an export set for languages.

If you do not select an export set, all of the languages in the EB GUIDE model are exported.

#### Step 6

Click **Browse**, and select a location where to export the binary files.

#### Step 7

Click **Select folder**.

#### Step 8

Click **Export**.

The binary files are exported to the selected location.

## 10.5.2. Exporting an EB GUIDE model using command line



### Exporting an EB GUIDE model using command line

Prerequisite:

- The EB GUIDE model is free of errors and warnings.

#### Step 1

In the command line, go to `$GUIDE_INSTALL_PATH/Studio`.

#### Step 2

Enter `Studio.Console.exe -e <destination dir> -p <profile> -s <skin_set> -t <language_set> -o "$GUIDE_PROJECT_PATH/project_name.ebguide"`.

The EB GUIDE model is exported to the selected location with the specified profile, skin set, and language set.

## 10.6. Changing the display language of EB GUIDE Studio



### Changing the display language of EB GUIDE Studio

#### Step 1

Click .

The project center opens.

#### Step 2

In the navigation area, click the **Options** tab.

#### Step 3

Select a language from the **Display language** drop-down list box.

#### Step 4

Restart EB GUIDE Studio.

After restarting the graphical user interface is displayed in the selected language.

## 10.7. Configuring profiles



EB GUIDE Studio offers the possibility to create different profiles for an EB GUIDE model.

You use profiles to do the following:

- ▶ Send messages
- ▶ Configure internal and user-defined libraries to load
- ▶ Configure a scene
- ▶ Configure a renderer

There are two default profiles: **Edit** and **Simulation**.

## 10.7.1. Adding a profile



### Adding a profile

To add a profile in EB GUIDE Studio, clone an existing profile.

Prerequisite:

- An EB GUIDE project is opened.
- The project center is displayed.

#### Step 1

In the navigation area, click **Configure > Profiles**.

#### Step 2

In the content area, select the **Simulation** profile.

#### Step 3

Click **Clone**.

A profile is added to the table. The profile is a clone of the default profile **Simulation**.

#### Step 4

Double-click in the table and rename the profile to `MySimulation`.

#### Step 5

Select **Use for simulation**.

The `MySimulation` profile is used for simulation on the PC.

## 10.7.2. Adding a library

The default delivery of EB GUIDE TF runs on operating systems that support shared libraries, for example Windows 10, Linux or QNX. EB GUIDE TF is divided into executable file and a set of libraries to fit most customer projects out of the box.

The following tasks show you how to add a user-defined library that interacts with the EB GUIDE model and provides additional functionality.



#### Adding a library: Platform

This task shows you how to add a library or several libraries that can be used by all EB GUIDE models on the current platform.

Prerequisite:

- An EB GUIDE project is opened.
- The project center is displayed.
- In the navigation area, the tab **Configure > Profiles** is selected.
- A profile `MySimulation` is added.
- Libraries `MyLibraryA` and `MyLibraryB` are available in `$GTF_INSTALL_PATH/platform/<platform name>/bin`.

#### Step 1

In the content area, select the `MySimulation` profile.

#### Step 2

Click the **Platform** tab.

#### Step 3

Enter the following code:

```
{
  "gtf":
  {
    "core":
    {
      "pluginstoload": ["MyLibraryA", "MyLibraryB"]
    }
  }
}
```

You added libraries `MyLibraryA` and `MyLibraryB` to the start-up code.

#### NOTE



#### JSON object notation

If you configure `platform.json` within EB GUIDE Studio, use the JSON object notation.

For an example, see the reference section of the EB GUIDE GTF user guide.

For more information about JSON format, see <http://www.json.org>.



#### Adding a library: Model

This task shows you how to add a library or several libraries that can be used only by the current EB GUIDE model.

Prerequisite:

- An EB GUIDE project is opened.
- The project center is displayed.
- In the navigation area, the tab **Configure > Profiles** is selected.
- A profile `MySimulation` is added.
- Libraries `MyLibraryA` and `MyLibraryB` are available in `$GUIDE_PROJECT_PATH/<project name>/resources`.

#### Step 1

In the content area, select the `MySimulation` profile.

#### Step 2

Click the **Model** tab.

#### Step 3

Enter the following code:

```
{
  "gtf":
  {
    "model":
    {
      "pluginstoload": ["resources/MyLibraryA", "resources/MyLibraryB"]
    }
  }
}
```

You added libraries `MyLibraryA` and `MyLibraryB` to the start-up code.

#### NOTE



#### JSON object notation

If you configure `model.json` in EB GUIDE Studio, use the JSON object notation.

For an example, see the reference section of the EB GUIDE GTF user guide.

For more information about JSON format, see <http://www.json.org>.

## 10.7.3. Configuring a scene

In EB GUIDE Studio it is possible to configure a scene for every state machine.

Projects can have more than one state machine for one of the following reasons:

- ▶ To separate the logic of the model into different state machines
- ▶ To use more than one display or layer



### Configuring a scene

Prerequisite:

- An EB GUIDE project is opened.
- The project center is displayed.
- In the navigation area, the tab **Configure** > **Profiles** is selected.

#### Step 1

In the content area, click the **Scenes** tab.

#### Step 2

From the **State machine** drop-down list box select the state machine of your main display, for example **Main**.

#### Step 3

To set the initial position of the window on the PC desktop, enter a value for *x* and *y*.

#### Step 4

Select a renderer from the **Renderer** drop-down list box.

#### Step 5

Adjust further properties. For information on each property, see [section 15.7, "Scenes"](#).

## 10.8. Exporting and importing language-dependent texts

EB GUIDE allows you to display text in the user's preferred language. To make this possible, you add language support to your EB GUIDE model. You can then export the language-dependent texts to an `.xliff` file, have them translated and import them back into your model.

### NOTE



#### Project and language specific IDs

A unique alphanumeric `sourcelanguageid` and `targetlanguageid` is created for every project and language pair. These IDs prevent unintentional import of an `.xliff` file from another project or target language. Also each datapool item receives a unique alphanumeric ID for each language.

When the languages or datapool items are changed in the EB GUIDE Studio model while the texts are exported for translation, the translated texts can still be assigned to the right datapool item and language due to the specific ID. Refer to the import logfile, to see, if all datapool items are assigned correctly.

### 10.8.1. Exporting language-dependent texts

#### TIP



#### Validating the EB GUIDE model

To avoid errors during export and import of texts, validate your EB GUIDE model before you start.

For more information, see [section 10.4.1.1, "Validating an EB GUIDE model in EB GUIDE Studio"](#).



#### Exporting language-dependent texts

To provide text in the user's preferred language, you export all language-dependent texts of datapool items and pass them on to your localization service provider.

Prerequisite:

- The languages to be translated are added to the EB GUIDE model. For more information, see [section 8.6.1, "Adding a language to the EB GUIDE model"](#).
- A datapool item of type `String` or `String list` is added.

- The datapool item has language support. For information, see [section 8.6.2, “Adding language support to a datapool item”](#).
- The EB GUIDE model is free of errors and warnings.

#### Step 1

Click .

The project center opens.

#### Step 2

In the navigation area, click **Configure > Languages**.

#### Step 3

In the content area, select the target language to be translated.

Multi-selection is possible.

#### Step 4

Click **Export**.

A dialog opens.

#### Step 5

Select a folder to export the files.

#### Step 6

Click **Select folder**.

Result: The export starts. The files are saved in the selected folder. The file has a language-dependent acronym and the format `.xliiff`. The file contains values for the source language and values for the target language.

#### NOTE

#### Structure and content of the exported file



- ▶ For each language you select in the project center, a separate `.xliiff` file is exported.
- ▶ The source language is the default language. Therefore, if a text was not translated yet, the `target-language` element always contains the source text.

For more information on language-dependent texts in an EB GUIDE model and `.xliiff` files, see [section 6.15, “Languages”](#).

## 10.8.2. Importing language-dependent texts

### 10.8.2.1. Importing language-dependent texts using EB GUIDE Studio



#### Importing language-dependent texts using EB GUIDE Studio

Prerequisite:

- At minimum one translated `.xliff` file for the selected EB GUIDE model is available.
- The datapool item to be translated and the target language still exists.
- The EB GUIDE model is free of errors and warnings.

#### Step 1

Click .

The project center opens.

#### Step 2

In the navigation area, click **Configure > Languages**.

#### Step 3

Click **Import**.

A dialog opens.

#### Step 4

Select the folder where the translated `.xliff` file is stored.

#### Step 5

Select the translated `.xliff` file.

Multi-selection is possible.

#### Step 6

Click **Open**.

The import starts. A dialog opens.

#### Step 7

Click **Close**.

Now all datapool items with language support show the corresponding language dependent text. See the logfile for more information on the import.

### 10.8.2.2. Importing language-dependent texts using command line



### Importing language-dependent texts using command line

Prerequisite:

- At minimum one translated `.xliff` file for the selected EB GUIDE model is available.
- The datapool item that was sent to translation still exists.
- The EB GUIDE model is free of errors and warnings.

#### Step 1

With command line navigate to `$GUIDE_INSTALL_PATH/Studio`.

#### Step 2

Enter `Studio.Console.exe -l <language file> -o "$GUIDE_PROJECT_PATH/project_name.ebguide"`.

If the import was successful, the EB GUIDE model is changed. If the import was not successful, the EB GUIDE model is not changed. In both cases a logfile is generated. A date and a time stamp are added to the name of the logfile.

## 10.9. Working with model interfaces

The instructions in the following sections guide you through the process of adding datapool items and events to the model interface, and exporting and importing the interface. For background information, see [section 6.18, "Model interfaces"](#).

### 10.9.1. Adding a model interface



### Adding a model interface

Prerequisite:

- An EB GUIDE project is created.

#### Step 1

Click .

The project center opens.

#### Step 2

In the navigation area, click **Configure > Model interfaces**.



All model interfaces of the current project are listed.

Step 3

Click **+**.

A user-defined model interface is added with a default name.

Step 4

Rename the model interface.

Step 5

There is a colored bar next to the model interface. This bar is also next to events or datapool items. It indicates which model interface they belong to. To change the color of this bar, right-click the interface and select **Select color**.

Now all model elements that belong to this model interface have the same color bar in the **Datapool** and **Events** components.

## 10.9.2. Exporting a model interface

### NOTE



#### Validation of the model interface

Before exporting a model interface, it is advisable to validate your EB GUIDE model. In case that there are errors in the model interface, the export fails and a validation error is shown.

At least one of the following model elements can cause validation errors during the model interface export:

- ▶ Event that belongs to an internal event group
- ▶ Datapool item with scripted value
- ▶ Linked datapool item

Make sure that these model elements are not part of the model interface to be exported.



#### Exporting a model interface

Prerequisite:

- An EB GUIDE project is created.
- A model interface is added to the EB GUIDE project.
- Events or datapool items are added to the model interface. For instructions on how to add model elements to a model interface, see [section 9.2.5, “Adding events to a model interface”](#) and [section 9.3.6, “Adding datapool items to a model interface”](#).

#### Step 1

Click .

The project center opens.

#### Step 2

In the navigation area, click **Configure > Model interfaces**.

All model interfaces of the current project are listed.

#### Step 3

Select the model interface to be exported.

#### Step 4

Click .

An explorer window opens.

#### Step 5

Save the `.json` file with the interface information in a location of your choice.

You exported the model interface. This `.json` file can now be used to import the model interface into another EB GUIDE model.

## 10.9.3. Importing a model interface



### Importing a model interface

Prerequisite:

- Two EB GUIDE projects are created.
- A model interface is exported from the first project and a `.json` file is created with the interface information.
- The second project is opened in EB GUIDE Studio.

#### Step 1

Click .


The project center opens.

#### Step 2

In the navigation area, click **Configure > Model interfaces**.

The **Model interfaces** menu opens.

#### Step 3

To import the model interface, click .

An explorer window opens.

#### Step 4

Navigate to the location of the `.json`, select the file, and click **Open**.

The interface is imported into the project. In the **Model interfaces** menu you can see the interfaces and you can see how many datapool items and events each interface provides.

#### NOTE



#### Duplicate event IDs

Event IDs in an event group must be unique. When you import several model interfaces, validation errors occur in case there are duplicate IDs of events that belong to different model interfaces but are within the same event group.

Since you cannot change the event IDs of the imported events in EB GUIDE Studio, revert the import, change the event IDs in the source model, export and import again. It is advisable to define event ID ranges for all EB GUIDE models beforehand.

## 10.9.4. Updating an imported model interface

To update an imported model interface, make the changes in the source EB GUIDE model, then export the updated model interface, and re-import it again.

For more information on how to export and import model interfaces, see [section 10.9.2, “Exporting a model interface”](#) and [section 10.9.3, “Importing a model interface”](#).

## 10.9.5. Deleting a model interface

#### NOTE



#### Default model interface

You cannot delete the default model interface.



### Deleting a model interface

Prerequisite:

- An EB GUIDE model has a user-defined or an imported model interface.

#### Step 1

Click .



The project center opens.

Step 2

In the navigation area, click **Configure > Model interfaces**.

All model interfaces of the current project are listed.

Step 3

Right-click the model interface that you want to delete and in the context menu click **Delete**.

If you delete a model interface that is listed under **Imported**, then the model interface with all included model elements is deleted from the EB GUIDE model.

If you delete a model interface that is listed under **User-defined**, then only the model interface is deleted from the EB GUIDE model. All model elements that were added to this model interface still exist.

# 11. Working with EB GUIDE Monitor

For more information on EB GUIDE Monitor, see [section 6.9, “EB GUIDE Monitor”](#) and [section 6.6.2, “Graphical user interface of EB GUIDE Monitor”](#).

## TIP



### Default window layout

All instructions and screenshots use the default window layout. If you want to follow the instructions, we recommend to set the EB GUIDE Studio or EB GUIDE Monitor window to default layout by selecting **Layout > Reset to default layout**.

## 11.1. Starting EB GUIDE Monitor as a stand-alone application

EB GUIDE Monitor starts automatically in EB GUIDE Studio during the simulation of an EB GUIDE model. But you can also start EB GUIDE Monitor as a stand-alone application.



### Starting EB GUIDE Monitor

Prerequisite:

- EB GUIDE is installed.
- An EB GUIDE model is exported to `$EXPORT_PATH`.

#### Step 1

In the file explorer, navigate to `$GUIDE_INSTALL_PATH/tools/monitor`.

#### Step 2

Double-click `Monitor.exe`

EB GUIDE Monitor starts.



### Starting EB GUIDE Monitor using command line

Prerequisite:

- EB GUIDE is installed.
- An EB GUIDE model is exported to `$EXPORT_PATH`.

#### Step 1

In the file explorer, navigate to `$GUIDE_INSTALL_PATH/tools/monitor`.

#### Step 2

Open command line and enter `Monitor.exe`

EB GUIDE Monitor starts.

## 11.2. Configuring EB GUIDE Monitor




### Connecting EB GUIDE Monitor

Prerequisite:

- EB GUIDE Monitor is started.
- An EB GUIDE model is running.

#### Step 1


To change the connection settings to EB GUIDE GTF, click .

#### Step 2

In **Host**, enter the host name, and in **Port**, enter the port address.

#### Step 3

Click .

EB GUIDE Monitor is connected and the status button is green: .

#### TIP

#### Disconnecting EB GUIDE Monitor



To disconnect EB GUIDE Monitor, click .

EB GUIDE Monitor is now disconnected and the status button is red: .



### Changing the display language of EB GUIDE Monitor

Prerequisite:

- EB GUIDE Monitor is started as a stand-alone application.

#### Step 1

From **File > Display language** select a language.

#### Step 2

Restart EB GUIDE Monitor.

After restarting, the graphical user interface is displayed in the selected language.

#### NOTE



#### EB GUIDE Monitor inherits the language from EB GUIDE Studio

If EB GUIDE Monitor is started in EB GUIDE Studio, you cannot change the display language of the graphical user interface. EB GUIDE Monitor has the same display language as EB GUIDE Studio.



#### Resetting the size of EB GUIDE Monitor window

The size of the EB GUIDE Monitor window and also its position on the screen are stored for each individual EB GUIDE project separately in `C:\<user>\AppData\Local\Temp\eb_guide_simulation_export\<project>`.

Prerequisite:

- EB GUIDE Monitor is started and its original position and window size were manipulated.

#### Step 1

To reset the size and the position to the default values, delete `monitor_layout.xml` and `monitor_model_config.json` in `C:\<user>\AppData\Local\Temp\eb_guide_simulation_export\<project>`.

#### Step 2

Restart the simulation, or, if EB GUIDE Monitor was started as a stand-alone application, restart EB GUIDE Monitor.

The new `monitor_layout.xml` and `monitor_settings.xml` files are created with the default size and position values.



#### Editing the number of log messages

In the `$INSTALL_PATH\tools\monitor\Monitor.exe.config` configuration file, you can define the behavior of EB GUIDE Monitor when the logger reaches its memory limit.

#### Step 1

Open the `$INSTALL_PATH\tools\monitor\Monitor.exe.config` configuration file.

#### Step 2

To define the number of entries that are displayed in the **Logger** component, change the `limit` value.

#### Step 3

To define the number of entries that are going to be deleted when the limit is reached, change the `remove-Count` value.

#### Step 4

Start EB GUIDE Monitor.

EB GUIDE Monitor uses the new settings from the changed configuration file.

## 11.3. Loading configurations into EB GUIDE Monitor



### Loading a configuration file into EB GUIDE Monitor

Prerequisite:

- EB GUIDE Monitor is started as a stand-alone application.
- An EB GUIDE model is exported to `$EXPORT_PATH`.
- In `$EXPORT_PATH`, the `monitor.cfg` configuration file is created.

#### Step 1

Select **File** > **Load configuration**.

A dialog opens.

#### Step 2

Navigate to `$EXPORT_PATH` and select the `monitor.cfg` configuration file.

#### Step 3

Click **Open**.

The configuration of your project is loaded into EB GUIDE Monitor.



### Loading a recent configuration file into EB GUIDE Monitor

Prerequisite:

- EB GUIDE Monitor is started as a stand-alone application.
- One or more configuration files have been used recently.



#### Step 1

Select **File > Recent configuration files**.

A dialog opens.

#### Step 2

Navigate to the respective location and select a configuration file.

The configuration file is loaded into EB GUIDE Monitor.

#### NOTE



#### EB GUIDE Monitor disconnects from EB GUIDE GTF

Before a new configuration file is loaded, EB GUIDE Monitor is automatically disconnected from the current EB GUIDE GTF.

EB GUIDE Monitor reconnects and loads the new configuration file.

## 11.4. Firing an event in EB GUIDE Monitor



### Firing an event in EB GUIDE Monitor

Prerequisite:

- The EB GUIDE model contains an event.
- The simulation of the EB GUIDE model is started.
- The EB GUIDE Monitor is started.

#### Step 1

In EB GUIDE Monitor, in the **Events** component, click **+**.

A dialog opens.

#### Step 2

Select the event to be fired and click **Accept**.

The event is added to the list.

#### Step 3

To fire an event, click  in the **Events** component next to the event.

The event is fired. In the **Logger** component a log message appears.

#### Step 4

If the event has parameters, do the following:

Step 4.1

Click  to expand parameters.

Step 4.2

Change parameters in the **Value** column.

Step 4.3

To fire an event, click  next to the event.

The event is fired with changed parameters. In the **Logger** component a log message appears.

## 11.5. Changing the value of the datapool item with EB GUIDE Monitor



### Changing the value of the datapool item in EB GUIDE Monitor

Prerequisite:

- The EB GUIDE model contains a datapool item.
- The simulation of the EB GUIDE model is started.
- The EB GUIDE Monitor is started.

Step 1

In EB GUIDE Monitor, in the **Datapool** component, click .

A dialog opens.

Step 2

Select the datapool item and click **Accept**.

The datapool item is added to the list.

Step 3

Change the value of the datapool item in the **Value** column.

#### NOTE



#### Supported types

You can change datapool items of the following data types:

- ▶ Boolean
- ▶ Color
- ▶ Integer
- ▶ Float
- ▶ String

The value of the datapool item is changed. In the **Logger** component a log message appears.

## 11.6. Using scripts in EB GUIDE Monitor

### 11.6.1. Writing script files for EB GUIDE Monitor

For more information on script methods, see the EB GUIDE Monitor API in `$GUIDE_INSTALL_PATH/doc/monitor/monitor_api.chm`.

The following is an example for basic EB GUIDE Monitor script functions.

#### NOTE



#### Using methods for states and state machines

If your EB GUIDE model has several states or state machines with identical names, use `uint` IDs. Find `uint` IDs that are relevant for your project in `$EXPORT_PATH/monitor.cfg`.



#### Example 11.1. Example script file for EB GUIDE Monitor

The following is an example script `MonitorExampleScript.cs`.

```
namespace MyProject
{
    using System.Threading.Tasks;

    using System.Windows.Media; // necessary for type color!

    using Elektrobit.Guide.Monitor.Scripting.MonitorContext;

    public class Basic
    {
```

```
public async Task PrintMessage(IMonitorContext monitor) //❶
{
    await monitor.Write("Hello World");
}

public async Task FireEvent(IMonitorContext monitor) //❷
{
    await monitor.FireEvent("nextView");
}

public class Events
{
    public async Task FireEventWithParameter(IMonitorContext monitor)
    {
        await monitor.FireEvent("setBool", true);
    }

    public async Task WaitForEvent(IMonitorContext monitor) //❸
    {
        var ev = await monitor.WaitForEvent("nextView");
        await monitor.Write("Even occurred: " + ev.EventModel.Name);
    }

    public async Task WaitForEventWithParameters(IMonitorContext monitor)
    {
        var ev = await monitor.WaitForEvent("setBool");

        bool mv1 = ev["value"]; // read parameter via name
        bool mv2 = ev[0]; // read the parameter via index

        await monitor.Write("Parameter 'value' is: " + mv1);
        await monitor.Write("Parameter [0] is: " + mv2);
    }
}

public class Datapool
{
    public async Task WriteDpValue(IMonitorContext monitor) //❹
    {
        await monitor.WriteDatapool("Boolean 1", true);
    }

    public async Task ReadDatapoolValue(IMonitorContext monitor) //❺
    {
        bool boolValue = await monitor.ReadDatapool("Boolean 1");
    }
}
```

```
        string stringValue = await monitor.ReadDatapool("String 1");
        int integerValue = await monitor.ReadDatapool("Integer 1");
        float floatValue = await monitor.ReadDatapool("Float 1");

        await monitor.Write("Boolean: " + boolValue);
        await monitor.Write("String: " + stringValue);
        await monitor.Write("Integer: " + integerValue);
        await monitor.Write("Float: " + floatValue);
    }

    public async Task ReadColor(IMonitorContext monitor)
    {
        Color colorValue = await monitor.ReadDatapool("Color 1");
        await monitor.Write("Boolean: " + colorValue);
    }
}

public class StateMachines
{
    public async Task WaitForStateChanges(IMonitorContext monitor)
    {
        var leftState = await monitor.WaitForStateExit
            ("Main", "State 1"); //❸
        await monitor.Write(string.Format("State {0} left",
            leftState.Name));

        var enteredState = await monitor.WaitForStateEnter
            ("Main", "State 2"); //❹
        await monitor.Write(string.Format("State {0} entered",
            enteredState.Name));
    }

    public async Task WaitForStateMachineChanges(IMonitorContext monitor)
    {
        var startedStateMachine = await monitor.WaitForStateMachineStart
            ("Dynamic state machine 1"); //❺
        await monitor.Write(string.Format("State Machine {0} started",
            startedStateMachine.Name));

        var stoppedStateMachine = await monitor.WaitForStateMachineStop
            ("Dynamic state machine 1"); //❻
        await monitor.Write(string.Format("State Machine {0} stopped",
            stoppedStateMachine.Name));
    }
}
```

```
public class Advanced
{
    public async Task CaptureScreenshot(IMonitorContext monitor) //10
    {
        // make sure remote framebuffer is enabled in profile
        uint sceneId = 0;
        await monitor.CaptureScreenshot(sceneId, @"d:/image.png");
    }

    public async Task CountTo10(IMonitorContext monitor)
    {
        for (var i = 0; i < 10; i++)
        {
            await monitor.Write("Hello World: " + i);
            await Task.Delay(1000, monitor.CancellationToken);

            monitor.CancellationToken.ThrowIfCancellationRequested();
        }
    }

    public async Task WaitForEventWithTimeout(IMonitorContext monitor) //11
    {
        // Disclaimer:
        // this is just one of many opportunities provided by
        // the .NET's "Task Parallel Library"

        var eventWaitTask = monitor.WaitForEvent("nextView");

        await Task.WhenAny(eventWaitTask, Task.Delay(5000));

        if (!eventWaitTask.IsCompleted || eventWaitTask.IsFaulted)
        {
            return;
        }

        await monitor.Write("event occurred");
    }
}

namespace MonitorScripting.EventScripts //12
{
    using Elektrobit.Guide.Monitor.Scripting.MonitorContext;
    using System.Threading;
    using System.Threading.Tasks;
    using Elektrobit.Guide.Monitor.Model.Event;
    using Elektrobit.Guide.Monitor.Model.Value;
```

```
public class MonitorScripts
{
    public async Task FireEventInNamespace(IMonitorContext monitor)
    {
        string[] namespacePath = { "Foo_namespace", "go_to_view2" };
        var identifier = new QualifiedIdentifier(namespacePath);

        await monitor.FireEvent(identifier);
    }

    public async Task FireEventInNestedNamespace(IMonitorContext monitor)
    {
        string[] namespacePath =
            { "Foo_namespace", "sub_namespace_under_foo", "go_to_view4" };
        var identifier = new QualifiedIdentifier(namespacePath);

        await monitor.FireEvent(identifier);
    }

    public async Task FireEventInRootNamespace(IMonitorContext monitor)
    {
        await monitor.FireEvent("go_to_view3");
    }
}
}
```

- ❶ Method to print out a message
- ❷ Method to fire an event
- ❸ Method to wait for an event
- ❹ Method to write a datapool value
- ❺ Method to read a datapool value
- ❻ Method to wait until the state is entered and then to report it
- ❼ Method to wait until the state is exited and then to report it
- ❽ Method to wait until the state machine is started and then to report it
- ❾ Method to wait until the state machine is stopped and then to report it
- ❿ Method to capture a screenshot
- 11 Method to wait for an event with timeout
- 12 Example how to work with namespaces

## 11.6.2. Starting scripts in EB GUIDE Monitor



### Starting scripts in EB GUIDE Monitor

Prerequisite:

- The simulation of the EB GUIDE model is started.
- The EB GUIDE Monitor is started.
- A `.cs` or a `.dll` file with a script is available on your computer. For script examples, see [section 11.6.1, “Writing script files for EB GUIDE Monitor”](#).

#### Step 1

To open the **Scripting** component, select **Layout > Scripting**.

The **Scripting** component opens as a docked component.

#### Step 2

In the **Scripting** component click the **Open** button.

The file explorer opens.

#### Step 3

Select a `.cs` or a `.dll` file and click **Open**.

All applicable methods and the corresponding classes, which were included in the file, are listed in the **Script** table.

#### Step 4

Select a method and click the start button.

The script is started. In the **Script output** area a log message appears.

## 11.7. Exporting and importing watch lists

The events and datapool items you use for a project are stored in watch lists. In order to use the items for other projects you can export the watch lists as an `.xml` file and later import them into your new projects.



### Exporting all watch lists

Prerequisite:

- EB GUIDE Monitor is started.
- An EB GUIDE model is already set up with items stored in **Datapool** or **Events** tabs.



Step 1

To export all watch lists, select **File > Export all watch lists**.

A dialog opens.

Step 2

Select a destination folder and enter a file name.

All datapool items and events are exported.



### Exporting a single watch list

Prerequisite:

- EB GUIDE Monitor is started.
- An EB GUIDE model is already set up with items stored in **Datapool** or **Events** components.

Step 1

Go to the **Datapool** or **Events** component that contains the items you want to export.

Step 2

To save the list of items of this tab, click **Export**.

A dialog opens.

Select a destination folder and enter a file name.

The datapool items or events of the component are exported.



### Importing watch lists

Prerequisite:

- EB GUIDE Monitor is started.
- An exported watch list is already available.

Step 1

To import a watch list, select **File > Import watch lists**.

A dialog opens.

Step 2

Select a watch list file to import.

The datapool items or events are opened in new **Datapool** or **Events** components.

---

**NOTE**



**Layout is not imported**

Only the datapool items and events are imported, not the layout.

The default layout is used for the newly opened **Datapool** and **Events** components.

---

## 12. Extending EB GUIDE Studio

This chapter provides information about the concepts that you need to understand to be able to create extensions for EB GUIDE Studio as well as instructions and examples. If you encounter problems while implementing an extension, contact our support. See [chapter 3, “Support”](#).

### 12.1. Concepts

#### 12.1.1. Dependency injection

EB GUIDE Studio is built with dependency injection in mind. In order to manage and resolve dependencies EB GUIDE Studio uses the Managed Extensibility Framework (MEF), which is part of the .NET Framework.

Dependencies are registered and injected based on the Attributed Programming Model. Extensions can provide new functionality by exporting an implementation of an interface and can use existing EB GUIDE Studio functionality by importing interfaces.

To import dependencies into your class, add the `ImportingConstructor` attribute to the constructor and add the required dependencies as constructor arguments. When an instance of your class is requested the MEF will try to satisfy all the dependencies and call the marked constructor.

Consider that the MEF only satisfies the dependencies of a class that is exported itself. To export a class, add the `Export` attribute to the class.



#### **Example 12.1.** **Example for the `Export` attribute**

The following example shows the usage of the `ImportingConstructor` and `Export` attributes in a generic way. An implementation of the `IFooService` interface is exported which in turn imports a dependency on the `IBarService` interface.

```
[Export(typeof(IFooService))]  
internal class MyFooService : IFooService  
{  
    [ImportingConstructor]  
    public MyFooService(IBarService barService) {}  
}
```

For a more detailed overview of the Managed Extensibility Framework, see <https://docs.microsoft.com/en-us/dotnet/framework/mef/>. For more details on the Attributed Programming Model, see <https://docs.microsoft.com/en-us/dotnet/framework/mef/attributed-programming-model-overview-mef>.

## 12.1.2. EB GUIDE model extensions

To ensure the consistency of the model, all modifications must be performed in sequential order. This is achieved by scheduling all modifications on a task scheduler that executes one action after another. In addition to that all modifications to the model must be performed in sessions. A session has two purposes:

- ▶ Grouping the modifications into a single changeset. As a result, everything performed in one session can be undone in one step.
- ▶ Telling the underlying storage which elements have changed. This means that only changes that are performed within a session are actually stored to the file system.

### WARNING

#### Data loss



If you do not use a session to modify an EB GUIDE model, you can damage your model and cause data loss.

EB GUIDE Studio provides an API to simplify the handling of task scheduling and sessions.

- ▶ `ITaskSchedulerProvider` provides access to the task scheduler that must execute all model modifications.
- ▶ The `IEventService` interface provides methods to create and modify events in the model.
- ▶ `ExecuteModelAction` is an extension method in the `ITaskSchedulerProvider` interface. It schedules the modification on the correct task scheduler and creates a session. The second argument to this extension method is a delegate that performs the actual model modification. The session is created before the delegate is invoked and automatically committed after the delegate has been executed. This means as a user you only call `ExecuteModelAction` and specify how you want to modify the model without taking care of task scheduling and session handling yourself.



### Example 12.2. Applying extensions to the EB GUIDE model

The following example shows, how to change the previous example to apply the extension to the EB GUIDE model. The previous example was changed so that it now imports the `ITaskSchedulerProvider` and `IEventService` dependencies.

```
[Export(typeof(IFooService))]  
internal class MyFooService : IFooService  
{
```

```
private readonly ITaskSchedulerProvider _schedulerProvider;
private readonly IEventService _eventService

[ImportingConstructor]
public MyFooService(
    ITaskSchedulerProvider schedulerProvider,
    IEventService eventService)
{
    _schedulerProvider = schedulerProvider;
    _eventService = eventService;
}

public async Task ModifyModel(IProjectContext projectContext)
{
    await _schedulerProvider.ExecuteModelAction(
        projectContext,
        session => _eventService.CreateEvent(
            session,
            projectContext,
            projectContext.Project.RootNamespace,
            "My Event"));
}
}
```

As mentioned above, this example also imports the `IEventService` interface as a dependency. It is highly recommended that you use the existing service interfaces to modify the model instead of making direct modifications to the model elements.

In case there is no existing service that fits your needs there are some rules to follow in order to ensure model consistency. A session is represented by the `IWriteSession` interface. You must make the correct calls on the session to store your changes in the model.

- ▶ When possible, build your model element trees before you add them to the model. This way you improve the performance by avoiding unnecessary model update notifications.
- ▶ An element that is newly created must be saved. If you created a whole element tree use the `SaveHierarchy` method. It automatically saves all child elements. As a rule of thumb, always call `SaveHierarchy` on newly created elements.
- ▶ Save modified elements with the `Save` method. Avoid using the `SaveHierarchy` method on existing elements because this can lead to a lot of unnecessary entries in the change set resulting in bad performance.
- ▶ Delete removed elements with the `Delete` method. Instead of calling this method directly, import the `IModelElementService` interface and use its `DeleteElements` method. This method recursively deletes the element and all its children.
- ▶ Do not call the `Commit` method explicitly. Calling `Commit` is handled by the `ExecuteModelAction` extension method mentioned above.

### 12.1.3. EB GUIDE Studio UI extensions

EB GUIDE Studio uses Windows Presentation Foundation (WPF) as its UI framework. For more information about WPF, see <https://docs.microsoft.com/en-us/dotnet/framework/wpf/>. In addition, the UI layer of EB GUIDE Studio is built with the Model-View-ViewModel (MVVM) pattern in mind. That means, for most cases you must provide a view model and a view implementation to extend the EB GUIDE Studio UI. For generic UI elements like menus there are existing views, but for custom UI elements a view implementation is required.

Custom view implementations are provided by exporting them using the MEF. A view model that is supposed to be used with WPF must follow certain conventions to work properly. A view model should implement the `INotifyPropertyChanged` interface. This interface is used by WPF's data binding engine to reflect changes in the view model to the view. If you do not implement this interface in your custom view model, the bindings will only transfer the initial values to the view layer and no updates are propagated. To simplify the creation of custom view models EB GUIDE Studio provides a base class called `ViewModel` implementing the `INotifyPropertyChanged` interface.



#### **Example 12.3.** **Custom view model implementation**

The following example shows a view model that propagates changes to its `Text` property to the view. It also has a command property that can be bound to the view. Each time the command is executed, e.g. by clicking the bound button, the `Text` property is changed, and the view is updated.

```
internal class MyViewModel : ViewModel
{
    private string _text;

    public string Text
    {
        get => _text;
        set => SetProperty(ref _text, value);
    }

    public ICommand DoSomethingCommand { get; }

    public MyViewModel()
    {
        Text = "Initial text";
        DoSomethingCommand = new DelegateCommand(DoSomething);
    }

    private void DoSomething()
    {
        Text = "Did something";
    }
}
```

```
}
```

A view is provided by creating a WPF `DataTemplate` for your view model. The `DataTemplate` is defined in XAML within a resource dictionary. You can provide custom resource dictionaries by exporting an implementation of the `IResourceProvider` interface.



#### **Example 12.4.** **DataTemplate for a custom view model**

The following code snippet shows how to provide a `DataTemplate` for a custom view model. The resource dictionary is defined in a file called `Resources.xaml` in the assembly `MyAssembly`. The resource provider implementation returns a URI to the XAML file.

```
[Export(typeof(IResourceProvider))]  
internal class MyResourceProvider : IResourceProvider  
{  
    public IEnumerable<Uri> GetResourceUris()  
    {  
        var uri = new Uri(  
            @"MyAssembly;Component/Resources.xaml",  
            UriKind.Relative);  
  
        return new[]{ uri };  
    }  
}
```



#### **Example 12.5.** **ResourceDictionary**

The following code snippet shows how to create a `ResourceDictionary` with a `DataTemplate` in XAML.

```
<ResourceDictionary  
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"  
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"  
    xmlns:mynamespace="clr-namespace:MyNamespace">  
    <DataTemplate  
        DataType="{x:Type mynamespace:MyViewModel}">  
        <mynamespace:MyView />  
    </DataTemplate>  
</ResourceDictionary>
```

URIs returned by `IResourceProvider` must follow the Pack URI syntax so that WPF can find and load them. For more information about the Pack URI syntax, see <https://docs.microsoft.com/en-us/dotnet/framework/wpf/app-development/pack-uris-in-wpf>.

Resource dictionaries provided by `IResourceProvider` implementations are loaded on application level. This means all resources in the dictionary are globally available. To avoid naming conflicts with existing resources with an explicit resource key you should prefix your resource keys with the name of your extension. For example, a custom button style for a special button could be named like this: `MyPlugin.MySpecialButton`.

Download the EB GUIDE extension examples from <https://github.com/Elektrobit/eb-guide-examples>. For instructions, see the `readme.md` files enclosed.

## 12.2. Creating an extension project



### Creating a Visual Studio project

Prerequisite:

- The EB GUIDE Studio installation folder is writeable. The default installation folder under `C:\Program Files\Elektrobit` is protected by Windows. For extension development use an installation directory that is not write protected.
- Visual Studio 2017 or later is installed.
- .NET Framework Developer Pack version 4.7.2 or later is installed.

#### Step 1

Open Visual Studio.

#### Step 2

Create a new project.

#### Step 3

In the **New Project** dialog select a project template:

For core extensions select `Class Library (.NET Framework)`.

For UI extensions select `WPF User Control Library (.NET Framework)`.

#### Step 4

Enter a name for your extension project and click **OK**.

The project is created.

#### Step 5

Right-click the project in the **Solution explorer** and select **Manage NuGet Packages...**



#### Step 6

Click **Browse** and search for `Elektrobit.Guide.Studio`.

#### Step 7

For core extensions install package `Elektrobit.Guide.Studio.Core`.

For UI extensions install package `Elektrobit.Guide.Studio.Ui`.

Your project is now set up and ready for coding.

## 12.3. Disabling copying of the assemblies



### Disabling copying of the assemblies

When building your project all assemblies from referenced NuGet packages are copied to your output folder by default. For the assemblies provided by the EB GUIDE NuGet packages this is not necessary because these assemblies are already part of the EB GUIDE installation that will be used to execute your extension. You can adapt the NuGet package reference to disable copying of the assemblies by following these steps.

Prerequisite:

- A plug-in project was created with Visual Studio.

#### Step 1

Open the project file (`.csproj`) of your extension project in a text editor.

#### Step 2

Find the `PackageReference` entry for the EB GUIDE NuGet package.

#### Step 3

Add the `IncludeAssets` property and set it to the value `compile`.

#### Step 4

Save the project file and reload the project in Visual Studio.

Now the `PackageReference` entry should look like this example snippet:

```
<PackageReference Include="Elektrobit.Guide.Studio.Ui">
  <Version>6.9.0</Version>
  <IncludeAssets>compile</IncludeAssets>
</PackageReference>
```

## 12.4. Running an extension



### Running an extension

#### Prerequisite:

- The EB GUIDE Studio installation folder is writeable. The default installation folder under `C:\Program Files\Elektrobit` is protected by Windows. For extension development use an installation directory that is not write protected.
- Your extension project is set as the start-up project of the solution.

#### Step 1

Open the project settings of your extension project and go to the **Debug** tab.

#### Step 2

Select `Start external program` and `Studio.exe` from your EB GUIDE Studio installation directory.

#### Step 3

Go to the **Build Events** tab and enter the following post-build script:

```
copy /Y $(TargetPath) <extension dir>
copy /Y $(TargetDir)$(TargetName).pdb <extension dir>
```

This script copies your extension into the EB GUIDE Studio plug-in folder after a successful build.

#### Step 4

Replace `<extension dir>` with the correct path of your EB GUIDE Studio installation.

For core extensions use `$GUIDE_INSTALL_PATH\studio\lib\core`.

For UI extensions use `$GUIDE_INSTALL_PATH\studio\lib\ui`.

Now you can run your extension from Visual Studio.

## 13. Best practices

The topics in this chapter are sorted alphabetically.

---

### TIP



#### Default window layout

All instructions and screenshots use the default window layout. If you want to follow the instructions, we recommend to set the EB GUIDE Studio or EB GUIDE Monitor window to default layout by selecting **Layout > Reset to default layout**.

---

### 13.1. Best practice: Handling scripted values

Properties of datapool items and widgets converted to a scripted value lead to EB GUIDE Script execution each time the property is read. In some use cases, to minimize the number of EB GUIDE Script executions and improve the performance, do the following:

1. If you have a scripted value, reset the property type: Use properties which are not converted to a scripted value but have their plain value. For more information, see [section 9.3.3, “Converting a property to a scripted value”](#) and [section 8.3.6, “Adding a user-defined property to a widget”](#).
2. To compute and set the current value, add a user-defined property of type `Conditional script`. Consider that this action has to be executed only if it is necessary, for example, on initialization or when an input property was changed.

## 14. Tutorials

### 14.1. Tutorial: Adding a dynamic state machine

#### TIP



#### Default window layout

All instructions and screenshots use the default window layout. If you want to follow the instructions, we recommend to set the EB GUIDE Studio or EB GUIDE Monitor window to default layout by selecting **Layout > Reset to default layout**.

Dynamic state machines allow pop-ups during run-time. You use dynamic state machines for example to display error messages that overlay the regular display.

The following instructions guide you through the process of creating a dynamic state machine. The instructions show you how to model a dynamic state machine for volume control. For best results, work through the following steps in the order presented.

Approximate duration: 20 minutes.



#### Adding events and datapool items

The following instructions guide you through the process of adding events and datapool items. These events are used to change the volume afterwards. The purpose of the datapool item is to change the position of a graphical element in a later section.

##### Step 1

Go to the **Events** component and click **+**.

An event is added to the table.

##### Step 2

Rename the event to `Volume up`.

##### Step 3

Add an event, and rename it to `Volume down`.

##### Step 4

Add an event, and rename it to `Close volume control`.

##### Step 5

Go to the **Datapool** component and click **+**.

A menu expands.

#### Step 6

In the menu, click **Integer**.

A datapool item of type `Integer` is added.

#### Step 7

Rename the datapool item to `Volume indicator`.

You added three events and a datapool item.



### Adding a dynamic state machine and modeling the behavior

The following instructions guide you through the process of adding a dynamic state machine. The haptic dynamic state machine that you model is used to control the volume.

Prerequisite:

- You completed the previous instruction.

#### Step 1

In the **Navigation** component, go to **Dynamic state machines** and click **+**.

A menu expands.

#### Step 2

In the menu, click **Haptic dynamic state machine**.

A Haptic dynamic state machine is added and displayed in the content area.

#### Step 3

Rename the dynamic state machine to `Volume control`.

#### Step 4

Drag an Initial state from the **Toolbox** into the dynamic state machine.

#### Step 5

Drag a View state from the **Toolbox** into the dynamic state machine.

Along with the View state, a View is added to the EB GUIDE model.

#### Step 6

In the **Navigation** component, click the View state.

#### Step 7

Press the **F2** key, and rename the View state to `Volume`.

#### Step 8

Add a transition from the Initial state to the `Volume View` state.



## Modeling a slider

The following instructions guide you through the process of modeling a horizontal slider indicator. The slider indicator shows the volume during run-time.

The slider indicator consists of two rectangles. One rectangle represents the background of the slider. The second rectangle indicates the volume.

Prerequisite:

- You completed the previous instruction.

### Step 1

In the **Navigation** component, expand the `Volume` view state. Double-click the **View**.

The content area displays the **View**.

### Step 2

Drag a **Rectangle** from the **Toolbox** into the **View**.

### Step 3

In the **Navigation** component, click the **Rectangle**, and press the **F2** key.

### Step 4

Rename the **Rectangle** to `Slider background`.

### Step 5

To change the appearance of `Slider background`, click the **Rectangle**, and go to the **Properties** component.

#### Step 5.1

Enter 500 in the `width` text box.

#### Step 5.2

Enter 125 in the `x` text box.

#### Step 5.3

Enter 300 in the `y` text box.

### Step 6

Drag a **Rectangle** from the **Toolbox** into `Slider background` in the **Navigation** component.

The **Rectangle** is added as a child widget to `Slider background`.

### Step 7

In the **Navigation** component, click the **Rectangle**, and press the **F2** key.

### Step 8

Rename the **Rectangle** to `Indicator`.

### Step 9

To change the appearance of `Indicator`, click the **Rectangle**, and go to the **Properties** component.


Step 9.1

Enter 40 in the `width` text box.

Step 9.2

Enter 80 in the `height` text box.

Step 9.3

Next to the `x` property, click the  button.

A menu expands.

Step 9.4

In the menu, click **Add link to datapool item**.


A dialog opens.

Step 9.5

From the list, select the `Volume indicator` datapool item.

Step 9.6

Click **Accept**.

The dialog closes. The  button is displayed next to the `x` property. The values of `x` and `Volume indicator` are now linked.

Step 9.7

Enter 10 in the `y` text box.

Step 9.8

Select black for the `fillColor` property.

You added two rectangles to the View. You changed the appearance of the rectangles.

Step 10

In the **Datapool** component, click the `Volume indicator` datapool item.

Step 11

In the `Value` text box enter 10.

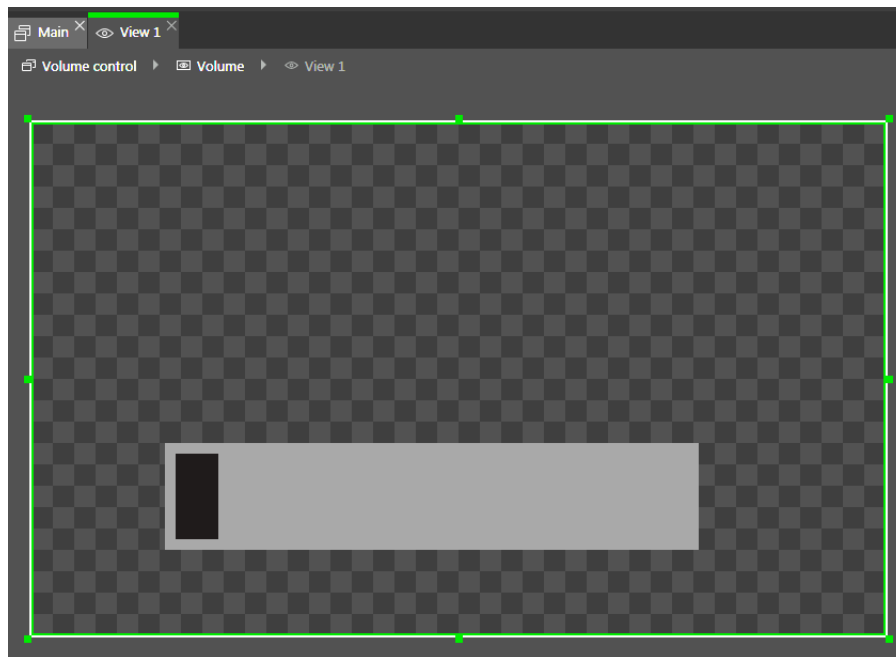


Figure 14.1. Appearance of `View 1` with two rectangles

In the content area, the `Indicator` rectangle changes the position.

The `Volume indicator datapool` item controls the `x` position of the `Indicator Rectangle`.



#### Adding states to the **Main** state machine

In the following instructions, you add an Initial state and a View state to the **Main** state machine. You use the View state to run the dynamic state machine in parallel to other state machines.

Prerequisite:

- You completed the previous instruction.

##### Step 1

In the **Navigation** component, double-click **Main**.

The **Main** state machine is displayed in the content area.

##### Step 2

Drag an Initial state from the **Toolbox** into the state machine.

##### Step 3

Drag a View state from the **Toolbox** into the state machine.

Along with the View state, a View is added to the EB GUIDE model.



#### Step 4

Rename the View state to `Home`.

#### Step 5

In the content area, click the Initial state.

#### Step 6

Add a transition from the Initial state to the `Home View` state.

#### Step 7

In the **Navigation** component, click **Main**.

#### Step 8

In the **Properties** component, select the `Dynamic state machine list` check box.

With these steps done, you can use EB GUIDE Script functions that are related to dynamic state machines.

You added an initial state and a view state to the **Main** state machine. The haptic dynamic state machine runs in parallel to the **Main** state machine.



### Adding internal transitions to the **Main** state machine

In the following instruction, you add internal transitions. You use the internal transitions to start (push) and stop (pop) the dynamic state machine during run-time.

Prerequisite:

- You completed the previous instruction.

#### Step 1

In the **Navigation** component, click the **Main** state machine.

#### Step 2

In the **Properties** component, go to **Internal transitions**, and click **+**.

An internal transition is added to the state machine. The internal transition is visible in the **Navigation** component.

#### Step 3

Add two more internal transitions.

#### Step 4

In the **Navigation** component, click the first internal transition.

##### Step 4.1

Go to the **Properties** component.

##### Step 4.2

In the **Trigger** combo box, search for the `Volume up` event and double-click it.

##### Step 4.3

Next to the **Action** property, click **+**.

#### Step 4.4

Enter the following EB GUIDE Script:

```
function()  
{  
  dp:"Volume indicator" = dp:"Volume indicator" + 20  
  f:pushDynamicStateMachine(popup_stack:Main, sm:"Volume control", 0)  
}
```

#### Step 4.5

Click **Accept**.

The action is added to the transition. In the **Navigation** component, the internal transition is renamed to Volume up.

#### Step 5

In the **Navigation** component, click the second internal transition.

##### Step 5.1

Go to the **Properties** component.

##### Step 5.2

In the **Trigger** combo box, search for the `Volume down` event and double-click it.

##### Step 5.3

Next to the **Action** property, click **+**.

##### Step 5.4

Enter the following EB GUIDE Script:

```
function()  
{  
  dp:"Volume indicator" = dp:"Volume indicator" - 20  
  f:pushDynamicStateMachine(popup_stack:Main, sm:"Volume control", 0)  
}
```

##### Step 5.5

Click **Accept**.

The action is added to the transition. In the **Navigation** component, the internal transition is renamed to Volume down.

#### Step 6

In the **Navigation** component, click the third internal transition.

##### Step 6.1

Go to the **Properties** component.

##### Step 6.2

In the **Trigger** combo box, search for the `Close volume control` event and double-click it.

### Step 6.3

Next to the **Action** property, click **+**.

### Step 6.4

Enter the following EB GUIDE Script:

```
function()  
{  
  f:popDynamicStateMachine (popup_stack:Main,sm:"Volume control")  
}
```

### Step 6.5

Click **Accept**.

The action is added to the transition. In the **Navigation** component, the internal transition is renamed to `Close volume control`.

You added three internal transitions which start and stop the dynamic state machine. Furthermore, the internal transitions `Volume up` and `Volume down` change the position of the `Indicator` rectangle.

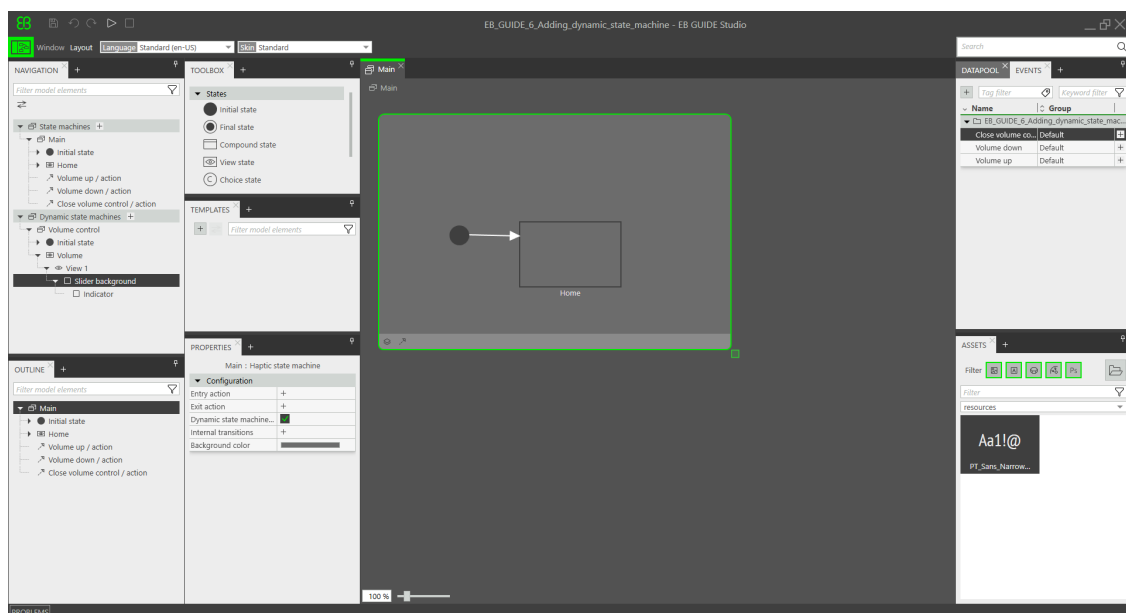


Figure 14.2. EB GUIDE model with all model elements



## Starting the simulation and testing the EB GUIDE model

Prerequisite:

- You completed the previous instruction.

To start the simulation, click **▶** in the command area.

The simulation and EB GUIDE Monitor start. The EB GUIDE model displays the `Home View` state.

#### Step 1

In EB GUIDE Monitor in the **Events** component, click .

A dialog opens.

#### Step 2

Select all of the events and click **Accept**.

#### Step 3

Next to the `Volume up` event, click  to fire the event.

The dynamic state machine is started and shows the slider indicator. The dynamic state machine overlays the `Home View` state.

When you fire the events `Volume up` or `Volume down` the black `Indicator` rectangle moves. If you fire the event `Close volume control`, the slider disappears from the `View`.

If you add additional states to the **Main** state machine, the `Volume control` dynamic state machine will overlay the other states as well.

## 14.2. Tutorial: Modeling button behavior with EB GUIDE Script

### TIP



### Default window layout

All instructions and screenshots use the default window layout. If you want to follow the instructions, we recommend to set the EB GUIDE Studio or EB GUIDE Monitor window to default layout by selecting **Layout > Reset to default layout**.

With EB GUIDE Script you can express property values, actions, or conditions and evaluate them during runtime.

The following instructions guide you through the process of using EB GUIDE Script to model the behavior of a button. The button increases in size when it is clicked and shrinks back to its original size when it reaches a defined maximum size. For best results, work through the steps in the order presented.

Approximate duration: 10 minutes.



### Adding widgets

Prerequisite:

- The **Main** state machine contains an `Initial` state and a `View` state.

- The Initial state has a transition to the View state.
- The content area displays the View.

#### Step 1

Drag a Rectangle from the **Toolbox** into the View.

#### Step 2

In the **Navigation** component, select the Rectangle, press the **F2** key, and rename the Rectangle to `Back-ground`.

#### Step 3

Drag a Rectangle from the **Toolbox** into the **Navigation** component. Place it as a child widget to the `Back-ground` Rectangle.

#### Step 4

In the **Navigation** component, select the new Rectangle, press the **F2** key, and rename the Rectangle to `Button`.

#### Step 5

Drag a Label from the **Toolbox** into the **Navigation** component. Place the Label as a child widget to the `Button` rectangle.

#### Step 6

In the **Navigation** component, select the Label, press the **F2** key, and rename the label to `Button text`.

Your widget hierarchy now looks as follows.

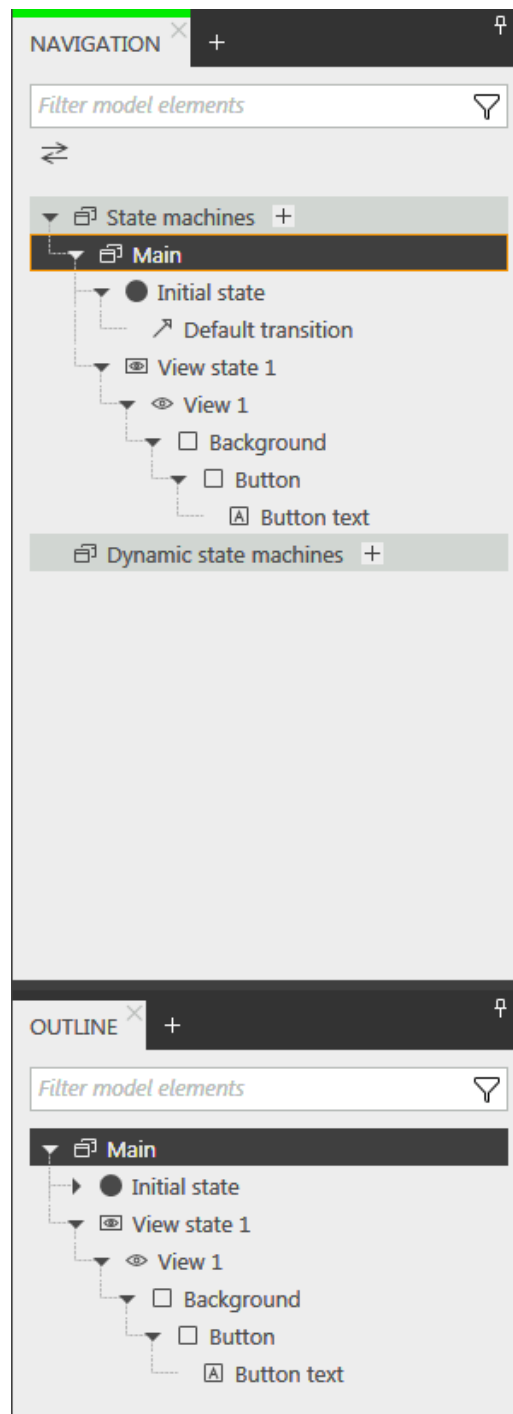


Figure 14.3. Widget hierarchy



## Configuring the background

Prerequisite:

- You completed the previous instruction.

### Step 1

In the **Navigation** component, click the `Background Rectangle`, and go to the **Properties** component.

### Step 2

Next to the `width` property, click the  button.

A menu expands.

### Step 3

In the menu, click **Add link to widget property**.


A dialog opens.

### Step 4

In the dialog, go to the view, and select its `width` property.

### Step 5

Click **Accept**.

The dialog closes. The  button is displayed next to the `width` property.

### Step 6

Link the `height` property of the `Background Rectangle` to the `height` property of the `View`.

### Step 7

Link the `x` property of the `Background Rectangle` to the `x` property of the `View`.

### Step 8

Link the `y` property of the `Background Rectangle` to the `y` property of the `View`.

The `Background Rectangle` covers the exact size and position of the `View`.




## Defining the maximum button width

A datapool item holds the value for the maximum width of the button. It can be changed during run-time.

Prerequisite:

- You completed the previous instruction.

### Step 1

In the **Datapool** component, click .

A menu expands.

Step 2

In the menu, click **Integer**.

A new datapool item of type `Integer` is added.

Step 3

Rename the datapool item to `Maximum width`.

Step 4

In the `Value` text box, enter 400.



### Configuring the button

Prerequisite:

- You completed the previous instruction.

Step 1

In the **Navigation** component, click the `Button Rectangle`, and go to the **Properties** component.

Step 1.1

Enter 50 in the `height` text box.

Step 1.2

Enter 350 in the `x` text box.

Step 1.3

Enter 215 in the `y` text box.

Step 1.4

Select blue for the `fillColor` property.

The button is now colored blue.

Step 2

In the **Widget feature properties** category, click **Add/Remove**.

The **Widget features** dialog is displayed.

Step 3

Under **Available widget features**, expand the **Input handling** category, and select the **Touch pressed** widget feature.

Step 4

Click **Accept**.

The related widget feature properties are added to the `Button Rectangle` and displayed in the **Properties** component.

Step 5

Next to the `touchPressed` property, click `{ }`.



An EB GUIDE Script editor opens.

#### Step 6

Replace the existing EB GUIDE Script with the following code:

```
function(v:touchId::int, v:x::int, v:y::int, v:fingerId::int)
{
    if (v:this.width > dp:"Maximum width") // If the button has grown
        // beyond its maximum size...
    {
        // ...reset its dimensions to the default values.
        v:this.height = 50
        v:this.width = 100
        v:this.x = 350
        v:this.y = 215
    }
    else // Otherwise...
    {

        // ... increase button size...
        v:this.width += 80
        v:this.height += 40

        // ...and move the button to keep it centered.
        v:this.x -= 40
        v:this.y -= 20
    }

    false
}
```

#### Step 7

Click **Accept**.

You configured the `Button Rectangle` and wrote an EB GUIDE Script which changes the size of the `Button Rectangle` in run-time.



### Configuring the button text

Prerequisite:

- You completed the previous instruction.

#### Step 1

In the **Navigation** component, click the `Button text Label`, and go to the **Properties** component.

#### Step 2

Enter `grow!` in the `text` text box.

Step 3

Link the `width` property of the `Button` text label to the `width` property of the `Button` rectangle.

Step 4

Link the `height` property of the `Button` text Label to the `height` property of the `Button` Rectangle.

Step 5

Enter 0 in the `x` text box.

Step 6

Enter 0 in the `y` text box.

Step 7

Next to the `horizontalAlign` property, select `center(1)`.

Now the `Button` text Label and the `Button` Rectangle are equal in size and position.



### Saving and testing the EB GUIDE model

Prerequisite:

- You completed the previous instruction.

Step 1

To save the project, click  in the command area.

Step 2

To start the simulation, click  in the command area.

Result:

The simulation starts the EB GUIDE model you created. It behaves as follows.

1. First, it displays a grey screen with a blue button in its center. The screen looks as follows.



Figure 14.4. Result

2. Whenever you click the button, it increases in size but keeps its position at the center of the screen.

3. As soon as the button width reaches the value of the `Maximum width datapool` item, it shrinks back to its original size and position.

## 14.3. Tutorial: Modeling a path gesture

### TIP



### Default window layout

All instructions and screenshots use the default window layout. If you want to follow the instructions, we recommend to set the EB GUIDE Studio or EB GUIDE Monitor window to default layout by selecting **Layout > Reset to default layout**.

Path gestures are shapes drawn by a finger on a touch screen or entered by some other input device.

The following instructions guide you through the process of modeling a path gesture.

Approximate duration: 10 minutes



Adding widgets and configuring default widget properties

Prerequisite:

- The **Main** state machine contains an Initial state and a View state.
- The Initial state has a transition to the View state.
- The content area displays a View.

#### Step 1

Drag a Rectangle from the **Toolbox** into the View.

#### Step 2

Drag a Label from the **Toolbox** into the Rectangle.

The Label is added as a child widget to the Rectangle.

The **Properties** component displays the properties of the Label.

#### Step 3

In the **Properties** component, enter 500 in the `width` text box.

#### Step 4

Select the Rectangle.

The **Properties** component displays the properties of the Rectangle.

#### Step 5

Enter 500 in the `width` text box.

#### Step 6

In the **Properties** component, go to **fillColor**, and select red.

You added two widgets and configured default widget properties.



#### Adding widget features to a rectangle

To enable the user to enter a shape starting on the widget, you add the widget feature **Path gesture** to the Rectangle. The shape is matched against a set of known shapes and, if a match is found, a gesture is recognized.

Prerequisite:

- You completed the previous instruction.

#### Step 1

Select the Rectangle.

The **Properties** component displays the properties of the Rectangle.

#### Step 2

In the **Properties** component, go to **Widget feature properties**, and click **Add/Remove**.

The **Widget features** dialog is displayed.

#### Step 3

Under **Available widget features**, expand the **Gestures** category, and select **Path gestures**.

The **Touched** widget feature is automatically selected, as it is required for the **Gestures** widget feature.

#### Step 4

Click **Accept**.

The related widget feature properties are added to the Rectangle and displayed in the **Properties** component.

#### Step 5

For the **Path gestures** widget feature edit the following properties:

##### Step 5.1

Next to the `onPath` property, click `{ }`.

An EB GUIDE Script editor opens.

##### Step 5.2

Enter the following EB GUIDE Script:

```
function (v:gestureId::int)
{
    v:this->"Label 1".text = "recognized path gesture #"
}
```

```
+ f:int2string(v:gestureId);  
}
```

#### Step 5.3

Click **Accept**.

#### Step 5.4

Next to the `onPathStart` property, click `{}`.

An EB GUIDE Script editor opens.

#### Step 5.5

Enter the following EB GUIDE Script:

```
function()  
{  
  v:this->"Label 1".text = "path gesture start";  
}
```

#### Step 5.6

Click **Accept**.

#### Step 5.7

Next to the `onPathNotRecognized` property, click `{}`.

An EB GUIDE Script editor opens.

#### Step 5.8

Enter the following EB GUIDE Script:

```
function()  
{  
  v:this->"Label 1".text = "shape not recognized";  
}
```

#### Step 5.9

Click **Accept**.

#### Step 6

To start the simulation, click  in the command area.

The simulation and EB GUIDE Monitor start. To see a reaction, draw a shape with the mouse inside the rectangle.

## 14.4. Tutorial: Creating a list with dynamic content

#### TIP



#### Default window layout

All instructions and screenshots use the default window layout. If you want to follow the instructions, we recommend to set the EB GUIDE Studio or EB GUIDE Monitor window to default layout by selecting **Layout > Reset to default layout**.

Instantiators allow creating lists dynamically during run-time. Based on a datapool item of type `list`, an instantiator displays all list elements in a pre-defined layout. If the content of the datapool item is modified, so is the appearance of the instantiator.

The following instructions guide you through the process of creating a list with dynamic content. Each list element consists of a labeled rectangle.

Approximate duration: 15 minutes.



#### Adding a datapool item

The following instructions guide you through the process of adding a datapool item of type `String list`. The datapool item provides a value for every list element of the instantiator. If the content of the datapool item is modified, so is the appearance of the instantiator.

Prerequisite:

- The **Main** state machine contains an Initial state and a View state.
- The Initial state has a transition to the View state.

#### Step 1

To display content in your list, add a datapool item of type `String list`.

In the **Datapool** component, click **+**.

A menu expands.

#### Step 2

In the menu, click **String list**.

A new datapool item of type `String list` is added.

#### Step 3

Rename the datapool item to `MyStringList`.

#### Step 4

Select the **Value** column and click  button.

An editor opens.

#### Step 4.1

Click **Add...**

A new entry is added to the table.

Step 4.2

Enter `One` in the `Value` text box.

Step 4.3

Add the values `Two`, `Three`, `Four`, and `Five` to the `MyStringList` datapool item.

Step 4.4

Click **Accept**.

You added a datapool item of type `String list`. The datapool item contains five entries.

The content of the list is displayed in the `Value` column.



## Adding widgets

Prerequisite:

- You completed the previous instruction.

Step 1

To add widgets to your View, double-click the View state in the content area.

The View is displayed in the content area.

Step 2

In the **Navigation** component, expand the View state and the View.

Step 3

Drag an Instantiator from the **Toolbox** into the view. Rename the Instantiator to `MyInstantiator`.

Step 4

Drag a Rectangle from the **Toolbox** into the Instantiator. Rename the Rectangle to `MyRectangle`.

Step 5

Drag a Label from the **Toolbox** into the Rectangle. Rename the Label to `MyLabel`.

The widget hierarchy now looks as follows.

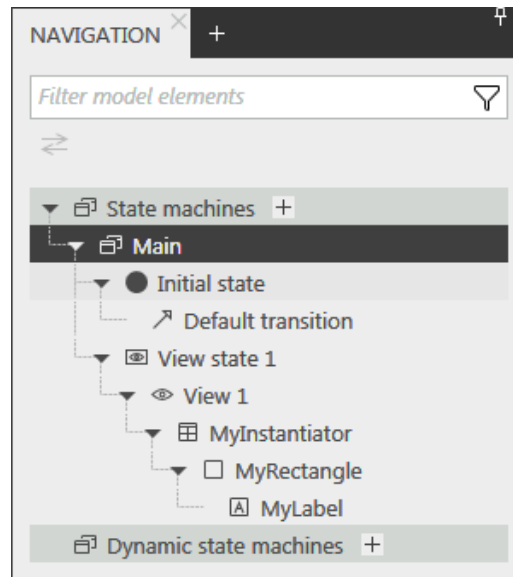


Figure 14.5. Widget hierarchy with an Instantiator



## Configuring the Instantiator

Prerequisite:

- You completed the previous instruction.

### Step 1

To change the properties of `MyInstantiator`, select the Instantiator and go to the **Properties** component.

### Step 2

Enter 300 in the `width` text box, and in the `height` text box.

### Step 3

Enter 250 in the `x` text box.

### Step 4

Enter 150 in the `y` text box.

### Step 5

To calculate the length of the list dynamically, add a conditional script.

In the **User-defined properties** category, click **+**.

A menu expands.

#### Step 5.1

In the menu, click **Conditional script**.

#### Step 5.2

Rename the property to `calculateNumItems`.



#### Step 5.3

Next to the property, select the **Value** column and click `{}`.

An EB GUIDE Script editor opens.

#### Step 5.4

Under **Triggers**, enter `dp:MyStringList`.

#### Step 5.5

Enter the following **On trigger** script:

```
function(v:arg0::bool)
{
    v:this.numItems = length dp:MyStringList;
    false
}
```

You added a script which automatically changes the number of list entries depending on the content of `MyStringList`.

#### Step 6

To arrange all labels within the Instantiator, add a layout to it.

In the **Widget feature properties** category, click **Add/Remove**.

The **Widget features** dialog is displayed.

#### Step 6.1

Under **Available widget features**, expand the **Layout** category, and select the **Box layout** widget feature to arrange the labels side by side.

#### Step 6.2

Click **Accept**.

The related widget feature properties are added to the Instantiator and displayed in the **Properties** component.

#### Step 6.3

Enter 5 in the `gap` text box to set a spacing of 5 px between each list element.

#### Step 6.4

From the `layoutDirection` drop-down list box select `vertical (1)`, to arrange the labels among each other.

You configured the Instantiator which defines the visual appearance of the list and adapts the number of list items dynamically.



### Configuring list element texts

Prerequisite:

- You completed the previous instruction.

#### Step 1

To change the appearance of the Label, select the `MyLabel` and go to the **Properties** component.

#### Step 2

Enter 0 in the `x` and `y` text boxes.

#### Step 3

Add a link from the `width` property of the Label to the `width` property of the Rectangle.

##### Step 3.1

Next to the `width` property, click the  button.

A menu expands.

##### Step 3.2

In the menu, click **Add link to widget property**.


A dialog opens.

##### Step 3.3

In the dialog, go to the Rectangle, and select its `width` property.

##### Step 3.4

Click **Accept**.

The dialog closes. The  button is displayed next to the `width` property.

#### Step 4

Add a link from the `height` property of the Label to the `height` property of the Rectangle.

#### Step 5

Next to the `horizontalAlign` property, select `center` (1).

You changed the appearance of the Label. The Label is now centered in the Rectangle.



### Configuring list elements

Prerequisite:

- You completed the previous instruction.

#### Step 1

To change the appearance of the rectangle, select the rectangle and go to the **Properties** component.

#### Step 2

To make sure that the list elements use the available width, add a link from the `width` property of the `Rectangle` to the `width` property of the `Instantiator`.

#### Step 3

Enter 50 in the `height` text box.

#### Step 4

To define a unique position for each line of your list, add the **Line index** widget feature.

##### Step 4.1

In the **Widget feature properties** category, click **Add/Remove**.

The **Widget features** dialog is displayed.

##### Step 4.2

Under **Available widget features**, expand the **List management** category, and select the **Line index** widget feature.

The `lineIndex` property is added.

#### Step 5

To fill the labels of the list with the content of `MyStringList`, add a conditional script.

##### Step 5.1

Next to the **User-defined properties** category, click **+**.

A menu expands.

##### Step 5.2

In the menu, click **Conditional script**.

##### Step 5.3

Rename the property to `setText`.

##### Step 5.4

Next to the `setText` property, select the **Value** column and click **{}**.

An EB GUIDE Script editor opens.

##### Step 5.5

Under **Triggers**, enter `v:this.lineIndex` and `dp:MyStringList`.

##### Step 5.6

Enter the following **On trigger** script:

```
function(v:arg0::bool)
{
    v:this->MyLabel.text=dp:MyStringList[v:this.lineIndex];
    false
}
```

You changed the appearance of the `Rectangle`. With the `setText` property, the labels of `MyStringList` are filled automatically with the content of `MyStringList`.



## Testing the EB GUIDE model

### Prerequisite:

- You completed the previous instruction.

### Step 1

To start the simulation, click ▶ in the command area.

### Result:

Since `MyStringList` contains five datapool items, five rectangles that are labeled from one to five are displayed in vertical arrangement.

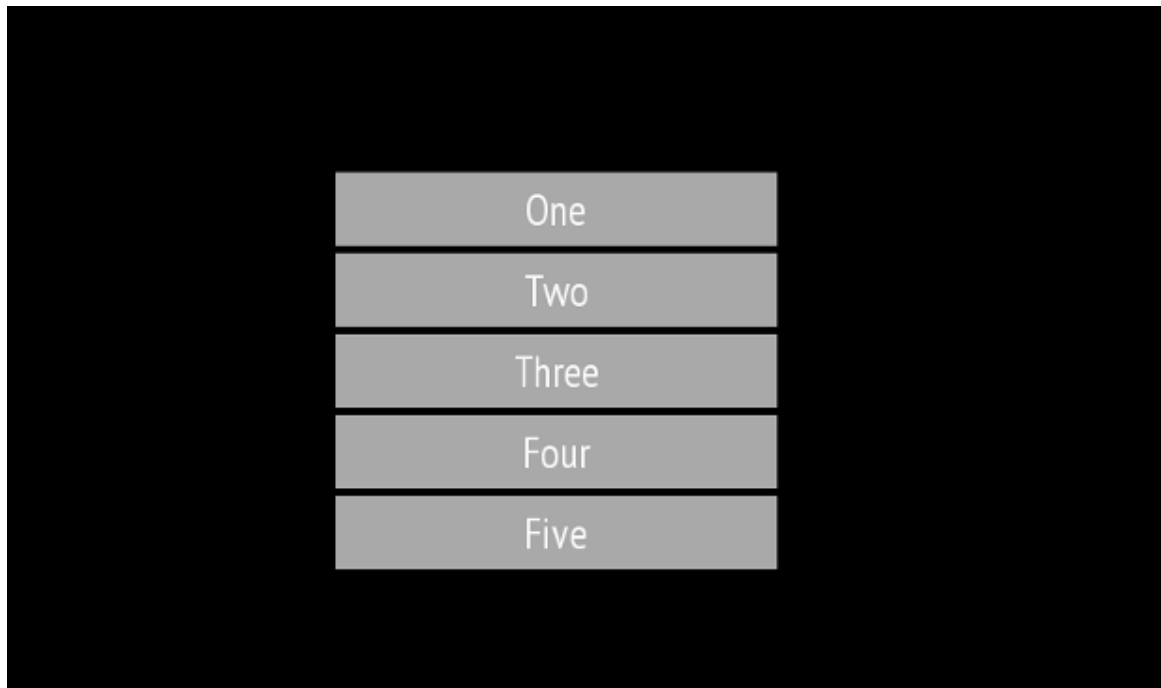


Figure 14.6. List created with an Instantiator

## 14.5. Tutorial: Making an ellipse move across the screen

### TIP



### Default window layout

All instructions and screenshots use the default window layout. If you want to follow the instructions, we recommend to set the EB GUIDE Studio or EB GUIDE Monitor window to default layout by selecting **Layout > Reset to default layout**.

The following instructions guide you through the process of animating an ellipse so that it continually moves across the screen when the simulation starts.

Approximate duration: Five minutes.



### Adding widgets

In the following steps, you add three widgets to the view and organize the hierarchy of the widgets.

Prerequisite:

- The content area displays the **Main** state machine.
- The **Main** state machine contains an Initial state and a View state.
- The Initial state has a transition to the View state.

#### Step 1

In the content area, double-click the View state.

The View is displayed in the content area.

#### Step 2

Drag an Ellipse from the **Toolbox** into the View.

#### Step 3

Drag an Animation from the **Toolbox** into the Ellipse.

#### Step 4

In the **Navigation** component, click the Animation, and press the **F2** key. Rename the Animation to `MyAnimation`.

Now, if you start the simulation, an ellipse is displayed in a View. The ellipse does not move yet.



### Adding a user-defined property of type `Conditional script`

As a next step, you add a user-defined property to the Ellipse. With the conditional script property, rendering the ellipse during simulation starts the animation.

Prerequisite:

- You completed the previous instruction.

#### Step 1

Select the Ellipse.

#### Step 2

In the **Properties** component, go to the **User-defined properties** category, and click **+**.

A menu expands.

#### Step 3

In the menu, click `Conditional script`.

A user-defined property of type `Conditional script` is added to the Ellipse.

#### Step 4

Rename the property to `startAnimation`.

#### Step 5

Next to the `startAnimation` property, select the **Value** column and click `{ }`.

An EB GUIDE Script editor opens.

#### Step 6

Enter the following EB GUIDE Script:

```
function(v:arg0::bool)
{
  f:animation_play(v:this->MyAnimation)
}
```



### Making the animation visible

The following instructions guide you through the process of making the animation visible.

Prerequisite:

- You completed the previous instruction.
- The content area displays the `View 1 View`.

#### Step 1

Go to the **Animation editor**. Next to **Animated properties**, click  and select `View 1`.


A menu expands.

#### Step 2

Under `Ellipse 1` select the `x` property and then the **Linear interpolation curve**.

#### Step 3

Click **Accept**.

The  button is displayed next to the `target` property.

#### Step 4

Link the `end` property to the `width` property of the `View`.

With these settings, when the animation starts, the `x` property of the `Ellipse` changes from zero to the width of the `View`. Thus the ellipse moves from the left boundary to the right boundary of the view.

#### Step 5

To make the animation run in infinite repetitions, enter 0 in the `repeat` property.

#### Step 6

Save the project.

#### Step 7

To start the simulation, click  in the command area.

Result:

The ellipse continually moves from the left side of the view to the right side of the view.

## 14.6. Tutorial: Adding a language-dependent text to a datapool item

### TIP



### Default window layout

All instructions and screenshots use the default window layout. If you want to follow the instructions, we recommend to set the EB GUIDE Studio or EB GUIDE Monitor window to default layout by selecting **Layout > Reset to default layout**.

EB GUIDE offers the possibility to display texts in the user's preferred language. The following instructions show you how to model a label that changes with an English, French, and German user interface.

Approximate duration: 15 minutes



### Linking a widget property to a datapool item

The following instructions guide you through the process of linking the `text` property of a Label to a datapool item. In run-time the displayed text is provided by the datapool item.

Prerequisite:

- Three languages are added to the EB GUIDE model: English, German, and French. The name of **Language 1** is set to `German` and the name of **Language 2** is set to `French`.
- The Main state machine contains an Initial state and a View state.
- The Initial state has a transition to the View state.
- The content area displays the View.
- The View state contains a Label.
- The datapool item is not linked to a datapool item or widget property.
- The datapool item does not have a scripted value.

#### Step 1

Click the Label.

#### Step 2

In the **Properties** component, go to the `text` property, and click the  button next to the property.

#### Step 3

In the menu, click **Add link to datapool item**.

A dialog opens.

#### Step 4

To add a new datapool item, enter `Welcome_text` in the text box.

#### Step 5

Click **Add datapool item**.

#### Step 6

Click **Accept**.

The datapool item `Welcome_text` is added.

In the content area, the Label no longer displays any text.



### Enter language-dependent text to the datapool item

The following instructions guide you through the process of adding language-dependent text to the datapool item. For every language the `Value` property has a different text.



Prerequisite:

- You completed the previous instruction.

Step 1

In the **Datapool** component, click the `Welcome_text` datapool item.

Step 2

Click the  button.

Step 3

In the menu, click **Add language support**.

In the **Properties** component, the language properties are displayed.

Step 4

In the **Datapool** component, in the `Value` text box, enter `Welcome`.

In the content area, the Label displays `Welcome`.

Step 5

Go to the **Properties** component.

Step 6

In the `German` text box, enter `Willkommen`.

In the `Language` box in the upper left corner, change the language to `German`.

In the content area, the Label displays `Willkommen`.

Step 7

In the `French` text box, enter `Bienvenue`.

In the `Language` box in the upper left corner, change the language to `French`.

In the content area, the Label displays `Bienvenue`.

You have added language support for English, German and French and defined a language-dependent text label.



### Changing the language during run-time

The following instructions guide you through the process of creating a script for changing the language during run-time. Each time, the user clicks the label, the display language changes.

Prerequisite:

- You completed the previous instruction.

Step 1

In the **Datapool** component, click **+**.

A menu expands.

#### Step 2

In the menu, click `Integer`.

A datapool item of type `Integer` is added.

#### Step 3

Rename the datapool item to `SelectedLanguage`.

#### Step 4

In the **Navigation** component, click the `Label 1 Label`.

#### Step 5

In the **Properties** component, go to the **Widget feature properties** and click **Add/Remove**.

The **Widget features** dialog is displayed.

#### Step 6

Under **Available widget features**, expand the **Input handling** category, and select the **Touch pressed** widget feature.

#### Step 7

Click **Accept**.

The related widget feature properties are added to the `Label` and displayed in the **Properties** component.

#### Step 8

Next to the `touchPressed` property, click `{ }`.

An EB GUIDE Script editor opens.

#### Step 9

Replace the existing EB GUIDE Script with the following code:

```
function(v:touchId::int, v:x::int, v:y::int, v:fingerId::int)
{
    if (dp:SelectedLanguage == 0) // Standard selected
    {
        f:setLanguage(l:German, true)
        dp:SelectedLanguage = 1
    }
    else if (dp:SelectedLanguage == 1) // German selected
    {
        f:setLanguage(l:French, true)
        dp:SelectedLanguage = 2
    }
    else if (dp:SelectedLanguage == 2) // French selected
    {
        f:setLanguage(l:Standard, true)
        dp:SelectedLanguage = 0
    }
    false
}
```

```
}
```

#### Step 10

Click **Accept**.

You configured the Label and wrote an EB GUIDE Script which changes the language of the Label during run-time.

Result:

You added a datapool item of type `String` to the EB GUIDE model. The datapool item has different values for languages. In English the value is `Welcome`. In German the value is `Willkommen`. In French the value is `Bienvenue`. The datapool item is linked to the `text` property of the Label. Every time you change the language of the EB GUIDE model the text of the Label changes too.

## 14.7. Tutorial: Working with a 3D graphic

### TIP



#### Default window layout

All instructions and screenshots use the default window layout. If you want to follow the instructions, we recommend to set the EB GUIDE Studio or EB GUIDE Monitor window to default layout by selecting **Layout > Reset to default layout**.

EB GUIDE Studio offers the possibility to use 3D graphics in your EB GUIDE model.

The following instructions guide you through the process of adding a 3D graphic to your EB GUIDE model. The instructions show you how to import a 3D graphic and how to modify the appearance of the imported 3D graphic using widget features. For best results, work through the following steps in order presented.

### NOTE



#### 3D graphic

To create a 3D graphic file, use third-party 3D modeling software.

Only the renderers for OpenGL ES 2.0 or higher can display 3D graphics. Make sure that your graphics driver is compatible to the version of the renderer. The supported 3D graphic formats are COLLADA ( `.dae` ) and Filmbox ( `.fbx` ). For best results, use the Filmbox format.

To be able to apply textures to a mesh, a 3D object needs to have texture coordinates. To add texture coordinates, use third-party 3D modeling software.

Approximate duration: 15 minutes.



## Importing a 3D graphic

The following instructions guide you through the process of importing a 3D graphic file to an EB GUIDE project.

Prerequisite:

- The content area displays the **Main** state machine.
- The **Main** state machine contains an Initial state and a View state.
- The Initial state has a transition to the View state.
- A 3D graphic file is available. The file contains a camera, a light source, and one object containing a mesh and at least one material.

### Step 1

In the content area, double-click the View state.

The View is displayed in the content area.

### Step 2

Drag a Scene graph from the **Toolbox** into the View.

The View displays the empty bounding box.

### Step 3

Rename the Scene graph to `My3DGraphic`.

### Step 4

In the **Properties** component, click **Import file**.

A dialog opens.

### Step 5

Navigate to the folder where the 3D graphic file is stored.

### Step 6

Select the 3D graphic file.

### Step 7

Click **Open**.

The import starts.

The **Import successful** or **Import with warnings** dialog is displayed. Here you can check the import log file.

### Step 8

Click **OK**.

The View displays the 3D graphic. The **Navigation** component displays the imported widget tree with the Scene graph as a parent node. `My3DGraphic` contains a `RootNode` that has at least one Mesh with Material, camera and several other child widgets depending on the content of your 3D graphic file.



## Adding widgets

The following instructions guide you through the process of adding an additional light source to your 3D graphic.

Prerequisite:

- You completed the previous instruction.

### Step 1

In the **Navigation** component, expand `RootNode`.

### Step 2

Drag a Directional light from the **Toolbox** to `RootNode`.

You added a Directional light to `My3DGraphic`. You can manipulate and transform this Directional light with the transformation properties of the `RootNode`.

### Step 3

To add the light source and place it with default widget properties different from the `RootNode` Scene graph, do the following:

#### Step 3.1

Drag a Scene graph node from the **Toolbox** to `RootNode`.

#### Step 3.2

Rename the Scene graph node to `MyLight`.

#### Step 3.3

Drag a Directional light from the **Toolbox** to `MyLight`.

You added a Directional light to `My3DGraphic`. To change the placing of the Directional light, change the properties of `MyLight`.



## Changing meshes

Prerequisite:

- You completed the previous instruction.
- The `$GUIDE_PROJECT_PATH/<project name>/resources/<3D graphic name>` folder contains an additional `.ebmesh` file.

### Step 1

In the **Navigation** component, click `Mesh 1`, and go to the **Properties** component.

### Step 2

From the `mesh` combo box select the `.ebmesh` file from the resource folder mentioned above.

The view displays the Scene graph with the new mesh.

#### Step 3

Alternatively, drag an `.ebmesh` file from the **Assets** component into the `mesh` drop-down list box.

The View displays the Scene graph with the new Mesh.



### Changing textures

The following instructions guide you through the process of adding and modifying textures of your 3D graphic.

Prerequisite:

- You completed the previous instruction.
- The `$GUIDE_PROJECT_PATH/<project name>/resources/<3D graphic name>` folder contains a `.png` or `.jpg` image file.

#### Step 1

In the **Navigation** component, click the Material, and go to the **Properties** component.

#### Step 2

In the **Widget feature properties** category, click **Add/Remove**.

The **Widget features** dialog is displayed.

#### Step 3

Under **Available widget features**, expand the **3D** category, and select a texture widget feature, for example **Diffuse texture**.

#### Step 4

Click **Accept**.

The related widget feature properties are added to the Material and displayed in the **Properties** component.

#### Step 5

In the **Properties** component, select an image from the `diffuseTexture` combo box.

The view displays a scene graph with the new texture.

#### NOTE



#### Usage of 3D widget features

These instructions are valid for the following widget features from the category **3D**:

- ▶ **Ambient texture**
- ▶ **Diffuse texture**
- ▶ **Base color texture**
- ▶ **Emissive texture**
- ▶ **Light map texture**
- ▶ **Normal map texture**
- ▶ **Opaque texture**
- ▶ **Reflection texture**
- ▶ **Specular texture**
- ▶ **Reflectance texture**



#### Displaying 3D object several times

The following instructions guide you through the process of adding an additional camera to be able to display the 3D object of your 3D graphic several times. You will be able to have different points of view of the same object.

Prerequisite:

- You completed the previous instruction.

##### Step 1

In the **Navigation** component, click `My3DGraphic` and go to the **Properties** component.

##### Step 2

Enter 800 in the `width` text box and 480 in the `height` text box.

The `My3DGraphic` Scene graph has the size of the view.

##### Step 3

In the **Navigation** component, expand `RootNode` and `Camera001`.

##### Step 4

Click `Camera 1` and go to the **Properties** component.

##### Step 5

In the **Widget feature properties** category, click **Add/Remove**.

The **Widget features** dialog is displayed.

Step 6

Under **Available widget features**, expand the **3D** category, and select **Camera viewport**.

Step 7

Click **Accept**.

The related widget feature properties are added to `Camera 1` and displayed in the **Properties** component.

Step 8

Drag a Camera from the **Toolbox** to the Scene graph node `Camera001`.

You added a second camera.

Step 9

Click `Camera 2` and go to the **Properties** component.

Step 10

In the `nearPlane`, `farPlane` and `fieldOfView` text boxes enter the same values that `Camera 1` has.

Both `Camera 1` and `Camera 2` have the same viewing position.

Step 11

In the **Widget feature properties** category, click **Add/Remove**.

The **Widget features** dialog is displayed.

Step 12

Under **Available widget features**, expand the **3D** category, and select **Camera viewport**.

Step 13

Click **Accept**.

The related widget feature properties are added to `Camera 2` and displayed in the **Properties** component.

Step 14

In the **Properties** component, enter 100 in `viewportX` and `viewportY` text boxes.

In the View, the 3D object is displayed two times with different x-coordinate and y-coordinate.

## 14.8. Tutorial: Rendering gamma correctly

**TIP**



**Default window layout**

All instructions and screenshots use the default window layout. If you want to follow the instructions, we recommend to set the EB GUIDE Studio or EB GUIDE Monitor window to default layout by selecting **Layout > Reset to default layout**.

In EB GUIDE Studio you can perform gamma correction for the following:

- Display



- ▶ Image
- ▶ Texture

The following instructions show you how to configure gamma correction. For best results, work through the steps in order presented.

Approximate duration: 15 minutes.



### Configuring gamma encoding for displays

The following steps guide you through the process of setting up EB GUIDE Studio to output gamma-encoded values to the display.

Prerequisite:

- The content area displays the **Main** state machine.
- The **Main** state machine contains an Initial state and a View state.
- The Initial state has a transition to the View state.

#### Step 1

Click .

The project center opens.

#### Step 2

In the navigation area, click **Configure > Profiles**.

#### Step 3

For each profile that is sRGB-aware configure the following:

##### Step 3.1

In the content area, click the **Scenes** tab.

##### Step 3.2

From the `colorMode` drop-down list box, select the following:

- ▶ If your rendering pipeline uses the hardware sRGB support of the GPU, select `32-bit sRGB (4)`
- ▶ If your rendering hardware does not support sRGB, select `32-bit sRGB (Emulated) (5)`.

To apply changes in the edit mode, restart EB GUIDE Studio.

#### NOTE



#### Rendering hardware

The OpenGL 3 renderer always has hardware sRGB support. The OpenGL renderer, which uses the OpenGL ES 2.0 API, uses only hardware sRGB support if the hardware supports it through the appropriate OpenGL ES extensions. This is automatically detected.

In case your OpenGL ES 2.0 hardware does not support sRGB, the renderer automatically falls back to `32-bit sRGB (Emulated) (5)`, which uses fragment shaders for the conversion.

Note that hardware sRGB mode does not work on all systems, even if they support OpenGL ES 3.0. In this case, switch to `32-bit sRGB (Emulated) (5)`.



#### Configuring gamma encoding for images

Prerequisite:

- You completed the previous instruction.
- An image file is available in the resource folder.

##### Step 1

Go to the project editor and double-click the View. Drag an Image from the **Toolbox** into the View.

##### Step 2

In the **Properties** component, from the `image` drop-down box, select an image file.

The image probably appears too bright.

This is caused by the sRGB color mode that was configured in the scene properties and now applies gamma-correction to an already gamma-corrected image.

##### Step 3

To configure the image to be gamma encoded, in the **Properties** component select `sRGB`.

The image is now displayed and processed correctly in blending operations.



#### Configuring gamma encoding for textures

Prerequisite:

- You completed the previous instructions.
- A 3D file, for example an `.fbx` file, with at least one textured 3D object is available with a diffuse texture.

#### Step 1

Drag a Scene graph from the **Toolbox** into the View.

#### Step 2

In the **Properties** component, click **Import file** and select a 3D file.

#### Step 3

In the **Navigation** component locate the imported Scene graph. Within the Scene graph structure, select a Material widget that uses a diffuse texture.

#### Step 4

In the **Properties** component, in the **Widget feature properties**, click **Add/Remove**.

#### Step 5

From the **3D** category, select **Diffuse texture**.

#### Step 6

Select the `diffuseSRGB` property.

The texture is treated as gamma-encoded image and is linearized before it is being used in lighting computations.

## 14.9. Tutorial: Using view transition animations

### TIP



### Default window layout

All instructions and screenshots use the default window layout. If you want to follow the instructions, we recommend to set the EB GUIDE Studio or EB GUIDE Monitor window to default layout by selecting **Layout > Reset to default layout**.

View transition animation (VTA) is an animation that is done while moving from one view to another. The following instructions guide you through the process of creating these animations. You are going to create a model with views and animations that are played when you change views. You are going to create the following elements:

- ▶ Two view states
- ▶ Navigation elements, such as buttons and labels
- ▶ Events to trigger changes to the view states
- ▶ Animations that are played when you transition to another state

Approximate duration: 30 minutes.



### Creating the first view state

Create the first view and a button.

Prerequisite:

- The content area displays the **Main** state machine.
- The **Main** state machine contains an Initial state and a View state.
- The Initial state has a transition to the View state.

#### Step 1

In the **Navigation** component, rename the View state to `FirstState` and the View to `FirstView`.

#### Step 2

Open the `FirstView`.

#### Step 3

From the **Toolbox** component, drag a Rectangle into the `FirstView` and rename it to `RectNextView`.

This Rectangle is for the button that triggers the transition.

#### Step 4

In the **Properties** component, go to the **Widget feature properties** category and click **Add/Remove**. The **Widget features** dialog is displayed.

#### Step 5

Under **Available widget features**, expand the **Effect** category and select **Border**.

#### Step 6

Click **Accept**.

The **Properties** component displays the related widget feature properties.

#### Step 7

In the **Properties** component, enter the following:

- ▶ In the `width` text box, enter 220.
- ▶ In the `height` text box, enter 70.
- ▶ In the `x` text box, enter 290.
- ▶ In the `y` text box, enter 150.
- ▶ Set `fillColor` to black.
- ▶ In the `borderThickness` text box, enter 2.
- ▶ Set `borderColor` to white.

#### Step 8

From the **Toolbox** component, drag a Label into the **Navigation** component and add it as a child widget of `FirstView`.

#### Step 9

Rename the Label to `LabelNextView`.

#### Step 10

In the **Properties** component, enter the following:

- ▶ In the **text** text box, enter `Go to the next view`.
- ▶ In the **font** text box, enter 25.
- ▶ Set the **horizontalAlign** to **centered**.

#### Step 11

Link the dimensions of the `Label` to the dimensions of the `Rectangle`. Link the following properties:

- ▶ Link the **width** property of `LabelNextView` to **width** of `RectNextView`.
- ▶ Link the **height** property of `LabelNextView` to **height** of `RectNextView`.
- ▶ Link the **x** property of `LabelNextView` to **x** of `RectNextView`.
- ▶ Link the **y** property of `LabelNextView` to **y** of `RectNextView`.

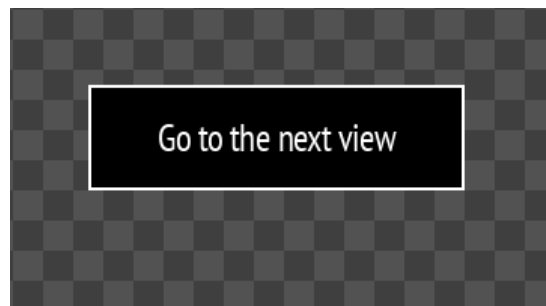


Figure 14.7. The `FirstView` with the button



### Creating the second view state

The second view contains a button. Create this view by copying and renaming the elements you already created.

Prerequisite:

- You completed the previous instruction.

#### Step 1

Select the **Main** tab.

#### Step 2

Copy and paste the `FirstState` state.

#### Step 3

In the **Navigation** component, find the new state you created and rename the following widgets:

- ▶ Rename the **View** state to `SecondState`.
- ▶ Rename the **View** to `SecondView`.
- ▶ Rename `RectNextView` to `RectGoBack`.

- Rename `LabelNextView` to `LabelGoBack`.

#### Step 4

Double-click `LabelGoBack`, and in the text text box, enter `Go back`.

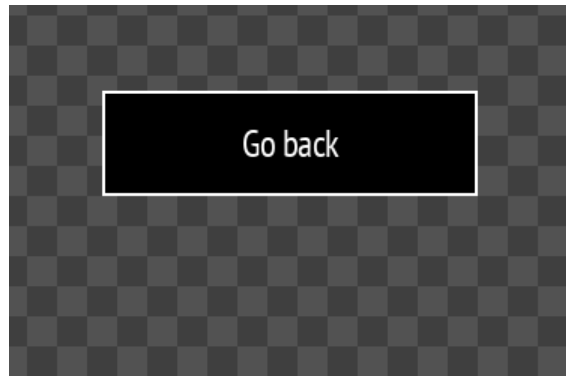


Figure 14.8. The `SecondView` with the button



### Creating transitions and events

Prerequisite:

- You completed the previous instruction.

#### Step 1

Double-click the **Main** state machine.

#### Step 2

Create transitions from the edges of the **Main** state machine to both view states.

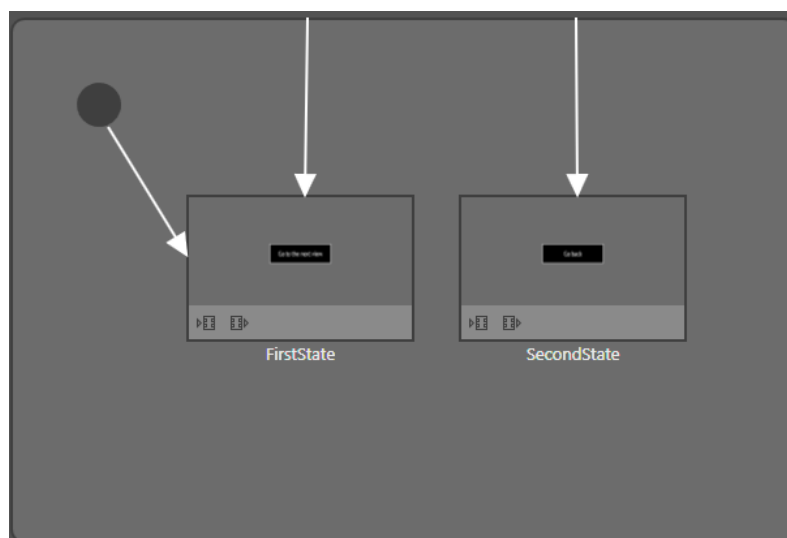


Figure 14.9. The `Main` state machine with transitions

#### Step 3

Select the transition from the **Main** state machine to `FirstState`.

#### Step 4

In the **Properties** component, in the **Trigger** combo box, enter `goToFirstState` and click **Add event**.

A new event is created.

#### Step 5

Select the transition to `SecondState`.

#### Step 6

In the **Properties** component, in the **Trigger** combo box, enter `goToSecondState` and click **Add event**.

A new event is created.



### Connecting buttons and events

Now you define the following behavior: when a button is clicked, the transition to another state is triggered. For this you use EB GUIDE Script.

Prerequisite:

- You completed the previous instruction.

#### Step 1

In the **Navigation** component, double-click `LabelNextView`.

##### Step 1.1

In the **Properties** component, go to the **Widget feature properties** category and click **Add/Remove**. The **Widget features** dialog is displayed.

##### Step 1.2

Under **Available widget features**, expand the **Input handling** category and select **Touch released**.

##### Step 1.3

Click **Accept**.

The related widget feature properties are added to the **Properties** component.

##### Step 1.4

Next to the `touchShortReleased` property, click `{ }`.

An EB GUIDE Script editor opens.

##### Step 1.5

Enter the following EB GUIDE Script that fires the `goToSecondState` event when the button is clicked:

```
function(v:touchId::int, v:x::int, v:y::int, v:fingerId::int)
{
    fire ev:goToSecondState()
```

```
false  
}
```

#### Step 1.6

Click **Accept**.

#### Step 2

In the **Navigation** component, double-click `LabelGoBack`.

#### Step 2.1

In the **Properties** component, go to the **Widget feature properties** category and click **Add/Remove**. The **Widget features** dialog is displayed.

#### Step 2.2

Under **Available widget features**, expand the **Input handling** category and select **Touch released**.

#### Step 2.3

Click **Accept**.

The related widget feature properties are added to the **Properties** component.

#### Step 2.4

Next to the **touchShortReleased** property, click `{ }`.

An EB GUIDE Script editor opens.

#### Step 2.5

Enter the following EB GUIDE Script that fires the `goToFirstState` event when the button is clicked:

```
function(v:touchId::int, v:x::int, v:y::int, v:fingerId::int)  
{  
    fire ev:goToFirstState()  
    false  
}
```

#### Step 2.6

Click **Accept**.



### Creating the `FirstView` entry animation

Enable the **VTA** component and create an animation that moves the button in from the right. To create an animation, you need to define which property is animated, how long the animation lasts, where it starts, and where it ends. In this tutorial only the `x` property is used.

Prerequisite:

- You completed the previous instruction.



#### Step 1

The **VTA** component is not visible in the default layout. You need to enable it.

In the command area click **Layout > VTA (view transition animations)**.

The **VTA** component is displayed.

#### Step 2

In the **Navigation** component double-click `FirstState`.

#### Step 3

In the **VTA** component click **+** and select **Entry animation**

The **Entry animation** table is displayed.

The **Animation editor** is displayed below the content area.

#### Step 4

Click **+** and select **Exit animation**.

The **Exit animation** table is displayed.

#### Step 5

In the **Animation editor** in the drop-down list select `Entry animation 1`.

#### Step 6

Click **+** and select **Destination: FirstView**. The **Animation properties** dialog is displayed.

#### Step 7

Click `RectNextView`, then `x` then **Fast start curve** and **Accept**.

A new animation is added to the **Animated properties** list.

#### Step 8

In the **Properties** component enter the following:

- ▶ In the `start` text box, enter 900.
- ▶ In the `end` text box, enter 290.

When you start the simulation you can see the button move in.



#### Creating the `FirstView` exit animation

Create an animation that moves the button out to the right.

Prerequisite:

- You completed the previous instruction.

#### Step 1

In the **Animation editor** in the drop-down list select `Exit animation 1`.

#### Step 2

Click **+** and select **Source:** `FirstView`. The **Animation properties** dialog is displayed.

#### Step 3

Click `RectNextView` then **x** then **Fast start curve** and **Accept**.

A new animation is added to the **Animated properties** list.

#### Step 4

In the **Properties** component enter the following:

- ▶ In the `duration` text box, enter 500.
- ▶ In the `start` text box, enter 290.
- ▶ In the `end` text box, enter 800.



### Creating the `SecondView` entry animation

Create an animation that moves the button in from the right.

Prerequisite:

- You completed the previous instruction.

#### Step 1

In the **Navigation** component double-click `SecondView`.

`SecondView` is displayed in the content area.

#### Step 2

In the **VTA** tab click **+**

#### Step 3

Add an **Entry animation** and an **Exit animation**.

The **Animation editor** is displayed below the content area.

#### Step 4

In the **Animation editor**, in the drop-down list, select `Entry animation 2`.

#### Step 5

Click **+** and select **Destination:** `SecondView`. The **Animation properties** dialog is displayed.

#### Step 6

Click `RectGoBack` then **x** then **Fast start curve** and **Accept**.

A new animation is added to the **Animated properties** list.

#### Step 7

In the **Properties** component enter the following:

- ▶ In the `start` text box, enter 900.

- ▶ In the `end` text box, enter 290.



#### Creating the `SecondView` exit animation

Create an animation that moves the button out to the right.

Prerequisite:

- You completed the previous instruction.

##### Step 1

In the **Animation editor**, in the drop-down list, select `Exit animation 2`.

##### Step 2

Click **+** and select **Source:** `SecondView`. The **Animation properties** dialog is displayed.

##### Step 3

Click `RectGoBack` then `x` then **Fast start curve** and **Accept**.

A new animation is added to the **Animated properties** list.

##### Step 4

In the **Properties** component enter the following:

- ▶ In the `duration` text box, enter 500.
- ▶ In the `start` text box, enter 290.
- ▶ In the `end` text box, enter 800.



#### Saving and testing the EB GUIDE model

Prerequisite:

- You completed the previous instruction.

##### Step 1

To save the project, click  in the command area.

##### Step 2

To start the simulation, click  in the command area.

Click `Go to the next view`. The view changes and an animation is played.

Click `Go back`. The view changes back to the first view and an animation is played.

## 14.10. Tutorial: Using script curves for animations

### TIP



#### Default window layout

All instructions and screenshots use the default window layout. If you want to follow the instructions, we recommend to set the EB GUIDE Studio or EB GUIDE Monitor window to default layout by selecting **Layout > Reset to default layout**.

Use a script curve when you want to define your own curve for an animation. Defining your own curve can be necessary when the other animation curves are not suitable or when you just want to define a custom curve. In this tutorial you are going to create a simple model with two script curves for two animations. You are going to create the following elements:

- ▶ A View state
- ▶ Two Rectangle widgets
- ▶ Two Animation widgets with script curves that animate the positions of the rectangle widgets

This results in a model with two rectangles. One rectangle moves down. The other moves to the side.

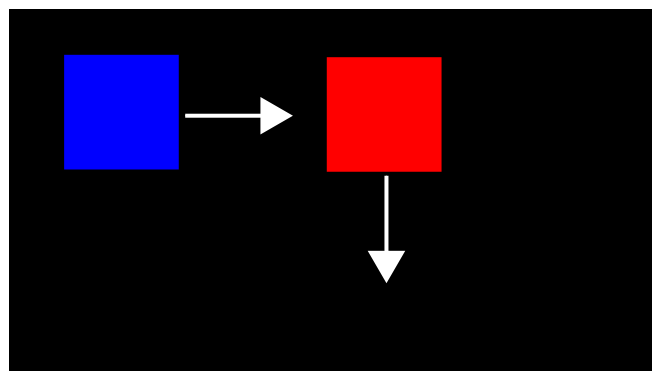


Figure 14.10. The rectangles with their movement direction

Approximate duration: 15 minutes.



#### Creating the first script curve

Prerequisite:

- The **Main** state machine contains an Initial state and a View state called `FirstState` and a View called `FirstView`.
- The Initial state has a transition to `FirstState`.
- The content area displays the `FirstView` View.

#### Step 1

From the **Toolbox** component, drag a Rectangle into the View and rename it to `BlueRectangle`

#### Step 2

In the **Properties** component, set the `fillColor` to blue.

#### Step 3

From the **Toolbox** component, drag an Animation into the view and rename it to `MoveAnimation`.

#### Step 4

In the **Datapool** component, add a datapool item of type `Float` and rename it to `xFloat`.

#### Step 5

In the **Navigation** component, select `BlueRectangle`.

#### Step 6

In the **Properties** component, go to the **User-defined properties** category, and click **+**.

A menu opens.

#### Step 7

In the menu, select **Conditional script**.

`Conditional script 1` is added to the **User-defined properties**.

#### Step 8

Rename `Conditional script 1` to `StartBlueAnimation`.

#### Step 9

Next to `StartBlueAnimation`, click `{ }`.

The EB GUIDE Script editor opens.

#### Step 10

Enter the following script:

```
function(v:arg0::bool)
{
    f:animation_play(v:this->^->"MoveAnimation")
}
```

#### Step 11

In the **Navigation** component, select `FirstView`.

#### Step 12

In the **Animation editor**, next to the **Animated properties** click **+** and select `FirstView`.

The **Animation properties** dialog opens.

#### Step 13

Under `BlueRectangle`, select the `x` property and then the `Script curve`

#### Step 14

Click **Accept**.

`Script curve 1` is added to the **Animation editor**.

#### Step 15

**Rename** Script curve 1 to BlueCurve.

#### Step 16

In the **Properties** component, next to the curve property click {}.

The EB GUIDE Script editor opens.

#### Step 17

Enter the following script:

```
function(v:diff::int, v:t_anim::int)
{
    dp:xFloat+=0.2
    f:floor(dp:xFloat*dp:xFloat)
}
```



### Creating the second script curve

Prerequisite:

- You have finished the previous instruction.

#### Step 1

From the **Toolbox** component, drag a Rectangle into **FirstView** and rename it to RedRectangle.

#### Step 2

In the **Properties** component, set the fillColor to red.

#### Step 3

In the **Datapool** component, add a datapool item of type Integer and rename it to 1\_diff.

#### Step 4

Add another datapool item of type Integer and rename it to 2t\_anim.

#### Step 5

**Select** RedRectangle.

#### Step 6

In the **Properties** component, go to the **User-defined properties** category, click + and add a property of type conditional script.

Conditional script 2 is added.

#### Step 7

**Rename** Conditional script 2 to StartRedAnimation.

#### Step 8

**Next to** StartRedAnimation click {}.

The EB GUIDE Script editor opens.

#### Step 9

Enter the following script:

```
function(v:arg0::bool)
{
    f:animation_play(v:this->^->"MoveAnimation")
}
```

#### Step 10

In the **Navigation** component, select `FirstView`.

#### Step 11

In the **Animation editor**, next to the **Animated properties** click **+** and select `FirstView`.

The **Animation properties** dialog opens.

#### Step 12

Under `RedRectangle`, select the `y` property and then the `Script curve`.

#### Step 13

Click **Accept**.

`Script curve 2` is added to the **Animation editor**.

#### Step 14

Rename `Script curve 2` to `RedCurve`.

#### Step 15

In the **Properties** component, next to the `curve` property, click **{ }**.

The EB GUIDE Script editor opens.

#### Step 16

Enter the following script:

```
function(v:diff::int, v:t_anim::int)
{
    dp:"1_diff"=v:diff
    dp:"2t_anim"=v:t_anim
    v:t_anim/2::int
}
```



### Saving and testing the EB GUIDE model

Prerequisite:

- You completed the previous instruction.

#### Step 1

To save the project, click  in the command area.

### Step 2

To start the simulation, click ▶ in the command area.

The animation is played at the start of the simulation.

## 14.11. Tutorial: Creating a horizontal progress bar

### TIP



### Default window layout

All instructions and screenshots use the default window layout. If you want to follow the instructions, we recommend to set the EB GUIDE Studio or EB GUIDE Monitor window to default layout by selecting **Layout > Reset to default layout**.

The following instructions guide you through the process of modeling a progress bar as shown below.

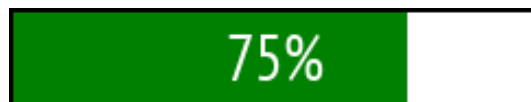


Figure 14.11. Progress bar

You can also have a look at the progress bar template in the widget template library. See <https://www.elektrobit.com/ebguide/examples/>.

Approximate duration: 10 minutes



### Adding the widgets

The following instructions guide you through the process of adding widgets for the progress bar.

Prerequisite:

- The **Main** state machine contains an Initial state and a View state.
- The Initial state has a transition to the View state.
- The content area displays a View.

### Step 1

In the **Templates** component, click + and then select **Container**.

A template is created that contains a Container.

### Step 2

Rename the template to `T_ProgressBar`.



#### Step 3

Rename the Container to `ProgressBar_Container`.

#### Step 4

Drag a Rectangle into the Container and rename it to `Background_Rectangle`.

#### Step 5

Drag another Rectangle into the Container and rename it to `Progress_Rectangle`.

This Rectangle visualizes the progress of the operation.

#### Step 6

Drag a Label into the Container and rename it to `Percentage_Text`.



#### Entering the properties for the progress bar

The following instructions guide you through the process of configuring the properties and adding scripts to the widgets.

Prerequisite:



- You completed the previous instruction.

#### Step 1

In the **Templates** component, select `ProgressBar_Container`.

#### Step 2

Add the properties `width`, `height`, `x`, `y` to the template interface.

To add a property to the template interface, in the **Properties** component, click the  button next to the property. In the menu, click **Add to template interface**. The icon  is displayed next to the property.

#### Step 3

In the **Properties** component, go to the **User-defined properties** category and click  and select `Integer`.

A user-defined property of type `Integer` is added to the Container.

#### Step 4

Rename the property to `progress`.

#### Step 5

Add `progress` to the template interface.

#### Step 6

In the **Templates** component, select `Background_Rectangle`.

#### Step 7

Link the `width` to the width property of `ProgressBar_Container`.

To link a property to another property, in **Properties** component, click the  button next to the property. In the menu, click **Add link to widget property**.

A dialog opens.

#### Step 8

In the dialog, select the `width` property of `ProgressBar_Container` and click **Accept**.

#### Step 9

Link the `height` property of `Background_Rectangle` to the `height` property of `ProgressBar_Container`.

#### Step 10

In the **Templates** component, select `Progress_Rectangle`.


#### Step 11

Link the `height` property to the `height` property of `ProgressBar_Container`.

#### Step 12

Set the `fillColor` to green.

#### Step 13

Next to the `width` property click  and then select `Convert to script`.

The `width` property defines the width as a percentage of the width of `ProgressBar_Container`.

#### Step 14

Click `{ }`.

An EB GUIDE Script editor opens.

#### Step 15

Enter the following EB GUIDE Script in the **Read** section:

```
function()  
{  
  v:this->^.width * v:this->^.progress / 100  
}
```

This script divides the value of the `progress` property by 100.

#### Step 16

Click **Add available triggers to list**.

Two triggers for `width` and `progress` are added.

#### Step 17

In the **Templates** component, select `Percentage_Text`.

#### Step 18

Link the `width` and `height` properties to the `width` and `height` of `ProgressBar_Container`.

#### Step 19

Set the `horizontalAlign` to `center (1)`.

#### Step 20

Convert the `text` property into a script.

The text will display the percentage of the width of the container.

#### Step 21

Click `{ }`.

An EB GUIDE Script editor opens.

#### Step 22

In the **Read** section, enter the following script:

```
function()  
{  
    f:int2string(v:this->^.progress) + "%"  
}
```

This script converts the percentage value into a string and adds the % character after the percentage number.

#### Step 23

Click **Add available triggers to list**.

The trigger for `progress` is added.

#### Step 24

Set the `x` and `y` properties of all widgets in the template to 0.

#### Step 25

In the **Navigation** component, double-click the View.

#### Step 26

From the **Toolbox** component, drag `T_ProgressBar` into the content area.

The template is added to the View. Now you can add an Animation to it to show the dynamic progress of an operation.



### Animating the progress

The following instructions guide you through the process of animating the progress bar, so that you can see better what happens when you change the percentage value.

Prerequisite:

- You completed the previous instruction.
- The content area displays the View.

#### Step 1

Drag an Animation into the View.

#### Step 2

Rename the Animation to `Loading_Animation`.

#### Step 3

In the **Properties** component, go to the **User-defined properties** category, click **+**, and select **Conditional script**.

#### Step 4

Rename the conditional script to `animateProgress`.

#### Step 5

Next to the conditional script property click `{ }`.

An EB GUIDE Script editor opens.

#### Step 6

In the **On trigger** section, enter the following script:

```
function(v:arg0::bool)
{
  f:animation_play(v:this)
  false
}
```

#### Step 7

In the **Navigation** component, double-click `Loading_Animation` to open the **Animation editor**.

#### Step 8

In the **Animation editor**, next to **Animated properties** click **+** and select `View 1`.

The **Animation properties** dialog opens.

#### Step 9

Under `T_ProgressBar 1`, select the `progress` property and then `Linear interpolation curve`. Click **Accept**.

#### Step 10

In the **Properties** component, set `end` property to `100`.

The progress animation will stop when the progress indicator reaches 100%.



### Saving and testing the EB GUIDE model

Prerequisite:

- You completed the previous instruction.

#### Step 1

To save the project, click  in the command area.

#### Step 2

To start the simulation, click  in the command area.

## 15. References

The following chapter provides you with lists and tables for example parameters, properties, and identifiers.

For EB GUIDE GTF specific parameters, properties, and identifiers, see EB GUIDE GTF user guide.

### 15.1. Command line options

#### 15.1.1. Command line options for `Studio.Console.exe`

The following table lists command line options available in EB GUIDE Studio for `Studio.Console.exe` and explains their meaning. Undefined command line options do not prompt error messages.

The general syntax of a command line is as follows:

```
Studio.Console.exe <option> "project_name.ebguide"
```

Table 15.1. Command line options for `Studio.Console.exe`

Option	Description
<code>-c &lt;logfile dir&gt;</code>	Validates an EB GUIDE model and writes a logfile to the directory specified as <code>logfile dir</code>
<code>-e &lt;destination dir&gt;</code>	Exports an EB GUIDE model to the destination directory <code>destination dir</code>  Use with the command line option <code>-p</code> , see an example below.
<code>-h</code>	Shows the help message
<code>-l &lt;language file&gt;</code>	Imports one language file that is saved as <code>language file (.xliff)</code> into an EB GUIDE model and creates a logfile
<code>-m</code>	Allows the migration of the project
<code>-o</code>	Opens the project file
<code>-p &lt;profile&gt;</code>	Uses the profile specified as <code>profile</code> during the export
<code>-s &lt;skin_set&gt;</code>	Defines the skin set that defines which skins are exported. If you do not select an export set, all of the skins in the EB GUIDE model are exported.
<code>-t &lt;language_set&gt;</code>	Defines the language set that defines which languages are exported. If you do not select an export set, all of the languages in the EB GUIDE model are exported.



### Example 15.1. Command line options

The command line `Studio.Console.exe -e "C:/temp/exported_project" -p "target_profile" -o "project_name.ebguide"` exports `project_name.ebguide` by using the profile `target_profile` to the specified destination directory `C:/temp/exported_project`.

For instructions, see the following:

- ▶ [section 10.4.1.2, “Validating an EB GUIDE model using command line”](#)
- ▶ [section 10.5.2, “Exporting an EB GUIDE model using command line”](#)
- ▶ [section 10.8.2.2, “Importing language-dependent texts using command line”](#)

## 15.1.2. Command line options for `Monitor.Console.exe`

The following table lists command line options available in EB GUIDE Monitor for `Monitor.Console.exe` and explains their meaning. Undefined command line options do not prompt error messages.

The general syntax of a command line is as follows:

```
Monitor.Console.exe <option> "monitor.cfg"
```

Table 15.2. Command line options for `Monitor.Console.exe`

Option	Description
<code>-c &lt;host:port&gt;</code>	Connects an EB GUIDE model to a running EB GUIDE GTF process
<code>-h</code>	Shows the help message
<code>-l &lt;language&gt;</code>	Sets the language of EB GUIDE Monitor to one of the following: <code>en</code> for English, <code>ja</code> for Japanese, <code>ko</code> for Korean, <code>zh-cn</code> for Chinese (Simplified).
<code>-o</code>	Opens the configuration file <code>monitor.cfg</code>
<code>-s</code>	Executes all methods in a defined script



### Example 15.2. Command line options

The command line `Monitor.Console.exe -l ko` sets the language of EB GUIDE Monitor to Korean.

For instructions on how to use EB GUIDE Monitor, see [chapter 11, “Working with EB GUIDE Monitor”](#).

## 15.2. Datapool items

Table 15.3. Properties of a datapool item

Property name	Description
Value	The initial value of the datapool item

## 15.3. Data types

The following section describes data types in EB GUIDE. You can add user-defined properties and datapool items from the types listed below.

### 15.3.1. Boolean

Boolean properties can have the values true and false.

Available operations are as follows:

- ▶ equal (==)
- ▶ not equal (!=)
- ▶ negation (!)
- ▶ and (&&)
- ▶ or (||)
- ▶ assign (writable properties) (=)

It is possible to store boolean properties in a list. For details about lists, see [section 15.3.12, “List”](#).

### 15.3.2. Color

Colors are stored in the RGBA8888 format.

Example: Red without transparency is (255, 0, 0, 255).

Available operations are as follows:

- ▶ equal (==)
- ▶ not equal (!=)
- ▶ assign (writable properties) (=)

It is possible to store color properties in a list. For details about lists, see [section 15.3.12, “List”](#).

### 15.3.3. Conditional script

Conditional scripts are used to react on initialization and on trigger. When you edit conditional scripts, the content area is divided into the following sections:

- ▶ In the **Triggers** section, you can select an event, datapool item, or widget property that triggers the execution of the **On trigger** script.
- ▶ In the **On trigger** section, you can add an EB GUIDE Script that is called on initialization, an event trigger, or after a value update of a datapool item or a widget property.

The parameter of the **On trigger** EB GUIDE Script indicates the cause for the execution of the script.

The `arg0` refers to the fact whether the EB GUIDE Script is executed during initialization or by a trigger. Consider the following:

- ▶ If the EB GUIDE Script is executed during initialization, `arg0` is `true`.
- ▶ If the EB GUIDE Script is executed by a trigger, `arg0` is `false`.

The return value of the **On trigger** EB GUIDE Script controls change notifications for the property.

The return value of the **On trigger** EB GUIDE Script regulates whether the EB GUIDE Script must produce a notification or not. Consider the following:

- ▶ If the return value is `true`, a notification is generated.
- ▶ If the return value is `false`, a notification is not generated.

To be able to execute the **On trigger** script, the conditions are to be fulfilled during the following:

- ▶ On initialization, for example, in case of datapool items during EB GUIDE model start-up, or in case of widget properties during the view creation.
- ▶ On processing an event from the trigger script. The EB GUIDE Script is executed once for each matching event.
- ▶ On processing the datapool notifications of one or more items from the trigger script. Multiple notifications may be processed at once.
- ▶ On processing the notifications of one or more widget properties from the trigger script. Multiple notifications may be processed at once.

### 15.3.4. Float

Float-point number data type represents a single-precision 32-bit IEEE 754 value.



Available operations are as follows:

- ▶ equal (==)
- ▶ not equal (!=)
- ▶ greater (>)
- ▶ greater or equal (>=)
- ▶ less (<)
- ▶ less or equal (<=)
- ▶ addition (+)
- ▶ subtraction (-)
- ▶ multiplication (\*)
- ▶ division (/)
- ▶ assign (writable properties) (=)

It is possible to store float properties in a list. For details about lists, see [section 15.3.12, “List”](#).

### 15.3.5. Font

To add a font to an EB GUIDE project, copy the font file to: `$GUIDE_PROJECT_PATH/<project name>/resources`

Available operations are as follows:

- ▶ assign (writable properties) (=)

It is possible to store font properties in a list. For details about lists, see [section 15.3.12, “List”](#).

### 15.3.6. Function () : bool

By means of `Function () : bool` you can create an own function.

The available operation for this data type is a read/run operation for all properties.

### 15.3.7. Ibl

Ibl is a data format that stores lighting information generated by the IBLGenerator.

To add an ibl to an EB GUIDE project, copy the `.ebibl` file to: `$GUIDE_PROJECT_PATH/<project name>/resources`

Available operations are as follows:

- ▶ assign (writable properties) (=)

It is possible to store ibl properties in a list. For details about lists, see [section 15.3.12, “List”](#).

## 15.3.8. Image

To add an image to an EB GUIDE project, copy the image file to: `$GUIDE_PROJECT_PATH/<project name>/resources`

Available operations are as follows:

- ▶ assign (writable properties) (=)

It is possible to store image properties in a list. For details about lists, see [section 15.3.12, “List”](#).

## 15.3.9. Integer

EB GUIDE supports signed 32-bit integers.

Available operations are as follows:

- ▶ equal (==)
- ▶ not equal (!=)
- ▶ greater (>)
- ▶ greater or equal (>=)
- ▶ less (<)
- ▶ less or equal (<=)
- ▶ addition (+)
- ▶ subtraction (-)
- ▶ multiplication (\*)
- ▶ division (/)
- ▶ modulo (%)
- ▶ assign (writable properties) (=)

It is possible to store integer properties in a list. For details about lists, see [section 15.3.12, “List”](#).

## 15.3.10. Mesh

Mesh defines the shape of the 3D object.

To add a mesh to an EB GUIDE project, copy the `.ebmesh` file to: `$GUIDE_PROJECT_PATH/<project name>/resources`

Available operations are as follows:

- ▶ assign (writable properties) (=)

It is possible to store mesh properties in a list. For details about lists, see [section 15.3.12, “List”](#).

## 15.3.11. String

EB GUIDE supports character strings, for example *Hello world*.

Available operations are as follows:

- ▶ equal (case sensitive) (==)
- ▶ not equal (case sensitive) (!=)
- ▶ equal (case insensitive, only in the ASCII range) (=Aa=)
- ▶ greater (>)
- ▶ greater or equal (>=)
- ▶ less (<)
- ▶ less or equal (<=)
- ▶ concatenation (+)
- ▶ assign (writable properties) (=)

It is possible to store string properties in a list. For details about lists, see [section 15.3.12, “List”](#).

## 15.3.12. List

EB GUIDE supports a list of values with the same data type.

The following list types are available:

- ▶ Boolean list
- ▶ Color list
- ▶ Float list

- ▶ Font list
- ▶ lbl list
- ▶ Image list
- ▶ Integer list
- ▶ Mesh list
- ▶ String list

The following types cannot be used in lists:

- ▶ List
- ▶ Property reference
- ▶ List element reference

Available operations are as follows:

- ▶ length: (length)
- ▶ element accessor: ([ ])

## 15.4. EB GUIDE Script

### 15.4.1. EB GUIDE Script keywords

The following is a list of reserved keywords in EB GUIDE Script. If you want to use these words as identifiers in a script, you must quote them.

Table 15.4. EB GUIDE Script keywords

Keyword	Description
cancel_fire	Cancels an event that is fired with <code>fire_delayed</code>
color:	A color parameter follows, for example {0,255,255}
dp:	A datapool item follows
else	An <code>if</code> condition is completed. The following block is executed as an alternative.
ev:	An event follows
f:	A user-defined function follows
false	A boolean literal value
fire	Fires an event

Keyword	Description
<code>fire_delayed</code>	Fires an event after a specified time. The time is specified in milliseconds.
<code>function</code>	Declares a function
<code>if</code>	A statement which tests a boolean expression follows. If the expression is true, the statement is executed.
<code>in</code>	Is a separator between a local variable declaration and the variable's scope of usage  Is used with <code>match_event</code> and <code>let</code> .
<code>l:</code>	A language follows. Is used on <code>f:setLanguage(l:English, true)</code> .
<code>length</code>	The length of a property
<code>let</code>	Declares a local variable that is accessible in the scope
<code>list</code>	Declares a type list, for example an integer list
<code>match_event</code>	Checks if the current event corresponds to an expected event and declares variables like <code>let</code>
<code>popup_stack</code>	The dynamic state machine list which defines the priority of dynamic state machines
<code>s:</code>	A skin follows. Is used on <code>f:setSkin(s:mySkin, true)</code> .
<code>sm:</code>	A state machine follows
<code>true</code>	A boolean literal value
<code>unit</code>	A value of type void
<code>v:</code>	A local variable follows
<code>while</code>	Repeats a statement as long as the condition is true

## 15.4.2. EB GUIDE Script operator precedence

The following is a list of the operators in EB GUIDE Script together with their precedence and associativity. Operators are listed top to bottom, in descending precedence.

Table 15.5. EB GUIDE Script operator precedence

Operator	Associativity
<code>(( )), ({ })</code>	none
<code>[ ]</code>	none
<code>(-&gt;)</code>	left
<code>(.)</code>	none

Operator	Associativity
::	left
length	none
(&)	right
(!), (-) unary minus	right
(*), (/), (%)	left
(+), (-)	left
(<), (>), (<=), (>=)	left
(!=), (==), (=Aa=)	left
(&&)	left
(  )	left
(=), (+=), (-=), (=>)	right
(,)	right
(;)	left

### 15.4.3. EB GUIDE Script standard library

The following chapter provides a description of all EB GUIDE Script functions.

#### 15.4.3.1. EB GUIDE Script functions A - B

##### 15.4.3.1.1. `abs`

The function returns the absolute value of the integer number `x`.

Table 15.6. Parameters of `abs`

Parameter	Type	Description
<code>x</code>	integer	The number to return the absolute value from
<code>&lt;return&gt;</code>	integer	The return value

##### 15.4.3.1.2. `absf`

The function returns the absolute value of the float number `x`.

Table 15.7. Parameters of `absf`

Parameter	Type	Description
<code>x</code>	float	The number to return the absolute value from
<code>&lt;return&gt;</code>	float	The return value

#### 15.4.3.1.3. `acosf`

The function returns the principal value of the arc cosine of `x`.

Table 15.8. Parameters of `acosf`

Parameter	Type	Description
<code>x</code>	float	The number to return the arc cosine from
<code>&lt;return&gt;</code>	float	The return value

#### 15.4.3.1.4. `animation_before`

The function checks if a running animation has passed a given point in time.

Table 15.9. Parameters of `animation_before`

Parameter	Type	Description
<code>animation</code>	GtfTypeRecord	The animation to manipulate
<code>time</code>	integer	The point in time
<code>&lt;return&gt;</code>	boolean	If true, the animation has not yet passed the point in time.

#### 15.4.3.1.5. `animation_beyond`

The function checks if a running animation has passed a given point in time.

Table 15.10. Parameters of `animation_beyond`

Parameter	Type	Description
<code>animation</code>	GtfTypeRecord	The animation to manipulate
<code>time</code>	integer	The point in time
<code>&lt;return&gt;</code>	boolean	If true, the animation has passed the point in time.

#### 15.4.3.1.6. `animation_cancel`

The function cancels an animation and leaves edited properties in the current state.

Table 15.11. Parameters of `animation_cancel`

Parameter	Type	Description
<code>animation</code>	<code>GtfTypeRecord</code>	The animation to manipulate
<code>&lt;return&gt;</code>	<code>boolean</code>	If true, the function succeeded.

#### 15.4.3.1.7. `animation_cancel_end`

The function cancels an animation and sets edited properties to the end state where possible.

Table 15.12. Parameters of `animation_cancel_end`

Parameter	Type	Description
<code>animation</code>	<code>GtfTypeRecord</code>	The animation to manipulate
<code>&lt;return&gt;</code>	<code>boolean</code>	If true, the function succeeded.

#### 15.4.3.1.8. `animation_cancel_reset`

The function cancels an animation and resets edited properties to the initial state where possible.

Table 15.13. Parameters of `animation_cancel_reset`

Parameter	Type	Description
<code>animation</code>	<code>GtfTypeRecord</code>	The animation to manipulate
<code>&lt;return&gt;</code>	<code>boolean</code>	If true, the function succeeded.

#### 15.4.3.1.9. `animation_pause`

The function pauses an animation.

Table 15.14. Parameters of `animation_pause`

Parameter	Type	Description
<code>animation</code>	<code>GtfTypeRecord</code>	The animation to manipulate
<code>&lt;return&gt;</code>	<code>boolean</code>	If true, the function succeeded.

#### 15.4.3.1.10. `animation_play`

The function starts or continues an animation.



Table 15.15. Parameters of `animation_play`

Parameter	Type	Description
<code>animation</code>	<code>GtfTypeRecord</code>	The animation to manipulate
<code>&lt;return&gt;</code>	<code>boolean</code>	If true, the animation is not running yet.

#### 15.4.3.1.11. `animation_reverse`

The function plays an animation backwards.

Table 15.16. Parameters of `animation_reverse`

Parameter	Type	Description
<code>animation</code>	<code>GtfTypeRecord</code>	The animation to manipulate
<code>&lt;return&gt;</code>	<code>boolean</code>	If true, the animation is not running yet.

#### 15.4.3.1.12. `animation_running`

The function checks if an animation is currently running.

Table 15.17. Parameters of `animation_running`

Parameter	Type	Description
<code>animation</code>	<code>GtfTypeRecord</code>	The animation to manipulate
<code>&lt;return&gt;</code>	<code>boolean</code>	If true, the animation is running.

#### 15.4.3.1.13. `animation_set_time`

The function sets the current time of an animation, can be used to skip or replay an animation.

Table 15.18. Parameters of `animation_set_time`

Parameter	Type	Description
<code>animation</code>	<code>GtfTypeRecord</code>	The animation to manipulate
<code>time</code>	<code>integer</code>	time
<code>&lt;return&gt;</code>	<code>boolean</code>	If true, the function succeeded.

#### 15.4.3.1.14. `asinf`

The functions calculates the principal value of the arc sine of  $x$ .

Table 15.19. Parameters of `asinf`

Parameter	Type	Description
<code>x</code>	float	The number to return the arc sine from
<code>&lt;return&gt;</code>	float	The return value

#### 15.4.3.1.15. `atan2f`

The function calculates the principal value of the arc tangent of  $y/x$ , using the signs of the two arguments to determine the quadrant of the result.

Table 15.20. Parameters of `atan2f`

Parameter	Type	Description
<code>y</code>	float	Argument <code>y</code>
<code>x</code>	float	Argument <code>x</code>
<code>&lt;return&gt;</code>	float	The return value

#### 15.4.3.1.16. `atan2i`

The function calculates the principal value of the arc tangent of  $y/x$ , using the signs of the two arguments to determine the quadrant of the result.

Table 15.21. Parameters of `atan2i`

Parameter	Type	Description
<code>y</code>	integer	Argument <code>y</code>
<code>x</code>	integer	Argument <code>x</code>
<code>&lt;return&gt;</code>	float	The return value

#### 15.4.3.1.17. `atanf`

The function calculates the principal value of the arc tangent of `x`.

Table 15.22. Parameters of `atanf`

Parameter	Type	Description
<code>x</code>	float	The number to return the arc tangent from
<code>&lt;return&gt;</code>	float	The return value

#### 15.4.3.1.18. bool2string

The function converts a boolean variable to either the string `true` or `false`.

Table 15.23. Parameters of `bool2string`

Parameter	Type	Description
<code>x</code>	boolean	The value to convert to a string
<code>&lt;return&gt;</code>	string	<code>true</code> in case <code>x</code> was true, and <code>false</code> otherwise

### 15.4.3.2. EB GUIDE Script functions C - H

#### 15.4.3.2.1. ceil

The function returns the smallest integral value that is not less than the argument.

Table 15.24. Parameters of `ceil`

Parameter	Type	Description
<code>value</code>	float	The value to round
<code>&lt;return&gt;</code>	integer	The rounded value

#### 15.4.3.2.2. changeDynamicStateMachinePriority

The function changes the priority of a dynamic state machine.

Table 15.25. Parameters of `changeDynamicStateMachinePriority`

Parameter	Type	Description
<code>stack</code>	Popupstack ID	The dynamic state machine list
<code>sm</code>	State machine ID	The dynamic state machine
<code>priority</code>	integer	The priority of the dynamic state machine in the list. Note that a higher number means a higher priority.

#### 15.4.3.2.3. character2unicode

The function returns the Unicode value of the first character in a string.

Table 15.26. Parameters of `character2unicode`

Parameter	Type	Description
<code>str</code>	string	The input string

Parameter	Type	Description
<return>	integer	The character as Unicode value  0 in case of errors

#### 15.4.3.2.4. clampf

The function clamps a floating-point value to a defined range [xmin, xmax], this means the function computes `max (xmin, min (xmax, x))`.

Table 15.27. Parameters of clampf

Parameter	Type	Description
x	float	The value to clamp
xmin	float	The minimum range
xmax	float	The maximum range
<return>	float	The x value clamped to the [xmin, xmax] range

#### 15.4.3.2.5. clampi

The function clamps an integer value to a defined range [xmin, xmax], this means the function computes `max (xmin, min (xmax, x))`.

Table 15.28. Parameters of clampi

Parameter	Type	Description
x	int	The value to clamp
xmin	int	The minimum range
xmax	int	The maximum range
<return>	int	The x value clamped to the [xmin, xmax] range

#### 15.4.3.2.6. clearAllDynamicStateMachines

The function removes all dynamic state machines from the dynamic state machine list.

Table 15.29. Parameters of clearAllDynamicStateMachines

Parameter	Type	Description
state		The state with the dynamic state machine list

#### 15.4.3.2.7. `color2string`

The function converts a color to eight hexadecimal values.

Table 15.30. Parameters of `color2string`

Parameter	Type	Description
<code>value</code>	color	The color to convert to string
<code>&lt;return&gt;</code>	string	The color formatted as a string of hexadecimal digits with # as prefix

#### NOTE



#### Formatting examples

The format of the returned string is `#RRGGBBAA` with two digits for each of the color channels red, green, blue and alpha.

For example, opaque pure red is converted to `#ff0000ff`, semi-transparent pure green is converted to `#00ff007f`.

#### 15.4.3.2.8. `cosf`

The function returns the cosine of `x`, where `x` is given in radians.

Table 15.31. Parameters of `cosf`

Parameter	Type	Description
<code>x</code>	float	The number to return the cosine from
<code>&lt;return&gt;</code>	float	The return value

#### 15.4.3.2.9. `deg2rad`

The function converts an angle from degrees to radians.

Table 15.32. Parameters of `deg2rad`

Parameter	Type	Description
<code>x</code>	float	The angle to convert from degrees to radians
<code>&lt;return&gt;</code>	float	The return value

#### 15.4.3.2.10. `expf`

The function returns the value of `e`, the base of natural logarithms, raised to the power of `x`.

Table 15.33. Parameters of `expf`

Parameter	Type	Description
<code>x</code>	float	The exponent
<return>	float	The return value

#### 15.4.3.2.11. `float2string`

The function converts simple float to string.

Table 15.34. Parameters of `float2string`

Parameter	Type	Description
<code>value</code>	float	The value to convert to string
<return>	string	The float value, formatted as string

#### 15.4.3.2.12. `floor`

The function returns the largest integral value not greater than the parameter value.

Table 15.35. Parameters of `floor`

Parameter	Type	Description
<code>value</code>	float	The value to round
<return>	integer	The rounded value

#### 15.4.3.2.13. `fmod`

The function computes the remainder of the floating-point division  $x/y$ .

Table 15.36. Parameters of `fmod`

Parameter	Type	Description
<code>x</code>	float	The floating point numerator
<code>y</code>	float	The floating point denominator
<return>	float	The remainder of the division $x/y$

#### 15.4.3.2.14. `focusMoveTo`

The function forces the focus manager to forward the focus to a dedicated focusable element.

Table 15.37. Parameters of `focusMoveTo`

Parameter	Type	Description
<code>widget</code>	<code>widget</code>	The widget on which the focus is moved.
<code>&lt;return&gt;</code>	<code>void</code>	

#### 15.4.3.2.15. `focusNext`

The function forces the focus manager to forward the focus to the next focusable element.

Table 15.38. Parameters of `focusNext`

Parameter	Type	Description
<code>&lt;return&gt;</code>	<code>void</code>	

#### 15.4.3.2.16. `focusPrevious`

The function forces the focus manager to return the focus to the previous focusable element.

Table 15.39. Parameters of `focusPrevious`

Parameter	Type	Description
<code>&lt;return&gt;</code>	<code>void</code>	

#### 15.4.3.2.17. `format_float`

The function formats a float value.

Table 15.40. Parameters of `format_float`

Parameter	Type	Description
<code>format</code>	<code>string</code>	<p>A string of the following structure:</p> <p><code>%[flags] [width] [.precision] type</code></p> <ul style="list-style-type: none"><li>▶ <b>flags:</b> Optional character or characters that control output justification and output of signs, blanks, leading zeros, decimal points, and octal and hexadecimal prefixes.</li><li>▶ <b>width:</b> Optional decimal number that specifies the minimum number of characters that are output.</li><li>▶ <b>precision:</b> Optional decimal number that specifies the number of significant digits or the number of digits after the decimal-point character .</li></ul>

Parameter	Type	Description
		<ul style="list-style-type: none"> <li>▶ type: Required conversion specifier character that determines whether the associated argument is interpreted as a character, a string, an integer, or a float number.</li> </ul>
useDotAsDelimiter	boolean	<p>Defines the delimiter sign.</p> <p>Possible values:</p> <ul style="list-style-type: none"> <li>▶ true: Use a dot as delimiter.</li> <li>▶ false: Use a comma as delimiter.</li> </ul>
value	float	The number to format

### WARNING



#### Adhere to `printf` specification for C++

The `format` parameter is defined according to the `printf` specification for C++.

Using values that do not comply with this specification can lead to unexpected behavior.

For example, allowed types for `format_float` are `f`, `a`, `g` and `e`, and not more than one type character is allowed.

#### 15.4.3.2.18. `format_int`

The function formats an integer value.

Table 15.41. Parameters of `format_int`

Parameter	Type	Description
format	string	<p>A string of the following structure:</p> <p>%[flags] [width] [.precision] type</p> <ul style="list-style-type: none"> <li>▶ flags: Optional character or characters that control output justification and output of signs, blanks, leading zeros, decimal points, and octal and hexadecimal prefixes.</li> <li>▶ width: Optional decimal number that specifies the minimum number of characters that are output.</li> <li>▶ precision: Optional decimal number that specifies the minimum number of digits that are printed.</li> <li>▶ type: Required conversion specifier character that determines whether the associated argument is interpreted as a character, a string, an integer, or a float number.</li> </ul>
value	int	The number to format



---

**WARNING** **Adhere to `printf` specification for C++**

The `format` parameter is defined according to the `printf` specification for C++.

Using values that do not comply with this specification can lead to unexpected behavior.

For example, allowed types for `format_int` are `d`, `i`, `o`, `x` and `u`, and not more than one type character is allowed.

---

#### 15.4.3.2.19. `frac`

The function computes the fractional part of a floating-point value. The return value lies in the interval `[0, 1]`. For example, the function returns 0.5 for the parameter value `x=1.5` or `x=-1.5`.

Table 15.42. Parameters of `frac`

Parameter	Type	Description
<code>x</code>	<code>float</code>	The floating point value
<code>&lt;return&gt;</code>	<code>float</code>	The fractional part of the floating-point value.

#### 15.4.3.2.20. `getAllLanguages`

The function fills a datapool item with a list of language UIDs from all models or for the current model only.

Table 15.43. Parameters of `getAllLanguages`

Parameter	Type	Description
<code>itemId</code>	<code>dp_id</code>	The datapool item ID where the language UIDs are stored. The datapool item type must be string list.
<code>isCoreScope</code>	<code>boolean</code>	Specifies the scope.  Possible values: <ul style="list-style-type: none"><li>▶ <code>true</code>: Applies to all models</li><li>▶ <code>false</code>: Applies only to the current model</li></ul>
<code>&lt;return&gt;</code>	<code>void</code>	

#### 15.4.3.2.21. `getAllSkins`

The function fills a datapool item with a list of skin UIDs from the core or model scope.

Table 15.44. Parameters of `getAllSkins`

Parameter	Type	Description
<code>itemId</code>	<code>dp_id</code>	The datapool item ID where the skin UIDs are stored. The datapool item type must be string list.
<code>isCoreScope</code>	<code>boolean</code>	Specifies the scope. <ul style="list-style-type: none"><li>▶ <code>true</code>: Applies to all models</li><li>▶ <code>false</code>: Applies only to the current model</li></ul>
<code>&lt;return&gt;</code>	<code>void</code>	

#### 15.4.3.2.22. `getConfigItem`

The function fills a datapool item with a configuration item value.

Table 15.45. Parameters of `getConfigItem`

Parameter	Type	Description
<code>itemId</code>	<code>dp_id</code>	The datapool ID where the configuration item is to be stored
<code>name</code>	<code>string</code>	The configuration item name
<code>&lt;return&gt;</code>	<code>boolean</code>	True if datapool item is successfully filled with a configuration item value

#### 15.4.3.2.23. `getFontAscender`

The function returns the ascender of the font passed as parameter.

Table 15.46. Parameters of `getFontAscender`

Parameter	Type	Description
<code>x</code>	<code>font</code>	The font to be evaluated  Note that if you have the multifont support added, only the default font is evaluated.
<code>&lt;return&gt;</code>	<code>integer</code>	The ascender of the font

#### 15.4.3.2.24. `getFontDescender`

The function returns the descender of the font passed as parameter.

Table 15.47. Parameters of `getFontDescender`

Parameter	Type	Description
x	font	The font to be evaluated  Note that if you have the multifont support added, only the default font is evaluated.
<return>	integer	The descender of the font

#### 15.4.3.2.25. `getFontLineGap`

The function returns the line gap of the font passed as parameter.

Table 15.48. Parameters of `getFontLineGap`

Parameter	Type	Description
x	font	The font to be evaluated  Note that if you have the multifont support added, only the default font is evaluated.
<return>	integer	The line gap of the font

#### 15.4.3.2.26. `getImageHeight`

The function returns the height in pixels of an image passed as parameter.

Table 15.49. Parameters of `getImageHeight`

Parameter	Type	Description
x	Image widget	The widget to evaluate
<return>	integer	The height in pixels of an image

#### 15.4.3.2.27. `getImageWidth`

The function returns the width in pixels of an image passed as parameter.

Table 15.50. Parameters of `getImageWidth`

Parameter	Type	Description
x	Image widget	The widget to be evaluated
<return>	integer	The width in pixels of an image

#### 15.4.3.2.28. `getLabelTextHeight`

The function returns the total height in pixels of a label's text. The total height is calculated using the formula:

$$\text{total\_height} = \text{line\_height} * \text{line\_count} + \text{line\_spacing} * (\text{line\_count} - 1)$$

The `line_spacing` is calculated as the sum of the font `lineGap` property and the `lineOffset` property of the **Multiple lines** widget feature. Both font `lineGap` and the `lineOffset` property can be negative.

Table 15.51. Parameters of `getLabelTextHeight`

Parameter	Type	Description
<code>widget</code>	Label widget	The widget to be evaluated
<code>&lt;return&gt;</code>	integer	The height in pixels of the text

#### 15.4.3.2.29. `getLabelTextWidth`

The function returns the width of the longest line of a label's text.

Table 15.52. Parameters of `getLabelTextWidth`

Parameter	Type	Description
<code>widget</code>	Label widget	The widget to evaluate
<code>&lt;return&gt;</code>	integer	The width in pixels of the longest line of the text

#### 15.4.3.2.30. `getLanguage`

The function returns the current language from all models or from the current model only.

Table 15.53. Parameters of `getLanguage`

Parameter	Type	Description
<code>isCoreScope</code>	boolean	Specifies the scope. <ul style="list-style-type: none"><li>▶ <code>true</code>: Applies to all models</li><li>▶ <code>false</code>: Applies only to the current model</li></ul>
<code>&lt;return&gt;</code>	string	The UID of the language.

#### 15.4.3.2.31. `getLanguageName`

The function returns the name of the specified language UID.

Table 15.54. Parameters of `getLanguageName`

Parameter	Type	Description
<code>languageUid</code>	string	Language for which the name is requested.
<code>&lt;return&gt;</code>	string	The name of the language.

#### 15.4.3.2.32. `getLanguageTag`

The function returns the tag of the specified language UID. The language tag consists of a language code that follows the ISO 639 standards for representing language names and a language region that follows ISO 3166-1 standards for representing country codes. For example, the `en-US` language tag means `English` language in region `United States`.

Table 15.55. Parameters of `getLanguageTag`

Parameter	Type	Description
<code>languageUid</code>	string	Language for which the tag is requested.
<code>&lt;return&gt;</code>	string	The tag of the language.

#### 15.4.3.2.33. `getLineCount`

The function returns the number of lines of a label's text.

Table 15.56. Parameters of `getLineCount`

Parameter	Type	Description
<code>widget</code>	Label widget	The widget to be evaluated
<code>&lt;return&gt;</code>	integer	The number of lines of the text

#### 15.4.3.2.34. `getLineHeight`

The function returns the height of a line written with the font passed as parameter.

Table 15.57. Parameters of `getLineHeight`

Parameter	Type	Description
<code>x</code>	font	The font to be evaluated  Note that if you have the multifont support added, only the default font is evaluated.

Parameter	Type	Description
<return>	integer	The height of a line written with the specified font

#### 15.4.3.2.35. getProductString

The function returns a string with the product name of EB GUIDE GTF.

Table 15.58. Parameters of `getProductString`

Parameter	Type	Description
<return>	string	The product name

#### 15.4.3.2.36. getSkin

The function returns the current skin from all models or from the current model only.

Table 15.59. Parameters of `getSkin`

Parameter	Type	Description
<code>isCoreScope</code>	boolean	Specifies the scope. <ul style="list-style-type: none"><li>▶ <code>true</code>: Applies to all models</li><li>▶ <code>false</code>: Applies only to the current model</li></ul>
<return>	string	The UID of the skin.

#### 15.4.3.2.37. getSkinName

The function returns the name of the specified skin UID.

Table 15.60. Parameters of `getSkinName`

Parameter	Type	Description
<code>skinUid</code>	string	Skin for which the name is required.
<return>	string	The name of the skin.

#### 15.4.3.2.38. getTextHeight

The function returns the height of a text with regard to its font resource. The height represents the sum of the font ascender and descender.

Table 15.61. Parameters of `getTextHeight`

Parameter	Type	Description
text	string	The text to evaluate
font	font	The font to evaluate
<return>	integer	The height of the text  If the size of the font is 0 or negative, the function returns 0.

**NOTE**



**`getTextHeight`**

The function always calculates the height value assuming that the text has a single line.

#### 15.4.3.2.39. `getTextLength`

The function returns the number of characters in a text.

Table 15.62. Parameters of `getTextLength`

Parameter	Type	Description
text	string	The text to evaluate
<return>	integer	The number of characters in the text

**NOTE**



**Escape sequences**

EB GUIDE Script does not resolve escape sequences like `\n` and counts every character. For example, for the text `Label\n` the `getTextLength` function returns 7.

#### 15.4.3.2.40. `getTextWidth`

The function returns the width of a text with regard to its font resource.

Table 15.63. Parameters of `getTextWidth`

Parameter	Type	Description
text	string	The text to evaluate
font	font	The font to evaluate
<return>	integer	The width of the text  If the size of the font is 0 or negative, the function returns 0.

---

**NOTE****getTextWidth**

The function always calculates the width value assuming that the text has a single line.

---

**15.4.3.2.41. getVersionString**

The function returns a string with the version number of EB GUIDE GTF.

Table 15.64. Parameters of `getVersionString`

Parameter	Type	Description
<return>	string	The version string

**15.4.3.2.42. has\_list\_window**

The function checks if the index is valid for a datapool item of type list. For windowed lists it also checks if the index is located inside at least one window.

Table 15.65. Parameters of `has_list_window`

Parameter	Type	Description
itemId	dp_id	The ID of the datapool item of type list
index	integer	The index within the datapool item
<return>	boolean	If true, the index within a datapool item is valid and located inside at least one window.

**15.4.3.2.43. hsba2color**

The function converts an HSB/HSV color to an EB GUIDE GTF color.

Table 15.66. Parameters of `hsba2color`

Parameter	Type	Description
hue	integer	The color value in degrees from 0 to 360
saturation	integer	The saturation in percent
brightness	integer	The brightness in percent
alpha	integer	The alpha value between 0 (totally transparent) and 255 (opaque)
<return>	color	The resulting EB GUIDE GTF color with the alpha value applied



### 15.4.3.3. EB GUIDE Script functions I - R

#### 15.4.3.3.1. `int2float`

The function returns the integer value converted to a float point value.

Table 15.67. Parameters of `int2float`

Parameter	Type	Description
value	integer	The value to convert to float
<return>	float	The integer value, converted to float

#### 15.4.3.3.2. `int2string`

The function converts a simple integer to string.

Table 15.68. Parameters of `int2string`

Parameter	Type	Description
value	integer	The value to convert to string
<return>	string	The integer value, in decimal notation, converted to string

#### 15.4.3.3.3. `isDynamicStateMachineActive`

The function checks if a dynamic state machine is contained in a dynamic state machine list.

Table 15.69. Parameters of `isDynamicStateMachineActive`

Parameter	Type	Description
stack	Popupstack ID	The dynamic state machine list
sm	State machine ID	The dynamic state machine

#### 15.4.3.3.4. `isWidgetOnActiveStatemachine`

The function checks if the widget belongs to an active state machine.

Table 15.70. Parameters of `isWidgetOnActiveStatemachine`

Parameter	Type	Description
widget	widget	The widget to be evaluated
<return>	boolean	True if the widget belongs to an active state machine

#### 15.4.3.3.5. language

**Deprecated:** Use `setLanguage` instead.

The function switches the language for all models. The operation is performed synchronous, but the model reacts asynchronously on this change.

Table 15.71. Parameters of `language`

Parameter	Type	Description
<code>language</code>	string	The language to switch to, for example <code>f:language(l:German)</code>
<code>&lt;return&gt;</code>	void	

#### 15.4.3.3.6. lerp

The function calculates the linear interpolation of two values `x` and `y` using the formula  $(1-s) * x + s * y$

Table 15.72. Parameters of `lerp`

Parameter	Type	Description
<code>x</code>	float	The first value
<code>y</code>	float	The second value
<code>s</code>	float	A value that linearly interpolates between the <code>x</code> and <code>y</code> values
<code>&lt;return&gt;</code>	float	Returns the linear interpolation $(1-s) * x + s * y$

#### 15.4.3.3.7. localtime\_day

The function extracts the day [1:31] in local time from a system time value.

Table 15.73. Parameters of `localtime_day`

Parameter	Type	Description
<code>time</code>	integer	A time stamp as returned by system time
<code>&lt;return&gt;</code>	integer	The extracted day

#### 15.4.3.3.8. localtime\_hour

The function extracts the hours from the local time of a system time value.

Table 15.74. Parameters of `localtime_hour`

Parameter	Type	Description
<code>time</code>	integer	A time stamp as returned by system time
<code>&lt;return&gt;</code>	integer	The extracted hour

#### 15.4.3.3.9. `localtime_minute`

The function extracts the minutes from the local time of a system time value.

Table 15.75. Parameters of `localtime_minute`

Parameter	Type	Description
<code>time</code>	integer	A time stamp as returned by system time
<code>&lt;return&gt;</code>	integer	The extracted minute

#### 15.4.3.3.10. `localtime_month`

The function extracts the month [0:11] from the local time of a system time value.

Table 15.76. Parameters of `localtime_month`

Parameter	Type	Description
<code>time</code>	integer	A time stamp as returned by system time
<code>&lt;return&gt;</code>	integer	The extracted month

#### 15.4.3.3.11. `localtime_second`

The function extracts the seconds from the local time of a system time value.

Table 15.77. Parameters of `localtime_second`

Parameter	Type	Description
<code>time</code>	integer	A time stamp as returned by system time
<code>&lt;return&gt;</code>	integer	The extracted second

#### 15.4.3.3.12. `localtime_weekday`

The function extracts the week day [0:6] from the local time of a system time value. 0 is Sunday.

Table 15.78. Parameters of `localtime_weekday`

Parameter	Type	Description
<code>time</code>	integer	A time stamp as returned by system time
<code>&lt;return&gt;</code>	integer	The extracted weekday

#### 15.4.3.3.13. `localtime_year`

The function extracts the year from the local time of a system time value.

Table 15.79. Parameters of `localtime_year`

Parameter	Type	Description
<code>time</code>	integer	A time stamp as returned by system time
<code>&lt;return&gt;</code>	integer	The extracted year

#### 15.4.3.3.14. `log10f`

The function returns the base 10 logarithm of `x`.

Table 15.80. Parameters of `log10f`

Parameter	Type	Description
<code>x</code>	float	The argument
<code>&lt;return&gt;</code>	float	The return value

#### 15.4.3.3.15. `logf`

The function returns the natural logarithm of `x`.

Table 15.81. Parameters of `logf`

Parameter	Type	Description
<code>x</code>	float	The argument
<code>&lt;return&gt;</code>	float	The return value

#### 15.4.3.3.16. `maxf`

The function computes the maximum of two floating-point values.

Table 15.82. Parameters of `maxf`

Parameter	Type	Description
<code>x</code>	float	The first value
<code>y</code>	float	The second value
<code>&lt;return&gt;</code>	float	The maximum of <code>x</code> and <code>y</code>

#### 15.4.3.3.17. `maxi`

The function computes the maximum of two integer values.

Table 15.83. Parameters of `maxi`

Parameter	Type	Description
<code>x</code>	int	The first value
<code>y</code>	int	The second value
<code>&lt;return&gt;</code>	int	The maximum of <code>x</code> and <code>y</code>

#### 15.4.3.3.18. `minf`

The function computes the minimum of two floating-point values.

Table 15.84. Parameters of `minf`

Parameter	Type	Description
<code>x</code>	float	The first value
<code>y</code>	float	The second value
<code>&lt;return&gt;</code>	float	The minimum of <code>x</code> and <code>y</code>

#### 15.4.3.3.19. `mini`

The function computes the minimum of two integer values.

Table 15.85. Parameters of `mini`

Parameter	Type	Description
<code>x</code>	int	The first value
<code>y</code>	int	The second value
<code>&lt;return&gt;</code>	int	The minimum of <code>x</code> and <code>y</code>

#### 15.4.3.3.20. `nearbyint`

The function rounds to nearest integer.

Table 15.86. Parameters of `nearbyint`

Parameter	Type	Description
<code>value</code>	float	The value to round
<code>&lt;return&gt;</code>	integer	The rounded value

#### 15.4.3.3.21. `popDynamicStateMachine`

The function removes the dynamic state machine from a dynamic state machine list.

Table 15.87. Parameters of `popDynamicStateMachine`

Parameter	Type	Description
<code>stack</code>	Popupstack ID	The dynamic state machine list
<code>sm</code>	State machine ID	The dynamic state machine

#### 15.4.3.3.22. `powf`

The function returns the value of `x` raised to the power of `y`.

Table 15.88. Parameters of `powf`

Parameter	Type	Description
<code>x</code>	float	The argument <code>x</code>
<code>y</code>	float	The argument <code>y</code>
<code>&lt;return&gt;</code>	float	The return value

#### 15.4.3.3.23. `pushDynamicStateMachine`

The function inserts a dynamic state machine in a dynamic state machine list.

Table 15.89. Parameters of `pushDynamicStateMachine`

Parameter	Type	Description
<code>stack</code>	Popupstack ID	The dynamic state machine list
<code>sm</code>	State machine ID	The dynamic state machine
<code>priority</code>	integer	The priority of the dynamic state machine in the list. Note that a higher number means a higher priority.

#### 15.4.3.3.24. rad2deg

The function converts an angle from radians to degree.

Table 15.90. Parameters of `rad2deg`

Parameter	Type	Description
<code>x</code>	float	The argument
<code>&lt;return&gt;</code>	float	The return value

#### 15.4.3.3.25. rand

The function gets a random value between 0 and  $2^{31}-1$ .

Table 15.91. Parameters of `rand`

Parameter	Type	Description
<code>&lt;return&gt;</code>	integer	A random number between 0 and $2^{31}-1$

#### 15.4.3.3.26. rgba2color

The function converts from RGB color space to EB GUIDE GTF color.

Table 15.92. Parameters of `rgba2color`

Parameter	Type	Description
<code>red</code>	integer	The red color coordinate, ranging from 0 to 255
<code>green</code>	integer	The green color coordinate, ranging from 0 to 255
<code>blue</code>	integer	The blue color coordinate, ranging from 0 to 255
<code>alpha</code>	integer	The alpha value, ranging from 0 (totally transparent) to 255 (opaque)
<code>&lt;return&gt;</code>	color	The color converted from RGB color space to EB GUIDE GTF color, with the alpha value applied

#### 15.4.3.3.27. round

The function rounds to nearest integer, but rounds halfway cases away from zero.

Table 15.93. Parameters of `round`

Parameter	Type	Description
<code>value</code>	float	The value to round

Parameter	Type	Description
<return>	integer	The rounded value

#### 15.4.3.4. EB GUIDE Script functions S - W

##### 15.4.3.4.1. saturate

The function clamps a floating-point value to [0, 1] range, i.e. the function computes  $\max(0, \min(1, x))$  and acts as a shorthand notation for `clampf(0, 1, x)`

Table 15.94. Parameters of `saturate`

Parameter	Type	Description
x	float	The value to clamp
<return>	float	The x value clamped to the [0, 1] range

##### 15.4.3.4.2. seed\_rand

The function sets the seed of the random number generator.

Table 15.95. Parameters of `seed_rand`

Parameter	Type	Description
seed	integer	The value to seed the random number generator
<return>	void	

##### 15.4.3.4.3. setLanguage

The function switches the language for all models or for the current model only. The operation is performed synchronous, but the model reacts asynchronously on this change.

Table 15.96. Parameters of `setLanguage`

Parameter	Type	Description
languageUid	string	The language to switch to, for example <code>f:setLanguage(1:German, true)</code> .
isCoreScope	boolean	Specifies the scope.  ► true: Applies to all models



Parameter	Type	Description
		► false: Applies only to the current model
<return>	void	

#### 15.4.3.4.4. setSkin

The function switches the skin for all models or for the current model only. The operation is performed synchronous, but the model reacts asynchronously on this change.

Table 15.97. Parameters of setSkin

Parameter	Type	Description
skinUid	string	The skin to switch to, for example <code>f:setSkin(s:mySkin, true)</code> .
isCoreScope	boolean	Specifies the scope. ► true: Applies to all models ► false: Applies only to the current model
<return>	void	

#### 15.4.3.4.5. shutdown

The function requests the framework to shutdown the program.

#### 15.4.3.4.6. sinf

The function returns the sine of  $x$ , where  $x$  is given in radians.

Table 15.98. Parameters of sinf

Parameter	Type	Description
$x$	float	The argument
<return>	float	The return value

#### 15.4.3.4.7. skin

**Deprecated:** Use `setSkin` instead.

The function switches the skin for all models. The operation is performed synchronous, but the model reacts asynchronously on this change.

Table 15.99. Parameters of `skin`

Parameter	Type	Description
<code>skin</code>	string	The skin to switch to, for example <code>f:skin(s:mySkin)</code>
<code>&lt;return&gt;</code>	void	

#### 15.4.3.4.8. `smoothstep`

The function computes the smooth hermite interpolation  $3z^2 - 2z^3$  with  $z = (x - x_{\min}) / (x_{\max} - x_{\min})$  in case it is in range `[xmin, xmax]` and 0 otherwise. The function returns a value in the interval `[0, 1]`.

Table 15.100. Parameters of `smoothstep`

Parameter	Type	Description
<code>xmin</code>	float	The <code>xmin</code> value
<code>xmax</code>	float	The <code>xmax</code> value
<code>x</code>	float	The value to be interpolated
<code>&lt;return&gt;</code>	float	Returns the hermite interpolation $3z^2 - 2z^3$ with $z = (x - x_{\min}) / (x_{\max} - x_{\min})$

#### 15.4.3.4.9. `sqrtof`

The function returns the non-negative square root of `x`.

Table 15.101. Parameters of `sqrtof`

Parameter	Type	Description
<code>x</code>	float	The argument
<code>&lt;return&gt;</code>	float	The return value

#### 15.4.3.4.10. `string2float`

The function converts the initial part of a string to float.

The expected form of the initial part of the string is as follows:

1. Optional leading white space
2. Optional plus ('+') or minus ('-') sign
3. One of the following:
  - Decimal number

- ▶ Hexadecimal number
- ▶ Infinity
- ▶ NAN (not-a-number)

Table 15.102. Parameters of `string2float`

Parameter	Type	Description
<code>str</code>	string	The string value
<code>&lt;return&gt;</code>	float	The return value

#### 15.4.3.4.11. `string2int`

The function converts the initial part of a string to integer. The result is clipped to the range from 2147483647 to -2147483648, if the input exceeds the range. If the string does not start with a number, the function returns 0.

Table 15.103. Parameters of `string2int`

Parameter	Type	Description
<code>str</code>	string	The string value
<code>&lt;return&gt;</code>	integer	The return value

#### 15.4.3.4.12. `string2string`

The function is used to truncate a string to a given number of characters.

Table 15.104. Parameters of `string2string`

Parameter	Type	Description
<code>str</code>	string	The string to truncate
<code>len</code>	integer	The maximum length of the string
<code>&lt;return&gt;</code>	string	The truncated string

#### 15.4.3.4.13. `substring`

The function creates a substring copy of the string. Negative end indexes are supported.

Examples:

- ▶ `substring("abc", 0, -1)` returns `abc`.
- ▶ `substring("abc", 0, -2)` returns `ab`.
- ▶ `substring ("abcd", 1, 3)` returns `bc`.

Table 15.105. Parameters of `substring`

Parameter	Type	Description
<code>str</code>	string	The input string
<code>startIndex</code>	integer	The first character index of the result string
<code>endIndex</code>	integer	The first character index that is not part of the result
<code>&lt;return&gt;</code>	string	The language string

#### 15.4.3.4.14. `system_time`

The function gets the current system time in seconds. The result is intended to be passed to the `localtime_*` functions.

Table 15.106. Parameters of `system_time`

Parameter	Type	Description
<code>&lt;return&gt;</code>	integer	The system time in seconds

#### 15.4.3.4.15. `system_time_ms`

The function gets the current system time in milliseconds.

Table 15.107. Parameters of `system_time_ms`

Parameter	Type	Description
<code>&lt;return&gt;</code>	integer	The system time in milliseconds

#### 15.4.3.4.16. `tanf`

The function returns the tangent of  $x$ , where  $x$  is given in radians.

Table 15.108. Parameters of `tanf`

Parameter	Type	Description
<code>x</code>	float	The argument
<code>&lt;return&gt;</code>	float	The return value

#### 15.4.3.4.17. `trace_dp`

The function writes debugging information about a datapool item to the trace log and the connection log.

Table 15.109. Parameters of `trace_dp`

Parameter	Type	Description
<code>itemId</code>	<code>dp_id</code>	The datapool ID of the item to trace debug information about
<code>&lt;return&gt;</code>	<code>void</code>	

#### 15.4.3.4.18. `trace_string`

The function writes a string to the trace log and the connection log.

Table 15.110. Parameters of `trace_string`

Parameter	Type	Description
<code>str</code>	<code>string</code>	The text to trace
<code>&lt;return&gt;</code>	<code>void</code>	

#### 15.4.3.4.19. `transformToScreenX`

The function takes a widget and a local coordinate and returns x-position in the screen-relative world coordinate system.

Table 15.111. Parameters of `transformToScreenX`

Parameter	Type	Description
<code>widget</code>	<code>widget</code>	The widget to which the coordinates are relative
<code>localX</code>	<code>integer</code>	The x-position of the local coordinate
<code>localY</code>	<code>integer</code>	The y-position of the local coordinate
<code>&lt;return&gt;</code>	<code>integer</code>	The x-position of the screen coordinate

#### 15.4.3.4.20. `transformToScreenY`

The function takes a widget and a local coordinate and returns y-position of a position in the screen-relative world coordinate system.

Table 15.112. Parameters of `transformToScreenY`

Parameter	Type	Description
<code>widget</code>	<code>widget</code>	The widget to which the coordinates are relative
<code>localX</code>	<code>integer</code>	The x-position of the local coordinate
<code>localY</code>	<code>integer</code>	The y-position of the local coordinate

Parameter	Type	Description
<return>	integer	The y-position of the screen coordinate

#### 15.4.3.4.21. `transformToWidgetX`

The function takes a widget and a screen coordinate as provided to the touch reactions and returns x-position in the widget-relative local coordinate system.

Table 15.113. Parameters of `transformToWidgetX`

Parameter	Type	Description
<code>widget</code>	widget	The widget to which the coordinates are relative
<code>screenX</code>	integer	The x-position of the screen coordinate
<code>screenY</code>	integer	The y-position of the screen coordinate
<return>	integer	The x-position of the local coordinate

#### 15.4.3.4.22. `transformToWidgetY`

The function takes a widget and a screen coordinate as provided to the touch reactions and returns y-position in the widget-relative local coordinate system.

Table 15.114. Parameters of `transformToWidgetY`

Parameter	Type	Description
<code>widget</code>	widget	The widget to which the coordinates are relative
<code>screenX</code>	integer	The x-position of the screen coordinate
<code>screenY</code>	integer	The y-position of the screen coordinate
<return>	integer	The y-position of the local coordinate

#### 15.4.3.4.23. `trunc`

The function rounds to the nearest integer value, always towards zero.

Table 15.115. Parameters of `trunc`

Parameter	Type	Description
<code>value</code>	float	The value to round
<return>	integer	The rounded value

#### 15.4.3.4.24. widgetGetChildCount

The function obtains the number of child widgets of the given widget.

Table 15.116. Parameters of widgetGetChildCount

Parameter	Type	Description
widget	widget	The widget of which to obtain the number of child widgets
<return>	integer	The number of child widgets

## 15.5. Events

Table 15.117. Properties of an event

Property name	Description
Name	The name of the event
Event ID	A numeric value that EB GUIDE TF uses to send and receive the event
Event group	The name of the event group  An event group has an ID that EB GUIDE TF uses to send and receive the event.

### 15.5.1. Decimal codes for key events

Table 15.118. Decimal codes of numpad keys

Numpad key	Decimal code
0	5
1	6
2	7
3	8
4	9
5	10
6	11
7	12
8	13
9	14

Table 15.119. Decimal codes of function keys

Function key	Decimal code
<b>F1</b>	18
<b>F2</b>	19
<b>F3</b>	20
<b>F4</b>	21
<b>F5</b>	22
<b>F6</b>	23
<b>F7</b>	24
<b>F8</b>	25
<b>F9</b>	26
<b>F10</b>	27
<b>F11</b>	28
<b>F12</b>	29

Table 15.120. Decimal codes of ASCII keys

ASCII key	Decimal code
<b>Space</b>	32
<b>a</b>	97
<b>b</b>	98
<b>c</b>	99
<b>d</b>	100
<b>e</b>	101
<b>f</b>	102
<b>g</b>	103
<b>h</b>	104
<b>i</b>	105
<b>j</b>	106
<b>k</b>	107
<b>l</b>	108
<b>m</b>	109
<b>n</b>	110
<b>o</b>	111













ASCII key	Decimal code
<b>p</b>	112
<b>q</b>	113
<b>r</b>	114
<b>s</b>	115
<b>t</b>	116
<b>u</b>	117
<b>v</b>	118
<b>w</b>	119
<b>x</b>	120
<b>y</b>	121
<b>z</b>	122

## 15.6. Buttons and icons

The following tables list icons that are used in EB GUIDE Studio and EB GUIDE Monitor and explain their meaning.

Table 15.121. General icons

General icons	Description
	Undo
	Redo
	Save
	Validates the project.
	Starts the simulation.
	Stops the simulation.
	Opens and closes the project center.
	Adds an element, for example, an event, a datapool item, or a state machine.
	Closes a component or a tab.
	Indicates a successful operation.


General icons	Description
	Indicates a piece of information.

Table 15.122. Project center icons



















Project center icons	Description
	Indicates the tab where a new project can be created.
	Indicates the tab where an existing project can be opened.
	Indicates the tab where several project options can be configured, such as, the model interface, event groups, or languages and skins.
	Indicates the tab where you can create an export of the EB GUIDE model.
	Indicates the tab where you can access the user documentation.
	Indicates the tab where you can change the user interface language.
	Indicates the tab where you can see which plug-ins were loaded.
	Imports the project interface.
	Exports the project interface.

Table 15.123. Project editor icons

Project editor icons	Description
	Hides a component.
	Shows a hidden component.
	Synchronizes content area and the <b>Navigation</b> or <b>Templates</b> component.
	<p>Opens a property-related context menu.</p> <p>The button's colors listed below indicate the following:</p> <ul style="list-style-type: none"> <li> Property is local.</li> <li> Property is linked to another property.</li> <li> Property is linked to a datapool item.</li> </ul>

Project editor icons	Description
	 Property value is equal to template value.  Datapool item is skin-dependent.  Datapool item is language-dependent.
{ }	Opens the EB GUIDE Script editor.
	Opens namespace options or shows selected namespace.
	Shows all namespaces.
	Includes sub-namespaces.
	Opens the settings.
	Indicates a datapool item.
	Indicates a transition.
	Indicates a dynamic state machine list.
	Indicates an entry action.
	Indicates an exit action.
	Indicates an internal transition.
	Indicates a template.
	Jumps to the template of this widget.
	Filters search results or list elements.
Ps	Indicates a Photoshop file.
	Indicates that something changed in the widget feature properties.
	Indicates a trigger list.
	Toggles grouping by model interface.

Table 15.124. State icons

State icons	Description
	Indicates a choice state.
	Indicates a deep history state.






State icons	Description
	Indicates a final state.
	Indicates an initial state.
	Indicates a state machine.
	Indicates a shallow history state.
	Indicates a view state.

Table 15.125. Basic widget icons












Basic widget icons	Description
	Indicates an alpha mask.
	Indicates a container.
	Indicates a custom widget.
	Indicates an ellipse.
	Indicates an image.
	Indicates an instantiator.
	Indicates a label.
	Indicates a rectangle.

Table 15.126. Animation icons

Animation icons	Description
	Indicates an animation widget.
	Indicates an animation with a constant curve.
	Indicates an animation with a fast start curve.




















Animation icons	Description
	Indicates an animation with a linear curve.
	Indicates an animation with a linear interpolation curve.
	Indicates an animation with a quadratic curve.
	Indicates an animation with a script curve.
	Indicates an animation with a sinus curve.
	Indicates an animation with a slow start curve.
	Indicates a change animation.
	Indicates an entry animation.
	Indicates an exit animation.
	Expands the animation editor.
	Minimizes the animation editor.
	Indicates a pop-up on animation.
	Indicates a pop-up off animation.
	Indicates a broken link.

Table 15.127. 3D widget icons

3D widget icons	Description
	Indicates an ambient light.
	Indicates a camera.
	Indicates a directional light.
	Indicates an image-based light.
	Indicates a material.






3D widget icons	Description
	Indicates a mesh.
	Indicates a point light.
	Indicates a scene graph.
	Indicates a scene graph node.
	Indicates a spot light.

Table 15.128. **Problems** component icons














Problems component icons	Description
	Validates the model.
	Indicates an issue.
	Indicates a warning.

Table 15.129. EB GUIDE Monitor icons

EB GUIDE Monitor icons	Description
	Fires an event or indicates that an event has been fired.
	Indicates that a key event has been fired.
	Indicates whether a connection to a host is established.
	Opens the connection configuration.
	Turns the automatic scrolling of the log on and off.
	Copies all log messages.
	Deletes the log messages.
	Exports the watch list.
	Indicates a log message.
	Toggles grouping by model interface.

## 15.7. Scenes

Table 15.130. Properties of a scene

Property name	Description
height	The height of the area in which the views of a haptic state machine are rendered on a target device
width	The width of the area in which the views of a haptic state machine are rendered on a target device
x	The x-offset of the area in which the views of a haptic state machine are rendered on a target device
y	The y-offset of the area in which the views of a haptic state machine are rendered on a target device
visible	If true, the state machine and its child widgets are visible.
projectName	The name of the EB GUIDE project
windowCaption	The text that is shown on the window frame
sceneID	The unique scene identifier which can be used, for example, for input handling
maxFPS	The redraw rate (FPS = Frames per second)  Set to 0 for an unlimited redraw rate.
hwLayerID	The ID of the hardware layer on the target device's display that is mapped to the current state machine
colorMode	<p>Possible values:</p> <ul style="list-style-type: none"> <li>▶ 32-bit (1): RGBA8888</li> <li>▶ 16-bit (2): RGB565</li> <li>▶ 24-bit (3): RGB888</li> <li>▶ 32-bit sRGB (4):</li> </ul> <p>This value uses GPU hardware support.</p> <p>Use this value, if you want to have sRGB support for an image widget or for a texture widget feature with an sRGB property.</p> <ul style="list-style-type: none"> <li>▶ 32-bit sRGB (Emulated) (5):</li> </ul> <p>Use this value only if 32-bit sRGB does not yield correct results.</p>
antiAliasing	<p>Possible values:</p> <ul style="list-style-type: none"> <li>▶ off (0): no anti-aliasing</li> </ul>

Property name	Description
	<ul style="list-style-type: none"> <li>▶ MSAA 2x (1): 2x anti-aliasing</li> <li>▶ MSAA 4x (2): 4x anti-aliasing</li> </ul> <p>Also see <a href="#">section 6.3, “Anti-aliasing”</a>.</p>
<code>enableRemoteFramebuffer</code>	If true, transfer of the off-screen buffer to the simulation window is enabled
<code>showWindowFrame</code>	If true, a frame is displayed on the simulation window. The frame allows the window to be grabbed and moved.
<code>showWindow</code>	If true, an additional window for simulation is opened on Windows based systems.
<code>disableVSync</code>	If true, vertical synchronization for the renderer is disabled.
<code>showFPS</code>	<p>Possible values:</p> <ul style="list-style-type: none"> <li>▶ <code>off (0)</code>: Do not show FPS</li> <li>▶ <code>on screen (1)</code>: Show FPS on the screen</li> <li>▶ <code>console (2)</code>: Show FPS on the console</li> <li>▶ <code>console &amp; on screen (3)</code>: Show FPS on the screen and on the console</li> <li>▶ <code>on screen (large text) (4)</code></li> <li>▶ <code>console &amp; on screen (large text) (5)</code></li> </ul>
<code>Renderer</code>	<p>Defines a renderer for the scene.</p> <p>Possible values:</p> <ul style="list-style-type: none"> <li>▶ <code>OpenGLRenderer</code></li> <li>▶ <code>OpenGL3Renderer</code></li> </ul>

#### NOTE



#### Using `sceneID` in the scene configuration

When using the same `sceneID` in the scene configuration, multiple state machines react to input handling at the same time.

To avoid that and to achieve that only one state machine reacts to input handling, assign different `sceneID` values to each state machine in the scene configuration.

## 15.8. Shortcuts



The following table lists shortcuts available in EB GUIDE Studio and EB GUIDE Monitor and explains their meaning.

Table 15.131. Shortcuts

Shortcut	Description
<b>Ctrl+A</b>	Select all elements
<b>Ctrl+C</b>	Copy the selection
<b>Ctrl+F</b>	Jump into search box
<b>Ctrl+S</b>	Save
<b>Ctrl+V</b>	Paste the copied selection
<b>Ctrl+Y</b>	Redo
<b>Ctrl+Z</b>	Undo
<b>Enter</b>	In tables, confirm the entered value and go to the next cell.
<b>Ctrl+Enter</b>	In the trigger filter box, add a new event.  In tables, confirm the entered value and stay in the cell.
<b>Ctrl+Shift+Insert</b>	In the <b>Namespaces</b> component, add a new namespace as a child to an existing namespace.
<b>Alt+F4</b>	Close the active window
<b>Shift+F1</b>	Open user documentation for EB GUIDE TF
<b>F1</b>	Open user documentation for EB GUIDE Studio
<b>F2</b>	Rename the selected element
<b>Shift+F2</b>	Rename the selected element globally, this means in all locations where the selected element is used, for example in EB GUIDE Script.
<b>F3</b>	Find all references of the selected element in the EB GUIDE model
<b>F4</b>	Jump to origin template when applied on a selected widget.  Jump to linked target when applied on a selected datapool item or widget property.
<b>F5</b>	Start simulation
<b>F6</b>	Validate
<b>Del</b>	Delete the selected element.
<b>-</b>	In trees, collapse the selected element.
<b>+</b>	In trees, expand the selected element.
<b>*</b>	In trees, expand the selected element and all children of this element.

Shortcut	Description
<b>Up/Down/Left/Right</b>	In the content area, move the selected state or widget one pixel up, down, left, or right.  In tables, go through the elements.
<b>Shift</b> and hold the mouse button	In the content area, move the selected transition label.
<b>Ctrl</b> and hold the mouse button	In the content area, rotate the selected transition label.
<b>Ctrl</b> and click the left mouse button  <b>Shift</b> and click the left mouse button or <b>Up arrow</b> or <b>Down arrow</b> keys	Select multiple elements.
<b>Ctrl</b> and rotate the wheel button  <b>Ctrl++</b>  <b>Ctrl+-</b>  <b>Ctrl+0</b>	In the content area, zoom in and out or reset the scaling of the content area to 100%.

## 15.9. State machines

### 15.9.1. Haptic state machine

A state machine that describes elements that are visible in the HMI.

Table 15.132. Properties of a haptic state machine

Property name	Description
Entry action	Defines an action that is executed every time the state is entered.
Exit action	Defines an action that is executed every time the state is exited.
Dynamic state machine list	Enables the usage of dynamic state machines in EB GUIDE Script.
Internal transitions	Adds an internal transition.
Background color	Configures the background color that is displayed in EB GUIDE Studio. This does not affect the background color on the target.

## 15.9.2. Logic state machine

A state machine that describes behavior that takes place outside of visible elements.

Table 15.133. Properties of a logic state machine

Property name	Description
Entry action	Defines an action that is executed every time the state is entered.
Exit action	Defines an action that is executed every time the state is exited.
Dynamic state machine list	Enables the usage of dynamic state machines in EB GUIDE Script.
Internal transitions	Adds an internal transition.
Background color	Configures the background color that is displayed in EB GUIDE Studio. This does not affect the background color on the target.

## 15.9.3. States

### 15.9.3.1. Initial state

Defines the starting point of the state machine. An initial state has an outgoing default transition that points to the first state. An initial state has no incoming transition.

These states have no properties.

### 15.9.3.2. Compound state

Table 15.134. Properties of a Compound state

Property name	Description
Entry action	Defines an action that is executed every time a state or state machine is entered.
Exit action	Defines an action that is executed every time a state or state machine is exited.
Dynamic state machine list	Enables the usage of dynamic state machines in EB GUIDE Script.
Internal transitions	Adds an internal transition.

Property name	Description
Background color	Configures the background color.

### 15.9.3.3. View state

Contains a view. A view represents a project specific HMI screen. The view is displayed while the corresponding view state is active.

Table 15.135. Properties of a View state

Property name	Description
Entry action	Defines an action that is executed every time a state or state machine is entered.
Exit action	Defines an action that is executed every time a state or state machine is exited.
Dynamic state machine list	Enables the usage of dynamic state machines in EB GUIDE Script.
Internal transitions	Adds an internal transition.
Background color	Configures the background color.

### 15.9.3.4. Choice state

Evaluates conditions and chooses a transition that matches the conditions.

These states have no properties.

### 15.9.3.5. Shallow history state

Stores the most recent active sub-state: the sub-state that was active just before exiting the compound state.

These states have no properties.

### 15.9.3.6. Deep history state

Stores a compound state and its complete sub-hierarchy just before the compound state is exited.

These states have no properties.

### 15.9.3.7. Final state

Is used to exit a compound state or state machine. If the final state of the state machine is entered, the state machine terminates. Any history states within the compound state are reset. A final state does not have any outgoing transitions.

These states have no properties.

## 15.9.4. Transitions

### 15.9.4.1. Default transition

A default transition is triggered automatically and not by any event or datapool item update. It has no condition, but can have an action. It is used with initial state, final state, choice state, and history states.

Table 15.136. Properties of a default transition

Property name	Description
Action	Defines an action that is executed every time the transition is executed.

### 15.9.4.2. Choice transition

A choice transition is an outgoing transition with a condition assigned to it. Its source state is a choice state. Choice transitions are triggered by the evaluation of their condition. They result in an action. The first choice transition that has condition true is executed.

Table 15.137. Properties of a choice transition

Property name	Description
Action	Defines an action that is executed every time the transition is executed.
Condition	Defines a condition that must be true for this transition to be executed.

### 15.9.4.3. Else transition

An else transition is the mandatory counterpart of a choice transition. Every choice state needs to have one else transition which is executed if the conditions of all its choice transitions evaluate to false.

Table 15.138. Properties of an else transition

Property name	Description
Action	Defines an action that is executed every time the transition is executed.

#### 15.9.4.4. Internal transition

An internal transition is a transition that has no target state and thus does not change the active state. The purpose of an internal transition is to react to an event without leaving the present state. It can have a condition and it results in an action.

Table 15.139. Properties of an internal transition

Property name	Description
Action	Defines an action that is executed every time the transition is executed.
Condition	Defines a condition that must be true for this transition to be executed.
Trigger	Defines a trigger for this transition.

#### 15.9.4.5. Self transition

A self transition is a transition with the same state as source state and target state. Unlike an internal transition, a self transition leaves and re-enters the state and thus executes its entry and exit actions.

Table 15.140. Properties of a self transition

Property name	Description
Action	Defines an action that is executed every time the transition is executed.
Condition	Defines a condition that must be true for this transition to be executed.
Trigger	Defines a trigger for this transition.

## 15.10. Widgets

### 15.10.1. View

Table 15.141. Properties of the View widget

Property name	Description
height	The height of the widget in pixels
width	The width of the widget in pixels
visible	If true, the widget and its child widgets are visible
x	The x-coordinate of the widget

Property name	Description
<code>y</code>	The y-coordinate of the widget

View states and view templates have additional properties for view transition animations. View transition animations apply for entry animations, exit animations, change animations, pop-up on animations and pop-up off animations.

Table 15.142. Properties of a view transition animation

Property name	Description
<code>enabled</code>	Defines whether the animation is executed.
<code>repeat</code>	The number of repetitions, 0 for infinite number.
<code>alternating</code>	<p>If true, the animation is executed repeatedly back and forth, i.e. bidirectional.</p> <p>If false, the animation is executed repeatedly only in one direction, i.e. unidirectional.</p> <p>The number of repetitions is defined in the <code>repeat</code> property.</p>
<code>scale</code>	The factor by which the animation time is multiplied.
<code>onPlay</code>	The reaction that is executed when the animation is started or continued. Parameters: Start time and play direction (true for forwards, false for backwards).
<code>onPause</code>	The reaction that is executed when the animation is paused. Parameter: Current animation time.
<code>onTerminate</code>	<p>The reaction that is executed when the animation completes. First parameter: Animation time. Second parameter: Reason for the termination, encoded as follows:</p> <ul style="list-style-type: none"> <li>▶ 0: Animation is completed</li> <li>▶ 1: Animation is cancelled, triggered by <code>f:animation_cancel</code></li> <li>▶ 2: Widget is destroyed due to view transition</li> <li>▶ 3: Animation jumps to its last step, triggered by <code>f:animation_cancel_end</code></li> <li>▶ 4: Animation jumps to its first step and is then canceled, triggered by <code>f:animation_cancel_reset</code></li> </ul>

## 15.10.2. Basic widgets

There are eight basic widgets:

- ▶ Alpha mask

- ▶ Animation
- ▶ Container
- ▶ Ellipse
- ▶ Image
- ▶ Instantiator
- ▶ Label
- ▶ Rectangle

The following sections list the properties of basic widgets.

---

**NOTE**



**Unique names**

Use unique names for two widgets with the same parent widget.

---

---

**NOTE**



**Negative values**

Do not use negative values for `height` and `width` properties. EB GUIDE Studio treats negative values as 0, this means the respective widget will not be depicted.

---

### 15.10.2.1. Alpha mask

An alpha mask is a container widget that controls the alpha channel, i.e. the opacity, of its child widgets with an image.

Table 15.143. Properties of the Alpha mask widget

Property name	Description
<code>visible</code>	If true, the widget and its child widgets are visible
<code>width</code>	The width of the widget in pixels
<code>height</code>	The height of the widget in pixels
<code>x</code>	The x-coordinate of the widget relative to its parent widget
<code>y</code>	The y-coordinate of the widget relative to its parent widget
<code>enabled</code>	If true, the alpha mask is applied to the child widgets
<code>image</code>	The image that controls the alpha channel, i.e. the opacity of the child widgets
<code>horizontalAlign</code>	The horizontal alignment of the image file within the boundaries of the widget
<code>verticalAlign</code>	The vertical alignment of the image file within the boundaries of the widget



Property name	Description
scaleMode	<p>The scale mode of the image. Possible values:</p> <ul style="list-style-type: none"> <li>▶ original size (0)</li> <li>▶ fit to size (1)</li> <li>▶ keep aspect ratio (2)</li> </ul>

#### NOTE



#### Supported image file types for alpha mask

The available image formats depend on the implementation of the renderer. The renderers for OpenGL ES 2.0 or higher support `.png` files and `.jpg` files. RGB images are converted to grayscale images before being used as alpha masks. Grayscale images are used as is. The alpha channel in the image is ignored.

Alpha mask functionality is not applied to 9-patch images. 9-patch images are handled the same way the PNG and JPEG file formats are.

### 15.10.2.2. Animation

Animation is achieved by changing a value over a duration of time. Animations are defined with animation curves and their properties. Animations curves are available through the **Animation editor**.

Table 15.144. Properties of the Animation widget

Property name	Description
enabled	Define whether the animation is enabled.
repeat	Define how many times the animation is repeated. 0 means infinite repetition.
alternating	If enabled, every second repetition is a reversed animation.
scale	Define the scaling of the duration. Values below 1 slow down the animation. Values above 1 speed up the animation.
onPlay	<p>Define an action that is executed when the animation is started or continued.</p> <ul style="list-style-type: none"> <li>▶ <code>animation_time</code>: The time at which the animation is started or continued.</li> <li>▶ <code>forward</code>: If true, the animation goes forward. If false, the animation is reversed.</li> </ul>
onPause	<p>Define an action that is executed when the animation is paused.</p> <ul style="list-style-type: none"> <li>▶ <code>animation_time</code>: The time at which the animation is paused.</li> </ul>
onTerminate	<p>Define an action that is executed when the animation is terminated.</p> <ul style="list-style-type: none"> <li>▶ <code>animation_time</code> The period of time for which the animation ran.</li> </ul>

Property name	Description
	<ul style="list-style-type: none"><li>▶ <code>terminate</code> The reason for the termination.<ul style="list-style-type: none"><li>▶ 0: The animation was completed.</li><li>▶ 1: <code>animation_cancel</code> canceled the animation.</li><li>▶ 2: A view transition canceled the animation.</li><li>▶ 3: <code>animation_cancel_end</code> canceled the animation.</li><li>▶ 4: <code>animation_cancel_reset</code> canceled the animation.</li></ul></li></ul>

#### 15.10.2.2.1. Constant curve

A constant curve changes a value instantly, not gradually. The following use cases are common:

- ▶ Switching a boolean value
- ▶ Highlighting elements of the HMI

Constant curves can be applied to values of the data types integer, boolean, float, and color.

Table 15.145. Properties of the constant curve

Property name	Description
<code>enabled</code>	Define whether the animation is enabled.
<code>delay</code>	Define how long the start of the animation is delayed in milliseconds.
<code>duration</code>	Define the duration of the animation in milliseconds.
<code>repeat</code>	Define how many times the animation is repeated. 0 means infinite repetition.
<code>alternating</code>	If enabled, every second repetition is a reversed animation.
<code>relative</code>	If enabled, the <code>target</code> value is added to the <code>value</code> .
<code>value</code>	Define towards which value the <code>target</code> value is changed.
<code>target</code>	Define which property or datapool item is animated and its value before the animation. This value is overwritten during the animation.

#### 15.10.2.2.2. Fast start curve

A fast start curve changes a value quickly at the start and decelerating when the change approaches the end value. The following use cases are common:

- ▶ Toggle buttons
- ▶ Opening menus

- ▶ Fading in and out
- ▶ View transition animation
- ▶ Highlighting elements of the HMI

Fast start curves can be applied to values of the data types integer, float, and color.

Table 15.146. Properties of the fast start curve

Property name	Description
enabled	Define whether the animation is enabled.
delay	Define how long the start of the animation is delayed in milliseconds.
duration	Define the duration of the animation in milliseconds.
repeat	Define how many times the animation is repeated. 0 means infinite repetition.
alternating	If enabled, every second repetition is a reversed animation.
relative	If enabled, the <code>target</code> value is added to the <code>start</code> value.
start	Define at which value the animation starts.
end	Define at which value the animation ends.
target	Define which property or datapool item is animated and its value before the animation. This value is overwritten during the animation.

#### 15.10.2.2.3. Slow start curves

A slow start curve changes a value slowly at the start and accelerating when the change approaches the end value. The following use cases are common:

- ▶ Toggle buttons
- ▶ Opening menus
- ▶ Fading in and out
- ▶ View transition animation
- ▶ Highlighting elements of the HMI

Slow start curves can be applied to values of the data types integer, float, and color.

Table 15.147. Properties of the slow start curve

Property name	Description
enabled	Define whether the animation is enabled.
delay	Define how long the start of the animation is delayed in milliseconds.

Property name	Description
duration	Define the duration of the animation in milliseconds.
repeat	Define how many times the animation is repeated. 0 means infinite repetition.
alternating	If enabled, every second repetition is a reversed animation.
relative	If enabled, the <code>target</code> value is added to the <code>start</code> value.
start	Define at which value the animation starts.
end	Define at which value the animation ends.
target	Define which property or datapool item is animated and its value before the animation. This value is overwritten during the animation.

#### 15.10.2.2.4. Quadratic curve

A quadratic curve accelerates the change of a value over a defined period of time. A quadratic equation is used to calculate the change. There is no defined end to the change. The change stops at the value that is reached at the end of the time. The following use cases are common:

- ▶ Fading in and out
- ▶ Perpetual animation
- ▶ Highlighting elements of the HMI

Quadratic curves can be applied to values of the data types integer, float, and color types.

Table 15.148. Properties of the quadratic curve

Property name	Description
enabled	Define whether the animation is enabled.
delay	Define how long the start of the animation is delayed in milliseconds.
duration	Define the duration of the animation in milliseconds.
repeat	Define how many times the animation is repeated. 0 means infinite repetition.
alternating	If enabled, every second repetition is a reversed animation.
relative	If enabled, the <code>target</code> value is added to the <code>constant</code> value.
acceleration	Define the value that is added to the <code>velocity</code> at every full second of the animation.
velocity	Define the value that is added gradually during one second of the animation.
constant	Define at which value the animation starts.
target	Define which property or datapool item is animated and its value before the animation. This value is overwritten during the animation.

#### 15.10.2.2.5. Sinus curve

A sinus curve changes a value gradually back and forth over a defined period of time. There is no defined end to the change. The change stops at the value that is reached at the end of the time. The following use cases are common:

- ▶ Perpetual animation
- ▶ Circular motion
- ▶ Indicating passage of time
- ▶ Highlighting elements of the HMI

Sinus curves can be applied to values of the data types integer, float, and color.

Table 15.149. Properties of the sinus curve

Property name	Description
enabled	Define whether the animation is enabled.
delay	Define how long the start of the animation is delayed in milliseconds.
duration	Define the duration of the animation in milliseconds.
repeat	Define how many times the animation is repeated. 0 means infinite repetition.
alternating	If enabled, every second repetition is a reversed animation.
relative	If enabled, the <code>target</code> value is added to the <code>constant</code> value.
amplitude	Define the amplitude of the oscillation.
constant	Define the fixed point of the oscillation.
frequency	Define the frequency of the oscillation in hertz.
phase	Define the angular phase at which the oscillation starts.
target	Define which property or datapool item is animated and its value before the animation. This value is overwritten during the animation.

#### 15.10.2.2.6. Script curve

The script curve is a curve that you can define yourself through EB GUIDE Script. Use the script curve in cases where you want to have an animation that is not possible with the other curves or that is your own, custom animation. This curve is especially useful if you want to have a customized trajectory for the movement of a widget. Script curves can be applied to values of the data types integer, boolean, float, and color.

Table 15.150. Properties of the script curve

Property name	Description
enabled	Define whether the animation is enabled.

Property name	Description
delay	Define how long the start of the animation is delayed in milliseconds.
duration	Define the duration of the animation in milliseconds.
repeat	Define how many times the animation is repeated. 0 means infinite repetition.
alternating	If enabled, every second repetition is a reversed animation.
relative	If enabled, the <code>target</code> value is added to the <code>curve</code> value.
curve	Define your curve function in EB GUIDE Script. Two parameters are provided: <ul style="list-style-type: none"> <li>▶ <code>diff</code>: The time in ms since the last execution. At the start of the animation <code>diff</code> is 0.</li> <li>▶ <code>t_anim</code>: The time in ms since the start of the animation.</li> </ul>
target	Define which property or datapool item is animated and its value before the animation. This value is overwritten during the animation.

#### 15.10.2.2.7. Linear curve

A linear curve changes a value by repeatedly adding a defined value over a defined period of time. There is no defined end to the change. The change stops at the value that is reached at the end of the time. The following use cases are common:

- ▶ Perpetual animation
- ▶ Indicating passage of time
- ▶ Highlighting elements of the HMI

Linear curves can be applied to values of the data types integer, float, and color.

Table 15.151. Properties of the linear curve

Property name	Description
enabled	Define whether the animation is enabled.
delay	Define how long the start of the animation is delayed in milliseconds.
duration	Define the duration of the animation in milliseconds.
repeat	Define how many times the animation is repeated. 0 means infinite repetition.
alternating	If enabled, every second repetition is a reversed animation.
relative	If enabled, the <code>target</code> value is added to the <code>constant</code> value.
velocity	Define the value that is added gradually during one second of the animation.
constant	Define at which value the animation starts.

Property name	Description
target	Define which property or datapool item is animated and its value before the animation. This value is overwritten during the animation.

#### 15.10.2.2.8. Linear interpolation curve

A linear interpolation curve changes a value gradually towards an end value adding increments of equal value per unit of time. The following use cases are common:

- ▶ Indicating limited time
- ▶ Indicating passage of time
- ▶ Highlighting elements of the HMI

Linear interpolation curves can be applied to values of the data types integer, float, and color.

#### NOTE



#### Linear key value interpolation curve

During import of a 3D graphic file, if the imported 3D scene has animations, linear key value interpolation integer curve and linear key value interpolation float curve are created. The underlying key-value pairs of these curves cannot be modified in EB GUIDE Studio.

Table 15.152. Properties of the linear interpolation curve

Property name	Description
enabled	Define whether the animation is enabled.
delay	Define how long the start of the animation is delayed in milliseconds.
duration	Define the duration of the animation in milliseconds.
repeat	Define how many times the animation is repeated. 0 means infinite repetition.
alternating	If enabled, every second repetition is a reversed animation.
relative	If enabled, the <code>target</code> value is added to the <code>start</code> value.
start	Define at which value the animation starts.
end	Define at which value the animation ends.
target	Define which property or datapool item is animated and its value before the animation. This value is overwritten during the animation.

#### 15.10.2.3. Container

A container holds several widgets as child widgets and thus groups the widgets.

Table 15.153. Properties of the Container widget

Property name	Description
height	The height of the widget in pixels
width	The width of the widget in pixels
visible	If true, the widget and its child widgets are visible
x	The x-coordinate of the widget relative to its parent widget
y	The y-coordinate of the widget relative to its parent widget

#### 15.10.2.4. Ellipse

An ellipse draws a colored ellipse with the dimensions and coordinates of the widget into a view. The widget can also be used to draw a sector or an arc.

Table 15.154. Properties of the Ellipse widget

Property name	Description
height	The height of the widget in pixels
width	The width of the widget in pixels
visible	If true, the widget and its child widgets are visible
x	The x-coordinate of the widget relative to its parent widget
y	The y-coordinate of the widget relative to its parent widget
fillColor	The color that fills the ellipse
arcWidth	The width of the arc of the ellipse
centralAngle	The angle in degrees which defines a sector of the ellipse
sectorRotation	The angle in degrees which describes the rotation of the ellipse's sector

#### 15.10.2.5. Image

An image places a picture into a view.

Table 15.155. Properties of the Image widget

Property name	Description
height	The height of the widget in pixels
width	The width of the widget in pixels
visible	If true, the widget and its child widgets are visible
x	The x-coordinate of the widget relative to its parent widget



Property name	Description
y	The y-coordinate of the widget relative to its parent widget
image	The image the widget displays
sRGB	<p>If this property is enabled, the image that is selected in <code>image</code>, is rendered using sRGB color space.</p> <p>Note that to use sRGB functionality, in the project center under <b>Configure &gt; Profiles</b> for the <code>colorMode</code> property select 32-bit sRGB (4) or 32-bit sRGB (Emulated) (5).</p>
horizontalAlign	The horizontal alignment of the image file within the boundaries of the widget
verticalAlign	The vertical alignment of the image file within the boundaries of the widget

#### NOTE



#### Supported image file types

The available image formats depend on the implementation of the renderer. The renderers for OpenGL ES 2.0 or higher support `.png` files and `.jpg` files.

### 15.10.2.6. Instantiator

An instantiator creates widget instances during run-time. You can use the instantiator to model lists or tables with dynamic or static content. The child widgets of an instantiator serve as line templates for the list or table which is created during run-time. By default the instantiator only instantiates the first line template.

Table 15.156. Properties of the Instantiator widget

Property name	Description
height	The height of the widget in pixels
width	The width of the widget in pixels
visible	If true the widget and its child widgets are visible
x	The x-coordinate of the widget relative to its parent widget
y	The y-coordinate of the widget relative to its parent widget
numItems	The number of instantiated child widgets. If <code>numItems</code> is 0, no child widgets are created.
lineMapping	Defines which child widget is the line template for which line, i.e. defines the order of instantiation

### 15.10.2.7. Label

A label places text into a view.

#### NOTE



#### Character replacement

When you enter a text to the `text` property of a label, the following characters are replaced:

- ▶ The sequence `\\\\` is replaced by `\\`.
- ▶ The sequence `\\n` is replaced by `\n`.
- ▶ In case the text is displayed in one line, `\n` is replaced by a space character.

Table 15.157. Properties of the Label widget

Property name	Description
<code>height</code>	The height of the widget in pixels
<code>width</code>	The width of the widget in pixels
<code>visible</code>	If true, the widget and its child widgets are visible
<code>x</code>	The x-coordinate of the widget relative to its parent widget
<code>y</code>	The y-coordinate of the widget relative to its parent widget
<code>text</code>	The text the label displays. If the text does not fit into the widget area it is truncated at the end by default.
<code>textColor</code>	The color in which the text is displayed
<code>font</code>	The font in which the text is displayed
<code>horizontalAlign</code>	The horizontal alignment of the text within the boundaries of the label
<code>verticalAlign</code>	The vertical alignment of the text within the boundaries of the label

### 15.10.2.8. Rectangle

A rectangle draws a colored rectangle with the dimensions and coordinates of the widget into a view.

Table 15.158. Properties of the Rectangle widget

Property name	Description
<code>height</code>	The height of the widget in pixels
<code>width</code>	The width of the widget in pixels
<code>visible</code>	If true, the widget and its child widgets are visible
<code>x</code>	The x-coordinate of the widget relative to its parent widget
<code>y</code>	The y-coordinate of the widget relative to its parent widget
<code>fillColor</code>	The color that fills the rectangle

## 15.10.3. 3D widgets

### 15.10.3.1. Ambient light

An ambient light is a light that uniformly illuminates the scene. An ambient light affects the `ambient` color property of material, PBR GGX material, and PBR Phong material.

Table 15.159. Properties of the Ambient light widget

Property name	Description
<code>enabled</code>	If true, the widget is enabled
<code>color</code>	The color of the light
<code>intensity</code>	The intensity of the light, with the lower limit value <code>0.0</code> as no ambient light

### 15.10.3.2. Camera

A camera defines the view of the scene from a particular point of view. Use several cameras to show the scene from different points of view.

Table 15.160. Properties of the Camera widget

Property name	Description
<code>enabled</code>	If true, the widget is enabled
<code>nearPlane</code>	The nearest distance from the camera in view direction at which the scene becomes visible. The measurement unit is defined when you create a 3D model in third-party 3D modeling software.
<code>farPlane</code>	The farthest distance from the camera in view direction up to which the scene is visible. The measurement unit is defined when you create a 3D model in third-party 3D modeling software.
<code>fieldOfView</code>	The camera's vertical viewing angle in degrees, with the maximum value of 180
<code>projectionType</code>	<p>Defines the projection type of the camera. The objects are rendered either with <code>perspective</code> (0) or <code>orthographic</code> (1) projection.</p> <p>If the projection type is orthographic, the viewing volume is calculated by using the <code>fieldOfView</code> angle.</p>

### 15.10.3.3. Directional light

A directional light illuminates the scene from one direction.

Table 15.161. Properties of the Directional light widget

Property name	Description
enabled	If true, the widget is enabled
color	The light's color
intensity	The intensity of the light, with the lower limit value 0.0 as no directional light

#### 15.10.3.4. Image-based light

An image-based light is a light that illuminates the scene by lighting information of the real world that was stored in a .pfm or .hdr file. The .pfm or .hdr files serve as input data for the IBLGenerator to create an .ebibl file.

Table 15.162. Properties of the Image-based light widget

Property name	Description
enabled	If true, the widget is enabled
ibl	The IBL file .ebibl created manually.
intensity	The intensity of the light, with 0.0 as no image-based light
environmentMap	Defines whether the IBL environment map is visible or not.

#### 15.10.3.5. Material

A material defines the visual appearance of the mesh surface using the Phong reflection model.

Table 15.163. Properties of the Material widget

Property name	Description
ambient	The color that the object reflects when it is illuminated by ambient light. If no ambient light is added to the parent scene graph, this property has no effect.
diffuse	The color that the object reflects evenly in all directions when it is illuminated by pure white light. If the <b>Diffuse texture</b> widget feature is added, this property has no effect.
emissive	The self-illumination color of the object. If the <b>Emissive texture</b> widget feature is added, this property has no effect.
shininess	<p>The shininess factor</p> <p>Note that only values between 0.0 and 1.0, as for example 0.3, are valid.</p> <p>When the <b>Shininess texture</b> widget feature is used, the shininess property is ignored.</p>

Property name	Description
<code>specular</code>	The color that an object with a shiny surface reflects. If the <b>Specular texture</b> widget feature is added or the <code>shininess</code> property is set to <code>0.0</code> , the <code>specular</code> property has no effect.
<code>opacity</code>	The opacity value  Note that only values between <code>0.0</code> and <code>1.0</code> , as for example <code>0.3</code> , are valid.

### 15.10.3.6. Mesh

A mesh defines the shape of the 3D object.

Table 15.164. Properties of the Mesh widget

Property name	Description
<code>visible</code>	If true, the widget and its child widgets are visible
<code>mesh</code>	The automatically created mesh file <code>*.ebmesh</code>
<code>culling</code>	Defines whether no triangles (0), only front-facing triangles (1), or only back-facing triangles (2) are culled from the mesh

### 15.10.3.7. PBR GGX material

A PBR GGX material defines the visual appearance of the mesh surface using the physically correct Cook-Torrance model.

Table 15.165. Properties of the PBR GGX material widget

Property name	Description
<code>ambient</code>	The color that the object reflects when it is illuminated by ambient light. If the <b>Ambient texture</b> widget feature is added, this property has no effect.
<code>baseColor</code>	The color that the object reflects evenly in all directions when it is illuminated by pure white light. If the <b>Base color texture</b> widget feature is added, this property has no effect.
<code>emissive</code>	The self-illumination color of the object. If the <b>Emissive texture</b> widget feature is added, this property has no effect.
<code>reflectance</code>	The color that an object with a shiny surface reflects. If the <b>Reflectance texture</b> widget feature is added or the <code>shininess</code> property is set to <code>0.0</code> , the <code>reflectance</code> property has no effect.
<code>metallic</code>	The value for the surface quality of being metallic  This value interpolates between the base color and the reflectance contribution.

Property name	Description
	Note that only values between 0 and 1 are valid, as for example 0.3.
roughness	<p>The value for the surface quality of being rough</p> <p>This value controls the surface's microstructure.</p> <p>Note that only values between 0 and 1 are valid, as for example 0.3.</p>
opacity	<p>The opacity value</p> <p>Note that only values between 0 and 1 are valid, as for example 0.3.</p>

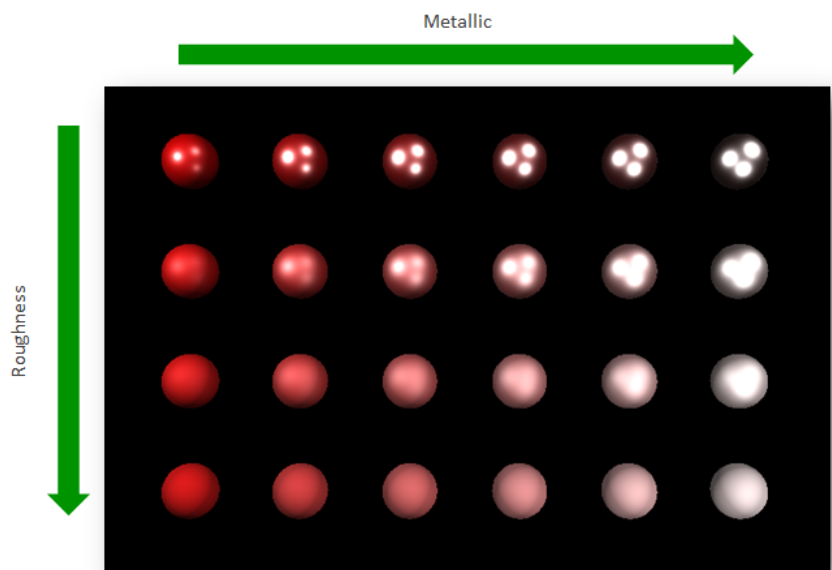


Figure 15.1. Example for a physically-based material

### 15.10.3.8. PBR Phong material

A PBR Phong material defines the visual appearance of the surface of the mesh using the physically correct Phong reflection model.

Table 15.166. Properties of the PBR Phong material widget

Property name	Description
ambient	The color that the object reflects when it is illuminated by ambient light. If the <b>Ambient texture</b> widget feature is added, this property has no effect.
diffuse	The color that the object reflects evenly in all directions when it is illuminated by pure white light. If the <b>Diffuse texture</b> widget feature is added, this property has no effect.

Property name	Description
<code>emissive</code>	The self-illumination color of the object. If the <b>Emissive texture</b> widget feature is added, this property has no effect.
<code>shininess</code>	The shininess factor
<code>specular</code>	The color that an object with a shiny surface reflects. If the <b>Specular texture</b> widget feature is added or the <code>shininess</code> property is set to <code>0.0</code> , the <code>specular</code> property has no effect.
<code>metallic</code>	The value for the surface quality of being metallic  This value interpolates between the diffuse and the specular contribution.  Note that only values between <code>0</code> and <code>1</code> are valid, as for example <code>0.3</code> .
<code>opacity</code>	The opacity value  Note that only values between <code>0</code> and <code>1</code> are valid , as for example <code>0.3</code> .

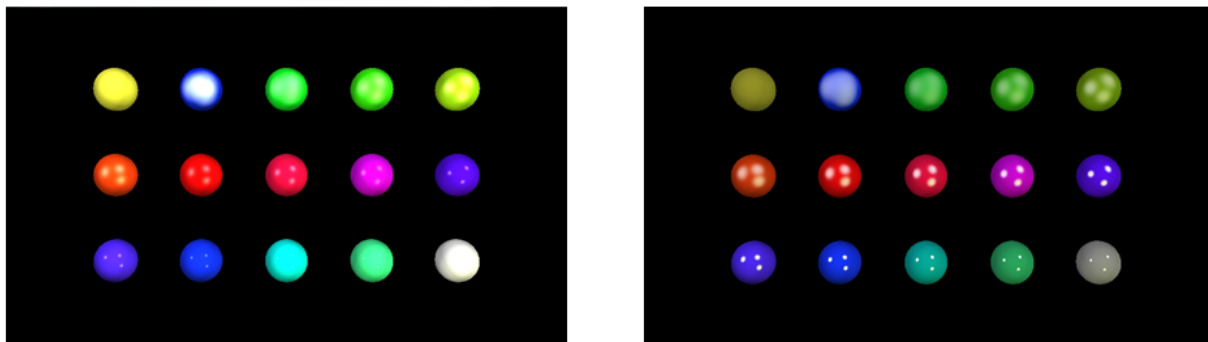


Figure 15.2. Example for a non-normalized material (left) and a normalized material (right)

### 15.10.3.9. Point light

A point light adds a light to the scene that emits light in all directions like a light bulb.

Table 15.167. Properties of the Point light widget

Property name	Description
<code>enabled</code>	If true, the widget is enabled
<code>color</code>	The light's color
<code>intensity</code>	The intensity of the light, with the lower limit value <code>0.0</code> as no point light and the upper limit value depending on attenuation factors

Property name	Description
attenuationConstant	The constant factor by which the light intensity weakens with increasing distance. The 0.0 value means that the factor is not used.
attenuationLinear	The linear factor by which the light intensity weakens with increasing distance. The 0.0 value means that the factor is not used.
attenuationQuadratic	The quadratic factor by which the light intensity weakens with increasing distance. The 0.0 value means that the factor is not used.

### 15.10.3.10. Scene graph

A scene graph places a 3D object into a view.

Table 15.168. Properties of the Scene graph widget

Property name	Description
visible	If true, the widget and its child widgets are visible
width	The width of the widget in pixels
height	The height of the widget in pixels
x	The x-coordinate of the widget relative to its parent widget
y	The y-coordinate of the widget relative to its parent widget
gamma	Corrects the luminance output of the scene graph. The default value is set to 2.-2.

### 15.10.3.11. Scene graph node

A scene graph node is a child node and is added to the scene graph or to another scene graph node. You use scene graph nodes to place 3D widgets in the 3D scene with transformation properties. You can add the following 3D widgets to the scene graph node:

- ▶ Ambient light
- ▶ Camera
- ▶ Directional light
- ▶ Image-based light
- ▶ Mesh
- ▶ Point light
- ▶ Spot light



Table 15.169. Properties of the Scene graph node widget

Property name	Description
visible	If true, the widget and its child widgets are visible
rotationX	The rotation around the x-axis
rotationY	The rotation around the y-axis
rotationZ	The rotation around the z-axis
scalingX	The scaling along the x-axis
scalingY	The scaling along the y-axis
scalingZ	The scaling along the z-axis
translationX	The translation along the x-axis
translationY	The translation along the y-axis
translationZ	The translation along the z-axis

### 15.10.3.12. Spot light

A spot light adds a light which restricts illumination to a cone of influence.

Table 15.170. Properties of the Spot light widget

Property name	Description
enabled	If true, the widget is enabled
color	The light's color
intensity	The intensity of the light, with the lower limit value 0.0 as no spot light and the upper limit value depending on attenuation factors
attenuationConstant	The constant factor by which the light intensity weakens with increasing distance
attenuationLinear	The linear factor by which the light intensity weakens with increasing distance
attenuationQuadratic	The quadratic factor by which the light intensity weakens with increasing distance
coneAngleInner	The light's inner cone angle in degrees, with the maximum value of 180
coneAngleOuter	The light's outer cone angle in degrees, with the maximum value of 180

## 15.11. Widget features

The following list contains a description of all widget features that are implemented, with a brief description on how to use them in an EB GUIDE model.

## 15.11.1. Common

### 15.11.1.1. Child visibility selection

The **Child visibility selection** widget feature handles the visibility of child widgets. You can define a single widget to be visible or you can define groups of child widgets to be visible at the same time. To define groups, map the index of child widgets to the same group value.

Table 15.171. Properties of the **Child visibility selection** widget feature

Property name	Description	Set by EB GUIDE GTF
containerIndex	<p>Controls the visibility of child widgets.</p> <p>If <code>containerMapping</code> is not filled, <code>containerIndex</code> makes a single child widget visible. The child widget that is visible is identified by its order in the widget tree. The topmost child has <code>containerIndex 0</code>, next <code>containerIndex 1</code> etc.</p> <p>If <code>containerMapping</code> is filled, <code>containerIndex</code> refers to a group of child widgets. Define the group in <code>containerMapping</code>.</p>	no
containerMapping	<p>Use this property to create groups of child widgets. The <b>Index</b> column identifies the child widget. The <b>Value</b> column defines the group.</p> <p>The number of rows must match the number of child widgets. Otherwise the mapping is not used.</p>	no

### 15.11.1.2. Enabled

The **Enabled** widget feature adds an `enabled` property to a widget.

Table 15.172. Properties of the **Enabled** widget feature

Property name	Description	Set by EB GUIDE GTF
enabled	If true, the widget reacts on touch and press input	no

### 15.11.1.3. Focused

The **Focused** widget feature enables a widget to have input focus.

Table 15.173. Properties of the **Focused** widget feature

Property name	Description	Set by EB GUIDE GTF
<code>focusable</code>	Defines whether the widget receives the focus or not. Possible values: <ul style="list-style-type: none"><li>▶ <code>not focusable (0)</code></li><li>▶ <code>only by touch (1)</code></li><li>▶ <code>only by key (2)</code></li><li>▶ <code>focusable (3)</code></li></ul>	no
<code>focused</code>	If true, the widget has focus	yes

#### 15.11.1.4. Font metrics

With the **Font metrics** widget feature, you can change settings of a font that is used in a label.

For instructions on how to change the `lineGap`, see [section 8.5.3, “Changing the line spacing”](#).

Restrictions:

- ▶ The **Font metrics** widget feature is only available for the Label widget.
- ▶ If the label has a multifont support added, the **Font metrics** widget feature overwrites the baseline value of the default font.

Table 15.174. Properties of the **Font metrics** widget feature

Property name	Description	Set by EB GUIDE GTF
<code>ascender</code>	The portion of a letter that extends above the baseline of a font.	no
<code>descender</code>	The portion of a letter that extends below the baseline of a font.	no
<code>lineGap</code>	The line spacing that is contained by default in every font. A positive value increases the spacing, a negative value decreases the spacing.	no

#### 15.11.1.5. Multiple lines

The **Multiple lines** widget feature enables line breaks. The line break is set between words or characters depending on the `width` property that is set for the label widget. To mark the end of a line, you can also use the hard line break character `\n`.

Restrictions:

- ▶ The **Multiple lines** widget feature is only available for the Label widget.

Table 15.175. Properties of the **Multiple lines** widget feature

Property name	Description	Set by EB GUIDE GTF
<code>lineOffset</code>	The size of the spacing between the lines. A positive value increases the spacing, a negative value decreases the spacing.  When the <code>lineOffset</code> is too small (high negative value), it has no effect anymore and the text is rendered in one line. This occurs for example, when the font style is set to <code>PT_Sans_Narrow</code> , size is set to 30 and the <code>lineOffset</code> is defined as -50.	no
<code>maxLineCount</code>	The maximum number of visible lines. 0 = no limitation	no

**TIP**



**Number of lines used**

With the script function `getLineCount`, you can obtain the number of lines of the text.

For more information on this, see [section 15.4.3.2.33, “getLineCount”](#).

**NOTE**



**Character replacement**

When you enter a text to the `text` property of a label, the following characters are replaced:

- ▶ The sequence `\\\\` is replaced by `\\`.
- ▶ The sequence `\\n` is replaced by `\n`.
- ▶ In case the text is displayed in one line, `\n` is replaced by a space character.

### 15.11.1.6. Pressed

The **Pressed** widget feature defines that a widget can be pressed.

Restrictions:

- ▶ Adding the **Pressed** widget feature automatically adds the **Focused** widget feature.

Table 15.176. Properties of the **Pressed** widget feature

Property name	Description	Set by EB GUIDE GTF
<code>pressed</code>	If true, a key is pressed while the widget is focused	yes

Combining the **Touched** widget feature with the **Touch pressed** widget feature allows modeling a push button.

#### 15.11.1.7. Selected

The **Selected** widget feature adds a `selected` property to a widget. It is typically set by the application or the HMI modeler. It is not changed by any other component of the framework.

Table 15.177. Properties of the **Selected** widget feature

Property name	Description	Set by EB GUIDE GTF
<code>selected</code>	If true, the widget is selected.  When the <b>Selection group</b> widget feature is added, it evaluates if <code>buttonID</code> and <code>buttonValue</code> are identical. If true, the button is selected.	no  Note: If the <b>Selection group</b> widget feature is added, the property is automatically set by EB GUIDE GTF.

#### 15.11.1.8. Selection group

The **Selection group** widget feature is used to model a list of radio buttons. In the list, every radio button has the **Selection group** widget feature and a unique button ID.

Use a datapool item for the `buttonValue` property. Assign the datapool item to all widgets in the radio button array.

Selecting and deselecting a widget within the button group can be done by an application that sets the `buttonValue` property. Alternatively, changes can be triggered by touch or key input as well as by adding a condition that sets the button value.

Restrictions:

- ▶ Adding the **Selection group** widget feature automatically adds the **Selected** widget feature.

Table 15.178. Properties of the **Selection group** widget feature

Property name	Description	Set by EB GUIDE GTF
<code>buttonId</code>	The ID that identifies a button within a button group	no

Property name	Description	Set by EB GUIDE GTF
buttonValue	The current value of a button. If this value matches the <code>buttonId</code> , the button is selected.	no

### 15.11.1.9. Spinning

The **Spinning** widget feature turns a widget into a rotary button. A widget with the **Spinning** widget feature reacts to increment and decrement events by changing an internal value. The **Spinning** widget feature can be used to create a scale, a progress bar, or a widget with a preview value.

Table 15.179. Properties of the **Spinning** widget feature

Property name	Description	Set by EB GUIDE GTF
currentValue	The current rotary value	yes
maxValue	The maximum value for the <code>currentValue</code> property	no
minValue	The minimum value for the <code>currentValue</code> property	no
incValueTrigger	If true, the <code>currentValue</code> property is incremented by 1	no
incValueReaction	The reaction to an incrementation of the <code>currentValue</code> property	no
decValueTrigger	If true, the current value is decremented by 1	no
decValueReaction	Reaction to a decrementation of the <code>currentValue</code> property	no
steps	The number of steps to calculate the increment or decrement for the <code>currentValue</code> property	no
valueWrapAround	Possible values: <ul style="list-style-type: none"> <li>▶ <b>true:</b> The <code>currentValue</code> property continues at the inverse border, if <code>minValue</code> or <code>maxValue</code> is exceeded.</li> <li>▶ <b>false:</b> The <code>currentValue</code> property does not decrease/increase, if <code>minValue</code> or <code>maxValue</code> is exceeded.</li> </ul>	no

### 15.11.1.10. Text truncation

The **Text truncation** widget feature truncates the content of the `text` property if it does not fit into the widget area. The widget feature enables a different truncation than the default setting `trailing`.

Restrictions:

- The **Text truncation** widget feature is only available for the Label widget.

Table 15.180. Properties of the **Text truncation** widget feature

Property name	Description	Set by EB GUIDE GTF
<code>truncationSymbol</code>	<p>The string that is shown instead of the text part. The default truncation symbol is . . . .</p> <p>If the label has a multifont support added, the truncation symbol will use the font and size as specified for its Unicode character range.</p>	no
<code>truncationPolicy</code>	<p>For single-line texts, the <code>truncationPolicy</code> property defines the position of the truncation. Possible values:</p> <ul style="list-style-type: none"> <li>► <code>leading (0)</code>: Text is replaced at the beginning of the text</li> <li>► <code>trailing (1)</code>: Text is replaced at the end of the text</li> </ul> <p>For multi-line texts, the <code>truncationPolicy</code> property defines where text is replaced. Possible values:</p> <ul style="list-style-type: none"> <li>► <code>leading (0)</code>: Lines at the beginning are replaced and text of the first visible line is truncated at the beginning of the text.</li> <li>► <code>trailing (1)</code>: Lines at the end are replaced and text of the last visible line is truncated at the end of the text.</li> </ul>	no

#### 15.11.1.11. Touched

The **Touched** widget feature enables a widget to react to touch input.

Table 15.181. Properties of the **Touched** widget feature

Property name	Description	Set by EB GUIDE GTF
<code>touchable</code>	If true, the widget reacts on touch input	no
<code>touched</code>	If true, the widget is currently touched	yes
<code>touchPolicy</code>	<p>Defines how to handle touch and movement that crosses widget boundaries. Possible values:</p> <ul style="list-style-type: none"> <li>► <code>press then react (0)</code>: Press first, then the widget reacts. Notifications of moving and releasing are only active within the widget area.</li> </ul>	no

Property name	Description	Set by EB GUIDE GTF
	<ul style="list-style-type: none"> <li>▶ <code>press and grab (1)</code>: Press to grab the contact. The contact remains grabbed even if it moves away from the widget area.</li> <li>▶ <code>press then react on contact (2)</code>: Even if the contact enters the pressed state outside the widget boundaries, the subsequent move and release events are delivered to the widget.</li> </ul>	
<code>touchBehavior</code>	<p>Defines touch evaluation. Possible values:</p> <ul style="list-style-type: none"> <li>▶ <code>whole area (0)</code>: To identify the touched widget, the renderer evaluates the widget's clipping rectangle.</li> <li>▶ <code>visible pixels (1)</code>: To identify the touched widget, the renderer evaluates the widget the touched pixel belongs to.</li> </ul> <p>Transparent pixels in an image with alpha transparency or pixels inside letters such as in O or A are not touchable.</p> <p>Note that the <code>Visible pixels</code> value has no effect on labels.</p>	no

Combining the **Touched** widget feature with the **Pressed** widget feature allows modeling a push button.

#### TIP



#### Performance recommendation

If performance is an important issue in your project, set the `touchBehavior` property to `whole area (0)`. EB GUIDE GTF evaluates `whole area (0)` faster than `visible pixels (1)`.

## 15.11.2. Effect

### 15.11.2.1. Border

The **Border** widget feature adds a configurable border to the widget. The border starts at the widget boundaries and is placed within the widget.

Restrictions:

- ▶ The **Border** widget feature is only available for the Rectangle widget.



Table 15.182. Properties of the **Border** widget feature

Property name	Description	Set by EB GUIDE GTF
<code>borderThickness</code>	The thickness of the border in pixels	no
<code>borderColor</code>	The color that is used to render the border	no
<code>borderStyle</code>	The style that is used to render the border	no

### 15.11.2.2. Coloration

The **Coloration** widget feature colors the widget and its widget subtree. It also affects transparency if the alpha value is not opaque.



#### Example 15.3. Usage of the Coloration widget feature

For all colors with RGBA components between 0.0 and 1.0, the algorithm in the **Coloration** widget feature multiplies the current color values of a widget by the `colorationColor` property value. Multiplication is done per pixel and component-wise.

A semi-transparent gray colored by an opaque blue results in semi-transparent darker blue as follows:

$$(0.5, 0.5, 0.5, 0.5) * (0.0, 0.0, 1.0, 1.0) = (0.0, 0.0, 0.5, 0.5)$$

Table 15.183. Properties of the **Coloration** widget feature

Property name	Description	Set by EB GUIDE GTF
<code>colorationEnabled</code>	If true, coloration is used	no
<code>colorationColor</code>	The color used for the coloration	no

### 15.11.2.3. Circular text

The **Circular text** widget feature adds a text effect to a Label widget to place text along a circle curved path.

Restrictions:

- ▶ Clipping of the text on widget boundaries is disabled when widget features **Circular text** and **Stroke** are both enabled. Use widget feature **Viewport** as a workaround.

- ▶ Only Latin script fonts are supported.
- ▶ **Circular text** cannot be used in combination with `verticalAlign`.
- ▶ **Circular text** cannot be used in combination with **Multiple lines**.
- ▶ **Circular text** cannot be used in combination with **Font metrics**.

If you want your text to follow a path of any arbitrary shaped curve, you can develop your own widget feature with the EB GUIDE SDK.

Table 15.184. Properties of the **Circular text** widget feature

Property name	Description	Set by EB GUIDE GTF
<code>enabled</code>	If true, circular text is used	no
<code>centerX</code>	The x-coordinate of the circle center	no
<code>centerY</code>	The y-coordinate of the circle center	no
<code>radius</code>	The radius of the circle	no
<code>direction</code>	Define the text direction along the circle. Possible values: <ul style="list-style-type: none"> <li>▶ <code>clockwise</code> (0)</li> <li>▶ <code>counterclockwise</code> (1)</li> </ul>	no
<code>startingAngle</code>	The starting angle of the text along the circular path	no
<code>characterSpacing</code>	Increase or decrease the character spacing for all characters	no

#### 15.11.2.4. Stroke

The **Stroke** widget feature activates a configurable text outline, i.e. a label border.

Restrictions:

- ▶ The **Stroke** widget feature is only available for the Label widget.

Table 15.185. Properties of the **Stroke** widget feature

Property name	Description	Set by EB GUIDE GTF
<code>strokeEnabled</code>	If true, stroke is used	no
<code>strokeThickness</code>	The thickness of the outline in pixels	no
<code>strokeColor</code>	The color that is used to render the outline	no

## 15.11.3. Focus

The **Focus** widget feature category provides the widget features relating to focus management.

### 15.11.3.1. Auto focus

With the **Auto focus** widget feature, the order in which child widgets are focused is pre-defined. The **Auto focus** widget feature checks the widget subtree for child widgets with the `focusable` property.

The order of the widgets in the layout is used to calculate focus order. Depending on layout orientation, the algorithm begins in the upper left or upper right corner.

Restrictions:

- ▶ The widget feature **Auto focus** automatically adds the **Focused** widget feature.

Table 15.186. Properties of the **Auto focus** widget feature

Property name	Description	Set by EB GUIDE GTF
<code>focusNext</code>	The condition on which the focus index is incremented	no
<code>focusPrev</code>	The condition on which the focus index is decremented	no
<code>focusFlow</code>	The behavior for focus changes within the hierarchy. Possible values: <ul style="list-style-type: none"><li>▶ <code>stop at hierarchy (0)</code></li><li>▶ <code>wrap within hierarchy level (1)</code></li><li>▶ <code>step up in hierarchy (2)</code></li></ul>	no
<code>focusedIndex</code>	The index of the currently focused child widget as the n-th child widget which is focusable	yes
<code>initFocus</code>	The index defines the focused child widget at initialization. If the widget is not focusable, the next focusable child is used.	no

### 15.11.3.2. User-defined focus

The **User-defined focus** widget feature enables additional focus functionality for the widget. A widget that uses the feature manages a local focus hierarchy for its widget subtree.

Restrictions:

- ▶ The widget feature **User-defined focus** automatically adds the **Focused** widget feature.

Table 15.187. Properties of the **User-defined focus** widget feature

Property name	Description	Set by EB GUIDE GTF
<code>focusNext</code>	The trigger that assigns the focus to the next child widget	no
<code>focusOrder</code>	<p>The <code>focusOrder</code> property makes it possible to skip child widgets when assigning focus. The ID of a child widget corresponds to its position in the subtree. Child widgets that are not focusable are skipped by default. Order in which the child widgets are focused:</p> <ul style="list-style-type: none"> <li>▶ defined: User-defined widget order is used</li> <li>▶ not defined: Default widget order is used instead</li> </ul> <p>Each child widget requires the <b>Focused</b> widget feature, otherwise widgets are ignored for focus management. Example: <code>focusOrder=1 0 2</code> means the second widget receives focus first, then the first widget receives focus, and finally the third widget.</p>	no
<code>focusPrev</code>	The trigger that assigns the focus to the previous child	no
<code>focusFlow</code>	<p>The behavior for focus changes within the hierarchy. Possible values:</p> <ul style="list-style-type: none"> <li>▶ <code>stop at hierarchy level (0)</code></li> <li>▶ <code>wrap within hierarchy level (1)</code></li> <li>▶ <code>step up in hierarchy (2)</code></li> </ul>	no
<code>focusedIndex</code>	The index defines the position of the child widget in the <code>focusOrder</code> list. If the widget is not focusable, the child next in the list is used.	yes
<code>initFocus</code>	The index of the focused child widget at initialization	no

## 15.11.4. Gestures

### 15.11.4.1. Flick gesture

A quick brush of a contact over a surface

Restrictions:

- ▶ Adding the **Flick gesture** widget feature automatically adds the **Gestures** and **Touched** widget features.

Table 15.188. Properties of the **Flick gesture** widget feature

Property name	Description	Set by EB GUIDE GTF
<code>onGestureFlick</code>	<p>The reaction that is triggered once the gesture is recognized</p> <p>Reaction arguments:</p> <ul style="list-style-type: none"> <li>▶ <code>speed</code>: relative speed of the flick gesture</li> </ul> <p>Speed in pixels/ms divided by <code>flickMinLength/flickMaxTime</code></p> <ul style="list-style-type: none"> <li>▶ <code>directionX</code>: x-part of the direction vector of the gesture</li> <li>▶ <code>directionY</code>: y-part of the direction vector of the gesture</li> </ul>	yes
<code>flickMaxTime</code>	The maximal time in milliseconds the contact may stay in place for the gesture to be recognized as a flick gesture	no
<code>flickMinLength</code>	The minimal distance in pixels a contact has to move on the surface to be recognized as a flick gesture	no

#### 15.11.4.2. Hold gesture

A hold gesture without movement

Restrictions:

- ▶ Adding the **Hold gesture** widget feature automatically adds the **Gestures** and **Touched** widget features.
- ▶ The **Hold gesture** widget feature does not trigger the **Touch lost** widget feature.

Table 15.189. Properties of the **Hold gesture** widget feature

Property name	Description	Set by EB GUIDE GTF
<code>onGestureHold</code>	<p>The reaction that is triggered once the gesture is recognized. The reaction is triggered only once per contact: when <code>holdDuration</code> is expired and the contact still is in a small boundary box around the initial touch position.</p> <p>Reaction arguments:</p> <ul style="list-style-type: none"> <li>▶ <code>x</code>: x-coordinate of the contact position</li> <li>▶ <code>y</code>: y-coordinate of the contact position</li> </ul>	yes
<code>holdDuration</code>	The minimal time in milliseconds the contact must stay in place for the gesture to be recognized as a hold gesture	no

### 15.11.4.3. Long hold gesture

A long hold gesture without movement

Restrictions:

- ▶ Adding the **Long hold gesture** widget feature automatically adds the **Gestures** and **Touched** widget features.
- ▶ The **Long hold gesture** widget feature does not trigger the **Touch lost** widget feature.

Table 15.190. Properties of the **Long hold gesture** widget feature

Property name	Description	Set by EB GUIDE GTF
onGestureLong-Hold	<p>The reaction that is triggered once the gesture is recognized. The reaction is triggered only once per contact: when <code>longHoldDuration</code> has expired and the contact still is in a small boundary box around the initial touch position.</p> <p>Reaction arguments:</p> <ul style="list-style-type: none"> <li>▶ <code>x</code>: x-coordinate of the contact position</li> <li>▶ <code>y</code>: y-coordinate of the contact position</li> </ul>	yes
longHoldDuration	The minimal time in milliseconds the contact must stay in place for the gesture to be recognized as a long hold gesture	no

### 15.11.4.4. Path gestures

A shape drawn by one contact is matched against a set of known shapes.

Restrictions:

- ▶ Adding the **Path gesture** widget feature automatically adds the **Gestures** and **Touched** widget features.

Table 15.191. Properties of the **Path gesture** widget feature





Property name	Description	Set by EB GUIDE GTF
onPath	<p>The reaction that is triggered when the entered shape matches. The reaction is only triggered if <code>onPathStart</code> has been triggered already. Reaction argument:</p> <ul style="list-style-type: none"> <li>▶ <code>gestureId</code>: ID of the path that was matched</li> </ul>	yes
onPathStart	The reaction that is triggered once a contact moves beyond the minimal box ( <code>pathMinXBox</code> , <code>pathMinYBox</code> ).	yes




Property name	Description	Set by EB GUIDE GTF
<code>onPathNotRecognized</code>	The reaction that triggered when the entered shape does not match. The reaction is only triggered if <code>onPathStart</code> has been triggered already.	yes
<code>pathMinXBox</code>	The x-coordinate of the minimal distance in pixels a contact must move so that the path gesture recognizer starts considering the input	no
<code>pathMinYBox</code>	The y-coordinate of the minimal distance in pixels a contact must move so that the path gesture recognizer starts considering the input	no

#### 15.11.4.4.1. Gesture IDs

Gesture identifiers depend on the configuration of the path gesture recognizer. The following table shows an example configuration which is included in EB GUIDE.

Table 15.192. Path gesture samples configuration included in EB GUIDE

ID	Shape	Description
0		Roof shape left to right
1		Roof shape right to left
2		Horizontal line left to right
3		Horizontal line right to left

ID	Shape	Description
4		Check mark
5		Wave shape left to right
6		Wave shape right to left

#### 15.11.4.5. Pinch gesture

Two contacts that move closer together or further apart

Restrictions:

- ▶ Adding the **Pinch gesture** widget feature automatically adds the **Gestures** and **Touched** widget features.

Table 15.193. Properties of the **Pinch gesture** widget feature

Property name	Description	Set by EB GUIDE GTF
onGesture-PinchStart	<p>The reaction that is triggered once the start of the gesture is recognized. Reaction arguments:</p> <ul style="list-style-type: none"> <li>▶ <b>ratio</b>: Current contact distance to initial contact distance ratio</li> <li>▶ <b>centerX</b>: x-coordinate of the current center point between the two contacts</li> <li>▶ <b>centerY</b>: y-coordinate of the current center point between the two contacts</li> </ul>	yes
onGesture-PinchUpdate	<p>The reaction that is triggered when the pinch ratio or center point change. Reaction arguments:</p> <ul style="list-style-type: none"> <li>▶ <b>ratio</b>: Current contact distance to initial contact distance ratio</li> </ul>	yes



Property name	Description	Set by EB GUIDE GTF
	<ul style="list-style-type: none"> <li>▶ <code>centerX</code>: x-coordinate of the current center point between the two contacts</li> <li>▶ <code>centerY</code>: y-coordinate of the current center point between the two contacts</li> </ul>	
<code>onGesture-PinchEnd</code>	<p>The reaction that is triggered once the gesture is finished. Reaction arguments:</p> <ul style="list-style-type: none"> <li>▶ <code>ratio</code>: Current contact distance to initial contact distance ratio</li> <li>▶ <code>centerX</code>: x-coordinate of the current center point between the two contacts</li> <li>▶ <code>centerY</code>: y-coordinate of the current center point between the two contacts</li> </ul>	yes
<code>pinchThreshold</code>	The minimal distance in pixels each contact has to move from its initial position for the gesture to be recognized	no

#### 15.11.4.6. Rotate gesture

Two contacts that move along a circle

Restrictions:

- ▶ Adding the **Rotate gesture** widget feature automatically adds the **Gestures** and **Touched** widget features.

Table 15.194. Properties of the **Rotate gesture** widget feature

Property name	Description	Set by EB GUIDE GTF
<code>onGestureRotateStart</code>	The reaction that is triggered once the start of the gesture is recognized	yes
<code>onGestureRotateUpdate</code>	The reaction that is triggered when the recognized angle or center point changes	yes
<code>onGestureRotateEnd</code>	The reaction that is triggered once the gesture is finished	yes
<code>rotateThreshold</code>	The minimal distance in pixels each contact has to move from its initial position for the start of the gesture to be recognized	no

Reaction arguments for `onGestureRotateEnd`, `onGestureRotateStart`, and `onGestureRotateUpdate`:

- ▶ **angle**: Angle between the line specified by the initial position of the two involved contacts and the line specified by the current position of the two contacts. The angle is measured counter-clockwise.
- ▶ **centerX**: x-coordinate of the current center point between the two contacts
- ▶ **centerY**: y-coordinate of the current center point between the two contacts

## 15.11.5. Input handling

### 15.11.5.1. Gestures

The **Gestures** widget feature enables the widget to react on touch gestures.

Restrictions:

- ▶ Adding the **Gestures** widget feature automatically adds the **Touched** widget feature.
- ▶ The **Gestures** widget feature has no additional properties.

### 15.11.5.2. Key pressed

The **Key pressed** widget feature enables a widget to react on a key being pressed.

Restrictions:

- ▶ Adding the **Key pressed** widget feature automatically adds the **Pressed** and **Focused** widget features.

Table 15.195. Properties of the **Key pressed** widget feature

Property name	Description	Set by EB GUIDE GTF
keyPressed	The widget's reaction on a key being pressed  Reaction argument: <ul style="list-style-type: none"><li>▶ <b>keyId</b>: The ID of the key that is processed</li></ul>	yes

### 15.11.5.3. Key released

The **Key released** widget feature enables a widget to react on a key being released.

Restrictions:

- ▶ Adding the **Key released** widget feature automatically adds the **Pressed** and **Focused** widget features.

Table 15.196. Properties of the **Key released** widget feature

Property name	Description	Set by EB GUIDE GTF
keyShortReleased	The widget's reaction on a key being released  Reaction argument:  ▶ <code>keyId</code> : The ID of the key that is processed	yes

#### 15.11.5.4. Key status changed

The **Key status changed** widget feature enables a widget to react on a key being pressed or released. It defines the reaction to key input such as **short press**, **long**, **ultra long** and **continuous**.

Restrictions:

- ▶ Adding the **Key status changed** widget feature automatically adds the **Pressed** and **Focused** widget features.

Table 15.197. Properties of the **Key status changed** widget feature

Property name	Description	Set by EB GUIDE GTF
keyStatusChanged	The widget's reaction on a key being pressed or released  Reaction arguments:  ▶ <code>keyId</code> : The ID of the key that is processed  ▶ <code>status</code> : The numeric ID of the status change	yes

#### 15.11.5.5. Key unicode

The **Key unicode** widget feature enables a widget to react on Unicode key input.

Restrictions:

- ▶ Adding the **Key unicode** widget feature automatically adds the **Pressed** and **Focused** widget features.

Table 15.198. Properties of the **Key unicode** widget feature

Property name	Description	Set by EB GUIDE GTF
keyUnicode	The widget's reaction on a Unicode key input	yes

Property name	Description	Set by EB GUIDE GTF
	Reaction argument: ► <code>keyId</code> : The ID of the key that is processed	

### 15.11.5.6. Move in

The **Move in** widget feature enables a widget to react on movement into its boundaries.

Restrictions:

- Adding the **Move in** widget feature automatically adds the **Touched** widget feature.

Table 15.199. Properties of the **Move in** widget feature

Property name	Description	Set by EB GUIDE GTF
<code>moveIn</code>	The widget's reaction on a movement into its boundaries  Reaction arguments: ► <code>touchId</code> : The ID of the touch screen the user has clicked or released ► <code>x</code> : The x-coordinate ► <code>y</code> : The y-coordinate ► <code>fingerId</code> : The ID of the contact that moves across the widget	yes

### 15.11.5.7. Move out

The **Move out** widget feature enables a widget to react on movement out of its boundaries.

Restrictions:

- Adding the **Move out** widget feature automatically adds the **Touched** widget feature.

Table 15.200. Properties of the **Move out** widget feature

Property name	Description	Set by EB GUIDE GTF
<code>moveOut</code>	The widget's reaction on a movement out of its boundaries	yes

Property name	Description	Set by EB GUIDE GTF
	<p>Reaction arguments:</p> <ul style="list-style-type: none"> <li>▶ <code>touchId</code>: The ID of the touch screen the user has clicked or released</li> <li>▶ <code>x</code>: The x-coordinate</li> <li>▶ <code>y</code>: The y-coordinate</li> <li>▶ <code>fingerId</code>: The ID of the contact that moves across the widget</li> </ul>	

#### 15.11.5.8. Move over

The **Move over** widget feature enables a widget to react on movement within its boundaries.

Restrictions:

- ▶ Adding the **Move over** widget feature automatically adds the **Touched** widget feature.

Table 15.201. Properties of the **Move over** widget feature

Property name	Description	Set by EB GUIDE GTF
<code>moveOver</code>	<p>The widget's reaction on a movement within its boundaries</p> <p>Reaction arguments:</p> <ul style="list-style-type: none"> <li>▶ <code>touchId</code>: The ID of the touch screen the user has clicked or released</li> <li>▶ <code>x</code>: The x-coordinate</li> <li>▶ <code>y</code>: The y-coordinate</li> <li>▶ <code>fingerId</code>: The ID of the contact that moves across the widget</li> </ul>	yes

#### 15.11.5.9. Moveable

The **Moveable** widget feature enables a widget to be moved by touch.

Restrictions:

- ▶ Adding the **Moveable** widget feature automatically adds the **Touched** and **Touch moved** widget features.

Table 15.202. Properties of the **Moveable** widget feature

Property name	Description	Set by EB GUIDE GTF
<code>moveDirection</code>	The direction into which the widget moves. Possible values: <ul style="list-style-type: none"><li>▶ <code>horizontal</code> (0)</li><li>▶ <code>vertical</code> (1)</li><li>▶ <code>free</code> (2)</li></ul>	no

#### 15.11.5.10. Rotary

The **Rotary** widget feature enables a widget to react on being rotated.

Restrictions:

- ▶ Adding the **Rotary** widget feature automatically adds the **Focused** widget feature.

Table 15.203. Properties of the **Rotary** widget feature

Property name	Description	Set by EB GUIDE GTF
<code>rotaryReaction</code>	The widget's reaction on being rotated. If true, the widget reacts on an incoming rotary event.  Reaction arguments: <ul style="list-style-type: none"><li>▶ <code>rotaryId</code>: integer ID</li><li>▶ <code>increment</code>: number of units the rotary input shifts when the incoming event is sent</li></ul>	yes

#### 15.11.5.11. Touch lost

The **Touch lost** widget feature enables a widget to react on a lost touch contact.

A contact can disappear when it is part of a gesture or leaves the touch screen without releasing. In these cases the `touchShortReleased` reaction is not executed.

Restrictions:

- ▶ Adding the **Touch lost** widget feature automatically adds the **Touched** widget feature.
- ▶ If you add **Touch lost**, in the `touchPolicy` drop-down box of the **Touched** widget feature, select `press` and `grab` (1).

**Touch lost** does not work with the other touch policies.

Table 15.204. Properties of the **Touch lost** widget feature

Property name	Description	Set by EB GUIDE GTF
<code>onTouchGrabLost</code>	<p>The widget's reaction on a lost touch contact</p> <p>Reaction arguments:</p> <ul style="list-style-type: none"> <li>▶ <code>touchId</code>: The ID of the touch screen the user has clicked or released</li> <li>▶ <code>x</code>: The x-coordinate</li> <li>▶ <code>y</code>: The y-coordinate</li> <li>▶ <code>fingerId</code>: The ID of the contact that moves across the widget</li> </ul>	yes

#### 15.11.5.12. Touch move

The **Touch move** widget feature enables a widget to react on being touched or on being touched and then moved.

Restrictions:

- ▶ Adding the **Touch move** widget feature automatically adds the **Touched** widget feature.

Table 15.205. Properties of the **Touch move** widget feature

Property name	Description	Set by EB GUIDE GTF
<code>touchMoved</code>	<p>The widget's reaction on being touched or on being touched and then moved</p> <p>Reaction arguments:</p> <ul style="list-style-type: none"> <li>▶ <code>touchId</code>: The ID of the touch screen the user has clicked or released</li> <li>▶ <code>x</code>: The x-coordinate</li> <li>▶ <code>y</code>: The y-coordinate</li> </ul>	yes

Property name	Description	Set by EB GUIDE GTF
	<ul style="list-style-type: none"> <li>▶ <code>fingerId</code>: The ID of the contact that moves across the widget</li> </ul>	

### 15.11.5.13. Touch pressed

The **Touch pressed** widget feature enables a widget to react on being pressed.

Restrictions:

- ▶ Adding the **Touch pressed** widget feature automatically adds the **Touched** widget feature.

Table 15.206. Properties of the **Touch pressed** widget feature

Property name	Description	Set by EB GUIDE GTF
<code>touchPressed</code>	<p>The widget's reaction on being pressed</p> <p>Reaction arguments:</p> <ul style="list-style-type: none"> <li>▶ <code>touchId</code>: The ID of the touch screen the user has clicked or released</li> <li>▶ <code>x</code>: The x-coordinate</li> <li>▶ <code>y</code>: The y-coordinate</li> <li>▶ <code>fingerId</code>: The ID of the contact that moves across the widget</li> </ul>	yes

### 15.11.5.14. Touch released

The **Touch released** widget feature enables a widget to react on being released.

Restrictions:

- ▶ Adding the **Touch released** widget feature automatically adds the **Touched** widget feature.

Table 15.207. Properties of the **Touch released** widget feature

Property name	Description	Set by EB GUIDE GTF
<code>touchShortReleased</code>	<p>The widget's reaction on being released</p> <p>Reaction arguments:</p>	yes



Property name	Description	Set by EB GUIDE GTF
	<ul style="list-style-type: none"> <li>▶ <code>touchId</code>: The ID of the touch screen the user has clicked or released</li> <li>▶ <code>x</code>: The x-coordinate</li> <li>▶ <code>y</code>: The y-coordinate</li> <li>▶ <code>fingerId</code>: The ID of the contact that moves across the widget</li> </ul>	

### 15.11.5.15. Touch status changed

The **Touch status changed** widget feature enables a widget to react on changes of its touch status.

Restrictions:

- ▶ Adding the **Touch status changed** widget feature automatically adds the **Touched** widget feature.

Table 15.208. Properties of the **Touch status changed** widget feature

Property name	Description	Set by EB GUIDE GTF
<code>touchStatusChanged</code>	<p>The widget's reaction on changes of its touch status</p> <p>Reaction arguments:</p> <ul style="list-style-type: none"> <li>▶ <code>touchId</code>: The ID of the touch screen the user has clicked or released</li> <li>▶ <code>x</code>: The x-coordinate</li> <li>▶ <code>y</code>: The y-coordinate</li> <li>▶ <code>touchStatus</code>: The ID of the type of touch</li> </ul> <p>Possible values:</p> <ul style="list-style-type: none"> <li>▶ 0: new contact</li> <li>▶ 1: touch press</li> <li>▶ 2: touch move</li> <li>▶ 3: touch released</li> <li>▶ 4: movement without touch</li> <li>▶ 5: touch gone</li> <li>▶ <code>fingerId</code>: The ID of the contact that moves across the widget</li> </ul>	yes

## 15.11.6. Layout

### 15.11.6.1. Absolute layout

The **Absolute layout** widget feature of a parent widget defines the position and size of the child widgets. Invisible child widgets are ignored. The added widget feature properties consist of integer lists. Each list element is mapped to one child widget.

Restrictions:

- ▶ The **Absolute layout** widget feature excludes the following widget features:
  - ▶ **Box layout**
  - ▶ **Flow layout**
  - ▶ **Grid layout**
  - ▶ **List layout**

Table 15.209. Properties of the **Absolute layout** widget feature

Property name	Description	Set by EB GUIDE GTF
<code>itemLeftOffset</code>	An integer list that stores the offset from the left border for the child widgets. Each list element is mapped to a child widget.	no
<code>itemTopOffset</code>	An integer list that stores the offset from the top border for the child widgets. Each list element is mapped to a child widget.	no
<code>itemRightOffset</code>	An integer list that stores the offset from the right border for the child widgets. Each list element is mapped to a child widget.	no
<code>itemBottomOffset</code>	An integer list that stores the offset from the bottom border for the child widgets. Each list element is mapped to a child widget.	no

### 15.11.6.2. Box layout

The **Box layout** widget feature defines position and size of each child widget.

Position and size properties of child widgets are set by the parent widget. Invisible child widgets are ignored in the calculation.

Restrictions:

- ▶ The **Box layout** widget feature excludes the following widget features:
  - ▶ **Absolute layout**
  - ▶ **Flow layout**

- ▶ **Grid layout**
- ▶ **List layout**

Table 15.210. Properties of the **Box layout** widget feature

Property name	Description	Set by EB GUIDE GTF
gap	The space between two child widgets, depending on the layout direction	no
layoutDirection	The direction in which the list elements i.e. the child widgets are positioned. Possible values: <ul style="list-style-type: none"> <li>▶ <code>horizontal (0)</code></li> <li>▶ <code>vertical (1)</code></li> </ul>	no

### 15.11.6.3. Flow layout

The **Flow layout** widget feature defines position and size of each child widget.

Position and size properties of child widgets are set by the parent widget. Invisible child widgets are ignored in the calculation.

Restrictions:

- ▶ The **Flow layout** widget feature excludes the following widget features:
  - ▶ **Absolute layout**
  - ▶ **Box layout**
  - ▶ **Grid layout**
  - ▶ **List layout**

Table 15.211. Properties of the **Flow layout** widget feature

Property name	Description	Set by EB GUIDE GTF
horizontalGap	The horizontal space between two child widgets	no
verticalGap	The vertical space between two child widgets	no
layoutDirection	The direction in which the list elements i.e. the child widgets are positioned. Possible values: <ul style="list-style-type: none"> <li>▶ <code>horizontal (0)</code></li> <li>▶ <code>vertical (1)</code></li> </ul>	no

Property name	Description	Set by EB GUIDE GTF
horizontal-ChildAlign	<p>The horizontal alignment of child widgets. Possible values:</p> <ul style="list-style-type: none"> <li>▶ <code>leading (0)</code>: The child widget is placed on the left side.</li> <li>▶ <code>center (1)</code>: The child widget is placed in the center.</li> <li>▶ <code>trailing (2)</code>: The child widget is placed on the right side.</li> </ul>	no
vertical-ChildAlign	<p>The vertical alignment of child widgets. Possible values:</p> <ul style="list-style-type: none"> <li>▶ <code>center (0)</code>: The child widget is placed in the center.</li> <li>▶ <code>top (1)</code>: The child widget is placed at the top</li> <li>▶ <code>bottom (2)</code>: The child widget is placed at the bottom.</li> </ul>	no

#### 15.11.6.4. Grid layout

The **Grid layout** widget feature defines position and size of each child widget.

Position and size properties of child widgets are set by the parent widget. Invisible child widgets are ignored in the calculation.

Restrictions:

- ▶ The **Grid layout** widget feature excludes the following widget features:
  - ▶ **Absolute layout**
  - ▶ **Box layout**
  - ▶ **Flow layout**
  - ▶ **List layout**

Table 15.212. Properties of the **Grid layout** widget feature

Property name	Description	Set by EB GUIDE GTF
horizontalGap	The horizontal space between two child widgets	no
verticalGap	The vertical space between two child widgets	no
numRows	Defines the number of rows	no
numColumns	Defines the number of columns	no

#### 15.11.6.5. Layout margins

The **Layout margins** widget feature adds configurable margins to a widget that uses the **Flow layout**, **Absolute layout**, **Box layout**, or **Grid layout** widget feature.

Table 15.213. Properties of the **Layout margins** widget feature

Property name	Description	Set by EB GUIDE GTF
<code>leftMargin</code>	The margin of the left border	no
<code>topMargin</code>	The margin of the top border	no
<code>rightMargin</code>	The margin of the right border	no
<code>bottomMargin</code>	The margin of the bottom border	no

#### 15.11.6.6. List layout

The **List layout** widget feature defines position and size of each child widget in pixels.

Position properties of child widgets and the `listIndex` property of the **List index** widget feature are set by the parent widget.

Best used in conjunction with instantiators to create the child widgets.

For details about the **List index** widget feature, see [section 15.11.7.2, “List index”](#).

Restrictions:

- ▶ The **List layout** widget feature is intended to be used with instantiator.
- ▶ The **List layout** widget feature excludes the following widget features:
  - ▶ **Absolute layout**
  - ▶ **Box layout**
  - ▶ **Flow layout**
  - ▶ **Grid layout**

Table 15.214. Properties of the **List layout** widget feature

Property name	Description	Set by EB GUIDE GTF
<code>layoutDirection</code>	The direction in which the list elements i.e. the child widgets are positioned. Possible values: <ul style="list-style-type: none"><li>▶ <code>horizontal</code> (0)</li><li>▶ <code>vertical</code> (1)</li></ul>	no
<code>scrollOffset</code>	The number of pixels to scroll the list	yes

Property name	Description	Set by EB GUIDE GTF
<code>scrollOffsetRebase</code>	If the <code>scrollOffsetRebase</code> property changes, the current <code>scrollOffset</code> is translated to <code>scrollIndex</code> . The remaining offset is written to the <code>scrollOffset</code> property.	no
<code>firstListIndex</code>	The list index of the first visible list element, defined by the width feature	yes
<code>scrollIndex</code>	The base list index the <code>scrollOffset</code> property applies to. Scrolling starts at the list elements given in the <code>scrollIndex</code> property.	yes
<code>scrollValue</code>	The current scroll value in pixels	yes
<code>scrollValueMax</code>	The maximum scroll value in pixels, which is mapped to the end of the list	no
<code>scrollValueMin</code>	The minimum scroll value in pixels, which is mapped to the beginning of the list	no
<code>bounceValue</code>	The <code>bounceValue</code> property is zero as long as the <code>scrollOffset</code> property results in a position inside the valid scroll range. It has a positive value if the scroll position exceeds the beginning of the list and a negative value if the scroll position exceeds the end of the list. If <code>bounceValue</code> is added to <code>scrollOffset</code> , the scroll position is back in range.	yes
<code>bounceValueMax</code>	The maximum value which <code>scrollOffset</code> can move outside the valid scroll range. <code>scrollOffset</code> is truncated if the user tries to scroll further.	no
<code>segments</code>	For horizontal layout direction: the number of rows  For vertical layout direction: the number of columns	no
<code>listLength</code>	The number of list elements	no
<code>wrapAround</code>	Possible values:  ► <code>true</code> : The <code>scrollValue</code> property continues at the inverse border, if <code>scrollValueMin</code> or <code>scrollValueMax</code> is exceeded.  ► <code>false</code> : The <code>scrollValue</code> property does not decrease/increase, if <code>scrollValueMin</code> or <code>scrollValueMax</code> is exceeded.	no

### 15.11.6.7. Scale mode

The **Scale mode** widget feature defines how an image is displayed if its size differs from the size of the widget.

Restrictions:

- ▶ The **Scale mode** widget feature is only available for the Image widget.

Table 15.215. Properties of the **Scale mode** widget feature

Property name	Description	Set by EB GUIDE GTF
scaleMode	The scale mode of the image. Possible values: <ul style="list-style-type: none"><li>▶ original size (0)</li><li>▶ fit to size (1)</li><li>▶ keep aspect ratio (2)</li></ul>	no

## 15.11.7. List management

### 15.11.7.1. Line index

The **Line index** widget feature defines the unique position for each line of your list or table.

Restrictions:

- ▶ The **Line index** widget feature is intended to be used in combination with the Instantiator widget.

Table 15.216. Properties of the **Line index** widget feature

Property name	Description	Set by EB GUIDE GTF
lineIndex	The index of the current line in a table	yes

### 15.11.7.2. List index

The **List index** widget feature defines the unique position of a widget in a list.

Restrictions:

- ▶ The **List index** widget feature is intended to be used in combination with the **List layout** widget feature.

Table 15.217. Properties of the **List index** widget feature

Property name	Description	Set by EB GUIDE GTF
<code>listIndex</code>	The index of the current widget in a list	yes

### 15.11.7.3. Template index

The **Template index** widget feature defines the unique position of the used line template.

Restrictions:

- ▶ The **Template index** widget feature is intended to be used in combination with the Instantiator widget.

Table 15.218. Properties of the **Template index** widget feature

Property name	Description	Set by EB GUIDE GTF
<code>lineTemplateIndex</code>	The index of the used line template	yes

### 15.11.7.4. Viewport

The **Viewport** widget feature clips oversized elements at the widget borders.

Restrictions:

- ▶ The **Viewport** widget feature is intended to be used in combination with the Container widget or lists.
- ▶ The **Viewport** widget feature takes effect on the following model elements:
  - ▶ Child widgets of the widget you added **Viewport** to are clipped inside the dimensions of the widget.
  - ▶ The widget you added **Viewport** is clipped inside the dimensions of its parent view.

Table 15.219. Properties of the **Viewport** widget feature

Property name	Description	Set by EB GUIDE GTF
<code>xOffset</code>	The horizontal offset of the visible clipping within the drawn area of child widgets	no
<code>yOffset</code>	The vertical offset of the visible clipping within the drawn area of child widgets	no



## 15.11.8. 3D

Widget features in the **3D** category are only available for 3D widgets.

### 15.11.8.1. Anti-aliasing mode

Table 15.220. Properties of the **Anti-aliasing mode** widget feature

Property name	Description	Set by EB GUIDE GTF
antiAliasing	<p>Defines the anti-aliasing mode for a scene graph. This overrides the configuration of the scene. To use the same value that is configured in the scene use <code>Global (5)</code>.</p> <ul style="list-style-type: none"><li>▶ <code>off (0)</code></li><li>▶ <code>MSAA 2x (1)</code></li><li>▶ <code>MSAA 4x (2)</code></li><li>▶ <code>MSAA 8x (3)</code></li><li>▶ <code>FXAA (4)</code></li><li>▶ <code>global (5)</code></li></ul>	no

### 15.11.8.2. Camera bloom

The **Camera bloom** widget feature is a post-processing effect. It produces fringes of light extending from the borders of bright areas in an image, contributing to the illusion of an extremely bright light overwhelming the camera or eye capturing the scene.

Restrictions:

- ▶ The **Camera bloom** widget feature is only available for the camera.

Table 15.221. Properties of the **Camera bloom** widget feature.

Property name	Description	Set by EB GUIDE GTF
enabled	If true, the bloom effect is applied to the camera.	no
threshold	The intensity (brightness) value that defines the split between the affected areas. Possible values are between 0.0 and 1.0.	no

Property name	Description	Set by EB GUIDE GTF
	If the <b>Tone mapping</b> widget feature is added to the parent scene graph, the <code>threshold</code> can be greater than 1.0.	
<code>strength</code>	Strength of the bloom effect.	no
<code>radius</code>	The radius of the glow.	no

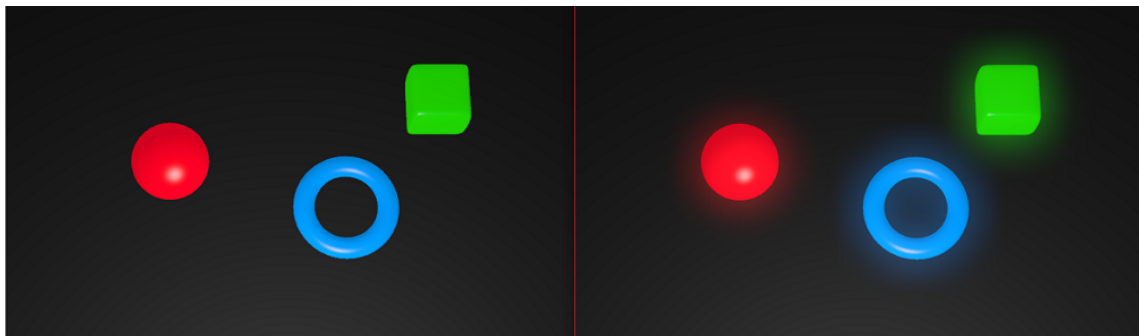


Figure 15.3. Example for objects without bloom effect (left) and with bloom effect (right)

### 15.11.8.3. Camera depth of field

The **Camera depth of field** widget feature is a post-processing effect. It simulates the focus properties of a camera lens by focusing sharply only on an object at a specific distance. Objects that are nearer or farther from the camera appear blurred.

Restrictions:

- The **Camera depth of field** widget feature is only available for the Camera widget.

Table 15.222. Properties of the **Camera depth of field** widget feature

Property name	Description	Set by EB GUIDE GTF
<code>enabled</code>	If true, the depth of field effect is applied to the camera.	no
<code>focusDistance</code>	Distance from the camera position, in world space. It should be between the <code>nearPlane</code> and <code>farPlane</code> property of the camera.	no
<code>fStop</code>	Defines the size of the focus region.	no

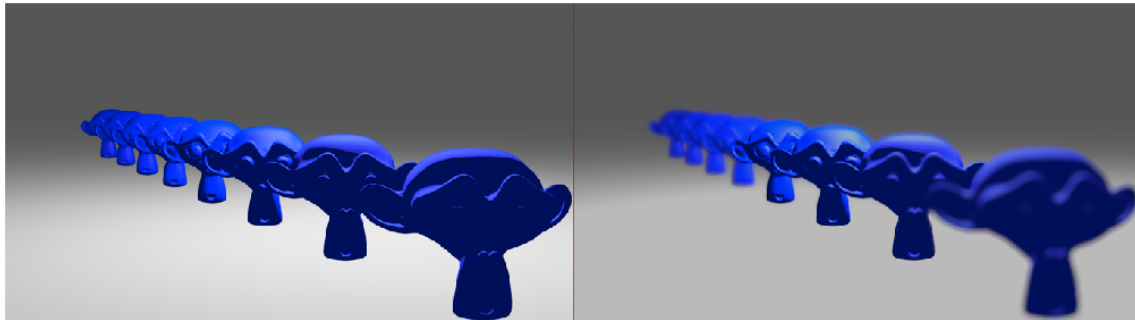


Figure 15.4. Example for objects with depth of field effect off (left) and with depth of field effect on (right)

#### 15.11.8.4. Camera viewport

The **Camera viewport** widget feature defines the camera's drawing region within the scene graph.

Restrictions:

- The **Camera viewport** widget feature is available for the Camera widget.

Table 15.223. Properties of the **Camera viewport** widget feature

Property name	Description	Set by EB GUIDE GTF
viewportX	The x-origin of the viewport within the scene graph	no
viewportY	The y-origin of the viewport within the scene graph	no
viewportWidth	The viewport's width in pixels	no
viewportHeight	The viewport's height in pixels	no

#### 15.11.8.5. Clear coat

The **Clear coat** widget feature adds a reflection layer to simulate a multi-layer surface.

Restrictions:

- The **Clear coat** widget feature is available for the PBR GGX material and PBR Phong material widgets.

Table 15.224. Properties of the **Clear coat** widget feature

Property name	Description	Set by EB GUIDE GTF
clearCoatStrength	The strength of the clear coat layer. Possible values are between 0.0 and 1.0.	no

Property name	Description	Set by EB GUIDE GTF
clearCoatRoughness	Perceived roughness of the clear coat layer. Possible values are between 0.0 and 1.0.	no

### 15.11.8.6. Ambient texture

The **Ambient texture** widget feature adds extended configuration values to a material.

Restrictions:

- ▶ The **Ambient texture** widget feature is available for the Material, PBR Phong material, and PBR GGX material widgets.
- ▶ When the **Ambient texture** is added, the `ambient` property is ignored.

Table 15.225. Properties of the **Ambient texture** widget feature

Property name	Description	Set by EB GUIDE GTF
ambientTexture	The file name of the texture	no
ambientTextureAddressModeU	The address mode of the texture along the u-direction. Possible values: <ul style="list-style-type: none"> <li>▶ <code>repeat (0)</code>: When accessed outside the texture bounds, the texture is repeated. Also known as wrap or tile</li> <li>▶ <code>clamp (1)</code>: When accessed outside the texture bounds, the pixels at the edge of the texture are used.</li> </ul>	no
ambientTextureAddressModeV	The address mode of the texture along the v-direction. Possible values: <ul style="list-style-type: none"> <li>▶ <code>repeat (0)</code>: When accessed outside the texture bounds, the texture is repeated. Also known as wrap or tile</li> <li>▶ <code>clamp (1)</code>: When accessed outside the texture bounds, the pixels at the edge of the texture are used.</li> </ul>	no
ambientFilterMode	The filtering mode of the texture. Possible values: <ul style="list-style-type: none"> <li>▶ <code>point (0)</code>: Texture is not smoothed at all. Least expensive but prone to aliasing artifacts when texture is minimized.</li> <li>▶ <code>linear (1)</code>: Also known as bilinear filtering. Smoothens the texture when minimized to reduce aliasing artifacts.</li> </ul>	no

Property name	Description	Set by EB GUIDE GTF
	► <code>trilinear (2)</code> : Most expensive, but yields better results than linear filtering.	
<code>ambientSRGB</code>	<p>If this property is enabled, the texture that is selected in <code>ambientTexture</code>, is rendered using sRGB color space.</p> <p>Note that to use sRGB functionality, open the project center, go to <b>Configure &gt; Profiles</b> and for the <code>colorMode</code> property select <code>32-bit sRGB (4)</code> or <code>32-bit sRGB (Emulated) (5)</code>.</p>	no

### 15.11.8.7. Anisotropy

The **Anisotropy** widget feature adds an anisotropic reflection to a surface. Materials with grooves, scratches or fibers on the surface, like brushed metal, can be replicated using an anisotropic model.

Restrictions:

- The **Anisotropy** widget feature is available for the PBR GGX material widget.
- For correct shading the tangent vectors must be available in the mesh.

Table 15.226. Properties of the **Anisotropy** widget feature

Property name	Description	Set by EB GUIDE GTF
<code>anisotropy</code>	Amount and direction of anisotropic reflection. Positive values give elongated highlights shaped along the bitangent direction. Negative values give highlights shaped along the tangent direction. Possible values are between -1.0 and 1.0.	no

### 15.11.8.8. Diffuse texture

The **Diffuse texture** widget feature adds extended configuration values to a material.

Restrictions:

- The **Diffuse texture** widget feature is available for the Material and PBR Phong material widgets.

For PBR GGX material, the **Base color texture** widget feature is available. For more information see [section 15.11.8.9, “Base color texture”](#).

- ▶ When the **Diffuse texture** is added, the `diffuse` property is ignored.

Table 15.227. Properties of the **Diffuse texture** widget feature

Property name	Description	Set by EB GUIDE GTF
<code>diffuseTexture</code>	The file name of the texture	no
<code>diffuseTextureAddressModeU</code>	<p>The address mode of the texture along the u-direction. Possible values:</p> <ul style="list-style-type: none"> <li>▶ <code>repeat (0)</code>: When accessed outside the texture bounds, the texture is repeated. Also known as wrap or tile</li> <li>▶ <code>clamp (1)</code>: When accessed outside the texture bounds, the pixels at the edge of the texture are used.</li> </ul>	no
<code>diffuseTextureAddressModeV</code>	<p>The address mode of the texture along the v-direction. Possible values:</p> <ul style="list-style-type: none"> <li>▶ <code>repeat (0)</code>: When accessed outside the texture bounds, the texture is repeated. Also known as wrap or tile</li> <li>▶ <code>clamp (1)</code>: When accessed outside the texture bounds, the pixels at the edge of the texture are used.</li> </ul>	no
<code>diffuseFilterMode</code>	<p>The filtering mode of the texture. Possible values:</p> <ul style="list-style-type: none"> <li>▶ <code>point (0)</code>: Texture is not smoothed at all. Least expensive but prone to aliasing artifacts when texture is minimized.</li> <li>▶ <code>linear (1)</code>: Also known as bilinear filtering. Smoothens the texture when minimized to reduce aliasing artifacts.</li> <li>▶ <code>trilinear (2)</code>: Most expensive, but yields better results than linear filtering.</li> </ul>	no
<code>diffuseSRGB</code>	<p>If this property is enabled, the texture that is selected in <code>diffuseTexture</code>, is rendered using sRGB color space.</p> <p>Note that to use sRGB functionality, open the project center, go to <b>Configure &gt; Profiles</b> and for the <code>colorMode</code> property select <code>32-bit sRGB (4)</code> or <code>32-bit sRGB (Emulated) (5)</code>.</p>	no

#### 15.11.8.9. Base color texture

The **Base color texture** widget feature adds extended configuration values to a PBR GGX material.

Restrictions:

- ▶ The **Base color texture** widget feature is available for the PBR GGX material widget.

For material or PBR Phong material, the **Diffuse texture** widget feature is available. For more information see [section 15.11.8.8, “Diffuse texture”](#).

- ▶ When the **Base color texture** is added, the `baseColor` property is ignored.

Table 15.228. Properties of the **Base color texture** widget feature

Property name	Description	Set by EB GUIDE GTF
<code>baseColorTexture</code>	The file name of the texture	no
<code>baseColorTextureAddressModeU</code>	The address mode of the texture along the u-direction. Possible values: <ul style="list-style-type: none"> <li>▶ <code>repeat (0)</code>: When accessed outside the texture bounds, the texture is repeated. Also known as wrap or tile</li> <li>▶ <code>clamp (1)</code>: When accessed outside the texture bounds, the pixels at the edge of the texture are used.</li> </ul>	no
<code>baseColorTextureAddressModeV</code>	The address mode of the texture along the v-direction. Possible values: <ul style="list-style-type: none"> <li>▶ <code>repeat (0)</code>: When accessed outside the texture bounds, the texture is repeated. Also known as wrap or tile</li> <li>▶ <code>clamp (1)</code>: When accessed outside the texture bounds, the pixels at the edge of the texture are used.</li> </ul>	no
<code>baseColorFilterMode</code>	The filtering mode of the texture. Possible values: <ul style="list-style-type: none"> <li>▶ <code>point (0)</code>: Texture is not smoothed at all. Least expensive but prone to aliasing artifacts when texture is minimized.</li> <li>▶ <code>linear (1)</code>: Also known as bilinear filtering. Smoothens the texture when minimized to reduce aliasing artifacts.</li> <li>▶ <code>trilinear (2)</code>: Most expensive, but yields better results than linear filtering.</li> </ul>	no
<code>baseColorSRGB</code>	If this property is enabled, the texture that is selected in <code>baseColorTexture</code> , is rendered using sRGB color space.  Note that to use sRGB functionality, open the project center, go to <b>Configure &gt; Profiles</b> and for the <code>colorMode</code> property select <code>32-bit sRGB (4)</code> or <code>32-bit sRGB (Emulated) (5)</code> .	no

### 15.11.8.10. Emissive texture

The **Emissive texture** widget feature adds extended configuration values to a material.

Restrictions:

- ▶ The **Emissive texture** widget feature is available for the Material, PBR Phong material, and PBR GGX material widgets.
- ▶ When the **Emissive texture** is added, the `emissive` property is ignored.

Table 15.229. Properties of the **Emissive texture** widget feature

Property name	Description	Set by EB GUIDE GTF
<code>emissiveTexture</code>	The file name of the texture	no
<code>emissiveTextureAddressModeU</code>	The address mode of the texture along the u-direction. Possible values: <ul style="list-style-type: none"> <li>▶ <code>repeat (0)</code>: When accessed outside the texture bounds, the texture is repeated. Also known as wrap or tile</li> <li>▶ <code>clamp (1)</code>: When accessed outside the texture bounds, the pixels at the edge of the texture are used.</li> </ul>	no
<code>emissiveTextureAddressModeV</code>	The address mode of the texture along the v-direction. Possible values: <ul style="list-style-type: none"> <li>▶ <code>repeat (0)</code>: When accessed outside the texture bounds, the texture is repeated. Also known as wrap or tile</li> <li>▶ <code>clamp (1)</code>: When accessed outside the texture bounds, the pixels at the edge of the texture are used.</li> </ul>	no
<code>emissiveFilterMode</code>	The filtering mode of the texture. Possible values: <ul style="list-style-type: none"> <li>▶ <code>point (0)</code>: Texture is not smoothed at all. Least expensive but prone to aliasing artifacts when texture is minimized.</li> <li>▶ <code>linear (1)</code>: Also known as bilinear filtering. Smoothens the texture when minimized to reduce aliasing artifacts.</li> <li>▶ <code>trilinear (2)</code>: Most expensive, but yields better results than linear filtering.</li> </ul>	no
<code>emissiveSRGB</code>	If this property is enabled, the texture that is selected in <code>emissiveTexture</code> , is rendered using sRGB color space.	no



Property name	Description	Set by EB GUIDE GTF
	Note that to use sRGB functionality, open the project center, go to <b>Configure &gt; Profiles</b> and for the <code>colorMode</code> property select <code>32-bit sRGB (4)</code> or <code>32-bit sRGB (Emulated) (5)</code> .	

### 15.11.8.11. Light map texture

The **Light map texture** widget feature adds extended configuration values to a material.

Restrictions:

- ▶ The **Light map texture** widget feature is available for the Material, PBR Phong material, and PBR GGX material widgets.

Table 15.230. Properties of the **Light map texture** widget feature

Property name	Description	Set by EB GUIDE GTF
<code>lightMapTexture</code>	The file name of the texture	no
<code>lightMapTextureAddressModeU</code>	The address mode of the texture along the u-direction. Possible values: <ul style="list-style-type: none"> <li>▶ <code>repeat (0)</code>: When accessed outside the texture bounds, the texture is repeated. Also known as wrap or tile</li> <li>▶ <code>clamp (1)</code>: When accessed outside the texture bounds, the pixels at the edge of the texture are used.</li> </ul>	no
<code>lightMapTextureAddressModeV</code>	The address mode of the texture along the v-direction. Possible values: <ul style="list-style-type: none"> <li>▶ <code>repeat (0)</code>: When accessed outside the texture bounds, the texture is repeated. Also known as wrap or tile</li> <li>▶ <code>clamp (1)</code>: When accessed outside the texture bounds, the pixels at the edge of the texture are used.</li> </ul>	no
<code>lightMapFilterMode</code>	The filtering mode of the texture. Possible values: <ul style="list-style-type: none"> <li>▶ <code>point (0)</code>: Texture is not smoothed at all. Least expensive but prone to aliasing artifacts when texture is minimized.</li> <li>▶ <code>linear (1)</code>: Also known as bilinear filtering. Smoothens the texture when minimized to reduce aliasing artifacts.</li> </ul>	no

Property name	Description	Set by EB GUIDE GTF
	<ul style="list-style-type: none"> <li>▶ <code>trilinear (2)</code>: Most expensive, but yields better results than linear filtering.</li> </ul>	

### 15.11.8.12. Metallic texture

The **Metallic** widget feature adds extended configuration values to a material. The texture controls the metallic parameter of the PBR GGX material and PBR Phong material widgets.

Restrictions:

- ▶ The **Metallic texture** widget feature is available for the PBR GGX material and PBR Phong material widgets.
- ▶ The **Metallic texture** is a grayscale image. For RGB color images, only the red channel is used.
- ▶ When the **Metallic texture** is added, the `metallic` property is ignored.

Table 15.231. Properties of the **Metallic texture** widget feature

Property name	Description	Set by EB GUIDE GTF
<code>metallicTexture</code>	The file name of the texture	no
<code>metallicMinFactor</code>	The minimal metallic parameter as a float to interpolate the texture values	no
<code>metallicMaxFactor</code>	The maximal metallic parameter as a float to interpolate the texture values	no
<code>metallicTextureAddressModeU</code>	<p>The address mode of the texture along the u-direction. Possible values:</p> <ul style="list-style-type: none"> <li>▶ <code>repeat (0)</code>: When accessed outside the texture bounds, the texture is repeated. Also known as wrap or tile</li> <li>▶ <code>clamp (1)</code>: When accessed outside the texture bounds, the pixels at the edge of the texture are used.</li> </ul>	no
<code>metallicTextureAddressModeV</code>	<p>The address mode of the texture along the v-direction. Possible values:</p> <ul style="list-style-type: none"> <li>▶ <code>repeat (0)</code>: When accessed outside the texture bounds, the texture is repeated. Also known as wrap or tile</li> <li>▶ <code>clamp (1)</code>: When accessed outside the texture bounds, the pixels at the edge of the texture are used.</li> </ul>	no

Property name	Description	Set by EB GUIDE GTF
metallicFilter-Mode	<p>The filtering mode of the texture. Possible values:</p> <ul style="list-style-type: none"> <li>▶ <code>point (0)</code>: Texture is not smoothed at all. Least expensive but prone to aliasing artifacts when texture is minimized.</li> <li>▶ <code>linear (1)</code>: Also known as bilinear filtering. Smoothens the texture when minimized to reduce aliasing artifacts.</li> <li>▶ <code>trilinear (2)</code>: Most expensive, but yields better results than linear filtering.</li> </ul>	no

### 15.11.8.13. Normal map texture

The **Normal map** widget feature adds extended configuration values to a material.

Restrictions:

- ▶ The **Normal map texture** widget feature is available for the Material, PBR Phong material, and PBR GGX material widgets.

Table 15.232. Properties of the **Normal map** widget feature

Property name	Description	Set by EB GUIDE GTF
normalMapTexture	The file name of the texture	no
normalMapTextureAddress-ModeU	<p>The address mode of the texture along the u-direction. Possible values:</p> <ul style="list-style-type: none"> <li>▶ <code>repeat (0)</code>: When accessed outside the texture bounds, the texture is repeated. Also known as wrap or tile</li> <li>▶ <code>clamp (1)</code>: When accessed outside the texture bounds, the pixels at the edge of the texture are used.</li> </ul>	no
normalMapTextureAddressModeV	<p>The address mode of the texture along the v-direction. Possible values:</p> <ul style="list-style-type: none"> <li>▶ <code>repeat (0)</code>: When accessed outside the texture bounds, the texture is repeated. Also known as wrap or tile</li> <li>▶ <code>clamp (1)</code>: When accessed outside the texture bounds, the pixels at the edge of the texture are used.</li> </ul>	no

Property name	Description	Set by EB GUIDE GTF
normalMapFilterMode	<p>The filtering mode of the texture. Possible values:</p> <ul style="list-style-type: none"> <li>▶ <code>point (0)</code>: Texture is not smoothed at all. Least expensive but prone to aliasing artifacts when texture is minimized.</li> <li>▶ <code>linear (1)</code>: Also known as bilinear filtering. Smoothens the texture when minimized to reduce aliasing artifacts.</li> <li>▶ <code>trilinear (2)</code>: Most expensive, but yields better results than linear filtering.</li> </ul>	no

#### 15.11.8.14. Opaque texture

The **Opaque texture** widget feature adds extended configuration values to a material.

Restrictions:

- ▶ The **Opaque texture** widget feature is available for the Material, PBR Phong material, and PBR GGX material widgets.

Table 15.233. Properties of the **Opaque texture** widget feature

Property name	Description	Set by EB GUIDE GTF
opaqueTexture	The file name of the texture	no
opaqueTextureAddressModeU	<p>The address mode of the texture along the u-direction. Possible values:</p> <ul style="list-style-type: none"> <li>▶ <code>repeat (0)</code>: When accessed outside the texture bounds, the texture is repeated. Also known as wrap or tile</li> <li>▶ <code>clamp (1)</code>: When accessed outside the texture bounds, the pixels at the edge of the texture are used.</li> </ul>	no
opaqueTextureAddressModeV	<p>The address mode of the texture along the v-direction. Possible values:</p> <ul style="list-style-type: none"> <li>▶ <code>repeat (0)</code>: When accessed outside the texture bounds, the texture is repeated. Also known as wrap or tile</li> <li>▶ <code>clamp (1)</code>: When accessed outside the texture bounds, the pixels at the edge of the texture are used.</li> </ul>	no
opaqueFilterMode	The filter mode of the texture. Possible values:	no

Property name	Description	Set by EB GUIDE GTF
	<ul style="list-style-type: none"> <li>▶ <code>point (0)</code>: Texture is not smoothed at all. Least expensive but prone to aliasing artifacts when texture is minimized.</li> <li>▶ <code>linear (1)</code>: Also known as bilinear filtering. Smoothens the texture when minimized to reduce aliasing artifacts.</li> <li>▶ <code>trilinear (2)</code>: Most expensive, but yields better results than linear filtering.</li> </ul>	

### 15.11.8.15. Reflection texture

The **Reflection texture** widget feature adds extended configuration values to a material.

Restrictions:

- ▶ The **Reflection texture** widget feature is available for the Material, PBR Phong material, and PBR GGX material widgets.

Table 15.234. Properties of the **Reflection texture** widget feature

Property name	Description	Set by EB GUIDE GTF
<code>reflectionTop-Texture</code>	The file name of the texture	no
<code>reflectionBottomTexture</code>	The file name of the texture	no
<code>reflectionLeft-Texture</code>	The file name of the texture	no
<code>reflection-RightTexture</code>	The file name of the texture	no
<code>reflection-FrontTexture</code>	The file name of the texture	no
<code>reflectionBack-Texture</code>	The file name of the texture	no
<code>reflectionFilterMode</code>	<p>The filtering mode of the texture. Possible values:</p> <ul style="list-style-type: none"> <li>▶ <code>point (0)</code>: Texture is not smoothed at all. Least expensive but prone to aliasing artifacts when texture is minimized.</li> </ul>	no

Property name	Description	Set by EB GUIDE GTF
	<ul style="list-style-type: none"> <li>▶ <code>linear (1)</code>: Also known as bilinear filtering. Smoothens the texture when minimized to reduce aliasing artifacts.</li> <li>▶ <code>trilinear (2)</code>: Most expensive, but yields better results than linear filtering.</li> </ul>	

#### NOTE



#### Reflection texture widget feature

EB GUIDE Studio displays the **Reflection texture** widget feature, only when an image file is selected for all of the following properties:

- ▶ `reflectionTopTexture`
- ▶ `reflectionBottomTexture`
- ▶ `reflectionLeftTexture`
- ▶ `reflectionRightTexture`
- ▶ `reflectionFrontTexture`
- ▶ `reflectionBackTexture`

The image files must have the same size and quadratic shape.

### 15.11.8.16. Roughness texture

The **Roughness texture** widget feature adds extended configuration values to a material. The texture controls the roughness parameter of the PBR GGX material widget.

Restrictions:

- ▶ The **Roughness texture** widget feature is available for the PBR GGX material widget.
- ▶ The **Roughness texture** is a grayscale image. For RGB color images, only the red channel is used.
- ▶ When the **Roughness texture** is active, the `roughness` property is ignored.

Table 15.235. Properties of the **Roughness texture** widget feature

Property name	Description	Set by EB GUIDE GTF
<code>roughnessTexture</code>	The file name of the texture	no
<code>roughnessMinFactor</code>	The minimal roughness parameter as a float to interpolate the texture values	no

Property name	Description	Set by EB GUIDE GTF
<code>roughnessMaxFactor</code>	The maximal roughness parameter as a float to interpolate the texture values	no
<code>roughnessTextureAddressModeU</code>	The address mode of the texture along the u-direction. Possible values: <ul style="list-style-type: none"> <li>▶ <code>repeat (0)</code>: When accessed outside the texture bounds, the texture is repeated. Also known as wrap or tile</li> <li>▶ <code>clamp (1)</code>: When accessed outside the texture bounds, the pixels at the edge of the texture are used.</li> </ul>	no
<code>roughnessTextureAddressModeV</code>	The address mode of the texture along the v-direction. Possible values: <ul style="list-style-type: none"> <li>▶ <code>repeat (0)</code>: When accessed outside the texture bounds, the texture is repeated. Also known as wrap or tile</li> <li>▶ <code>clamp (1)</code>: When accessed outside the texture bounds, the pixels at the edge of the texture are used.</li> </ul>	no
<code>roughnessFilterMode</code>	The filtering mode of the texture. Possible values: <ul style="list-style-type: none"> <li>▶ <code>point (0)</code>: Texture is not smoothed at all. Least expensive but prone to aliasing artifacts when texture is minimized.</li> <li>▶ <code>linear (1)</code>: Also known as bilinear filtering. Smoothens the texture when minimized to reduce aliasing artifacts.</li> <li>▶ <code>trilinear (2)</code>: Most expensive, but yields better results than linear filtering.</li> </ul>	no

### 15.11.8.17. Shininess texture

The **Shininess texture** widget feature adds extended configuration values to a material. The texture modulates the shininess strength by multiplying the texture value with the scalar `shininess` property.

Restrictions:

- ▶ The **Shininess texture** widget feature is available for the Material and PBR Phong material widgets.
- ▶ The **Shininess texture** is a grayscale image. For RGB color images, only the red channel is used.
- ▶ When the **Shininess texture** widget feature is used, the `shininess` property is ignored.

Table 15.236. Properties of the **Shininess texture** widget feature

Property name	Description	Set by EB GUIDE GTF
shininessTexture	The file name of the texture	no
shininessMinFactor	The minimal shininess parameter as a float to interpolate the texture values	no
shininessMaxFactor	The maximal shininess parameter as a float to interpolate the texture values	no
shininessTextureAddressModeU	The address mode of the texture along the u-direction. Possible values: <ul style="list-style-type: none"> <li>▶ <code>repeat (0)</code>: When accessed outside the texture bounds, the texture is repeated. Also known as wrap or tile</li> <li>▶ <code>clamp (1)</code>: When accessed outside the texture bounds, the pixels at the edge of the texture are used.</li> </ul>	no
shininessTextureAddressModeV	The address mode of the texture along the v-direction. Possible values: <ul style="list-style-type: none"> <li>▶ <code>repeat (0)</code>: When accessed outside the texture bounds, the texture is repeated. Also known as wrap or tile</li> <li>▶ <code>clamp (1)</code>: When accessed outside the texture bounds, the pixels at the edge of the texture are used.</li> </ul>	no
shininessFilterMode	The filtering mode of the texture. Possible values: <ul style="list-style-type: none"> <li>▶ <code>point (0)</code>: Texture is not smoothed at all. Least expensive but prone to aliasing artifacts when texture is minimized.</li> <li>▶ <code>linear (1)</code>: Also known as bilinear filtering. Smoothens the texture when minimized to reduce aliasing artifacts.</li> <li>▶ <code>trilinear (2)</code>: Most expensive, but yields better results than linear filtering.</li> </ul>	no

### 15.11.8.18. Specular texture

The **Specular texture** widget feature adds extended configuration values to a material.

Restrictions:

- ▶ The **Specular texture** widget feature is available for the Material and PBR Phong material widgets.



For PBR GGX material, the **Reflectance texture** widget feature is available. For more information see [section 15.11.8.19, “Reflectance texture”](#).

- ▶ When the **Specular texture** is added, the `specular` property is ignored.

Table 15.237. Properties of the **Specular texture** widget feature

Property name	Description	Set by EB GUIDE GTF
<code>specularTexture</code>	The file name of the texture	no
<code>specularTextureAddressModeU</code>	The address mode of the texture along the u-direction. Possible values: <ul style="list-style-type: none"> <li>▶ <code>repeat (0)</code>: When accessed outside the texture bounds, the texture is repeated. Also known as wrap or tile</li> <li>▶ <code>clamp (1)</code>: When accessed outside the texture bounds, the pixels at the edge of the texture are used.</li> </ul>	no
<code>specularTextureAddressModeV</code>	The address mode of the texture along the v-direction. Possible values: <ul style="list-style-type: none"> <li>▶ <code>repeat (0)</code>: When accessed outside the texture bounds, the texture is repeated. Also known as wrap or tile</li> <li>▶ <code>clamp (1)</code>: When accessed outside the texture bounds, the pixels at the edge of the texture are used.</li> </ul>	no
<code>specularFilterMode</code>	The filtering mode of the texture. Possible values: <ul style="list-style-type: none"> <li>▶ <code>point (0)</code>: Texture is not smoothed at all. Least expensive but prone to aliasing artifacts when texture is minimized.</li> <li>▶ <code>linear (1)</code>: Also known as bilinear filtering. Smoothens the texture when minimized to reduce aliasing artifacts.</li> <li>▶ <code>trilinear (2)</code>: Most expensive, but yields better results than linear filtering.</li> </ul>	no
<code>specularSRGB</code>	If this property is enabled, the texture that is selected in <code>specularTexture</code> , is rendered using sRGB color space.  Note that to use sRGB functionality, open the project center, go to <b>Configure &gt; Profiles</b> and for the <code>colorMode</code> property select <code>32-bit sRGB (4)</code> or <code>32-bit sRGB (Emulated) (5)</code> .	no

### 15.11.8.19. Reflectance texture

The **Reflectance texture** widget feature adds extended configuration values to a PBR GGX material.

Restrictions:

- ▶ The **Reflectance texture** widget feature is available for the PBR GGX material widget.

For the Material and PBR Phong material widgets, the **Specular texture** widget feature is available. For more information see [section 15.11.8.18, “Specular texture”](#).

- ▶ When the **Reflectance texture** is added, the `reflectance` property is ignored.

Table 15.238. Properties of the **Reflectance texture** widget feature

Property name	Description	Set by EB GUIDE GTF
<code>reflectanceTexture</code>	The file name of the texture	no
<code>reflectanceTextureAddressModeV</code>	<p>The address mode of the texture along the v-direction. Possible values:</p> <ul style="list-style-type: none"> <li>▶ <code>repeat (0)</code>: When accessed outside the texture bounds, the texture is repeated. Also known as wrap or tile</li> <li>▶ <code>clamp (1)</code>: When accessed outside the texture bounds, the pixels at the edge of the texture are used.</li> </ul>	no
<code>reflectanceFilterMode</code>	<p>The filtering mode of the texture. Possible values:</p> <ul style="list-style-type: none"> <li>▶ <code>point (0)</code>: Texture is not smoothed at all. Least expensive but prone to aliasing artifacts when texture is minimized.</li> <li>▶ <code>linear (1)</code>: Also known as bilinear filtering. Smoothens the texture when minimized to reduce aliasing artifacts.</li> <li>▶ <code>trilinear (2)</code>: Most expensive, but yields better results than linear filtering.</li> </ul>	no
<code>reflectanceSRGB</code>	<p>If this property is enabled, the texture that is selected in <code>reflectanceTexture</code> is rendered using sRGB color space.</p> <p>Note that to use sRGB functionality, open the project center, go to <b>Configure &gt; Profiles</b> and for the <code>colorMode</code> property select 32-bit sRGB (4) or 32-bit sRGB (Emulated) (5).</p>	no

### 15.11.8.20. Texture coordinate transformation

The **Texture coordinate transformation** widget feature makes it possible to modify the coordinates of the material texture.

Restrictions:

- This feature is available for the Material, PBR Phong material, and PBR GGX material.

Table 15.239. Properties of the **Texture coordinate transformation** widget feature

Property name	Description	Set by EB GUIDE GTF
uOffset	Defines the offset of the texture coordinates in u-direction	no
vOffset	Defines the offset of the texture coordinates in v-direction	no
uScale	Defines the scaling of the texture coordinates in u-direction	no
vScale	Defines the scaling of the texture coordinates in v-direction	no



#### Example 15.4. Example for Texture coordinate transformation

uOffset: 1.0	0.5	1.0	1.0	1.0
vOffset: 1.0	1.0	0.5	1.0	1.0
uScale: 1.0	1.0	1.0	0.5	3.0
vScale: 1.0	1.0	1.0	0.5	3.0
				

### 15.11.8.21. Tone mapping

The **Tone mapping** widget feature enables tone mapping, i.e. the technique to map a luminance value to a limited range, for the scene graph.

Restrictions:

- The **Tone mapping** widget feature is available for the Scene graph widget.

Note that the **Tone mapping** widget feature implements the global tone mapping operator described by Erik Reinhard et al.<sup>1</sup>

Table 15.240. Properties of the **Tone mapping** widget feature

Property name	Description	Set by EB GUIDE GTF
<code>pureWhiteLuminance</code>	The smallest luminance value that is mapped to pure white. Note that only values bigger or equal to 0 are valid.	no

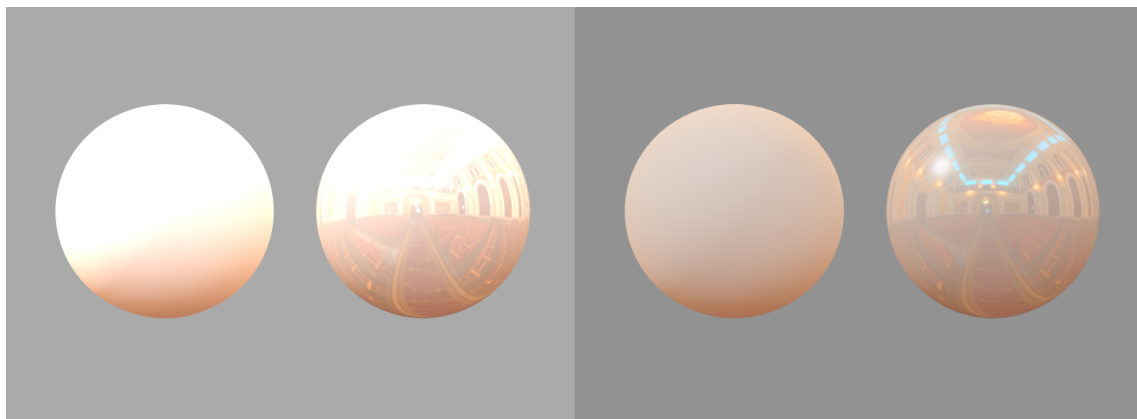


Figure 15.5. Example for image without tone mapping (left) and with tone mapping (right)

#### 15.11.8.22. Screen space ambient occlusion

The **Screen space ambient occlusion** widget feature is a post-processing effect. It provides an approximation of indirect lighting by calculating how ambient light will affect a scene.

Restrictions:

- The **Screen space ambient occlusion** widget feature is only available for the Scene graph widget.

Table 15.241. Properties of the **Screen space ambient occlusion** widget feature

Property name	Description	Set by EB GUIDE GTF
<code>enabled</code>	If true, the effect is applied.	no
<code>radius</code>	Defines the radius of the samples sphere, in which <code>sample-sCount</code> will be used, around the point of interest. The value is a positive value between 0.0 and 1.0.	no
<code>falloff</code>	Inner sphere that defines the minimal distance, beginning with which points can be occluded. The value is a positive value be-	no

<sup>1</sup>*Photographic tone reproduction for digital images* Reinhard, Erik et al. in "Proceedings of the 29th annual conference on Computer graphics and interactive techniques" 2002, Pages 267-276

Property name	Description	Set by EB GUIDE GTF
	tween 0.0 and 1.0. The value should be lower than the value of radius.	
samples	Number of samples in the samples sphere. The higher the number, the higher the quality of occlusion. Value is between 8 and 64 samples.	no

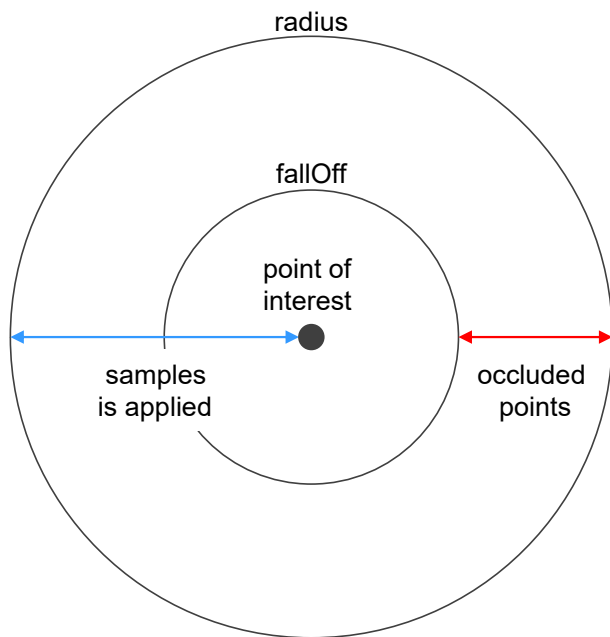


Figure 15.6. Sample sphere and properties of the **Screen space ambient occlusion** widget feature

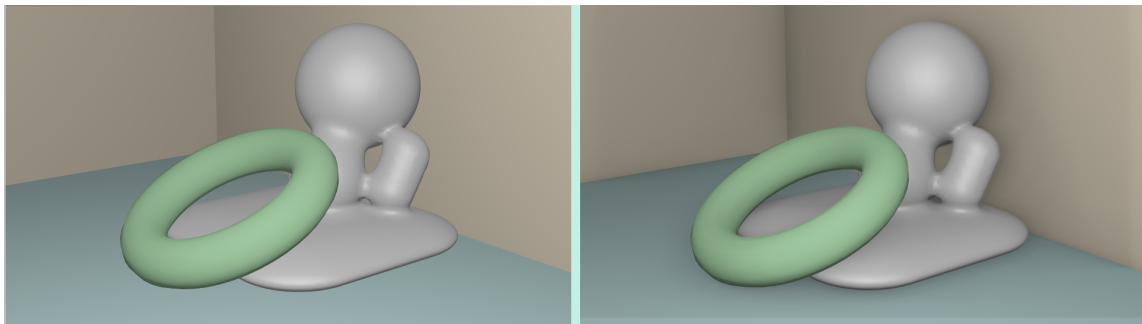


Figure 15.7. Example for objects with **Screen space ambient occlusion** off (left) and on (right)

### 15.11.9. Transformation

The widget features of the category **Transformation** modify location, form, and size of widgets.

The order in which transformations are executed is equal to the order in the widget tree. If multiple transformations are applied to one widget at the same widget tree hierarchy level, the order is as follows:

1. Translation
2. Shearing
3. Scaling
4. Rotation around z-axis
5. Rotation around y-axis
6. Rotation around x-axis

### 15.11.9.1. Pivot

The **Pivot** widget feature defines the pivot point of transformations which are applied to the widget. If no pivot point is configured, the default pivot point is at (0.0, 0.0, 0.0).

Restrictions:

- Adding the **Pivot** widget feature automatically adds the **Rotation**, **Scaling** and **Shearing** widget features.

Table 15.242. Properties of the **Pivot** widget feature

Property name	Description	Set by EB GUIDE GTF
<code>pivotX</code>	The pivot point on the x-axis relative to parent widget	no
<code>pivotY</code>	The pivot point on the y-axis relative to parent widget	no
<code>pivotZ</code>	The pivot point on the z-axis relative to parent widget if widget is a scene graph	no

### 15.11.9.2. Rotation

The **Rotation** widget feature is used to rotate the widget and its subtree.

Table 15.243. Properties of the **Rotation** widget feature

Property name	Description	Set by EB GUIDE GTF
<code>rotationEnabled</code>	Defines whether rotation is used or not	no
<code>rotationAngleX</code>	The rotation angle on the x-axis. This property only affects scene graph.	no

Property name	Description	Set by EB GUIDE GTF
rotationAngleY	The rotation angle on the y-axis. This property only affects scene graph.	no
rotationAngleZ	The rotation angle on the z-axis	no

### 15.11.9.3. Scaling

The **Scaling** widget feature is used to scale the widget and its subtree.

Table 15.244. Properties of the **Scaling** widget feature

Property name	Description	Set by EB GUIDE GTF
scalingEnabled	Defines whether scaling is used or not	no
scalingX	The scaling on the x-axis in percent	no
scalingY	The scaling on the y-axis in percent	no
scalingZ	The scaling on the z-axis in percent if widget is a scene graph	no

### 15.11.9.4. Shearing

The **Shearing** widget feature is used to distort the widget and its subtree.

Table 15.245. Properties of the **Shearing** widget feature

Property name	Description	Set by EB GUIDE GTF
shearingEnabled	Defines whether shearing is used or not	no
shearingXbyY	The shearing of x-axis by y-axis	no
shearingXbyZ	The shearing of x-axis by z-axis if widget is a scene graph	no
shearingYbyX	The shearing of y-axis by x-axis	no
shearingYbyZ	The shearing of y-axis by z-axis if widget is a scene graph	no
shearingZbyX	The shearing of z-axis by x-axis if widget is a scene graph	no
shearingZbyY	The shearing of z-axis by y-axis if widget is a scene graph	no

### 15.11.9.5. Translation



The **Translation** widget feature is used to translate the widget and its subtree. It moves widgets in x, y and z directions.

Table 15.246. Properties of the **Translation** widget feature

Property name	Description	Set by EB GUIDE GTF
translationEnabled	Defines whether translation is used or not	no
translationX	The translation on the x-axis	no
translationY	The translation on the y-axis	no
translationZ	The translation on the z-axis if widget is a scene graph	no



## 16. Installation of EB GUIDE Studio

### 16.1. Background information

#### 16.1.1. Restrictions

---

**NOTE****Compatibility**

EB GUIDE product line 6 is not compatible with any previous major version.

---

**NOTE****User rights**

To install EB GUIDE on Windows 10 systems, you require administrator rights.

---

#### 16.1.2. System requirements

Observe the following settings:

Table 16.1. Recommended settings for EB GUIDE Studio

Hardware	PC with quad core CPU with at least 2 GHz CPU speed and 8 GB RAM
Operating system	Windows 10 (64-bit)
Screen resolution	1920 x 1080 pixels or more Two separate monitors recommended
Software	Microsoft .NET Framework 4.7.2 or later

Table 16.2. Recommended settings for EB GUIDE SDK

Development environment (IDE)	Microsoft Visual Studio 2013 or newer
File integration	CMake

## 16.2. Downloading EB GUIDE

To download the community edition of EB GUIDE, go to <https://www.elektrobit.com/ebguide/try-eb-guide/> and follow the instructions.

To download the enterprise edition of EB GUIDE, go to EB Command.

### NOTE



#### Activate your account

After ordering a product, you receive an email from sales department. Click the link in the email. Follow the steps to create an account as directed in the email and in the browser, then proceed to log in.

EB Command is the server from which you are going to download the EB GUIDE product line software. For the instructions on how to download from EB Command, see <https://www.elektrobit.com/support/downloading-from-eb-command/>.

## 16.3. Installing EB GUIDE



### Installing EB GUIDE

Prerequisite:

- You downloaded the setup file `studio_setup.exe`.
- You have administrator rights on the operating system.

#### Step 1

Double-click the setup file `studio_setup.exe`.

A dialog opens.

#### Step 2

Click **Yes**.

The **Setup - EB GUIDE Studio** dialog opens.

#### Step 3

Accept the license agreement and click **Next**.

#### Step 4

Select a folder for installation.

The default installation folder is `C:/Program Files/Elektrobit/EB GUIDE <version>`.

Step 5

Click **Next**.

A summary dialog displays all selected installation settings.

Step 6

To confirm the installation with the settings displayed, click **Install**.

The installation starts.

Step 7

To exit the setup click **Finish**.

You have installed EB GUIDE.

**TIP**

**Multiple installations**

It is possible to install more than one EB GUIDE versions.



## 16.4. Uninstalling EB GUIDE



### Uninstalling EB GUIDE

**NOTE**

**Removing EB GUIDE permanently**

If you follow the instruction, you remove EB GUIDE permanently from your PC.



Prerequisite:

- EB GUIDE is installed.
- You have administrator rights on the operating system.

Step 1

On the Windows **Start** menu, click **All Programs**.

Step 2

On **Elektrobit** menu, click the version you want to uninstall.

Step 3

On the submenu, click **Uninstall**.

# Glossary

## #

3D graphic	A 3D graphic is a virtual picture of a 3D scene. A 3D scene is a collection of 3D models (meshes or shapes), materials, light sources, and cameras. Materials define the visual appearance of 3D models through colors and textures and the behavior under virtual lighting. A camera provides the view point from where a virtual picture of the 3D scene is taken.
------------	--

## A

ADAS ECU	Advanced Driver Assistance System Electronic Control Unit  Open scalable platform based on ISO 26262 hardware design and software architecture that hosts algorithms from Autoliv, customers, or third parties.
ADASIS	Advanced Driver Assistance System Interface Specifications
anti-aliasing	Technique used in computer graphics to remove the aliasing effect. The anti-aliasing effect gives graphics a smoother appearance.
API	Application programming interface
application	In the context of EB GUIDE, an application is computer software that interacts with one or more EB GUIDE models at EB GUIDE GTF run-time by means of, for example, the event system and datapool. An application is, for example, entertainment software like media player, communication software like phone, etc. See Also API.
aspect	In EB GUIDE, an aspect is an appearance-related modification of an EB GUIDE model that is applied at EB GUIDE GTF run-time. Two types of aspects exist: skins, with which you can define different looks for your EB GUIDE model, and languages.

## C

communication context	The communication context describes the environment in which communication occurs. Each communication context is identified by a unique numerical ID.
-----------------------	---

## D

datapool	The datapool is a data cache in an EB GUIDE model that provides access to datapool items during run-time. It is used for data exchange between the application and the HMI.
datapool item	Datapool items store and exchange data. Each item in the datapool has a communication direction.

## E

EB GUIDE aware	EB GUIDE aware is a software framework that enables the creation of augmented reality solutions to enhance the driving experience.
EB GUIDE GTF	EB GUIDE GTF is the graphics target framework of the EB GUIDE product line and is part of EB GUIDE TF. EB GUIDE GTF represents the run-time environment to execute EB GUIDE models on target devices.
EB GUIDE GTF SDK	EB GUIDE GTF SDK is the development environment contained in EB GUIDE GTF. It is a sub-set of the EB GUIDE SDK. Another sub-set is the EB GUIDE Studio SDK.
EB GUIDE model	An EB GUIDE model is the sum of all elements that defines the look and behavior of an HMI. It is built entirely in EB GUIDE Studio. You can simulate the EB GUIDE model on your PC.
EB GUIDE product line	The EB GUIDE product line is a collection of software libraries and tools which are needed to specify an HMI model and convert the HMI model into a graphical user interface that runs on an embedded environment system.
EB GUIDE project	An EB GUIDE project consists of an EB GUIDE model and settings that are needed for running the EB GUIDE model on the target device.
EB GUIDE Script	EB GUIDE Script is the scripting language of the EB GUIDE product line. EB GUIDE Script enables accessing the datapool, model elements such as widgets and the state machine, and system events.
EB GUIDE SDK	EB GUIDE SDK is a product component of EB GUIDE. It is the software development kit for the EB GUIDE product line. It includes the EB GUIDE Studio SDK and the EB GUIDE GTF SDK.
EB GUIDE Studio	EB GUIDE Studio is the tool for modeling and specifying an HMI with a graphical user interfaces.

EB GUIDE Studio SDK	EB GUIDE Studio SDK is an application programming interface (API) to communicate with EB GUIDE Studio. It is a sub-set of the EB GUIDE SDK. Another sub-set is the EB GUIDE GTF SDK.
EB GUIDE TF	EB GUIDE TF is the run-time environment of EB GUIDE. It consists of EB GUIDE GTF. It is required to run an EB GUIDE model.
extension	In EB GUIDE, an extension is an addition to any of the EB GUIDE products. An extension is a plug-in in form of libraries ( <code>.dll</code> or <code>.so</code> files) that add a certain functionality to EB GUIDE Studio, EB GUIDE GTF, or EB GUIDE Monitor. Such functionality could be, for example, a data exporter or an additional widget feature.

## G

GL	Graphics library
GPS	Global Positioning System
GUI	Graphical user interface

## H

HMI	human machine interface
HMI model	An HMI model is the sum of all elements that defines the look and behavior of an HMI. It is created with an HMI software tool.

## I

IPC	Inter-process communication
IBL	image-based lighting
IBLGenerator	IBLGenerator is the tool to process environment lighting information.

## L

library	A library is a collection of pre-compiled software parts, sub-routines, or programs that are used in EB GUIDE. Libraries that are necessary for an EB GUIDE project are defined in the project center. Two file types are supported: <code>.dll</code> and <code>.so</code> .
---------	---

## M

MEF	Managed Extensibility Framework. See <a href="https://docs.microsoft.com/en-us/dotnet/framework/mef/">https://docs.microsoft.com/en-us/dotnet/framework/mef/</a> .
model element	A model element is an object within an EB GUIDE model, for example a state, a widget, or a datapool item. See Also EB GUIDE model.
model interface	The defined set of model elements that are used for the communication between exported EB GUIDE models on the target device.
multifont support	Aggregation of multiple fonts for different character ranges acting as a single font.
MVC	Model-view-controller

## N

namespace	In EB GUIDE Studio, with namespaces you create groups of model elements like datapool items and events. These groups have usually a defined functionality. Each namespace creates a naming scope for model elements so that model elements in different namespaces can have the same name.
-----------	--

## O

OS	Operating system
----	------------------

## P

PBR	Physically-based rendering
profile	In the project center, a profile is a set of specifications. In a profile you define libraries, messages and scenes for your project. During export of an EB GUIDE model the data in the profile is written to the <code>model.json</code> configuration file.
project center	All project-related functions are located in the project center, for example profiles and languages.
project editor	In the project editor you model the behavior and the appearance of the human machine interface.

## R

**resource** A resource is a data package that is part of the EB GUIDE project. Examples for resources are fonts, images, meshes. Resources are stored outside of the EB GUIDE model, for example in files, depending on the operating system.

**RomFS** Read-only memory file system

## S

**shared library** A shared library, as opposed to a static library, can be loaded when preparing a program for execution. On Windows platforms shared libraries are called dynamic link libraries and have a `.dll` file extension. On Unix systems shared libraries are called shared objects and have an `.so` file extension.

**state** A state defines the status of the state machine. States and state transitions are modeled in state charts.

**state machine** A state machine is a set of states, transitions between those states, and actions. A state machine describes the dynamic behavior of the system.

## T

**transition** A transition defines the change from one state to another. A transition is usually triggered by an event.

## U

**UI** User interface

## V

**view** A view is a graphical representation of a project-specific HMI-screen and is related to a specific state machine state. A view consists of a tree of widgets.

**VTA** View transition animation

## W

**widget** A widget is a basic graphical element. Widgets are used for interaction with a graphical user interface.

**WPF** Windows Presentation Foundation. See <https://docs.microsoft.com/en-us/dotnet/framework/wpf/>.



# Index

## Symbols

- .psd file format, 163
- 3D graphics, 38, 86, 87, 456
  - adding, 162
  - image-based lighting, 86
  - importing, 39, 295
  - meshes, 87
  - supported formats, 38, 86, 87
- 3D objects, 38
- 3D widgets, 38, 86, 87, 125
  - references, 391

## A

- absolute layout
  - references, 422
- actions
  - adding, 142
- ADAS ECU, 456
- ADASIS, 456
- adding entry animation, 155
- alpha mask
  - adding, 161
  - references, 380
- Alpha mask
  - adding, 161
- ambient lights
  - references, 391
- ambient texture
  - references, 432
- Animation, 41
  - adding, 153
- animation curves, 41
- animations, 41
  - adding, 289, 312
  - adding change animations, 155
  - change animations, 42
  - colors, 160
  - datapool items, 159
  - entry animations, 42, 154

- exit animations, 42, 154
- rearranging animations, 156
- references, 381
- script curves, 44
- view transitions, 154
- widgets, 153
- anisotropy
  - references, 433
- anti-aliasing, 44, 456
- API (see application programming interface)
- application programming interface, 45, 456
- applications, 456
- aspects, 456
  - languages, 74
  - skins, 103
- auto focus
  - references, 407
- auto-hide, 58

## B

- base color texture
  - references, 434
- basic widgets, 125
  - references, 379
- best practices
  - conditional scripts, 263
- boolean
  - data types, 323
- boolean lists
  - data types, 323
- border
  - references, 404
- box layout
  - references, 422
- buttons
  - user interface, 365

## C

- camera bloom
  - references, 429
- camera depth of field
  - references, 430

- camera viewport
  - references, 431
- cameras
  - references, 391
- change animations, 155
  - references, 379
- character
  - position, 187
- child visibility selection
  - references, 398
- circular text, 193
  - references, 405
- clear coat
  - reference, 431
- coloration
  - references, 405
- colors
  - data types, 323
- command area
  - project editor, 55
- command line, 235, 321
  - EB GUIDE Monitor options, 322
  - exporting, 228
  - options for EB GUIDE Studio, 321
  - syntax, 321
  - validating, 226
- command line interpreter (see command line)
- communication context, 456
  - adding, 217
  - reader application, 45
  - writer application, 45
- components
  - docking, 58
  - undocking, 58
- conditional scripts
  - data types, 324
- conditions
  - adding, 141
- console (see command line)
- constant curves
  - references, 382
- Container

- adding, 150
- containers
  - adding, 150
  - references, 387
- content area
  - project center, 47
  - project editor, 51
- copy
  - datapool items, 215
  - events, 210

## D

- data types
  - boolean, 323
  - boolean lists, 323
  - colors, 323
  - conditional scripts, 324
  - floats, 324
  - fonts, 325
  - Function () : bool, 325
  - ibl, 325
  - images, 326
  - integers, 326
  - linking, 172
  - lists, 327
  - mesh lists, 327
  - meshes, 327
  - references, 323
  - strings, 327
- datapool, 59, 457
  - accessing with EB GUIDE Script, 96
  - concepts, 59
  - datapool items, 59
  - windowed lists, 60
- datapool items, 457
  - adding, 214
  - adding language support, 291
  - animating, 158
  - changing in EB GUIDE Monitor, 246
  - copying, 215
  - deleting, 220
  - editing lists, 215

- exporting, 233
- importing, 234
- linking, 172, 218
- pasting, 215
- references, 323
- diffuse texture
  - references, 433
- directional lights
  - references, 391
- display
  - configuring, 232
- docking
  - components, 58
- dynamic state machines
  - adding, 134, 264

## E

- easing, 41
- EB GUIDE aware, 457
- EB GUIDE extensions (see extensions)
- EB GUIDE GTF, 457
- EB GUIDE GTF extensions, 69, 457
- EB GUIDE GTF SDK, 457
- EB GUIDE model, 457
  - simulating, 226
  - validating, 225
- EB GUIDE models, 60
  - .gdata, 61
  - export set, 227
  - exporting, 227
  - exporting languages, 227
  - exporting skins, 227
  - exporting using command line, 228
  - model elements, 61
  - storage format, 61
  - validating, 64, 65
- EB GUIDE Monitor, 65
  - changing datapool items, 246
  - changing language, 242
  - command line, 241
  - components, 56, 58
  - configuring, 242

- connection, 242
- exporting watch list, 252
- firing events, 245
- importing watch list, 252
- loading configuration, 244
- monitor.cfg, 244
- scripting example, 247
- simulating, 226
- stand-alone, 241
- starting, 241
- starting scripts, 251
- tabs, 56
- EB GUIDE Monitor extensions, 69, 457
- EB GUIDE product line, 457
- EB GUIDE project, 457
- EB GUIDE projects, 60
- EB GUIDE Script, 88, 457
  - best practices, 263
  - comments, 89
  - converting, 216
  - datapool access, 96
  - events, 99
  - expressions, 90
  - foreign function calls, 95
  - functions, 330
  - identifiers, 88
  - if-then-else, 94
  - keywords, 328
  - l-values, 92
  - lists, 98
  - local variables, 92
  - minimal evaluation, 94
  - operators, 329
  - prefixes, 88
  - r-values, 92
  - references, 328, 329, 330
  - scripted values, 101
  - short-circuit evaluation, 94
  - standard library, 101
  - string formatting, 101
  - todo comments, 89
  - types, 89

- using functions, 272
  - while loops, 93
  - widget properties, 97
- EB GUIDE SDK, 457
- EB GUIDE Studio, 457
- EB GUIDE Studio extensions, 68, 457
- EB GUIDE Studio SDK, 457
- EB GUIDE TF, 457
- effect
  - widget feature, 404
- Ellipse
  - adding, 146
  - editing, 147
- ellipses
  - references, 388
- emissive texture
  - references, 436
- enabled
  - references, 398
- entry actions (see states)
- entry animations, 155
  - references, 379
- event groups
  - adding, 211
- event system, 66
- events, 66
  - adding, 209
  - adding parameters, 210
  - copying, 210
  - deleting, 213
  - EB GUIDE Script functions, 99
  - event group, 66
  - event ID, 66
  - firing in EB GUIDE Monitor, 245
  - key events, 363
  - mapping keys, 211
  - pasting, 210
  - references, 363
- exit actions (see states)
- exit animations, 154
  - references, 379
- export

- language-dependent texts, 233
- extension developers, 19
  - required knowledge, 19
- extensions, 68, 457
  - EB GUIDE model, 256
  - EB GUIDE Studio, 255
  - EB GUIDE UI, 258

## F

- fast start curves
  - references, 382
- finger ID, 122
- flick gesture
  - references, 408
- floats
  - data types, 324
- flow layout
  - references, 423
- focused
  - references, 398
- font
  - changing, 184
- Font metrics
  - ascender, 187
  - descender, 187
- font metrics
  - references, 399
- font settings
  - changing, 183
- fonts, 83
  - bitmap fonts, 83
  - data types, 325
  - multifont support, 84
  - opentype fonts, 83
  - truetype fonts, 83
- Function () : bool
  - data types, 325

## G

- gamma
  - rendering, 300
- gamma correction

- concepts, 71
  - rendering, 71
- gesture IDs
  - references, 411
- gestures, 121
  - finger ID, 122
  - multi-touch input, 122
  - non-path gestures, 121
  - path gestures, 122
  - references, 408, 414
- Global Positioning System, 458
- graphics library, 458
- grid layout
  - references, 424
- GUI (see user interface)

## H

- HMI model, 458
- hold gesture
  - references, 409
- human machine interface, 458

## I

- IBL (see image-based lighting)
- ibl
  - data types, 325
- IBLGenerator, 73, 458
- icons
  - user interface, 365
- Image
  - adding, 148
- Image-based light, 73
  - adding, 164
- image-based lighting, 73, 86
  - IBLGenerator, 73
  - importing, 164
  - rendering, 74
- image-based lightings, 458
- image-based lights
  - references, 392
- images
  - 9-patch, 86

- adding, 148
  - data types, 326
  - references, 388
  - supported formats, 86
- import
  - language-dependent texts, 234
- Instantiator
  - adding, 151
- instantiators
  - adding, 151
  - line template, 151
  - line templates, 389
  - references, 389
  - using, 281
- integers
  - data types, 326
- inter-process communication, 458
- internal transitions
  - adding, 143
- IPC, 458

## K

- key pressed
  - references, 414
- key released
  - references, 414
- key status changed
  - references, 415
- key unicode
  - references, 415
- keyboard keys, 211
- keyboard shortcuts (see shortcuts)

## L

- Label
  - adding, 149
- label
  - character position, 187
  - circular text, 193
  - text position, 187
- Label settings
  - changing, 183

- labels
  - adding, 149
  - references, 389
- language-dependent texts, 291
  - exporting, 233
  - importing, 234
- languages, 194
  - adding, 194
  - changing, 228, 291
  - deleting, 195
  - export, 74, 196
  - exporting, 75, 227
  - importing, 75
  - language support, 74
- layout margins
  - references, 424
- libraries (see extensions)
  - adding, 229
- light map texture
  - references, 437
- line index
  - references, 427
- line spacing
  - changing, 185
- linear curves
  - references, 386
- linear interpolation curves
  - references, 387
- linear interpolation integer curve
  - adding, 289
- lineGap, 185
- lineOffset, 185
- links
  - datapool items, 218
  - widget property to datapool item, 171
  - widget property to widget property, 168
- list index
  - references, 427
- list layout
  - references, 425
- lists
  - creating, 281

- data types, 327
- editing, 215
- linking, 172
- long hold gesture
  - references, 410

## M

- materials
  - PBR GGX materials, 393, 459
  - PBR Phong materials, 394, 459
  - references, 392, 393, 394
- mesh lists
  - data types, 327
- meshes, 87
  - data types, 327
  - references, 393
- metallic texture
  - references, 438
- model elements, 61, 459
  - deleting, 138
  - renaming, 224
  - renaming globally, 224
- model interface, 60, 459
  - importing, 238
- model interfaces, 80
  - adding, 236
  - datapool items, 80
  - deleting, 239
  - event groups, 81
  - events, 80
  - exporting, 237
  - namespaces, 81
- model view controller, 459
- modelers, 18
  - required knowledge, 18
- move in
  - references, 416
- move out
  - references, 416
- move over
  - references, 417
- moveable

- references, 417
- multifont support, 84, 459
  - adding, 190
  - adding for type list font, 191
- multiple lines
  - references, 399
- multisampling (see anti-aliasing)

## N

- namespaces, 78, 459
  - adding, 206
  - adding model elements, 207
  - deleting, 208
  - moving model elements, 207
  - renaming, 206
- naming
  - model elements, 64
- navigation area
  - project center, 46
- navigation component
  - project editor, 48
- normal map texture
  - references, 439

## O

- opaque texture
  - references, 440
- operating systems, 459

## P

- paste
  - datapool items, 215
  - events, 210
- path gesture
  - references, 410
- path gestures
  - adding, 279
  - references, 411
- Photoshop file format, 82
  - extracting, 82
  - importing, 82
- pinch gesture

- references, 412
- pivot
  - references, 450
- plug-ins (see extensions)
- point lights
  - references, 395
- pop-up off animations
  - references, 379
- pop-up on animations
  - references, 379
- pressed
  - references, 400
- problems component
  - project editor, 56
- profiles, 228, 459
  - adding, 229
  - cloning, 229
- progress bar
  - modelling, 316
- project center, 46, 459
  - content area, 47
  - navigation area, 46
- project editor, 47, 459
  - command area, 54
  - content area, 51
  - navigation component, 48
  - problems component, 56
  - templates component, 56
  - toolbox, 50
  - toolbox component, 50
  - VTA component, 56
- projects
  - creating, 222
  - exporting, 226
  - opening, 222
  - simulating, 226
  - validating, 225
- properties component
  - best practices, 263
  - command area, 50
  - project editor, 50
- PSD (see Photoshop file format)

## Q

quadratic curves  
 references, 384

## R

read-only memory file system, 459  
 rectangles  
 references, 390  
 reflectance texture  
 references, 445  
 reflection texture  
 references, 441  
 renderer  
 configuring, 232  
 rendering  
 gamma correction, 71  
 resource  
 image-based lighting, 86  
 resource management (see resources)  
 resources, 82, 459  
 .psd file format, 82  
 3D graphics, 86, 87  
 fonts, 83  
 images, 86  
 meshes, 87  
 Photoshop file format, 82  
 PSD, 82  
 revision control system, 61  
 rotary  
 references, 418  
 rotate gesture  
 references, 413  
 rotation  
 references, 450  
 roughness texture  
 references, 442

## S

scale mode  
 references, 426  
 scaling  
 references, 451

Scene graph, 38  
 adding, 162  
 customizing, 295  
 scene graph nodes  
 references, 396  
 scene graphs, 38, 86, 87  
 adding, 162  
 customizing, 295  
 references, 396  
 scenes  
 configuring, 232, 371  
 references, 371  
 Screen space ambient occlusion  
 references, 448  
 script curve  
 adding, 312  
 script curves, 44  
 references, 385  
 scripted values, 101  
 best practices, 263  
 converting, 216  
 selected  
 references, 401  
 selection group  
 references, 401  
 shared libraries, 460  
 shearing  
 references, 451  
 shell (see command line)  
 shininess texture  
 references, 443  
 shortcuts  
 user interface, 372  
 simulation, 226  
 sinus curves  
 references, 385  
 skins  
 adding, 197  
 deleting, 199  
 export, 103, 200  
 exporting, 227  
 support, 103



- switching, 199
- slow start curve
  - references, 383
- specular texture
  - references, 444
- spinning
  - references, 402
- spot lights
  - references, 397
- state, 268
- state machines, 104, 460
  - adding, 131
  - adding actions to transitions, 142
  - adding conditions to transitions, 141
  - adding dynamic state machines, 134
  - adding internal transitions, 143
  - comparison to UML, 119
  - deleting, 133
  - Dynamic state machine, 104
  - entry actions, 132
  - executing, 116
  - exit actions, 132
  - haptic, 104
  - Haptic state machine, 104
  - logic, 104
  - Logic state machine, 104
  - Main state machine, 104
  - states, 106, 116
  - Transition, 113
  - transitions, 113, 116
  - UML 2.5 notation, 119
  - using transitions, 139
  - using triggers for transitions, 141
- states, 106, 116, 460
  - adding, 135
  - adding a Choice state, 136
  - adding to a Compound state, 135
  - Choice state, 108
  - choice states, 108
  - Compound state, 107
  - compound states, 107
  - connecting, 139

- Deep history state, 109
- deep history states, 109
- deleting, 138
- entry actions, 137
- exit actions, 137
- Final state, 112
- final states, 112
- history states, 109
- Initial state, 107
- initial states, 107
- Shallow history state, 109
- shallow history states, 109
- using transitions, 138
- View state, 107
- view states, 107
- strings
  - data types, 327
- stroke
  - references, 406

## T

- template index
  - references, 428
- template interface
  - adding property, 202
  - removing property, 202
- templates, 127
  - adding, 201
  - creating, 201
  - deleting, 203
  - template interface, 202
  - using, 203
  - widget template interface, 128
- templates component
  - project editor, 56
- text
  - position, 187
- text truncation
  - references, 402
- tone mapping
  - references, 447
- toolbox (see toolbox component)

- toolbox component
  - project editor, 50
- touch gestures (see gestures)
- touch input (see gestures)
- touch lost
  - references, 418
- touch move
  - references, 419
- touch pressed
  - references, 420
- touch released
  - references, 420
- touch status changed
  - references, 421
- touched
  - references, 403
- transitions, 113, 116, 460
  - adding, 139
  - adding actions, 142
  - adding conditions, 141
  - adding internal transitions, 143
  - changing priority, 144
  - labels, 140
  - moving, 139
  - shape, 140
  - types, 114
  - using triggers, 141
- translation
  - references, 451
- triggers
  - defining, 141

## U

- UI (see user interface)
- undocking
  - components, 58
- user interface, 45, 458
  - problems component, 225
- user-defined focus
  - references, 407
- user-defined properties
  - adding, 175

- adding Function (:): bool, 176
- renaming, 177

## V

- version control system, 61
- View
  - adding, 145
- view states
  - references, 379
- view templates
  - references, 378, 379
- view transition animation
  - adding, 303
- view transition animations, 154, 460
- viewport
  - references, 428
- views, 123, 460
  - adding, 145
  - references, 378
- visibility, 398 (see widgets visibility)
- visibility group, 398
- VTA (see view transition animations)
- VTA component
  - project editor, 56

## W

- watch lists (see EB GUIDE Monitor)
- widget feature, 127
- widget features, 126, 129
  - adding, 180
  - adding path gesture, 279
  - focus, 70
  - list management, 130
  - removing, 182
- widget properties, 126
  - accessing with EB GUIDE Script, 97
  - adding user-defined properties, 175
  - linking to datapool item, 171, 172
  - linking to widget property, 168
  - templates, 128
  - type list, 177
  - user-defined property, 175

- widget property
  - default property, 127
  - user-defined property, 127
  - widget feature property, 127
- widget templates (see templates)
- widgets, 123, 460
  - 3D widgets, 125
  - adding, 146
  - basic, 125
  - custom widgets, 125
  - deleting, 166
  - grouping, 150
  - Instantiator, 130
  - instantiators, 130
  - linking, 76
  - linking to datapool item, 171, 172
  - linking to widget property, 168
  - managing visibility, 178
  - positioning, 166
  - resizing, 167
  - templates, 125
  - types, 123
  - View, 123
  - views, 123
  - widget features, 126, 129
  - widget properties, 126
  - widget templates, 128
- widgets visibility, 178
  - multiple child widgets, 179
  - single child widget, 179
- windowed lists, 60