



Elektrobit

# EB GUIDE Studio

User manual

Version 6.4.1.0



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.

ProOSEK®, tresos®, and street director® are registered trademarks of Elektrobit Automotive GmbH.

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

Copyright 2017, Elektrobit Automotive GmbH.

# Table of Contents

1. About this documentation .....	14
1.1. Target audience: Modelers .....	14
1.2. Structure of user documentation .....	14
1.3. Typography and style conventions .....	15
1.4. Naming conventions .....	17
2. Safe and correct use .....	18
2.1. Intended use .....	18
2.2. Possible misuse .....	18
3. Support .....	19
4. Introduction to EB GUIDE .....	20
4.1. The EB GUIDE product line .....	20
4.2. EB GUIDE Studio .....	20
4.2.1. Modeling HMI behavior .....	20
4.2.2. Modeling HMI appearance .....	21
4.2.3. Handling data .....	21
4.2.4. Simulating the EB GUIDE model .....	21
4.2.5. Exporting the EB GUIDE model .....	22
4.3. EB GUIDE TF .....	22
5. Tutorial: Getting started .....	24
5.1. Starting EB GUIDE .....	24
5.2. Creating a project .....	25
5.3. Modeling HMI behavior .....	26
5.4. Modeling HMI appearance .....	29
5.5. Starting the simulation .....	32
6. Background information .....	33
6.1. 3D graphics .....	33
6.1.1. Supported 3D graphic formats .....	33
6.1.2. Settings for 3D graphic files .....	33
6.1.3. Import of a 3D graphic file .....	34
6.2. Animations .....	35
6.2.1. Animations for widgets .....	35
6.2.2. Animations for view transitions .....	36
6.3. Application programming interface between application and model .....	36
6.4. Communication context .....	37
6.5. Components of the graphical user interface .....	37
6.5.1. Project center .....	37
6.5.1.1. Navigation area .....	38
6.5.1.2. Content area .....	38
6.5.2. Project editor .....	39

6.5.2.1. Navigation area .....	39
6.5.2.2. Content area .....	41
6.5.2.3. Command area .....	42
6.5.2.4. Toolbox .....	43
6.5.2.5. Properties panel .....	44
6.5.2.6. Status bar .....	45
6.5.2.7. Problems area .....	45
6.6. Datapool .....	45
6.6.1. Concept .....	45
6.6.2. Datapool items .....	46
6.6.3. Windowed lists .....	46
6.7. EB GUIDE model and EB GUIDE project .....	47
6.8. Event handling .....	48
6.8.1. Event system .....	48
6.8.2. Events .....	48
6.9. Extensions .....	48
6.9.1. EB GUIDE Studio extension .....	49
6.9.2. EB GUIDE GTF extension .....	49
6.10. Languages .....	49
6.10.1. Display languages in EB GUIDE Studio .....	49
6.10.2. Languages in the EB GUIDE model .....	49
6.10.3. Export and import of language dependent texts .....	50
6.11. Resource management .....	50
6.11.1. Fonts .....	51
6.11.2. Images .....	51
6.11.2.1. 9-patch images .....	51
6.11.3. Meshes for 3D graphics .....	52
6.12. Scripting language EB GUIDE Script .....	52
6.12.1. Capabilities and areas of application .....	53
6.12.2. Namespaces and identifiers .....	53
6.12.3. Comments .....	54
6.12.4. Types .....	54
6.12.5. Expressions .....	55
6.12.6. Constants and references .....	55
6.12.7. Arithmetic and logic expressions .....	56
6.12.8. L-values and r-values .....	57
6.12.9. Local variables .....	57
6.12.10. While loops .....	58
6.12.11. If-then-else .....	59
6.12.12. Foreign function calls .....	60
6.12.13. Datapool access .....	61
6.12.14. Widget properties .....	62

6.12.15. Lists .....	63
6.12.16. Events .....	63
6.12.17. String formatting .....	65
6.12.18. The standard library .....	66
6.13. Scripted values .....	66
6.14. Shortcuts, buttons and icons .....	67
6.14.1. Shortcuts .....	67
6.14.2. Buttons .....	68
6.14.3. Icons .....	69
6.15. State machines and states .....	70
6.15.1. State machines .....	70
6.15.1.1. Haptic state machine .....	70
6.15.1.2. Logic state machine .....	70
6.15.1.3. Dynamic state machine .....	70
6.15.2. States .....	71
6.15.2.1. Compound state .....	71
6.15.2.2. View state .....	72
6.15.2.3. Initial state .....	72
6.15.2.4. Final state .....	73
6.15.2.5. Choice state .....	74
6.15.2.6. History states .....	75
6.15.3. Transitions .....	78
6.15.4. Execution of a state machine .....	82
6.15.5. EB GUIDE notation in comparison to UML notation .....	86
6.15.5.1. Supported elements .....	87
6.15.5.2. Not supported elements .....	87
6.15.5.3. Deviations .....	87
6.16. Touch input .....	88
6.16.1. Non-path gestures .....	88
6.16.2. Path gestures .....	88
6.16.3. Input processing and gestures .....	89
6.16.4. Multi-touch input .....	89
6.17. Widgets .....	90
6.17.1. View .....	90
6.17.2. Widget categories .....	91
6.17.3. Widget properties .....	92
6.17.4. Widget templates .....	93
7. Modeling HMI behavior .....	95
7.1. Modeling a state machine .....	95
7.1.1. Adding a state machine .....	95
7.1.2. Adding a dynamic state machine .....	95
7.1.3. Defining an entry action for a state machine .....	96

7.1.4. Defining an exit action for a state machine .....	96
7.1.5. Deleting a state machine .....	97
7.2. Modeling states .....	97
7.2.1. Adding a state .....	97
7.2.2. Adding a state to a compound state .....	98
7.2.3. Adding a choice state .....	99
7.2.4. Defining an entry action for a state .....	100
7.2.5. Defining an exit action for a state .....	101
7.2.6. Deleting a model element from a state machine .....	102
7.3. Connecting states through transitions .....	102
7.3.1. Adding a transition between two states .....	102
7.3.2. Moving a transition .....	103
7.3.3. Defining a trigger for a transition .....	104
7.3.4. Adding a condition to a transition .....	105
7.3.5. Adding an action to a transition .....	106
7.3.6. Adding an internal transition to a state .....	108
8. Modeling HMI appearance .....	109
8.1. Working with widgets .....	109
8.1.1. Adding a view .....	109
8.1.2. Adding a widget to a view .....	109
8.1.3. Deleting a widget from a view .....	110
8.1.4. Adding an image to a view .....	111
8.1.5. Adding a scene graph to a view .....	111
8.1.6. Changing the font of a label .....	112
8.1.7. Grouping widgets using a container .....	113
8.1.8. Adding an instantiator to a view .....	113
8.2. Working with widget properties .....	114
8.2.1. Positioning a widget .....	114
8.2.2. Resizing a widget .....	115
8.2.3. Linking between widget properties .....	116
8.2.4. Linking a widget property to a datapool item .....	118
8.2.5. Adding a user-defined property to a widget .....	119
8.2.5.1. Adding a user-defined property of type <code>Function (): bool</code> .....	120
8.2.6. Renaming a user-defined property .....	121
8.3. Extending a widget by widget features .....	122
8.3.1. Adding a widget feature .....	122
8.3.2. Removing a widget feature .....	124
8.4. Adding a language to the EB GUIDE model .....	125
8.4.1. Adding a language .....	126
8.4.2. Deleting a language .....	126
8.5. Adding animations .....	127
8.5.1. Animating a widget .....	127

8.5.2. Animating a view transition .....	129
8.6. Re-using a widget .....	130
8.6.1. Adding a template .....	130
8.6.2. Defining the template interface .....	131
8.6.3. Using a template .....	132
8.6.4. Deleting a template .....	132
9. Handling data .....	134
9.1. Adding an event .....	134
9.2. Adding a parameter to an event .....	134
9.3. Addressing an event .....	135
9.4. Deleting an event .....	136
9.5. Adding a datapool item .....	136
9.6. Editing datapool items of a list type .....	137
9.7. Converting a property to a scripted value .....	138
9.8. Establishing external communication .....	139
9.9. Linking between datapool items .....	141
9.10. Deleting a datapool item .....	142
10. Handling a project .....	143
10.1. Creating a project .....	143
10.2. Opening a project .....	143
10.2.1. Opening a project from the file explorer .....	143
10.2.2. Opening a project within EB GUIDE Studio .....	144
10.3. Testing and improving an EB GUIDE model .....	144
10.3.1. Validating an EB GUIDE model .....	144
10.3.2. Starting and stopping the simulation .....	145
10.4. Exporting an EB GUIDE model .....	146
10.5. Changing the display language of EB GUIDE Studio .....	147
10.6. Configuring profiles .....	147
10.6.1. Cloning a profile .....	148
10.6.2. Adding a library .....	148
10.6.3. Adding messages .....	150
10.6.4. Configuring a scene .....	151
10.7. Exporting and importing language dependent texts .....	151
10.7.1. Exporting language dependent texts .....	152
10.7.2. Importing language dependent texts .....	153
11. Tutorials .....	155
11.1. Tutorial: Adding a dynamic state machine .....	155
11.2. Tutorial: Modeling button behavior with EB GUIDE Script .....	163
11.3. Tutorial: Modeling a path gesture .....	169
11.4. Tutorial: Creating a list with dynamic content .....	172
11.5. Tutorial: Making a rectangle move across the screen .....	178
11.6. Tutorial: Adding a language dependent text to a datapool item .....	182



- 11.7. Tutorial: Working with a 3D graphic ..... 185
- 12. References ..... 191
  - 12.1. Android events ..... 191
  - 12.2. Datapool items ..... 192
  - 12.3. Data types ..... 193
    - 12.3.1. Mesh ..... 193
    - 12.3.2. Boolean ..... 193
    - 12.3.3. Color ..... 193
    - 12.3.4. Conditional script ..... 194
    - 12.3.5. Float ..... 194
    - 12.3.6. Font ..... 195
    - 12.3.7. Image ..... 195
    - 12.3.8. Integer ..... 195
    - 12.3.9. List ..... 196
    - 12.3.10. String ..... 197
  - 12.4. EB GUIDE Script ..... 197
    - 12.4.1. EB GUIDE Script keywords ..... 197
    - 12.4.2. EB GUIDE Script operator precedence ..... 198
    - 12.4.3. EB GUIDE Script standard library ..... 199
      - 12.4.3.1. EB GUIDE Script functions A ..... 199
        - 12.4.3.1.1. abs ..... 199
        - 12.4.3.1.2. absf ..... 199
        - 12.4.3.1.3. acosf ..... 200
        - 12.4.3.1.4. animation\_before ..... 200
        - 12.4.3.1.5. animation\_beyond ..... 200
        - 12.4.3.1.6. animation\_cancel ..... 201
        - 12.4.3.1.7. animation\_cancel\_end ..... 201
        - 12.4.3.1.8. animation\_cancel\_reset ..... 201
        - 12.4.3.1.9. animation\_pause ..... 201
        - 12.4.3.1.10. animation\_play ..... 202
        - 12.4.3.1.11. animation\_reverse ..... 202
        - 12.4.3.1.12. animation\_running ..... 202
        - 12.4.3.1.13. animation\_set\_time ..... 202
        - 12.4.3.1.14. asinf ..... 203
        - 12.4.3.1.15. atan2f ..... 203
        - 12.4.3.1.16. atan2i ..... 203
        - 12.4.3.1.17. atanf ..... 203
      - 12.4.3.2. EB GUIDE Script functions C - H ..... 204
        - 12.4.3.2.1. ceil ..... 204
        - 12.4.3.2.2. changeDynamicStateMachinePriority ..... 204
        - 12.4.3.2.3. character2unicode ..... 204
        - 12.4.3.2.4. clearAllDynamicStateMachines ..... 205



12.4.3.2.5. color2string .....	205
12.4.3.2.6. cosf .....	205
12.4.3.2.7. deg2rad .....	206
12.4.3.2.8. expf .....	206
12.4.3.2.9. float2string .....	206
12.4.3.2.10. floor .....	206
12.4.3.2.11. focusNext .....	207
12.4.3.2.12. focusPrevious .....	207
12.4.3.2.13. format_float .....	207
12.4.3.2.14. format_int .....	208
12.4.3.2.15. getLineCount .....	209
12.4.3.2.16. getTextHeight .....	209
12.4.3.2.17. getTextLength .....	209
12.4.3.2.18. getTextWidth .....	210
12.4.3.2.19. has_list_window .....	210
12.4.3.2.20. hsba2color .....	210
<b>12.4.3.3. EB GUIDE Script functions I - R .....</b>	<b>211</b>
12.4.3.3.1. int2float .....	211
12.4.3.3.2. int2string .....	211
12.4.3.3.3. isDynamicStateMachineActive .....	211
12.4.3.3.4. language .....	211
12.4.3.3.5. localtime_day .....	212
12.4.3.3.6. localtime_hour .....	212
12.4.3.3.7. localtime_minute .....	212
12.4.3.3.8. localtime_month .....	212
12.4.3.3.9. localtime_second .....	213
12.4.3.3.10. localtime_weekday .....	213
12.4.3.3.11. localtime_year .....	213
12.4.3.3.12. log10f .....	213
12.4.3.3.13. logf .....	214
12.4.3.3.14. nearbyint .....	214
12.4.3.3.15. popDynamicStateMachine .....	214
12.4.3.3.16. powf .....	214
12.4.3.3.17. pushDynamicStateMachine .....	215
12.4.3.3.18. rad2deg .....	215
12.4.3.3.19. rand .....	215
12.4.3.3.20. request_runlevel .....	215
12.4.3.3.21. rgba2color .....	216
12.4.3.3.22. round .....	216
<b>12.4.3.4. EB GUIDE Script functions S - W .....</b>	<b>216</b>
12.4.3.4.1. seed_rand .....	216
12.4.3.4.2. sinf .....	217

12.4.3.4.3. sqrtf .....	217
12.4.3.4.4. string2float .....	217
12.4.3.4.5. string2int .....	218
12.4.3.4.6. string2string .....	218
12.4.3.4.7. substring .....	218
12.4.3.4.8. system_time .....	219
12.4.3.4.9. system_time_ms .....	219
12.4.3.4.10. tanf .....	219
12.4.3.4.11. trace_dp .....	219
12.4.3.4.12. trace_string .....	220
12.4.3.4.13. transformToScreenX .....	220
12.4.3.4.14. transformToScreenY .....	220
12.4.3.4.15. transformToWidgetX .....	221
12.4.3.4.16. transformToWidgetY .....	221
12.4.3.4.17. trunc .....	221
12.4.3.4.18. widgetGetChildCount .....	222
12.5. Events .....	222
12.6. gtfStartup.cfg configuration file .....	222
12.6.1. Mapping rule structure .....	222
12.6.2. Signals .....	223
12.6.3. Actions .....	223
12.6.4. Messages .....	225
12.7. Scenes .....	229
12.8. Touch screen types supported by EB GUIDE GTF .....	231
12.9. Widgets .....	231
12.9.1. View .....	231
12.9.2. Basic widgets .....	232
12.9.2.1. Label .....	233
12.9.2.2. Rectangle .....	233
12.9.2.3. Image .....	234
12.9.2.4. Container .....	234
12.9.2.5. Instantiator .....	235
12.9.3. Animations .....	235
12.9.3.1. Animation .....	235
12.9.3.2. Constant curves .....	236
12.9.3.3. Fast start curves .....	237
12.9.3.4. Slow start curves .....	237
12.9.3.5. Quadratic curves .....	238
12.9.3.6. Sinus curves .....	238
12.9.3.7. Script curves .....	239
12.9.3.8. Linear curves .....	239
12.9.3.9. Linear interpolation curves .....	240



- 12.9.4. 3D widgets ..... 240
  - 12.9.4.1. Scene graph ..... 241
  - 12.9.4.2. Scene graph node ..... 241
  - 12.9.4.3. Camera ..... 242
  - 12.9.4.4. Directional light ..... 242
  - 12.9.4.5. Material ..... 242
  - 12.9.4.6. Mesh ..... 243
  - 12.9.4.7. Point light ..... 243
  - 12.9.4.8. Spot light ..... 243
- 12.10. Widget features ..... 244
  - 12.10.1. Common ..... 244
    - 12.10.1.1. Child visibility selection ..... 244
    - 12.10.1.2. Enabled ..... 245
    - 12.10.1.3. Focused ..... 245
    - 12.10.1.4. Pressed ..... 245
    - 12.10.1.5. Selected ..... 245
    - 12.10.1.6. Selection group ..... 246
    - 12.10.1.7. Spinning ..... 246
    - 12.10.1.8. Touched ..... 247
    - 12.10.1.9. Text truncation ..... 248
    - 12.10.1.10. Multiple lines ..... 249
  - 12.10.2. Effects ..... 249
    - 12.10.2.1. Border ..... 249
    - 12.10.2.2. Coloration ..... 249
  - 12.10.3. Focus ..... 250
    - 12.10.3.1. User-defined focus ..... 250
    - 12.10.3.2. Auto focus ..... 251
  - 12.10.4. Gestures ..... 252
    - 12.10.4.1. Flick gesture ..... 252
    - 12.10.4.2. Hold gesture ..... 252
    - 12.10.4.3. Long hold gesture ..... 253
    - 12.10.4.4. Path gestures ..... 253
      - 12.10.4.4.1. Gesture IDs ..... 254
    - 12.10.4.5. Pinch gesture ..... 255
    - 12.10.4.6. Rotate gesture ..... 256
  - 12.10.5. Input handling ..... 256
    - 12.10.5.1. Move over ..... 256
    - 12.10.5.2. Move out ..... 257
    - 12.10.5.3. Move in ..... 257
    - 12.10.5.4. Touch pressed ..... 257
    - 12.10.5.5. Touch released ..... 258
    - 12.10.5.6. Touch lost ..... 258



12.10.5.7. Touch status changed .....	258
12.10.5.8. Touch move .....	259
12.10.5.9. Gestures .....	259
12.10.5.10. Key pressed .....	260
12.10.5.11. Key unicode .....	260
12.10.5.12. Key released .....	260
12.10.5.13. Key status changed .....	260
12.10.5.14. Rotary .....	261
12.10.5.15. Moveable .....	261
12.10.6. Layout .....	262
12.10.6.1. Absolute layout .....	262
12.10.6.2. Box layout .....	262
12.10.6.3. Flow layout .....	262
12.10.6.4. Grid layout .....	263
12.10.6.5. Layout margins .....	263
12.10.6.6. List layout .....	264
12.10.6.7. Scale mode .....	265
12.10.7. List management .....	265
12.10.7.1. Line index .....	265
12.10.7.2. List index .....	265
12.10.7.3. Template index .....	266
12.10.7.4. View port .....	266
12.10.8. 3D .....	266
12.10.8.1. Camera viewport .....	266
12.10.8.2. Ambient texture .....	267
12.10.8.3. Diffuse texture .....	267
12.10.8.4. Emissive texture .....	268
12.10.8.5. Light map texture .....	268
12.10.8.6. Normal map texture .....	269
12.10.8.7. Opaque texture .....	269
12.10.8.8. Reflection texture .....	269
12.10.8.9. Specular texture .....	270
12.10.9. Transformations .....	271
12.10.9.1. Pivot .....	271
12.10.9.2. Rotation .....	271
12.10.9.3. Scaling .....	272
12.10.9.4. Shearing .....	272
12.10.9.5. Translation .....	272
13. Installation .....	274
13.1. Background information .....	274
13.1.1. Restrictions .....	274
13.1.2. System requirements .....	274



---

13.2. Downloading from EB Command .....	275
13.3. Installing EB GUIDE .....	276
13.4. Uninstalling EB GUIDE .....	277
Glossary .....	278
Index .....	282

# 1. About this documentation

## 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

## 1.2. 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 technological parameters and tables as well as the EB GUIDE Monitor API documentation.





▶ 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

Throughout the documentation you will see that words and phrases are displayed in bold or italic font, or in monospaced font. To find out what these conventions mean, please consult the following table. All default text is written in Arial Regular font without any markup.

Convention	Item is used	Example
Arial italics	to emphasize	If your project's release version is mixed, all content types are available. It is thus called <i>mixed version</i> .
Arial boldface	for menus and submenus	Select the <b>Options</b> menu.
Arial boldface	for buttons	Select <b>OK</b> .
Arial boldface	for keyboard keys	Press <b>F2</b> .
Arial boldface	for keyboard combination of keys	Press <b>Ctrl+Alt+Delete</b> .
Arial boldface	for commands	Convert the <code>.xdm</code> file to the newer version by using the <b>legacy convert</b> command.
Monospaced font (Courier)	for file names, directory names and chapter names	Put your script in the <code>function_name\abcdirectory</code> .
Monospaced font (Courier)	for code	<pre>CC_FILES_TO_BUILD =(PROJECT_- PATH)\source\network\can_node.- c CC_FILES_TO_BUILD += \$(PROJECT_- PATH)\source\network\can_config.c</pre>

Convention	Item is used	Example
Monospaced font (Courier)	for function names, methods, or routines	The <code>cos</code> function finds the cosine of each array element. Syntax line example is <code>MLGetVar ML_var_name</code> .
Monospaced font (Courier)	for user input or variable text	Enter a <code>three-digit prefix</code> in the menu line.
Square brackets [ ]	for optional parameters or for command syntax with optional parameters	<code>insertBefore [&lt;opt&gt;]</code>
Curly brackets { }	for mandatory parameters or for command syntax with mandatory parameters	<code>insertBefore {&lt;file&gt;}</code>
Three dots ...	for further parameters or for command syntax	<code>insertBefore [&lt;opt&gt;...]</code>
Warning	to warn about danger of death or severe personal injury	<p><b>WARNING</b> This is an example for a warning</p>  <p>This is what a warning looks like.</p>
Caution	to warn about danger of slight personal injury or material damage	<p><b>CAUTION</b> This is an example for a caution</p>  <p>This is what a caution looks like.</p>
Notice	to give additional but not vital information on a subject	<p><b>NOTE</b> This is an example for a notice</p>  <p>This is what a notice looks like.</p>
Tip	to provide helpful hints and tips	<p><b>TIP</b> This is an example for a tip</p>  <p>This is what a tip looks like.</p>
Example	to demonstrate or illustrate information	<p><b>E</b> <b>Example 1.1.</b>  <b>This is an example</b></p> <p>This is what an example looks like.</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 documentation the following directory names are used:

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

For example:

```
C:\Program Files\Elektrobit\EB GUIDE Studio 6.2
```

- ▶ The directory 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 6.4\platform\win32
```

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

For example:

```
C:\Users\[user name]\Documents\EB GUIDE 6.4\projects\
```

## 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 EB 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 EB for clarification at [chapter 3, "Support"](#).



## 3. Support

EB GUIDE support is available in the following ways.

▶ For community edition:

Find comprehensive information in our articles, blogs, and forums.

▶ For enterprise edition:

Contact us according to your support contract.

When you look for support, prepare the version number of your EB GUIDE installation. To find the version number, 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 or speech 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 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.

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

The EB GUIDE model that is created with EB GUIDE Studio and the exported EB GUIDE model that is executed on the EB GUIDE TF are completely separated. They interact with each other, but cannot block one another.

### 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 and the speech user interface of the EB GUIDE model.

### Widgets

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.

### Spidgets

To create a speech user interface EB GUIDE Studio offers spidgets. Spidgets are used to specify the fundamental parts of a speech dialog. Speech recognition as user input and speech synthesis as system output. A prompt spidget allows the modeling of text that is played through a text-to-speech synthesizer (TTS). A command spidget allows the modeling of grammars that describe what a speech recognizer understands. Related spidgets are grouped together through model elements. This group is called talk.

## 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 that needs to be displayed and all other internal information. Datapool items store and exchange data.

### Event system

Events are temporary triggers. Events can be sent to both parties 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.

## 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 ASCII files.

## 4.3. EB GUIDE TF

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

Depending on the project type selected in EB GUIDE Studio you execute:

- ▶ EB GUIDE GTF

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

- ▶ EB GUIDE STF

EB GUIDE Speech Target Framework is the run-time environment executing speech functionality in the 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 the 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. An EB GUIDE model contains element properties such as geometry and lighting. The data contained in the exported EB GUIDE model 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. The EB GUIDE TF supports various renderers for different platforms.



▶ Audio abstraction

The speech user interface requires access to audio hardware. The audio abstraction provides access to microphones and speakers. EB GUIDE STF implements speech recognition and text-to-speech synthesis. For this purpose EB GUIDE STF incorporates third-party speech engines.

## 5. Tutorial: Getting started

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.

### 5.1. Starting EB GUIDE



#### Starting EB GUIDE

Prerequisite:

- EB GUIDE is installed.

#### Step 1

In the Windows **Start** menu, click **All Programs**.

#### Step 2

In the **Elektrobit** menu, click the version you want to start.

EB GUIDE Studio starts. The project center is displayed.



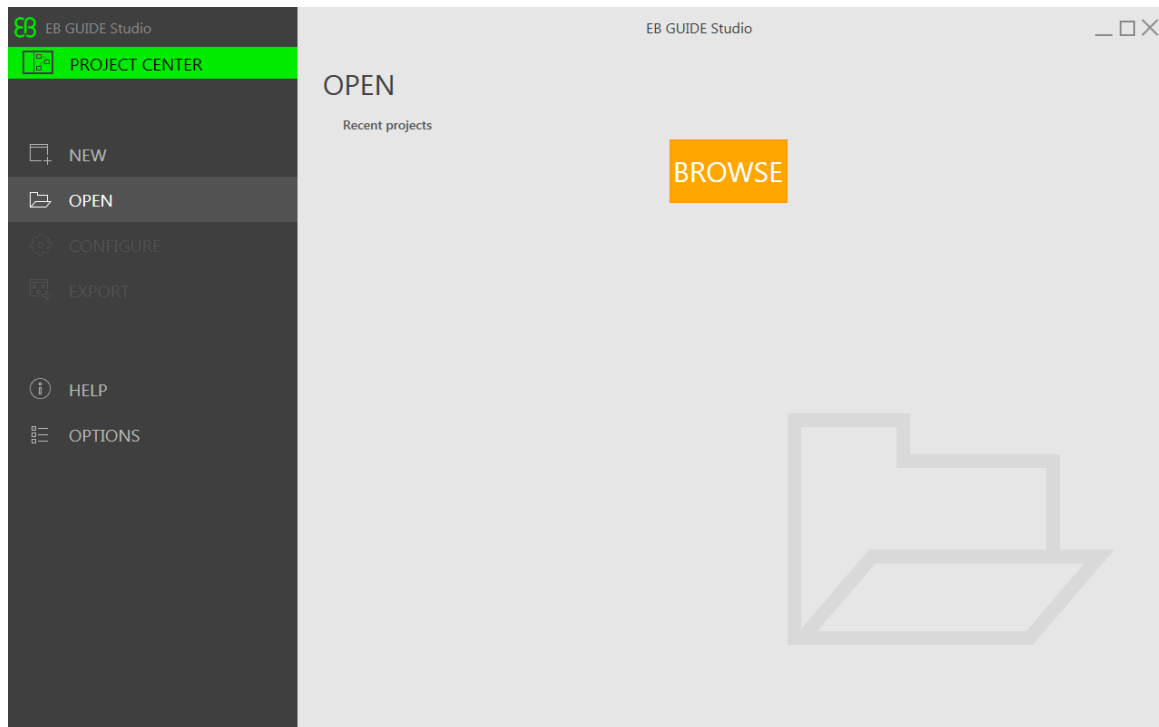


Figure 5.1. Project center

## 5.2. Creating a project



### Creating a project

Prerequisite:

- EB GUIDE Studio is started.
- A directory `C:\temp` is created.

#### Step 1

In the navigation area, click **New**.

#### Step 2

In the content area, select the `C:\temp` directory.

#### 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.

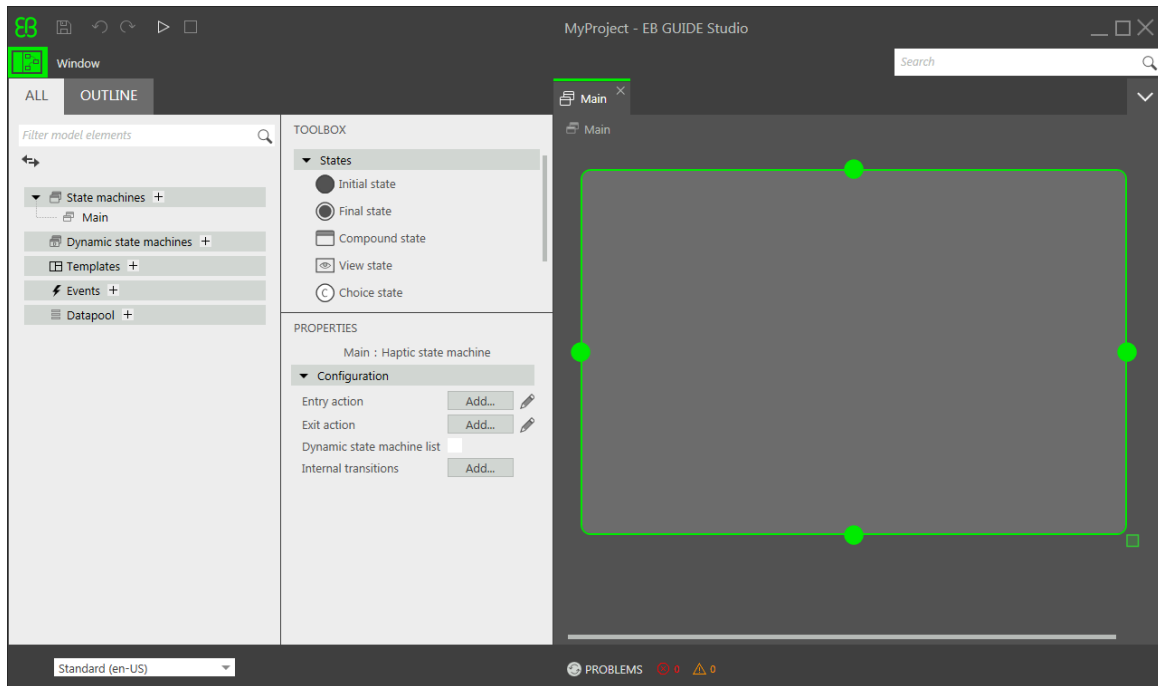


Figure 5.2. Project editor with **Main** state machine

## 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 area as a hierarchical tree view.

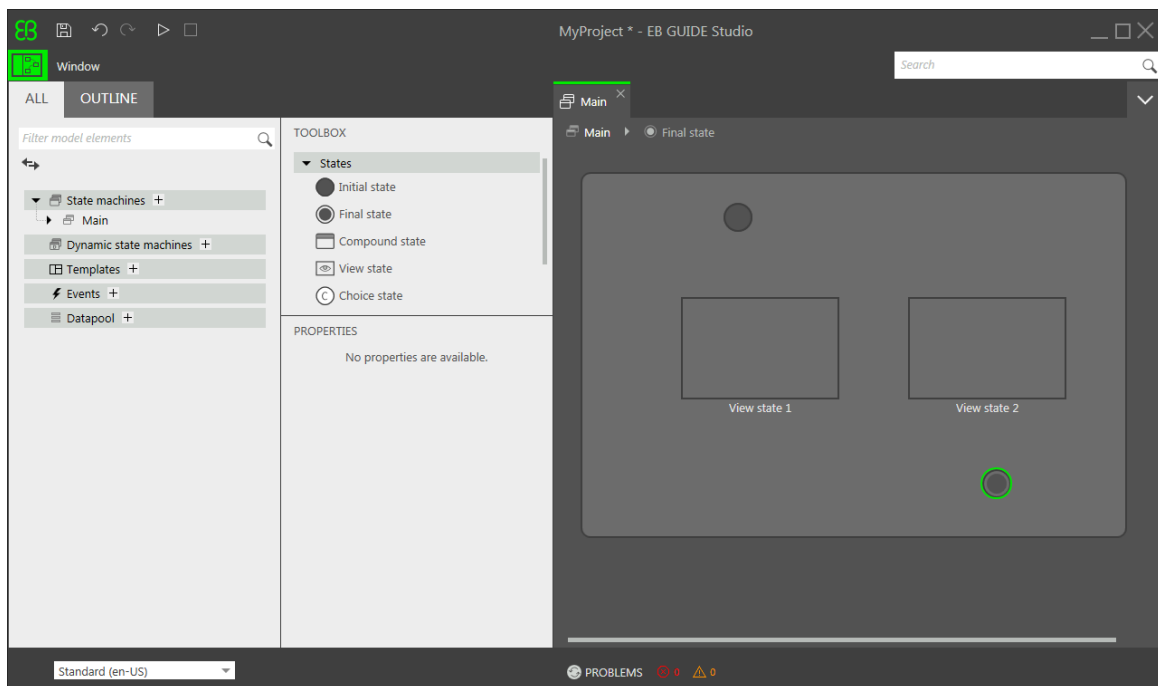


Figure 5.3. Project editor with states



### Adding a transition

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 as a source state for the transition.

#### Step 2

Click the green drag point and keep the mouse button pressed.

#### Step 3

Drag the mouse into the target state, **View state 1**.

#### Step 4

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

A transition is created and displayed as a green arrow.

#### Step 5

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

Select **View state 1** and repeat steps 2 - 4.

#### Step 6

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

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

#### Step 7

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 8

Add a transition between **View state 2** and the final state.

Select **View state 2**, and repeat steps 2 - 4.

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

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

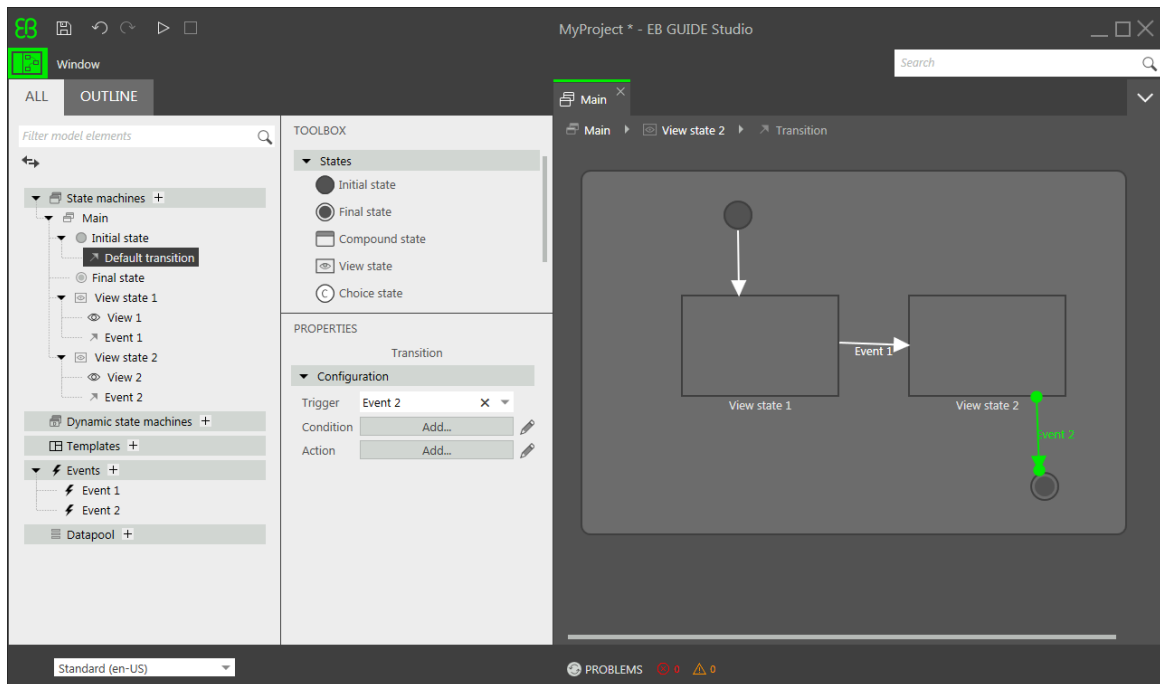


Figure 5.4. States linked by transitions with events

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:

- The navigation area displays the **All** tab.
- **View state 1** is added.

#### 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 state 1**.

### Step 1

Drag a rectangle from the **Toolbox** into the view.

### Step 2

In the **Properties** panel, 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** panel.

### Step 4

In the **Properties** panel, select `Press then react` from the `touchPolicy` drop-down list box.

The rectangle reacts on touch input.

### Step 5

Go to the `touchShortReleased` property, and click **Edit**.

### Step 6

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, **Event 1** is fired after 500 milliseconds.

### Step 7

Click **Accept**.

### Step 8

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

Step 9

In the navigation area, double-click **View 2**.

The content area displays **View 2**.

Step 10

Repeat steps 1-5.

Step 11

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
}
```

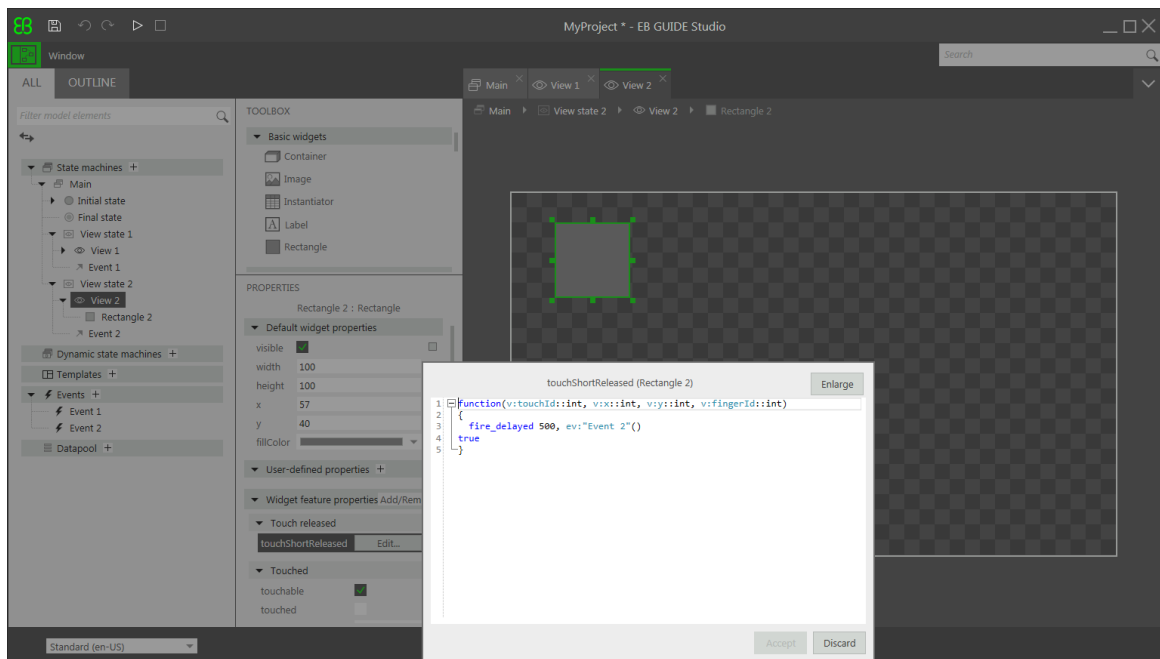


Figure 5.5. Widget property with an EB GUIDE Script

Step 12

Click **Accept**.

If the rectangle is touched, **Event 2** is fired after 500 milliseconds.

Step 13

In the **Properties** panel, 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.

### 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 OpenGL ES 2.0 and DirectX 11 renderers 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
- ▶ At least one light source

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
- ▶ 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

**TIP**



**Setting up the 3D graphic file**

Be aware that 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.

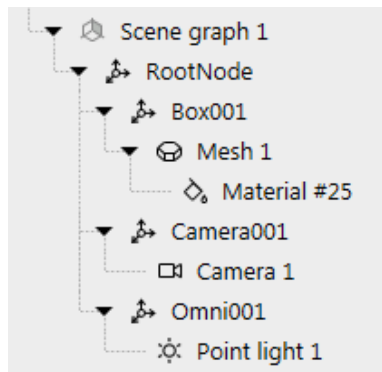


Figure 6.1. Example of a scene graph as displayed in the navigation area

**NOTE**



**Only one material per mesh is allowed**

In EB GUIDE Studio only one material per mesh is allowed. If your 3D graphic has more than one material per mesh, during import EB GUIDE Studio creates additional mesh for each additional material.

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



**Example 6.1.  
Naming of the import directory**

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 subdirectory named `car_-20160102_103029`.

The subdirectory 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 6.17, “Widgets”](#), [section 12.9.4, “3D widgets”](#), and [section 12.10.8, “3D”](#).

For instructions see [section 8.1.5, “Adding a scene graph to a view”](#), and [section 11.7, “Tutorial: Working with a 3D graphic”](#).

## 6.2. Animations

Animations bring motion and visual effects into your EB GUIDE model. In EB GUIDE, you can use animations for different use cases. You can animate widgets within a view and you can animate the transition from one view to another.

### 6.2.1. Animations for widgets

Animating a widget means moving a widget along a view. The movement is defined by curves. Therefore, the **Animations** category in the **Toolbox** includes a widget called animation and a set of curves. For example, there are constant curves, linear interpolation curves, or sinus curves. A curve has a `target` widget property and describes the time-based change of the `target` property.

Each animation has one or more curves associated to it.

Among others, animating a widget can do the following:

- ▶ Move a widget within a view
- ▶ Change the size of a widget
- ▶ Gradually change the color of a widget

An animation is controlled by the EB GUIDE Script functions `f:animation_play`, `f:animation_pause`, `f:animation_cancel`, etc.

**TIP**



**Concurrent animations**

In EB GUIDE, animations are concurrent animations and curves are executed in parallel. This means: If the curves of several animations use the same widget property as a target, the curves overwrite that `target` property's value concurrently.

For animation and curve properties see [section 12.9.3, "Animations"](#).

For instructions see [section 8.5.1, "Animating a widget"](#).

## 6.2.2. Animations for view transitions

To animate a view transition means to define a moving or fading animation for entering or exiting a view. A view change triggers such an animation.

You define view transition animations for view templates. Every time you re-use the view template, the instance inherits the entry and exit animations.

There are various types of view transition animations. An entry animation is, for example, move in from right or move in from bottom. An exit animation is, for example, move out from top to bottom.

For animation properties in view templates see [section 12.9.1, "View"](#).

For instructions see [section 8.5.2, "Animating a view transition"](#).

## 6.3. Application programming interface between application and model

EB GUIDE abstracts all communication data between an application and the 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

## 6.4. 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 context and updated by the reader context.

A communication context is identified by a unique name and numerical ID (0...255) in the project configuration.

A datapool item has one property for the communication context that writes a value, and another property for the communication context that is notified about the changed value and reacts on the value change.

For instructions see [section 9.8, “Establishing external communication”](#)

## 6.5. Components of the graphical user interface

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.

### 6.5.1. 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.

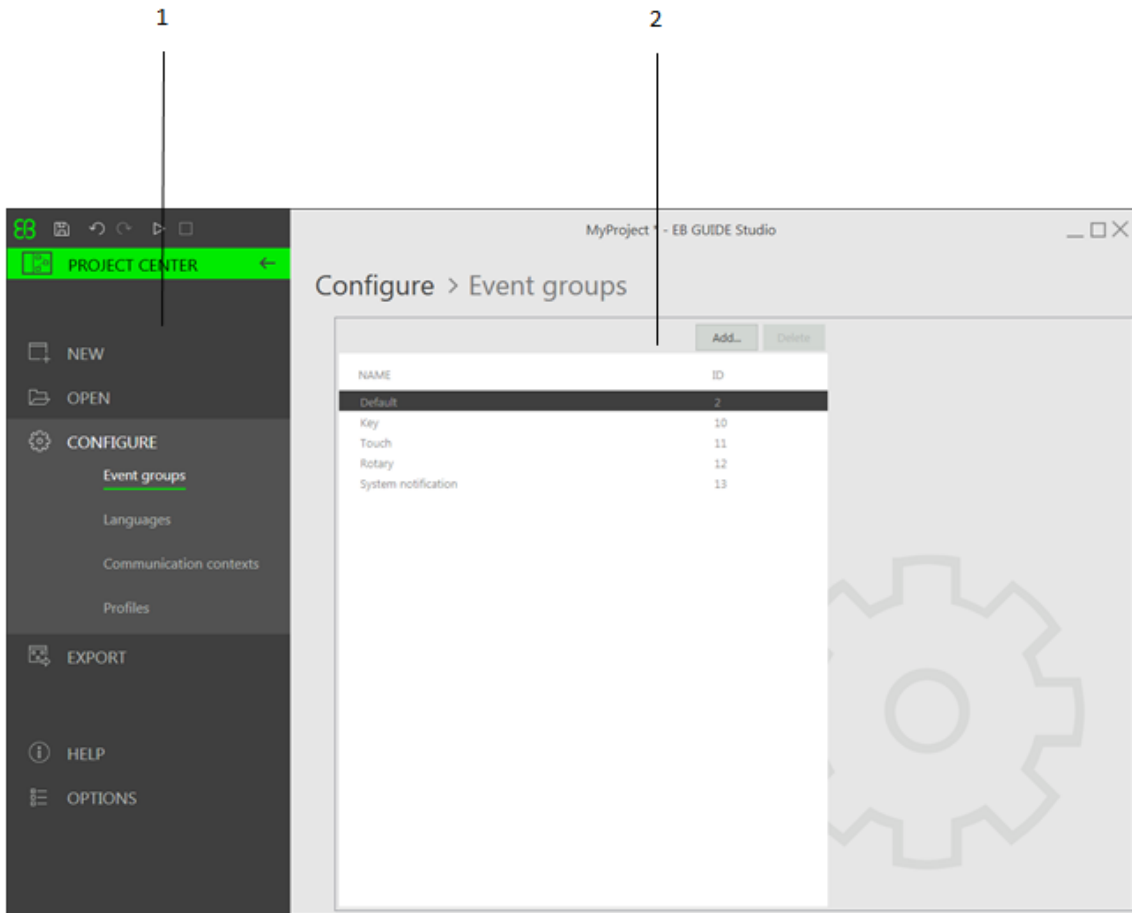


Figure 6.2. Project center with navigation area (1) and content area (2)

### 6.5.1.1. Navigation area

The navigation area of the project center consists of function tabs such as **Configure** or **Export**. You click a tab in the navigation area and the content area displays the corresponding functions and settings.

### 6.5.1.2. Content area

The content area of the project center is where project management and configuration takes place. For example, you select a directory 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.

## 6.5.2. 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 the datapool. The project editor consists of the following areas.

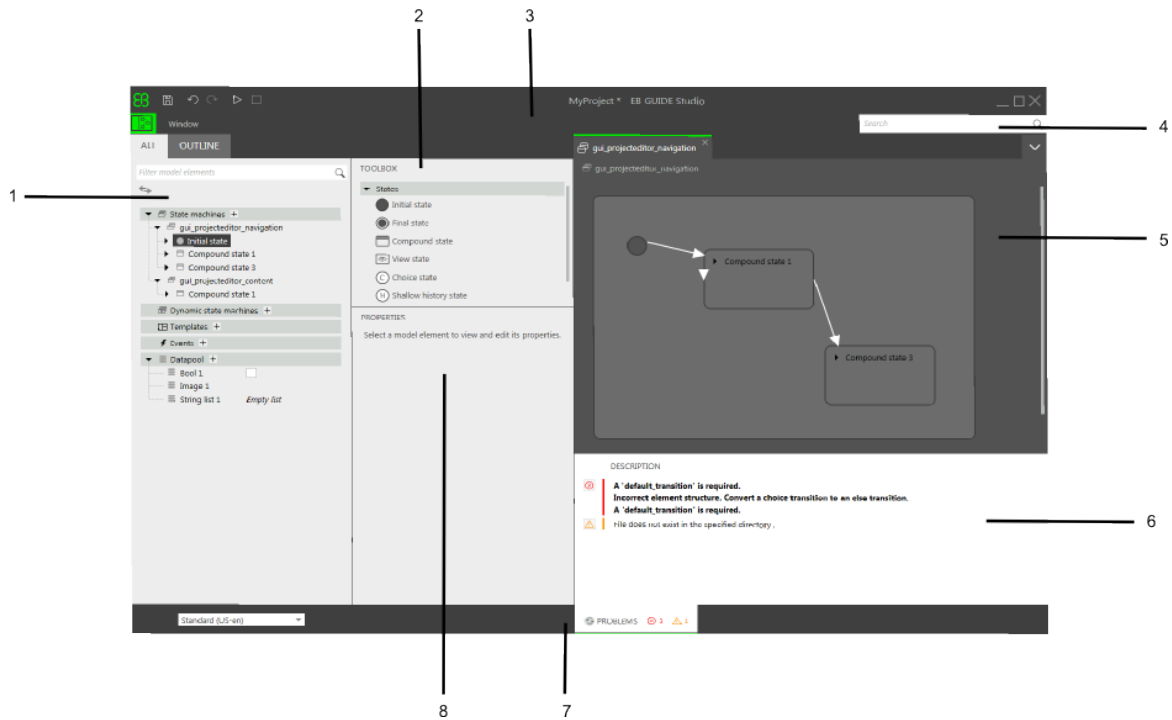


Figure 6.3. Project editor with its areas

- 1 Navigation area
- 2 **Toolbox**
- 3 Command area
- 4 Search box
- 5 Content area
- 6 **Problems** area
- 7 Status bar
- 8 **Properties** panel

### 6.5.2.1. Navigation area



The navigation area displays the model elements of your EB GUIDE model as a hierarchical structure and allows you to navigate to any element. Double-clicking a model element in the navigation area displays the model element in the content area.

The navigation area is divided into two tabs, the **All** tab and the **Outline** tab.

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

The **All** tab is also where you add elements to your EB GUIDE model, for example state machines, views, events and datapool items.

- ▶ The **Outline** tab displays the structure of the selected view tree element and its sub-elements.

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

Clicking a model element in the navigation area and pressing **F3** starts a reference search: It opens the search results window and lists all occurrences of the selected model element in the EB GUIDE model.



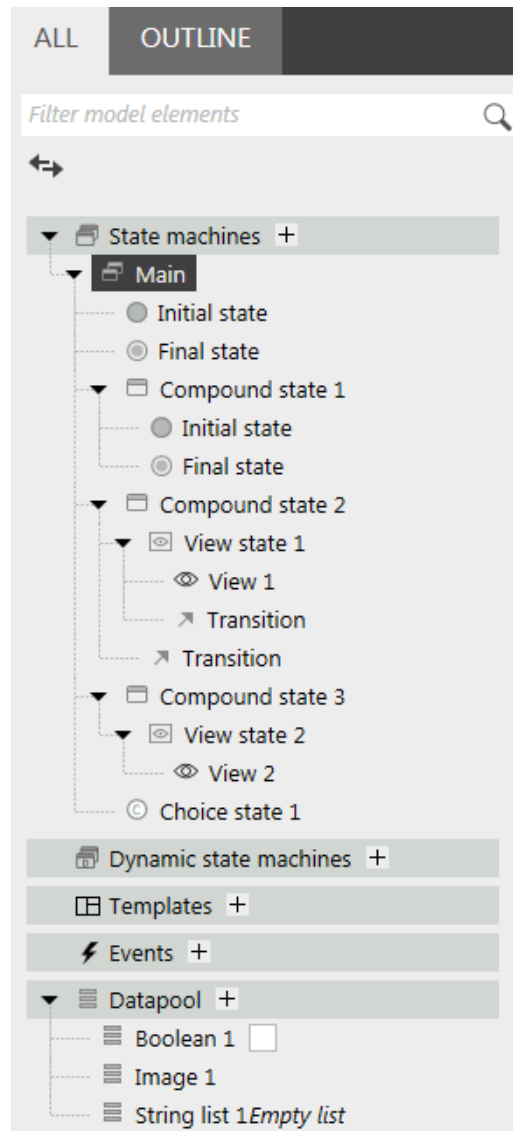


Figure 6.4. Navigation area in project editor

### 6.5.2.2. Content area

What is displayed in the content area depends on the selection in the navigation area. To edit a model element, you double-click the model element in the navigation area 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.

Clicking a state or a widget in the content area and pressing **F3** starts a reference search: It opens the search results window and lists all occurrences of the selected state or widget in the EB GUIDE model.

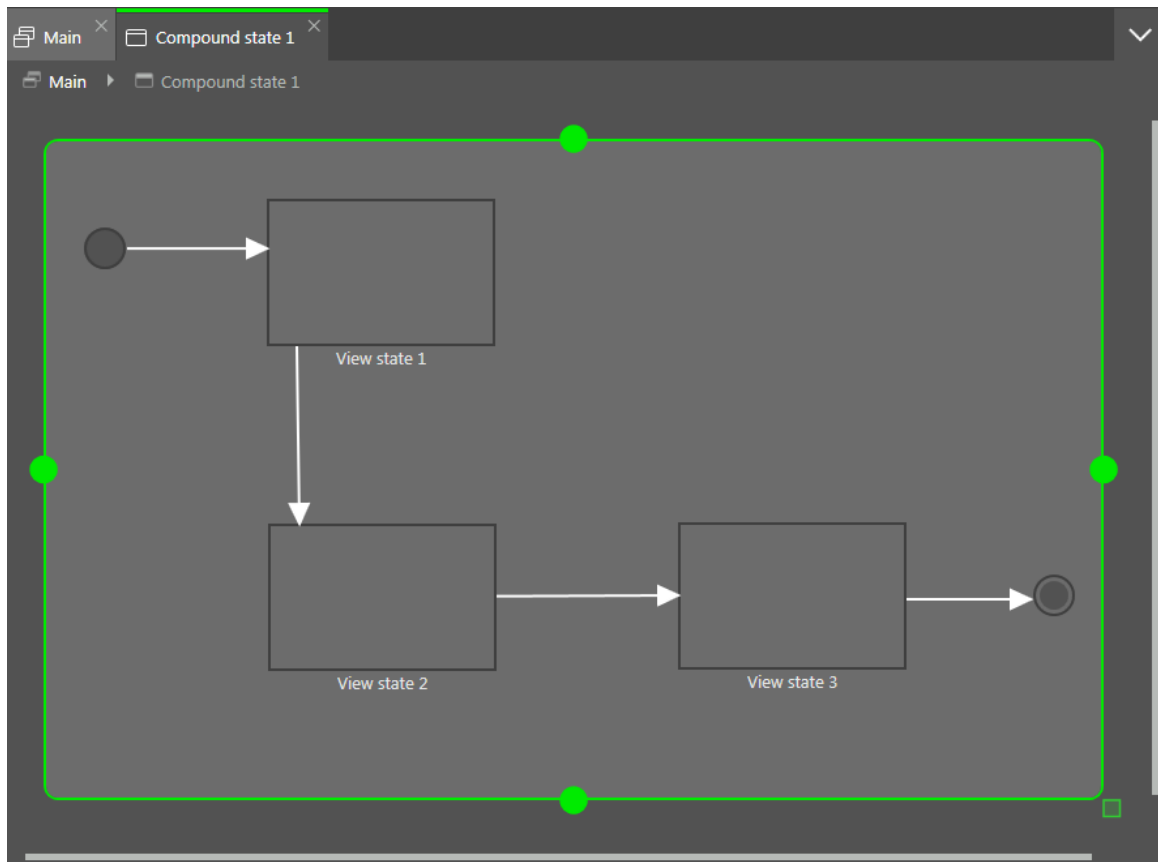



Figure 6.5. Content area in project editor

### 6.5.2.3. Command area

In the command area, you find the  button, which opens the project center, search box or 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.
- ▶ 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.

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 6.1. Categories in search box

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

#### 6.5.2.4. Toolbox

All tools you need for modeling are available in the **Toolbox**. Depending on the element that is displayed in the content area, the **Toolbox** offers a different set of tools. For example, the **Toolbox** can 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 and animations 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.

You drag model elements from the **Toolbox** into the content area.

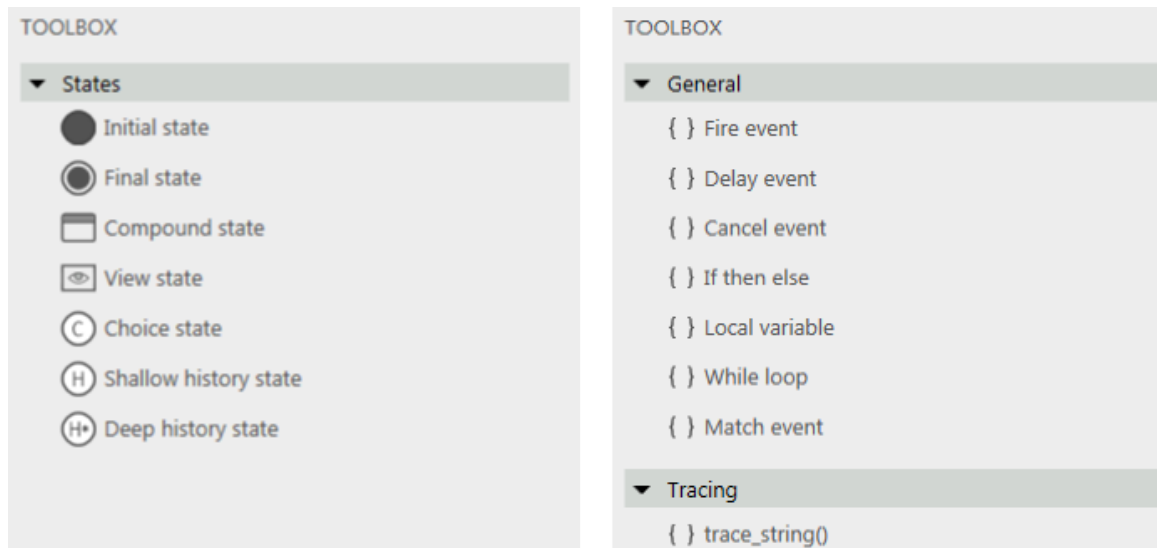


Figure 6.6. Toolbox in project editor

### 6.5.2.5. Properties panel

The **Properties** panel displays the properties of the selected model element, for example of a widget or a state. Properties in the **Properties** panel are grouped by categories. If a model element is selected, you can edit its properties in the **Properties** panel.

Clicking a property in the **Properties** panel and pressing **F3** starts a reference search: It opens the search results window and lists all occurrences of the selected property in the EB GUIDE model.

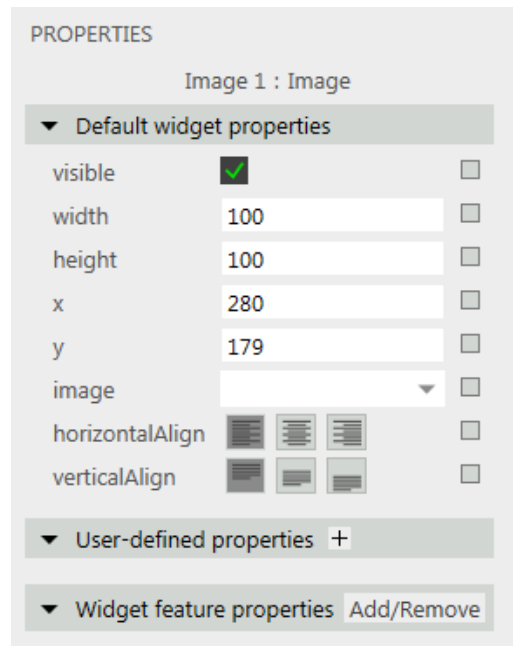


Figure 6.7. **Properties** panel displaying properties of a widget

### 6.5.2.6. Status bar

The status bar displays status information about EB GUIDE Studio.

### 6.5.2.7. Problems area

The **Problems** area displays errors and warnings for the EB GUIDE model.

## 6.6. Datapool

### 6.6.1. 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.

## 6.6.2. Datapool items

Datapool items are used to do the following:

- ▶ Send data from the applications to the HMI
- ▶ Send data from the HMI to the applications
- ▶ Store data which is only used in either HMI or applications

For instructions see [section 9.5. “Adding a datapool item”](#).

To channel communication, you use the communication context.

With the `Writer context` property you define which communication context writes new values.

With the `Reader context` property you define which communication context is notified about changed values and reacts on the value change.

In internal communication, one communication context acts as both reader and writer of a datapool item. Internal communication is used to store data. For example, datapool items with internal communication are used in widget properties.

Using two different communication contexts establishes external communication. External communication is only possible if the `Read-only` property of a datapool item is cleared.

For instructions see [section 9.8. “Establishing external communication”](#).

## 6.6.3. Windowed lists

Using the datapool item property **Windowed**, 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 directory. Those lists are typically provided by one communication context, for example media application, and are only partially displayed by another communication context, for example HMI.

---

### NOTE



A datapool item with the enabled property **Windowed** needs a writer context and reader contexts which differ.

---

The writer communication context defines a virtual list length and a number of windows, which possibly contain only parts of the list. The reader communication context reads data only from locations that are covered by

windows. Reading from other locations fails. In such a use case, the reader communication context has to inform the writer communication context 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. Because of the asynchronous communication based on events and datapool updates, 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.

## 6.7. 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 modeling. It includes project-specific options, extensions, resources, and, for graphical projects, the description of a haptic dialog.

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. Event handling

### 6.8.1. Event system

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

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.

### 6.8.2. Events

An event in EB GUIDE has a unique event ID and belongs to an event group. The event ID is used by EB GUIDE TF to send and receive the event.

Event group IDs between 0 and 65535 are reserved for internal use within the EB GUIDE product line. Exceptions to that are the event groups that are listed in the following table.

Table 6.2. Allowed event groups and IDs

Event group	ID
Default	2
Key input events	10
Touch input events	11
Rotary input events	12
System notification events	13

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

For instructions see:

- ▶ [section 9.1, “Adding an event”](#)
- ▶ [section 9.3, “Addressing an event”](#)

## 6.9. Extensions



## 6.9.1. EB GUIDE Studio extension

A piece of software that extends EB GUIDE Studio and that is valid for all EB GUIDE models is called an EB GUIDE Studio extension. The EB GUIDE Studio extension does not concern EB GUIDE GTF.

Typical EB GUIDE Studio extensions are:

- ▶ Additional toolbar buttons
- ▶ Additional data exporters

## 6.9.2. EB GUIDE GTF extension

An EB GUIDE GTF extension is a piece of software that extends 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 the following directory:

```
$GUIDE_PROJECT_PATH/<project name>/resources/target
```

# 6.10. Languages

## 6.10.1. 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 10.5, “Changing the display language of EB GUIDE Studio”](#).

## 6.10.2. 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. You add a language for an EB GUIDE model in the project configuration.

For instructions see [section 8.4.1, “Adding a language”](#).

It is possible to make datapool items language dependent. A datapool item defines a value for each language. To support languages select the **Language support** property.



#### **Example 6.4.** **Language dependent texts**

In the project configuration three languages are added: English, German, and French. A datapool item has the value *Welcome* in English and the values *Willkommen* in German and *Bienvenue* in French.

For instructions see [section 11.6, “Tutorial: Adding a language dependent text to a datapool item”](#).

The current language of the exported EB GUIDE model can be set during run-time.

### 6.10.3. Export and import of language dependent texts

Use the export and import functionality in EB GUIDE Studio to export, edit, and import all language dependent texts. You export texts to an `.xliff` file and forward the file to a translation agency. `.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.

After translation you import the translated `.xliff` file in the corresponding language in EB GUIDE Studio.

For instructions see [section 10.7, “Exporting and importing language dependent texts”](#).

## 6.11. 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 Studio project in the resources directory.

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

There are three types of resource in EB GUIDE:

1. Fonts
2. Images
3. Meshes for 3D graphics

In order to use resources in the project, add the resource files to the directory.

## 6.11.1. Fonts

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

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

For instructions see [section 8.1.6, “Changing the font of a label”](#).

## 6.11.2. Images

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

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

For instructions see [section 8.1.4, “Adding an image to a view”](#).

### 6.11.2.1. 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 will not be scaled. Marked areas will be scaled. Markers are not displayed in EB GUIDE Studio.

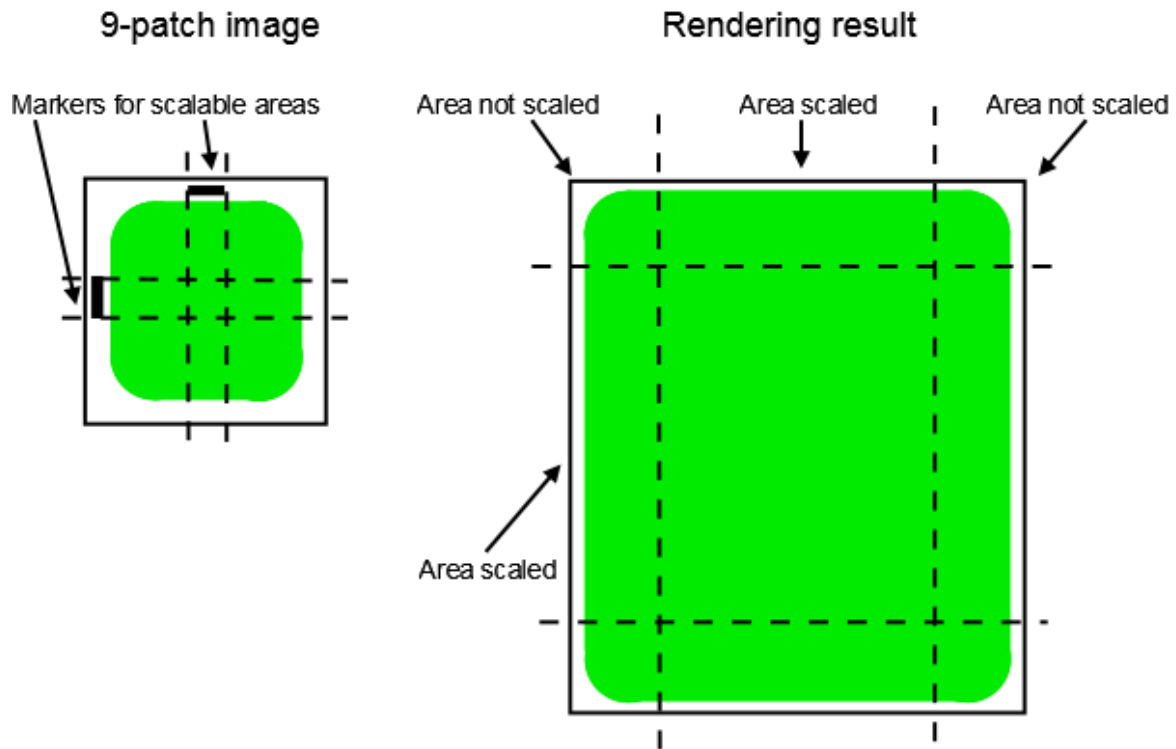


Figure 6.8. 9-patch example

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

- ▶ 9-patch processing works with the OpenGL ES 2.0 and the DirectX renderer only.
- ▶ 9-patch processing works with `.png` images only. `.ppm` images do not support 9-patch processing.
- ▶ For 9-patch images the `*.9.png` extension is mandatory.
- ▶ 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.

### 6.11.3. Meshes for 3D graphics

It is possible to import 3D graphic files in EB GUIDE Studio. After importing a 3D graphic file in EB GUIDE Studio, in `$GUIDE_PROJECT_PATH/<project name>/resources` you find a subdirectory. 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 8.1.5, "Adding a scene graph to a view"](#).

## 6.12. 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.

## 6.12.1. 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

## 6.12.2. Namespaces 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*. EB GUIDE Script namespaces make this possible. Every identifier, i.e. name of an object, in EB GUIDE Script must be prefixed with a namespace and a colon.

The set of namespaces is fixed in EB GUIDE Script, you cannot introduce new namespaces. The following namespaces 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 namespace 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 6.5. 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!";
```

## 6.12.3. 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 6.6. 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 area 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.

## 6.12.4. 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 integer type)
- ▶ 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.

## 6.12.5. 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 6.7. Evaluation of an integer value

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

## 6.12.6. 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 6.8. 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
```

## 6.12.7. 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 (==) and (!=).

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 6.9.



### 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
```

## 6.12.8. 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.

## 6.12.9. 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 namespace `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 6.10.

### 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;
    }
}
```

## 6.12.10. 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 6.11.**  
**Usage of the `while` loop**

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

## 6.12.11. 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.

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 6.12.**  
**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 {}
```

## 6.12.12. 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 6.13. 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);
```

## 6.12.13. Datapool access

Scripts written in EB GUIDE Script can read and write datapool items. An identifier prefixed by a namespace `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.

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 6.14. 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-reference operator (=>) assigns a different datapool item to a datapool reference. It is only applied to datapool references.

## 6.12.14. Widget properties

If a script is part of a widget, it can access the local properties of that widget. EB GUIDE Script creates a local 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 local widget property, for example as an input reaction such as click or button press.



### Example 6.15. 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 (`->`) 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 `^` 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 local variables for the efficient manipulation of multiple properties.



### Example 6.16. 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;  
}
```

## 6.12.15. 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 6.17. 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.

## 6.12.16. 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 6.18.**  
**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 6.19.**  
**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 6.20.**



### 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 {}
```

If a 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 6.21. 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;
}
```

## 6.12.17. 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 6.22. 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);
```

## 6.12.18. 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 12.4.3, “EB GUIDE Script standard library”](#).

## 6.13. 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 9.7, “Converting a property to a scripted value”](#).

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

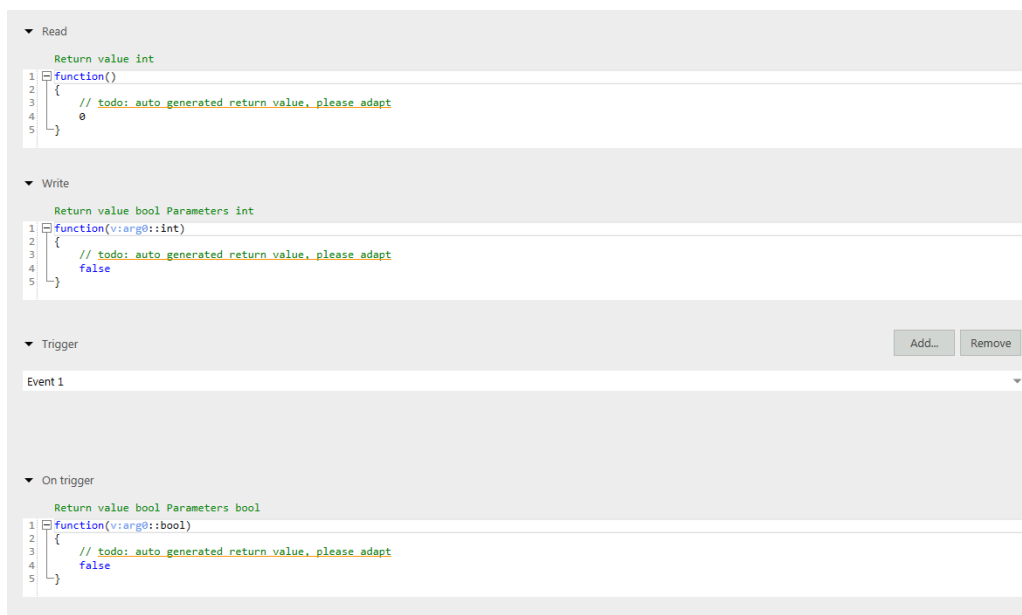


Figure 6.9. EB GUIDE Script editor in EB GUIDE Studio

- ▶ The **Read** script is called when the scripted value property is read. If the property is of a list type, 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 a list type, 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 **Trigger** list contains a list of events, datapool items and widget properties that trigger the execution of the **On 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 **Trigger** 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 a list type.

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

## 6.14. Shortcuts, buttons and icons

### 6.14.1. Shortcuts

The following table lists shortcuts available in EB GUIDE Studio and explains their meaning.

Table 6.3. Shortcuts

Shortcut	Description
Ctrl+C	Copy the selection
Ctrl+F	Jump into search box
Ctrl+S	Save

Shortcut	Description
<b>Ctrl+V</b>	Paste the copied selection
<b>Ctrl+Y</b>	Redo
<b>Ctrl+Z</b>	Undo
<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 model element in the navigation area
<b>F3</b>	Find all occurrences of the selected element in the EB GUIDE model
<b>F5</b>	Start simulation
<b>F6</b>	Validate
<b>Del</b>	Delete the selected model element from the content area or the navigation area
-	Collapse the selected model element in the navigation area
* and +	Expand the selected model element in the navigation area
<b>Up/Down/Left/Right</b>	Move the selected state or widget in the content area one pixel up, down, left, or right

The following table lists command line options available in EB GUIDE Studio and explains their meaning.



Table 6.4. Command line options














Option	Description
-h	Shows the help message
-e <project file, destination dir, profile>	Exports an EB GUIDE model that is saved as <code>project file</code> to the destination directory <code>destination dir</code> using a selected <code>profile</code>
-m	Allows the migration of the project during export

## 6.14.2. Buttons

The following table lists buttons that are used in EB GUIDE Studio and explains their meaning.

Table 6.5. Buttons in EB GUIDE Studio








Button	Description
	Undo
	Redo




Button	Description
	Save
	Validate the project
	Start the simulation
	Stop the simulation
	Open the project center
	Open an additional editor
	Synchronize content area and navigation area
	Add an event, a datapool item, or a state machine
	Open a property-related context menu.  Depending on the button's color it indicates the following: <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> <li> Property value is equal to template value.</li> </ul>

### 6.14.3. Icons

The following table lists icons that are used in EB GUIDE Studio and explains their meaning.

Table 6.6. Icons in EB GUIDE Studio

Icon	Description
	Indicates an exit animation of a view template
	Indicates an entry animation of a view template
	Indicates an entry action of a state machine or state
	Indicates an exit action of a state machine or state
	Opens a context menu to delete an entry or exit action
	Indicates that a dynamic state machine list is enabled
	Indicates a template

Icon	Description
	Indicates a transition
	Indicates an internal transition
	Widget template: Indicates that a property is added to the widget template interface

## 6.15. State machines and states

### 6.15.1. State machines

A state machine is a deterministic finite automaton and describes the dynamic behavior of the system. In EB GUIDE, a state machine consists of an arbitrary number of hierarchically ordered states and of transitions between the states.

In EB GUIDE you can create the following types of state machines:

#### 6.15.1.1. Haptic state machine

Haptic state machine allows the specification of GUI.

#### 6.15.1.2. Logic state machine

Logic state machine allows the specification of some logic without GUI.

#### 6.15.1.3. Dynamic state machine

Dynamic state machine runs parallel to other state machines.

Dynamic state machine does not start automatically at system start. The start and stop of dynamic state machines is initiated by another state machine.

There are two kinds of dynamic state machines:

- ▶ Haptic dynamic state machine
- ▶ Logic dynamic state machine

For instructions see [section 11.1, “Tutorial: Adding a dynamic state machine”](#).

## 6.15.2. States

EB GUIDE uses a concept of states. States determine the status and behavior of a state machine. States are linked by transitions. Transitions are the connection between states and define a state change from a source state to a destination state.

A state has the following properties:

- ▶ Entry action
- ▶ Exit action
- ▶ Internal transitions

### 6.15.2.1. 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.

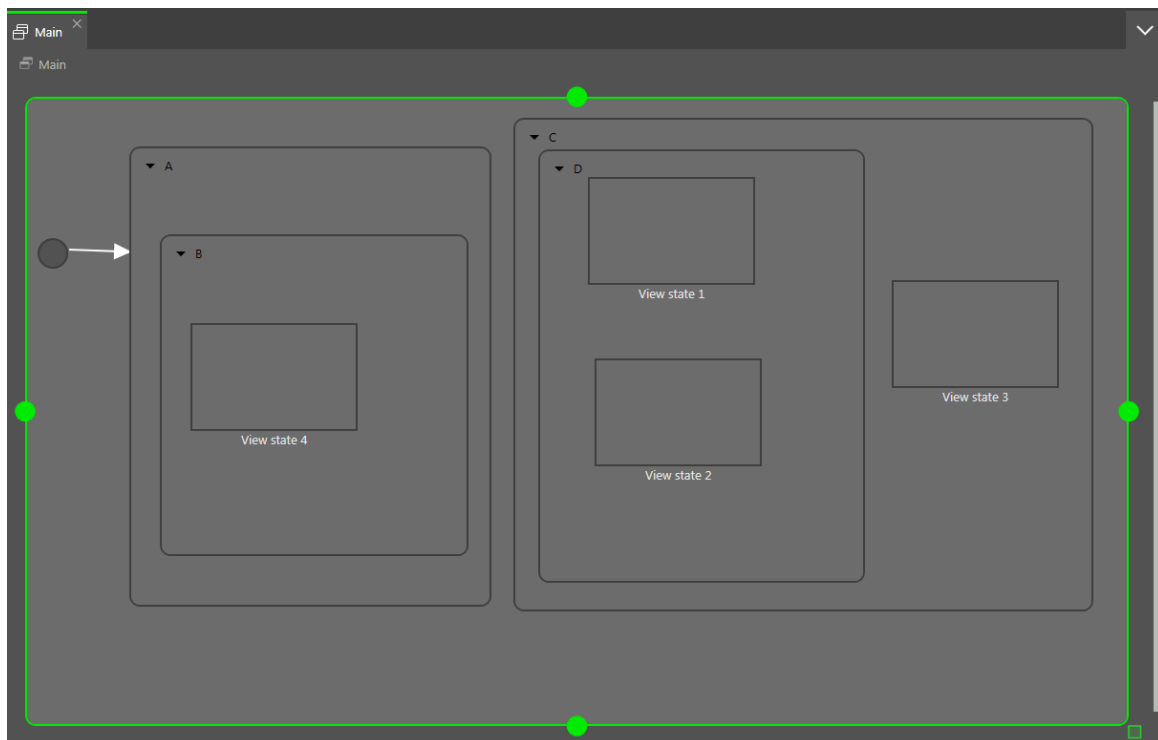


Figure 6.10. Compound states

In the navigation area, the state hierarchy is shown as a tree structure.

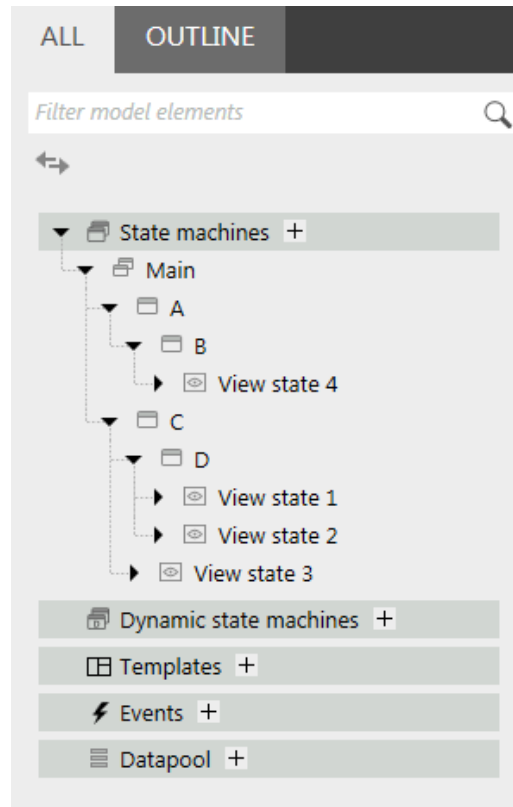


Figure 6.11. State hierarchy as a tree

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.

### 6.15.2.2. View state

A view state contains a view. 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.

### 6.15.2.3. Initial state

An 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.

Initial state can be used as starting point of a compound state or to enter a compound state in the following ways:



- ▶ With a transition to compound state, initial state is mandatory
- ▶ With a transition to a child state of a compound state

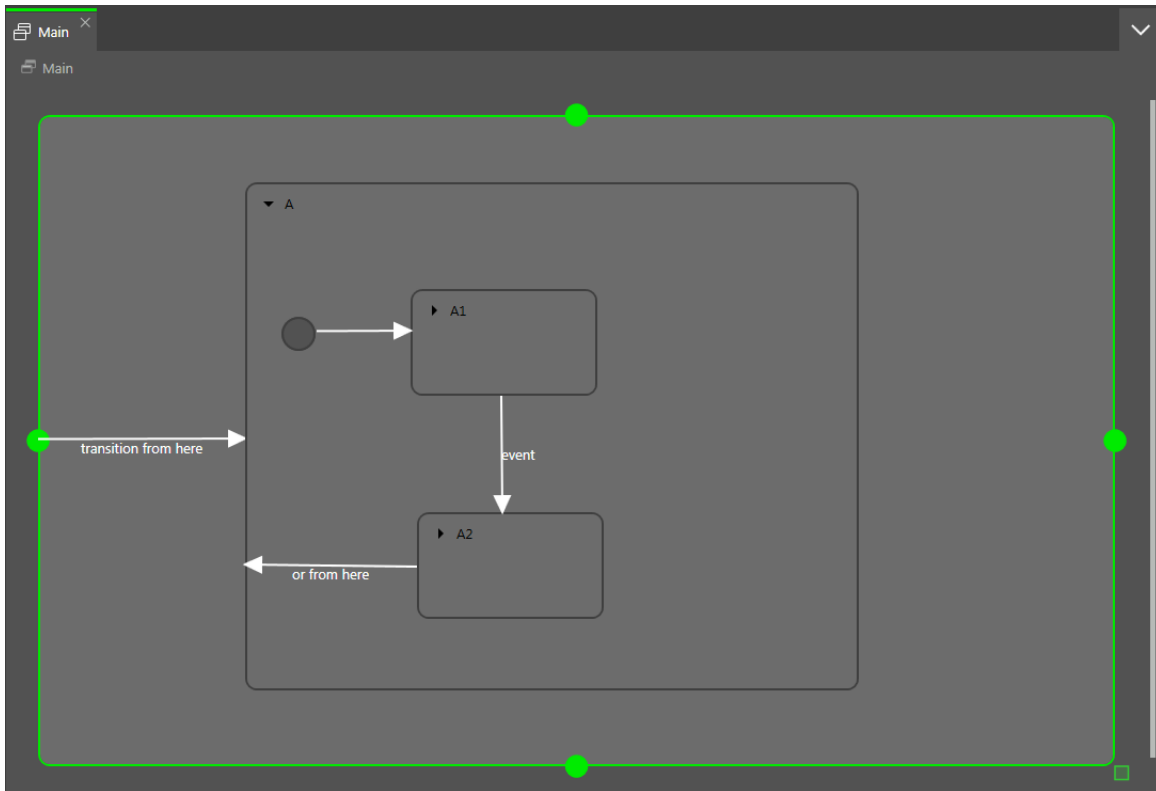


Figure 6.12. An example of an initial state

#### 6.15.2.4. Final state

A final state is used to exit a compound state. 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.

A compound state can have only one final state. The 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)

If a compound state contains a final state, the compound state must have an outgoing transition.

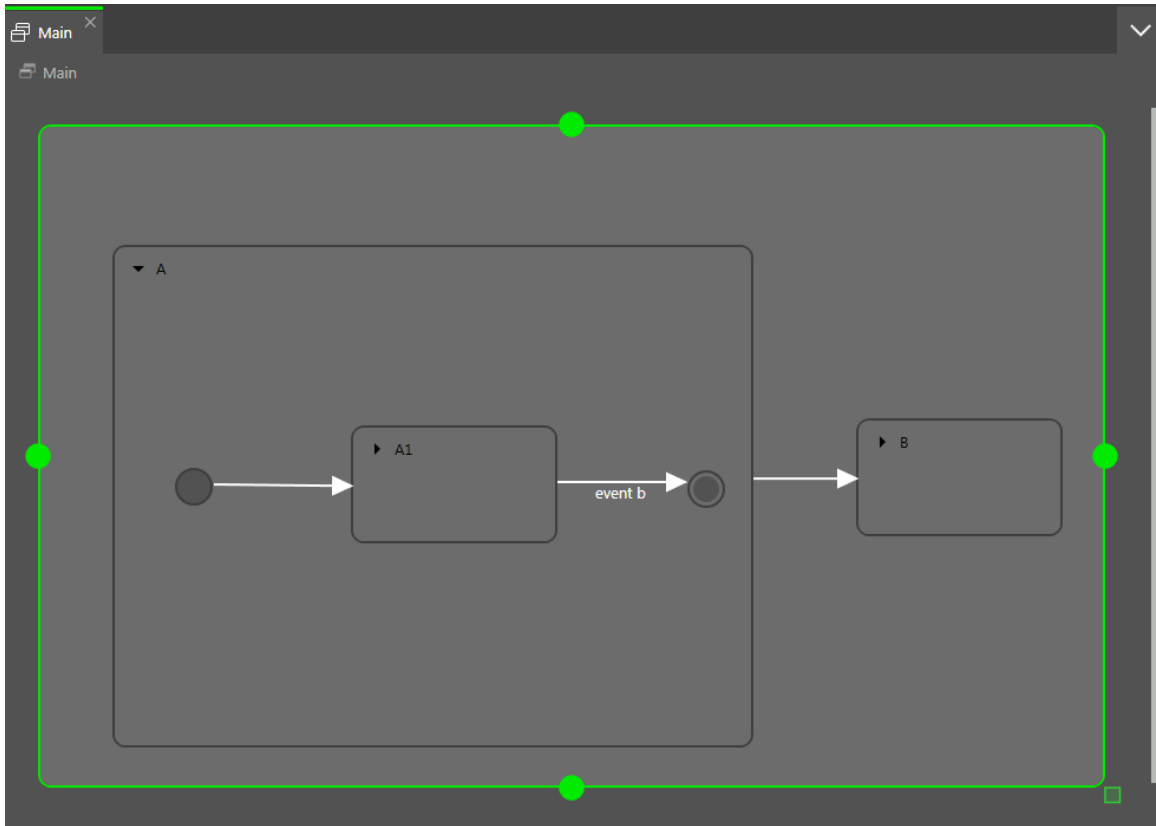


Figure 6.13. Final state usage in a compound state

### 6.15.2.5. Choice state

A choice state realizes a dynamic conditional branch. It is used when firing an event depends on conditions. A choice state is the connection between a source state and a destination 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`. The `else` transition is mandatory.

It is possible that several of the outgoing transitions are true, thus it is necessary to define the order in which the outgoing transitions are evaluated.

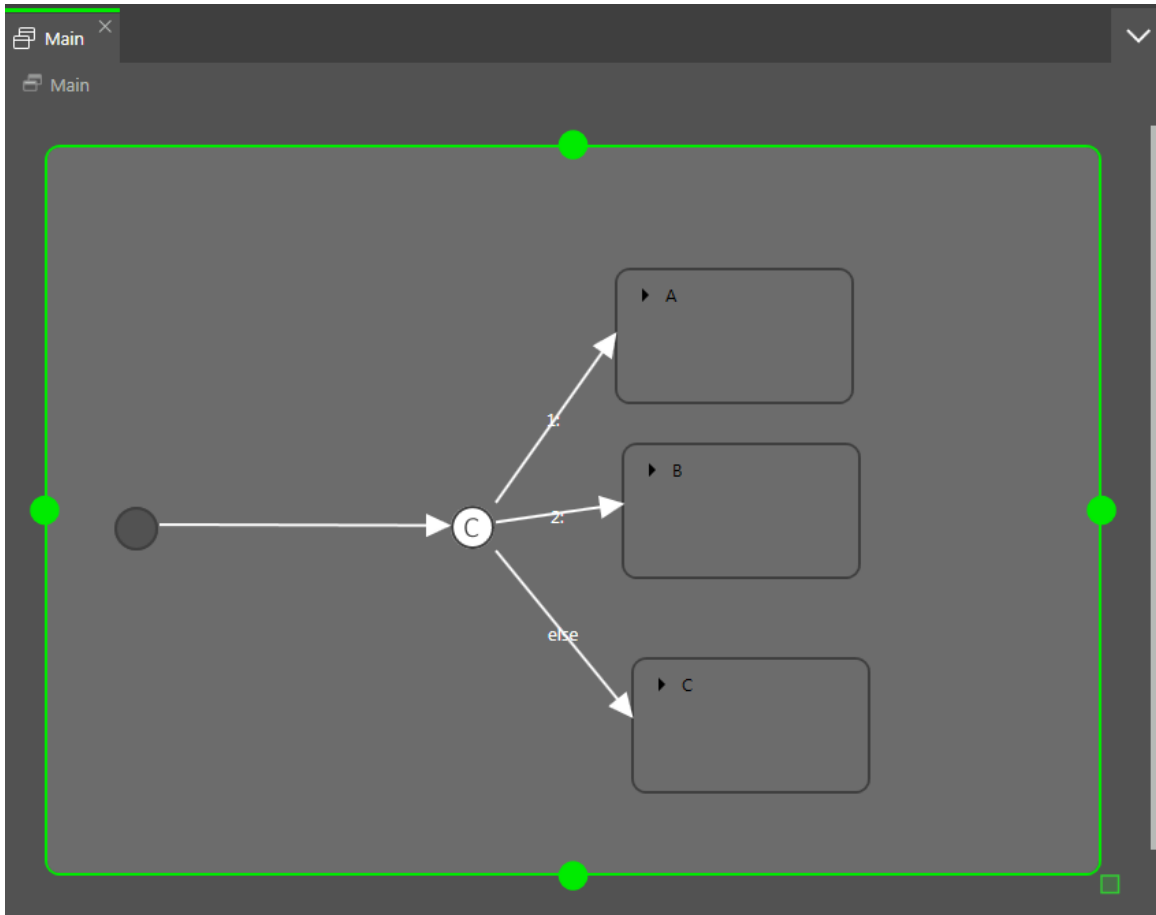


Figure 6.14. Choice state with incoming and outgoing transitions

### 6.15.2.6. History states

EB GUIDE supports two types of history states:

- ▶ Shallow history state stores the most recent active sub-state: the sub-state that was active just before exiting the compound state.
- ▶ Deep history state stores a compound state and its complete sub-hierarchy just before the compound state is exited.

When the parent state of a history state is entered for the first time, the last active child state is restored.

A shallow history state only remembers the last state that was active before compound state was exited. It cannot remember hierarchies.

A shallow history state restores the last active state recorded within a compound state. It has an outgoing default transition without conditions but can have multiple incoming transitions.

When a compound state is entered for the first time the shallow history state is empty. When an empty shallow history state is entered the shallow history state default transition determines the next state.



### Example 6.23. Shallow history state

A shallow history state can be used as follows.

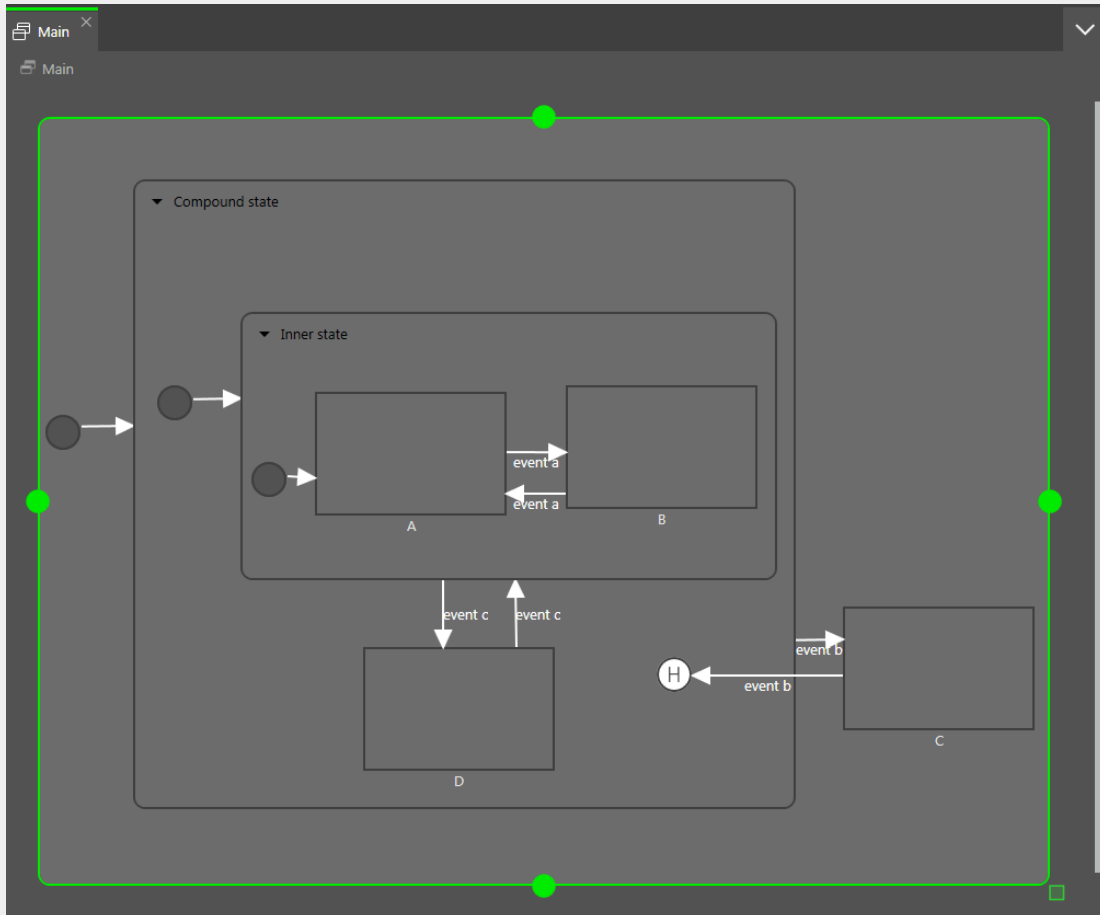


Figure 6.15. Shallow history state

- ▶ 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 Compound 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 state last active but cannot remember hierarchies.
4. Entering `Inner state` leads to state A.

A deep history state is able to save hierarchical histories.



### Example 6.24. Deep history state

A deep history state can be used as follows.

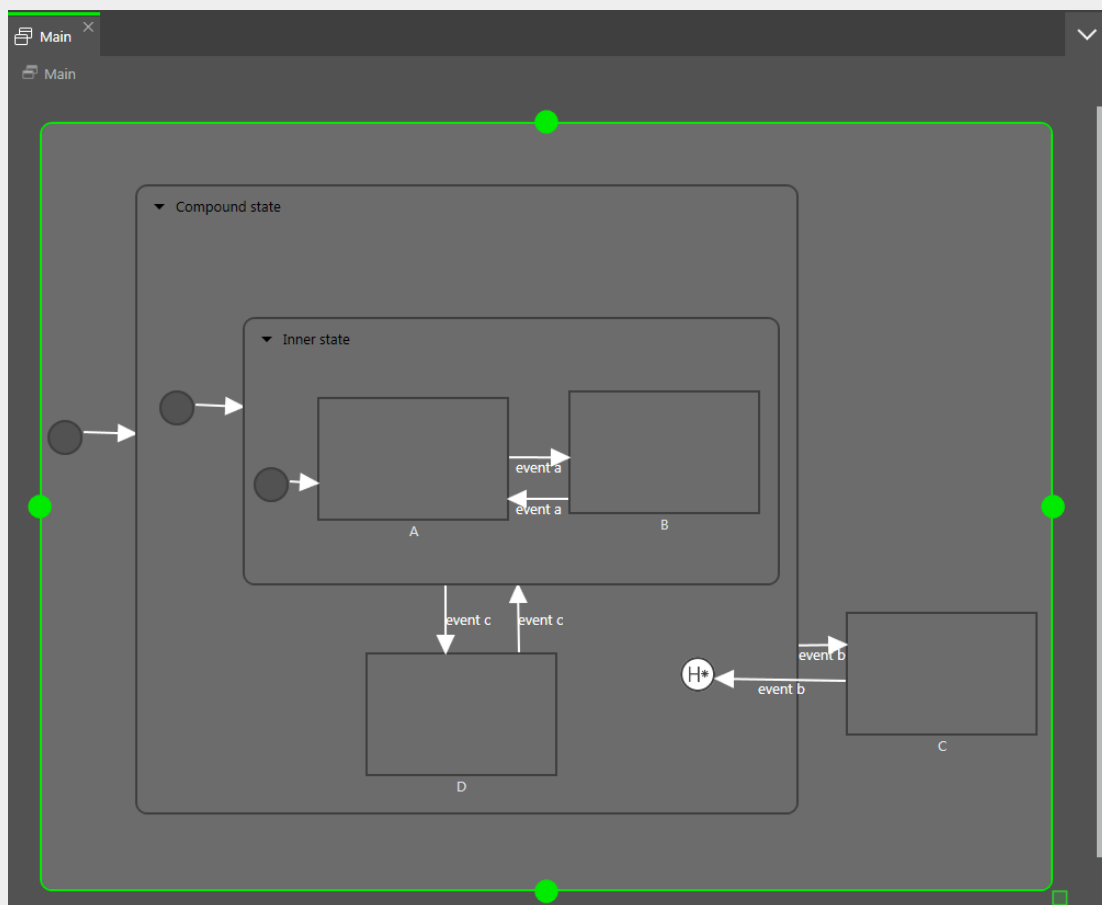


Figure 6.16. Deep history state

- ▶ **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 `Compound 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.

One state can have either a shallow history state or deep history state. You can have a history state in a parent state and another history state in a child state.

### 6.15.3. 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. 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

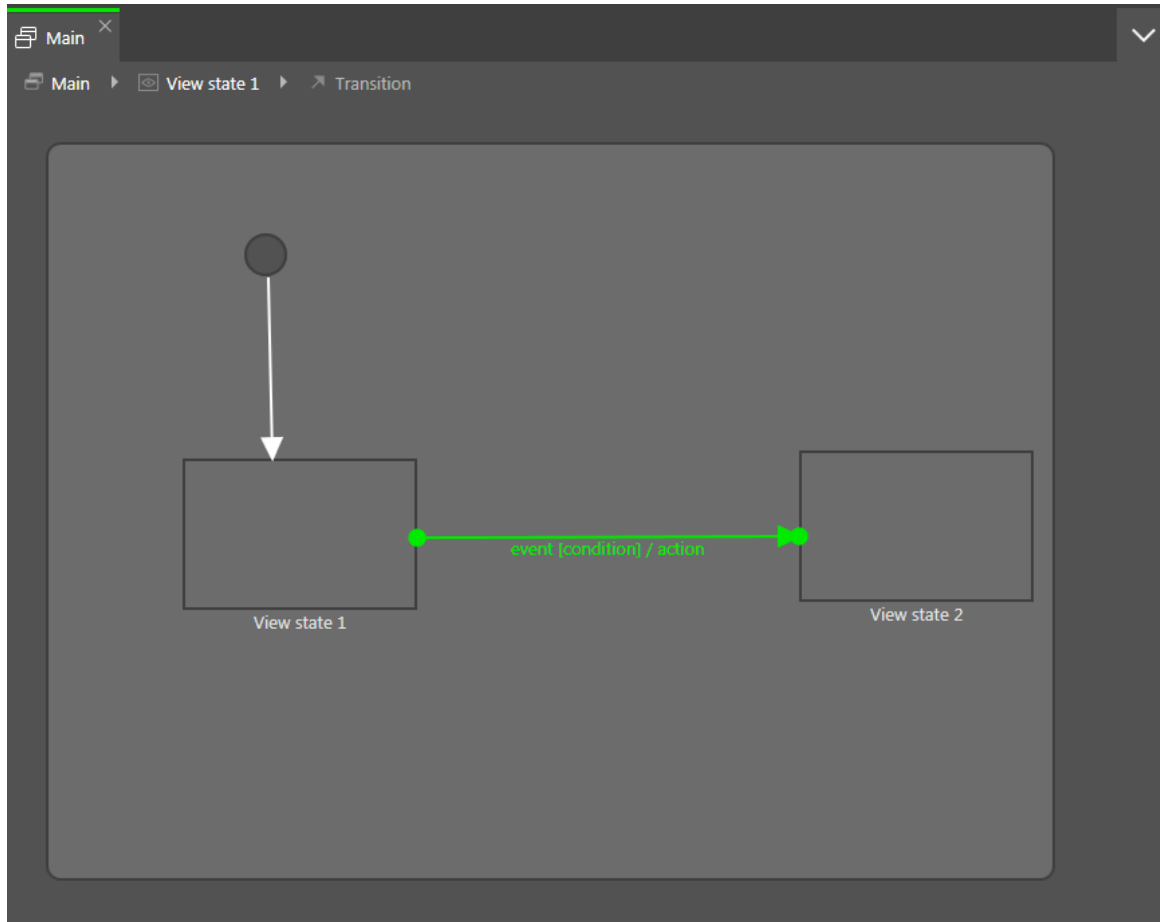


Figure 6.17. A transition

**NOTE**



**Transitions are deterministic**

It is not possible to have more than one transition from a particular source state for the same event even with different conditions. If the state machine is supposed to jump to different destination states depending on different conditions, use a choice state.

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 and thus reduce the number of conditions.



**Example 6.25.**

### Transition inheritance

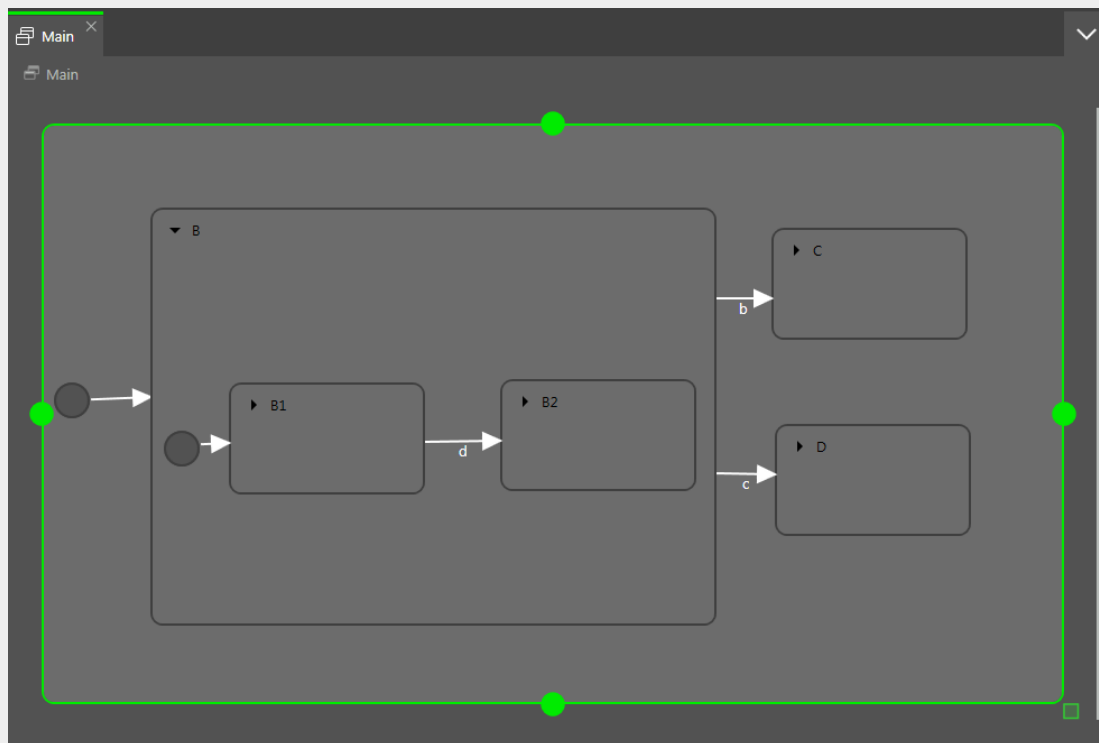


Figure 6.18. Transition inheritance

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 external transition from the parent state, transition inheritance is overridden.



#### Example 6.26.



### Transition override

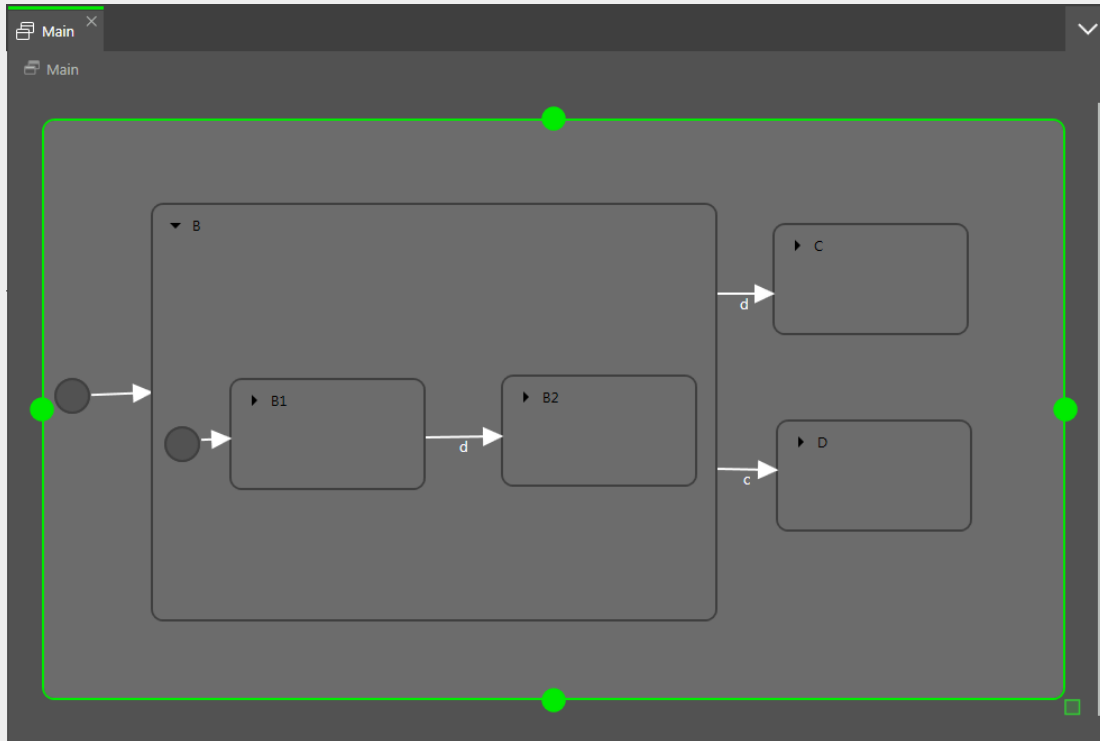


Figure 6.19. Transition override

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.

#### NOTE

#### Execution hierarchy



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 compared to external transitions.

There are different types of transitions.

#### ▶ 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

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.

▶ 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`.

▶ Internal transition

An internal transition is a transition that has no destination 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

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

## 6.15.4. 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.

The state machine cycle is as follows:

1. The state machine is started by entering its initial state.
2. The state machine waits for incoming events.
  - a. Internal transitions are found.
    - i. Start at the current state and search for the first internal transition that is triggered by the current event and has condition `true`. If such a transition is found, it is executed.
    - ii. If no transition is found, go to the parent state and search for the first internal transition that is triggered by the current event and has condition `true`.
    - iii. If no transition is found, repeat the previous step until the top-level state is reached.
  - b. Internal transitions are processed.

Executing an internal transition only triggers the action that is connected to the internal transition. The state is not exited and re-entered.

- c. Transitions are found.
  - i. Start at the current state and search for a transition that is triggered by the current event and has condition `true`. If such a transition is found, it is executed.
  - ii. If no transition is found, go up to the parent state and search for a transition.
  - iii. Repeat the previous step until the first fitting transition is found.
- d. Transitions are processed.

Executing a transition changes the state machine from one state to another state. The source state is exited and the destination state is entered.

A transition is only executed when its corresponding event is fired and the condition is evaluated to `true`.

A transition can exit and enter several compound states in the state hierarchy. Between the exit cascade and the entry cascade the transition's action is executed.

Entering a state can require a subsequent transition, for example entering a compound state requires executing the transition of an initial state as a subsequent transition. A chain of several subsequent transitions is possible.

- 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.



**Example 6.27.**

### Executing a transition

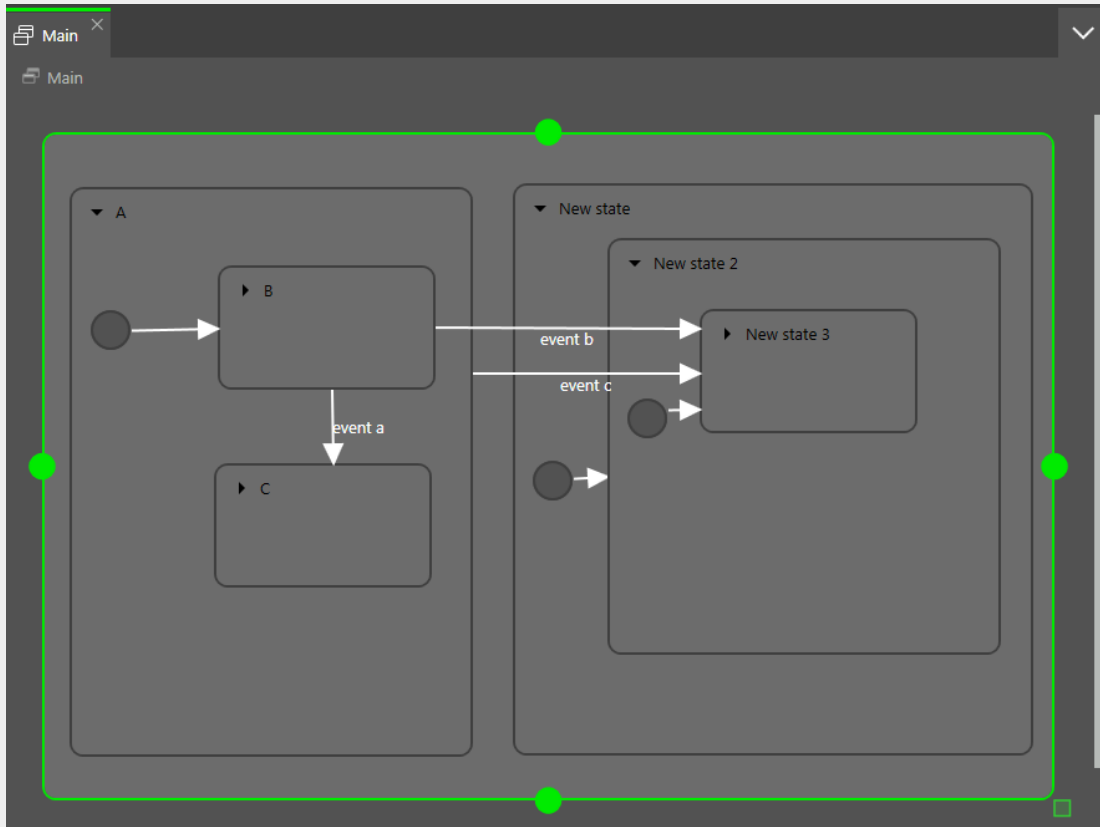


Figure 6.20. Executing a transition

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 6.28.**  
**Executing a transition**

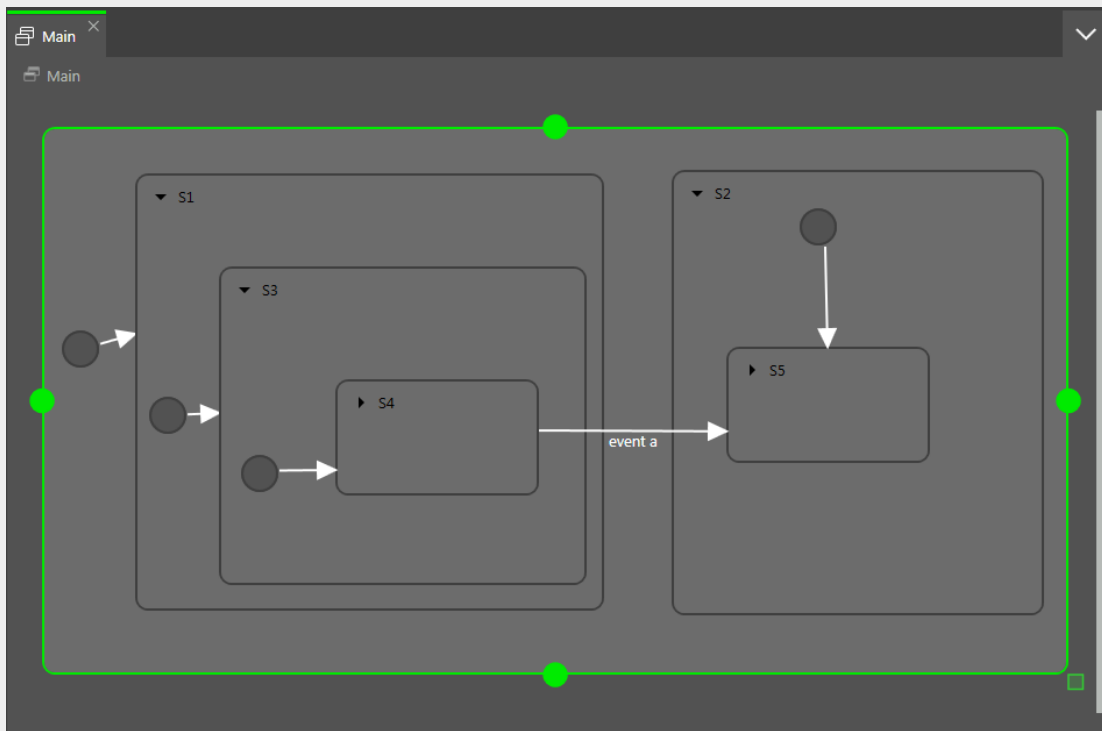


Figure 6.21. Executing a transition

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 6.29.**

### Executing a transition

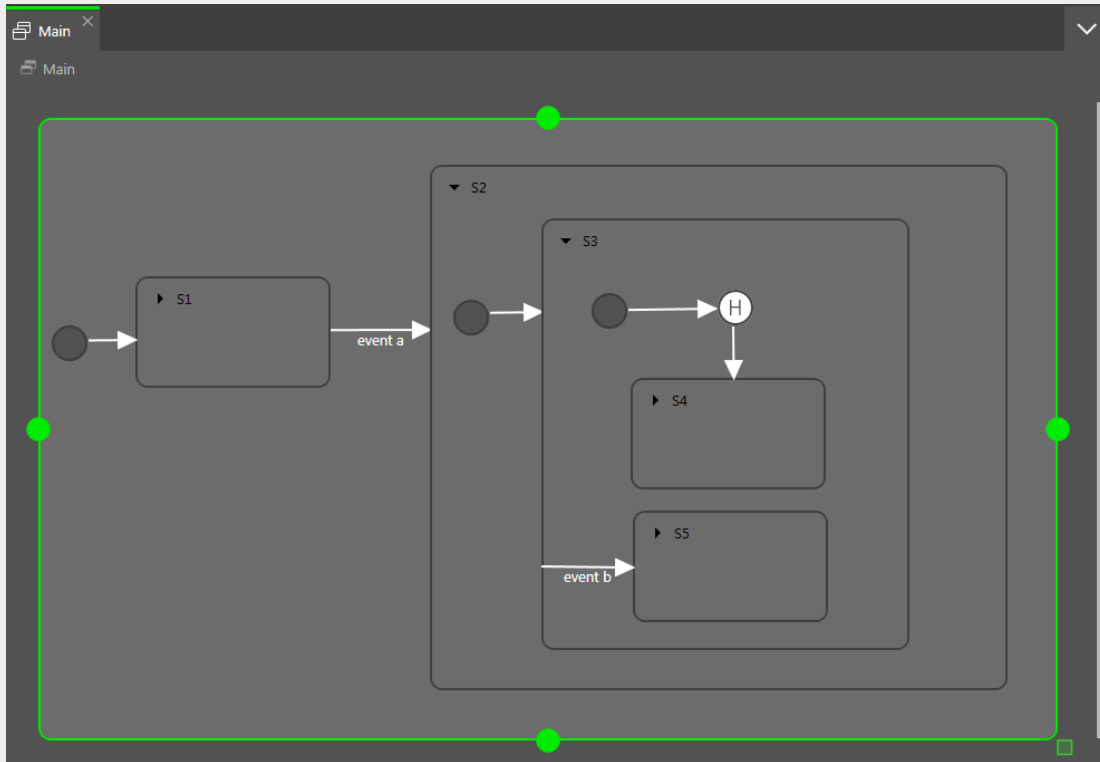


Figure 6.22. Executing a transition

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.

## 6.15.5. 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.

### 6.15.5.1. 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:

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 transition <sup>a</sup>

<sup>a</sup>EB GUIDE does not differentiate between external and local transitions.

### 6.15.5.2. Not supported elements

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

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

### 6.15.5.3. Deviations

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 concepts.

Concept in UML 2.5	Workaround with EB GUIDE
Parallel states	Concept is implemented using dynamic state machines.
Number of triggers per transition	Concept is implemented using EB GUIDE Script in a datapool item or a view.

Concept in UML 2.5	Workaround with EB GUIDE
Time triggers at transitions	Concept is implemented using EB GUIDE Script ( <code>fire_delayed</code> ) in a state machine, a datapool item, a transition, or a view.

## 6.16. 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

### 6.16.1. 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.

### 6.16.2. 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 11.3, “Tutorial: Modeling a path gesture”](#).

### 6.16.3. 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.

---

**TIP****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.

---

### 6.16.4. 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.

## 6.17. Widgets

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

Widgets can be customized: Editing the properties of a widget adapts the widget to individual needs. Example properties are size, color, layout, or behavior when being touched or moved.

Widgets can be combined: Out of small building blocks, complex structures are created. For example, a button can be made up of a rectangle, an image, and a label.

Widgets can be nested: In a widget hierarchy, the subordinate widgets are referred to as child widgets, the superordinate widgets are referred to as parent widgets.

### 6.17.1. View

A view is the topmost widget of each scene. While modeling, basic widgets, 3D widgets, animations, and widget templates are placed into views. Every view is associated to exactly one view state. A view cannot exist without a view state.

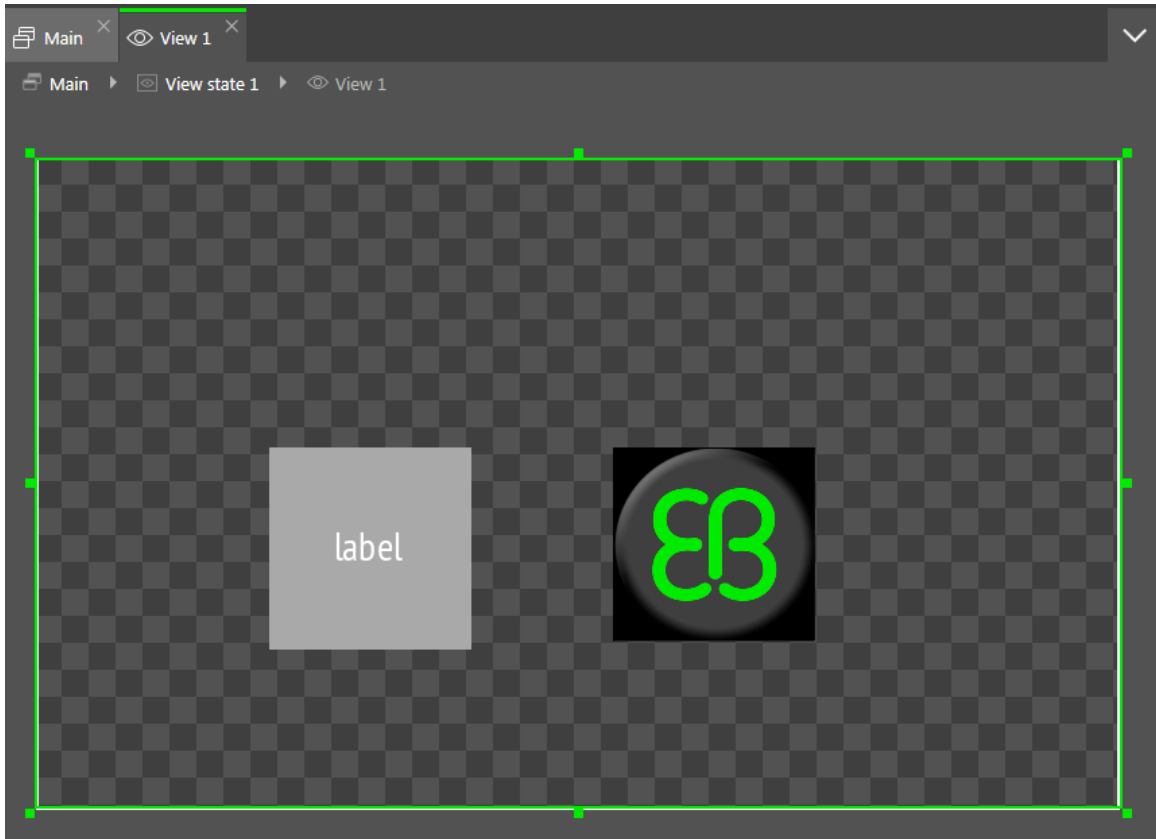


Figure 6.23. A view that contains a rectangle, a label, and an image

## 6.17.2. Widget categories

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

- ▶ Basic widgets

The basic widgets are label, rectangle, container, image, and instantiator.

- ▶ Animations

The **Animations** category provides the animation and a set of curves to specify animation details. For each curve, there is one widget per supported data type.

- ▶ 3D widgets

The **3D widgets** category contains widgets to display a 3D graphic. The 3D widgets are scene graph, scene graph node, material, mesh, camera, directional light, point light, and spot light.

**NOTE**



**Supported renderers**

To display 3D graphics, OpenGL ES 2.0 or DirectX 11 renderer is required.

▶ Widget 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 please refer to our website <https://www.elektrobit.com/ebguide/learn/resources/>.

### 6.17.3. Widget properties

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

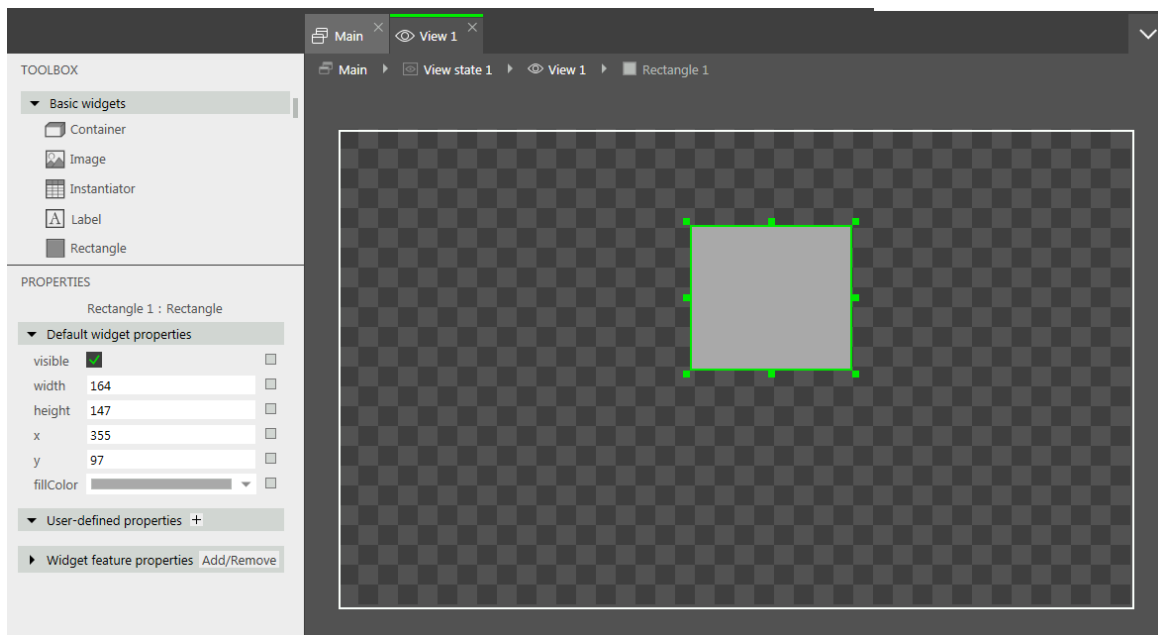


Figure 6.24. A rectangle and its properties

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 12.9, "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.

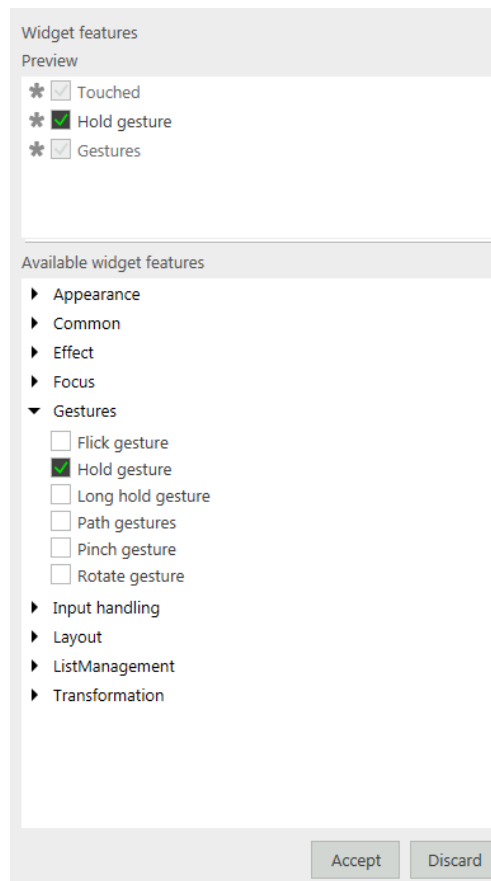


Figure 6.25. Widget features





### Example 6.30. 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 two integer properties `touchPolicy` and `touchBehavior` determine how the widget reacts on touch input.

## 6.17.4. Widget templates

A widget template allows the definition of a customized widget that can be used multiple times in an EB GUIDE model. You can 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.



### **Example 6.31.** **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 8.6, "Re-using a widget"](#).

## 7. Modeling HMI behavior

### 7.1. Modeling a state machine

#### 7.1.1. Adding a state machine



##### Adding a state machine

Prerequisite:

- The navigation area displays the **All** tab.

##### Step 1

In the navigation area, go to **State machines**, and click .

A menu expands.

##### Step 2

Click a type for the state machine.

A new state machine of the selected type is added.

##### Step 3

Rename the state machine.

#### 7.1.2. Adding 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.



##### Adding a dynamic state machine

You use a dynamic state machine for example to show an error message that overlays the regular screen.

Prerequisite:

- The navigation area displays the **All** tab.

- A state machine, view state, or compound state is added to the EB GUIDE model.

#### Step 1

In the navigation area, go to **Dynamic state machines**, and click .

A menu expands.

#### Step 2

Click a type for the dynamic state machine.

A new dynamic state machine of the selected type is added.

#### Step 3

In the navigation area, click a state machine, view state, or compound state to which you want to run in parallel the dynamic state machine.

#### Step 4

In the **Properties** panel, select the `Dynamic state machine list` check box.

With these steps done, you use EB GUIDE Script functions that are related to dynamic state machines.

For details see [section 11.1, "Tutorial: Adding a dynamic state machine"](#).

## 7.1.3. 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** panel, go to the **Entry action** property, and click **Add**.

#### Step 3

Enter an action using EB GUIDE Script.

For background information see [section 6.12, "Scripting language EB GUIDE Script"](#).

#### Step 4

Click **Accept**.

You defined an entry action for a state machine.

## 7.1.4. 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** panel, go to the `Exit` action property, and click **Add**.

#### Step 3

Enter an action using EB GUIDE Script.

For background information see [section 6.12, “Scripting language EB GUIDE Script”](#).

#### Step 4

Click **Accept**.

You defined an exit action for a state machine.

## 7.1.5. Deleting a state machine



### Deleting a state machine

Prerequisite:

- The navigation area displays the **All** tab.

#### Step 1

In the navigation area, right-click the state machine.

#### Step 2

In the context menu, click **Delete**.

The state machine is deleted.

## 7.2. Modeling states

### 7.2.1. 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 and final state are unique



Inserting initial state and final state is only possible 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.

## 7.2.2. 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 content area, click  to expand the compound state.

#### Step 2

Drag a state from the **Toolbox** into the compound state.

The state is added as a child state to the compound state.

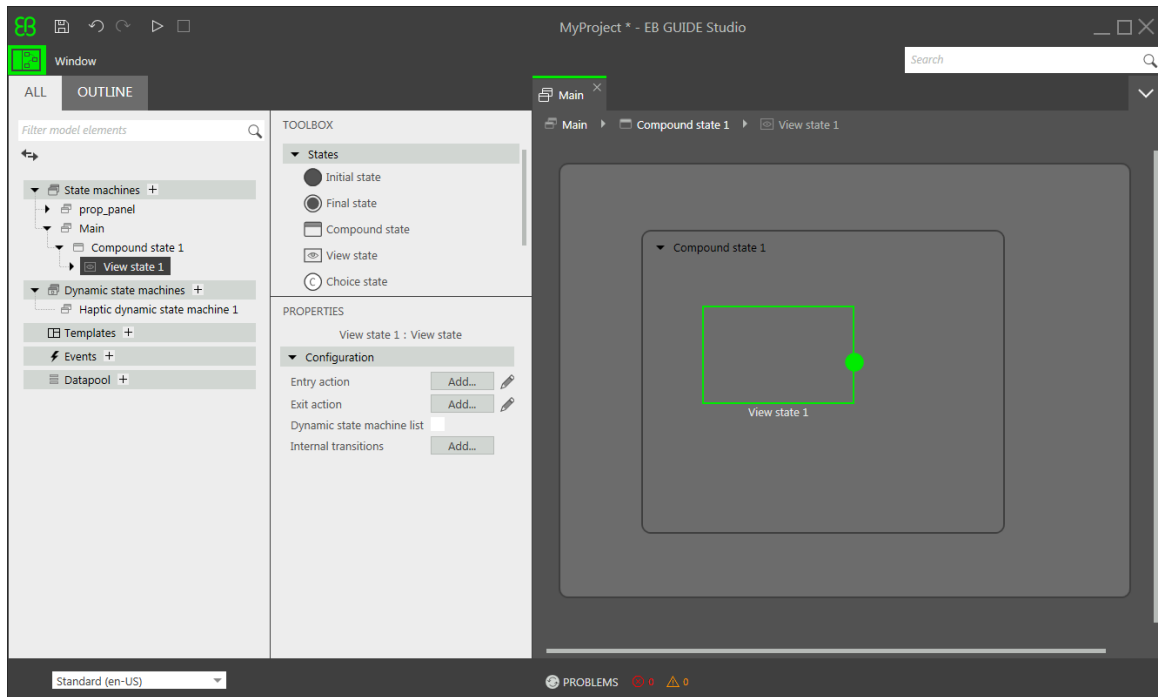


Figure 7.1. A compound state with a nested view state

### 7.2.3. 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 an outgoing transition from the choice state.

#### Step 3

Add a condition to the outgoing transition. For details see [section 7.3.4, “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 4

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 5

Add an outgoing transition from the choice state.

Step 6

In the navigation area, 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`.

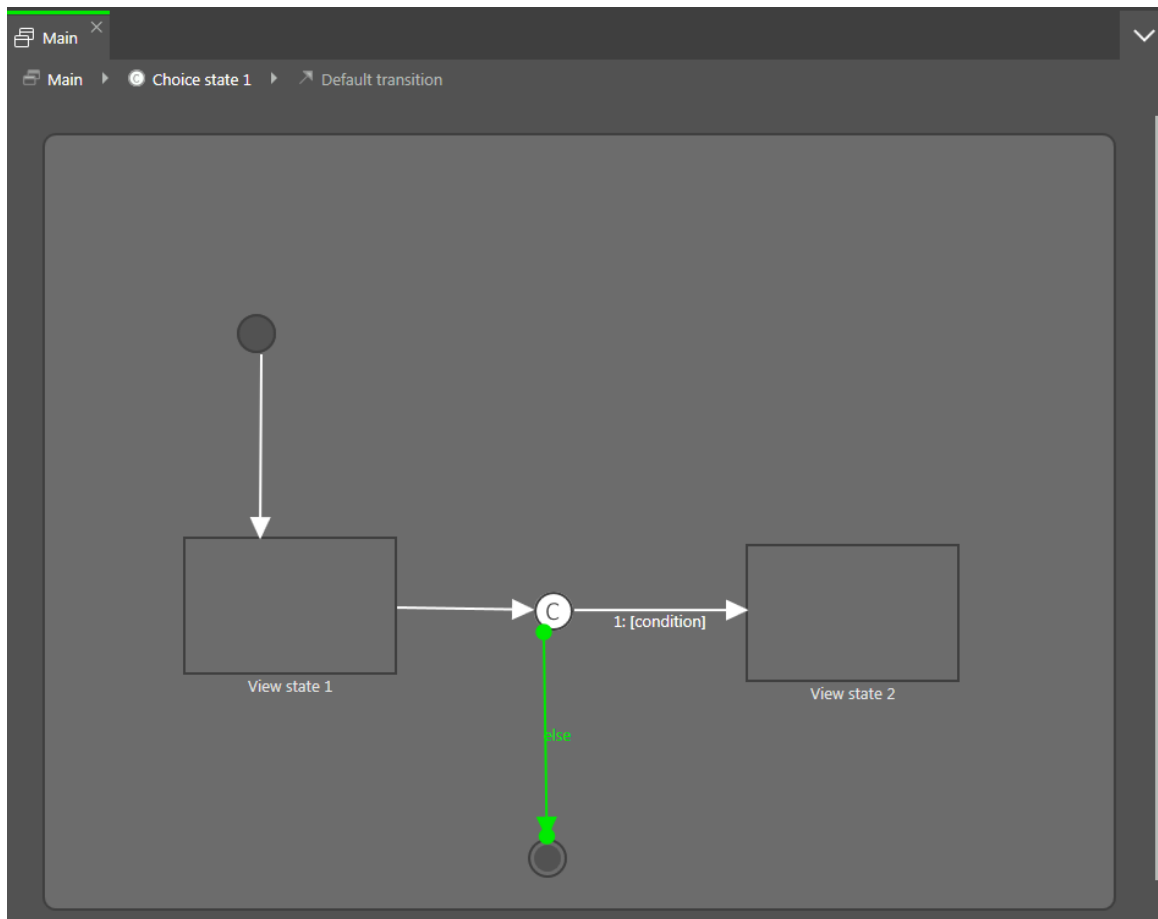


Figure 7.2. A choice state with its choice transitions

## 7.2.4. 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** panel, go to the `Entry action` property, and click **Add**.

#### Step 3

Enter an action using EB GUIDE Script.

For background information see [section 6.12, “Scripting language EB GUIDE Script”](#).

#### Step 4

Click **Accept**.

## 7.2.5. 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** panel, go to the `Exit action` property, and click **Add**.

#### Step 3

Enter an action using EB GUIDE Script.

For background information see [section 6.12, “Scripting language EB GUIDE Script”](#).

[Step 4](#)  
Click **Accept**.

## 7.2.6. 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 area, right-click a model element.

[Step 2](#)  
In the context menu, click **Delete**.

The model element is deleted.

## 7.3. Connecting states through transitions

### 7.3.1. 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.

[Step 2](#)  
Click the green drag point, and keep the mouse button pressed.

Step 3

Drag the mouse into the target state.

Step 4

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

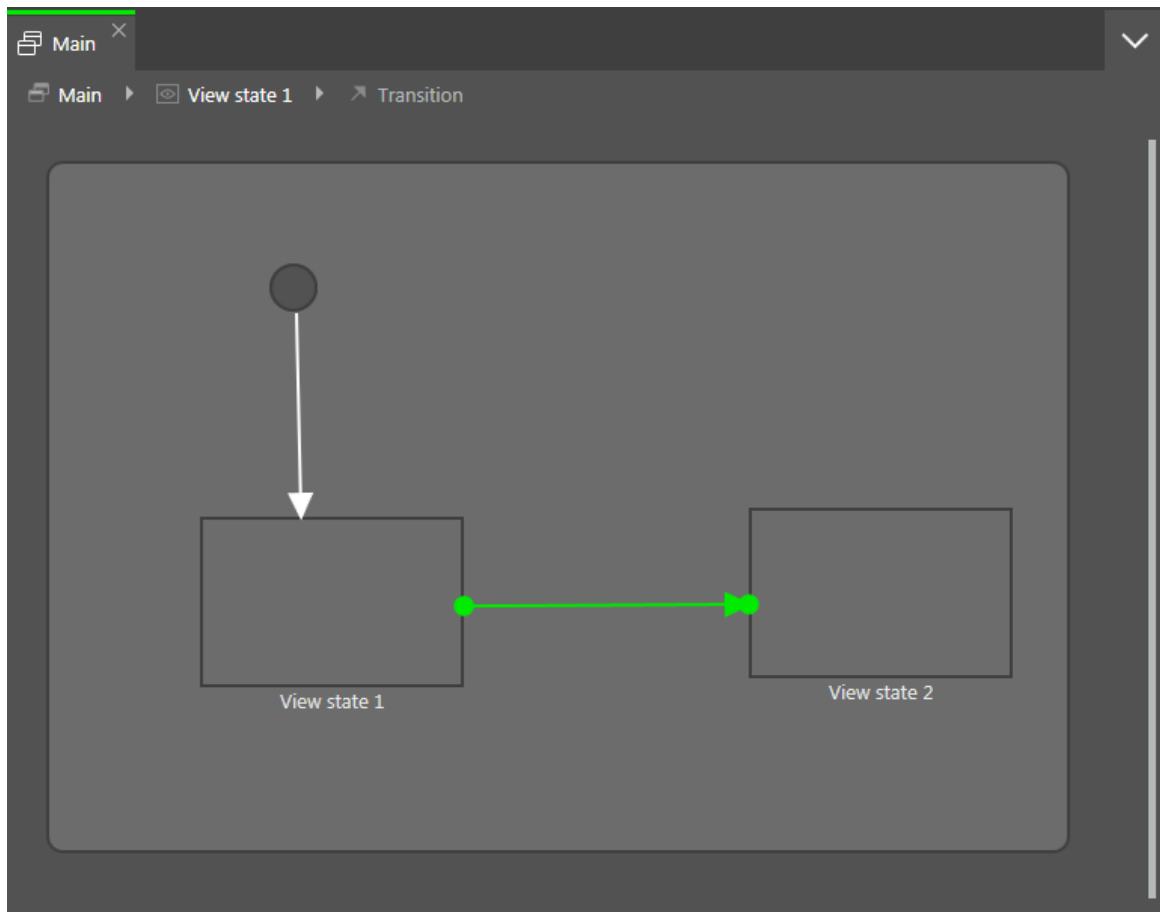


Figure 7.3. A transition

A transition is added and 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.

## 7.3.2. 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

Click the drag point you would like to move, and keep the mouse button pressed.

#### 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.

## 7.3.3. 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** panel, expand the **Trigger** combo box.



Step 3

Select an event.

Step 4

To create a new event, enter a name in the **Trigger** combo box, and click **Add event**.

The event is added as a transition trigger.

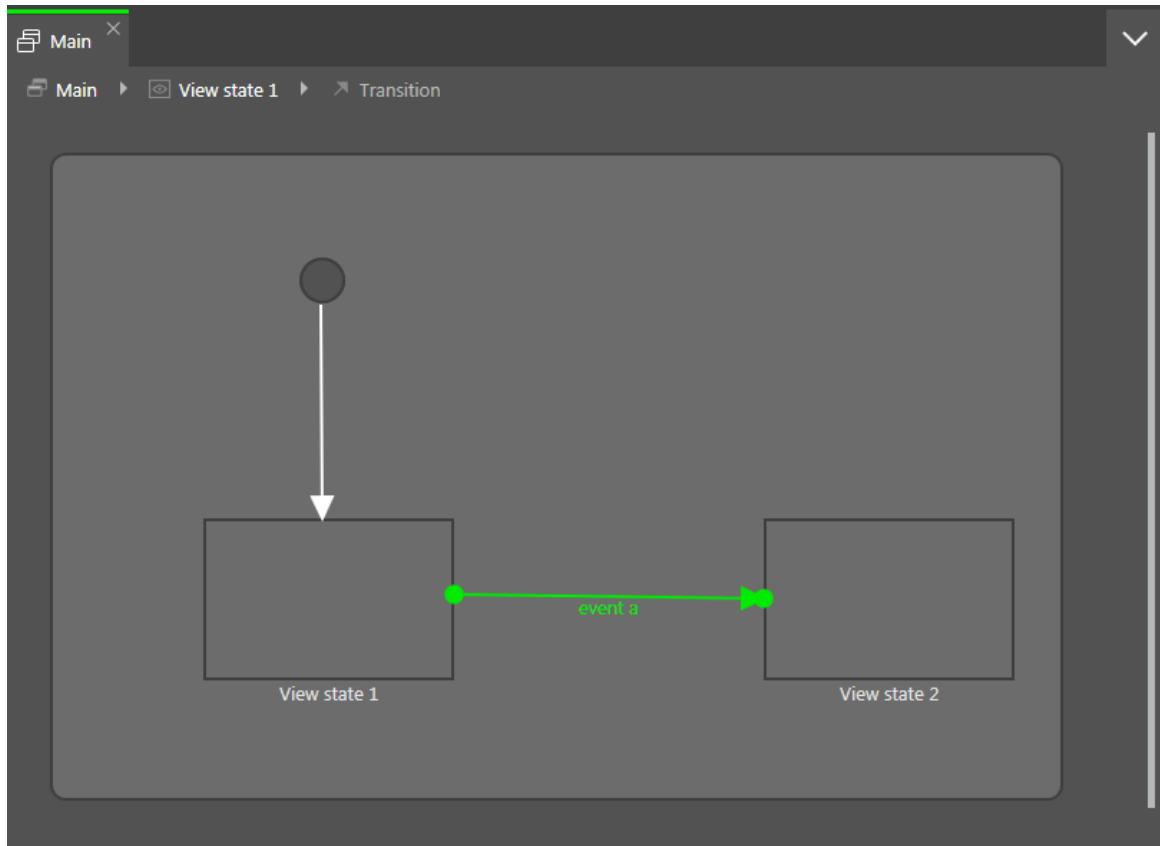


Figure 7.4. A transition with a trigger

### 7.3.4. 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** panel. Next to the `Condition` property, click **Add**.

Step 3

Enter a condition using EB GUIDE Script.

For background information see [section 6.12, "Scripting language EB GUIDE Script"](#).

Step 4

Click **Accept**.

The condition is added to the transition.

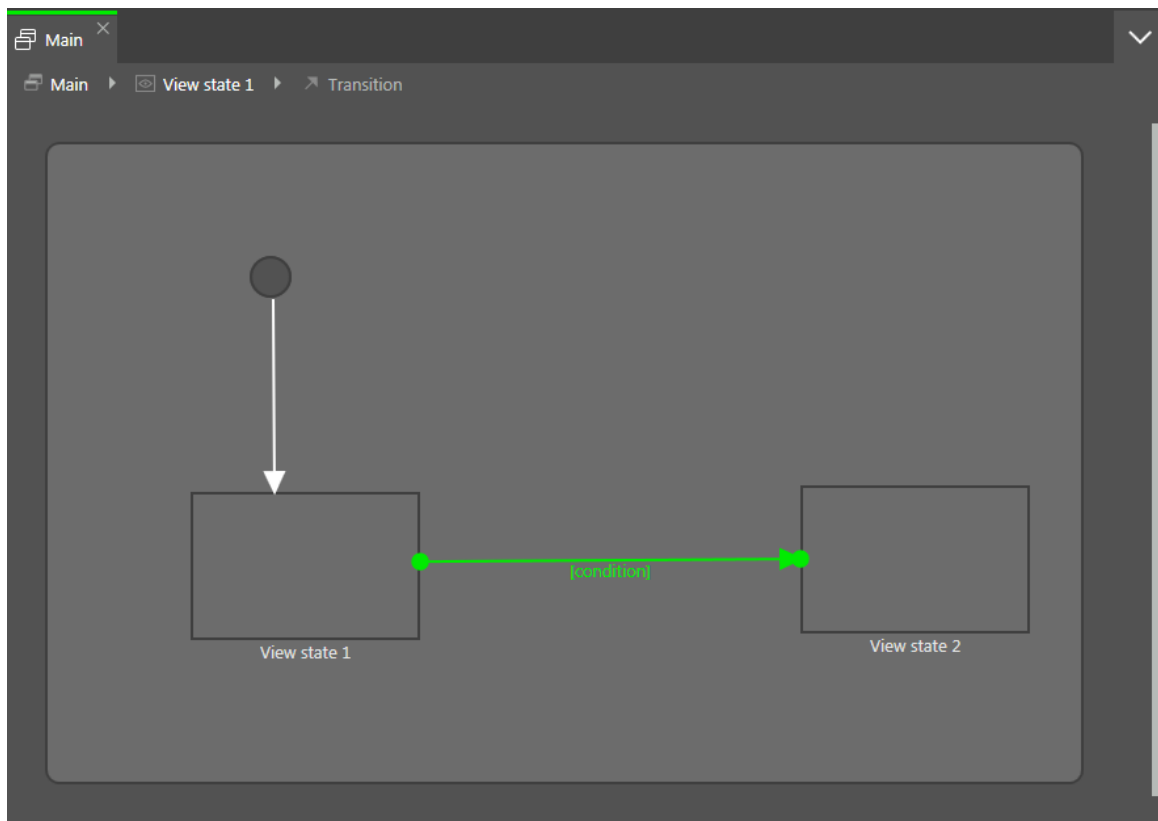


Figure 7.5. A transition with a condition

### 7.3.5. 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** panel. Next to the `Action` property, click **Add**.

Step 3

Enter an action using EB GUIDE Script.

For background information see [section 6.12, "Scripting language EB GUIDE Script"](#).

Step 4

Click **Accept**.

The action is added to the transition.

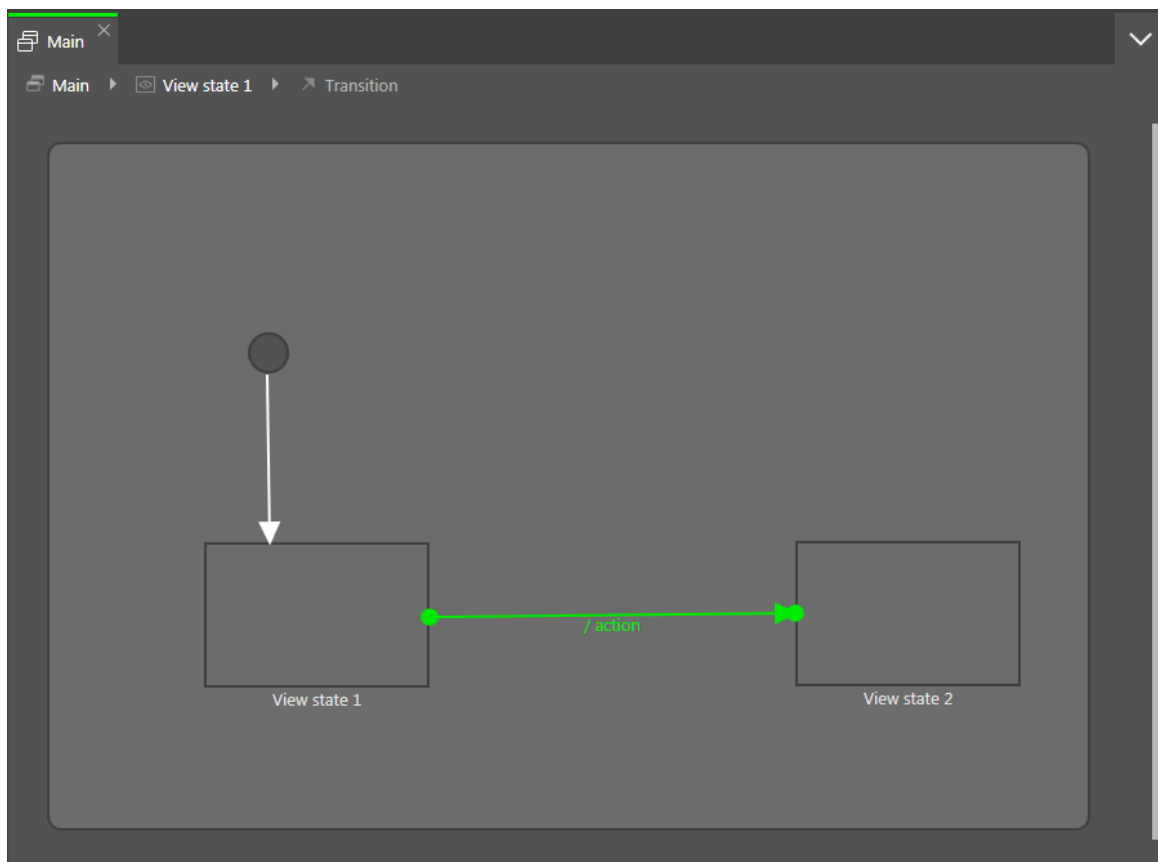


Figure 7.6. A transition with an action

## 7.3.6. Adding an internal transition to a state



### Adding an internal transition to a state

Prerequisite:

- A state machine contains a state.

#### Step 1

Select a state.

#### Step 2

In the **Properties** panel, go to **Internal transitions**, and click **Add**.

An internal transition is added to the state. The internal transition is visible in the navigation area.

## 8. Modeling HMI appearance

### 8.1. Working with widgets

#### 8.1.1. 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 model.

##### Step 2

In the navigation area, 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.

#### **TIP**



#### **Copying and finding views**

Alternatively, you can copy and paste an existing view using the context menu or **Ctrl+C** and **Ctrl+V**.

To find a specific view within your EB GUIDE model, enter the name of the view in the search box or use **Ctrl+F**. To jump to a view, double-click it in the hit list.

#### 8.1.2. Adding a widget to a view



### Adding a widget to a view

Prerequisite:

- The content area displays a view.

#### Step 1

Drag a widget from the **Toolbox** into the view.

The widget is added to the view.

#### **TIP**



#### **Copying and finding widgets**

Alternatively, you can copy and paste an existing widget using the context menu or **Ctrl+C** and **Ctrl+V**.

To find a specific widget within your EB GUIDE model, enter the name of the widget in the search box or use **Ctrl+F**. To jump to a widget, double-click it in the hit list.

## 8.1.3. 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 area, 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.

## 8.1.4. Adding an image to a view



### Adding an image to a view

Prerequisite:

- An image file is located in the `$GUIDE_PROJECT_PATH\resources` directory. For supported file types see [section 12.9.2.3, “Image”](#).
- The content area displays a view.

Step 1

Drag an image from the **Toolbox** into the view.

Step 2

In the **Properties** panel, select an image from the `image` drop-down list box.

The view displays the image.

## 8.1.5. 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. 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** panel, click **Import file**.

A dialog opens.

#### Step 3

Navigate to the directory 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 area displays the imported widget tree with the scene graph as a parent node.

#### **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`.

## 8.1.6. Changing the font of a label



### Changing the font of a label

Prerequisite:

- A `.ttf` file is located in the `$GUIDE_PROJECT_PATH\resources` directory.
- 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** panel, select a font from the `font` drop-down list box.

The view displays the label with the new font.



## 8.1.7. Grouping widgets using a container



### Grouping widgets

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.

## 8.1.8. Adding an instantiator to a view



### 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.

#### Step 3

Select the instantiator, and go to the **Properties** panel.

##### 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 8.3.1, “Adding a widget feature”](#).

In the view, the basic widget is displayed as many times as specified by the `numItems` property and in the layout specified for the instantiator.

For a detailed example of how to use instantiators see [section 11.4, “Tutorial: Creating a list with dynamic content”](#).

## 8.2. Working with widget properties

### 8.2.1. 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** panel 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.

## 8.2.2. 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** panel displays the properties of the selected widget.

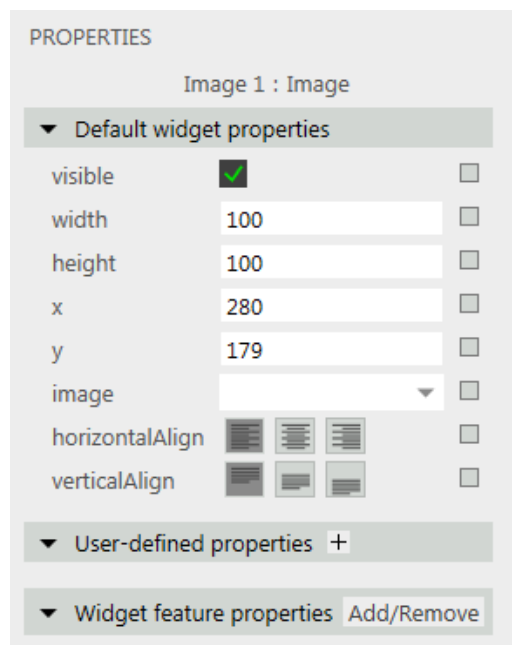


Figure 8.1. Properties of an image

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.

## 8.2.3. 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.

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

Click the rectangle.

The **Properties** panel displays the properties of the rectangle.

### Step 2

In the **Properties** panel, 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.

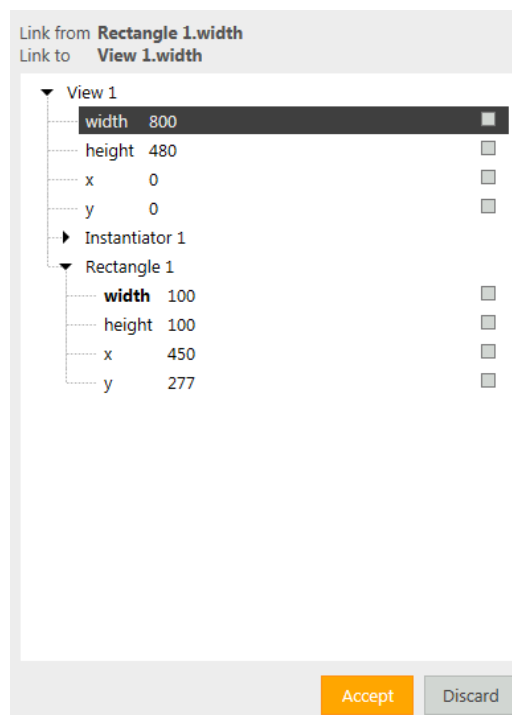



Figure 8.2. Linking between widget properties

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**.

## 8.2.4. 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.

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

Click the image.

The **Properties** panel displays the properties of the image.

Step 2

In the **Properties** panel, 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 combo box.

Step 5

Click **Add datapool item**.

Step 6

Click **Accept**.

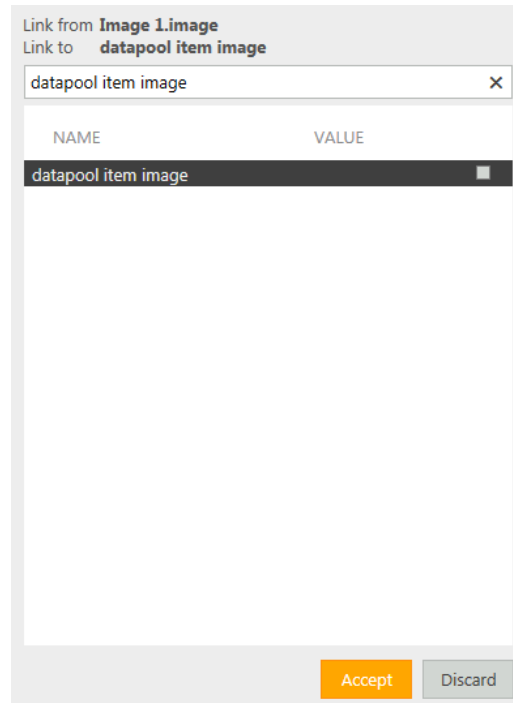



Figure 8.3. Linking to a datapool item

A new datapool item is added.


Step 7

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. 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**.

## 8.2.5. 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** panel displays the properties of the selected widget.

#### Step 2

In the **Properties** panel, 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.

### 8.2.5.1. 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** panel displays the properties of the selected widget.

#### Step 2

In the **Properties** panel, 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 property, click **Edit**.

A script editor opens.

Step 6

Define the behavior of the new function using EB GUIDE Script.

Step 7

Click **Accept**.



**Example 8.1.**  
**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`.

In any EB GUIDE Script code in the EB GUIDE model, you can call the script in the property as follows:

```
"Background color".change()
```

## 8.2.6. 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 area, select the widget with the user-defined property.

Step 2

In the **Properties** panel, click the  button next to the property.

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. 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.

### 8.3.1. Adding a widget feature



#### Adding a widget feature

Prerequisite:

- The EB GUIDE model contains a widget.

Step 1

In the navigation area, click a widget.

The **Properties** panel displays the properties of the selected widget.

Step 2

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

The **Widget features** dialog is displayed.

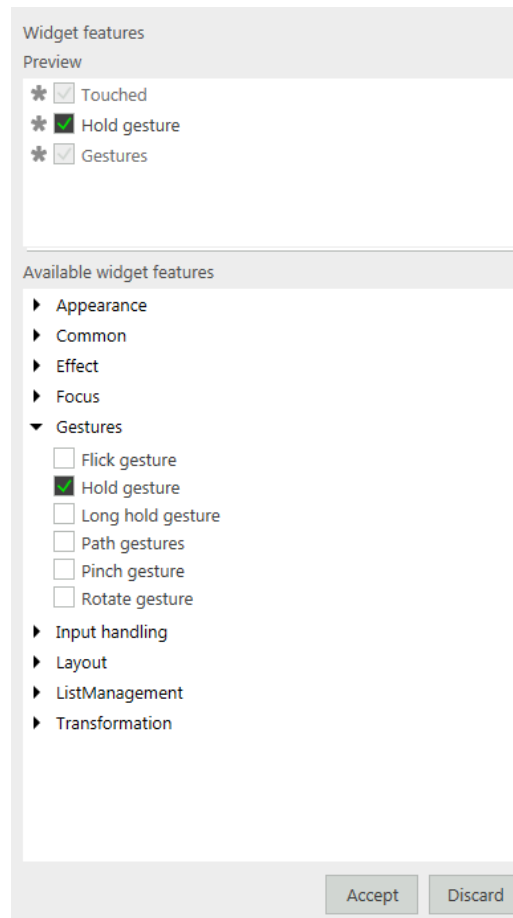


Figure 8.4. Widget features dialog

### 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**.

### TIP



#### 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 12.10, "Widget features"](#).

For tutorials see the following:

- ▶ [section 11.3, “Tutorial: Modeling a path gesture”](#)
- ▶ [section 11.4, “Tutorial: Creating a list with dynamic content”](#)
- ▶ [section 11.2, “Tutorial: Modeling button behavior with EB GUIDE Script”](#)

## 8.3.2. 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 area, click a widget.

The **Properties** panel displays the properties of the selected widget.

#### Step 2

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

The **Widget features** dialog is displayed.

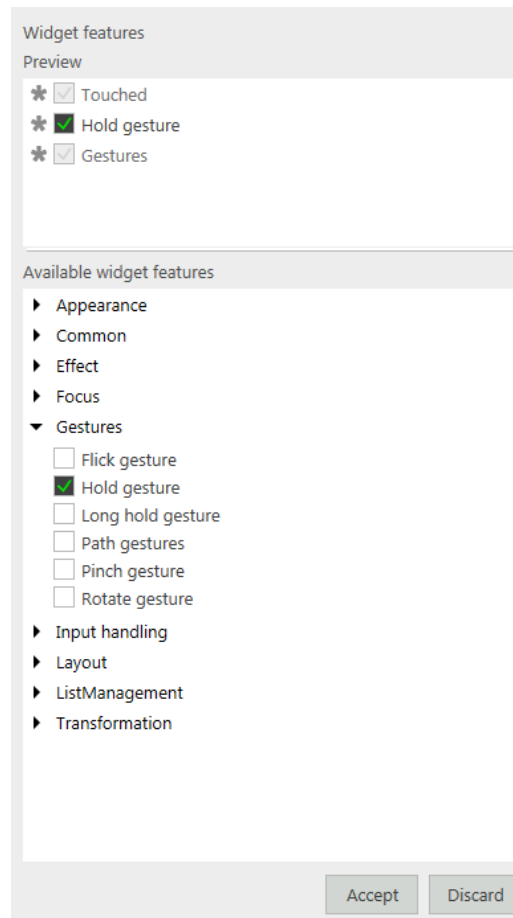


Figure 8.5. Widget features dialog

### Step 3

Under **Preview** clear the widget feature you want to remove.

Click **Accept**.

The related widget feature properties are removed from the **Properties** panel.

#### 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.

## 8.4. Adding a language to the EB GUIDE model

To enable language support during run-time, you add languages to the EB GUIDE model.

## 8.4.1. Adding a language



### Adding a language

The first language in the list is always the default language and cannot be deleted. If you add a language, the language uses the standard language settings as initial values.

#### 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, click **Add**.

A language is added to the table.

#### Step 4

Press **F2**, and 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.

You added a language.

For instructions on how to change the language during run-time see [section 11.6, “Tutorial: Adding a language dependent text to a datapool item”](#).

## 8.4.2. Deleting a language



### 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, select a language.

#### Step 4

In the content area, click **Delete**.

The language is deleted from the table.

## 8.5. Adding animations

### 8.5.1. Animating a widget



#### Animating a widget

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

Double-click the view state.

The view is displayed in the content area.

#### Step 2

Drag one of the basic widgets from the **Toolbox** into the view.

#### Step 3

Drag an animation from the **Toolbox** into the widget you added.

#### Step 4

Drag a curve from the **Toolbox** into the widget you added.

#### Step 5

In the navigation area, move the curve in the hierarchy so that it becomes a child widget of the animation.

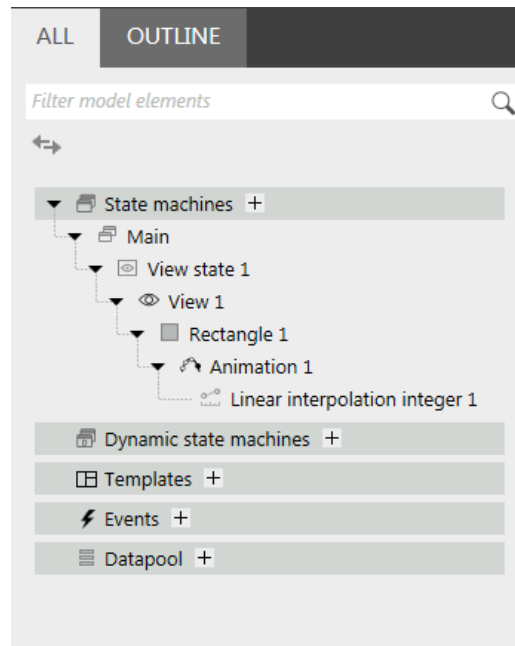


Figure 8.6. Widget hierarchy with an animation and a curve child widget

#### Step 6

Select the basic widget, and add a user-defined property of type `Conditional script`. For details see [section 8.2.5, “Adding a user-defined property to a widget”](#).

#### Step 7

Next to the name of the property, click **Edit**.

A script editor opens in the content area.

#### Step 8

Enter the following EB GUIDE Script:

```
function(v:arg0::bool)
{
  f:animation_play(v:this->"Animation 1")
}
```

**Animation 1** is the default name of the animation that is added first. If the animation you added in step two has a different name, replace the name in the **On trigger** script.

#### Step 9

Select the curve you added in step four.

#### Step 10

Add a link from the curve's `target` property to the property you would like to animate. For details see [section 8.2.3, “Linking between widget properties”](#).



**NOTE**



**Identical types for linked properties**

The type of the curve's `target` property and the basic widget's animated property must be identical. If the basic widget does not have a property of the required type, change the curve to a different type.

Step 11

Start the simulation.

The linked property of your widget gradually changes as specified by the curve you added.

As a follow-up step, you can change the properties of the animation or the curve to change the way the basic widget is animated. For details on curves and for a description of curve properties see [section 12.9.3, “Animations”](#). For a concrete animation example see [section 11.5, “Tutorial: Making a rectangle move across the screen”](#).

## 8.5.2. Animating a view transition



### Adding an entry animation

To make a view appear with a moving or fading animation, you add an entry animation to a view template.

Prerequisite:

- A view template is added.

Step 1

In the navigation area, click a view template.

Step 2

Go to the **Properties** panel.

Step 3

To define an animation that is played when the view is entered, select the **Entry animation** check box.

Step 4

From the **Transition type** drop-down list box, select a type for the view transition.

Step 5

Enter a duration in milliseconds in the **Duration** text box.

Step 6

Select the **Play after exit animation** check box.

Result: Every view you derive from this view template is entered with the animation you defined. With the **Play after exit animation** check box you defined that the entry animation waits until the exit animation of the previous view is finished.



### Adding an exit animation

To make a view disappear with a moving or fading animation, you add an exit animation to a view template.

Prerequisite:

- A view template is added.

#### Step 1

In the navigation area, click a view template.

#### Step 2

Go to the **Properties** panel.

#### Step 3

To define an exit animation that is played when the view is entered, select the **Exit animation** check box.

#### Step 4

From the **Transition type** drop-down list box, select a type for the view transition.

#### Step 5

Enter a duration in milliseconds in the **Duration** text box.

#### Step 6

Enter a delay in milliseconds in the **Delay** text box.

Result: Every view you derive from this view template is exited with the animation you defined.

## 8.6. Re-using a widget

### 8.6.1. Adding a template




### Adding a template

Prerequisite:

- The navigation area displays the **All** tab.

#### Step 1

In the navigation area, go to **Templates**, and click .

A menu expands.

#### Step 2

In the menu, click a type for the template.

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** panel, edit the template's properties, and define the template interface.

#### TIP



#### Templates of templates

A type for the template can be an existing template. EB GUIDE thus allows creating templates from templates.

#### 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 use **Ctrl+F**. To jump to a template, double-click it in the hit list.

## 8.6.2. Defining the template interface



### Defining the template interface


Prerequisite:

- The navigation area displays the **All** tab.
- The EB GUIDE model contains a template.

#### Step 1


Select a template.

#### Step 2

To add a property to the template interface, 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.

### 8.6.3. 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** panel displays the properties which belong to the template interface.

**TIP**



**Define the template interface**


If the **Properties** panel 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** panel, 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**.

### 8.6.4. Deleting a template



## Deleting a template

Prerequisite:

- The navigation area displays the **All** tab.

### Step 1

In the navigation area, right-click a template.

### Step 2

In the context menu, click **Delete**.

The template is deleted.

## 9. Handling data

### 9.1. Adding an event




#### Adding an event

Prerequisite:

- The navigation area displays the **All** tab.

#### Step 1

In the navigation area, go to **Events**, and click .

An event is added to the navigation area.

#### Step 2

Rename the event.

#### 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. Adding a parameter to an event



#### Adding a parameter to an event

Prerequisite:

- The navigation area displays the **All** tab.

- An event is added.

#### Step 1


In the navigation area, click an event.

The **Properties** panel displays the properties of the selected event.

#### Step 2

In the **Properties** panel, point to **Parameters**.

#### Step 3

Click .

#### Step 4

Select a type for the parameter.

A parameter of the selected type is added to the event.

#### Step 5

Rename the parameter.

## 9.3. 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.
- The navigation area displays the **All** tab.
- An event is added.

### Step 1

In the navigation area, click an event.

The **Properties** panel displays the properties of the selected event. Go to the **Properties** panel.

### Step 2

Insert an ID in the `Event ID` text box.

### Step 3

Select an event group from the `Event group` drop-down list box.

## 9.4. Deleting an event



## Deleting an event

Prerequisite:

- The navigation area displays the **All** tab.
- An event is added.

### Step 1

In the navigation area, right-click the event.

### Step 2

In the context menu, click **Delete**.

The event is deleted.

## 9.5. Adding a datapool item






### Adding a datapool item

Prerequisite:

- The navigation area displays the **All** tab.

#### Step 1

In the navigation area, go to **Datapool**, and click .

A menu expands.

#### Step 2

In the menu, click a type for the datapool item.

A new datapool item of the selected type is added. The datapool item is prepared for internal use.

#### Step 3

Rename the datapool item.

#### 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 prevent duplicate IDs, the `Reader ID` and `Writer ID` of pasted datapool items is set to `-1` automatically.

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.

## 9.6. Editing datapool items of a list type



### Editing datapool items of a list type

Prerequisite:

- The navigation area displays the **All** tab.
- A datapool item of a list type is added.

#### Step 1

In the navigation area, click a datapool item of a list type.

The **Properties** panel displays the properties of the selected datapool item.

#### Step 2

In the **Properties** panel, go to the `Value` property, and click the  button next to the property.

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 drop-down list 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 next to `Value`.

## 9.7. 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.

Prerequisite:


- The navigation area displays the **All** tab.
- A datapool item is added.
- The datapool item is not language dependent.
- The datapool item is not linked.

#### Step 1

In the navigation area, click a datapool item.

The **Properties** panel displays the properties of the selected datapool item.

#### Step 2

Go to the **Properties** panel, and click the  button next to the `Value` property.

A menu expands.

### Step 3

In the menu, click **Convert to script**.

The datapool item is converted to a scripted value.

### Step 4


Next to the `Value` property, click **Edit**.

A script editor opens in the content area.

### Step 5

Edit the EB GUIDE Script.

### Step 6

To convert the datapool item back to its plain value, click the  button next to the `Value` property.

A menu expands.

### Step 7

In the menu, click **Convert to plain value**.

The datapool item is converted to its plain value.

## 9.8. 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 change a communication context ID, double-click ID text box, and enter a number.

**Step 6**

To run the communication context in an own thread, select **Use own thread**.

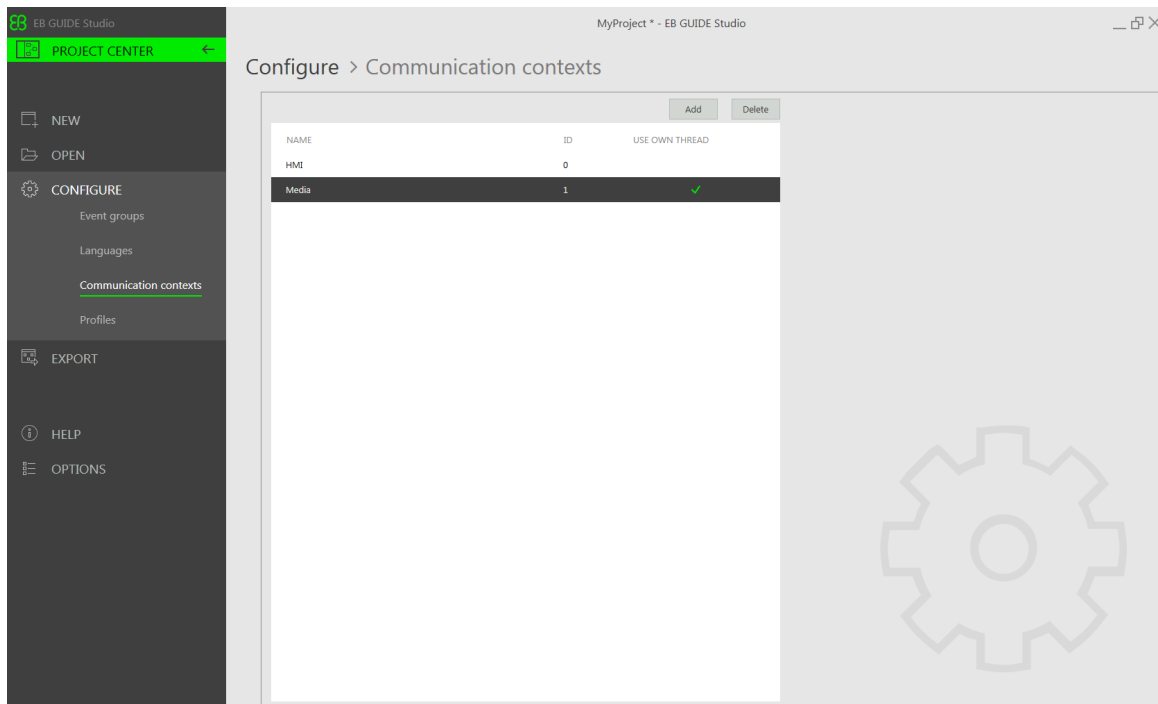


Figure 9.1. Communication context *Media*.



Using external communication in a datapool item

**Prerequisite:**

- At minimum two communication contexts are added to the EB GUIDE model .
- A datapool item is added.

**Step 1**

Open the project editor.

**Step 2**

In the navigation area, click the datapool item.

The **Properties** panel displays the properties of the selected datapool item.

**Step 3**

In the **Properties** panel, select a communication context from the `Reader context` drop-down list box, for example `HMI`.

#### Step 4

In the **Properties** panel, select a different communication context from the `Writer` context drop-down list box, for example `Media`.

The datapool item has two different communication contexts. After the export of the EB GUIDE model, the datapool item sends data from `Reader` context to `Writer` context.

In the instruction above, the data is sent from `HMI` to `Media`.

## 9.9. Linking between datapool items



### Linking between datapool items

Prerequisite:

- The navigation area displays the **All** tab.
- A datapool item is added.
- The datapool item is not language dependent.
- The datapool item is not a scripted value.

#### Step 1

In the navigation area, click a datapool item.

The **Properties** panel displays the properties of the datapool item.

#### Step 2

In the **Properties** panel, go to the `Value` 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 combo box.

#### Step 5

Click **Add datapool item**.

A dialog opens.

#### Step 6

Click **Accept**.

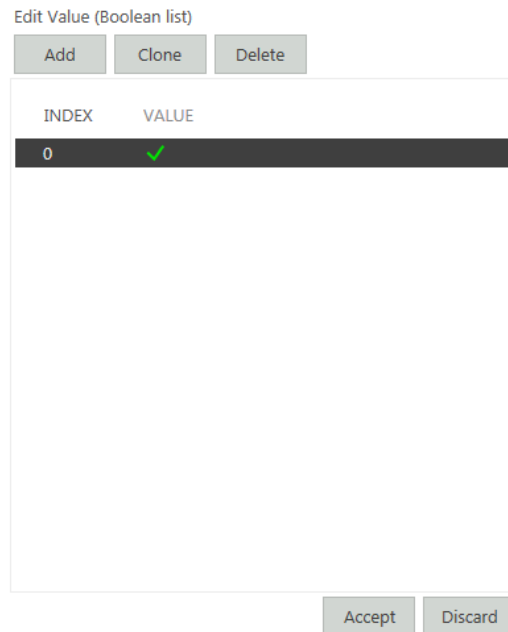



Figure 9.2. Linking between datapool items

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 changes as well.

## 9.10. Deleting a datapool item



### Deleting a datapool item

Prerequisite:

- The navigation area displays the **All** tab.
- A datapool item is added.

#### Step 1

In the navigation area, right-click the datapool item.

#### Step 2

In the context menu, click **Delete**.

The datapool item is deleted.

# 10. Handling a project

## 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

### 10.2.1. Opening a project from the file explorer



### Opening a project from the file explorer

Prerequisite:

- An EB GUIDE Studio project is created.

#### Step 1

Open the file explorer, and select the EB GUIDE Studio project file you would like to open. EB GUIDE Studio project files have the file extension `.ebguide`.

Step 2

Double-click the EB GUIDE Studio 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 Studio 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 Studio project file you would like to open. EB GUIDE Studio project files have the file extension `.ebguide`.

The project opens in EB GUIDE Studio.

## 10.3. Testing and improving an EB GUIDE model

Before exporting an EB GUIDE model to the target device, you resolve errors and simulate the model on your PC.

### 10.3.1. Validating an EB GUIDE model



### Validating an EB GUIDE model

In the problems area, EB GUIDE displays the following:



- ▶ errors
- ▶ warnings

Step 1

In the problems area, click .

The number of errors and warnings is displayed.

Step 2

Click **Problems** to expand the problems area.

A list of errors and warnings is displayed.

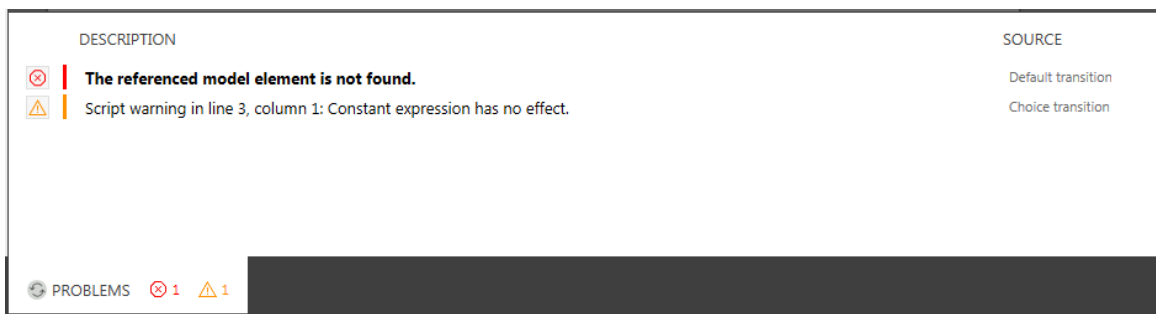


Figure 10.1. Problems area

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 area.

Step 6

To collapse the problems area, click **Problems** once again.

## 10.3.2. 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.

## 10.4. Exporting an EB GUIDE model



### Exporting an EB GUIDE model

To copy the EB GUIDE model to the target device, you need to export it in EB GUIDE Studio.

For every export of an EB GUIDE model you select a profile. Profiles write the EB GUIDE TF start-up configuration file `gtfStartup.cfg`.

#### 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

Click **Browse**, and select a location where to export the binary files.

#### Step 5

Click **Select folder**.

#### Step 6

Click **Export**.

The binary files are exported to the selected location.

**TIP**



**Use command line to export an EB GUIDE model**

To export an EB GUIDE model, you can also use a command line option `-e <project file, destination dir, profile>`.

## 10.5. 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.

## 10.6. Configuring profiles

EB GUIDE Studio offers the possibility to create different profiles for an EB GUIDE model. Profiles write the EB GUIDE TF start-up configuration file `gtfStartup.cfg`.

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**.

## 10.6.1. Cloning a profile



### Cloning a profile

Prerequisite:

- An EB GUIDE Studio 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 the radio button **Use for simulation**.

The `MySimulation` profile is used for simulation on the PC.

## 10.6.2. Adding a library



### 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 sections shows you how to add a user-defined library that interacts with the EB GUIDE model and provides additional functionality.

Prerequisite:


- An EB GUIDE Studio project is opened.
- The project center is displayed.

- In the navigation area, the tab **Configure > Profiles** is selected.
- A profile `MySimulation` is added.
- A library `MyLibraryA` is available in `$GUIDE_PROJECT_PATH\resources`.

Step 1

In the content area, select the `MySimulation` profile.

Step 2

Click  to expand the libraries.

The **Load** table with all included libraries is displayed.

Step 3

Click **Add**.

A new row is added to the table.

Step 4

In the table select `MODEL_PATH` from the drop-down list box under **Location**.

Step 5

Enter `MyLibraryA` in the **Name** text box.

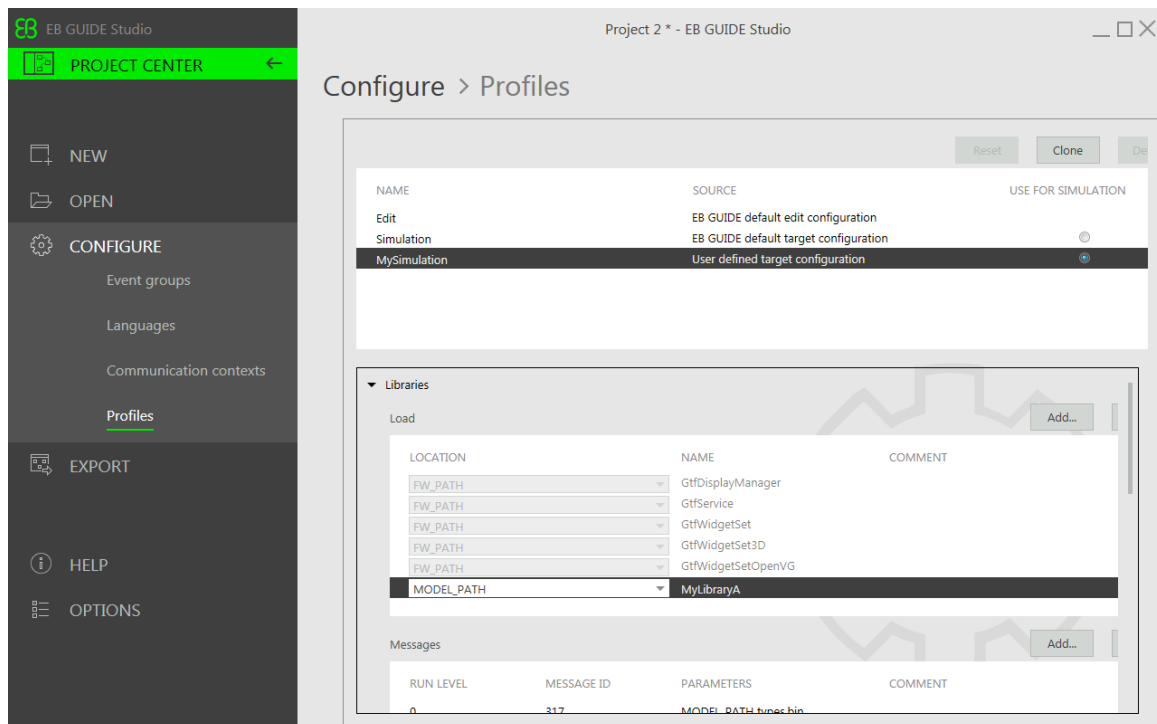


Figure 10.2. Table of libraries

You added the library `MyLibraryA` to the start-up code. `MODEL_PATH` indicates a directory relative to the `gtfStartup.cfg` configuration file.

Use `FW_PATH` to indicate a directory relative to the `GtfStartup` executable file.

### 10.6.3. Adding messages

You can start and stop software modules or alter the behavior of software modules by sending system messages. System messages have a run level that defines at which point during the start-up process they are sent. Additionally system messages have an identifying ID and optional parameters.

For details see *Software module structure of EB GUIDE TF* in the EB GUIDE TF documentation.

#### NOTE



#### Predefined messages in EB GUIDE TF

Message ID range 0...0xFFFF is reserved for EB GUIDE TF and the EB GUIDE product line.

Message ID range 0x10000...0xFFFFFFFF can be managed by you.

Message IDs and parameters of pre-defined messages are documented in the [GtfMessageId.h](#) file and in [section 12.6, "gtfStartup.cfg configuration file"](#).



#### Adding messages

Prerequisite:

- An EB GUIDE Studio project is opened.
- The project center is displayed.
- In the navigation area, the tab **Configure > Profiles** is selected.

#### Step 1

In the content area, select a profile.

#### Step 2

Click  to expand the libraries.

#### Step 3

The **Messages** table with all included libraries is displayed.

#### Step 4

Click **Add**.

A new row is added to the table.

#### Step 5

Enter 0 in the **Run level** text box.

#### Step 6

Enter 300 in the **Message ID** text box.

#### Step 7

Enter `UINT32 0xDEADBEEF` in the **Parameters** text box.

You added a system message.

The message `GTF_MID_GTF_CORE_CREATE_MODEL` makes EB GUIDE GTF create a `GtfCoreModel` with the ID `0xDEADBEEF`.

## 10.6.4. 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 Studio 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  to expand the scenes.

#### 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 12.7, “Scenes”](#).

## 10.7. Exporting and importing language dependent texts

## 10.7.1. 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.



### Exporting language dependent texts

To provide texts in the user's preferred language, you export all language dependent texts of datapool items and pass on the texts to translators.

Prerequisite:

- A datapool item of type `String` or `String list` is added.
- The datapool item has language support. For information on how to add language dependent texts, see [section 11.6, "Tutorial: Adding a language dependent text to a datapool item"](#).
- At minimum two languages are added to the EB GUIDE model.
- 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 which needs to be translated.

Multi selection is possible.

#### Step 4

Click **Export**.

A dialog opens.

#### Step 5

Select a directory to export the file.

#### Step 6

Click **Select folder**.



Result: The export starts. A file is saved in the selected directory. The file has a language dependent acronym and the format `.xliff`. The file contains values for the source language and values for the target language.

**NOTE**



**One file per language is exported**

For every language you select in the project center, a separate file is exported.

## 10.7.2. Importing language dependent texts



### Import language dependent texts

Prerequisite:

- A datapool item of type `String` or `String list` is added.
- The datapool item has language support. For information on how to add language dependent texts, see [section 11.6, “Tutorial: Adding a language dependent text to a datapool item”](#).
- At minimum two languages are added to the EB GUIDE model.
- The EB GUIDE model is free of errors and warnings.
- At minimum one translated `.xliff` file is available.

#### 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 directory 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**.

# 11. Tutorials

The topics in this chapter are sorted alphabetically.

## 11.1. Tutorial: Adding a dynamic state machine

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.

Prerequisite:

- The navigation area displays the **All** tab.

#### Step 1

In the navigation area, go to **Events**, and click .

An event is added to the navigation area.

#### 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

In the navigation area, go to **Datapool**, 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 area, 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.

Step 3

Rename the dynamic state machine to `Volume control`.

Step 4

In the navigation area, double-click `Volume control`.

The dynamic state machine is displayed in the content area.

Step 5

Drag an initial state from the **Toolbox** into the state machine.

Step 6

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 7

In the navigation area, click the view state.

Step 8

Press the **F2** key, and rename the view state to `Volume`.

Step 9

In the content area, click the initial state.

Step 10

Click the green drag point, and keep the mouse button pressed.

Step 11

Drag the mouse into the view state.

Step 12

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

A transition is added and displayed as a green arrow.



## 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 rectangle. 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 area, 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 area, 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** panel.

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 content area.

The rectangle is added as a child widget to `Slider background`.

Step 7

In the navigation area, 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** panel.

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

Select the `Volume indicator` datapool item from the drop-down list box.

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.

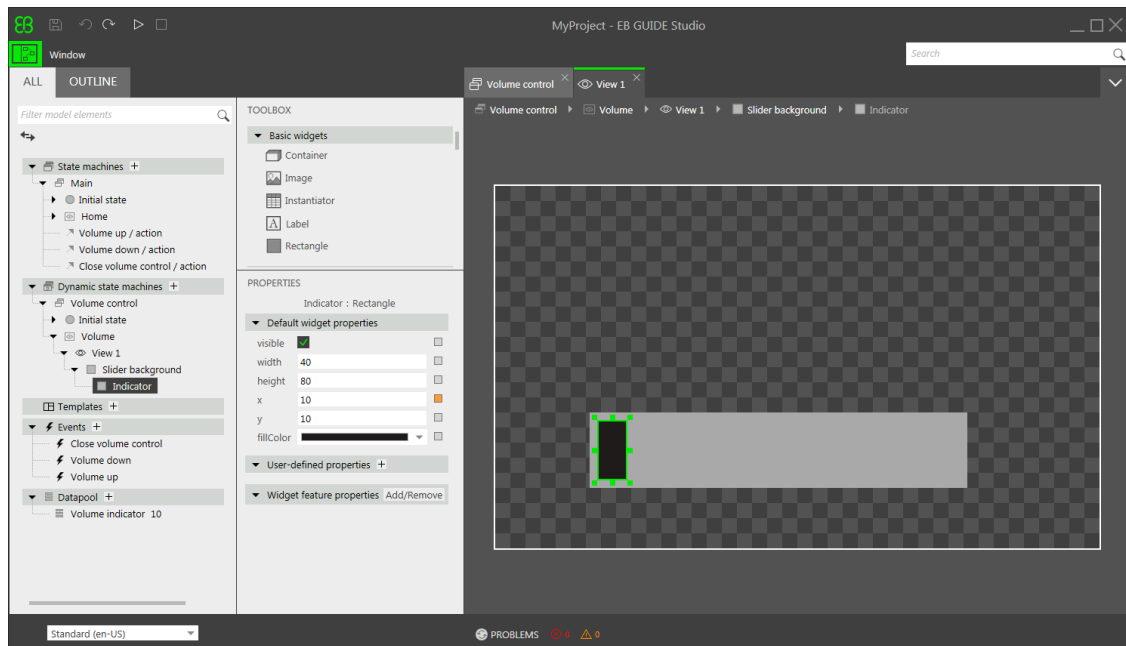


Figure 11.1. Appearance of **View 1** with two rectangles

#### Step 10

In the navigation area, click the `Volume indicator` datapool item.

#### Step 11

Go to the **Properties** panel, and enter 10 in the `Value` text box.

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 area, 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 area, click **Main**.

### Step 8

In the **Properties** panel, 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 area, click the **Main** state machine.

### Step 2

In the **Properties** panel, go to **Internal transitions**, and click **Add**.

An internal transition is added to the state machine. The internal transition is visible in the navigation area.

### Step 3

Add two more internal transitions.

### Step 4

In the navigation area, click the first internal transition.

#### Step 4.1

Go to the **Properties** panel.

#### Step 4.2

In the **Trigger** combo box, select `Volume up`.



Step 4.3

Next to the **Action** property, click **Add**.

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 area, the internal transition is renamed to `Volume up`.

Step 5

In the navigation area, click the second internal transition.

Step 5.1

Go to the **Properties** panel.

Step 5.2

In the **Trigger** combo box, select `Volume down`.

Step 5.3

Next to the **Action** property, click **Add**.

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 area, the internal transition is renamed to `Volume down`.

Step 6

In the navigation area, click the third internal transition.

Step 6.1

Go to the **Properties** panel.

Step 6.2

In the **Trigger** combo box, select `Close volume control`.

Step 6.3

Next to the **Action** property, click **Add**.

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 area, 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.

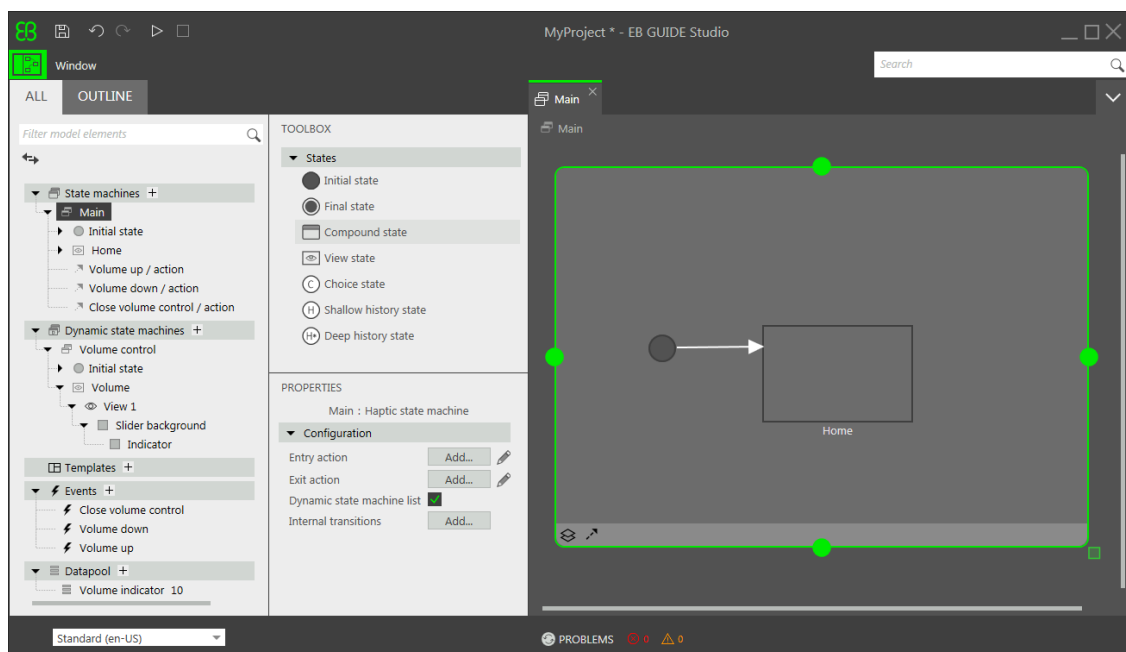


Figure 11.2. EB GUIDE model with all model elements



## 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 the EB GUIDE Monitor toolbar, click **Connect**.

Step 2

In EB GUIDE Monitor, double-click `Volume up` 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.

## 11.2. Tutorial: Modeling button behavior with EB GUIDE Script

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 area, click the rectangle, press the **F2** key, and rename the rectangle to `Background`.

#### Step 3

Drag a rectangle from the **Toolbox** into the navigation area. Place it as a child widget to the `Background` rectangle.

#### Step 4

In the navigation area, click the new rectangle, press the **F2** key, and rename the rectangle to `Button`.

#### Step 5

Drag a label from the **Toolbox** into the navigation area. Place the label as a child widget to the `Button` rectangle.

#### Step 6

In the navigation area, click the label, press the **F2** key, and rename the label to `Button text`.

Your widget hierarchy now looks as follows.

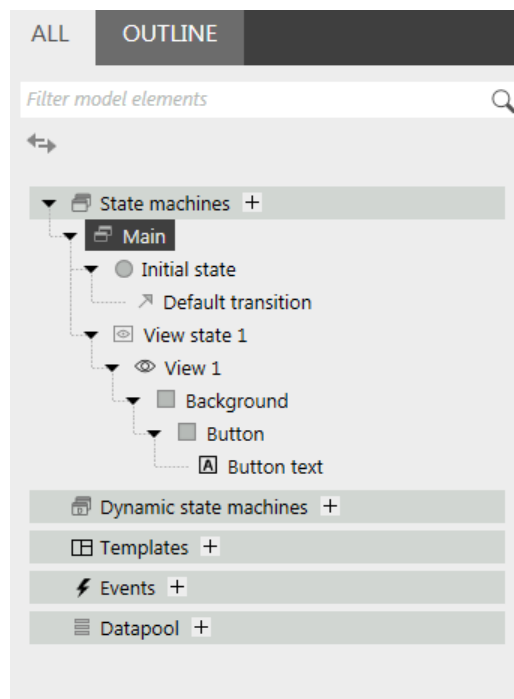


Figure 11.3. Widget hierarchy



## Configuring the background

Prerequisite:

- You completed the previous instruction.

### Step 1

In the navigation area, click the `Background` rectangle, and go to the **Properties** panel.

### 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 navigation area, go to **Datapool**, and 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

Go to the **Properties** panel, and enter 400 in the `Value` text box.



### Configuring the button

Prerequisite:

- You completed the previous instruction.

Step 1

In the navigation area, click the `Button` rectangle, and go to the **Properties** panel.

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** panel.

Step 5

Next to the `touchPressed` property, click **Edit**.

### 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 area, click the `Button text` label, and go to the **Properties** panel.

### 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, click .

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 11.4. 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.

## 11.3. Tutorial: Modeling a path gesture

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** panel displays the properties of the label.

Step 3

In the **Properties** panel, enter 500 in the `width` text box.

Step 4

Select the rectangle.

The **Properties** panel displays the properties of the rectangle.

Step 5

Enter 500 in the `width` text box.

Step 6

In the **Properties** panel, 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** panel displays the properties of the rectangle.

Step 2

In the **Properties** panel, 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** panel.

#### Step 5

For the **Path gestures** widget feature edit the following properties:

##### Step 5.1

Next to the `onPath` property, click **Edit**.

##### 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 **Edit**.

##### 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 **Edit**.

##### 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.

## 11.4. Tutorial: Creating a list with dynamic content

Instantiators allow creating lists dynamically during run-time. Based on a datapool item of a list type, 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 navigation area, go to **Datapool**, and 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 `MyStringList` datapool item, and go to the **Properties** panel.

#### Step 5

Next to the `Value` property, click the  button.

An editor opens.

[Step 5.1](#)

Click **Add**.

A new entry is added to the table.

[Step 5.2](#)

Enter `One` in the `Value` text box.

[Step 5.3](#)

Add the values `Two`, `Three`, `Four`, and `Five` to the `MyStringList` datapool item.

[Step 5.4](#)

Click **Accept**.

You added a datapool item of type `String list`. The datapool item contains five entries.

The content of the list is displayed next to the `Value` property.



## 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 area, 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.

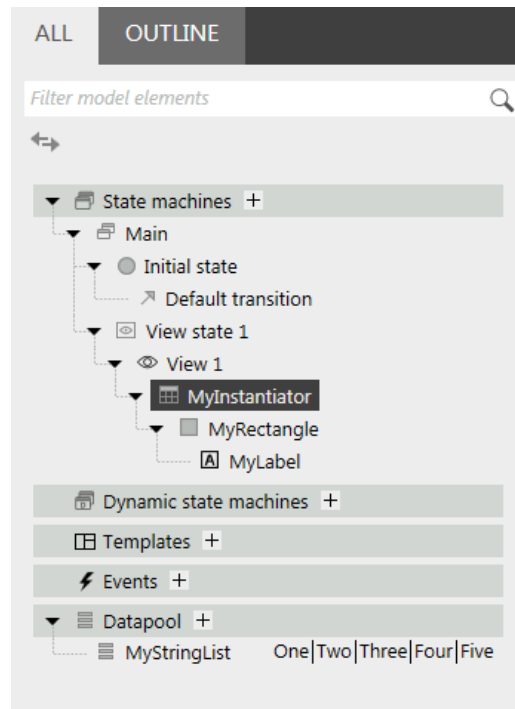


Figure 11.5. Widget hierarchy with an instantiator



## 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** panel.

### 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 `calculateNumItems` property click **Edit**.

A script editor opens in the content area.

Step 5.4

Add the `MyStringList` datapool item to the **Trigger** list.

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.

The related widget feature properties are added to the instantiator and displayed in the **Properties** panel.

Step 6.2

Click **Accept**.

Step 6.3

Enter 5 in the `gap` text box to set a spacing of 5 px between each list element.

Step 6.4

Select **vertical (=1)** from the `layoutDirection` drop-down list box 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 label and go to the **Properties** panel.

### Step 2

Enter 0 in the `x` and `y` text box.

### Step 3

Add a link from the label's `width` property to the rectangle's `width` property.

#### 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 label's `height` property to the rectangle's `height` property.

### Step 5

Next to the `horizontalAlign` property, click .

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** panel.



### Step 2

To make sure that the list elements use the available width, add a link from the rectangle's `width` property to the instantiator's `width` property.

### 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 to the rectangle's properties.

### 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, click **Edit**.

A script editor opens in the content area.

#### Step 5.5

Add the `lineIndex` property of the rectangle and the `MyStringList` datapool item to the **Trigger** list.

#### 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.

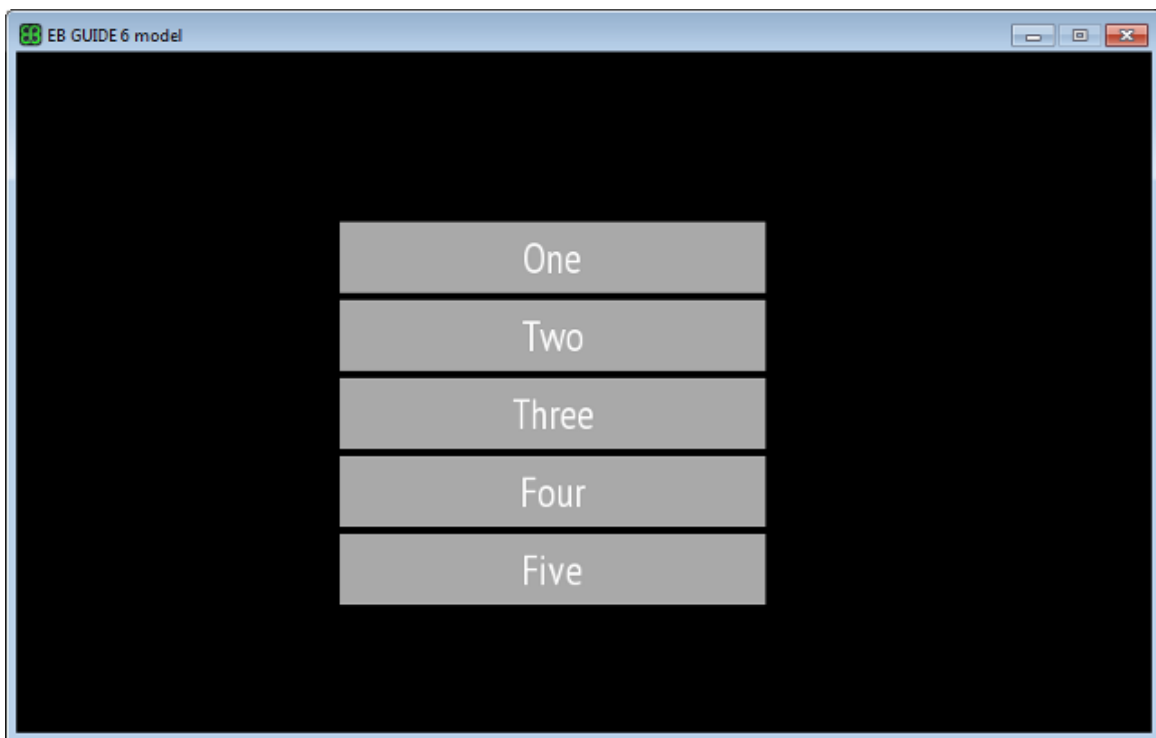


Figure 11.6. List created with an instantiator

## 11.5. Tutorial: Making a rectangle move across the screen

The following instructions guide you through the process of animating a rectangle 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 a rectangle from the **Toolbox** into the view.

#### Step 3

Drag an animation from the **Toolbox** into the rectangle.

#### Step 4

In the navigation area, click the animation, and press the **F2** key. Rename the animation to `MyAnimation`.

#### Step 5

Drag a linear interpolation integer widget from the **Toolbox** into the rectangle.

#### Step 6

In the navigation area, move the linear interpolation integer in the hierarchy so that it becomes a child widget of the animation.

Now, if you start the simulation, a rectangle is displayed in a view. The rectangle 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 rectangle. With the conditional script property, rendering the rectangle during simulation starts the animation.

Prerequisite:

- You completed the previous instruction.

Step 1

Select the rectangle.

Step 2

In the **Properties** panel, 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 rectangle.

Step 4

Rename the property to `startAnimation`.

Step 5

Next to the `startAnimation` property, click **Edit**.

A script editor opens in the content area.

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.

Step 1

Select the linear interpolation integer widget.

Step 2

In the **Properties** panel, go to the `target` 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 rectangle, and select its `x` property.

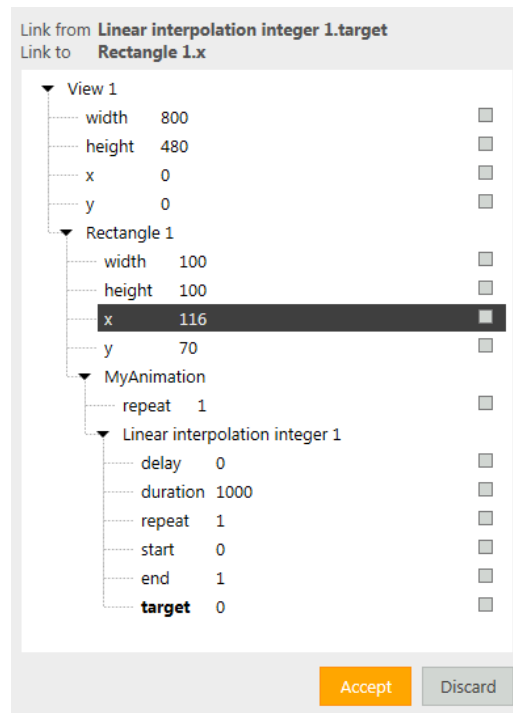



Figure 11.7. Linking between widget properties

Step 5

Click **Accept**.

The dialog closes. The  button is displayed next to the `target` property.

Step 6

Link the `end` property to the view's `width` property.

With these settings, when the animation starts, the `x` property of the rectangle changes from zero to the width of the view. Thus the rectangle moves from the left boundary to the right boundary of the view.

Step 7

To make the animation run in infinite repetitions, enter 0 in the `repeat` property.

Step 8

Save the project.

Step 9

To start the simulation, click  in the command area.

Result:

The rectangle continually moves from the left side of the view to the right side of the view.

## 11.6. Tutorial: Adding a language dependent text to a datapool item

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

### NOTE



### Prerequisites to language dependency

To add language support to a datapool item, do the following:

- ▶ If its `Value` property is linked to another datapool item or widget property, remove the link.
- ▶ If its `Value` property is a scripted value, convert the property to a plain value.



### Linking a widget property to a datapool item

The following instructions guide you through the process of linking the label's `text` property 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 content area displays a view.
- The view contains a label.
- The `text` property of the label is not a scripted value.

#### Step 1

Click the label.

#### Step 2

In the **Properties** panel, 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 combo 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 navigation area, click the `Welcome_text` datapool item.

Step 2

In the **Properties** panel, select the **Language support** check box.

Step 3

In the `Value` text box, enter `Welcome`.

In the content area, the label displays `Welcome`.

Step 4

Below the navigation area, select a language from the drop-down list box, for example `German`.

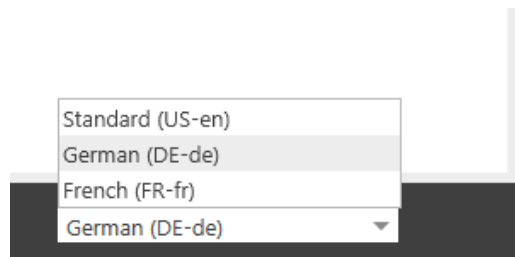


Figure 11.8. Language drop-down list box

Step 5

Go to the **Properties** panel.

Step 6

In the `Value` text box, enter `Willkommen`.

In the content area, the label displays `Willkommen`.

Step 7

Below the navigation area, select a language from the drop-down list box, for example `French`.

Step 8

Go to the **Properties** panel.

Step 9

In the `Value` text box, enter `Bienvenue`.

In the content area, the label displays `Bienvenue`.




### 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 **Navigation area**, go to **Datapool**, and 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 area, click the `Label 1` label.

Step 5

In the Properties panel, 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** panel.

Step 8

Next to the `touchPressed` property, click **Edit**.

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:language(1:German)
    dp:SelectedLanguage = 1
  }
  else if (dp:SelectedLanguage == 1) // German selected
  {
    f:language(1:French)
    dp:SelectedLanguage = 2
  }
  else if (dp:SelectedLanguage == 2) // French selected
  {
    f:language(1:Standard)
    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 runtime.

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.

## 11.7. Tutorial: Working with a 3D graphic

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 OpenGL ES 2.0 and DirectX 11 renderers can display 3D graphics. 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 EB GUIDE Studio 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.

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** panel, click **Import file**.

A dialog opens.

Step 5

Navigate to the directory where the 3D graphic file is stored.

Step 6

Select the 3D graphic file.

#### Step 7

Click **Open**.

The import starts. The **Import successful** dialog is displayed. Here you have the possibility to check the import log file.

#### Step 8

Click **OK**.

The view displays the 3D graphic. The navigation area 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 area, expand `RootNode`.

#### Step 2

Drag a directional light from the **Toolbox** to `RootNode`.

You added a directional light to `My3DGraphic`. The directional light has the same transformation properties in the 3D scene as `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>` directory contains an additional `.ebmesh` file.

### Step 1

In the navigation area, click `Mesh 1`, and go to the **Properties** panel.

### Step 2

Select an `.ebmesh` file from 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>` directory contains a `.png` or `.jpg` image file.

### Step 1

In the navigation area, click the material, and go to the **Properties** panel.

### 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** panel.

### Step 5

In the **Properties** panel, select an image from the `diffuseTexture` drop-down list box.

The view displays a scene graph with the new texture.

**NOTE**



**Usage of 3D widget features**

This instruction is valid for the following widget features from the category **3D**:

- ▶ **Ambient texture**
- ▶ **Emissive texture**
- ▶ **Light map texture**
- ▶ **Normal map texture**
- ▶ **Opaque texture**
- ▶ **Reflection texture**
- ▶ **Specular 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 area, click `My3DGraphic` and go the **Properties** panel.

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 area, expand `RootNode` and `Camera001`.

Step 4

Click `Camera 1` and go to the **Properties** panel.

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** panel.

Step 8

In the navigation area, go to `RootNode`.

Step 9

Drag a camera from the **Toolbox** to the scene graph node `Camera001`.

You added a second camera.

Step 10

Click `Camera 2` and go to the **Properties** panel.

Step 11

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 12

In the **Widget feature properties** category, click **Add/Remove**.

The **Widget features** dialog is displayed.

Step 13

Under **Available widget features**, expand the **3D** category, and select **Camera viewport**.

Step 14

Click **Accept**.

The related widget feature properties are added to `Camera 2` and displayed in the **Properties** panel.

Step 15

In the **Properties** panel, 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.

## 12. References

The following chapter provides you with lists and tables for example parameters, properties, and identifiers.

### 12.1. Android events

Android events belong to the `SystemNotifications` event group and have event group ID 13.

Table 12.1. Android events

Event ID	Name	Description
1	<code>RendererEnabled</code>	Is sent by the application when Android lifecycle management stops or starts the renderer  Parameters: <ul style="list-style-type: none"><li>▶ <code>enabled</code>: If true, the renderer is enabled. If false, the renderer is set to sleep mode.</li></ul>
2	<code>setKeyboardVisibility</code>	Is sent by the EB GUIDE model if a virtual keyboard is intended to be shown  Parameters: <ul style="list-style-type: none"><li>▶ <code>visibility</code>: If true, a virtual keyboard is made visible. If false, it is invisible.</li></ul>
3	<code>onKeyboardVisibilityChanged</code>	Is sent by the application if a virtual keyboard is shown  Parameters: <ul style="list-style-type: none"><li>▶ <code>visibility</code>: If true, a virtual keyboard is visible. If false, it is invisible.</li></ul>
4	<code>onLayoutChanged</code>	Is sent by the application when the visible area of the screen changes  Parameters (in pixels): <ul style="list-style-type: none"><li>▶ <code>x</code>: The x-coordinate of the top left corner of the visible screen area</li></ul>



Event ID	Name	Description
		<ul style="list-style-type: none"> <li>▶ <code>y</code>: The y-coordinate of the top left corner of the visible screen area</li> <li>▶ <code>width</code>: The width of the visible screen area</li> <li>▶ <code>height</code>: The height of the visible screen area</li> </ul>

## 12.2. Datapool items

Table 12.2. Properties of a datapool item

Property name	Description
<code>Value</code>	<p>The initial value of the datapool item</p> <p>If the value is provided by EB GUIDE Studio, the exporter provides the property value to EB GUIDE GTF. Otherwise EB GUIDE GTF initializes the property value at system start-up.</p>
<code>Read-only</code>	<p>If true, only internal communication is available. <code>Value</code> is static during run-time and only changes if you reinitialize it at language switching.</p> <p>If false, external communication is available. <code>Value</code> can change during run-time.</p>
<code>Reader ID</code>	<p>The address that the reader's communication context uses to access the datapool item. If <code>Reader ID</code> is not defined, it is calculated automatically. If you modify an EB GUIDE model, it is possible that the <code>Reader ID</code> for the datapool item changes.</p>
<code>Reader context</code>	<p>The communication context which is notified about changed values and reacts on value changes.</p>
<code>Writer ID</code>	<p>The address that the writer's communication context uses to access the datapool item. If <code>Writer ID</code> is not defined, it is calculated automatically. If you modify an EB GUIDE model, it is possible that the <code>Writer ID</code> for the datapool item changes.</p>
<code>Writer context</code>	<p>The communication context which writes new values</p>
<code>Windowed</code>	<p>Available in lists only</p> <p>If true, EB GUIDE TF handles the datapool item in windowed list operating mode. No default value is used for initialization.</p> <p>If false, EB GUIDE TF handles the datapool item in standard list operating mode.</p>



## 12.3. 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.

### 12.3.1. Mesh

Mesh defines the shape of the 3D object.

Available operations are as follows:

- ▶ assign (writable properties) (=)

It is possible to store meshes in a list. For details about lists see [section 12.3.9, "List"](#).

### 12.3.2. 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 12.3.9, "List"](#).

### 12.3.3. 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 12.3.9, “List”](#).

## 12.3.4. 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.

- ▶ The **Trigger** drop-down list box contains a list of events and datapool items that trigger the execution of the **On trigger** script.
- ▶ The **On trigger** script is called on initialization, after an event trigger, or after a value update of a datapool item..

The parameter of the **On trigger** script indicates the cause for the execution of the script.

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

If true, it triggers a change notification.

If false, it does not trigger a change notification.

## 12.3.5. 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 12.3.9, “List”](#).

## 12.3.6. Font

To add a font to an EB GUIDE project, copy the font file in the following directory: `$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 12.3.9, “List”](#).

## 12.3.7. Image

To add an image to an EB GUIDE project, copy the image file in the following directory: `$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 12.3.9, “List”](#).

## 12.3.8. Integer

EB GUIDE supports signed 32-bit integers.

Available operations are as follows:

- ▶ equal (==)
- ▶ not equal (!=)
- ▶ greater (>)

- ▶ greater or equal ( $\geq$ )
- ▶ less ( $<$ )
- ▶ less or equal ( $\leq$ )
- ▶ addition (+)
- ▶ subtraction (-)
- ▶ multiplication (\*)
- ▶ division (/)
- ▶ modulo (%)
- ▶ assign (writable properties) (=)

It is possible to store integer properties in a list. For details about lists, see [section 12.3.9, "List"](#).

## 12.3.9. List

EB GUIDE supports a list of values with the same data type.

The following list types are available:

- ▶ Mesh list
- ▶ Boolean list
- ▶ Color list
- ▶ Float list
- ▶ Font list
- ▶ Image list
- ▶ Integer 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: ([])

## 12.3.10. 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 12.3.9, “List”](#).

## 12.4. EB GUIDE Script

### 12.4.1. 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.

Keyword	Description
color:	A color parameter follows, for example {0,255,255}.
dp:	A datapool item follows.
l:	A language 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.

Keyword	Description
false	A boolean literal value
fire	Fires an event
if	A statement which tests a boolean expression follows. If the expression is true, the statement is executed.
in	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> .
function	Declares a function
length	The length of a property
let	Declares a local variable that is accessible in the scope
list	Declares a list type, for example an integer list
match_event	Checks if the current event corresponds to an expected event and declares variables like <code>let</code>
popup_stack	The dynamic state machine list which defines the priority of dynamic state machines
sm:	A state machine follows
true	A boolean literal value
unit	A value of type void
v:	A local variable follows.
view:	A view follows.
while	Repeats a statement as long as the condition is true

## 12.4.2. 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 12.3. EB GUIDE Script operator precedence

Operator	Associativity
(()), ({}), ({})	none
([])	none
(->)	left
(.)	none

Operator	Associativity
(::)	left
length	none
(&)	right
(!), (-) unary minus	right
(*), (/), (%)	left
(+), (-)	left
(<), (>), (<=), (>=)	left
(!=), (==), (=Aa=)	left
(&&)	left
(  )	left
(=), (+=), (-=), (=>)	right
(,)	right
(;)	left

### 12.4.3. EB GUIDE Script standard library

The following chapter provides a description of all EB GUIDE Script functions.

#### 12.4.3.1. EB GUIDE Script functions A

##### 12.4.3.1.1. abs

The function returns the absolute value of the integer number x.

Table 12.4. Parameters of `abs`

Parameter	Type	Description
x	integer	The number to return the absolute value from
<return>	integer	The return value

##### 12.4.3.1.2. absf

The function returns the absolute value of the float number x.

Table 12.5. 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

#### 12.4.3.1.3. `acosf`

The function returns the principal value of the arc cosine of `x`.

Table 12.6. 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

#### 12.4.3.1.4. `animation_before`

The function checks if a running animation has passed a given point in time.

Table 12.7. 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.

#### 12.4.3.1.5. `animation_beyond`

The function checks if a running animation has passed a given point in time.

Table 12.8. 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.



#### 12.4.3.1.6. `animation_cancel`

The function cancels an animation and leaves edited properties in the current state.

Table 12.9. 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.

#### 12.4.3.1.7. `animation_cancel_end`

The function cancels an animation and sets edited properties to the end state where possible.

Table 12.10. 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.

#### 12.4.3.1.8. `animation_cancel_reset`

The function cancels an animation and resets edited properties to the initial state where possible.

Table 12.11. 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.

#### 12.4.3.1.9. `animation_pause`

The function pauses an animation.

Table 12.12. 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.

#### 12.4.3.1.10. `animation_play`

The function starts or continues an animation.

Table 12.13. 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.

#### 12.4.3.1.11. `animation_reverse`

The function plays an animation backwards.

Table 12.14. 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.

#### 12.4.3.1.12. `animation_running`

The function checks if an animation is currently running.

Table 12.15. 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.

#### 12.4.3.1.13. `animation_set_time`

The function sets the current time of an animation, can be used to skip or replay an animation.

Table 12.16. 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



Parameter	Type	Description
<return>	boolean	If true, the function succeeded.

#### 12.4.3.1.14. `asinf`

The functions calculates the principal value of the arc sine of x.

Table 12.17. Parameters of `asinf`

Parameter	Type	Description
x	float	The number to return the arc sine from
<return>	float	The return value

#### 12.4.3.1.15. `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 12.18. Parameters of `atan2f`

Parameter	Type	Description
y	float	Argument y
x	float	Argument x
<return>	float	The return value

#### 12.4.3.1.16. `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 12.19. Parameters of `atan2i`

Parameter	Type	Description
y	integer	Argument y
x	integer	Argument x
<return>	float	The return value

#### 12.4.3.1.17. `atanf`

The function calculates the principal value of the arc tangent of x.

Table 12.20. 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

### 12.4.3.2. EB GUIDE Script functions C - H

#### 12.4.3.2.1. `ceil`

The function returns the smallest integral value that is not less than the argument.

Table 12.21. Parameters of `ceil`

Parameter	Type	Description
<code>value</code>	float	The value to round
<code>&lt;return&gt;</code>	integer	The rounded value

#### 12.4.3.2.2. `changeDynamicStateMachinePriority`

The function changes the priority of a dynamic state machine.

Table 12.22. Parameters of `changeDynamicStateMachinePriority`

Parameter	Type	Description
<code>state</code>		The state with the dynamic state machine list
<code>sm</code>	integer	The dynamic state machine
<code>priority</code>	integer	The priority of the dynamic state machine in the list

#### 12.4.3.2.3. `character2unicode`

The function returns the Unicode value of the first character in a string.

Table 12.23. Parameters of `character2unicode`

Parameter	Type	Description
<code>str</code>	string	The input string

Parameter	Type	Description
<return>	integer	The character as Unicode  0 in case of errors

#### 12.4.3.2.4. `clearAllDynamicStateMachines`

The function removes all dynamic state machines from the dynamic state machine list.

Table 12.24. Parameters of `clearAllDynamicStateMachines`

Parameter	Type	Description
state		The state with the dynamic state machine list

#### 12.4.3.2.5. `color2string`

The function converts a color to eight hexadecimal values.

Table 12.25. Parameters of `color2string`

Parameter	Type	Description
value	color	The color to convert to string
<return>	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".

#### 12.4.3.2.6. `cosf`

The function returns the cosine of x, where x is given in radians.

Table 12.26. Parameters of `cosf`

Parameter	Type	Description
x	float	The number to return the cosine from

Parameter	Type	Description
<return>	float	The return value

#### 12.4.3.2.7. deg2rad

The function converts an angle from degrees to radians.

Table 12.27. Parameters of `deg2rad`

Parameter	Type	Description
x	float	The angle to convert from degrees to radians
<return>	float	The return value

#### 12.4.3.2.8. expf

The function returns the value of e (the base of natural logarithms) raised to the power of x.

Table 12.28. Parameters of `expf`

Parameter	Type	Description
x	float	The exponent
<return>	float	The return value

#### 12.4.3.2.9. float2string

The function converts simple float to string.

Table 12.29. Parameters of `float2string`

Parameter	Type	Description
value	float	The value to convert to string
<return>	string	The float value, formatted as string

#### 12.4.3.2.10. floor

The function returns the largest integral value not greater than the parameter value.

Table 12.30. Parameters of `floor`

Parameter	Type	Description
value	float	The value to round

Parameter	Type	Description
<return>	integer	The rounded value

#### 12.4.3.2.11. focusNext

The function forces the focus manager to forward the focus to the next focusable element.

Table 12.31. Parameters of `focusNext`

Parameter	Type	Description
<return>	void	

#### 12.4.3.2.12. focusPrevious

The function forces the focus manager to return the focus to the previous focusable element.

Table 12.32. Parameters of `focusPrevious`

Parameter	Type	Description
<return>	void	

#### 12.4.3.2.13. format\_float

The function formats a float value.

Table 12.33. Parameters of `format_float`

Parameter	Type	Description
<code>format</code>	string	<p>A string of the following structure:</p> <p><code>%[flags] [width] [.precision] type</code></p> <ul style="list-style-type: none"> <li>▶ <code>flags</code>: Optional character or characters that control output justification and output of signs, blanks, leading zeros, decimal points, and octal and hexadecimal prefixes.</li> <li>▶ <code>width</code>: Optional decimal number that specifies the minimum number of characters that are output.</li> <li>▶ <code>precision</code>: 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>
<code>useDotAsDelimiter</code>	boolean	Defines the delimiter sign.  Possible values: <ul style="list-style-type: none"> <li>▶ true: Use a dot as delimiter.</li> <li>▶ false: Use a comma as delimiter.</li> </ul>
<code>value</code>	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.

#### 12.4.3.2.14. `format_int`

The function formats an integer value.

Table 12.34. Parameters of `format_int`

Parameter	Type	Description
<code>format</code>	string	A string of the following structure:  <code>%[flags] [width] [.precision] type</code> <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>



Parameter	Type	Description
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.

#### 12.4.3.2.15. `getLineCount`

The function returns the number of lines for a widget.

Table 12.35. Parameters of `getLineCount`

Parameter	Type	Description
widget	widget	The widget to evaluate
<return>	integer	The number of lines

#### 12.4.3.2.16. `getTextHeight`

The function returns the height of a text with regard to its font resource.

Table 12.36. 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

#### 12.4.3.2.17. `getTextLength`

The function returns the number of characters in a text.

Table 12.37. Parameters of `getTextLength`

Parameter	Type	Description
text	string	The text to evaluate



Parameter	Type	Description
<return>	integer	The number of characters in the text

#### 12.4.3.2.18. `getTextWidth`

The function returns the width of a text with regard to its font resource.

Table 12.38. Parameters of `getTextWidth`

Parameter	Type	Description
<code>text</code>	string	The text to evaluate
<code>font</code>	font	The font to evaluate
<return>	integer	The width of the text

#### 12.4.3.2.19. `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 12.39. Parameters of `has_list_window`

Parameter	Type	Description
<code>itemId</code>	<code>dp_id</code>	The ID of the datapool item of type list
<code>index</code>	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.

#### 12.4.3.2.20. `hsba2color`

The function converts an HSB/HSV color to a GTF color.

Table 12.40. Parameters of `hsba2color`

Parameter	Type	Description
<code>hue</code>	integer	The color value in degrees from 0 to 360
<code>saturation</code>	integer	The saturation in percent
<code>brightness</code>	integer	The brightness in percent
<code>alpha</code>	integer	The alpha value between 0 (totally transparent) and 255 (opaque)



Parameter	Type	Description
<return>	color	The resulting GTF color with the alpha value applied

### 12.4.3.3. EB GUIDE Script functions I - R

#### 12.4.3.3.1. int2float

The function returns the integer value converted to a float point value.

Table 12.41. Parameters of `int2float`

Parameter	Type	Description
value	integer	The value to convert to float
<return>	float	The integer value, converted to float

#### 12.4.3.3.2. int2string

The function converts a simple integer to string.

Table 12.42. 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

#### 12.4.3.3.3. isDynamicStateMachineActive

The function checks if the state with the dynamic state machine list is active.

Table 12.43. Parameters of `isDynamicStateMachineActive`

Parameter	Type	Description
state		The state with the dynamic state machine list
sm	integer	The dynamic state machine

#### 12.4.3.3.4. language

The function switches the language of all datapool items.

Table 12.44. Parameters of `language`

Parameter	Type	Description
<code>language</code>	<code>languageType</code>	The language to switch to, for example <code>f:language(1:German)</code>
<code>&lt;return&gt;</code>	<code>void</code>	

#### 12.4.3.3.5. `localtime_day`

The function extracts the day [1:31] in local time from a system time value.

Table 12.45. Parameters of `localtime_day`

Parameter	Type	Description
<code>time</code>	<code>integer</code>	A time stamp as returned by system time
<code>&lt;return&gt;</code>	<code>integer</code>	The extracted day

#### 12.4.3.3.6. `localtime_hour`

The function extracts the hours from the local time of a system time value.

Table 12.46. Parameters of `localtime_hour`

Parameter	Type	Description
<code>time</code>	<code>integer</code>	A time stamp as returned by system time
<code>&lt;return&gt;</code>	<code>integer</code>	The extracted hour

#### 12.4.3.3.7. `localtime_minute`

The function extracts the minutes from the local time of a system time value.

Table 12.47. Parameters of `localtime_minute`

Parameter	Type	Description
<code>time</code>	<code>integer</code>	A time stamp as returned by system time
<code>&lt;return&gt;</code>	<code>integer</code>	The extracted minute

#### 12.4.3.3.8. `localtime_month`

The function extracts the month [0:11] from the local time of a system time value.

Table 12.48. 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

#### 12.4.3.3.9. `localtime_second`

The function extracts the seconds from the local time of a system time value.

Table 12.49. 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

#### 12.4.3.3.10. `localtime_weekday`

The function extracts the week day [0:6] from the local time of a system time value. 0 is Sunday.

Table 12.50. 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

#### 12.4.3.3.11. `localtime_year`

The function extracts the year from the local time of a system time value.

Table 12.51. 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

#### 12.4.3.3.12. `log10f`

The function returns the base 10 logarithm of x.

Table 12.52. Parameters of `log10f`

Parameter	Type	Description
<code>x</code>	float	The argument
<code>&lt;return&gt;</code>	float	The return value

#### 12.4.3.3.13. `logf`

The function returns the natural logarithm of `x`.

Table 12.53. Parameters of `logf`

Parameter	Type	Description
<code>x</code>	float	The argument
<code>&lt;return&gt;</code>	float	The return value

#### 12.4.3.3.14. `nearbyint`

The function rounds to nearest integer.

Table 12.54. Parameters of `nearbyint`

Parameter	Type	Description
<code>value</code>	float	The value to round
<code>&lt;return&gt;</code>	integer	The rounded value

#### 12.4.3.3.15. `popDynamicStateMachine`

The function removes the dynamic state machine on the top of the priority queue.

Table 12.55. Parameters of `popDynamicStateMachine`

Parameter	Type	Description
<code>state</code>		The state with the dynamic state machine list
<code>sm</code>	integer	The dynamic state machine

#### 12.4.3.3.16. `powf`

The function returns the value of `x` raised to the power of `y`.

Table 12.56. 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

#### 12.4.3.3.17. `pushDynamicStateMachine`

The function inserts the dynamic state machine in a priority queue.

Table 12.57. Parameters of `pushDynamicStateMachine`

Parameter	Type	Description
<code>state</code>		The state with the dynamic state machine list
<code>sm</code>	integer	The dynamic state machine
<code>priority</code>	integer	The priority of the dynamic state machine in the list

#### 12.4.3.3.18. `rad2deg`

The function converts an angle form radians to degree.

Table 12.58. Parameters of `rad2deg`

Parameter	Type	Description
<code>x</code>	float	The argument
<code>&lt;return&gt;</code>	float	The return value

#### 12.4.3.3.19. `rand`

The function gets a random value between 0 and  $2^{31}-1$ .

Table 12.59. Parameters of `rand`

Parameter	Type	Description
<code>&lt;return&gt;</code>	integer	A random number between 0 and $2^{31}-1$

#### 12.4.3.3.20. `request_runlevel`

The function requests the framework to switch to a different run level. The only supported run level is 0, meaning to shutdown the program.

Table 12.60. Parameters of `request_runlevel`

Parameter	Type	Description
<code>runlevel</code>	integer	The requested run level
<return>	void	

#### 12.4.3.3.21. `rgba2color`

The function converts from RGB color space to GTF color.

Table 12.61. 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)
<return>	color	The color converted from RGB color space to GTF color, with the alpha value applied

#### 12.4.3.3.22. `round`

The function rounds to nearest integer, but rounds halfway cases away from zero.

Table 12.62. Parameters of `round`

Parameter	Type	Description
<code>value</code>	float	The value to round
<return>	integer	The rounded value

### 12.4.3.4. EB GUIDE Script functions S - W

#### 12.4.3.4.1. `seed_rand`

The function sets the seed of the random number generator.



Table 12.63. Parameters of `seed_rand`

Parameter	Type	Description
<code>seed</code>	integer	The value to seed the random number generator
<code>&lt;return&gt;</code>	void	

#### 12.4.3.4.2. `sinf`

The function returns the sine of `x`, where `x` is given in radians.

Table 12.64. Parameters of `sinf`

Parameter	Type	Description
<code>x</code>	float	The argument
<code>&lt;return&gt;</code>	float	The return value

#### 12.4.3.4.3. `sqrtf`

The function returns the non-negative square root of `x`.

Table 12.65. Parameters of `sqrtf`

Parameter	Type	Description
<code>x</code>	float	The argument
<code>&lt;return&gt;</code>	float	The return value

#### 12.4.3.4.4. `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. An optional leading white space
2. An optional plus ('+') or minus ('-') sign
3. One of the following:
  - ▶ A decimal number
  - ▶ A hexadecimal number
  - ▶ An infinity
  - ▶ An NAN (not-a-number)

Table 12.66. Parameters of `string2float`

Parameter	Type	Description
<code>str</code>	string	The string value
<code>&lt;return&gt;</code>	float	The return value

#### 12.4.3.4.5. `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 12.67. Parameters of `string2int`

Parameter	Type	Description
<code>str</code>	string	The string value
<code>&lt;return&gt;</code>	integer	The return value

#### 12.4.3.4.6. `string2string`

The function formats strings.

Table 12.68. Parameters of `string2string`

Parameter	Type	Description
<code>str</code>	string	The string to format
<code>len</code>	integer	The maximum length of the string
<code>&lt;return&gt;</code>	string	The language string

#### 12.4.3.4.7. `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".

Table 12.69. Parameters of `substring`

Parameter	Type	Description
<code>str</code>	string	The input string



Parameter	Type	Description
<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
<return>	string	The language string

#### 12.4.3.4.8. `system_time`

The function gets the current system time in seconds. The result is intended to be passed to the `localtime_*` functions.

Table 12.70. Parameters of `system_time`

Parameter	Type	Description
<return>	integer	The system time in seconds

#### 12.4.3.4.9. `system_time_ms`

The function gets the current system time in milliseconds.

Table 12.71. Parameters of `system_time_ms`

Parameter	Type	Description
<return>	integer	The system time in milliseconds

#### 12.4.3.4.10. `tanf`

The function returns the tangent of x, where x is given in radians.

Table 12.72. Parameters of `tanf`

Parameter	Type	Description
<code>x</code>	float	The argument
<return>	float	The return value

#### 12.4.3.4.11. `trace_dp`

The function writes debugging information about a datapool item to the trace log and the connection log.

Table 12.73. 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>	

#### 12.4.3.4.12. `trace_string`

The function writes a string to the trace log and the connection log.

Table 12.74. 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>	

#### 12.4.3.4.13. `transformToScreenX`

The function takes a widget and a local coordinate and returns x-position in the screen-relative world coordinate system.

Table 12.75. 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

#### 12.4.3.4.14. `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 12.76. 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

#### 12.4.3.4.15. 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 12.77. Parameters of `transformToWidgetX`

Parameter	Type	Description
widget	widget	The widget to which the coordinates are relative
screenX	integer	The x-position of the screen coordinate
screenY	integer	The y-position of the screen coordinate
<return>	integer	The x-position of the local coordinate

#### 12.4.3.4.16. 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 12.78. Parameters of `transformToWidgetY`

Parameter	Type	Description
widget	widget	The widget to which the coordinates are relative
screenX	integer	The x-position of the screen coordinate
screenY	integer	The y-position of the screen coordinate
<return>	integer	The y-position of the local coordinate

#### 12.4.3.4.17. trunc

The function rounds to the nearest integer value, always towards zero.

Table 12.79. Parameters of `trunc`

Parameter	Type	Description
value	float	The value to round
<return>	integer	The rounded value



#### 12.4.3.4.18. widgetGetChildCount

The function obtains the number of child widgets of the given widget.

Table 12.80. 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

## 12.5. Events

Table 12.81. 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.

## 12.6. gtfStartup.cfg configuration file

### 12.6.1. Mapping rule structure

A mapping rule in EB GUIDE TF consists of one signal and one action. Both signals and actions are made up of tokens.

Tokens are separated by spaces or tabulators. Everything between a double slash (//) and the end of line is ignored. Empty lines are ignored. Text within double quotes (") is parsed as one token even if it includes spaces, tabulators, or comments. Decimal and hexadecimal number format is supported.

Table 12.82. Escape sequences used to enter special characters

Escape sequence	Special character
\n	line feed

Escape sequence	Special character
\r	carriage return
\\	\
\"	"
\t	TAB

## 12.6.2. Signals

The first token of a signal is a keyword which defines the signal type. The token after that is a type specific parameter.

Table 12.83. Signals

Keyword	Parameter	Description
INIT	none	Loading <code>gtfStartup.cfg</code> is finished.
STARTUP	<run level>	<run level> (0...0xFFFF) is reached during start-up.

## 12.6.3. Actions

The first token of an action is a keyword which defines the action type. The tokens after that are type specific parameters.

Table 12.84. Actions

Keyword	First parameter	Following parameters	Description
MESSAGE	<MsgID>	message parameters	Sends a message with the <MsgID> (0...-0xFFFFFFFF) and message parameters specified.
LOAD (supported by INIT signal only)	FW_PATH or MODEL_PATH	<file path>	Loads an EB GUIDE GTF extension file and initializes the included modules. FW_PATH means a path relative to the <code>gtfStartup.exe</code> executable file. MODEL_PATH

Keyword	First parameter	Following parameters	Description
			means a path relative to the <code>gtfStartup.cfg</code> file. For absolute paths the key words <code>FW_PATH</code> and <code>MODEL_PATH</code> lead to the same result.
<code>LOAD_ALL</code>  (supported by INIT signal only)	<code>FW_PATH</code> or  <code>MODEL_PATH</code>	<directory path>	Loads all EB GUIDE GTF extension files in the directory and initializes the included modules. <code>FW_PATH</code> means a path relative to the <code>GtfStartup.exe</code> executable file. <code>MODEL_PATH</code> means a path relative to the <code>gtfStartup.cfg</code> file. For absolute paths the key words <code>FW_PATH</code> and <code>MODEL_PATH</code> lead to the same result.

Message parameters consist of a keyword token followed by a value token.

Table 12.85. Message parameters

Keyword	Value	Description
<code>UINT8</code>	number	8-bit unsigned integer
<code>UINT16</code>	number	16-bit unsigned integer
<code>UINT32</code>	number	32-bit unsigned integer
<code>INT8</code>	number	8-bit signed integer
<code>INT16</code>	number	16-bit signed integer
<code>INT32</code>	number	32-bit signed integer
<code>STRBUF</code>	string	Pointer to a buffer storing the string, available until shutdown is completed
<code>STRING</code>	string	String
<code>FW_PATH</code>	string	Same as <code>STRBUF</code> , but the string is interpreted as a path relative to the <code>GtfStartup.exe</code> executable file.



Keyword	Value	Description
MODEL_PATH	string	Same as STRBUF, but the given string is interpreted as a path relative to the <code>gtfStartup.cfg</code> file.

The EB GUIDE TF message system is used for run level and interface management as well as for configuration of framework modules, for example EB GUIDE TF modules or applications provided by the customer. Message IDs and parameters of pre-defined messages are documented in the [GtfMessageId.h](#) file.

**TIP**



**Working with message IDs**

Message IDs are organized in message groups. That means, searching message ID 401 in the [GtfMessageId.h](#) file will not lead to any result. Instead, search the following line:

```
#define GTF_MID_RANGE_GTF_DISPLAY 400
```

All display-related messages are relative to ID 400. Searching the string `GTF_MID_RANGE_GTF_DISPLAY` will lead to the following entry for message ID 401:

```
#define GTF_MID_GTF_DISPLAY_CONNECT  
(uint32_t) (GTF_MID_RANGE_GTF_DISPLAY + 1)
```

**NOTE**



**Predefined messages in EB GUIDE TF**

Message ID range 0...0xFFFF is reserved for EB GUIDE TF and the EB GUIDE product line.

Message ID range 0x10000...0xFFFFFFFF can be managed by you.

## 12.6.4. Messages

The following messages use the example `GtfCoreModel` with the ID `0xDEADBEEF`.

- ▶ Configure the type manager

Run level	Message ID	Parameters
0	317	MODEL_PATH "types.bin"

This message (`GTF_MID_GTF_TYPEMANAGER_CONFIG`) specifies the location of the `types.bin` binary file.

- ▶ Create a `GtfCoreRuntime`

Run level	Message ID	Parameters
0	306	UINT32 0xDEADBEEF UINT8 0

This message (`GTF_MID_GTF_CORE_CREATE_CORE_HOOK_ATF_WORKLOOP`) creates the `GtfCore-  
Runtime` in the thread that is identified by the communication context ID, which is 0 in the example. See [GtfMessageId.h](#) for variations with different communication context IDs.

**NOTE**



**Context ID**

The communication context ID of a state machine can be seen and configured in EB GUIDE Studio. By default, the communication context ID is 0.

- ▶ Create a `GtfViewFactory`

Run level	Message ID	Parameters
403	450	UINT32 0xBAADF00D MODEL_PATH "views.- bin"

This message (`GTF_MID_GTF_VIEWFACTORY_BINARY`) creates a `GtfViewFactory` which loads view descriptions from the `views.bin` binary file and defines the unique ID of the `GtfViewFactory` to be `0xBAADF00D`. The keyword `MODEL_PATH` makes the file path relative to the `gtfStartup.cfg` file.

- ▶ Configure the `.bdf` input file

Run level	Message ID	Parameters
499	308	UINT32 0xDEADBEAF MODEL_PATH "dat- apool.bdf"

This message (`GTF_MID_GTF_DATAPOOL_DECLARATIONS`) makes the `GtfCoreModel` with the ID `0xDEADBEAF` load the specified `.bdf` file.

- ▶ Configure the `.bvf` input file

Run level	Message ID	Parameters
499	309	UINT32 0xDEADBEAF MODEL_PATH "dat- apool.bvf"

This message (`GTF_MID_GTF_DATAPOOL_VALUES`) makes the `GtfCoreModel` with the ID `0xDEAD-  
BEAF` load the specified `.bvf` file.

- ▶ Configure the state machine file

Run level	Message ID	Parameters
499	311	UINT32 0xDEADBEAF MODEL_PATH "mod- el.bin"

This message (`GTF_MID_GTF_STATE_MACHINE_MODEL`) makes the `GtfCoreModel` with the ID `0xDEADBEEF` load the specified binary state machine file `model.bin`.

- ▶ Enable state machines

Run level	Message ID	Parameters
501	320	UINT32 0xDEADBEEF STRBUF "Main"

This message (`GTF_MID_GTF_ENABLE_STATE_MACHINE`) enables the state machine called `Main` in the `GtfCoreModel` with the ID `0xDEADBEEF`.

- ▶ Configure displays

Run level	Message ID	Parameters
0	511	STRBUF "Main" STRBUF "windowCaption" STRBUF "My Model"
0	512	STRBUF "Main" STRBUF "hwLayerId" INT32 0
0	513	STRBUF "Main" STRBUF "showWin- dowFrame" UINT8 1

These messages (`GTF_MID_GTF_DISPLAY_CONFIG_STRING`, `GTF_MID_GTF_DISPLAY_CONFIG_INT` and `GTF_MID_GTF_DISPLAY_CONFIG_BOOL`) apply to the display that belongs to the state machine called `Main`. Message 511 is used for string options, message 512 for integer options and message 513 for boolean options.

- ▶ Configure resource configuration file

Run level	Message ID	Parameters
499	312	UINT32 0xDEADBEEF MODEL_PATH "re- sources.bin"

This message (`GTF_MID_GTF_RESOURCE_MODEL`) makes the `GtfCoreModel` with the ID `0xDEADBEEF` load the binary resource configuration file `resources.bin`.

- ▶ Configure the debug database (optional)

Run level	Message ID	Parameters
499	318	UINT32 0xDEADBEEF MODEL_PATH "de- bug.bin"

This message (`GTF_MID_GTF_DEBUGDATABASE_CONFIG`) includes the debug database file `debug.bin`.

- ▶ Configure the service mapper TCP/IP port

Run level	Message ID	Parameters
0	305	UINT16 5456

This message (`GTF_MID_GTF_SERVICE_MAPPER`) makes the debugger service of EB GUIDE TF listen to TCP/IP port 5456.

- ▶ Load a RomFS container

Run level	Message ID	Parameters
0	701	MODEL_PATH "container.romfs"

This message (`GTF_MID_GTF_FILESYSTEM_LOAD_ROMFS`) loads the RomFS container specified by `container.romfs` into EB GUIDE TF.

- ▶ Configure how font files are accessed by EB GUIDE TF (optional)

Run level	Message ID	Parameters
0	510	UINT8 1

This message (`GTF_MID_GTF_FREETYPE_STREAM_TYPE`) configures the font access component of EB GUIDE GTF. If the `UINT8` parameter value is 0, it uses a ROM-mapped file. If the `UINT8` parameter value is 1, it uses a plain file I/O.

**NOTE**



**ROM-mapped file approach vs. plain file I/O approach**

The ROM-mapped file approach in general provides higher performance. But on some systems, for example QNX, it consumes more memory than the plain file I/O approach. Plain file I/O in general consumes less memory than the ROM-mapped file approach. But it can lead to lower performance.

- ▶ Disable the output of EB GUIDE Script trace functions

Run level	Message ID	Parameters
0	321	UINT32 0xDEADBEEF

This message suppresses the output of `f:trace_string()` and `f:trace_dp()` in EB GUIDE Script code.

- ▶ Debug monitoring information

To display monitoring information during run-time, some renderers need additional resources independent of the EB GUIDE model. Such resources are located in the `monitoring` directory inside the EB GUIDE GTF run-time directory. If your start-up configuration does not configure monitoring displays such as frames per second (FPS), you can safely remove these resources. To enable FPS monitoring, set the appropriate bit in the operating mode value of the renderer. For details see EB GUIDE GTF API documentation.



► Configure FreeType Cache (optional)

Run level	Message ID	Parameters
0	515	UINT32 1000 UINT32 2000 UINT32 4000

This message (`GTF_MID_GTF_FREETYPE_CACHE_CONFIGURATION`) sets the FreeType cache parameters as described at [http://www.freetype.org/freetype2/docs/reference/ft2-cache\\_subsystem.html](http://www.freetype.org/freetype2/docs/reference/ft2-cache_subsystem.html). The default values are as follows:

- `max_faces`: 1000
- `max_sizes`: 2000
- `max_bytes`: 4000 kB

Due to the way EB GUIDE GTF handles font sizes, `ft_size` objects are not cached separately from `ft_face` objects at the moment. It is recommended to use meaningful values for `max_sizes`.

► Configure the resource cache

Run level	Message ID	Parameters
0	520	UINT32 61441 UINT32 1048576

This message (`GTF_MID_GTF_RENDERER_TEXTURE_CACHE`) creates a resource cache for the default display ID (61441) with a size of 1 048 576 bytes. Add the message several times to configure more display IDs. Assigning display ID 0 configures all display IDs which are not configured otherwise to use the same resource cache.

## 12.7. Scenes

Table 12.86. Properties of a scene

Property name	Description
<code>height</code>	The height of the area in which the views of a haptic state machine are rendered on a target device
<code>width</code>	The width of the area in which the views of a haptic state machine are rendered on a target device
<code>x</code>	The x-offset of the area in which the views of a haptic state machine are rendered on a target device
<code>y</code>	The y-offset of the area in which the views of a haptic state machine are rendered on a target device
<code>visible</code>	If true, the state machine and its child widgets are visible.
<code>projectName</code>	The name of the EB GUIDE project



Property name	Description
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	Possible values:  ▶ 1: 32 bit ▶ 2: 16 bit
multisampling	Possible values:  ▶ Off (= 0): no multisampling ▶ 2x (=1): 2x multisampling ▶ 4x (=2): 4x multisampling
enableRemoteFramebuffer	If true, transfer of the off-screen buffer to the simulation window is enabled
showWindowFrame	If true, a frame is displayed on the simulation window. The frame allows the window to be grabbed and moved.
showWindow	If true, an additional window for simulation is opened on Windows based systems.
disableVSync	If true, vertical synchronization for the renderer is disabled.
showFPS	Possible values:  ▶ 0: Do not show FPS ▶ 1: Show FPS on the screen ▶ 2: Show FPS on the console ▶ 3: Show FPS on the screen and on the console
Renderer	Defines a renderer for the scene.  Possible values:  ▶ DirectX ▶ OpenGL

**TIP**



**Settings for multisampling**

The higher the resolution for multisampling is the better the quality of the rendering result. However, be aware that multisampling decreases the rendering performance, especially on a target device. At small displays with high resolution the multisampling has almost no effect.

Start with no multisampling and, if the performance is good, try the settings 2x or 4x multisampling. If there is no big difference with higher multisampling, use a lower setting.

## 12.8. Touch screen types supported by EB GUIDE GTF

The supported types depend on the target device.

Table 12.87. Touch screen types supported by EB GUIDE GTF

Value	Description	Platform
0	Galaxy	Linux
1	IMX WVGA	Linux
2	Touch screen connected to mouse interface	All
3	General platform-dependent touch-screen interface	All
4	Lilliput 889GL	QNX
5	General platform-dependent multitouch touch-screen interface	Linux

## 12.9. Widgets

### 12.9.1. View

Table 12.88. Properties of a view

Property name	Description
name	The name of the 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
y	The y-coordinate of the widget

View templates have additional properties for view transition animations. An entry animation is executed when the view is entered.

Table 12.89. Properties of an entry animation

Property name	Description
Entry animation	If true, instances of the view template have an entry animation.
Type	The type of the entry animation, for example <b>Move in from left</b> , <b>Fade in from center</b> or <b>Show view immediately</b> .
Duration	The duration of the entry animation in milliseconds
Delay	The delay of the entry animation in milliseconds
Play after exit animation	If true, the start time of the entry animation depends on the duration of a previous exit animation.

An exit animation is executed when the view is exited.

Table 12.90. Properties of an exit animation

Property name	Description
Exit animation	If true, instances of the view template have an exit animation.
Type	The type of the exit animation, for example <b>Move out to top</b> , <b>Fade out to center</b> or <b>Hide view immediately</b> .
Duration	The duration of the exit animation in milliseconds
Delay	The delay of the exit animation in milliseconds

## 12.9.2. Basic widgets

There are five basic widgets.

- ▶ Label
- ▶ Image



- ▶ Rectangle
- ▶ Container
- ▶ Instantiator

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.

### 12.9.2.1. Label

A label places text into a view.

Table 12.91. Properties of the label

Property name	Description
<code>name</code>	The name of the widget
<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

### 12.9.2.2. Rectangle

A rectangle draws a colored rectangle with the dimensions and coordinates of the widget into a view.

Table 12.92. Properties of the rectangle

Property name	Description
name	The name of the widget
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 rectangle

### 12.9.2.3. Image

An image places a picture into a view.

Table 12.93. Properties of the image

Property name	Description
name	The name of the widget
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
image	The image the widget displays
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. DirectX 11 and OpenGL ES 2.0 support `.png` files and `.jpeg` files.

### 12.9.2.4. Container

A container holds several widgets as child widgets and thus groups the widgets.

Table 12.94. Properties of the container

Property name	Description
name	The name of the widget
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

### 12.9.2.5. Instantiator

An instantiator creates widget instances during run-time. You can use the instantiator to model lists or tables. The child widgets of an instantiator serve as line templates for the list or table which is created during run-time.

Table 12.95. Properties of the instantiator

Property name	Description
name	The name of the widget
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 elements
lineMapping	Defines which child is the template for which line

## 12.9.3. Animations

The following sections list the properties of the widgets in the **Animations** category.

### 12.9.3.1. Animation

An animation influences its parent widget. An animation requires at least one curve as a child widget.

Table 12.96. Properties of the animation

Property name	Description
name	The name of the animation
alternating	Defines if the animation is executed repeatedly
repeat	The number of repetitions, 0 for infinite number
enabled	Defines if the animation is executed
scale	The factor by which the animation time is multiplied
onPause	The reaction that is executed when the animation is paused. Parameter: Current animation time.
onPlay	The reaction that is executed when the animation is started or continued. Parameters: Start time and play direction (true for forwards, false for backwards)
onTerminate	The reaction that is executed when the animation completes. First parameter: Animation time. Second parameter: Reason for the termination, encoded as follows: <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>

### 12.9.3.2. Constant curves

A constant curve sets a target value after a defined delay. Constant curves are available for integer, boolean, float, and color types.

Table 12.97. Properties of constant curves

Property name	Description
name	The name of the curve
delay	The delay in ms relative to the animation start
duration	The duration of the curve segment in ms
enabled	Defines if the animation is executed
alternating	Defines if the animation is executed repeatedly



Property name	Description
relative	Defines if update values are applied on the initial value
repeat	The number of repetitions
target	The target property the resulting value is assigned to
value	The resulting constant value

### 12.9.3.3. Fast start curves

A fast start curve periodically sets a value that increases fast in the beginning but loses speed constantly until the end. Fast start curves are available for integer, float, and color types.

Table 12.98. Properties of fast start curves

Property name	Description
name	The name of the curve
delay	The delay in ms relative to the animation start
duration	The duration of the curve segment in ms
enabled	Defines if the animation is executed
alternating	Defines if the animation is executed repeatedly
relative	Defines if update values are applied on the initial value
repeat	The number of repetitions
target	The target property the resulting value is assigned to
startt	The initial value
end	The final value

### 12.9.3.4. Slow start curves

A slow start curve periodically sets a value that increases slowly in the beginning but rises constantly until the end. Slow start curves are available for integer, float, and color types.

Table 12.99. Properties of slow start curves

Property name	Description
name	The name of the curve
delay	The delay in ms relative to the animation start



Property name	Description
duration	The duration of the curve segment in ms
enabled	Defines if the animation is executed
alternating	Defines if the animation is executed repeatedly
relative	Defines if update values are applied on the initial value
repeat	The number of repetitions
target	The target property the resulting value is assigned to
start	The initial value
end	The final value

### 12.9.3.5. Quadratic curves

A quadratic curve periodically sets a value using a quadratic function curve. Quadratic curves are available for integer, float, and color types.

Table 12.100. Properties of quadratic curves

Property name	Description
name	The name of the curve
delay	The delay in ms relative to the animation start
duration	The duration of the curve segment in ms
enabled	Defines if the animation is executed
alternating	Defines if the animation is executed repeatedly
relative	Defines if update values are applied on the initial value
repeat	The number of repetitions
target	The target property the resulting value is assigned to
velocity	The velocity to calculate the result
acceleration	The acceleration of the curve
constant	The constant value to calculate the result

### 12.9.3.6. Sinus curves

A sinus curve periodically sets a value using a sinus function curve. Sinus curves are available for integer, float, and color types.

Table 12.101. Properties of sinus curves

Property name	Description
name	The name of the curve
delay	The delay in ms relative to the animation start
duration	The duration of the curve segment in ms
enabled	Defines if the animation is executed
alternating	Defines if the animation is executed repeatedly
relative	Defines if update values are applied on the initial value
repeat	The number of repetitions
target	The target property the resulting value is assigned to
amplitude	The amplitude of the sinus curve
constant	The constant value to calculate the result
phase	The angular phase translation in radians
frequency	The frequency of the curve in hertz

### 12.9.3.7. Script curves

A script curve sets a value using a curve that is described by EB GUIDE Script. Script curves are available for integer, boolean, float, and color types.

Table 12.102. Properties of script curves

Property name	Description
name	The name of the curve
delay	The delay in ms relative to the animation start
duration	The duration of the curve segment in ms
enabled	Defines if the animation is executed
alternating	Defines if the animation is executed repeatedly
relative	Defines if update values are applied on the initial value
repeat	The number of repetitions
target	The target property the resulting value is assigned to
curve	The resulting curve function

### 12.9.3.8. Linear curves

A linear curve periodically sets a value using a linear progression curve. Linear curves are available for integer, float, and color types.

Table 12.103. Properties of linear curves

Property name	Description
name	The name of the curve
delay	The delay in ms relative to the animation start
duration	The duration of the curve segment in ms
enabled	Defines if the animation is executed
alternating	Defines if the animation is executed repeatedly
relative	Defines if update values are applied on the initial value
repeat	The number of repetitions
target	The target property the resulting value is assigned to
velocity	The velocity to calculate the result

### 12.9.3.9. Linear interpolation curves

A linear interpolation curve periodically sets a value using a linear interpolation curve. Linear interpolation curves are available for integer, float, and color types.

Table 12.104. Properties of linear interpolation curves

Property name	Description
name	The name of the curve
delay	The delay in ms relative to the animation start
duration	The duration of the curve segment in ms
enabled	Defines if the animation is executed
alternating	Defines if the animation is executed repeatedly
relative	Defines if update values are applied on the initial value
repeat	The number of repetitions
target	The target property the resulting value is assigned to
start	The initial value
end	The final value

### 12.9.4. 3D widgets



### 12.9.4.1. Scene graph

A scene graph places a 3D object into a view.

Table 12.105. Properties of the scene graph

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

### 12.9.4.2. 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. The following 3D widgets can be added to the scene graph node:

- ▶ Camera
- ▶ Directional light
- ▶ Mesh
- ▶ Point light
- ▶ Spot light

Table 12.106. Properties of the scene graph node

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

Property name	Description
translationZ	The translation along the z-axis

### 12.9.4.3. 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 12.107. Properties of the camera

Property name	Description
enabled	If true, the widget is enabled
nearPlane	The nearest distance from the camera in view direction at which the scene becomes visible
farPlane	The farthest distance from the camera in view direction up to which the scene is visible
fieldOfView	The camera's vertical viewing angle in degrees

### 12.9.4.4. Directional light

A directional light adds a light that illuminates the scene from one direction.

Table 12.108. Properties of the directional light

Property name	Description
enabled	If true, the widget is enabled
color	The light's color
intensity	The light's intensity

### 12.9.4.5. Material

A material defines the visual appearance of the surface of the mesh.

Table 12.109. Properties of the material

Property name	Description
ambient	The color the object reflects when it is illuminated by ambient light
diffuse	The color the object reflects evenly in all directions when it is illuminated by pure white light

Property name	Description
emissive	The self-illumination color of the object
shininess	The shininess factor
specular	The color an object with a shiny surface reflects
opacity	The opacity value  Note that only values between 0 and 1, as for example 0.3, are valid.

### 12.9.4.6. Mesh

A mesh defines the shape of the 3D object.

Table 12.110. Properties of the mesh

Property name	Description
visible	If true, the widget and its child widgets are visible
mesh	The automatically created mesh file *.ebmesh
culling	Defines whether no triangles (0), only front facing triangles (1), or only back facing triangles (2) are culled from the mesh

### 12.9.4.7. Point light

A point light adds a light to the scene that emits light in all directions like a light bulb.

Table 12.111. Properties of the point light

Property name	Description
enabled	If true, the widget is enabled
color	The light's color
intensity	The light's intensity
attenuationConstant	The constant factor by which the light weakens with increasing distance
attenuationLinear	The linear factor by which the light weakens with increasing distance
attenuationQuadratic	The quadratic factor by which the light weakens with increasing distance

### 12.9.4.8. Spot light

A spot light adds a light which restricts illumination to a cone of influence.

Table 12.112. Properties of the spot light

Property name	Description
enabled	If true, the widget is enabled
color	The light's color
intensity	The light's intensity
attenuationConstant	The constant factor by which the light weakens with increasing distance
attenuationLinear	The linear factor by which the light weakens with increasing distance
attenuationQuadratic	The quadratic factor by which the light weakens with increasing distance
coneAngleInner	The light's inner cone angle
coneAngleOuter	The light's outer cone angle

## 12.10. 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.

### 12.10.1. Common

#### 12.10.1.1. Child visibility selection

The **Child visibility selection** widget feature handles the visibility of child widgets. Only the content of one child widget is visible at a time.

Table 12.113. Properties of the **Child visibility selection** widget feature

Property name	Description
containerIndex	The index of the child widgets of the parent widget
containerMapping	If a mapping is set, each child of the container is re-addressed by its appropriate value in <code>containerMapping</code> .  If a mapping is not set, undefined, or if the length does not match the number of child widgets in the container, the mapping is not used. Instead, the order of

Property name	Description
	widgets in the widget tree is used as their index. The topmost child has index 0, next index 1 etc.

### 12.10.1.2. Enabled

The **Enabled** widget feature adds an `enabled` property to a widget.

Table 12.114. Properties of the **Enabled** widget feature

Property name	Description
<code>enabled</code>	If true, the widget reacts on touch and press input

### 12.10.1.3. Focused

The **Focused** widget feature enables a widget to have input focus.

Table 12.115. Properties of the **Focused** widget feature

Property name	Description
<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>
<code>focused</code>	If true, the widget has focus

### 12.10.1.4. Pressed

The **Pressed** widget feature defines that a widget can be pressed.

Table 12.116. Properties of the **Pressed** widget feature

Property name	Description
<code>pressed</code>	If true, a key is pressed while the widget is focused

Combining the **Touched** widget feature with the **Touch pressed** widget feature allows modeling a push button.

### 12.10.1.5. 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 12.117. Properties of the **Selected** widget feature

Property name	Description
<code>selected</code>	If true, the widget is selected

### 12.10.1.6. Selection group

The **Selection group** widget feature is used to model an array of radio buttons. In an array, 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 external 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.

Table 12.118. Properties of the **Selection group** widget feature

Property name	Description
<code>buttonId</code>	The ID that identifies a button within a button group
<code>buttonValue</code>	The current value of a button. If this value matches the <code>buttonId</code> , the button is selected.
<code>selected</code>	Evaluates if <code>buttonID</code> and <code>buttonValue</code> are identical. If true, the button is selected.

### 12.10.1.7. 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 12.119. Properties of the **Spinning** widget feature

Property name	Description
<code>currentValue</code>	The current rotary value
<code>maxValue</code>	The maximum value for the <code>currentValue</code> property
<code>minValue</code>	The minimum value for the <code>currentValue</code> property
<code>incValueTrigger</code>	If true, the <code>currentValue</code> property is incremented by 1

Property name	Description
<code>incValueReaction</code>	The reaction to an incrementation of the <code>currentValue</code> property
<code>decValueTrigger</code>	If true, the current value is decremented by 1
<code>decValueReaction</code>	Reaction to a decrementation of the <code>currentValue</code> property
<code>steps</code>	The number of steps to calculate the increment or decrement for the <code>currentValue</code> property
<code>valueWrapAround</code>	<p>Possible values:</p> <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>

### 12.10.1.8. Touched

The **Touched** widget feature enables a widget to react to touch input.

Table 12.120. Properties of the **Touched** widget feature

Property name	Description
<code>touchable</code>	If true, the widget reacts on touch input
<code>touched</code>	If true, the widget is currently touched
<code>touchPolicy</code>	<p>Defines how to handle touch and movement that crosses widget boundaries.</p> <p>Possible values:</p> <ul style="list-style-type: none"> <li>▶ <code>Press then react (=0)</code></li> </ul> <p>Press first, then the widget reacts. Notifications of moving and releasing are only active within the widget area.</p> <ul style="list-style-type: none"> <li>▶ <code>Press and grab (=1)</code></li> </ul> <p>Press to grab the contact. The contact remains grabbed even if it moves away from the widget area.</p> <ul style="list-style-type: none"> <li>▶ <code>Press then react on contact (=3)</code></li> </ul> <p>Even if the contact enters the pressed state outside the widget boundaries, the subsequent move and release events are delivered to the widget.</p>
<code>touchBehavior</code>	<p>Defines touch evaluation. Possible values:</p> <ul style="list-style-type: none"> <li>▶ <code>Whole area (=0)</code></li> </ul>

Property name	Description
	<p>To identify the touched widget, the renderer evaluates the widget's clipping rectangle.</p> <ul style="list-style-type: none"> <li>▶ <code>Visible pixels (=1)</code></li> </ul> <p>To identify the touched widget, the renderer evaluates the widget the touched pixel belongs to.</p> <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>

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**. EB GUIDE GTF evaluates **Whole area** faster than **Visible pixels**.

### 12.10.1.9. 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 features enables a different truncation than the default setting `trailing`.

Table 12.121. Properties of the **Text truncation** widget feature

Property name	Description
<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>
<code>truncationSymbol</code>	The string that is shown instead of the replaced text part



### 12.10.1.10. Multiple lines

The **Multiple lines** widget feature enables line breaks for a label.

Table 12.122. Properties of the **Multiple lines** widget feature

Property name	Description
<code>lineGap</code>	The size of the gap between the lines. A negative value decreases the gap, a positive value increases the gap.
<code>maxLineCount</code>	The maximum number of visible lines. 0 = no limitation

#### NOTE



#### Character replacement

Sequences of `\\` `\\` are replaced by `\\` . Sequences of `\\` `'n'` are replaced by `'n'`.

If the size of the label is increased so that one line is sufficient to display the text, `'n'` is replaced by `'` .

## 12.10.2. Effects

### 12.10.2.1. 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.

Requirements:

- ▶ The widget feature is available for rectangles.

Table 12.123. Properties of the **Border** widget feature

Property name	Description
<code>borderThickness</code>	The thickness of the border in pixels
<code>borderColor</code>	The color that is used to render the border
<code>borderStyle</code>	The style that is used to render the border

### 12.10.2.2. Coloration

The **Coloration** widget feature colors the widget and its widget subtree. It also affects transparency if the alpha value is not opaque.



**Example 12.1.**  
**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 12.124. Properties of the **Coloration** widget feature

Property name	Description
<code>colorationEnabled</code>	If true, coloration is used
<code>colorationColor</code>	The coloration used. Possible values: <ul style="list-style-type: none"> <li>▶ Pure</li> <li>▶ Opaque</li> <li>▶ White</li> </ul>

## 12.10.3. Focus

### 12.10.3.1. 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.

Table 12.125. Properties of the **User-defined focus** widget feature

Property name	Description
<code>focusNext</code>	The trigger that assigns the focus to the next child widget
<code>focusOrder</code>	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: <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>

Property name	Description
	Each child widget requires the <b>Focused</b> widget feature, otherwise widgets are ignored for focus handling. 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.
<code>focusPrevious</code>	The trigger that assigns the focus to the previous child
<code>focusFlow</code>	The behavior for focus changes within the hierarchy. Possible values: <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>
<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.
<code>initFocus</code>	The index of the focused child widget at initialization

### 12.10.3.2. Auto focus

With the **Auto focus** widget feature, the order in which child widgets are focused is pre-defined. Focusable child widgets cannot be skipped. A widget with the **Auto focus** widget feature manages a local focus hierarchy for its widget subtree. 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.

Table 12.126. Properties of the **Auto focus** widget feature

Property name	Description
<code>focusNext</code>	The condition on which the focus index is incremented
<code>focusPrevious</code>	The condition on which the focus index is decremented
<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 hierachy level (=1)</code></li> <li>▶ <code>step up in hierarchy (=2)</code></li> </ul>
<code>focusedIndex</code>	The index of the currently focused child widget as the n-th child widget which is focusable
<code>initFocus</code>	The index defines the focused child widget at initialization. If the widget is not focusable, the next focusable child is used.

## 12.10.4. Gestures

### 12.10.4.1. Flick gesture

A quick brush of a contact over a surface

Table 12.127. Properties of the **Flick gesture** widget feature

Property name	Description
<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>
<code>flickMaxTime</code>	The maximal time in milliseconds the contact may stay in place for the gesture to be recognized as a flick gesture
<code>flickMinLength</code>	The minimal distance in pixels a contact has to move on the surface to be recognized as a flick gesture

### 12.10.4.2. Hold gesture

A hold gesture without movement

**NOTE** The **Hold gesture** widget feature does not trigger the **Touch lost** widget feature.



Table 12.128. Properties of the **Hold gesture** widget feature

Property name	Description
<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> </ul>

Property name	Description
	▶ y: y-coordinate of the contact position
holdDuration	The minimal time in milliseconds the contact must stay in place for the gesture to be recognized as a hold gesture

### 12.10.4.3. Long hold gesture

A long hold gesture without movement

**NOTE**

The **Long hold gesture** widget feature does not trigger the **Touch lost** widget feature.



Table 12.129. Properties of the **Long hold gesture** widget feature

Property name	Description
onGestureLongHold	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  Reaction arguments:  ▶ x: x-coordinate of the contact position ▶ y: y-coordinate of the contact position
longHoldDuration	The minimal time in milliseconds the contact must stay in place for the gesture to be recognized as a long hold gesture

### 12.10.4.4. Path gestures

A shape drawn by one contact is matched against a set of known shapes.

Table 12.130. Properties of the **Path gesture** widget feature

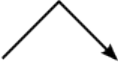
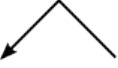



Property name	Description
onPath	The reaction that is triggered when the entered shape matches. The reaction is only triggered if <code>onPathStart</code> has been triggered already.
onPathStart	The reaction that is triggered once a contact moves beyond the minimal box ( <code>pathMinXBox</code> , <code>pathMinYBox</code> .) Reaction argument:  ▶ <code>gestureId</code> : ID of the path that was matched



Property name	Description
<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.
<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
<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

#### 12.10.4.4.1. 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 12.131. 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
4		Check mark

ID	Shape	Description
5		Wave shape left to right
6		Wave shape right to left

### 12.10.4.5. Pinch gesture

Two contacts that move closer together or further apart

Table 12.132. Properties of the **Pinch gesture** widget feature

Property name	Description
<code>onGesturePinchStart</code>	The reaction that is triggered once the start of the gesture is recognized. Reaction arguments: <ul style="list-style-type: none"> <li>▶ ratio: Current contact distance to initial contact distance ratio</li> <li>▶ centerX: x-coordinate of the current center point between the two contacts</li> <li>▶ centerY: y-coordinate of the current center point between the two contacts</li> </ul>
<code>onGesturePinchUpdate</code>	The reaction that is triggered when the pinch ratio or center point change. Reaction arguments: <ul style="list-style-type: none"> <li>▶ ratio: Current contact distance to initial contact distance ratio</li> <li>▶ centerX: x-coordinate of the current center point between the two contacts</li> <li>▶ centerY: y-coordinate of the current center point between the two contacts</li> </ul>
<code>onGesturePinchEnd</code>	The reaction that is triggered once the gesture is finished. Reaction arguments: <ul style="list-style-type: none"> <li>▶ ratio: Current contact distance to initial contact distance ratio</li> <li>▶ centerX: x-coordinate of the current center point between the two contacts</li> </ul>

Property name	Description
	▶ centerY: y-coordinate of the current center point between the two contacts
pinchThreshold	The minimal distance in pixels each contact has to move from its initial position for the gesture to be recognized

### 12.10.4.6. Rotate gesture

Two contacts that move along a circle

Table 12.133. Properties of the **Rotate gesture** widget feature

Property name	Description
onGestureRotateStart	The reaction that is triggered once the start of the gesture is recognized
onGestureRotateUpdate	The reaction that is triggered when the recognized angle or center point changes
onGestureRotateEnd	The reaction that is triggered once the gesture is finished
rotateThreshold	The minimal distance in pixels each contact has to move from its initial position for the start of the gesture to be recognized

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

## 12.10.5. Input handling

### 12.10.5.1. Move over

The **Move over** widget feature enables a widget to react on movement within its boundaries.

Table 12.134. Properties of the **Move over** widget feature

Property name	Description
moveOver	The widget's reaction on a movement within its boundaries  Reaction arguments:



Property name	Description
	<ul style="list-style-type: none"><li>▶ touchId: The ID of the touch screen the user has clicked or released</li><li>▶ fingerId: The ID of the contact that moves across the widget</li></ul>

### 12.10.5.2. Move out

The **Move out** widget feature enables a widget to react on movement out of its boundaries.

Table 12.135. Properties of the **Move out** widget feature

Property name	Description
moveOut	The widget's reaction on a movement out of its boundaries  Reaction arguments: <ul style="list-style-type: none"><li>▶ touchId: The ID of the touch screen the user has clicked or released</li><li>▶ fingerId: The ID of the contact that moves across the widget</li></ul>

### 12.10.5.3. Move in

The **Move in** widget feature enables a widget to react on movement into its boundaries.

Table 12.136. Properties of the **Move in** widget feature

Property name	Description
moveIn	The widget's reaction on a movement into its boundaries  Reaction arguments: <ul style="list-style-type: none"><li>▶ touchId: The ID of the touch screen the user has clicked or released</li><li>▶ fingerId: The ID of the contact that moves across the widget</li></ul>

### 12.10.5.4. Touch pressed

The **Touch pressed** widget feature enables a widget to react on being pressed.

Table 12.137. Properties of the **Touch pressed** widget feature

Property name	Description
touchPressed	The widget's reaction on being pressed

Property name	Description
	Reaction arguments: <ul style="list-style-type: none"> <li>▶ touchId: The ID of the touch screen the user has clicked or released</li> <li>▶ fingerId: The ID of the contact that moves across the widget</li> </ul>

### 12.10.5.5. Touch released

The **Touch released** widget feature enables a widget to react on being released.

Table 12.138. Properties of the **Touch released** widget feature

Property name	Description
touchShortReleased	The widget's reaction on being released Reaction arguments: <ul style="list-style-type: none"> <li>▶ touchId: The ID of the touch screen the user has clicked or released</li> <li>▶ fingerId: The ID of the contact that moves across the widget</li> </ul>

### 12.10.5.6. 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.

Table 12.139. Properties of the **Touch lost** widget feature

Property name	Description
onTouchGrabLost	The widget's reaction on a lost touch contact Reaction arguments: <ul style="list-style-type: none"> <li>▶ touchId: The ID of the touch screen the user has clicked or released</li> <li>▶ fingerId: The ID of the contact that moves across the widget</li> </ul>

### 12.10.5.7. Touch status changed

The **Touch status changed** widget feature enables a widget to react on changes of its touch status.

Table 12.140. Properties of the **Touch status changed** widget feature

Property name	Description
touchStatusChanged	<p>The widget's reaction on changes of its touch status</p> <p>Reaction arguments:</p> <ul style="list-style-type: none"><li>▶ touchId: The ID of the touch screen the user has clicked or released</li><li>▶ touchStatus: 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>▶ 6: any status change</li></ul> <li>▶ fingerId: The ID of the contact that moves across the widget</li>

### 12.10.5.8. Touch move

The **Touch move** widget feature enables a widget to react on being touched and moved.

Table 12.141. Properties of the **Touch move** widget feature

Property name	Description
touchMoved	<p>The widget's reaction on being touched and moved</p> <p>Reaction arguments:</p> <ul style="list-style-type: none"><li>▶ touchId: The ID of the touch screen the user has clicked or released</li><li>▶ fingerId: The ID of the contact that moves across the widget</li></ul>

### 12.10.5.9. Gestures

The **Gestures** widget feature enables the widget to react on touch gestures.

The **Gestures** widget feature has no additional properties.

### 12.10.5.10. Key pressed

The **Key pressed** widget feature enables a widget to react on a key being pressed.

Table 12.142. Properties of the **Key pressed** widget feature

Property name	Description
keyPressed	The widget's reaction on a key being pressed  Reaction argument:  ▶ keyId: The ID of the key that is processed

### 12.10.5.11. Key unicode

The **Key unicode** widget feature enables a widget to react on Unicode key input.

Table 12.143. Properties of the **Key unicode** widget feature

Property name	Description
keyUnicode	The widget's reaction on a Unicode key input  Reaction argument:  ▶ keyId: The ID of the key that is processed

### 12.10.5.12. Key released

The **Key released** widget feature enables a widget to react on a key being released.

Table 12.144. Properties of the **Key released** widget feature

Property name	Description
keyShortReleased	The widget's reaction on a key being released  Reaction argument:  ▶ keyId: The ID of the key that is processed

### 12.10.5.13. 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**.

Table 12.145. Properties of the **Key status changed** widget feature

Property name	Description
<code>keyStatusChanged</code>	The widget's reaction on a key being pressed or released  Reaction arguments: <ul style="list-style-type: none"><li>▶ <code>keyId</code>: The ID of the key that is processed</li><li>▶ <code>status</code>: The numeric ID of the status change</li></ul>

#### 12.10.5.14. Rotary

The **Rotary** widget feature enables a widget to react on being rotated.

Table 12.146. Properties of the **Rotary** widget feature

Property name	Description
<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>

#### 12.10.5.15. Moveable

The **Moveable** widget feature enables a widget to be moved by touch.

Table 12.147. Properties of the **Moveable** widget feature

Property name	Description
<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>

## 12.10.6. Layout

### 12.10.6.1. 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.

Table 12.148. Properties of the **Absolute layout** widget feature

Property name	Description
<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.
<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.
<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.
<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.

### 12.10.6.2. 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.

Table 12.149. Properties of the **Box layout** widget feature

Property name	Description
<code>gap</code>	The space between two child widgets, depending on the layout direction
<code>layoutDirection</code>	The direction in which the list elements i.e. the child widgets are positioned.

### 12.10.6.3. 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.

Table 12.150. Properties of the **Flow layout** widget feature

Property name	Description
<code>horizontalGap</code>	The horizontal space between two child widgets
<code>verticalGap</code>	The vertical space between two child widgets
<code>layoutDirection</code>	The direction in which the list elements i.e. the child widgets are positioned.
<code>horizontalChildAlign</code>	The horizontal alignment of child widgets
<code>verticalChildAlign</code>	The vertical alignment of child widgets <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>

#### 12.10.6.4. 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.

Table 12.151. Properties of the **Grid layout** widget feature

Property name	Description
<code>horizontalGap</code>	The horizontal space between two child widgets
<code>verticalGap</code>	The vertical space between two child widgets
<code>numRows</code>	Defines the number of rows
<code>numColumns</code>	Defines the number of columns

#### 12.10.6.5. 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 12.152. Properties of the **Layout margins** widget feature

Property name	Description
<code>leftMargin</code>	The margin of the left border
<code>topMargin</code>	The margin of the top border

Property name	Description
<code>rightMargin</code>	The margin of the right border
<code>bottomMargin</code>	The margin of the bottom border

### 12.10.6.6. List layout

The **List layout** widget feature defines position and size of each child widget.

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, refer to [section 12.10.7.2, “List index”](#).

Table 12.153. Properties of the **List layout** widget feature

Property name	Description
<code>layoutDirection</code>	The direction in which the list elements i.e. the child widgets are positioned.
<code>scrollOffset</code>	The amount of pixels to scroll the list
<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.
<code>firstListIndex</code>	The list index of the first visible list element, defined by the widget feature
<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.
<code>scrollValue</code>	The current scroll value
<code>scrollValueMax</code>	The maximum scroll value, which is mapped to the end of the list
<code>scrollValueMin</code>	The minimum scroll value, which is mapped to the beginning of the list
<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.
<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.
<code>segments</code>	For horizontal layout direction: the number of rows  For vertical layout direction: the number of columns



Property name	Description
listLength	The number of list elements
wrapAround	Possible values: <ul style="list-style-type: none"> <li>▶ true: The <code>scrollValue</code> property continues at the inverse border, if <code>scrollValueMin</code> or <code>scrollValueMax</code> is exceeded.</li> <li>▶ false: The <code>scrollValue</code> property does not decrease/increase, if <code>scrollValueMin</code> or <code>scrollValueMax</code> is exceeded.</li> </ul>

### 12.10.6.7. Scale mode

The **Scale mode** widget feature defines how an image is displayed if its size differs from the size of the widget.

Table 12.154. Properties of the **Scale mode** widget feature

Property name	Description
scaleMode	The scale mode of the image. Possible values: <ul style="list-style-type: none"> <li>▶ 0 = original size</li> <li>▶ 1 = fit to size</li> <li>▶ 2 = keep aspect ratio</li> </ul>

## 12.10.7. List management

### 12.10.7.1. Line index

The **Line index** widget feature adds a line index property to a widget. It is intended to be used in combination with tables.

Table 12.155. Properties of the **Line index** widget feature

Property name	Description
lineIndex	The index of the current line in a table

### 12.10.7.2. List index

The **List index** widget feature adds a `listIndex` property to a widget. It is intended to be used in combination with the **List layout** widget feature.

Table 12.156. Properties of the **List index** widget feature

Property name	Description
<code>listIndex</code>	The index of the current widget in a list

### 12.10.7.3. Template index

The **Template index** widget feature adds a line template index property to a widget. It is intended to be used in combination with instantiators.

Table 12.157. Properties of the **Template index** widget feature

Property name	Description
<code>lineTemplateIndex</code>	The index of the used line template

### 12.10.7.4. View port

The **View port** widget feature clips oversized elements at the widget borders. It is intended to be used in combination with containers or lists.

The **View port** widget feature takes effect on the following model elements:

- ▶ Child widgets of the widget you added **View port** to are clipped inside the dimensions of the widget.
- ▶ The widget you added **View port** is clipped inside the dimensions of its parent view.

Table 12.158. Properties of the **View port** widget feature

Property name	Description
<code>xOffset</code>	The horizontal offset of the visible clipping within the drawn area of child widgets
<code>yOffset</code>	The vertical offset of the visible clipping within the drawn area of child widgets

## 12.10.8. 3D

Widget features in the **3D** category are only available for 3D widgets.

### 12.10.8.1. Camera viewport

The **Camera viewport** widget feature defines the camera's drawing region within the scene graph.

Requirements:

- ▶ The **Camera viewport** widget feature is available for camera.

Table 12.159. Properties of the **Camera viewport** widget feature

Property name	Description
viewportX	The x-origin of the viewport within the scene graph
viewportY	The y-origin of the viewport within the scene graph
viewportWidth	The viewport's width
viewportHeight	The viewport's height

### 12.10.8.2. Ambient texture

The **Ambient texture** widget feature adds extended configuration values to a material.

Requirements:

- ▶ The **Ambient texture** widget feature is available for material.

Table 12.160. Properties of the **Ambient texture** widget feature

Property name	Description
ambientTexture	The file name of the texture
ambientTextureAddressModeU	The address mode of the texture along the u-direction
ambientTextureAddressModeV	The address mode of the texture along the v-direction
ambientFilterMode	The filter mode of the texture

### 12.10.8.3. Diffuse texture

The **Diffuse texture** widget feature adds extended configuration values to a material.

Requirements:

- ▶ The **Diffuse texture** widget feature is available for material.

Table 12.161. Properties of the **Diffuse texture** widget feature

Property name	Description
diffuseTexture	The file name of the texture

Property name	Description
diffuseTextureAddressModeU	The address mode of the texture along the u-direction
diffuseTextureAddressModeV	The address mode of the texture along the v-direction
diffuseFilterMode	The filter mode of the texture

#### 12.10.8.4. Emissive texture

The **Emissive texture** widget feature adds extended configuration values to a material.

Requirements:

- ▶ The **Emissive texture** widget feature is available for material.

Table 12.162. Properties of the **Emissive texture** widget feature

Property name	Description
emissiveTexture	The file name of the texture
emissiveTextureAddress-ModeU	The address mode of the texture along the u-direction
emissiveTextureAddressModeV	The address mode of the texture along the v-direction
emissiveFilterMode	The filter mode of the texture

#### 12.10.8.5. Light map texture

The **Light map texture** widget feature adds extended configuration values to a material.

Requirements:

- ▶ The **Light map texture** widget feature is available for material.

Table 12.163. Properties of the **Light map texture** widget feature

Property name	Description
lightMapTexture	The file name of the texture
lightMapTextureAddress-ModeU	The address mode of the texture along the u-direction
lightMapTextureAddressModeV	The address mode of the texture along the v-direction
lightMapFilterMode	The filter mode of the texture

### 12.10.8.6. Normal map texture

The **Normal map** widget feature adds extended configuration values to a material.

Requirements:

- ▶ The **Normal map texture** widget feature is available for material.

Table 12.164. Properties of the **Normal map** widget feature

Property name	Description
normalMapTexture	The file name of the texture
normalMapTextureAddress-ModeU	The address mode of the texture along the u-direction
normalMapTextureAddress-ModeV	The address mode of the texture along the v-direction
normalMapFilterMode	The filter mode of the texture

### 12.10.8.7. Opaque texture

The **Opaque texture** widget feature adds extended configuration values to a material.

Requirements:

- ▶ The **Opaque texture** widget feature is available for material.

Table 12.165. Properties of the **Opaque texture** widget feature

Property name	Description
opaqueTexture	The file name of the texture
opaqueTextureAddressModeU	The address mode of the texture along the u-direction
opaqueTextureAddressModeV	The address mode of the texture along the v-direction
opaqueFilterMode	The filter mode of the texture

### 12.10.8.8. Reflection texture

The **Reflection texture** widget feature adds extended configuration values to a material.

Requirements:

- ▶ The **Reflection texture** widget feature is available for material.

Table 12.166. Properties of the **Reflection texture** widget feature

Property name	Description
reflectionTopTexture	The file name of the texture
reflectionBottomTexture	The file name of the texture
reflectionLeftTexture	The file name of the texture
reflectionRightTexture	The file name of the texture
reflectionFrontTexture	The file name of the texture
reflectionBackTexture	The file name of the texture
reflectionFilterMode	The filter mode of the texture

**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.

### 12.10.8.9. Specular texture

The **Specular texture** widget feature adds extended configuration values to a material.

Requirements:

- ▶ The **Specular texture** widget feature is available for material.

Table 12.167. Properties of the **Specular texture** widget feature

Property name	Description
specularTexture	The file name of the texture
specularTextureAddress- ModeU	The address mode of the texture along the u-direction
specularTextureAddressMod- eV	The address mode of the texture along the v-direction

Property name	Description
specularFilterMode	The filter mode of the texture

## 12.10.9. Transformations

Transformations 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

### 12.10.9.1. 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).

Table 12.168. Properties of the **Pivot** widget feature

Property name	Description
pivotX	The pivot point on the x-axis relative to parent widget
pivotY	The pivot point on the y-axis relative to parent widget
pivotZ	The pivot point on the z-axis relative to parent widget if widget is a scene graph

### 12.10.9.2. Rotation

The **Rotation** widget feature is used to rotate the widget and its subtree.

Table 12.169. Properties of the **Rotation** widget feature

Property name	Description
rotationEnabled	Defines whether rotation is used or not



Property name	Description
rotationAngleX	The rotation angle on the x-axis. This property only affects scene graph.
rotationAngleY	The rotation angle on the y-axis. This property only affects scene graph.
rotationAngleZ	The rotation angle on the z-axis

### 12.10.9.3. Scaling

The **Scaling** widget feature is used to scale the widget and its subtree.

Table 12.170. Properties of the **Scaling** widget feature

Property name	Description
scalingEnabled	Defines whether scaling is used or not
scalingX	The scaling on the x-axis in percent
scalingY	The scaling on the y-axis in percent
scalingZ	The scaling on the z-axis in percent if widget is a scene graph

### 12.10.9.4. Shearing

The **Shearing** widget feature is used to distort widgets in the widget subtree.

Table 12.171. Properties of the **Shearing** widget feature

Property name	Description
shearingEnabled	Defines whether shearing is used or not
shearingXbyY	The shearing amount of x-axis by y-axis
shearingXbyZ	The shearing amount of x-axis by z-axis if widget is a scene graph
shearingYbyX	The shearing amount of y-axis by x-axis
shearingYbyZ	The shearing amount of y-axis by z-axis if widget is a scene graph
shearingZbyX	The shearing amount of z-axis by x-axis if widget is a scene graph
shearingZbyY	The shearing amount of z-axis by y-axis if widget is a scene graph

### 12.10.9.5. Translation

The **Translation** widget feature is used to translate the widget and its subtree. It moves widgets in x, y and z directions.





Table 12.172. Properties of the **Translation** widget feature

<b>Property name</b>	<b>Description</b>
translationEnabled	Defines whether translation is used or not
translationX	The translation on the x-axis
translationY	The translation on the y-axis
translationZ	The translation on the z-axis if widget is a scene graph

# 13. Installation

## 13.1. Background information

### 13.1.1. Restrictions

**NOTE**



**Compatibility**

EB GUIDE product line 6 is not compatible with any previous major version.

**NOTE**



**EB GUIDE Speech Extension**

EB GUIDE Speech Extension is licensed as an add-on product that is enabled only when purchased.

**NOTE**



**User rights**

To install EB GUIDE on Windows 7 or Windows 10 systems, you require administrator rights.

### 13.1.2. System requirements

Observe the following settings:

Table 13.1. 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 7, Windows 10
Screen resolution	Usage of 2 separate monitors with 1920 x 1080 pixels
Software	Microsoft .NET Framework 4.5.1.

	DirectX 11
--	------------

Table 13.2. Recommended settings for EB GUIDE SDK

Development environment (IDE)	Microsoft Visual Studio 2013 or newer
File integration	CMake

## 13.2. Downloading from EB Command

EB Command is the server from which you are going to download the EB GUIDE product line software.

### NOTE



### Activate your account

After ordering a product, you receive a mail 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.



### Downloading from EB Command

Prerequisite:

- Your user account is activated.

#### Step 1

Open a browser and go to [https://command.elektrobit.com/command/mod\\_perl/login.pl](https://command.elektrobit.com/command/mod_perl/login.pl).

The EB Command front page opens.

#### Step 2

To change the language, toggle the language in the lower left corner of the screen.

#### Step 3

Type in your alias, which is your user name.

#### Step 4

Type in your password and click the **Login** button.

The main page opens.

#### Step 5

Select a project, for example EB GUIDE Studio. The project overview opens.

#### Step 6

Select the distribution container in the version you want to download, for example EB GUIDE Studio Core 6.x. An overview of all downloadable items open.

#### Step 7

Select the **Actions** check box beside the file you want to download.

Step 8

Click **Download Selection**.

**TIP**



**Downloading multiple files**

If you select multiple files for download, a download package is generated. You are prompted to save the file `CommandDownload<date>.zip` to your local system.

The download starts. To log out from EB Command, click the **Logout** button.

## 13.3. 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 directory for installation.

The default installation directory is `C:\Program Files (x86)\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.

## 13.4. 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

API

Application programming interface

## 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 GTF

EB GUIDE GTF is the graphics target framework of the EB GUIDE product line and is part of the 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 description of an HMI created with EB GUIDE Studio.

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 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 the EB GUIDE product line. It consists of EB GUIDE GTF and EB GUIDE STF. It is required to run an EB GUIDE model.

## G

GL	Graphical library
GUI	Graphical user interface

## H

HMI	Human machine interface
-----	-------------------------

## L

library	A library is a set of resources used in EB GUIDE Studio. Libraries that are necessary for an EB GUIDE project are defined in the project center.
---------	--

## M

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.

## O

OS Operating system

## P

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 `gtfStartup.cfg` 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.

## 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.

## W

widget                      A widget is a basic graphical element. Widgets are used for interaction with a graphical user interface.

# Index

## Symbols

- 3D graphic, 33, 52, 185, 278
  - add, 111
  - import, 185
  - mesh, 52
  - supported formats, 33, 52
- 3D object, 33
- 3D widget, 52, 91
- 3D widgets, 33
  - reference, 240

## A

- absolute layout
  - reference, 262
- action, 223
  - entry action, 96
  - exit action, 96
  - transition, 106
- Ambient texture
  - reference, 267
- animation, 35, 91, 127, 129, 178
  - entry animation, 36, 129
  - exit animation, 36, 129
  - reference, 235
- API, 278 (see application programming interface)
- application programming interface, 36
- auto focus
  - reference, 251

## B

- basic widget, 91
- basic widgets
  - reference, 232
- boolean
  - data type, 193
- boolean list
  - data type, 193
- border
  - reference, 249

- box layout
  - reference, 262
- button
  - user interface, 67

## C

- camera
  - reference, 242
- camera viewport
  - reference, 266
- child visibility selection
  - reference, 244
- choice state, 99
- color
  - data type, 193
- coloration
  - reference, 249
- command area
  - project editor, 42
- command line, 67
- communication context, 37, 139, 278
- compound state, 98
- condition
  - transition, 105
- conditional script
  - data type, 194
- configuration file, 222
- configure
  - display, 151
- console (see command line)
- constant curve
  - reference, 236
- container
  - add, 113
  - reference, 234
- content area
  - project center, 38
  - project editor, 41
- copy
  - datapool item, 137
  - event, 134

**D**

## data type

- boolean, 193
- boolean list, 193
- color, 193
- conditional script, 194
- float, 194
- font, 195
- image, 195
- integer, 195
- list, 196
- mesh, 193
- mesh list, 193
- string, 197

## datapool, 45, 278

## datapool item, 46, 138, 278

- add, 136
- copy, 137
- export, 151
- import, 153
- language support, 182
- link, 141
- list, 137
- paste, 137
- reference, 192
- windowed list, 46

## Diffuse texture

- reference, 267

## directional light

- reference, 242

## display

- configure, 151

## dynamic state machine

- add, 95, 155

**E**

## EB GUIDE extension, 48

## EB GUIDE GTF, 278

## EB GUIDE GTF SDK, 278

## EB GUIDE model, 47, 278

- model element, 47

## EB GUIDE Monitor, 145

## EB GUIDE product line, 278

## EB GUIDE project, 47

## EB GUIDE Script, 52, 138, 278

- comment, 54
- datapool access, 61
- event, 64
- expression, 55
- foreign function call, 60
- identifier, 53
- if-then-else, 59
- l-value, 57
- list, 63
- local variable, 57
- namespace, 53
- r-value, 57
- scripted value, 66
- standard library, 66
- string formatting, 65
- tutorial, 163
- types, 54
- while loop, 58
- widget property, 62

## EB GUIDE SDK, 278

## EB GUIDE Studio, 278

## EB GUIDE Studio SDK, 278

## EB GUIDE TF, 278

## effects

- widget feature, 249

## Emissive texture

- reference, 268

## enabled

- reference, 245

## entry action, 100

- state machine, 96

## entry animation, 129

- reference, 232

## event, 48, 63

- add, 134
- copy, 134
- paste, 134
- reference, 222

## event system, 48

exit action, 101  
    state machine, 97  
exit animation, 129  
    reference, 232  
export  
    language dependent text, 151

## F

fast start curve  
    reference, 237  
finger ID, 89  
flick gesture  
    reference, 252  
float  
    data type, 194  
flow layout  
    reference, 262  
focused  
    reference, 245  
font, 51  
    data type, 195

## G

gesture, 88  
    non-path gesture, 88  
    path gesture, 88  
    reference, 259  
gesture ID  
    reference, 254  
GL, 279  
grid layout  
    reference, 263  
gtfStartup.cfg, 222  
    profile, 147  
GUI, 279

## H

HMI, 279  
hold gesture  
    reference, 252

## I

icon  
    user interface, 67  
image  
    9-patch, 51  
    add, 111  
    data type, 195  
    reference, 234  
    supported formats, 51  
import  
    language dependent text, 153  
instantiator, 172  
    add, 113  
    reference, 235  
integer  
    data type, 195  
internal transition, 108

## K

key pressed  
    reference, 260  
key released  
    reference, 260  
key status changed  
    reference, 260  
key unicode  
    reference, 260

## L

label  
    font, 112  
    reference, 233  
Language  
    change, 182  
language dependent text, 182  
    export, 151  
    import, 153  
layout margins  
    reference, 263  
library, 279  
    add, 148  
Light map texture

- reference, 268
- line index
  - reference, 265
- linear curve, 239
- linear interpolation curve, 240
- linear interpolation integer, 179
- link
  - datapool item, 141
  - widget property, 116, 118
- list, 137
  - create, 172
  - data type, 196
- list index
  - reference, 265
- list layout
  - reference, 264
- long hold gesture
  - reference, 253

## M

- material
  - reference, 242
- mesh, 52
  - data type, 193
  - reference, 243
- mesh list
  - data type, 193
- message, 223
  - add, 150
- model element, 47, 279
  - delete, 102
- move in
  - reference, 257
- move out
  - reference, 257
- move over
  - reference, 256
- moveable
  - reference, 261
- multi-touch input, 89
- multiple lines
  - reference, 249

- multisampling, 231

## N

- navigation area
  - project center, 38
  - project editor, 39
- Normal map texture
  - reference, 269

## O

- Opaque texture
  - reference, 269
- OS, 280

## P

- paste
  - datapool item, 137
  - event, 134
- path gesture, 169
  - reference, 253, 254
- pinch gesture
  - reference, 255
- pivot
  - reference, 271
- point light
  - reference, 243
- pressed
  - references, 245
- problems area, 144
  - project editor, 45
- profile, 280
  - clone, 148
  - gtfStartup.cfg, 147
- project center, 37, 280
  - content area, 38
  - navigation area, 38
- project editor, 39, 280
  - command area, 42, 45
  - content area, 41
  - navigation area, 39
  - problems area, 45
  - toolbox, 43

properties panel  
   command area, 44  
   project editor, 44

## Q

quadratic curve  
   reference, 238

## R

reader communication context, 37, 140  
 rectangle  
   reference, 233  
 Reflection texture  
   reference, 269  
 renderer  
   configure, 151  
 resource, 280  
   3D graphic, 52  
   font, 51  
   image, 51  
   mesh, 52  
 resource management, 50  
 rotary  
   reference, 261  
 rotate gesture  
   reference, 256  
 rotation  
   reference, 271

## S

scale mode  
   reference, 265  
 scaling  
   reference, 272  
 scene configuration  
   reference, 229  
 scene graph, 33, 52, 111, 185  
   add, 111  
   reference, 241  
   texture, 185  
 scene graph node  
   reference, 241

script curve, 239  
 scripted value, 66, 138  
 selected  
   reference, 245  
 Selection group  
   reference, 246  
 shared library, 280  
 shearing  
   reference, 272  
 shortcut  
   user interface, 67  
 signal, 223  
 simulation, 145  
 sinus curve  
   reference, 238  
 slow start curve  
   reference, 237  
 Specular texture  
   reference, 270  
 Spinning  
   reference, 246  
 spot light  
   reference, 243  
 state, 71, 97, 98, 159, 280  
   choice state, 74  
   compound state, 71  
   entry action, 100  
   exit action, 101  
   final state, 73  
   history state, 75  
   initial state, 72  
   transition, 102  
   view state, 72  
 state machine, 70, 280  
   add, 95  
   comparison to UML, 86  
   delete, 97  
   dynamic state machine, 70  
   execution of state machine, 82  
   haptic state machine, 70  
   include state machine, 70, 87  
   logic state machine, 70

- state, 71
- transition, 78
- UML 2.5 notation, 86
- status bar
  - project editor, 45
- string
  - data type, 197
- system message, 150

## T

- template
  - create, 130
  - delete, 132
  - use, 132
- template index
  - reference, 266
- template interface, 131
  - add property, 131
  - remove property, 131
- text truncation
  - reference, 248
- todo
  - EB GUIDE Script, 54
- token, 223
- toolbox
  - project editor, 43
- touch gesture (see gesture)
- touch input (see gesture)
- touch lost
  - reference, 258
- touch move
  - reference, 259
- touch pressed
  - reference, 257
- touch released
  - reference, 258
- touch status changed
  - reference, 258
- touched
  - reference, 247
- transition, 78, 102, 280
  - action, 106

- add, 102
- condition, 105
- internal, 108
- move, 103
- trigger, 104
- translation
  - reference, 272
- trigger
  - transition, 104

## U

- UI, 281
- user-defined focus
  - reference, 250
- user-defined property, 119

## V

- view, 90, 281
  - add, 109
  - reference, 231
- view port
  - reference, 266
- view template
  - reference, 231, 232

## W

- widget, 90, 281
  - 3D widget, 91
  - add, 109
  - animation, 91
  - basic, 91
  - delete, 110
  - group, 113
  - position, 114
  - resize, 115
- widget feature
  - add, 122
  - path gesture, 169
  - remove, 124
- widget property, 92
  - add, 119
  - default property, 92

- EB GUIDE Script, 62
  - link to datapool item, 118
  - link to widget property, 116
  - user-defined property, 92, 119
  - widget feature property, 92
  - widget template, 94
- widget template, 93, 130, 133
- widget template interface, 94
- windowed list
  - datapool item, 46
- writer communication context, 37, 140