



Elektrobit

EB GUIDE Studio

User manual

Version 6.5.1.137731



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 2018, 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	38

6.5.2.1. Navigation component	39
6.5.2.2. Outline component	40
6.5.2.3. Toolbox component	41
6.5.2.4. Properties component	42
6.5.2.5. Content area	42
6.5.2.6. Events component	43
6.5.2.7. Datapool component	43
6.5.2.8. Assets component	44
6.5.2.9. Command area	44
6.5.2.10. Problems component	45
6.5.3. Dockable component	45
6.5.4. EB GUIDE Monitor	46
6.6. Datapool	48
6.6.1. Concept	48
6.6.2. Datapool items	48
6.6.3. Windowed lists	49
6.7. EB GUIDE model and EB GUIDE project	49
6.8. Event handling	50
6.8.1. Event system	50
6.8.2. Events	50
6.9. Extensions	51
6.9.1. EB GUIDE Studio extension	51
6.9.2. EB GUIDE GTF extension	51
6.10. Languages	51
6.10.1. Display languages in EB GUIDE Studio	51
6.10.2. Languages in the EB GUIDE model	52
6.10.3. Export and import of language-dependent texts	52
6.11. Skins	53
6.12. Resource management	53
6.12.1. Fonts	53
6.12.2. Images	54
6.12.2.1. 9-patch images	54
6.12.3. Meshes for 3D graphics	55
6.12.4. .psd file format	55
6.13. Scripting language EB GUIDE Script	55
6.13.1. Capabilities and areas of application	55
6.13.2. Namespaces and identifiers	56
6.13.3. Comments	57
6.13.4. Types	57
6.13.5. Expressions	58
6.13.6. Constants and references	58
6.13.7. Arithmetic and logic expressions	59

6.13.8. L-values and r-values	60
6.13.9. Local variables	60
6.13.10. While loops	61
6.13.11. If-then-else	62
6.13.12. Foreign function calls	63
6.13.13. Datapool access	64
6.13.14. Widget properties	65
6.13.15. Lists	66
6.13.16. Events	66
6.13.17. String formatting	68
6.13.18. The standard library	69
6.14. Scripted values	69
6.15. Shortcuts, buttons and icons	71
6.15.1. Shortcuts	71
6.15.1.1. Shortcuts in command line	72
6.15.2. Buttons	73
6.15.3. Icons	73
6.16. State machines and states	74
6.16.1. State machines	74
6.16.1.1. Haptic state machine	74
6.16.1.2. Logic state machine	74
6.16.1.3. Dynamic state machine	75
6.16.2. States	75
6.16.2.1. Compound state	75
6.16.2.2. View state	77
6.16.2.3. Initial state	77
6.16.2.4. Final state	78
6.16.2.5. Choice state	79
6.16.2.6. History states	80
6.16.3. Transitions	83
6.16.4. Execution of a state machine	87
6.16.5. EB GUIDE notation in comparison to UML notation	91
6.16.5.1. Supported elements	92
6.16.5.2. Not supported elements	92
6.16.5.3. Deviations	92
6.17. Touch input	93
6.17.1. Non-path gestures	93
6.17.2. Path gestures	93
6.17.3. Input processing and gestures	94
6.17.4. Multi-touch input	94
6.18. Widgets	95
6.18.1. View	95

6.18.2. Widget categories	96
6.18.3. Widget properties	97
6.18.4. Widget templates	98
6.18.5. Widget features	99
6.18.5.1. Focus widget feature category	100
6.18.5.2. List management widget feature category	101
7. Modeling HMI behavior	103
7.1. Modeling a state machine	103
7.1.1. Adding a state machine	103
7.1.2. Adding a dynamic state machine	103
7.1.3. Defining an entry action for a state machine	104
7.1.4. Defining an exit action for a state machine	104
7.1.5. Deleting a state machine	105
7.2. Modeling states	105
7.2.1. Adding a state	105
7.2.2. Adding a state to a compound state	106
7.2.3. Adding a choice state	107
7.2.4. Defining an entry action for a state	108
7.2.5. Defining an exit action for a state	109
7.2.6. Deleting a model element from a state machine	110
7.3. Connecting states through transitions	110
7.3.1. Adding a transition between two states	110
7.3.2. Moving a transition	111
7.3.3. Defining a trigger for a transition	112
7.3.4. Adding a condition to a transition	113
7.3.5. Adding an action to a transition	114
7.3.6. Adding an internal transition to a state	116
8. Modeling HMI appearance	117
8.1. Working with widgets	117
8.1.1. Adding a view	117
8.1.2. Adding a basic widget to a view	118
8.1.2.1. Adding a rectangle	118
8.1.2.2. Adding an ellipse	118
8.1.2.2.1. Editing an ellipse	118
8.1.2.3. Adding an image	119
8.1.2.4. Adding a label	121
8.1.2.4.1. Changing the font of a label	122
8.1.2.5. Adding a container	122
8.1.2.6. Adding an instantiator	123
8.1.3. Adding a 3D widget to a view	125
8.1.3.1. Adding a scene graph to a view	125
8.1.4. Adding a .psd file to a view	126

8.1.5. Deleting a widget from a view	127
8.2. Working with widget properties	127
8.2.1. Positioning a widget	127
8.2.2. Resizing a widget	128
8.2.3. Linking between widget properties	129
8.2.4. Linking a widget property to a datapool item	131
8.2.5. Adding a user-defined property to a widget	133
8.2.5.1. Adding a user-defined property of type <code>Function (): bool</code>	134
8.2.6. Renaming a user-defined property	135
8.3. Extending a widget by widget features	136
8.3.1. Adding a widget feature	136
8.3.2. Removing a widget feature	138
8.4. Adding a language to the EB GUIDE model	139
8.4.1. Adding a language	140
8.4.2. Deleting a language	141
8.5. Working with skin support	141
8.5.1. Adding a skin to the EB GUIDE model	142
8.5.2. Adding skin support to a datapool item	142
8.5.3. Switching between skins	143
8.5.4. Deleting a skin	144
8.6. Adding animations	144
8.6.1. Animating a widget	144
8.6.2. Animating a view transition	146
8.7. Re-using a widget	147
8.7.1. Adding a template	147
8.7.2. Defining the template interface	148
8.7.3. Using a template	149
8.7.4. Deleting a template	149
9. Handling data	151
9.1. Adding an event	151
9.2. Adding a parameter to an event	151
9.3. Addressing an event	152
9.4. Deleting an event	153
9.5. Adding a datapool item	153
9.6. Editing datapool items of a list type	154
9.7. Converting a property to a scripted value	154
9.8. Establishing external communication	155
9.9. Linking between datapool items	156
9.10. Deleting a datapool item	158
10. Handling a project	159
10.1. Creating a project	159
10.2. Opening a project	159

10.2.1. Opening a project from the file explorer	159
10.2.2. Opening a project within EB GUIDE Studio	160
10.3. Renaming model elements, datapool items and events	160
10.4. Validating and simulating an EB GUIDE model	161
10.4.1. Validating an EB GUIDE model	162
10.4.1.1. Validating an EB GUIDE model using EB GUIDE Studio	162
10.4.1.2. Validating an EB GUIDE model using command line	163
10.4.2. Starting and stopping the simulation	163
10.5. Working with EB GUIDE Monitor	163
10.5.1. Firing an event in EB GUIDE Monitor	163
10.5.2. Changing value of the datapool item with EB GUIDE Monitor	164
10.5.3. Starting scripts in EB GUIDE Monitor	165
10.5.4. Starting EB GUIDE Monitor using command line	168
10.6. Exporting an EB GUIDE model	169
10.6.1. Exporting an EB GUIDE model using EB GUIDE Studio	169
10.6.2. Exporting an EB GUIDE model using command line	170
10.7. Changing the display language of EB GUIDE Studio	170
10.8. Configuring profiles	171
10.8.1. Cloning a profile	171
10.8.2. Adding a library	172
10.8.3. Configuring a scene	174
10.9. Exporting and importing language-dependent texts	175
10.9.1. Exporting language-dependent texts	175
10.9.2. Importing language-dependent texts	176
10.9.2.1. Importing language-dependent texts using EB GUIDE Studio	176
10.9.2.2. Importing language-dependent texts using command line	177
11. Tutorials	178
11.1. Tutorial: Adding a dynamic state machine	178
11.2. Tutorial: Modeling button behavior with EB GUIDE Script	186
11.3. Tutorial: Modeling a path gesture	193
11.4. Tutorial: Creating a list with dynamic content	196
11.5. Tutorial: Making an ellipse move across the screen	203
11.6. Tutorial: Adding a language-dependent text to a datapool item	207
11.7. Tutorial: Working with a 3D graphic	210
12. References	216
12.1. Android events	216
12.2. Datapool items	217
12.3. Data types	217
12.3.1. Mesh	217
12.3.2. Boolean	217
12.3.3. Color	218
12.3.4. Conditional script	218

12.3.5. Float	219
12.3.6. Font	219
12.3.7. Image	219
12.3.8. Integer	220
12.3.9. List	220
12.3.10. String	221
12.4. EB GUIDE Script	222
12.4.1. EB GUIDE Script keywords	222
12.4.2. EB GUIDE Script operator precedence	223
12.4.3. EB GUIDE Script standard library	223
12.4.3.1. EB GUIDE Script functions A	224
12.4.3.1.1. abs	224
12.4.3.1.2. absf	224
12.4.3.1.3. acosf	224
12.4.3.1.4. animation_before	224
12.4.3.1.5. animation_beyond	225
12.4.3.1.6. animation_cancel	225
12.4.3.1.7. animation_cancel_end	225
12.4.3.1.8. animation_cancel_reset	225
12.4.3.1.9. animation_pause	226
12.4.3.1.10. animation_play	226
12.4.3.1.11. animation_reverse	226
12.4.3.1.12. animation_running	226
12.4.3.1.13. animation_set_time	227
12.4.3.1.14. asinf	227
12.4.3.1.15. atan2f	227
12.4.3.1.16. atan2i	227
12.4.3.1.17. atanf	228
12.4.3.2. EB GUIDE Script functions C - H	228
12.4.3.2.1. ceil	228
12.4.3.2.2. changeDynamicStateMachinePriority	228
12.4.3.2.3. character2unicode	229
12.4.3.2.4. clearAllDynamicStateMachines	229
12.4.3.2.5. color2string	229
12.4.3.2.6. cosf	229
12.4.3.2.7. deg2rad	230
12.4.3.2.8. expf	230
12.4.3.2.9. float2string	230
12.4.3.2.10. floor	230
12.4.3.2.11. focusNext	231
12.4.3.2.12. focusPrevious	231
12.4.3.2.13. format_float	231

12.4.3.2.14. format_int	232
12.4.3.2.15. getLineCount	233
12.4.3.2.16. getTextHeight	233
12.4.3.2.17. getTextLength	233
12.4.3.2.18. getTextWidth	234
12.4.3.2.19. has_list_window	234
12.4.3.2.20. hsba2color	234
12.4.3.3. EB GUIDE Script functions I - R	235
12.4.3.3.1. int2float	235
12.4.3.3.2. int2string	235
12.4.3.3.3. isDynamicStateMachineActive	235
12.4.3.3.4. language	236
12.4.3.3.5. localtime_day	236
12.4.3.3.6. localtime_hour	236
12.4.3.3.7. localtime_minute	236
12.4.3.3.8. localtime_month	237
12.4.3.3.9. localtime_second	237
12.4.3.3.10. localtime_weekday	237
12.4.3.3.11. localtime_year	237
12.4.3.3.12. log10f	237
12.4.3.3.13. logf	238
12.4.3.3.14. nearbyint	238
12.4.3.3.15. popDynamicStateMachine	238
12.4.3.3.16. powf	238
12.4.3.3.17. pushDynamicStateMachine	239
12.4.3.3.18. rad2deg	239
12.4.3.3.19. rand	239
12.4.3.3.20. shutdown	239
12.4.3.3.21. rgba2color	240
12.4.3.3.22. round	240
12.4.3.4. EB GUIDE Script functions S - W	240
12.4.3.4.1. seed_rand	240
12.4.3.4.2. sinf	240
12.4.3.4.3. skin	241
12.4.3.4.4. sqrtf	241
12.4.3.4.5. string2float	241
12.4.3.4.6. string2int	242
12.4.3.4.7. string2string	242
12.4.3.4.8. substring	242
12.4.3.4.9. system_time	243
12.4.3.4.10. system_time_ms	243
12.4.3.4.11. tanf	243

12.4.3.4.12. trace_dp	243
12.4.3.4.13. trace_string	244
12.4.3.4.14. transformToScreenX	244
12.4.3.4.15. transformToScreenY	244
12.4.3.4.16. transformToWidgetX	244
12.4.3.4.17. transformToWidgetY	245
12.4.3.4.18. trunc	245
12.4.3.4.19. widgetGetChildCount	245
12.5. Events	246
12.6. model.json configuration file	246
12.6.1. Example model.json in EB GUIDE Studio	252
12.7. platform.json configuration file	255
12.7.1. Example platform.json in EB GUIDE Studio	257
12.8. Scenes	258
12.9. Touch screen types supported by EB GUIDE GTF	259
12.10. Widgets	260
12.10.1. View	260
12.10.2. Basic widgets	261
12.10.2.1. Container	261
12.10.2.2. Ellipse	262
12.10.2.3. Image	262
12.10.2.4. Instantiator	263
12.10.2.5. Label	263
12.10.2.6. Rectangle	264
12.10.3. Animations	264
12.10.3.1. Animation	265
12.10.3.2. Constant curves	265
12.10.3.3. Fast start curves	266
12.10.3.4. Slow start curves	266
12.10.3.5. Quadratic curves	267
12.10.3.6. Sinus curves	267
12.10.3.7. Script curves	268
12.10.3.8. Linear curves	268
12.10.3.9. Linear interpolation curves	269
12.10.4. 3D widgets	270
12.10.4.1. Scene graph	270
12.10.4.2. Scene graph node	270
12.10.4.3. Camera	271
12.10.4.4. Directional light	271
12.10.4.5. Material	271
12.10.4.6. Mesh	272
12.10.4.7. Point light	272

12.10.4.8. Spot light	273
12.11. Widget features	273
12.11.1. Common	273
12.11.1.1. Child visibility selection	273
12.11.1.2. Enabled	274
12.11.1.3. Focused	274
12.11.1.4. Multiple lines	274
12.11.1.5. Pressed	275
12.11.1.6. Selected	275
12.11.1.7. Selection group	276
12.11.1.8. Spinning	276
12.11.1.9. Text truncation	277
12.11.1.10. Touched	277
12.11.2. Effect	278
12.11.2.1. Border	278
12.11.2.2. Coloration	279
12.11.3. Focus	279
12.11.3.1. Auto focus	280
12.11.3.2. User-defined focus	280
12.11.4. Gestures	281
12.11.4.1. Flick gesture	281
12.11.4.2. Hold gesture	282
12.11.4.3. Long hold gesture	282
12.11.4.4. Path gestures	283
12.11.4.4.1. Gesture IDs	283
12.11.4.5. Pinch gesture	284
12.11.4.6. Rotate gesture	285
12.11.5. Input handling	286
12.11.5.1. Gestures	286
12.11.5.2. Key pressed	286
12.11.5.3. Key released	287
12.11.5.4. Key status changed	287
12.11.5.5. Key unicode	287
12.11.5.6. Move in	288
12.11.5.7. Move out	288
12.11.5.8. Move over	289
12.11.5.9. Moveable	289
12.11.5.10. Rotary	289
12.11.5.11. Touch lost	290
12.11.5.12. Touch move	290
12.11.5.13. Touch pressed	291
12.11.5.14. Touch released	291

12.11.5.15. Touch status changed	292
12.11.6. Layout	292
12.11.6.1. Absolute layout	292
12.11.6.2. Box layout	293
12.11.6.3. Flow layout	293
12.11.6.4. Grid layout	294
12.11.6.5. Layout margins	295
12.11.6.6. List layout	295
12.11.6.7. Scale mode	296
12.11.7. List management	297
12.11.7.1. Line index	297
12.11.7.2. List index	297
12.11.7.3. Template index	297
12.11.7.4. Viewport	298
12.11.8. 3D	298
12.11.8.1. Camera viewport	298
12.11.8.2. Ambient texture	299
12.11.8.3. Diffuse texture	299
12.11.8.4. Emissive texture	300
12.11.8.5. Light map texture	301
12.11.8.6. Normal map texture	302
12.11.8.7. Opaque texture	303
12.11.8.8. Reflection texture	304
12.11.8.9. Specular texture	304
12.11.9. Transformation	305
12.11.9.1. Pivot	306
12.11.9.2. Rotation	306
12.11.9.3. Scaling	306
12.11.9.4. Shearing	307
12.11.9.5. Translation	307
13. Installation	308
13.1. Background information	308
13.1.1. Restrictions	308
13.1.2. System requirements	308
13.2. Downloading EB GUIDE	309
13.3. Installing EB GUIDE	309
13.4. Uninstalling EB GUIDE	310
Glossary	312
Index	316

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.

► Demonstration

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

1.3. Typography and style conventions

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

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

WARNING



Source and kind of problem

What can happen to the software?

What are the consequences of the problem?

How does the user avoid the problem?

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

NOTE



Important information

Gives important information on a subject.

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

TIP



Helpful hints

Gives helpful hints

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

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

All default text is written in Arial Regular font.

Font	Description	Example
Arial italics	to emphasize new or important terms	The <i>basic building blocks</i> of a configuration are module configurations.
Arial boldface	for GUI elements and keyboard keys	1. In the Project drop-down list box, select Project_A. 2. Press the Enter key.
Monospaced font (Courier)	for file names, directory names and chapter names	Put your script in the <code>function_name\abcdi-rectory</code> .
Monospaced font (Courier)	for user input, code, and file directories	<pre>CC_FILES_TO_BUILD =(PROJECT_- PATH)\source\network\can_node.- c CC_FILES_TO_BUILD += \$(PROJECT_- PATH)\source\network\can_config.c</pre> <p>The module calls the <code>BswM_Dcm_RequestSessionMode()</code> function.</p> <p>For the project name, enter <code>Project_Test</code>.</p>
Square brackets []	to denote optional parameters; for command syntax with optional parameters	<code>insertBefore [<opt>]</code>
Curly brackets {}	to denote mandatory parameters; for command syntax with mandatory parameters	<code>insertBefore {<file>}</code>
Three dots ...	to indicate further parameters; for command syntax with multiple parameters	<code>insertBefore [<opt>...]</code>
A vertical bar	to indicate all available parameters; for command syntax in which you select one of the available parameters	<code>allowinvalidmarkup {on off}</code>



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 6.5
```

- ▶ 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 6.5\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 6.5\projects\
```

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

2. Safe and correct use

2.1. Intended use

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

2.2. Possible misuse

WARNING**Possible misuse and liability**

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

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

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

3. Support

EB GUIDE support is available in the following ways.

- ▶ For 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.

EB GUIDE TF executes an EB GUIDE model created in EB GUIDE Studio. 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 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 to be displayed and further internal information. Datapool items store and exchange data.

Event system

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

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

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

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

4.2.5. Exporting the EB GUIDE model

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

4.3. EB GUIDE TF

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

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 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. 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

NOTE



Default window layout

All instructions and screenshots of this user manual use the default window layout. If you want to follow the instructions, we recommend to reset the EB GUIDE Studio window to default layout by selecting **Layout > Reset to default layout**.

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

Approximate duration: 20 minutes.

5.1. Starting EB GUIDE



Starting EB GUIDE

Prerequisite:

- EB GUIDE is installed.

Step 1

In the Windows **Start** menu, 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.

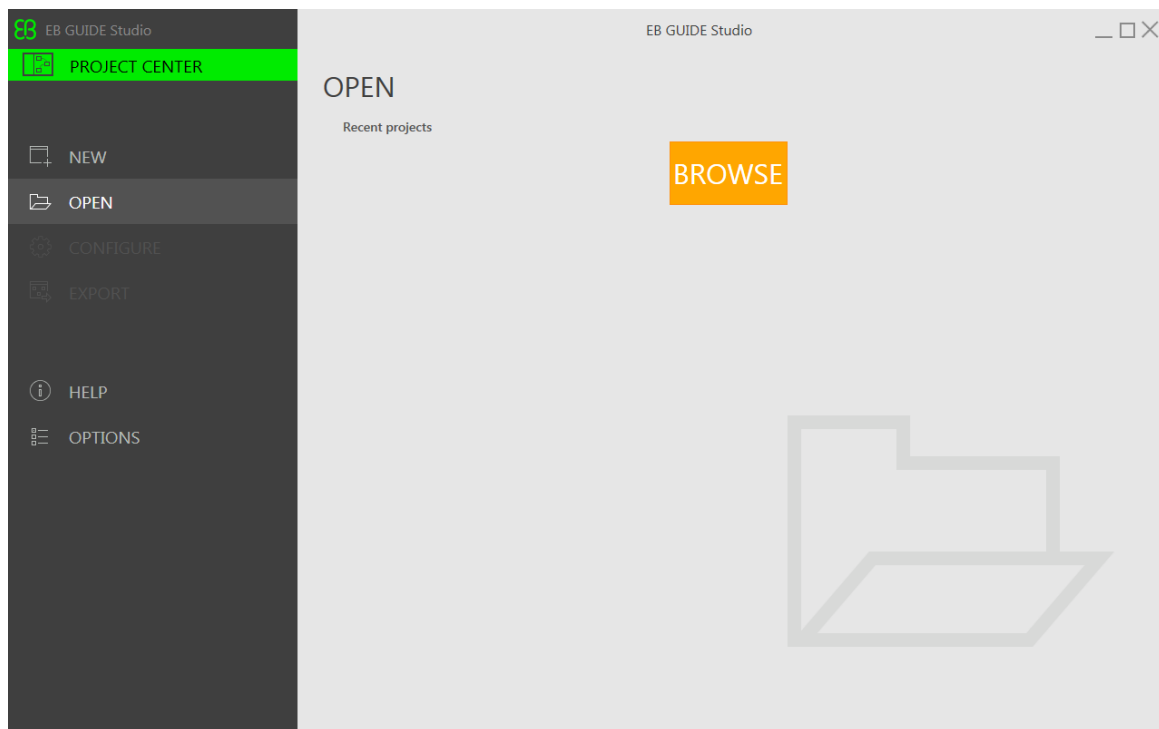


Figure 5.1. Project center

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 of the project center, click **New**.

Step 2

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

Step 3

Enter the project name `MyProject`.

Step 4

Click **Create**.

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

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

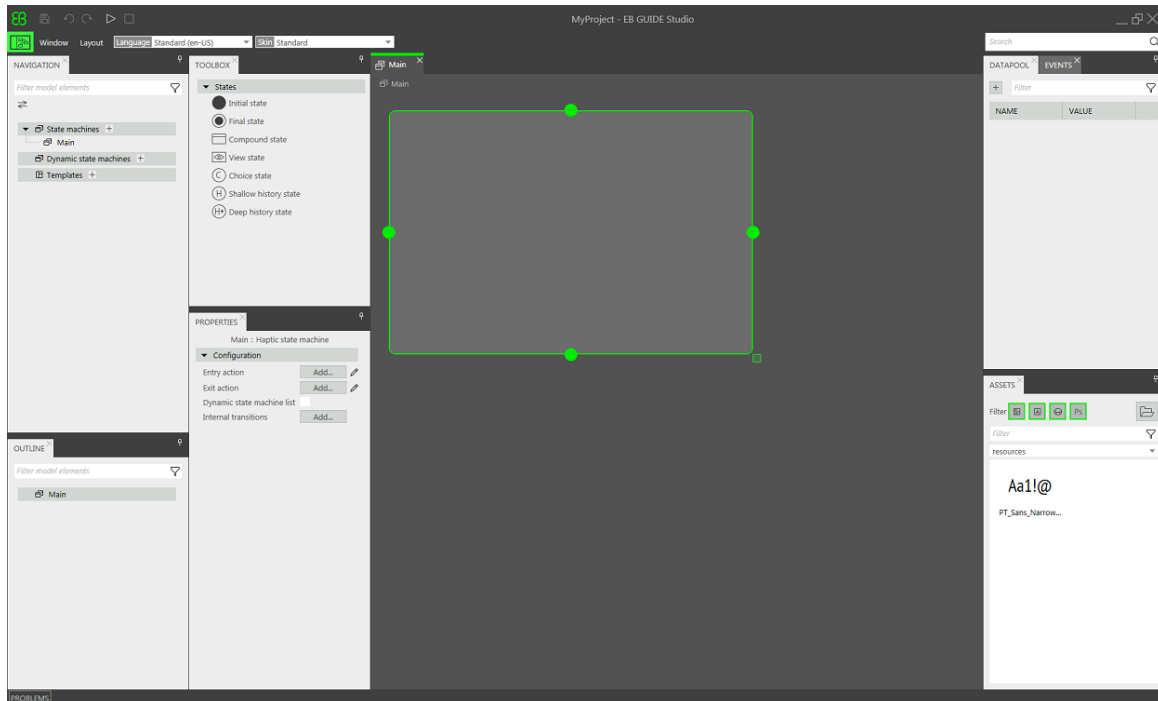


Figure 5.2. Project editor with **Main** state machine

5.3. Modeling HMI behavior

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

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



Adding states to the state machine

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

Prerequisite:

- The project `MyProject` is created.

- The content area displays the **Main** state machine.

Step 1

Drag a view state from the **Toolbox** into the state machine.

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

Step 2

Repeat step 1.

View state 2 is added.

Step 3

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

Step 4

Drag a final state from the **Toolbox** into the state machine.

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

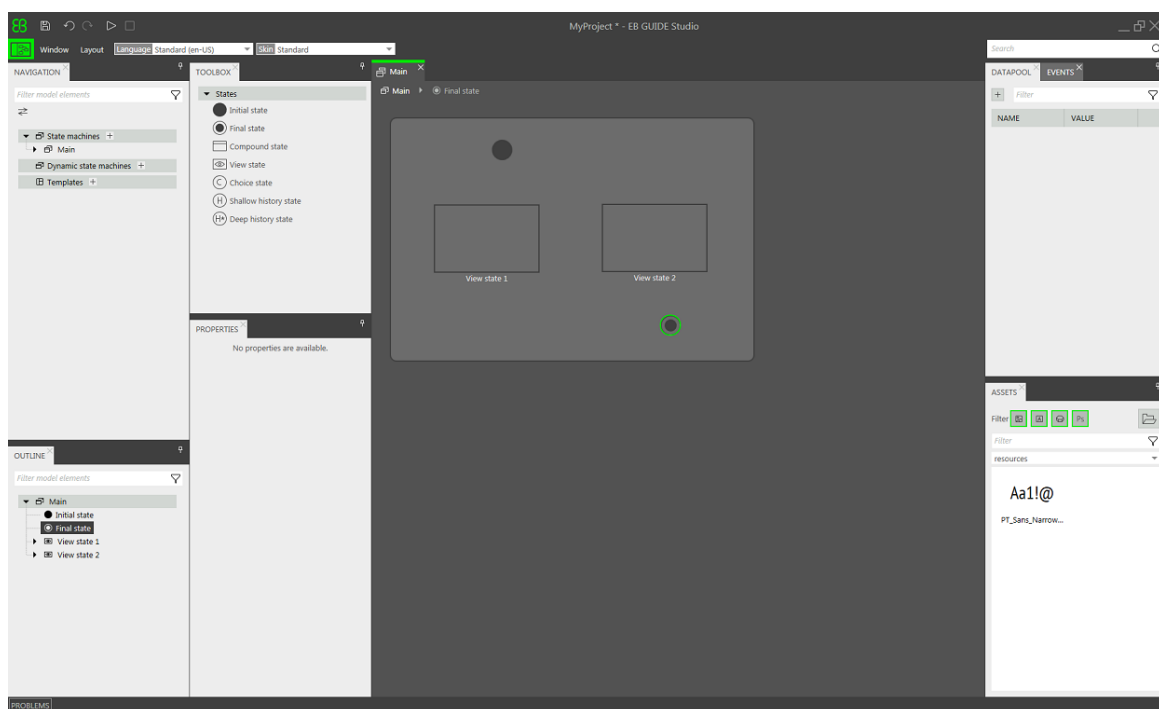


Figure 5.3. Project editor with states



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

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

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

Step 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:

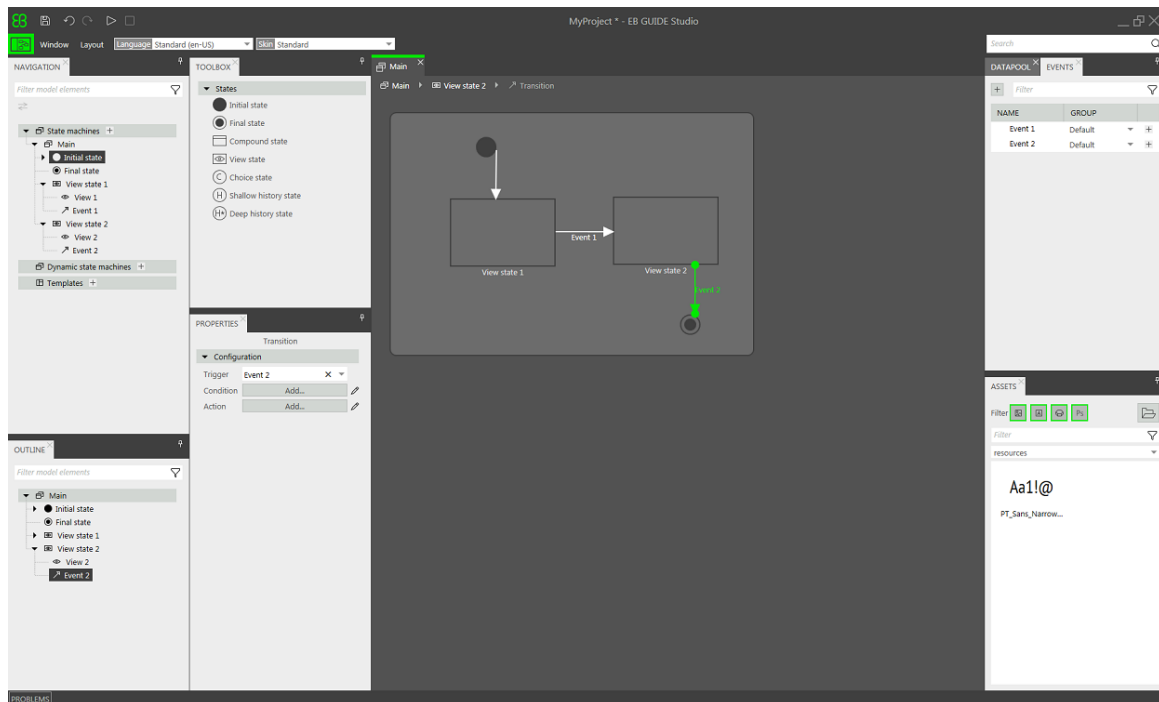


Figure 5.4. States linked by transitions with events

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:

- **View state 1** is added to the model.

Step 1

Double-click **View state 1**.

The content area displays **View 1**.



Adding a button to a view

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

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

Prerequisite:

- The content area displays **View 1**.

Step 1

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

Step 2

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

The **Widget features** dialog is displayed.

Step 3

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

Click **Accept**.

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

Step 4

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

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

Step 9

In the **Navigation** component, 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
}
```

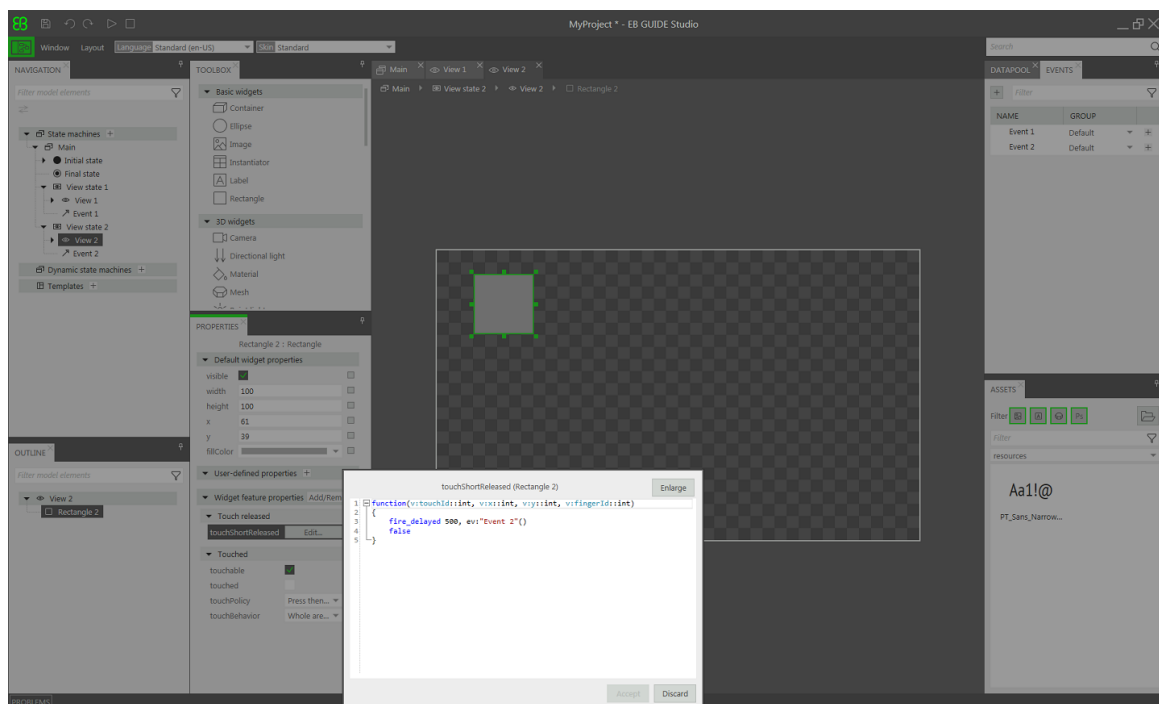


Figure 5.5. Widget property with an EB GUIDE Script

Step 12

Click **Accept**.

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

Step 13

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

5.5. Starting the simulation

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




Starting the simulation

Step 1


To save the project, click  in the command area.

Step 2

In the command area, click .

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

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

Then, **View 2** is displayed. A click on the blue rectangle in **View 2** terminates the state machine. This is because the click fires **Event 2** and **Event 2** executes the transition from **View state 2** to the final state. The simulation window remains open. To stop the simulation, click .

6. Background information

The topics in this chapter are sorted alphabetically.

NOTE



Default window layout

All instructions and screenshots of this user manual use the default window layout. If you want to follow the instructions, we recommend to reset the EB GUIDE Studio window to default layout by selecting **Layout > Reset to default layout**.

6.1. 3D graphics

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

6.1.1. Supported 3D graphic formats

Only the OpenGL ES version 2.0 or higher 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 containing a mesh and at least one material
- ▶ 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. If the 3D scene of the imported 3D graphic file contains animations, EB GUIDE Studio imports these animations using linear key value interpolation integer curve and linear key value interpolation float curve.

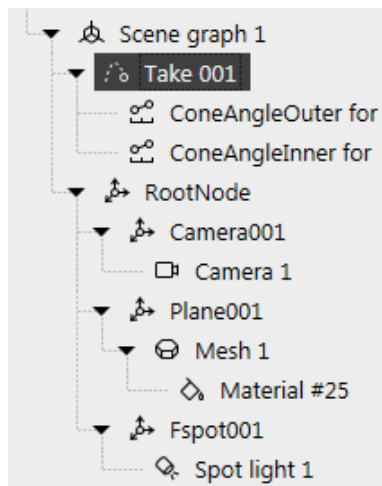


Figure 6.1. Example of a scene graph as displayed in the **Navigation** component

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 6.18, “Widgets”](#), [section 12.10.4, “3D widgets”](#), and [section 12.11.8, “3D”](#).

For instructions see [section 8.1.3.1, “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 that, 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 12.10.3, “Animations”](#).

For instructions see [section 8.6.1, “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 12.10.1, “View”](#).

For instructions see [section 8.6.2, “Animating a view transition”](#).

6.3. Application programming interface between application and model

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

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



Example 6.2.
Contents of an API

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

6.4. 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 application and updated by the reader application.

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

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.

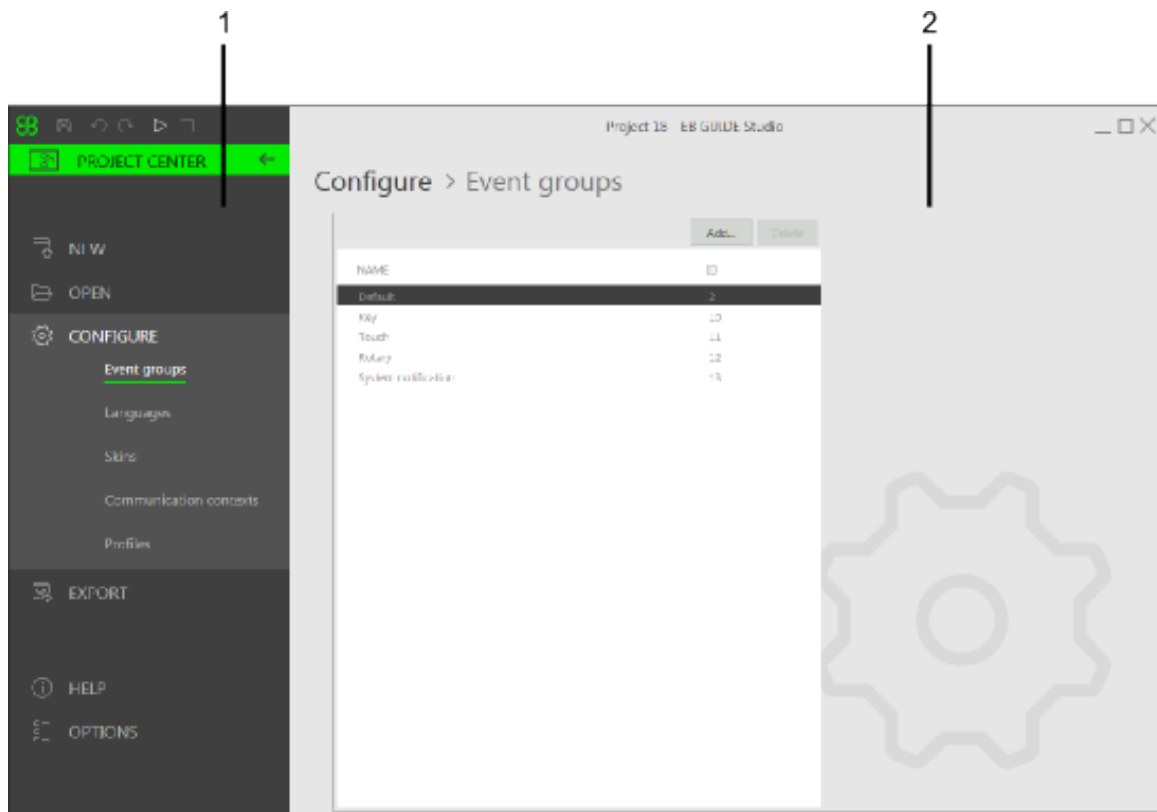


Figure 6.2. Project center with navigation area (1) and content area (2)

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 and components. All components of the project editor can either be docked or floating and placed at any position of the project editor except the content area.

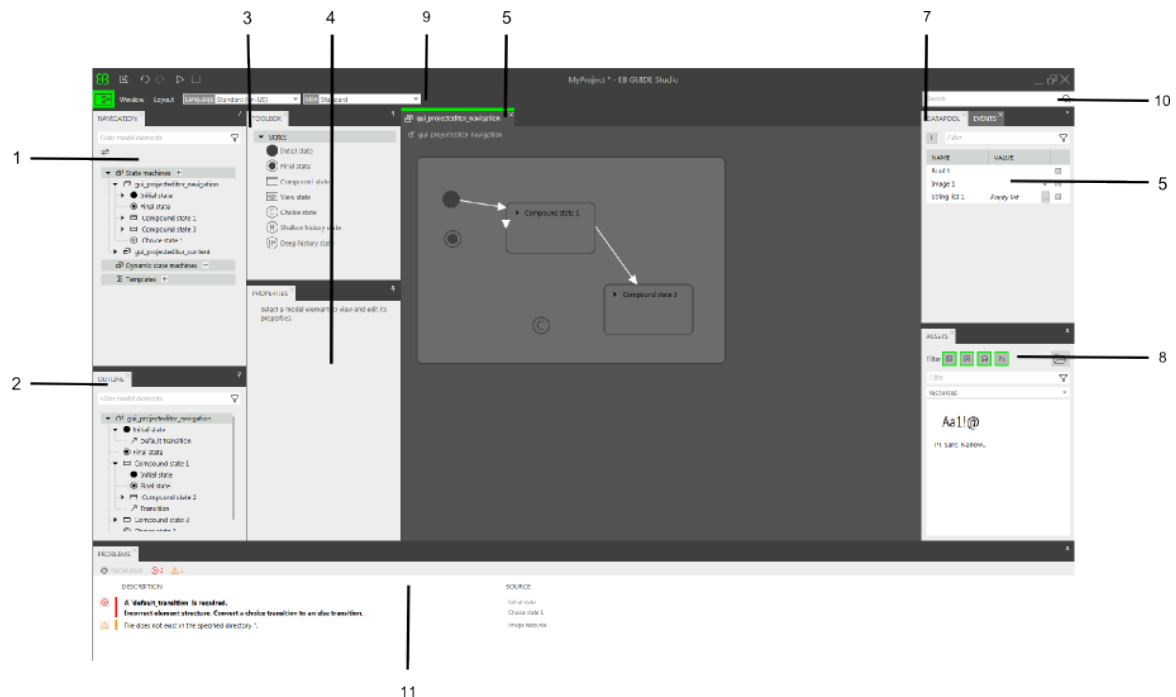


Figure 6.3. Project editor with its areas and components

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

6.5.2.1. Navigation component

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

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

It is also where you add elements to your EB GUIDE model, such as state machines, dynamic state machines, and templates. Elements from the **Toolbox** such as widgets and animations can be added via drag-and-drop.

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

Clicking a model element in the **Navigation** component 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.

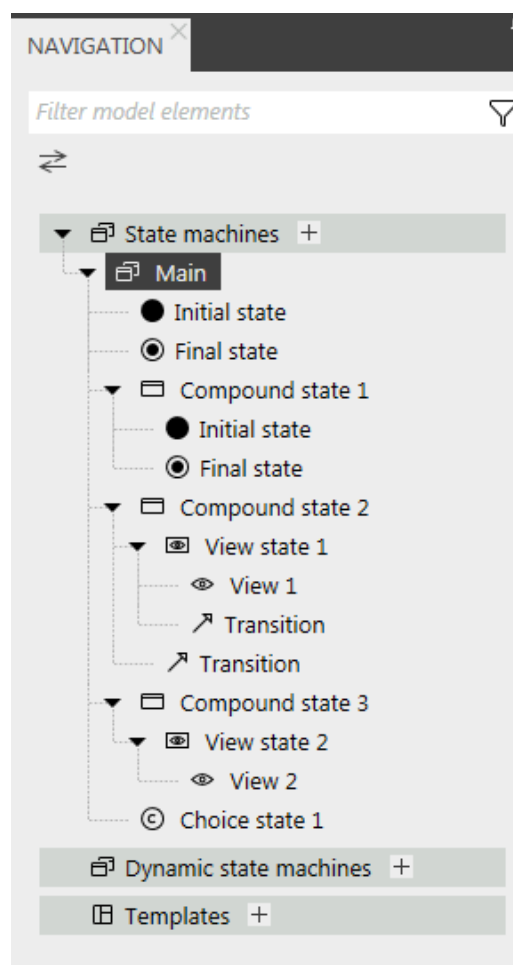


Figure 6.4. **Navigation** component in project editor

6.5.2.2. Outline component

Displays only the structure and model elements contained in the tree part selected in the **Navigation** component or in the editor component currently displayed in the content area.

NOTE



Filter box

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

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

6.5.2.3. Toolbox component

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

- ▶ If the content area displays a state machine, the **Toolbox** contains states you can add to the state machine.
- ▶ If the content area displays a view, the **Toolbox** contains widgets 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.

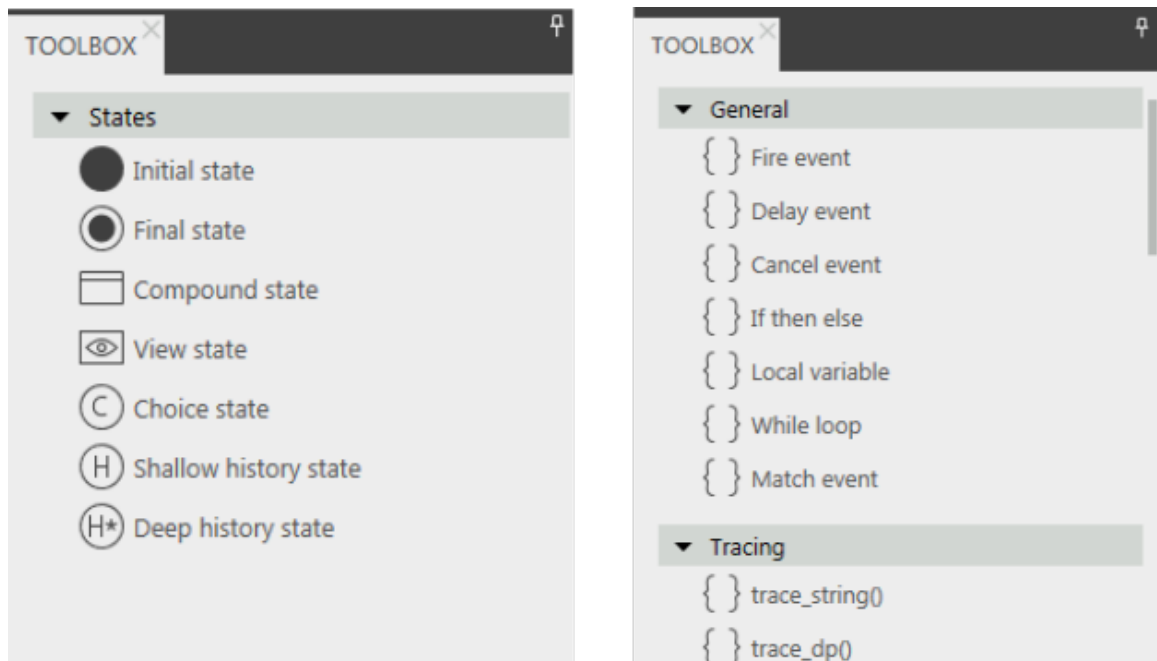


Figure 6.5. Toolbox in project editor

6.5.2.4. Properties component

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

Clicking a property 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.

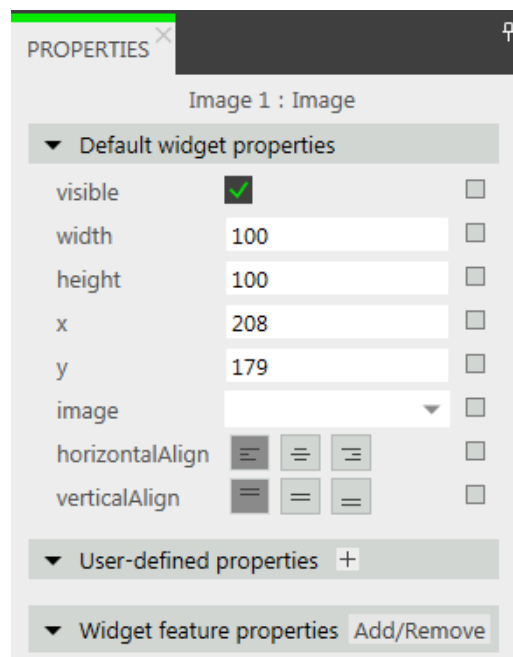


Figure 6.6. **Properties** component displaying properties of a widget

6.5.2.5. Content area

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

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.

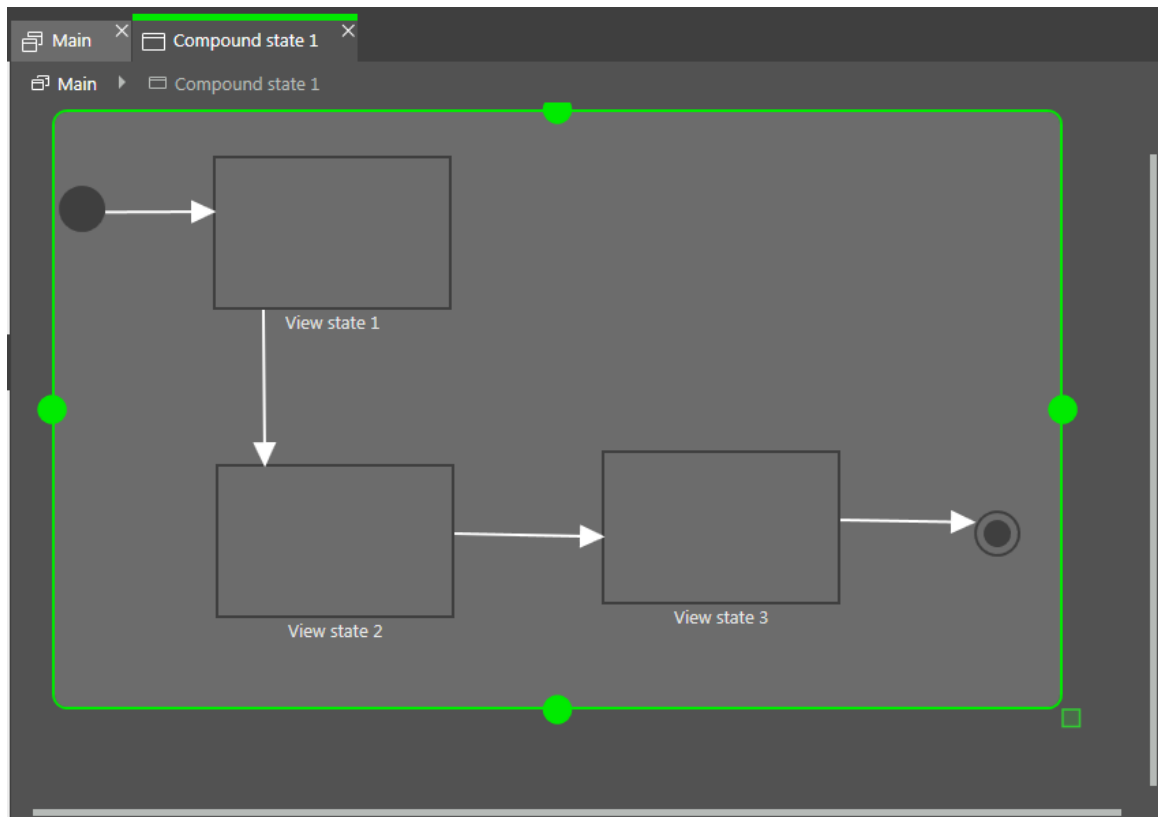


Figure 6.7. Content area in project editor

6.5.2.6. Events component

Here you can add events to your model and edit the properties such as **Name**, **Group**, **Type** and **Parameter name** in the event table.

NOTE



Filter box

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

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

6.5.2.7. Datapool component

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

NOTE



Filter box

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

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

6.5.2.8. Assets component

NOTE



Add component to layout

The **Assets** component is not shown in the default layout. To show this component select **Layout > Assets**.

Here you can add resources such as images, fonts, `.ebmesh` and `.psd` files. All resource located in the `$GUIDE_PROJECT_PATH\resources` directory and its subdirectories are displayed in the preview area of the component and can be added to the model via drag-and-drop.

NOTE




Filter box

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

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

6.5.2.9. Command area

In the command area, you find:

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

Search box

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

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

6.5.2.10. Problems component

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

6.5.3. Dockable component

All components of the project editor can be docked as tabs or undocked as floating components. You can drag a component as floating component to any part of the project center except the content area.

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

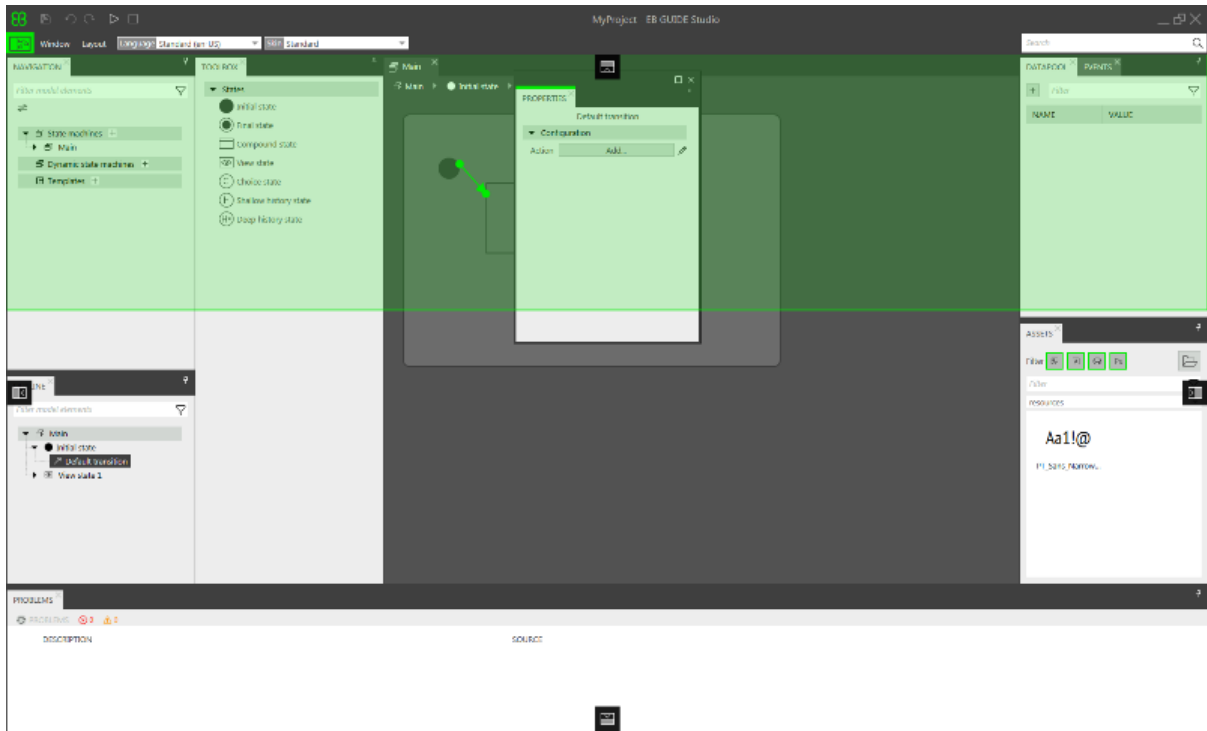


Figure 6.8. Docking control and live preview

NOTE



Default layout

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

NOTE



Auto-hide

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

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

6.5.4. EB GUIDE Monitor

EB GUIDE provides the tool EB GUIDE Monitor to observe and control an EB GUIDE model during the simulation. EB GUIDE Monitor includes mechanisms for the communication with datapool, the event system, and the state machines of the EB GUIDE model. EB GUIDE Monitor is started automatically in EB GUIDE Studio during the EB GUIDE model simulation. EB GUIDE Monitor can also be used as a stand-alone application.

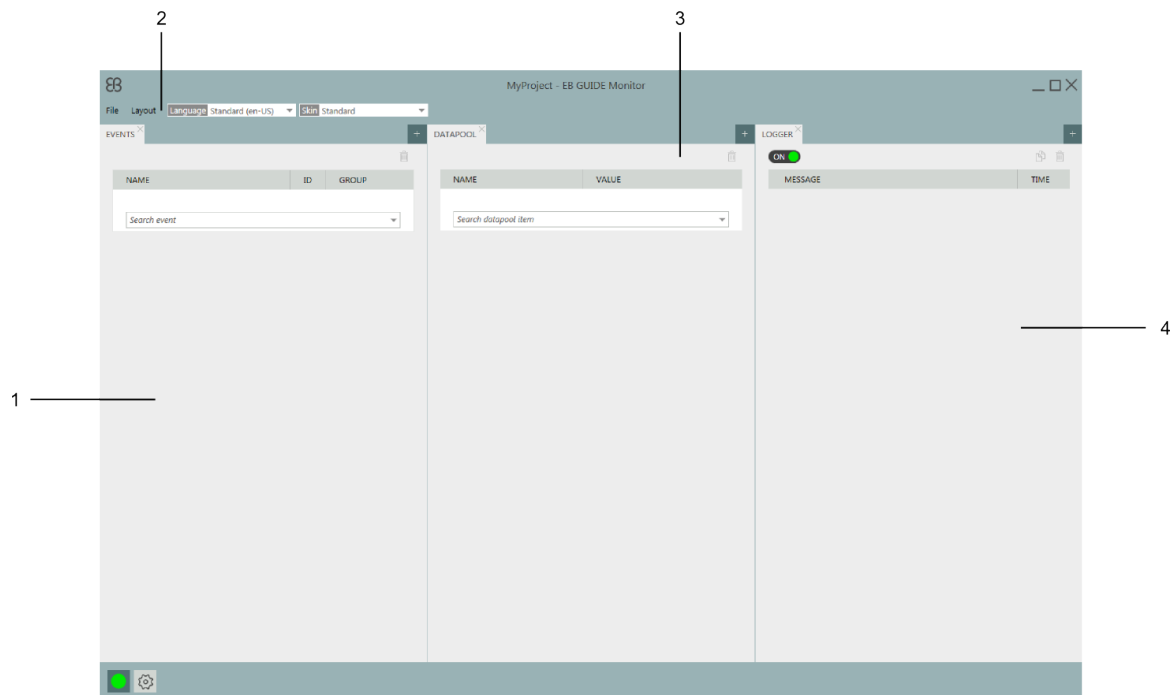


Figure 6.9. EB GUIDE Monitor

1 **Events** component

2 **Layout**

3 **Datapool** component


4 **Logger** component



EB GUIDE Monitor contains the following components:

- ▶ In the **Events** component you can search and fire events.
- ▶ In the **Datapool** component you can search for datapool items and change their values.
- ▶ In the **Logger** component all changes are tracked.
- ▶ In the **Scripting** component you can start scripts and see the output script messages. Note that the **Scripting** component is not in the default layout. You can add the component by clicking **Layout > Scripting**.

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

In the left bottom corner of the EB GUIDE Monitor window you find the following buttons for the connection status.

Button	Status
	EB GUIDE Monitor is connected.

Button	Status
	If you click the button, EB GUIDE Monitor disconnects.
	EB GUIDE Monitor is disconnected. If you click the button, EB GUIDE Monitor connects.
	EB GUIDE Monitor is disconnected. If you click the button, you can configure the connection settings of EB GUIDE Monitor.

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

For instructions see [section 10.5, “Working with EB GUIDE Monitor”](#).

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 writer and reader applications.

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

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

6.6.3. Windowed lists

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

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



Example 6.3. Windowed list

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

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

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

6.7. 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 applications.

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

6.8.2. 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

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

Typical EB GUIDE GTF extensions are:

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

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

Place the EB GUIDE GTF extension, including their third party libraries in 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 10.7, “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”](#).

NOTE



No skin support available

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

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 the translator. `.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 10.9, “Exporting and importing language-dependent texts”](#).

6.11. Skins

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

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

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

NOTE**No language support available**

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

For instructions see [section 8.5, “Working with skin support”](#).

6.12. 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`.

EB GUIDE supports the following types of resource files:

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

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

6.12.1. 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 8.1.2.4.1, “Changing the font of a label”](#).

6.12.2. 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), JPEG (*.jpg) and 9-patch images (*.9.png).

For instructions see [section 8.1.2.3, “Adding an image”](#).

6.12.2.1. 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.

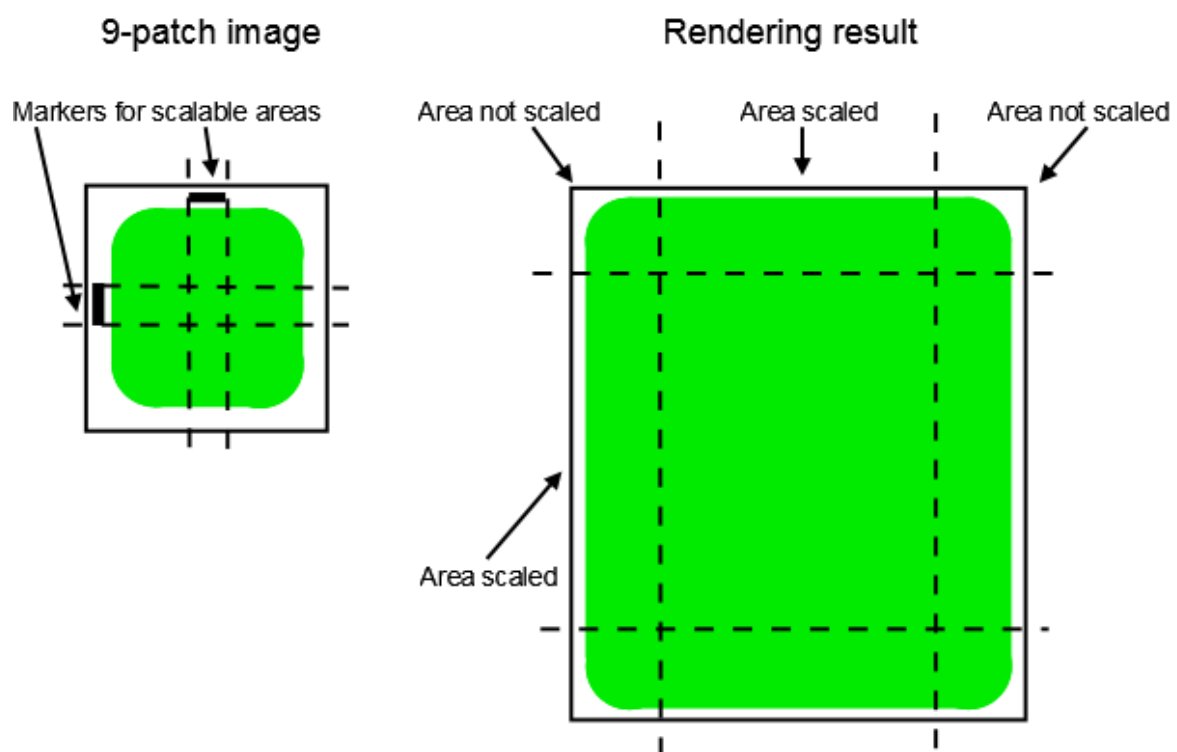


Figure 6.10. 9-patch example

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

- ▶ 9-patch processing works with the OpenGL ES version 2.0 or higher and the DirectX renderer only.
- ▶ 9-patch processing works with .png images only.

- ▶ 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.

For instructions see [section 8.1.2.3, “Adding an image”](#).

6.12.3. 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 8.1.3.1, “Adding a scene graph to a view”](#).

6.12.4. .psd file format

EB GUIDE Studio supports the `.psd` file format. After importing a `.psd` file in EB GUIDE Studio, a widget tree is created. The widget tree consists of containers and images that are created during the import from the layers of the `.psd` file. Note the following:

- ▶ If a layer in the `.psd` file was set to invisible, the check box next to the `visible` property of the corresponding container or image is cleared.
- ▶ If a layer in the `.psd` file has the transparency value set, after the import the **Coloration** widget feature is added to the corresponding image. The `alphaChannel` of the `colorationColor` property is set to the same transparency value as in the `.psd` file.

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

6.13. 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.

6.13.1. 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

6.13.2. 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!";
```

6.13.3. 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** component when you validate a project. Use this feature to mark all your open tasks and display them at a glance.

NOTE



Default comment for conditional scripts

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

6.13.4. 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.

6.13.5. 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
```

6.13.6. 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
```

6.13.7. Arithmetic and logic expressions

EB GUIDE Script supports the following arithmetic expressions:

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

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

NOTE



Availability of equality operators

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

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



Example 6.9. 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
```

6.13.8. 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.

6.13.9. 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;
    }
}
```

6.13.10. 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();
}
```

6.13.11. 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 {}
```

6.13.12. 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);
```

6.13.13. 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-link operator (`=>`) changes the link target of a datapool item. Link source can only be a datapool item that was already linked.

6.13.14. Widget properties

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

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



Example 6.15. 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 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;
}
```

6.13.15. 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.

6.13.16. 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;
}
```

6.13.17. 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);
```


6.13.18. 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”](#).

6.14. 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 6.11. 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.

6.15. Shortcuts, buttons and icons

6.15.1. 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
Ctrl+V	Paste the copied selection
Ctrl+Y	Redo
Ctrl+Z	Undo
Alt+F4	Close the active window
Shift+F1	Open user documentation for EB GUIDE TF
F1	Open user documentation for EB GUIDE Studio
Shift+F2	Rename the selected element in the Datapool or Events component and in all locations where the selected element is used, e.g. in EB GUIDE Script. Applicable to datapool items and events.
F2	Rename the selected element
F3	Find all occurrences of the selected element in the EB GUIDE model
F5	Start simulation
F6	Validate
Del	Delete the selected element from the Content area or the component
-	Collapse the selected element in the Navigation or Outline component
* and +	Expand the selected model element in the Navigation or Outline component

Shortcut	Description
Up/Down/Left/Right	Move the selected state or widget in the content area one pixel up, down, left, or right

6.15.1.1. Shortcuts in command line

The following table lists command line options available in EB GUIDE Studio for `Studio.Console.exe` and explains their meaning. Undefined command line options will be ignored.

The general syntax of a command line is as follows:

```
Studio.Console.exe <option> "project_name.ebguide"
```

Table 6.4. Command line options

Option	Description
-c <logfile dir>	Validates an EB GUIDE model and writes an logfile to the as logfile dir specified directory
-e <destination dir>	Exports an EB GUIDE model to the destination directory destination dir Use with the command line option -p, see an example below.
-h	Shows the help message
-l <language file>	Imports one language file that is saved as language file(.xliff) into an EB GUIDE model and creates a logfile
-m	Allows the migration of the project
-p <profile>	Uses the as profile specified profile during export



Example 6.23. Command line options

```
Studio.Console.exe -e "C:/temp/exported_project" -p "target_profile"
"project_name.ebguide" will export project_name.ebguide by using the profile target_pro-
file to the specified destination directory C:/temp/exported_project.
```

For instructions see the following:

















- ▶ [section 10.4.1.2, "Validating an EB GUIDE model using command line"](#)
- ▶ [section 10.6.2, "Exporting an EB GUIDE model using command line"](#)

► [section 10.9.2.2, “Importing language-dependent texts using command line”](#)

6.15.2. Buttons

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











Table 6.5. Buttons in EB GUIDE Studio

Button	Description
	Undo
	Redo
	Save
	Validate the project
	Start the simulation
	Stop the simulation
	Open the project center
	Open an additional editor
	Synchronize content area and Navigation component
	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: <div style="display: flex; flex-direction: column; gap: 5px;"> <div> Property is local.</div> <div> Property is linked to another property.</div> <div> Property is linked to a datapool item.</div> <div> Property value is equal to template value.</div> </div>
	Fire an event

6.15.3. 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
	Indicates a transition
	Indicates an internal transition
	Widget template: Indicates that a property is added to the widget template interface

6.16. State machines and states

6.16.1. 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:

6.16.1.1. Haptic state machine

Haptic state machine allows the specification of GUI.

6.16.1.2. Logic state machine

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

6.16.1.3. 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”](#).

6.16.2. 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

6.16.2.1. 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 6.12. Compound states

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

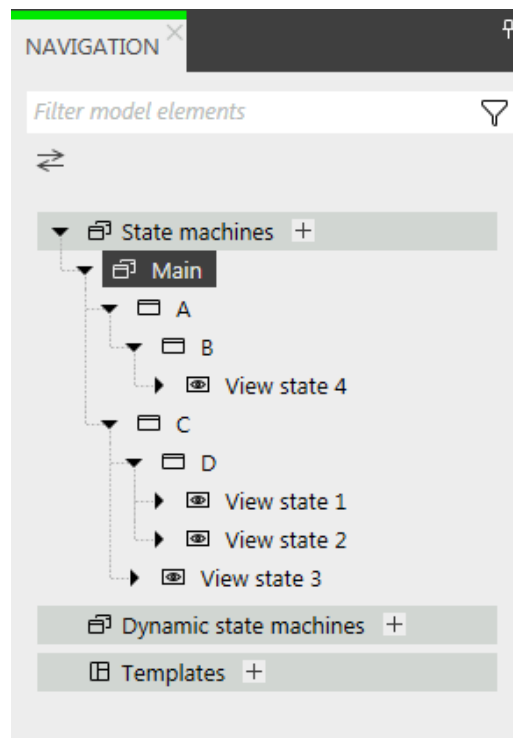


Figure 6.13. State hierarchy as a tree

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.

6.16.2.2. 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.

6.16.2.3. 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

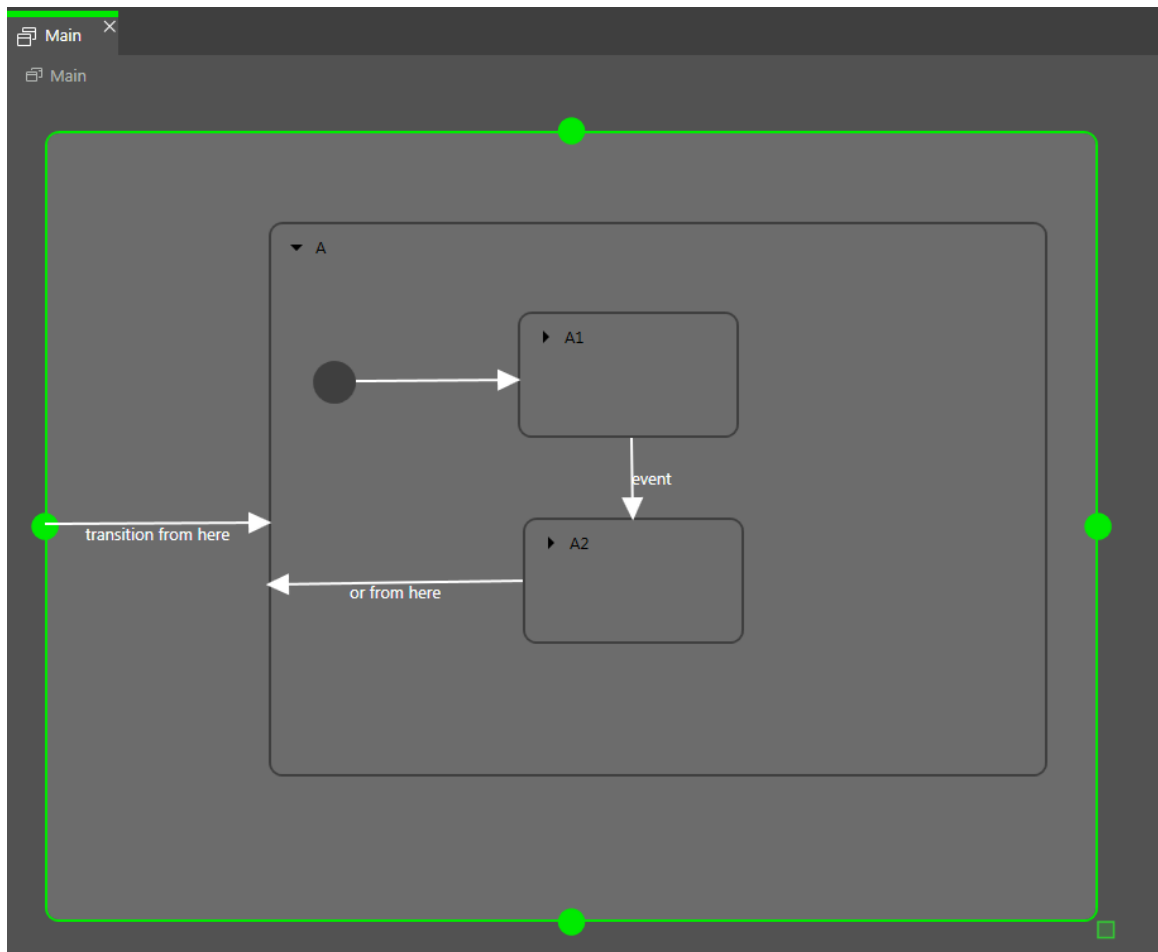


Figure 6.14. An example of an initial state

6.16.2.4. 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.

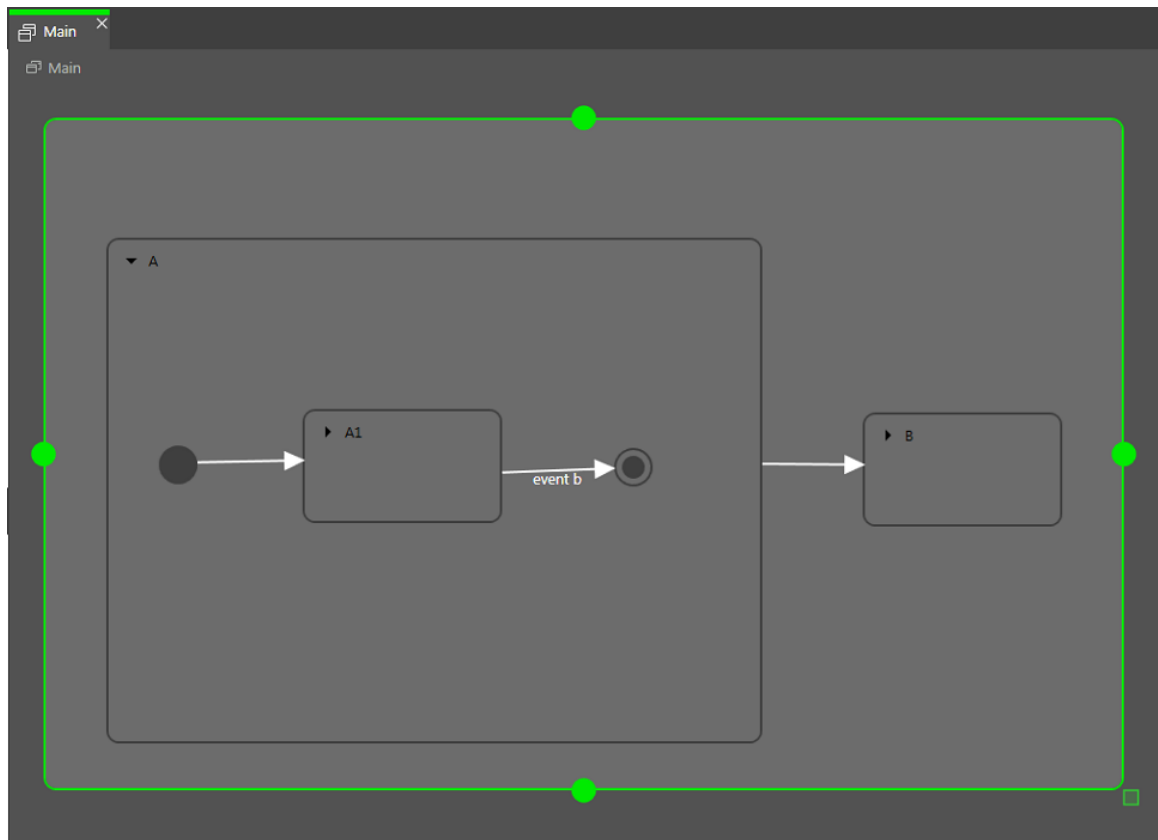


Figure 6.15. Final state usage in a compound state

6.16.2.5. 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.

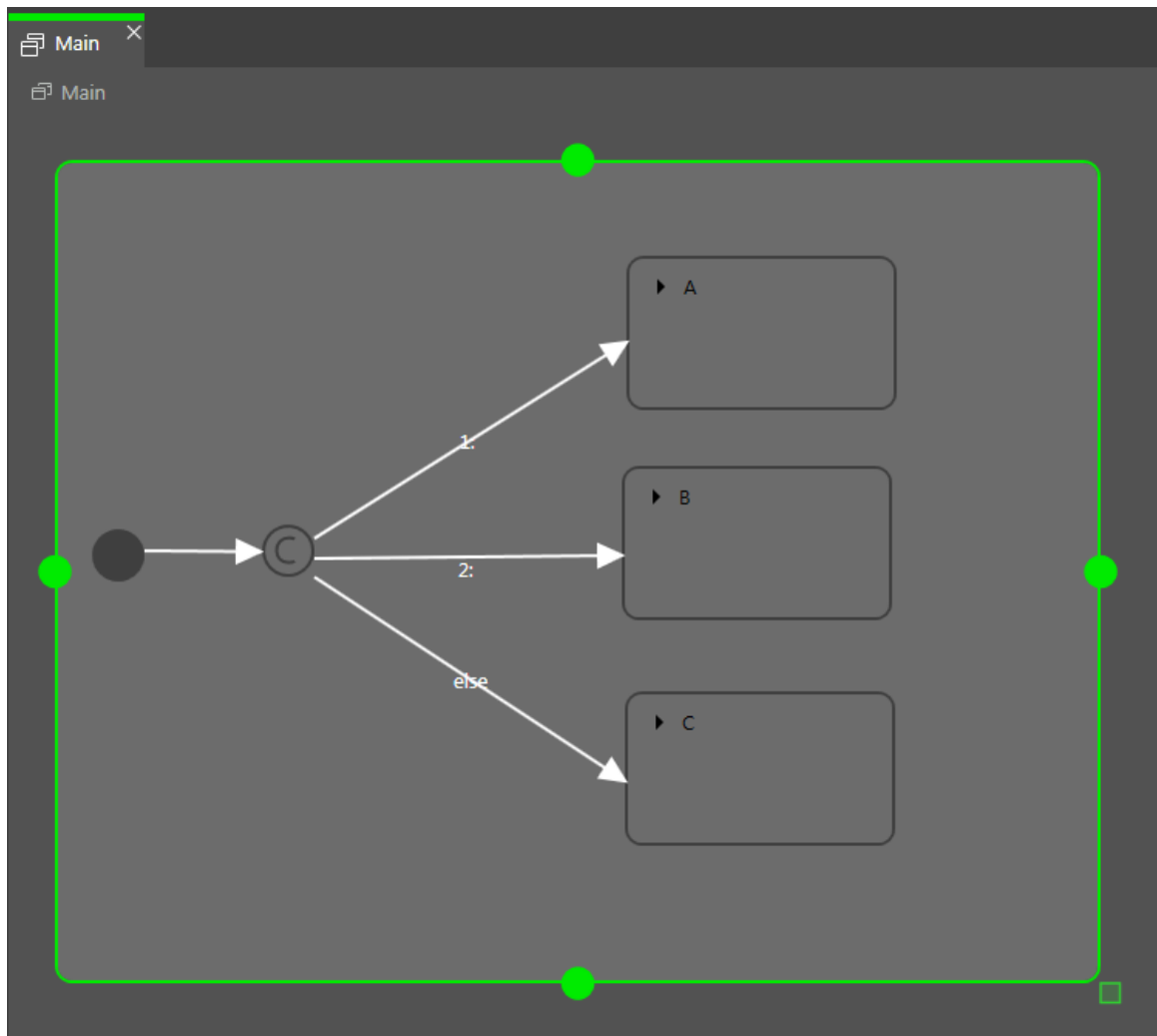


Figure 6.16. Choice state with incoming and outgoing transitions

6.16.2.6. 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 6.24. Shallow history state

A shallow history state can be used as follows.

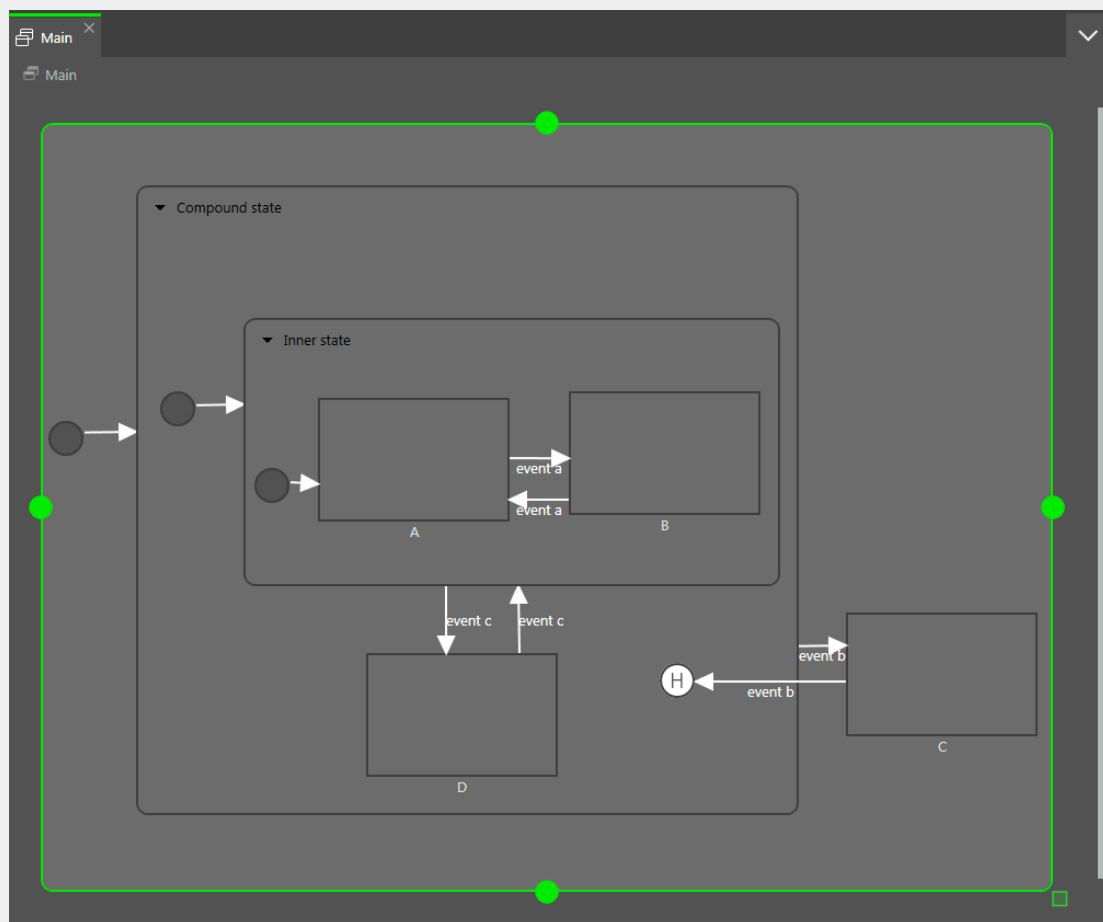


Figure 6.17. Shallow history state

- ▶ 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 6.25. Deep history state

A deep history state can be used as follows.

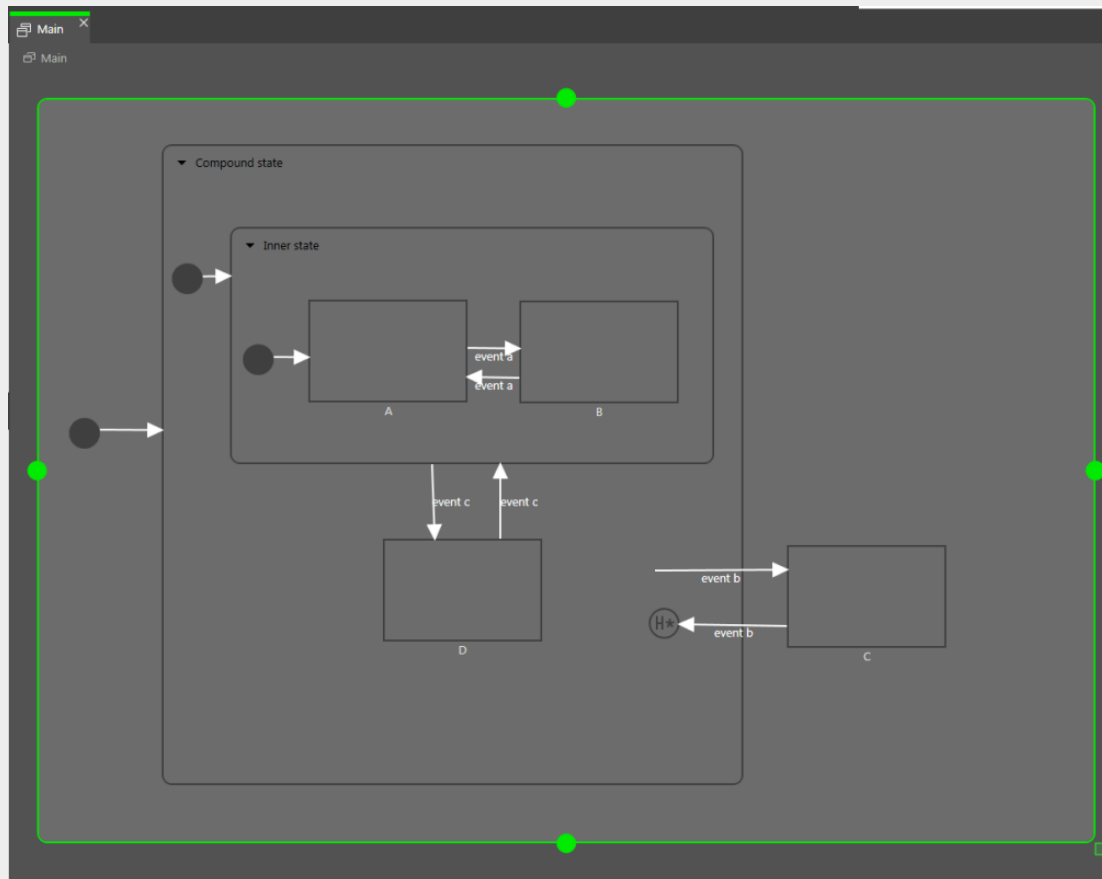


Figure 6.18. Deep history state

► 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.

6.16.3. 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

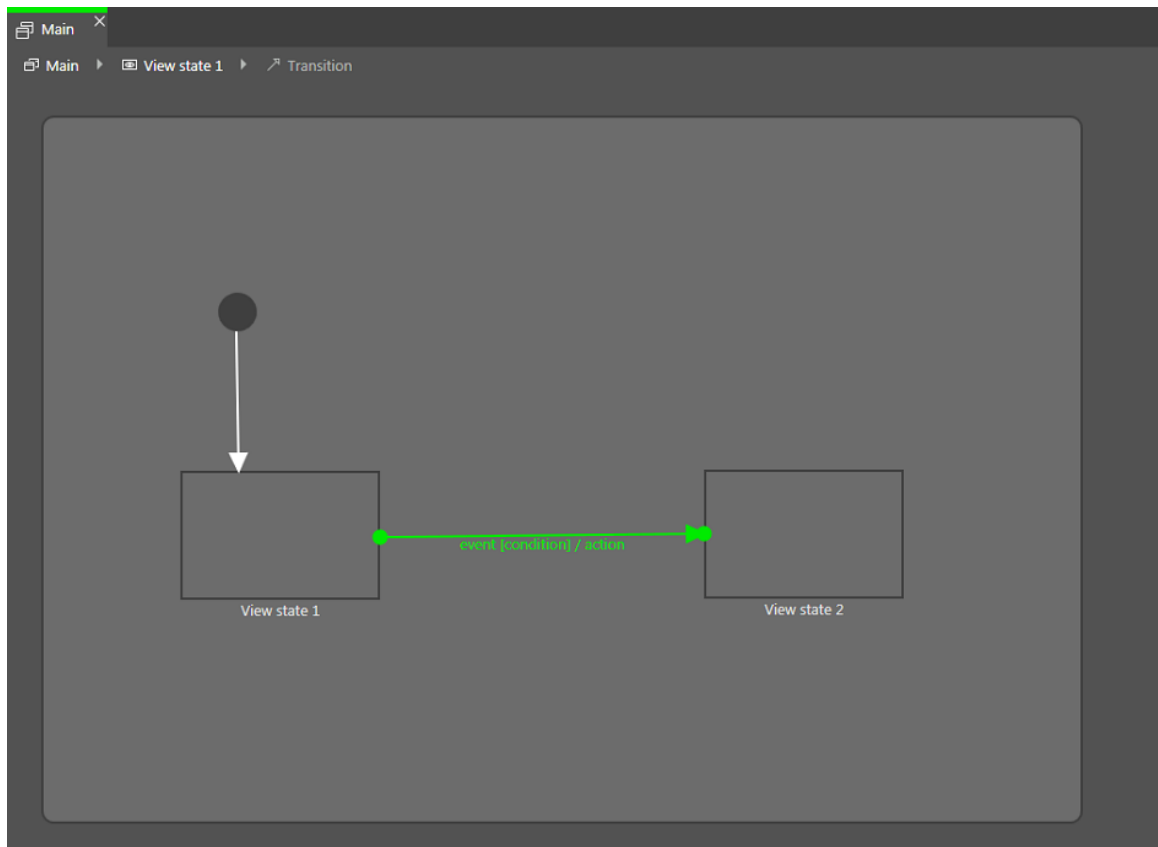


Figure 6.19. A transition

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 6.26.

Transition inheritance

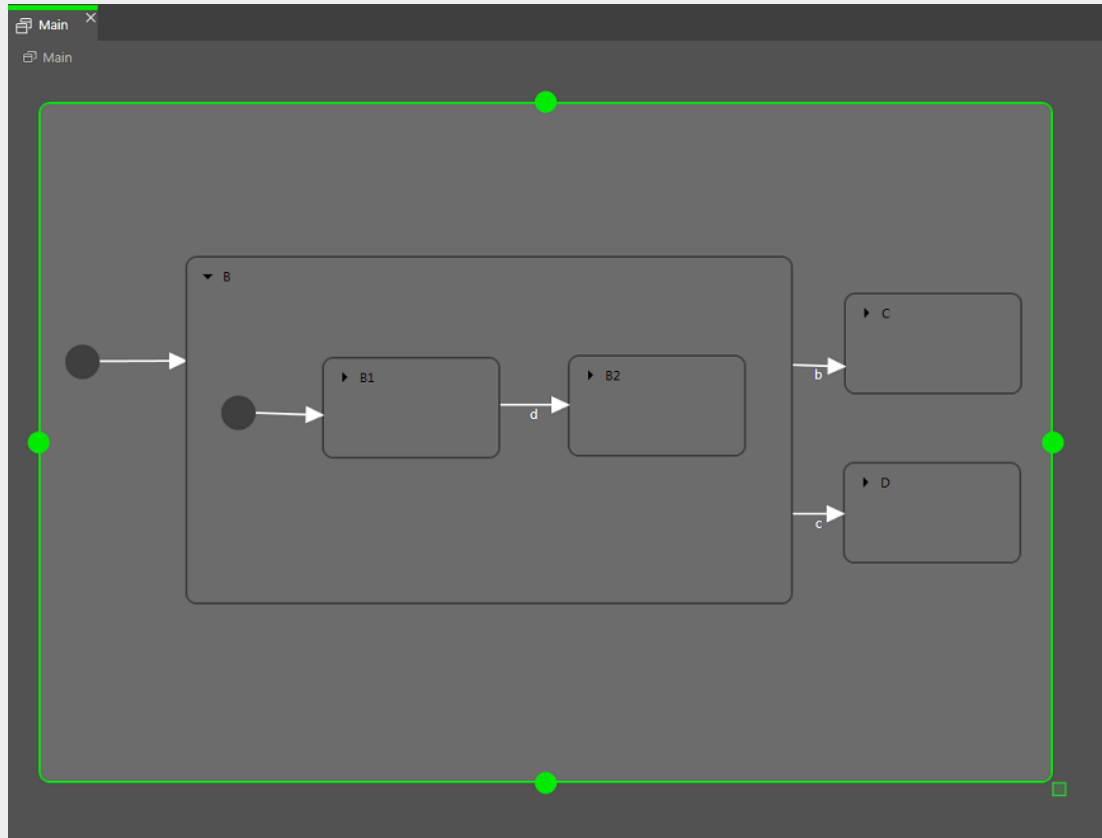


Figure 6.20. Transition inheritance

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 6.27.

Transition override

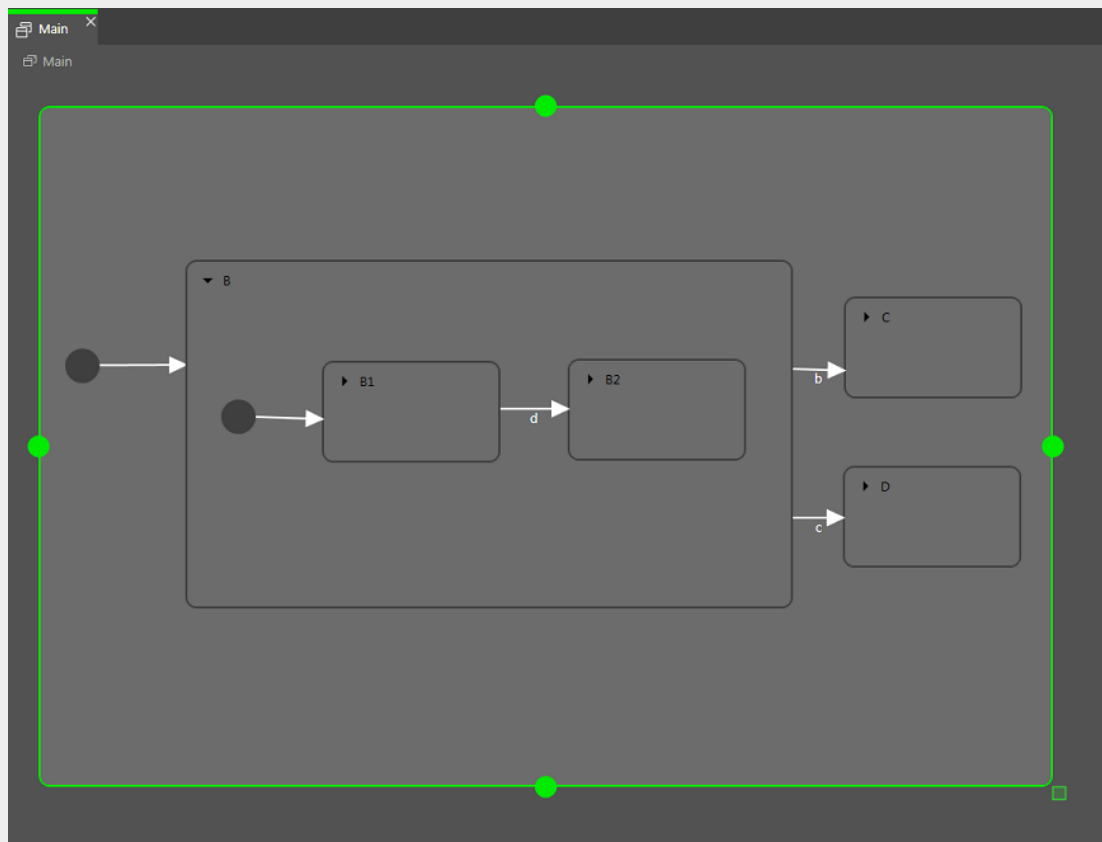


Figure 6.21. Transition override

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.

6.16.4. 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 6.28.

Executing a transition

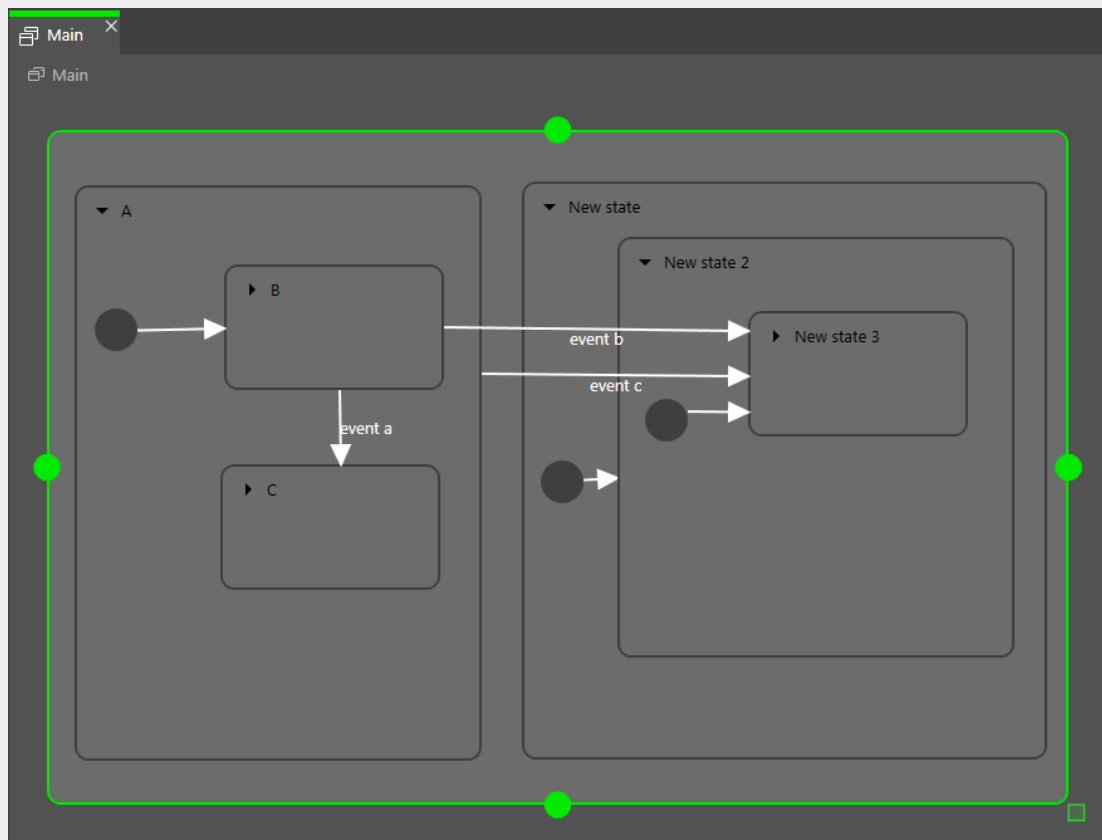


Figure 6.22. Executing a transition

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 6.29. Executing a transition

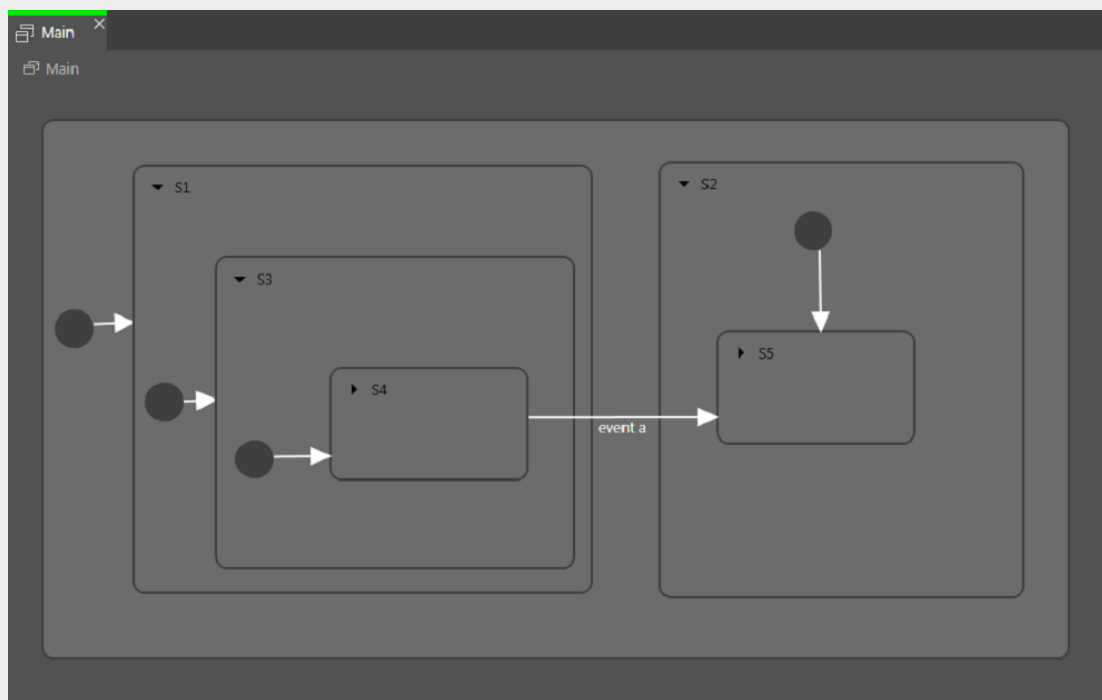


Figure 6.23. Executing a transition

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 6.30.

Executing a transition

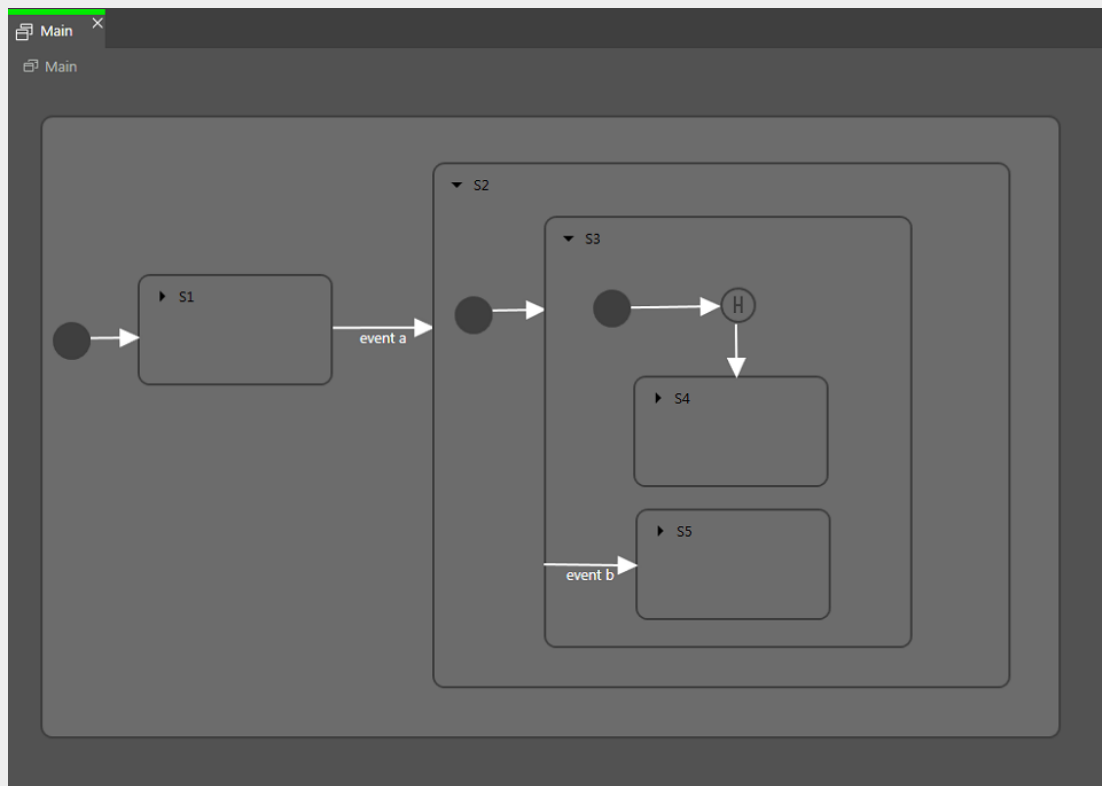


Figure 6.24. Executing a transition

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.

6.16.5. 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.

6.16.5.1. 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 ^a

^aEB GUIDE does not differentiate between external and local transitions.

6.16.5.2. Not supported elements

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

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

6.16.5.3. 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.

6.17. 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

6.17.1. 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.

6.17.2. 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”](#).

6.17.3. 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.

6.17.4. 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.

6.18. 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 an ellipse, an image, a label, and a rectangle.

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.

6.18.1. 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.

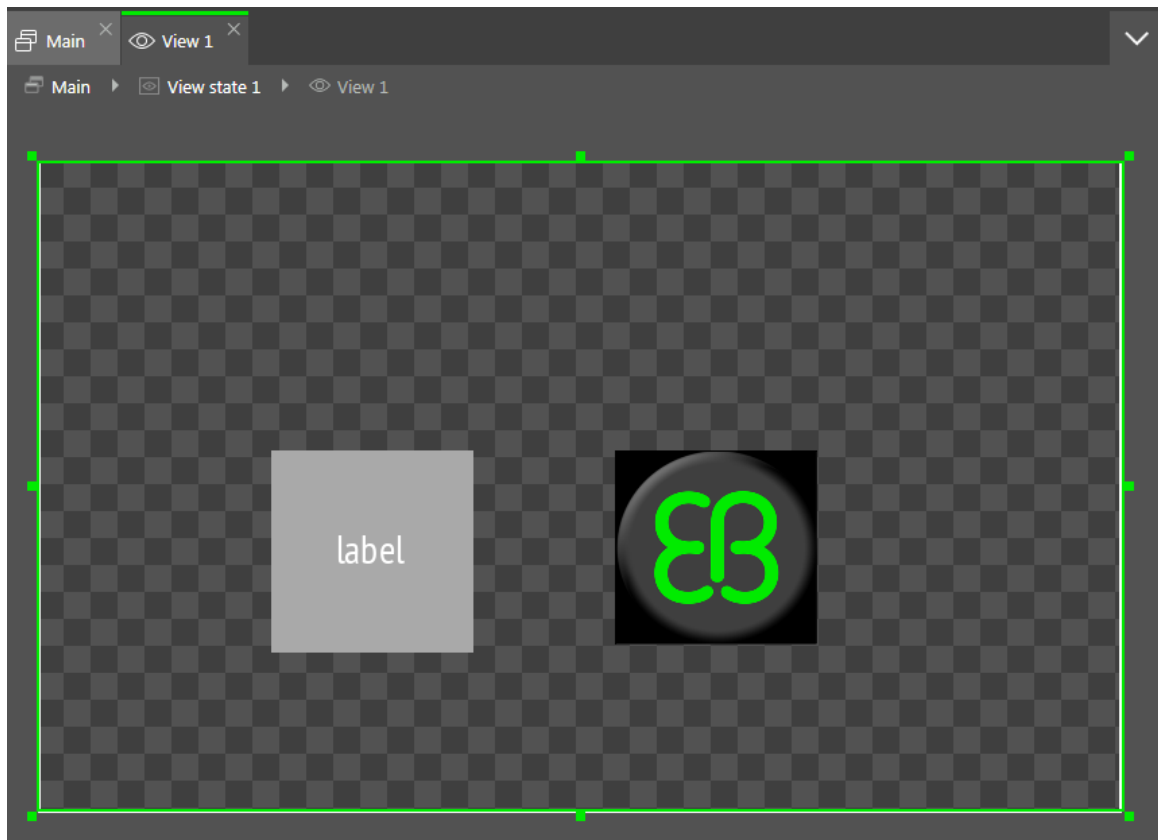


Figure 6.25. A view that contains a rectangle, a label, and an image

6.18.2. Widget categories

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

- ▶ Basic widgets

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

- ▶ 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 version 2.0 or higher or DirectX 11 renderer is required. Make sure that your graphic driver is compatible to the version of the renderer.

▶ 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 see our website <https://www.elektrobit.com/ebguide/learn/resources/>.

6.18.3. Widget properties

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

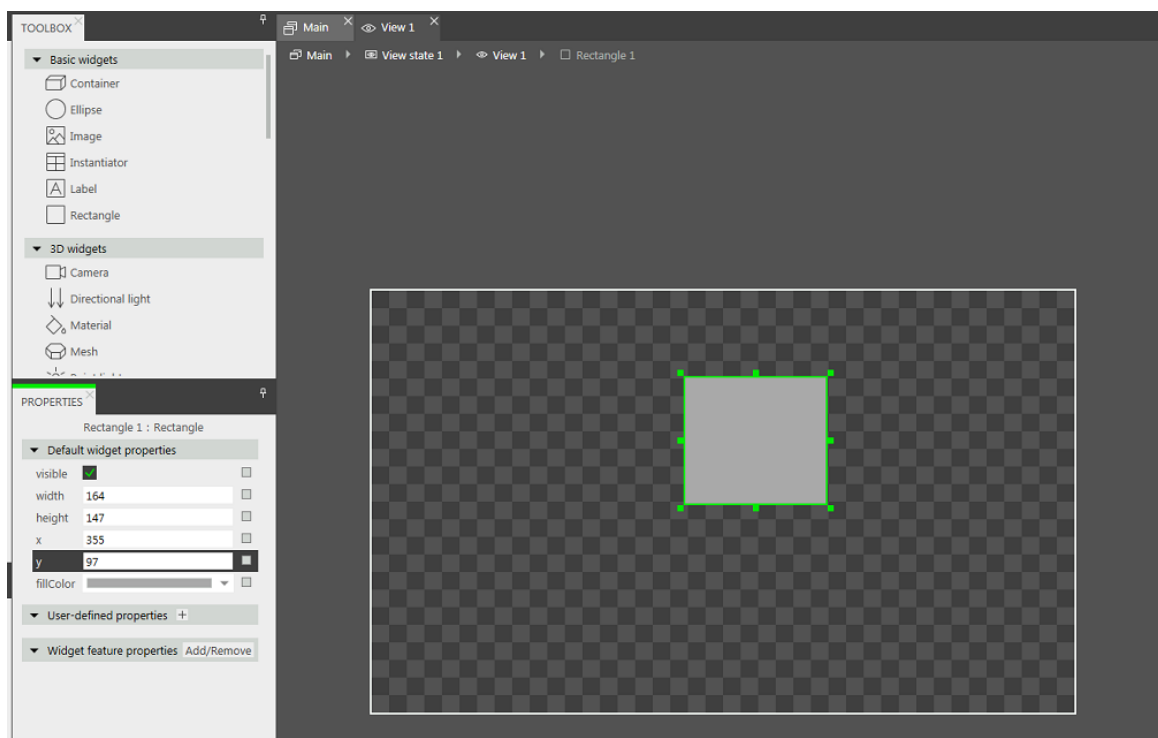


Figure 6.26. A rectangle and its properties

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 12.10, “Widgets”](#).
- ▶ User-defined widget properties are created by the modeler in addition to the default ones.
- ▶ Widget feature properties are created by EB GUIDE Studio when the modeler adds a widget feature to a widget. Widget feature properties are grouped by categories. Widget features add more functionality for the appearance and behavior of widgets.





Example 6.31.
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.

6.18.4. 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 6.32.
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 8.7, “Re-using a widget”](#).

6.18.5. Widget features

Widgets and widget templates can be extended in their functionality using *widget features*. Widget features have predefined widget properties. Widget features are grouped into categories.

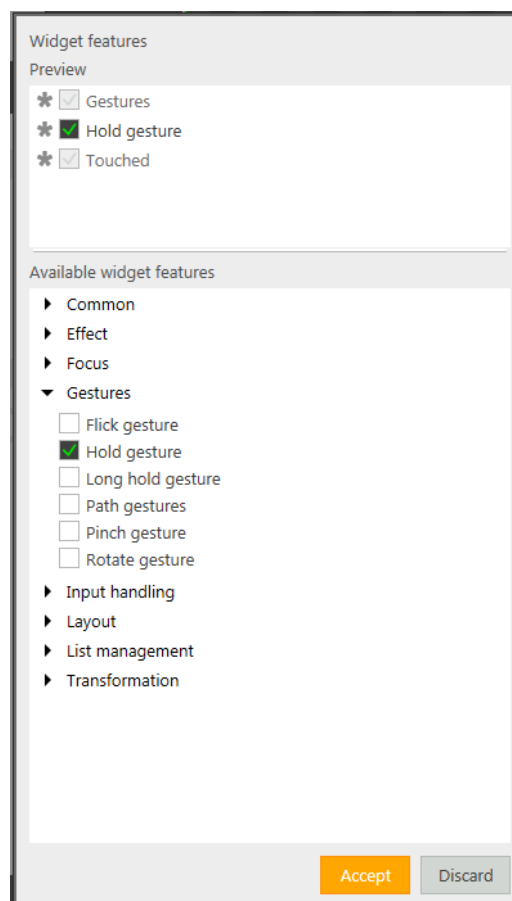


Figure 6.27. Widget features

If you add a widget feature to a widget template, any created widget template instance inherits the added widget feature. Note that you cannot add widget features to a widget template instance or to a template that was created from a template.

Restrictions for usage of widget features are as follows:

- ▶ Widget features do not have an inheritance hierarchy.
- ▶ It is not possible to add a widget feature more than once per widget.
- ▶ Some widget features are interdependent. This means, to add one widget feature, you have to add another, or widget features may exclude each other.
- ▶ Widget features can be restricted to a particular type of widgets.
- ▶ Widget features cannot be activated or deactivated during run-time.

By default all widget features are disabled. If you need a specific widget feature, you must add it to a widget.

For instructions see [section 8.3, “Extending a widget by widget features”](#). For a list of all widget features see [section 12.11, “Widget features”](#).

6.18.5.1. Focus widget feature category

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

The following two focus directions are available:

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

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

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

Auto focus

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

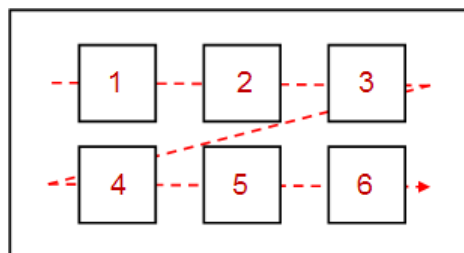


Figure 6.28. The policy of the **Auto focus** widget feature

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

User-defined focus

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

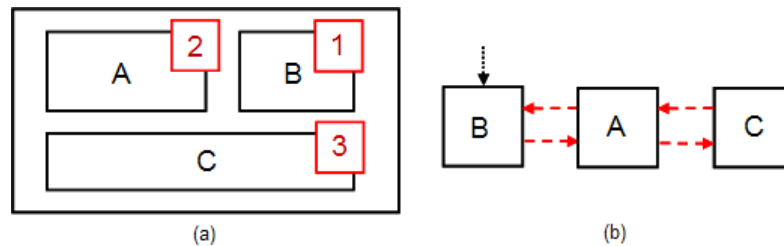


Figure 6.29. The policy of the **User-defined focus** widget feature

In [figure 6.29, “The policy of the User-defined focus widget feature”](#), (a) shows the view, while (b) shows the focus order. The order, in which the focus changes are processed, may differ from the widget tree structure.

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

6.18.5.2. List management widget feature category

The **Line index** and **Template index** widget features allow you to connect data, for example images, song titles, to the corresponding dynamically created line templates of an instantiator.

Line index

The **Line index** widget feature is used to customize the line templates of the instantiator widget. The **Line index** widget feature defines the unique position for each line of your list or table.



Example 6.33. Line index widget feature

If you want to model a list, you would expect that each entry of the list has a specific value that reflects the entry in a list property. To access a certain entry in a list, the instance of the line template needs to know which of the instantiator's child it is. The **Line index** widget feature adds the `lineIndex` property. While the instantiator creates the instances of line templates, it fills `lineIndex` with values: The index starts with zero for the first instance. If you have two elements in the instantiator, the second element receives the `lineIndex` value 1.

For instructions see [section 11.4, “Tutorial: Creating a list with dynamic content”](#).

Template index

The **Template index** widget feature allows complex data abstraction. For very complex lists or tables you require more than one data list to visualize an entry or a set of entries. For example, a table with mixed image and text content requires a list of images and a list of strings. To cover such complex cases, the **Template index** widget feature provides the property `lineTemplateIndex`.



Example 6.34. **Template index widget feature**

If you model a list using an instantiator with the property `lineMapping` set to `0|1` and the property `numItems` set to 5, the `lineTemplateIndex` results in `0|0|1|1|2`.

7. Modeling HMI behavior

NOTE



Default window layout

All instructions and screenshots of this user manual use the default window layout. If you want to follow the instructions, we recommend to reset the EB GUIDE Studio window to default layout by selecting **Layout > Reset to default layout**.

7.1. Modeling a state machine

7.1.1. Adding a state machine



Adding a state machine

Step 1

In the **Navigation** component, go to **State machines**, and click **+**.

A menu expands.

Step 2

Select a type for the state machine.

A new state machine of the selected type is added.

Step 3

Rename the state machine.

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:

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

Step 1

In the **Navigation** component, go to **Dynamic state machines**, and click **+**.

A menu expands.

Step 2

Select a type for the dynamic state machine.

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

Step 3

In the **Navigation** component, click the state machine, view state, or compound state to which you want to run in parallel the dynamic state machine.

Step 4

In the **Properties** component, 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** component, go to the **Entry action** property, and click **Add**.

Step 3

Enter an action using EB GUIDE Script.

For background information see [section 6.13, "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** component, go to the `Exit action` property, and click **Add**.

Step 3

Enter an action using EB GUIDE Script.

For background information see [section 6.13, “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

Step 1

In the **Navigation** component, right-click the state machine.

Step 2

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

The state machine is deleted.

7.2. Modeling 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 **Navigation** component, 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.

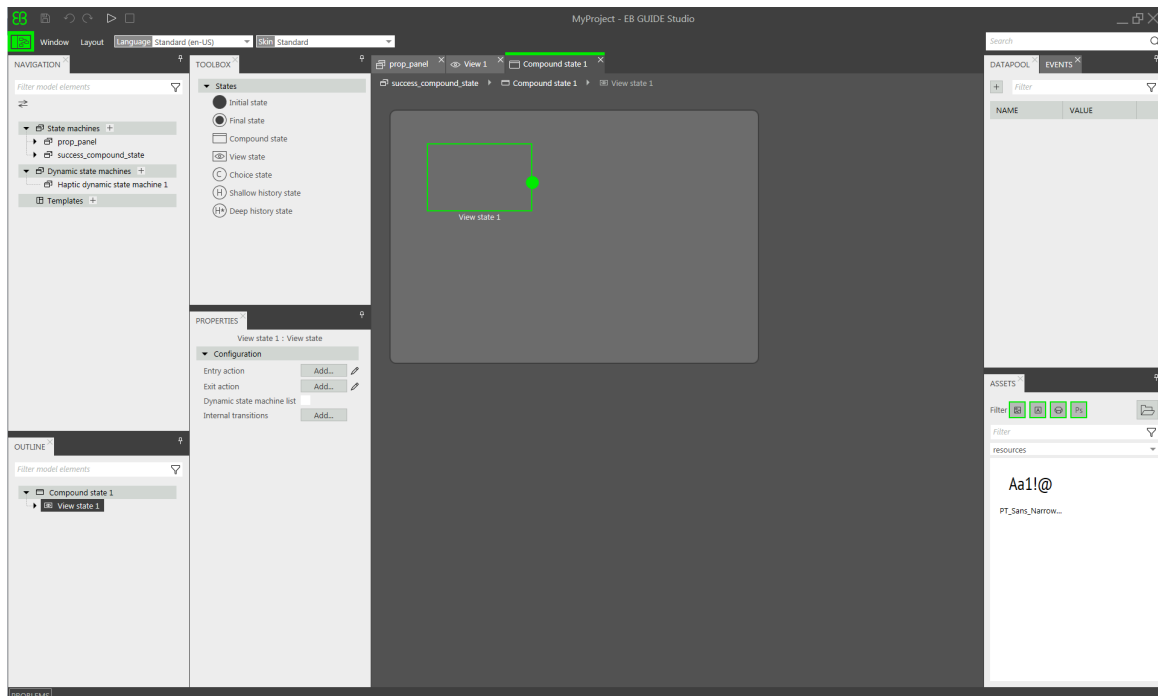


Figure 7.1. A compound state with a nested view state

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** component, right-click the transition. In the context menu, click **Convert to else**.

You added an else transition. The else transition is executed when all conditions which are assigned to outgoing choice transitions evaluate to `false`.

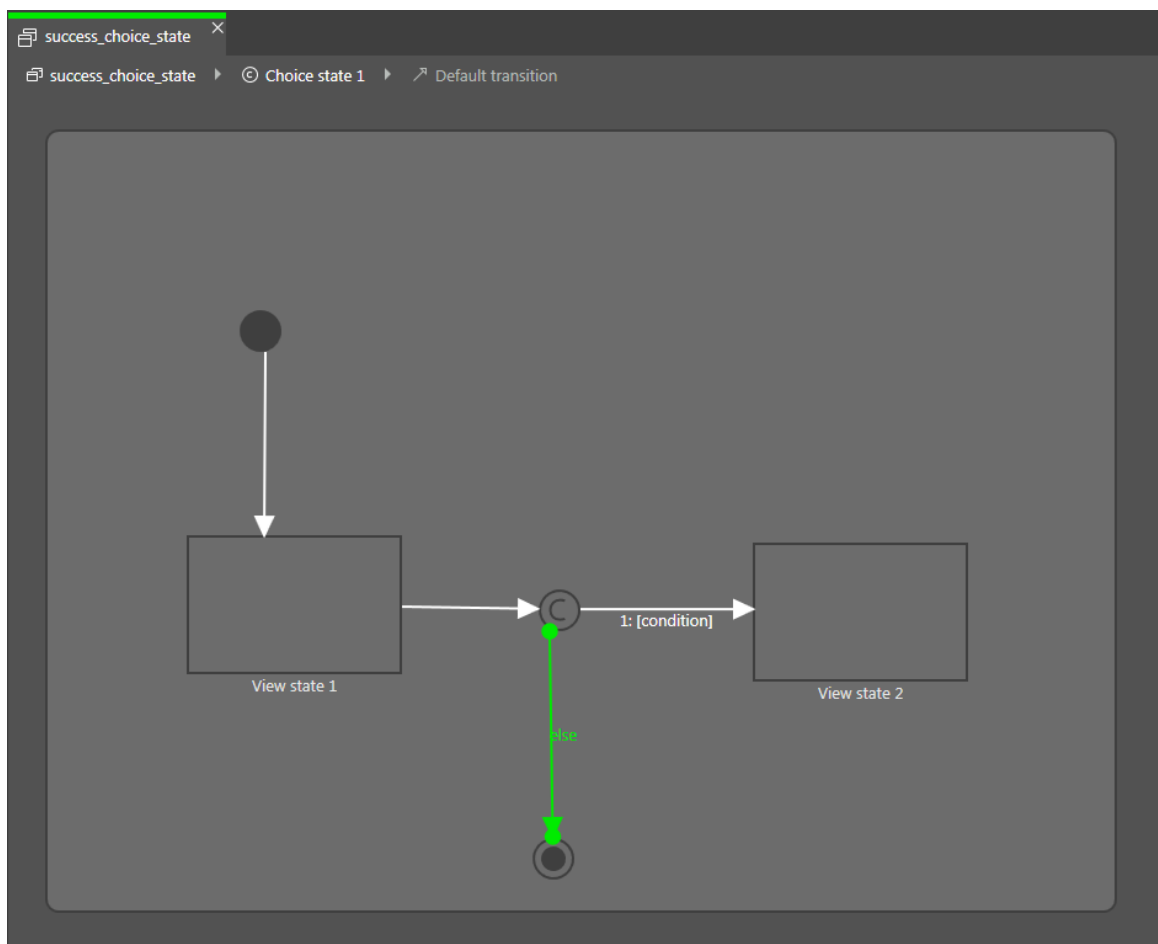


Figure 7.2. A choice state with its choice transitions

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** component, go to the `Entry action` property, and click **Add**.

Step 3

Enter an action using EB GUIDE Script.

For background information see [section 6.13, “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** component, go to the `Exit action` property, and click **Add**.

Step 3

Enter an action using EB GUIDE Script.

For background information see [section 6.13, “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** component, 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.

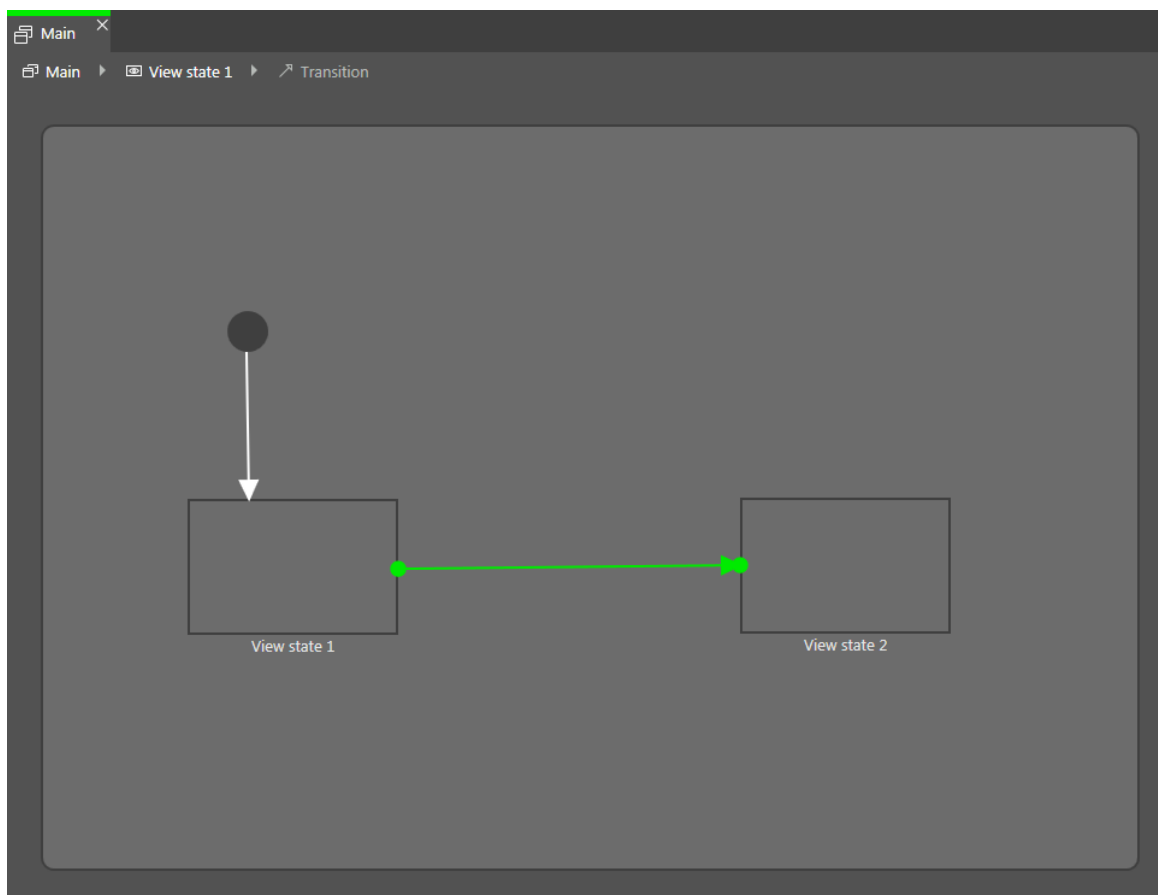


Figure 7.3. A transition

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** component, 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.

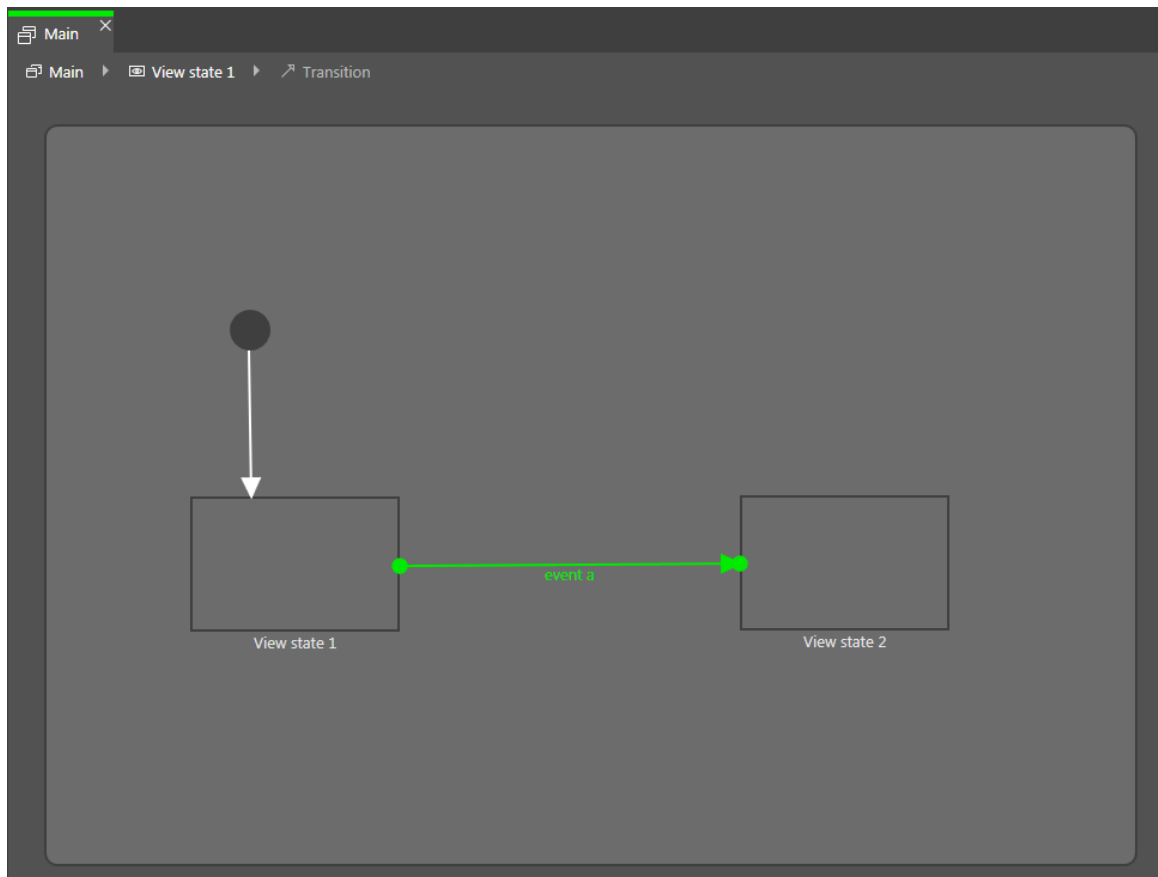


Figure 7.4. A transition with a trigger

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** component. Next to the `Condition` property, click **Add**.

Step 3

Enter a condition using EB GUIDE Script.

For background information see [section 6.13, "Scripting language EB GUIDE Script"](#).

Step 4

Click **Accept**.

The condition is added to the transition.

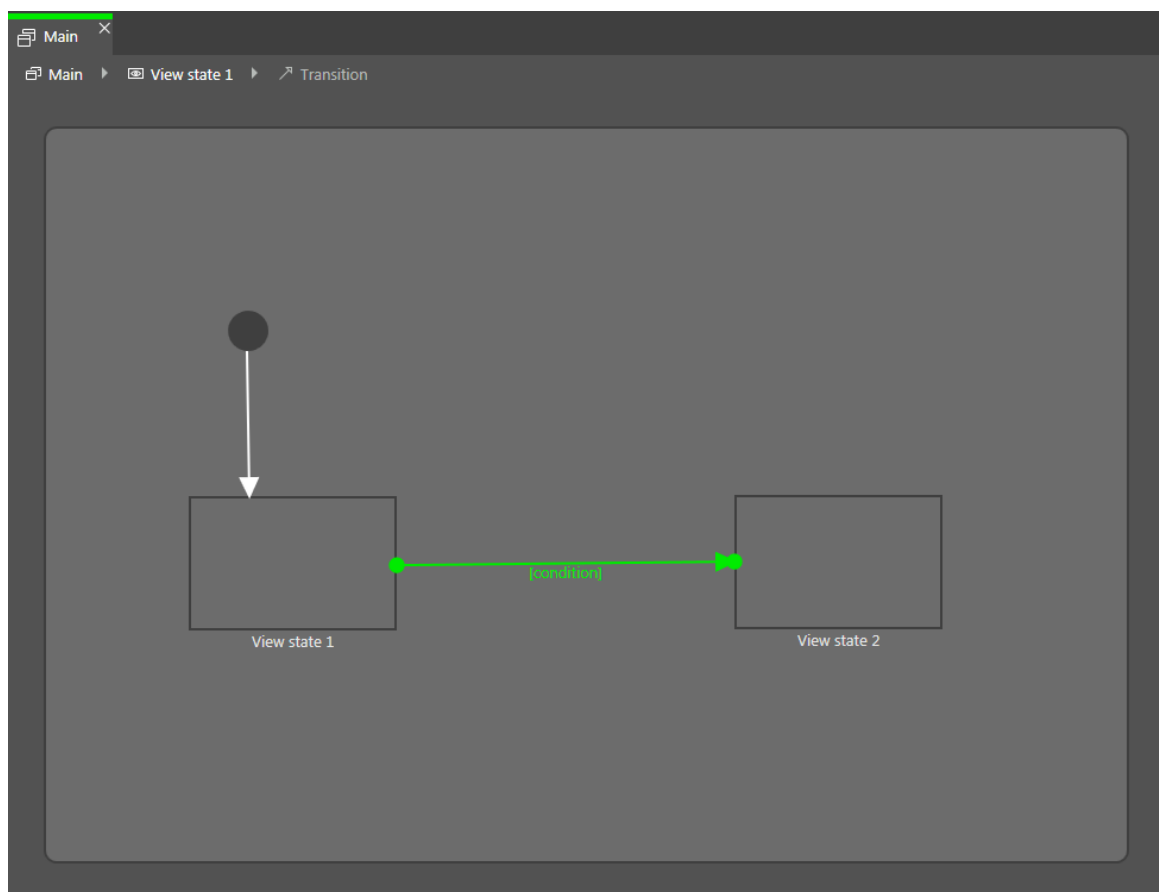


Figure 7.5. A transition with a condition

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

Step 3

Enter an action using EB GUIDE Script.

For background information see [section 6.13, "Scripting language EB GUIDE Script"](#).

Step 4

Click **Accept**.

The action is added to the transition.

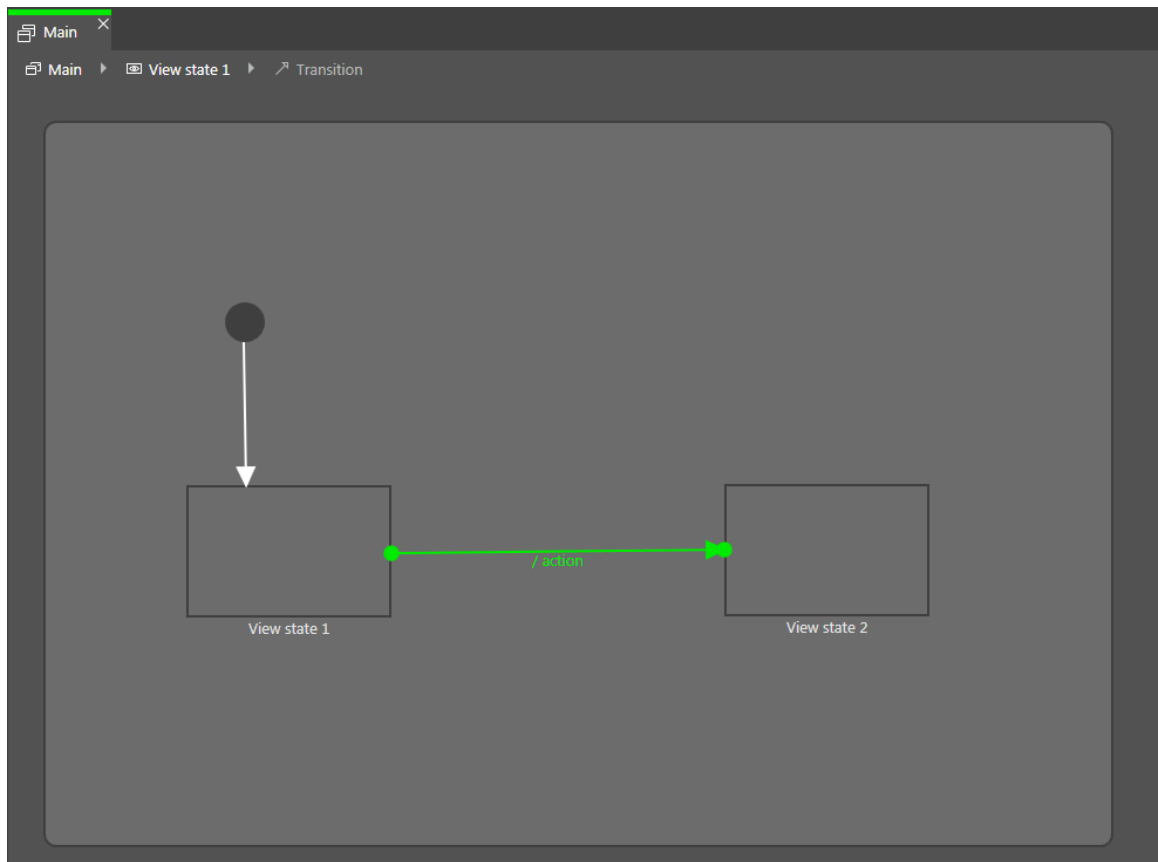


Figure 7.6. A transition with an action

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** component, go to **Internal transitions**, and click **Add**.

An internal transition is added to the state. The internal transition is visible in the **Navigation** component.

8. Modeling HMI appearance

NOTE



Default window layout

All instructions and screenshots of this user manual use the default window layout. If you want to follow the instructions, we recommend to reset the EB GUIDE Studio window to default layout by selecting **Layout > Reset to default layout**.

8.1. Working with widgets

TIP



Copying and finding views and widgets

You can copy and paste an existing view or widget using the context menu or **Ctrl+C** and **Ctrl+V**.

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

8.1.1. 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** component, click the view.

Step 3

Press the **F2** key, and rename the view.

Step 4

Double-click the view state in the content area.

The content area displays the new view.

8.1.2. Adding a basic widget to a view

For details on basic widgets, see [section 12.10.2, “Basic widgets”](#).

8.1.2.1. Adding a rectangle



Adding a rectangle

Prerequisite:

- The content area displays a view.

Step 1

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

The rectangle is added to the view.

8.1.2.2. Adding an ellipse



Adding an ellipse

Prerequisite:

- The content area displays a view.

Step 1

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

The widget is added to the view.

8.1.2.2.1. Editing an ellipse

You can draw just a sector of an ellipse and you can change the arc of an ellipse.



Creating a circular sector

Prerequisite:

- The view contains an ellipse.

Step 1

Click the ellipse and go to the **Properties** component.

Step 2

Enter the angle of the sector in the `centralAngle` text box.

Step 3

Enter the orientation of the sector in the `sectorRotation` text box.

You created a circular sector.



Creating a circular arc

Prerequisite:

- The view contains an ellipse.

Step 1

Click the ellipse and go to the **Properties** component.

Step 2

Enter the width of the arc in the `arcWidth` text box.

You created a circular arc.

8.1.2.3. Adding an image



Adding an image using **Toolbox**

Prerequisite:

- An image file is located in the `$GUIDE_PROJECT_PATH\resources` directory. For supported file types see [section 6.12.2, “Images”](#).
- The content area displays a view.

Step 1

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

Step 2

In the **Properties** component, select an image from the `image` drop-down list box. Alternatively, drag another image from the **Assets** component into the `image` drop-down list box.

The view displays the image.



Adding an image using **Assets** component

Prerequisite:

- An image file is located in the `$GUIDE_PROJECT_PATH\resources` directory. For supported file types see [section 6.12.2, “Images”](#).
- The content area displays a view.
- The **Assets** component is open.

Step 1

Drag an image file from the **Assets** component into the view.

The view displays the image.

Step 2

To change the image file, go to the **Properties** component and select an image from the `image` drop-down list box. Alternatively, drag another image from the **Assets** component into the `image` drop-down list box.

The view displays the image.



Adding 9-patch images

Prerequisite:

- A 9-patch image file is located in the `$GUIDE_PROJECT_PATH\resources` directory. For background information on 9-patch images see [section 6.12.2.1, “9-patch images”](#).
- The content area displays a view.
- An image is added to the EB GUIDE model.

Step 1

Select the image, and go to the **Properties** component.

Step 2

From the `image` drop-down list box select a 9-patch image.

Step 3

Go to the **Widget features properties** and click **Add/Remove**.

The **Widget feature** dialog is displayed.

Step 4

Under **Available widget features**, expand the **Layout** category, and select **Scale mode**.

Step 5

Click **Accept**.

The related widget properties are added to the image and displayed in the **Properties** component.

Step 6

In the **Properties** component, for the `scaleMode` property select `fit to Size (=1)`.

NOTE



Adding 9-patch images

If you do not add the **Scale mode** widget feature or if for the `scaleMode` property you select `original Size (=0)` or `keep aspect ratio (=2)`, the 9-patch image is scaled like a normal `.png` image.

8.1.2.4. Adding a label



Adding a label using **Toolbox**

Prerequisite:

- The content area displays a view.

Step 1

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

The label is added to the view. The label has the default font `PT_Sans_Narrow.ttf`.



Adding a label using **Assets** component

Prerequisite:

- A font file is located in the `$GUIDE_PROJECT_PATH\resources` directory. For supported file types see [section 6.12.1, “Fonts”](#).
- The content area displays a view.
- The **Assets** component is open.

Step 1

Drag a font file from the **Assets** component into the view.

The view displays the label with the selected font.

8.1.2.4.1. Changing the font of a label



Changing the font of a label

Prerequisite:

- A font file is located in the `$GUIDE_PROJECT_PATH\resources` directory. For supported file types see [section 6.12.1, “Fonts”](#).
- The EB GUIDE model contains a view state.
- The view contains a label.

Step 1

Select the label in the view.

Step 2

In the **Properties** component, select a font from the `font` drop-down list box.

Alternatively, drag a font file from the **Assets** component into the `font` drop-down list box.

The view displays the label with the new font.

NOTE



Calculation of text height and line gap

The following figure shows, how text height, line height, and line gap are calculated in EB GUIDE Studio. Take this into account when changing font style, size or line gap of a label.

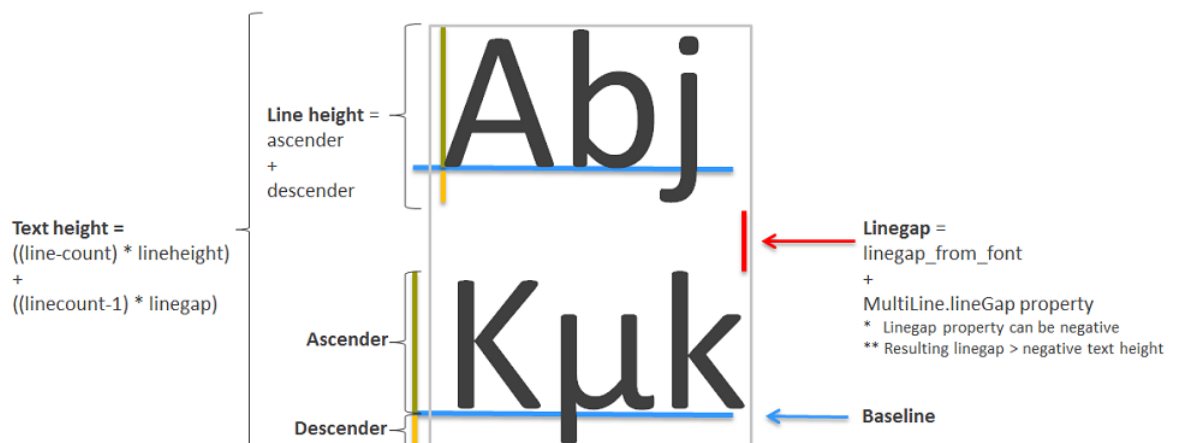


Figure 8.1. Calculation of text height, line height, and line gap

8.1.2.5. Adding a container



Adding a container

A container allows grouping widgets.

Prerequisite:

- The content area displays a view.

Step 1

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

Step 2

In the content area, enlarge the container by dragging one of its corners.

Step 3

Drag two or more widgets from the **Toolbox** into the container.

The widgets are modeled as child widgets of the container. Moving the container moves its child widgets along with it.

8.1.2.6. Adding an instantiator



Adding an instantiator

Prerequisite:

- The content area displays a view.

Step 1

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

Step 2

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

The widget serves as a line template.

Step 3

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

Step 3.1

For the `numItems` property enter a value that is greater than one.

Step 3.2

Add one of the following widget features to the instantiator:

- ▶ **Box layout**

- ▶ **Flow layout**
- ▶ **Grid layout**
- ▶ **List layout**

For details see [section 8.3.1, “Adding a widget feature”](#).

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

Step 4

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

You added the second child widget that serves as the second line template.

Step 5

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

Step 5.1

Next to the `lineMapping` click .

Step 5.2

Click the **Add** button.

The new entry is added to the table.

Step 5.3

In the `Value` text box enter 0.

Step 5.4

Click the **Add** button.

The new entry is added to the table.

Step 5.5

In the `Value` text box enter 1.

You defined the order in which the line templates are instantiated.



Example 8.1. Instantiation order

The `lineMapping` property defines the order of instantiation. For example, if you enter the values 1 | 0, the instantiator instantiates the line template 1 as the first child widget and the line template 0 as the second child widget.

The `lineMapping` property is applied iteratively. This means that if for the `numItems` property you enter 10, the result is the order 1|0|1|0|1|0|1|0|1|0.

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

NOTE



Linking of properties of the line templates

The following are the rules for linking:

- ▶ You cannot link properties between line templates.
- ▶ You cannot link from the outside of the instantiator to its line templates.
- ▶ You can link from a line template to the corresponding instantiator.

8.1.3. Adding a 3D widget to a view

8.1.3.1. Adding a scene graph to a view



Adding a scene graph to a view

For restrictions and recommendations see [section 6.1.2, “Settings for 3D graphic files”](#).

Prerequisite:

- A 3D graphic file is available. The file contains a camera, a light source, and one object containing a mesh and at least one material. For supported 3D graphic file formats see [section 6.1.1, “Supported 3D graphic formats”](#).
- The content area displays a view.

Step 1

Drag a scene graph from the **Toolbox** into the view.

The view displays the empty bounding box.

Step 2

In the **Properties** component, click **Import file**.

A dialog opens.

Step 3

Navigate to the 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** component displays the imported widget tree with the scene graph as a parent node. If the imported 3D scene has animations, the linear key value interpolation integer or linear key value interpolation float curve are added. Note that you cannot modify the underlying key-value pairs of these curves in EB GUIDE Studio.

TIP



Multiple import

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

After importing, multiple 3D graphics are rendered on top of each other. To display 3D objects separately, use the `visible` property of `RootNode`.

8.1.4. Adding a .psd file to a view



Adding a .psd file to a view

Prerequisite:

- A `.psd` file is available in `$GUIDE_PROJECT_PATH/<project name>/resources`. For background information see [section 6.12.4, “.psd file format”](#).
- The content area displays a view.
- The **Assets** component is open.

Step 1

In the **Assets** component, select the `resource` folder.

Step 2

From the preview area, drag the `.psd` file into the content area.

The import status message appears.

Step 3

Click **OK**.

If the import was successful, the **Navigation** component displays the widget tree that was created from the `.psd` file. The widget tree consists of containers and images and reflects the structure of the `.psd` file. In the `$GUIDE_PROJECT_PATH/<project name>/resources` directory a subdirectory with all extracted images is created.

NOTE



Extracting images from a .psd file

To extract images from a .psd file without importing it (i.e. creating a widget tree), in the **Assets** component right-click the .psd file and select **Generate images from PSD**. The extracted images are added to a subdirectory in \$GUIDE_PROJECT_PATH/<project name>/resources.

8.1.5. Deleting a widget from a view



Deleting a widget from a view

Prerequisite:

- The EB GUIDE model contains a widget.

Step 1

In the **Navigation** component, right-click a widget.

Step 2

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

The widget is deleted.

TIP



Deleting widgets from the content area

It is also possible to delete a widget by selecting it in the content area and pressing the **Delete** key.

8.2. 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** component displays the properties of the selected widget.

Step 2

To define the x-coordinate of the widget enter a value in the x text box.

Step 3

To define the y-coordinate of the widget enter a value in the y text box.

Step 4

Click outside the text box.

The content area displays the widget at the entered position.

TIP



Alternative approach

To position a widget by visual judgment, select the widget in the content area and move it with the mouse.

8.2.2. 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** component displays the properties of the selected widget.

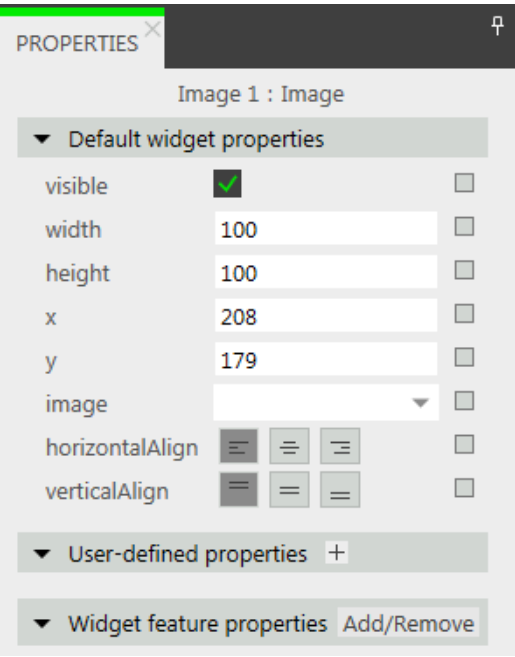


Figure 8.2. Properties of an image

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** component displays the properties of the rectangle.

Step 2

In the **Properties** component, go to the `width` property, and click the  button next to the property.

A menu expands.

Step 3

In the menu, click **Add link to widget property**.

A dialog opens.

Step 4

In the dialog, go to the view, and select its `width` property.

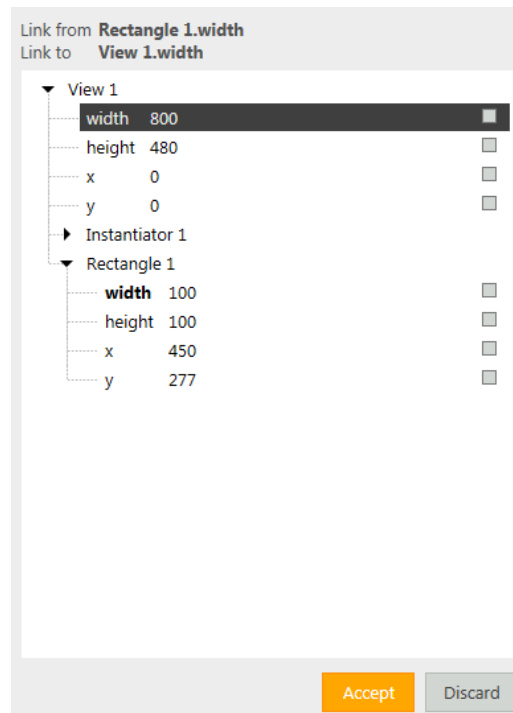



Figure 8.3. Linking between widget properties

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** component displays the properties of the image.

Step 2

In the **Properties** component, go to the `image` property, and click the  button next to the property.

A menu expands.

Step 3

In the menu, click **Add link to datapool item**.

A dialog opens.

Step 4

To add a new datapool item, enter a name in the combo box.

Step 5

Click **Add datapool item**.

Step 6

Click **Accept**.

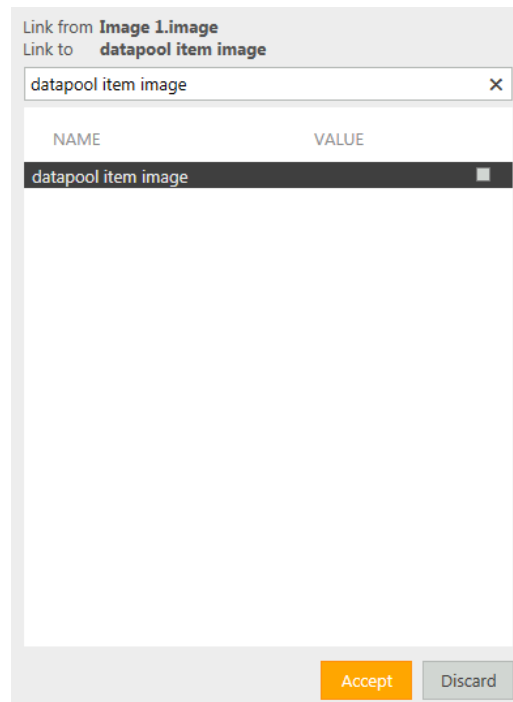



Figure 8.4. Linking to a datapool item

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** component displays the properties of the selected widget.

Step 2

In the **Properties** component, go to the **User-defined properties** category, and click **+**.

A menu expands.

Step 3

In the menu, click a type for the user-defined property.

A new widget property of the selected type is added to the widget.

Step 4

Rename the property.

8.2.5.1. 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** component displays the properties of the selected widget.

Step 2

In the **Properties** component, go to the **User-defined properties** category, and click **+**.

A menu expands.

Step 3

In the menu, click `Function () : bool`.

A new widget property of type `Function () : bool` is added to the widget.

Step 4

Rename the property.

Step 5

Next to the 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 8.2.
Calling a property of type `Function () : bool`

In your EB GUIDE model, there is a rectangle called `Background color`. You added a property of type `Function () : bool` to it. The property is called `change`.

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** component, select the widget with the user-defined property.

Step 2

In the **Properties** component, 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** component, click a widget.

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

Step 2

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

The **Widget features** dialog is displayed.

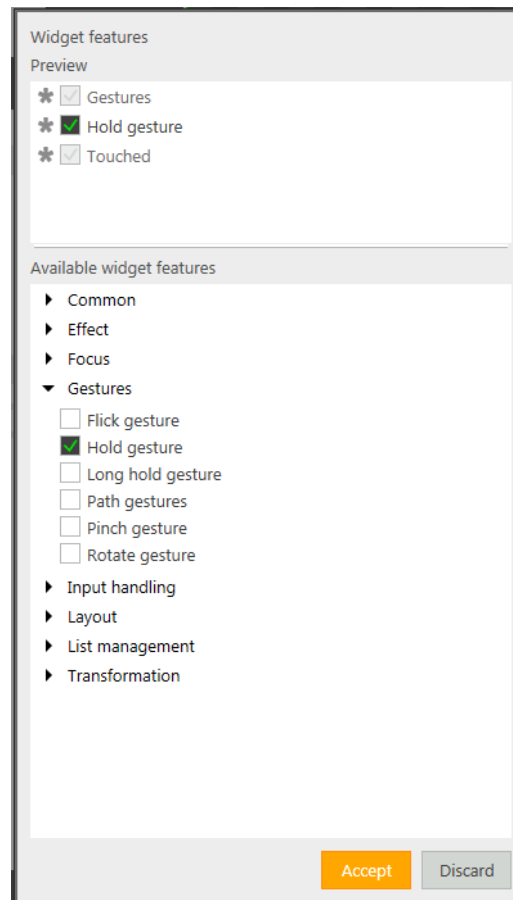


Figure 8.5. Widget features dialog

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 12.11, “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** component, click a widget.

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

Step 2

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

The **Widget features** dialog is displayed.

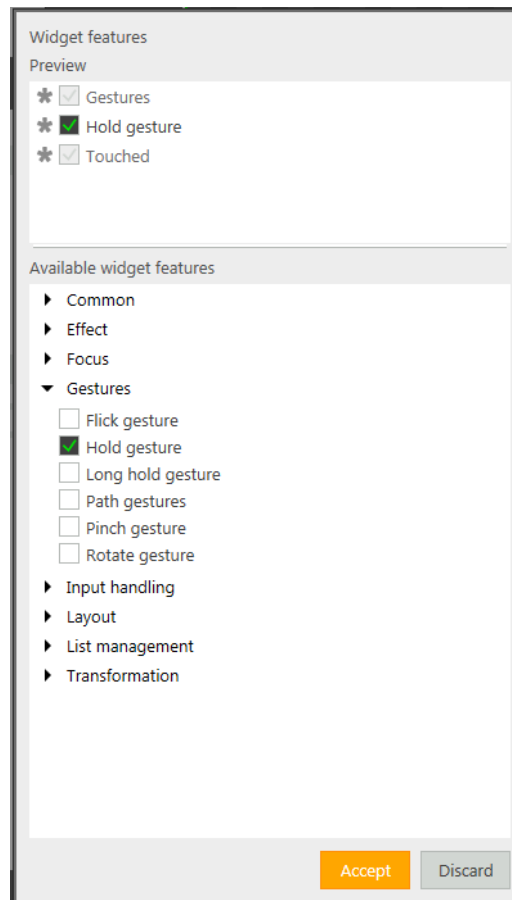


Figure 8.6. Widget features dialog

Step 3

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

Click **Accept**.

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

NOTE



Removing widget features with dependencies

Widget features which were added automatically due to dependencies are not deleted automatically. They cannot be removed directly. Clear the parent widget feature before you clear the child widget feature.

8.4. 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

NOTE



No skin support available

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



Adding a language

The 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. Working with skin support

With skin support you can define different datapool values for your model. This way you can define different looks for the same model, as for example night and day mode.

8.5.1. Adding a skin to the EB GUIDE model

NOTE



No language support available

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



Adding a skin to the EB GUIDE model

Step 1

Click  .

The project center opens.

Step 2

In the navigation area, click **Configure > Skins**.

A standard skin is added to each model by default.

Step 3

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

A skin is added to the table.

Step 4

Press **F2** and rename the skin.

The new skin is added to the model. In the project editor, the new skin can be selected in the drop-down list box of the command area.

8.5.2. Adding skin support to a datapool item



Adding skin support to a datapool item

To define different datapool values and thus define various looks for the your EB GUIDE model, you first need to add a skin support to the datapool item.


Prerequisite:

- The EB GUIDE model contains datapool items.
- A skin is added to the model.

Step 1

In the project editor, go to the **Datapool** component.


Step 2

Next to the **Value** property of a datapool item, click the  button.

A menu expands.

Step 3

In the menu, click **Add skin support**.

The dialog closes. Next to the **Value** property, the button  is displayed. It indicates that a skin support is added to this datapool item and now different values for each skin can be defined.

Step 4

To define different values for the datapool item, select the datapool in the **Datapool** component.

The **Properties** component displays a table with all skins available in the EB GUIDE model.

Step 5

Define a value for each skin in the table.

8.5.3. Switching between skins



Switching between skins

Prerequisite:

- The EB GUIDE model contains datapool items.
- A skin is added to the model.

Step 1

In the project editor go to the command area.

Step 2

Select a skin in the drop-down list box.

The content area displays the model with the datapool values valid for this skin. Also the simulation mode will display the model with the specific skin values.

8.5.4. Deleting a skin




Deleting a skin

Prerequisite:

- A skin is added to the model.

Step 1

Click  .

The project center opens.

Step 2

In the navigation area, click **Configure** > **Skins**.

All skins of the current project are listed.

Step 3

Select the skin to be deleted and click **Delete**.

The skin is deleted from the table.

8.6. Adding animations

8.6.1. Animating a widget



Animating a widget

For details on curves and for a description of curve properties see [section 12.10.3, “Animations”](#).

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 animation you added.

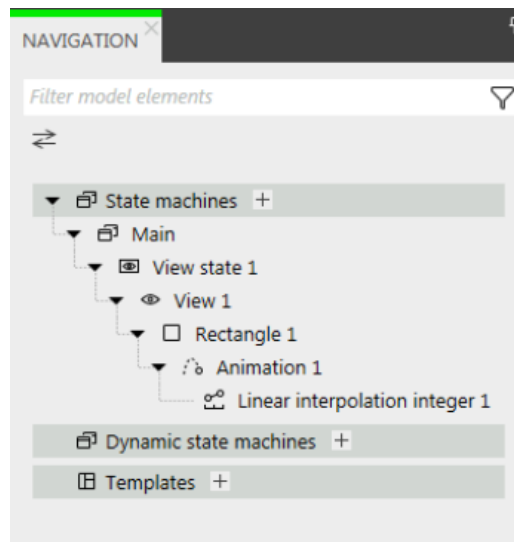


Figure 8.7. Widget hierarchy with an animation and a curve child widget

Step 5

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 6

In the **Properties** component next to the name of the property, click **Edit**.

A script editor opens.

Step 7

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 8

Select the curve you added in step four.

Step 9

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 10

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.

For a concrete animation example see [section 11.5, "Tutorial: Making an ellipse move across the screen"](#).

8.6.2. 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** component, click a view template.

Step 2

Go to the **Properties** component.

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** component, click a view template.

Step 2

Go to the **Properties** component.

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.

8.7. Re-using a widget

8.7.1. Adding a template



Adding a template

Step 1

In the **Navigation** component, 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** component, 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.

8.7.2. Defining the template interface



Defining the template interface

Prerequisite:

- The EB GUIDE model contains a template.

Step 1


Select a template.

Step 2

To add a property to the template interface, in the **Properties** component click the  button next to the property. In the menu, click **Add to template interface**.

The  icon is displayed next to the property.

Step 3

To remove a property from the template interface, click the  button next to the property. In the menu, click **Remove from template interface**.

The  icon is no longer displayed next to the property.

NOTE



Instantiator templates

For templates of instantiators, it is not possible to add properties of the instantiator's child widgets to the template interface.

8.7.3. 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** component displays the properties which belong to the template interface.

TIP



Define the template interface


If the **Properties** component does not display any properties for a template instance, no properties have been added to the template interface. Define the template interface to change that.

Step 2

In the **Properties** component, edit the properties of the template instance.

After editing a property, the  button changes to the  button.

Step 3

To reset a property value to the value of the template, click the  button next to the property. In the menu, click **Reset to template value**.

8.7.4. Deleting a template



Deleting a template

Step 1

In the **Navigation** component, right-click a template.

Step 2

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

The template is deleted.

9. Handling data

NOTE



Default window layout

All instructions and screenshots of this user manual use the default window layout. If you want to follow the instructions, we recommend to reset the EB GUIDE Studio window to default layout by selecting **Layout > Reset to default layout**.

9.1. Adding an event



Adding an event

Step 1

In the **Events** component, click **+**.

An event is added to the table.

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:

- An event is added to the EB GUIDE model.

Step 1

In the **Events** component, click an event.

Step 2

In the events table click **+** next to the event.

Step 3

From the drop-down list box select a type for the parameter.

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

Step 4

Rename the parameter.

9.3. Addressing an event

Event IDs and event group IDs are used to address events. EB GUIDE TF uses the IDs to send and receive the events at run-time.



Adding an event group

Step 1

Click  .

The project center opens.

Step 2

In the navigation area, click **Configure > Event groups**.

Step 3

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

An event group is added to the table.

Step 4

Rename the event group.

Step 5

To change an event group ID, double-click the **ID**, and type a number.



Addressing an event for EB GUIDE TF

Prerequisite:

- An event group is added.
- An event is added to the EB GUIDE model.

Step 1

In the **Events** component, click an event.

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

Step 2

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

Step 3

Go to the **Events** component and select an event group from the `Group` drop-down list box.

9.4. Deleting an event



Deleting an event

Prerequisite:

- An event is added to the EB GUIDE model.

Step 1

In the **Events** component, 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

Step 1

In the **Datapool** component, 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 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:

- A datapool item of a list type is added.

Step 1

In the **Datapool** component, click a datapool item of a list type.

Step 2

In the `Value` column, click .

An editor opens.

Step 3

To add an item to the list datapool item, click **Add**.

A new entry is added to the table.

Step 4

Enter a value for the new entry in the `Value` text box or select a value from the 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 in the `Value` column.

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:

- A datapool item is added.
- The datapool item is not language-dependent.
- The datapool item is not linked.

Step 1

In the **Datapool** component, click a datapool item and click the  button.

A menu expands.

Step 2

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

The datapool item is converted to a scripted value.

Step 3


In the `Value` column, click **Edit**.

A script editor opens in the content area.

Step 4

Edit the EB GUIDE Script.

Step 5

To convert the datapool item back to its plain value, click the  button.

A menu expands.

Step 6

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

The datapool item is converted to its plain value.

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

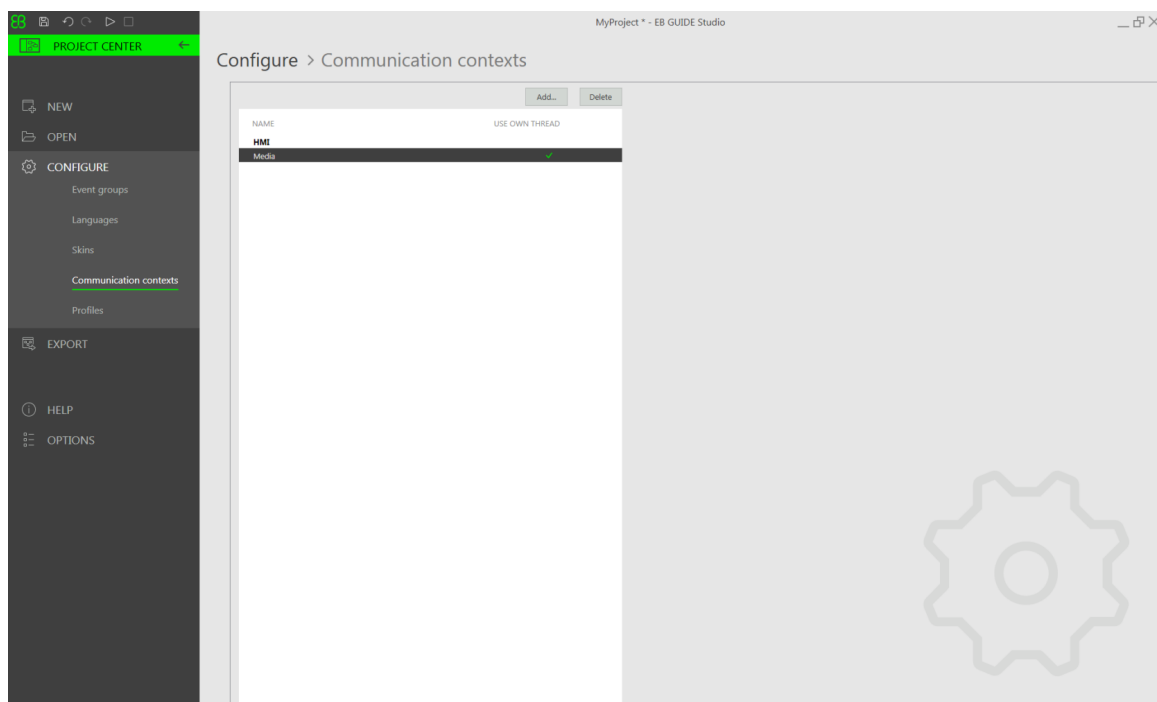


Figure 9.1. Communication context `Media`.

9.9. Linking between datapool items



Linking between datapool items

Prerequisite:

- A datapool item is added.
- The datapool item is not language-dependent.
- The datapool item is not a scripted value.

Step 1

In the **Datapool** component, click a datapool item.

Step 2

Click the  button.

A menu expands.

Step 3

In the menu, click **Add link to datapool item**.

A dialog opens.

Step 4

To add a new datapool item, enter a name in the text box.

Step 5

Click **Add datapool item**.

Step 6

Click **Accept**.

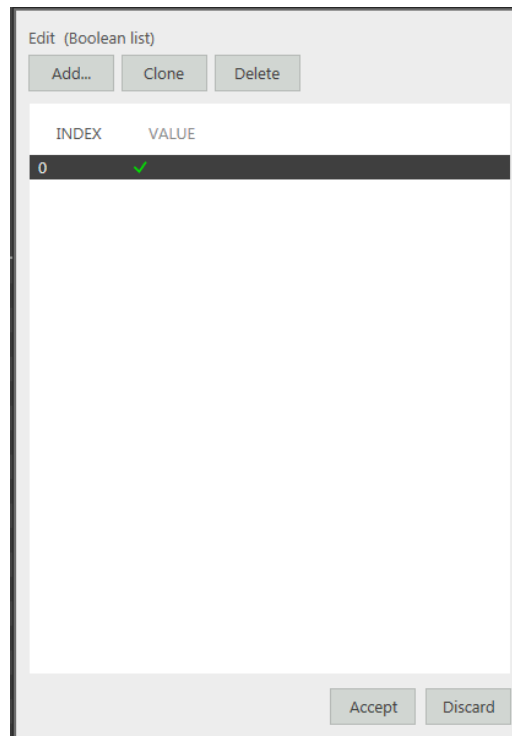



Figure 9.2. Linking between datapool items

The dialog closes. Next to the `Value` property, the  button is displayed. It indicates that the `Value` property is linked to a datapool item. Whenever one of the datapool items changes its value, the value of the other datapool item changes as well.

9.10. Deleting a datapool item



Deleting a datapool item

Prerequisite:

- A datapool item is added.

Step 1

In the **Datapool** component, right-click the datapool item.

Step 2

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

The datapool item is deleted.

10. Handling a project

NOTE



Default window layout

All instructions and screenshots of this user manual use the default window layout. If you want to follow the instructions, we recommend to reset the EB GUIDE Studio window to default layout by selecting **Layout > Reset to default layout**.

10.1. Creating a project



Creating a project

Step 1

Click  .

The project center opens.

Step 2

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

Step 3

Enter a project name, and select a location.

Step 4

Click **Create**.

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

10.2. Opening a project

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. Renaming model elements, datapool items and events



Renaming model elements

The following instruction guides you through the process of renaming model elements such as states, widgets, transitions and animations.

Prerequisite:

- A model element is added to the EB GUIDE model.

Step 1

In the **Navigation** component right-click the model element.

The context menu opens.

Step 2

In the context menu, click **Rename** and confirm with **Enter**.

The changed name is shown.



Renaming datapool items and events

The following instruction guide you through the process of renaming datapool items.

NOTE



Renaming events

The same procedure applies for renaming events in the **Events** component.

Prerequisite:

- A datapool item is added to the EB GUIDE model.

Step 1

In the **Datapool** component right-click the datapool item.

The context menu opens.

Step 2

In the context menu, click either of the following:

- ▶ Click **Rename** to rename only the selected datapool item.
- ▶ Click **Rename global** to rename the selected datapool item and also its occurrences in the EB GUIDE model, for example in EB GUIDE Script.

10.4. Validating and simulating an EB GUIDE model

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



10.4.1. Validating an EB GUIDE model

10.4.1.1. Validating an EB GUIDE model using EB GUIDE Studio




Validating an EB GUIDE model using EB GUIDE Studio

In the **Problems** component, EB GUIDE displays the following:

- ▶  errors
- ▶  warnings

Step 1

In the **Problems** component, click .

The number of errors and warnings is displayed.

Step 2

Click **Problems** to expand the **Problems** component.

A list of errors and warnings is displayed.

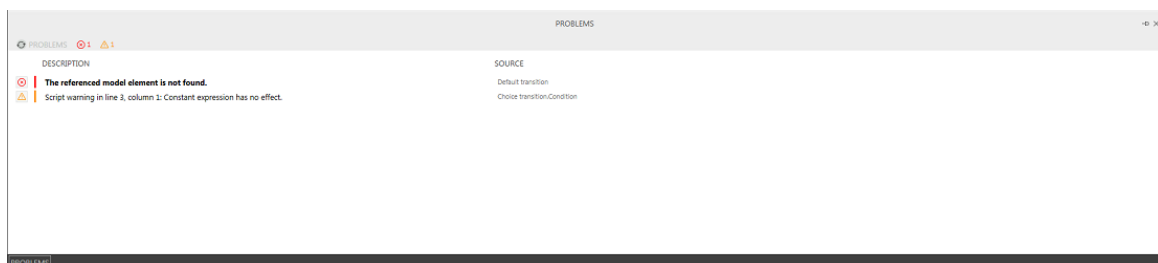


Figure 10.1. Problems component

Step 3


To navigate to the source of a problem, double-click the corresponding line.

The element that causes the problem is highlighted.

Step 4

Solve the problem.

Step 5

Click .

The problem you solved is no longer listed in the **Problems** component.

Step 6

To collapse the **Problems** component, click **Problems** once again.

If there are no errors, the EB GUIDE model is valid. The EB GUIDE model is also valid if there are some warnings.

10.4.1.2. Validating an EB GUIDE model using command line



Validating an EB GUIDE model using command line

Step 1

Start `$GUIDE_INSTALL_PATH\Studio\Studio.Console.exe`.

Step 2

Enter `Studio.Console.exe -c <logfile dir> "project_name.ebguide"`.

The EB GUIDE model is validated and the result is saved to a logfile at the specified location `<logfile dir>`.

10.4.2. Starting and stopping the simulation



Starting and stopping the simulation

Step 1

To start the simulation, click ► in the command area.

The simulation and EB GUIDE Monitor start. The simulation starts with its own configuration.

To change the configuration, go to the project center, and click **Configure > Profiles**.

Step 2

To stop the simulation, click □ in the command area.

The simulation and EB GUIDE Monitor stop.

10.5. Working with EB GUIDE Monitor

10.5.1. Firing an event in EB GUIDE Monitor



Firing an event in EB GUIDE Monitor

Prerequisite:

- The simulation of the EB GUIDE model is started.
- The EB GUIDE Monitor is started.

Step 1

In EB GUIDE Monitor, in the **Events** component search for an event to be fired in the **Search for event** search box.

Step 2

Click the event.

The event is added to the list.


Step 3

To fire an event, click  in the **Events** component next to the event.

The event is fired. In the **Logger** component a log message appears.

Step 4

Step 4.1

If the event has parameters, click  to expand parameters.

Step 4.2

Change parameters in the **Value** column.

Step 4.3

To fire an event, click  next to the event.

The event is fired with changed parameters. In the **Logger** component a log message appears.

10.5.2. Changing value of the datapool item with EB GUIDE Monitor



Changing value of the datapool item in EB GUIDE Monitor

Prerequisite:

- The simulation of the EB GUIDE model is started.
- The EB GUIDE Monitor is started.

Step 1

In EB GUIDE Monitor, in the **Datapool** component search for a datapool item in the **Search for datapool item** search box.

Step 2

Click the datapool item.

The datapool item is added to the list.

Step 3

Change the value of the datapool item in the **Value** column.

NOTE



Supported types

You can change datapool items of the following data types:

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

The value of the datapool item is changed. In the **Logger** component a log message appears.

10.5.3. Starting scripts in EB GUIDE Monitor



Starting scripts in EB GUIDE Monitor

Prerequisite:

- The simulation of the EB GUIDE model is started.
- The EB GUIDE Monitor is started.
- A `.cs` or a `.dll` file with a script is available on your computer.

Step 1

To open the **Scripting** component, select **Layout > Scripting**.

The **Scripting** component opens as a docked component.

Step 2

In the **Scripting** component click the **Open** button.

The file explorer opens.

Step 3

Select a `.cs` or a `.dll` file and click **Open**.

All applicable methods and the corresponding classes, which were included in the file, are listed in the **Script** table.

Step 4

Select a method and click the start button.

The script is started. In the **Script output** area a log message appears.



Example 10.1. Example script file for EB GUIDE Monitor

The following is an example script `MonitorScriptSample.cs`.

```
namespace MyProject
{
    using System.Threading.Tasks;

    using System.Windows.Media; // necessary for Color type!

    using Elektrobit.Guide.Monitor.Scripting.ScriptApi;

    public class Basic
    {
        public async Task PrintMessage(IMonitorContext monitor)
        {
            await monitor.Write("Hello World"); ❶
        }

        public async Task FireEvent(IMonitorContext monitor)
        {
            await monitor.FireEvent("nextView"); ❷
        }
    }

    public class Events
    {
        public async Task FireEventWithParameter(IMonitorContext monitor)
        {
            await monitor.FireEvent("setBool", true);
        }

        public async Task WaitForEvent(IMonitorContext monitor)
        {
            var ev = await monitor.WaitForEvent("nextView"); ❸
            await monitor.Write("Even occurred: " + ev.EventModel.Name);
        }
    }
}
```

```
    }

    public async Task WaitForEventWithParameters(IMonitorContext monitor)
    {
        var ev = await monitor.WaitForEvent("setBool");

        bool mv1 = ev["value"]; // read parameter via name
        bool mv2 = ev[0]; // read the parameter via index

        await monitor.Write("Parameter 'value' is: " + mv1);
        await monitor.Write("Parameter [0] is: " + mv2);
    }
}

public class Datapool
{
    public async Task WriteDpValue(IMonitorContext monitor)
    {
        await monitor.WriteDatapool("Boolean 1", true); ❹
    }

    public async Task ReadDatapoolValue(IMonitorContext monitor)
    {
        bool boolValue = await monitor.ReadDatapool("Boolean 1"); ❺
        string stringValue = await monitor.ReadDatapool("String 1");
        int integerValue = await monitor.ReadDatapool("Integer 1");
        float floatValue = await monitor.ReadDatapool("Float 1");

        await monitor.Write("Boolean: " + boolValue);
        await monitor.Write("String: " + stringValue);
        await monitor.Write("Integer: " + integerValue);
        await monitor.Write("Float: " + floatValue);
    }

    public async Task ReadColor(IMonitorContext monitor)
    {
        Color colorValue = await monitor.ReadDatapool("Color 1");
        await monitor.Write("Boolean: " + colorValue);
    }
}

public class Advanced
{
    public async Task CaptureScreenshot(IMonitorContext monitor)
    {
        // make sure remote framebuffer is enabled in profile
        var sceneId = 0;
```

```
        await monitor.CaptureScreenshot(sceneId, @"d:\image.png"); ❸
    }

    public async Task CountTo10(IMonitorContext monitor)
    {
        for (var i = 0; i < 10; i++)
        {
            await monitor.Write("Hello World: " + i);
            await Task.Delay(1000, monitor.CancellationToken);

            monitor.CancellationToken.ThrowIfCancellationRequested();
        }
    }

    public async Task WaitForEventWithTimeout(IMonitorContext monitor)
    {
        // Disclaimer:
        // this is just one of many opportunities provided by
        // the .NET's "Task Parallel Library"

        var eventWaitTask = monitor.WaitForEvent("nextView"); ❷

        await Task.WhenAny(eventWaitTask, Task.Delay(5000));

        if (!eventWaitTask.IsCompleted || eventWaitTask.IsFaulted)
        {
            return;
        }

        await monitor.Write("event occurred");
    }
}
```

- ❶ Method to print out a message
- ❷ Method to fire an event
- ❸ Method to wait for an event
- ❹ Method to write a datapool value
- ❺ Method to read a datapool value
- ❻ Method to capture a screenshot
- ❼ Method to wait for an event with timeout

10.5.4. Starting EB GUIDE Monitor using command line



Starting EB GUIDE Monitor using command line

EB GUIDE Monitor starts automatically in EB GUIDE Studio during the simulation of an EB GUIDE model. But it is also possible to start EB GUIDE Monitor using command line.

Prerequisite:

- EB GUIDE is installed.

Step 1

In the file explorer, navigate to `$GUIDE_INSTALL_PATH\tools\monitor`.

Step 2

Open command line and enter the following: `Monitor.exe -c <ip address>:<port> <$EXPORT_PATH\monitor.cfg>`

EB GUIDE Monitor starts.

TIP



Reusing preconfigured datapool items and events

In EB GUIDE Monitor you can configure datapool items and events. The configured values are stored in `C:\Users\<username>\AppData\Local\Temp\eb_guide_simulation_export\<guide_project>\monitor_settings.xml`. To reuse the preconfigured values, copy `monitor_settings.xml` to `$EXPORT_PATH`.

10.6. Exporting an EB GUIDE model

10.6.1. Exporting an EB GUIDE model using EB GUIDE Studio




Exporting an EB GUIDE model using EB GUIDE Studio

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

For every export of an EB GUIDE model you select a profile.

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.

10.6.2. Exporting an EB GUIDE model using command line



Exporting an EB GUIDE model using command line

Prerequisite:

- The EB GUIDE model is free of errors and warnings.

Step 1

Start `$GUIDE_INSTALL_PATH\Studio\Studio.Console.exe`.

Step 2

Enter `Studio.Console.exe -e <destination dir> -p <profile> "project_name.ebguide"`.

The EB GUIDE model is exported to the selected location `<destination dir>` with the specified profile `<profile>`.

10.7. 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.

10.8. Configuring profiles

EB GUIDE Studio offers the possibility to create different profiles for an EB GUIDE model.

You use profiles to do the following:

- ▶ Send messages
- ▶ Configure internal and user-defined libraries to load
- ▶ Configure a scene
- ▶ Configure a renderer

There are two default profiles: **Edit** and **Simulation**.

10.8.1. 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.

10.8.2. Adding a library

The default delivery of EB GUIDE TF runs on operating systems that support shared libraries, for example Windows 10, Linux or QNX. EB GUIDE TF is divided into executable file and a set of libraries to fit most customer projects out of the box.

The following tasks show you how to add a user-defined library that interacts with the EB GUIDE model and provides additional functionality.



Adding a library: Platform

This task shows you how to add a library or several libraries that can be used by all EB GUIDE models on the current platform.

Prerequisite:

- An EB GUIDE Studio project is opened.
- The project center is displayed.
- In the navigation area, the tab **Configure > Profiles** is selected.
- A profile `MySimulation` is added.
- Libraries `MyLibraryA` and `MyLibraryB` are available in `$GTF_INSTALL_PATH\platform\<platform name>`.

Step 1

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

Step 2

Click the **Platform** tab.

Step 3

Enter the following code:

```
{  
  "gtf":
```



```
{
  "core":
  {
    "pluginstoload": ["MyLibraryA", "MyLibraryB"]
  }
}
```

You added libraries `MyLibraryA` and `MyLibraryB` to the start-up code.

NOTE



JSON object notation

If you configure `platform.json` within EB GUIDE Studio, use the JSON object notation.

For an example, see [section 12.7.1, “Example platform.json in EB GUIDE Studio”](#).

For more information about JSON format, see <http://www.json.org>.



Adding a library: Model

This task shows you how to add a library or several libraries that can be used only by the current EB GUIDE model.

Prerequisite:

- An EB GUIDE Studio project is opened.
- The project center is displayed.
- In the navigation area, the tab **Configure > Profiles** is selected.
- A profile `MySimulation` is added.
- Libraries `MyLibraryA` and `MyLibraryB` are available in `$GUIDE_PROJECT_PATH\resources`.

Step 1

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

Step 2

Click the **Model** tab.

Step 3

Enter the following code:

```
{
  "gtf":
  {
    "model":
    {
      "pluginstoload": ["resources/MyLibraryA", "resources/MyLibraryB"]
    }
  }
}
```

```
}  
}  
}
```

You added libraries `MyLibraryA` and `MyLibraryB` to the start-up code.

NOTE



JSON object notation

If you configure `model.json` in EB GUIDE Studio, use the JSON object notation.

For an example, see [section 12.6.1, “Example `model.json` in EB GUIDE Studio”](#).

For more information about JSON format, see <http://www.json.org>.

10.8.3. Configuring a scene

In EB GUIDE Studio it is possible to configure a scene for every state machine.

Projects can have more than one state machine for one of the following reasons:

- ▶ To separate the logic of the model into different state machines
- ▶ To use more than one display or layer



Configuring a scene

Prerequisite:

- An EB GUIDE 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 the **Scenes** tab.

Step 2

From the **State machine** drop-down list box select the state machine of your main display, for example **Main**.

Step 3

To set the initial position of the window on the PC desktop, enter a value for *x* and *y*.

Step 4

Select a renderer from the **Renderer** drop-down list box.

Step 5

Adjust further properties. For information on each property see [section 12.8, “Scenes”](#).

10.9. Exporting and importing language-dependent texts

10.9.1. 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.

10.9.2. Importing language-dependent texts

10.9.2.1. Importing language-dependent texts using EB GUIDE Studio




Importing language-dependent texts using EB GUIDE Studio

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**.

10.9.2.2. Importing language-dependent texts using command line



Importing language-dependent texts using command line

Prerequisite:

- At minimum two languages are added to the EB GUIDE model.
- The EB GUIDE model is free of errors and warnings.
- One translated `.xliff` language file is available.

Step 1

Start `$GUIDE_INSTALL_PATH\Studio\Studio.Console.exe`.

Step 2

Enter `Studio.Console.exe -l <language file> "project_name.ebguide"`.

If the import was successful, the EB GUIDE model is saved. If the import was not successful, the EB GUIDE model is not saved. In both cases a logfile is generated. A date and a time stamp are added to the name of the logfile.

11. Tutorials

NOTE



Default window layout

All instructions and screenshots of this user manual use the default window layout. If you want to follow the instructions, we recommend to reset the EB GUIDE Studio window to default layout by selecting **Layout > Reset to default layout**.

11.1. Tutorial: Adding a dynamic state machine

NOTE



Default window layout

All instructions and screenshots of this user manual use the default window layout. If you want to follow the instructions, we recommend to reset the EB GUIDE Studio window to default layout by selecting **Layout > Reset to default layout**.

Dynamic state machines allow pop-ups during run-time. You use dynamic state machines for example to display error messages that overlay the regular display.

The following instructions guide you through the process of creating a dynamic state machine. The instructions show you how to model a dynamic state machine for volume control. For best results, work through the following steps in the order presented.

Approximate duration: 20 minutes.



Adding events and datapool items

The following instructions guide you through the process of adding events and datapool items. These events are used to change the volume afterwards. The purpose of the datapool item is to change the position of a graphical element in a later section.

Step 1

Go to the **Events** component and click **+**.

An event is added to the table.

Step 2

Rename the event to `Volume up`.

Step 3

Add an event, and rename it to `Volume down`.

Step 4

Add an event, and rename it to `Close volume control`.

Step 5

Go to the **Datapool** component and click **+**.

A menu expands.

Step 6

In the menu, click **Integer**.

A datapool item of type `Integer` is added.

Step 7

Rename the datapool item to `Volume indicator`.

You added three events and a datapool item.



Adding a dynamic state machine and modeling the behavior

The following instructions guide you through the process of adding a dynamic state machine. The haptic dynamic state machine that you model is used to control the volume.

Prerequisite:

- You completed the previous instruction.

Step 1

In the **Navigation** component, go to **Dynamic state machines** and click **+**.

A menu expands.

Step 2

In the menu, click **Haptic dynamic state machine**.

A haptic dynamic state machine is added and displayed in the content area.

Step 3

Rename the dynamic state machine to `Volume control`.

Step 4

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

Step 5

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 6

In the **Navigation** component, click the view state.

Step 7

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

Step 8

Add a transition from the initial state to the `Volume` view state.



Modeling a slider

The following instructions guide you through the process of modeling a horizontal slider indicator. The slider indicator shows the volume during run-time.

The slider indicator consists of two rectangles. One rectangle represents the background of the slider. The second rectangle indicates the volume.

Prerequisite:

- You completed the previous instruction.

Step 1

In the **Navigation** component, expand the `Volume` view state. Double-click the view.

The content area displays the view.

Step 2

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

Step 3

In the **Navigation** component, click the rectangle, and press the **F2** key.

Step 4

Rename the rectangle to `Slider background`.

Step 5

To change the appearance of `Slider background`, click the rectangle, and go to the **Properties** component.

Step 5.1

Enter 500 in the `width` text box.

Step 5.2

Enter 125 in the `x` text box.

Step 5.3

Enter 300 in the `y` text box.

Step 6

Drag a rectangle from the **Toolbox** into `Slider background` in the **Navigation** component.

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

Step 7

In the **Navigation** component, click the rectangle, and press the **F2** key.

Step 8

Rename the rectangle to `Indicator`.

Step 9

To change the appearance of `Indicator`, click the rectangle, and go to the **Properties** component.

Step 9.1

Enter 40 in the `width` text box.

Step 9.2

Enter 80 in the `height` text box.

Step 9.3

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

A menu expands.

Step 9.4

In the menu, click **Add link to datapool item**.


A dialog opens.

Step 9.5

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.

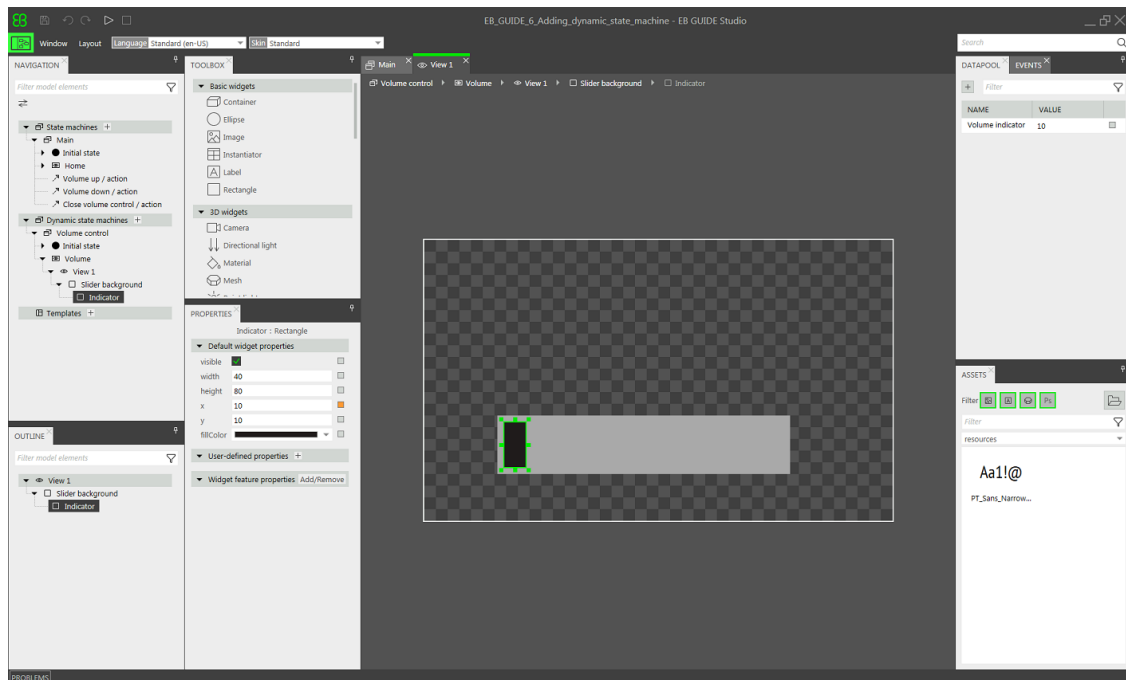


Figure 11.1. Appearance of **View 1** with two rectangles

Step 10

In the **Datapool** component, click the `Volume indicator` datapool item.

Step 11

In the `Value` text box enter 10.

In the content area, the `Indicator` rectangle changes the position.

The `Volume indicator` datapool item controls the `x` position of the `Indicator` rectangle.



Adding states to the **Main** state machine

In the following instructions, you add an initial state and a view state to the **Main** state machine. You use the view state to run the dynamic state machine in parallel to other state machines.

Prerequisite:

- You completed the previous instruction.

Step 1

In the **Navigation** component, double-click **Main**.

The **Main** state machine is displayed in the content area.

Step 2

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

Step 3

Drag a view state from the **Toolbox** into the state machine.

Along with the view state, a view is added to the EB GUIDE model.

Step 4

Rename the view state to `Home`.

Step 5

In the content area, click the initial state.

Step 6

Add a transition from the initial state to the `Home` view state.

Step 7

In the **Navigation** component, click **Main**.

Step 8

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

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

You added an initial state and a view state to the **Main** state machine. The haptic dynamic state machine runs in parallel to the **Main** state machine.



Adding internal transitions to the **Main** state machine

In the following instruction, you add internal transitions. You use the internal transitions to start (push) and stop (pop) the dynamic state machine during run-time.

Prerequisite:

- You completed the previous instruction.

Step 1

In the **Navigation** component, click the **Main** state machine.

Step 2

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

An internal transition is added to the state machine. The internal transition is visible in the **Navigation** component.

Step 3

Add two more internal transitions.

Step 4

In the **Navigation** component, click the first internal transition.

Step 4.1

Go to the **Properties** component.

Step 4.2

In the **Trigger** combo box, 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** component, the internal transition is renamed to `Volume up`.

Step 5

In the **Navigation** component, click the second internal transition.

Step 5.1

Go to the **Properties** component.

Step 5.2

In the **Trigger** combo box, 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** component, the internal transition is renamed to `Volume down`.

Step 6

In the **Navigation** component, click the third internal transition.

Step 6.1

Go to the **Properties** component.

Step 6.2

In the **Trigger** combo box, 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** component, the internal transition is renamed to `Close volume control`.

You added three internal transitions which start and stop the dynamic state machine. Furthermore, the internal transitions `Volume up` and `Volume down` change the position of the `Indicator` rectangle.

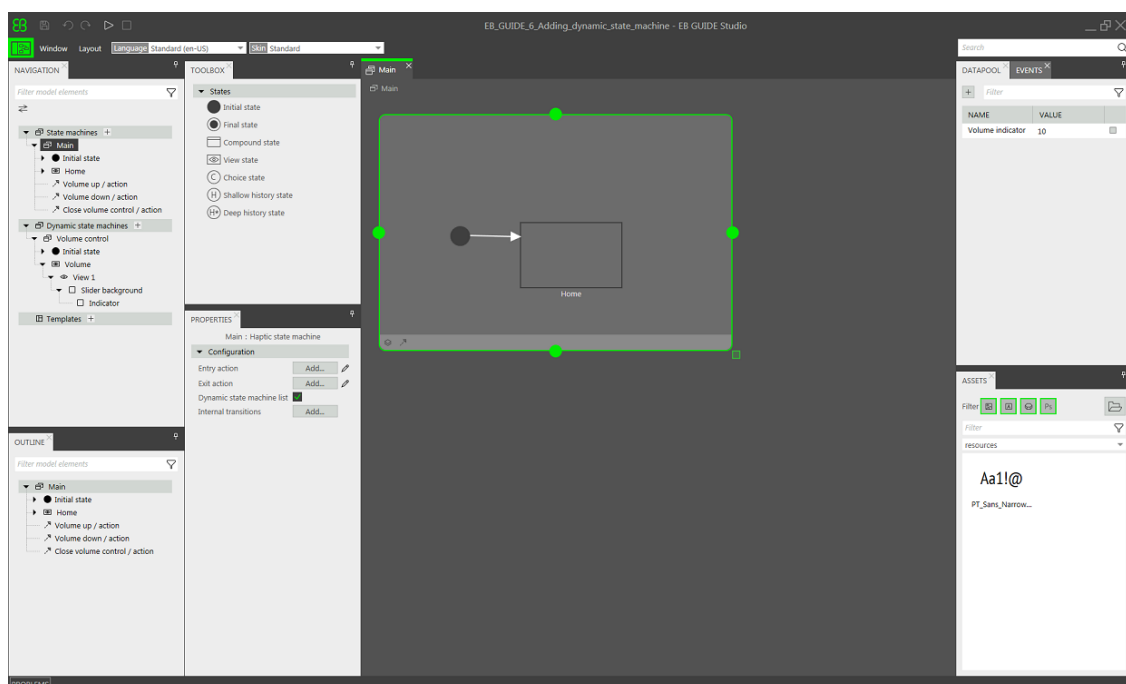


Figure 11.2. EB GUIDE model with all model elements



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 EB GUIDE Monitor in the **Events** component, select the `Volume up` event and click ⚡ to fire the event.

The dynamic state machine is started and shows the slider indicator. The dynamic state machine overlays the `Home` view state.

When you fire the events `Volume up` or `Volume down` the black `Indicator` rectangle moves. If you fire the event `Close volume control`, the slider disappears from the view.

If you add additional states to the **Main** state machine, the `Volume control` dynamic state machine will overlay the other states as well.

11.2. Tutorial: Modeling button behavior with EB GUIDE Script

NOTE



Default window layout

All instructions and screenshots of this user manual use the default window layout. If you want to follow the instructions, we recommend to reset the EB GUIDE Studio window to default layout by selecting **Layout > Reset to default layout**.

With EB GUIDE Script you can express property values, actions, or conditions and evaluate them during run-time.

The following instructions guide you through the process of using EB GUIDE Script to model the behavior of a button. The button increases in size when it is clicked and shrinks back to its original size when it reaches a defined maximum size. For best results, work through the steps in the order presented.

Approximate duration: 10 minutes.



Adding widgets

Prerequisite:

- The **Main** state machine contains an initial state and a view state.
- The initial state has a transition to the view state.
- The content area displays the view.

Step 1

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

Step 2

In the **Navigation** component, click the rectangle, press the **F2** key, and rename the rectangle to `Back-ground`.

Step 3

Drag a rectangle from the **Toolbox** into the **Navigation** component. Place it as a child widget to the `Back-ground` rectangle.

Step 4

In the **Navigation** component, click the new rectangle, press the **F2** key, and rename the rectangle to `But-ton`.

Step 5

Drag a label from the **Toolbox** into the **Navigation** component. Place the label as a child widget to the `But-ton` rectangle.

Step 6

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

Your widget hierarchy now looks as follows.

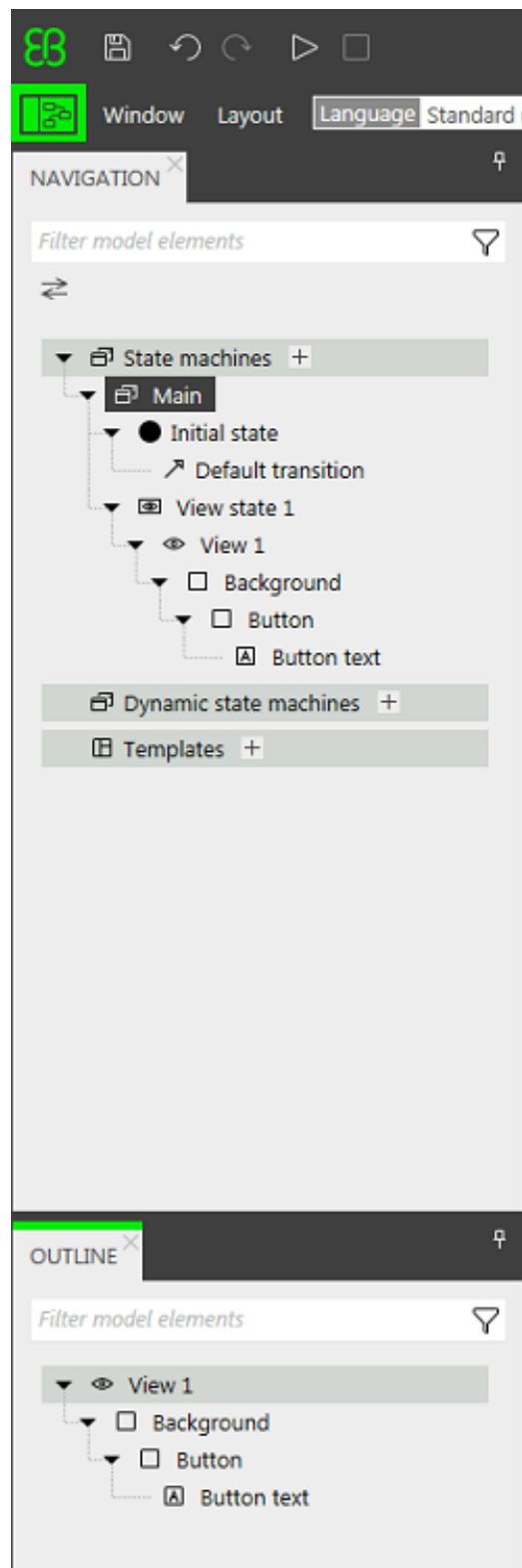


Figure 11.3. Widget hierarchy



Configuring the background

Prerequisite:

- You completed the previous instruction.

Step 1

In the **Navigation** component, click the `Background` rectangle, and go to the **Properties** component.

Step 2

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

A menu expands.

Step 3

In the menu, click **Add link to widget property**.


A dialog opens.

Step 4

In the dialog, go to the view, and select its `width` property.

Step 5

Click **Accept**.

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

Step 6

Link the `height` property of the `Background` rectangle to the `height` property of the view.

Step 7

Link the `x` property of the `Background` rectangle to the `x` property of the view.

Step 8

Link the `y` property of the `Background` rectangle to the `y` property of the view.

The `Background` rectangle covers the exact size and position of the view.



Defining the maximum button width

A datapool item holds the value for the maximum width of the button. It can be changed during run-time.

Prerequisite:

- You completed the previous instruction.

Step 1

In the **Datapool** component, click **+**.

A menu expands.

Step 2

In the menu, click **Integer**.

A new datapool item of type `Integer` is added.

Step 3

Rename the datapool item to `Maximum width`.

Step 4

In the `Value` text box, enter 400.



Configuring the button

Prerequisite:

- You completed the previous instruction.

Step 1

In the **Navigation** component, click the `Button` rectangle, and go to the **Properties** component.

Step 1.1

Enter 50 in the `height` text box.

Step 1.2

Enter 350 in the `x` text box.

Step 1.3

Enter 215 in the `y` text box.

Step 1.4

Select blue for the `fillColor` property.

The button is now colored blue.

Step 2

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

The **Widget features** dialog is displayed.

Step 3

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

Step 4

Click **Accept**.

The related widget feature properties are added to the `Button` rectangle and displayed in the **Properties** component.

Step 5

Next to the `touchPressed` property, click **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** component, click the `Button` text label, and go to the **Properties** component.

Step 2

Enter `grow!` in the `text` text box.

Step 3

Link the `width` property of the `Button` text label to the `width` property of the `Button` rectangle.

Step 4

Link the `height` property of the `Button` text label to the `height` property of the `Button` rectangle.

Step 5

Enter 0 in the `x` text box.

Step 6

Enter 0 in the `y` text box.

Step 7

Next to the `horizontalAlign` property, 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

NOTE



Default window layout

All instructions and screenshots of this user manual use the default window layout. If you want to follow the instructions, we recommend to reset the EB GUIDE Studio window to default layout by selecting **Layout > Reset to default layout**.

Path gestures are shapes drawn by a finger on a touch screen or entered by some other input device.

The following instructions guide you through the process of modeling a path gesture.

Approximate duration: 10 minutes



Adding widgets and configuring default widget properties

Prerequisite:

- The **Main** state machine contains an initial state and a view state.
- The initial state has a transition to the view state.
- The content area displays a view.

Step 1

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

Step 2

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

The label is added as a child widget to the rectangle.

The **Properties** component displays the properties of the label.

Step 3

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

Step 4

Select the rectangle.

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

Step 5

Enter 500 in the `width` text box.

Step 6

In the **Properties** component, go to **fillColor**, and select red.

You added two widgets and configured default widget properties.



Adding widget features to a rectangle

To enable the user to enter a shape starting on the widget, you add the widget feature **Path gesture** to the rectangle. The shape is matched against a set of known shapes and, if a match is found, a gesture is recognized.

Prerequisite:

- You completed the previous instruction.

Step 1

Select the rectangle.

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

Step 2

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

The **Widget features** dialog is displayed.

Step 3

Under **Available widget features**, expand the **Gestures** category, and select `Path gestures`.

The **Touched** widget feature is automatically selected, as it is required for the **Gestures** widget feature.

Step 4

Click **Accept**.

The related widget feature properties are added to the rectangle and displayed in the **Properties** component.

Step 5

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

Step 5.1

Next to the `onPath` property, click **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

NOTE



Default window layout

All instructions and screenshots of this user manual use the default window layout. If you want to follow the instructions, we recommend to reset the EB GUIDE Studio window to default layout by selecting **Layout > Reset to default layout**.

Instantiators allow creating lists dynamically during run-time. Based on a datapool item of 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 **Datapool** component, click **+**.

A menu expands.

Step 2

In the menu, click **String list**.

A new datapool item of type `String list` is added.

Step 3

Rename the datapool item to `MyStringList`.

Step 4

Click the  button.

An editor opens.

Step 4.1

Click **Add**.

A new entry is added to the table.

Step 4.2

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

Step 4.3

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

Step 4.4

Click **Accept**.

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

The content of the list is displayed 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** component, expand the view state and the view.

Step 3

Drag an instantiator from the **Toolbox** into the view. Rename the instantiator to `MyInstantiator`.

Step 4

Drag a rectangle from the **Toolbox** into the instantiator. Rename the rectangle to `MyRectangle`.

Step 5

Drag a label from the **Toolbox** into the rectangle. Rename the label to `MyLabel`.

The widget hierarchy now looks as follows.

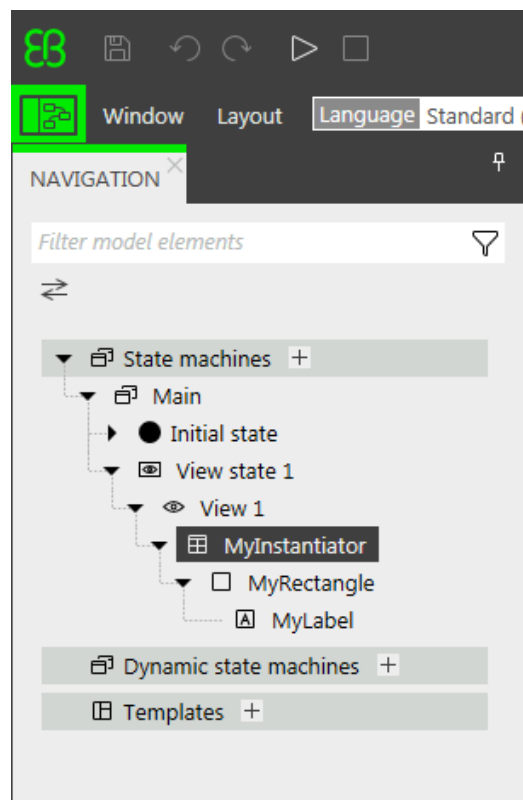


Figure 11.5. Widget hierarchy with an instantiator



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** component.

Step 2

Enter 300 in the `width` text box, and in the `height` text box.

Step 3

Enter 250 in the `x` text box.

Step 4

Enter 150 in the `y` text box.

Step 5

To calculate the length of the list dynamically, add a conditional script.

In the **User-defined properties** category, click **+**.

A menu expands.

Step 5.1

In the menu, click **Conditional script**.

Step 5.2

Rename the property to `calculateNumItems`.

Step 5.3

Next to the `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** component.

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** component.

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** component.

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.

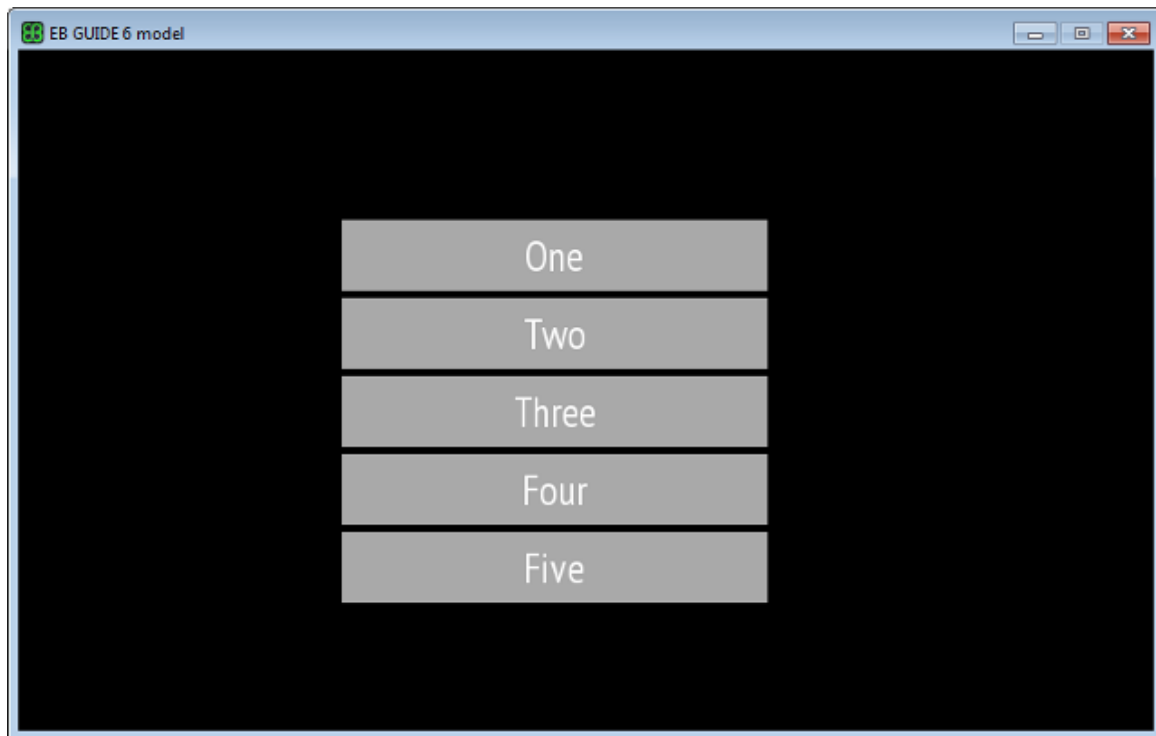


Figure 11.6. List created with an instantiator

11.5. Tutorial: Making an ellipse move across the screen

NOTE



Default window layout

All instructions and screenshots of this user manual use the default window layout. If you want to follow the instructions, we recommend to reset the EB GUIDE Studio window to default layout by selecting **Layout > Reset to default layout**.

The following instructions guide you through the process of animating an ellipse so that it continually moves across the screen when the simulation starts.

Approximate duration: Five minutes.



Adding widgets

In the following steps, you add three widgets to the view and organize the hierarchy of the widgets.

Prerequisite:

- The content area displays the **Main** state machine.
- The **Main** state machine contains an initial state and a view state.
- The initial state has a transition to the view state

Step 1

In the content area, double-click the view state.

The view is displayed in the content area.

Step 2

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

Step 3

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

Step 4

In the **Navigation** component, click the animation, and press the **F2** key. Rename the animation to `MyAnimation`.

Step 5

Drag a linear interpolation integer widget from the **Toolbox** into the **Navigation** component and drop it so that it becomes a child widget of the animation.

Now, if you start the simulation, an ellipse is displayed in a view. The ellipse does not move yet.



Adding a user-defined property of type `Conditional script`

As a next step, you add a user-defined property to the ellipse. With the conditional script property, rendering the ellipse during simulation starts the animation.

Prerequisite:

- You completed the previous instruction.

Step 1

Select the ellipse.

Step 2

In the **Properties** component, go to the **User-defined properties** category, and click **+**.

A menu expands.

Step 3

In the menu, click `Conditional script`.

A user-defined property of type `Conditional script` is added to the ellipse.

Step 4

Rename the property to `startAnimation`.

Step 5

Next to the `startAnimation` property, 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** component, 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 ellipse, and select its `x` property.

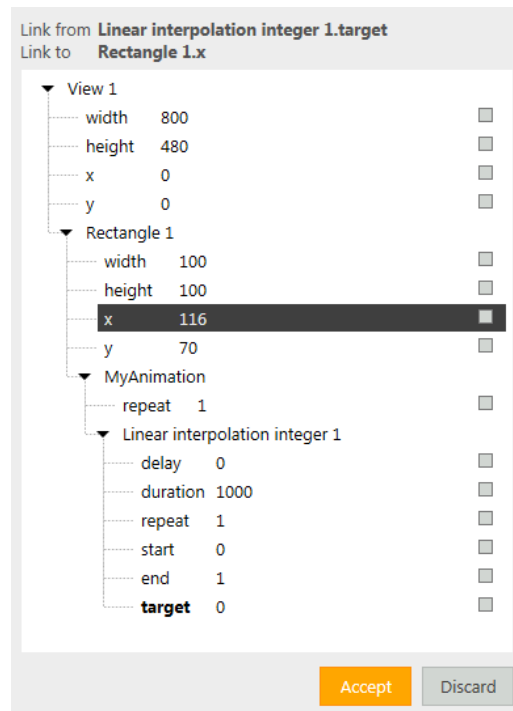



Figure 11.7. Linking between widget properties

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 ellipse changes from zero to the width of the view. Thus the ellipse moves from the left boundary to the right boundary of the view.

Step 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 ellipse 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

NOTE



Default window layout

All instructions and screenshots of this user manual use the default window layout. If you want to follow the instructions, we recommend to reset the EB GUIDE Studio window to default layout by selecting **Layout > Reset to default layout**.

EB GUIDE offers the possibility to display texts in the user's preferred language. The following instructions show you how to model a label that changes with an English, French, and German user interface.

Approximate duration: 15 minutes

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** component, go to the `text` property, and click the  button next to the property.

Step 3

In the menu, click **Add link to datapool item**.

A dialog opens.

Step 4

To add a new datapool item, enter `Welcome_text` in the 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 **Datapool** component, click the `Welcome_text` datapool item.

Step 2

Click the  button.

Step 3

In the menu, click **Add language support**.

In the **Properties** component, the language properties are displayed.

Step 4

In the **Datapool** component, in the `Value` text box, enter `Welcome`.

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

Step 5

Go to the **Properties** component.

Step 6

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

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

Step 7

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

You have added language support for English, German and French and defined a language-dependent text label.



Changing the language during run-time

The following instructions guide you through the process of creating a script for changing the language during run-time. Each time, the user clicks the label, the display language changes.

Prerequisite:

- You completed the previous instruction.

Step 1

In the **Datapool** component, click **+**.

A menu expands.

Step 2

In the menu, click `Integer`.

A datapool item of type `Integer` is added.

Step 3

Rename the datapool item to `SelectedLanguage`.

Step 4

In the **Navigation** component, click the `Label 1` label.

Step 5

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

The **Widget features** dialog is displayed.

Step 6

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

Step 7

Click **Accept**.

The related widget feature properties are added to the label and displayed in the **Properties** component.

Step 8

Next to the `touchPressed` property, click **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 run-time.

Result:

You added a datapool item of type `String` to the EB GUIDE model. The datapool item has different values for languages. In English the value is `Welcome`. In German the value is `Willkommen`. In French the value is `Bienvenue`. The datapool item is linked to the `text` property of the label. Every time you change the language of the EB GUIDE model the text of the label changes too.

11.7. Tutorial: Working with a 3D graphic

NOTE



Default window layout

All instructions and screenshots of this user manual use the default window layout. If you want to follow the instructions, we recommend to reset the EB GUIDE Studio window to default layout by selecting **Layout > Reset to default layout**.

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

The following instructions guide you through the process of adding a 3D graphic to your EB GUIDE model. The instructions show you how to import a 3D graphic and how to modify the appearance of the imported 3D graphic using widget features. For best results, work through the following steps in order presented.

NOTE



3D graphic

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

Only the OpenGL ES version 2.0 or higher and DirectX 11 renderers can display 3D graphics. Make sure that your graphic driver is compatible to the version of the renderer. The supported 3D graphic formats are COLLADA (.dae) and Filmbox (.fbx). For best results, use the Filmbox format.

To be able to apply textures to a mesh, a 3D object needs to have texture coordinates. To add texture coordinates, use third-party 3D modeling software.

Approximate duration: 15 minutes.



Importing a 3D graphic

The following instructions guide you through the process of importing a 3D graphic file to 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. The file contains a camera, a light source, and one object containing a mesh and at least one material.

Step 1

In the content area, double-click the view state.

The view is displayed in the content area.

Step 2

Drag a scene graph from the **Toolbox** into the view.

The view displays the empty bounding box.

Step 3

Rename the scene graph to `My3DGraphic`.

Step 4

In the **Properties** component, click **Import file**.

A dialog opens.

Step 5

Navigate to the 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** component displays the imported widget tree with the scene graph as a parent node. `My3DGraphic` contains a `RootNode` that has at least one mesh with material, camera and several other child widgets depending on the content of your 3D graphic file.



Adding widgets

The following instructions guide you through the process of adding an additional light source to your 3D graphic.

Prerequisite:

- You completed the previous instruction.

Step 1

In the **Navigation** component, expand `RootNode`.

Step 2

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

You added a directional light to `My3DGraphic`. 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** component, click `Mesh 1`, and go to the **Properties** component.

Step 2

From the `mesh` drop-down list box select the `.ebmesh` file from the resource folder mentioned above.

The view displays the scene graph with the new mesh.

Step 3

Alternatively, use the **Assets** component:

Step 3.1

On the **Layout** menu, click **Assets**.

The **Assets** component is opened.

Step 3.2

Drag an `.ebmesh` file from the **Assets** component into the `mesh` drop-down list box.

The view displays the scene graph with the new mesh.



Changing textures

The following instructions guide you through the process of adding and modifying textures of your 3D graphic.

Prerequisite:

- You completed the previous instruction.
- The `$GUIDE_PROJECT_PATH/<project name>/resources/<3D graphic name>` directory contains a `.png` or `.jpg` image file.

Step 1

In the **Navigation** component, click the material, and go to the **Properties** component.

Step 2

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

The **Widget features** dialog is displayed.

Step 3

Under **Available widget features**, expand the **3D** category, and select a texture widget feature, for example **Diffuse texture**.

Step 4

Click **Accept**.

The related widget feature properties are added to the material and displayed in the **Properties** component.

Step 5

In the **Properties** component, select an image from the `diffuseTexture` 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** component, click `My3DGraphic` and go to the **Properties** component.

Step 2

Enter 800 in the `width` text box and 480 in the `height` text box.

The `My3DGraphic` scene graph has the size of the view.

Step 3

In the **Navigation** component, expand `RootNode` and `Camera001`.

Step 4

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

Step 5

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

The **Widget features** dialog is displayed.

Step 6

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

Step 7

Click **Accept**.

The related widget feature properties are added to `Camera 1` and displayed in the **Properties** component.

Step 8

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

You added a second camera.

Step 9

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

Step 10

In the `nearPlane`, `farPlane` and `fieldOfView` text boxes enter the same values that `Camera 1` has.

Both `Camera 1` and `Camera 2` have the same viewing position.

Step 11

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

The **Widget features** dialog is displayed.

Step 12

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

Step 13

Click **Accept**.

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

Step 14

In the **Properties** component, enter 100 in `viewportX` and `viewportY` text boxes.

In the view, the 3D object is displayed two times with different x-coordinate and y-coordinate.

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

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

12.2. Datapool items

Table 12.2. Properties of a datapool item

Property name	Description
Value	The initial value of the datapool item

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 (>=)
- ▶ less (<)
- ▶ less or equal (<=)
- ▶ addition (+)
- ▶ subtraction (-)
- ▶ multiplication (*)
- ▶ division (/)
- ▶ modulo (%)
- ▶ assign (writable properties) (=)

It is possible to store integer properties in a list. For details about lists, see [section 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
<code>color:</code>	A color parameter follows, for example {0,255,255}.
<code>dp:</code>	A datapool item follows.
<code>l:</code>	A language follows.
<code>else</code>	An <code>if</code> condition is completed. The following block is executed as an alternative.
<code>ev:</code>	An event follows.
<code>f:</code>	A user-defined function follows.
<code>false</code>	A boolean literal value
<code>fire</code>	Fires an event
<code>if</code>	A statement which tests a boolean expression follows. If the expression is true, the statement is executed.
<code>in</code>	Is a separator between a local variable declaration and the variable's scope of usage Is used with <code>match_event</code> and <code>let</code> .
<code>function</code>	Declares a function
<code>length</code>	The length of a property
<code>let</code>	Declares a local variable that is accessible in the scope
<code>list</code>	Declares a list type, for example an integer list
<code>match_event</code>	Checks if the current event corresponds to an expected event and declares variables like <code>let</code>
<code>popup_stack</code>	The dynamic state machine list which defines the priority of dynamic state machines
<code>sm:</code>	A state machine follows
<code>true</code>	A boolean literal value
<code>unit</code>	A value of type void
<code>v:</code>	A local variable follows.

Keyword	Description
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
::	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
<code>x</code>	integer	The number to return the absolute value from
<code><return></code>	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><return></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><return></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><return></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
animation	GtfTypeRecord	The animation to manipulate
time	integer	The point in time
<return>	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
animation	GtfTypeRecord	The animation to manipulate
<return>	boolean	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
animation	GtfTypeRecord	The animation to manipulate
<return>	boolean	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
animation	GtfTypeRecord	The animation to manipulate
<return>	boolean	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><return></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><return></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><return></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><return></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
<code><return></code>	<code>boolean</code>	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
<code>x</code>	<code>float</code>	The number to return the arc sine from
<code><return></code>	<code>float</code>	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
<code>y</code>	<code>float</code>	Argument y
<code>x</code>	<code>float</code>	Argument x
<code><return></code>	<code>float</code>	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
<code>y</code>	<code>integer</code>	Argument y

Parameter	Type	Description
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
x	float	The number to return the arc tangent from
<return>	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
value	float	The value to round
<return>	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
state		The state with the dynamic state machine list
sm	integer	The dynamic state machine
priority	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
<code><return></code>	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
<code>state</code>		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
<code>value</code>	color	The color to convert to string
<code><return></code>	string	The color formatted as a string of hexadecimal digits with # as prefix

NOTE



Formatting examples

The format of the returned string is `#RRGGBBAA` with two digits for each of the color channels red, green, blue and alpha.

For example, opaque pure red is converted to `"#ff0000ff"`, semi-transparent pure green is converted to `"#00ff007f"`.

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
<code>x</code>	float	The number to return the cosine from
<code><return></code>	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
<code>x</code>	float	The angle to convert from degrees to radians
<code><return></code>	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
<code>x</code>	float	The exponent
<code><return></code>	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
<code>value</code>	float	The value to convert to string
<code><return></code>	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
<code>value</code>	<code>float</code>	The value to round
<code><return></code>	<code>integer</code>	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
<code><return></code>	<code>void</code>	

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
<code><return></code>	<code>void</code>	

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>	<code>string</code>	<p>A string of the following structure:</p> <p><code>%[flags] [width] [.precision] type</code></p> <ul style="list-style-type: none">▶ flags: Optional character or characters that control output justification and output of signs, blanks, leading zeros, decimal points, and octal and hexadecimal prefixes.▶ width: Optional decimal number that specifies the minimum number of characters that are output.▶ precision: Optional decimal number that specifies the number of significant digits or the number of digits after the decimal-point character .

Parameter	Type	Description
		<ul style="list-style-type: none"> type: Required conversion specifier character that determines whether the associated argument is interpreted as a character, a string, an integer, or a float number.
useDotAsDelimiter	boolean	Defines the delimiter sign. Possible values: <ul style="list-style-type: none"> true: Use a dot as delimiter. false: Use a comma as delimiter.
value	float	The number to format

WARNING Adhere to printf specification for C++



The `format` parameter is defined according to the printf specification for C++.

Using values that do not comply with this specification can lead to unexpected behavior.

For example, allowed types for `format_float` are `f`, `a`, `g` and `e`, and not more than one type character is allowed.

12.4.3.2.14. `format_int`

The function formats an integer value.

Table 12.34. Parameters of `format_int`

Parameter	Type	Description
format	string	A string of the following structure: <code>%[flags] [width] [.precision] type</code> <ul style="list-style-type: none"> flags: Optional character or characters that control output justification and output of signs, blanks, leading zeros, decimal points, and octal and hexadecimal prefixes. width: Optional decimal number that specifies the minimum number of characters that are output. precision: Optional decimal number that specifies the minimum number of digits that are printed. type: Required conversion specifier character that determines whether the associated argument is interpreted as a character, a string, an integer, or a float number.
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 of a text for a widget.

Table 12.35. Parameters of `getLineCount`

Parameter	Type	Description
<code>widget</code>	<code>widget</code>	The widget to evaluate
<code><return></code>	<code>integer</code>	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
<code>text</code>	<code>string</code>	The text to evaluate
<code>font</code>	<code>font</code>	The font to evaluate
<code><return></code>	<code>integer</code>	The height of the text If the size of the font is 0 or negative, the function returns 0.

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
<code>text</code>	<code>string</code>	The text to evaluate
<code><return></code>	<code>integer</code>	The number of characters in the text

NOTE**Escape sequences**

EB GUIDE Script does not resolve escape sequences like `\n` and counts every character. For example, for the text `Label\n` the `getTextLength` function returns 7.

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
<code><return></code>	integer	The width of the text If the size of the font is 0 or negative, the function returns 0.

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
<code><return></code>	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

Parameter	Type	Description
brightness	integer	The brightness in percent
alpha	integer	The alpha value between 0 (totally transparent) and 255 (opaque)
<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. This operation is performed asynchronously.

Table 12.44. Parameters of `language`

Parameter	Type	Description
language	languageType	The language to switch to, for example <code>f:language(l:German)</code>
<return>	void	

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
time	integer	A time stamp as returned by system time
<return>	integer	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
time	integer	A time stamp as returned by system time
<return>	integer	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
time	integer	A time stamp as returned by system time
<return>	integer	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
time	integer	A time stamp as returned by system time
<return>	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
time	integer	A time stamp as returned by system time
<return>	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
time	integer	A time stamp as returned by system time
<return>	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
time	integer	A time stamp as returned by system time
<return>	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><return></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><return></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><return></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><return></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 from radians to degree.

Table 12.58. Parameters of `rad2deg`

Parameter	Type	Description
<code>x</code>	float	The argument
<code><return></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><return></code>	integer	A random number between 0 and $2^{31}-1$

12.4.3.3.20. `shutdown`

The function requests the framework to shutdown the program.

12.4.3.3.21. `rgba2color`

The function converts from RGB color space to GTF color.

Table 12.60. Parameters of `rgba2color`

Parameter	Type	Description
<code>red</code>	integer	The red color coordinate, ranging from 0 to 255
<code>green</code>	integer	The green color coordinate, ranging from 0 to 255
<code>blue</code>	integer	The blue color coordinate, ranging from 0 to 255
<code>alpha</code>	integer	The alpha value, ranging from 0 (totally transparent) to 255 (opaque)
<code><return></code>	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 12.61. Parameters of `round`

Parameter	Type	Description
<code>value</code>	float	The value to round
<code><return></code>	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 12.62. Parameters of `seed_rand`

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

12.4.3.4.2. `sinf`

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

Table 12.63. Parameters of `sinf`

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

12.4.3.4.3. `skin`

The function switches the skin of all datapool items. This operation is performed asynchronously.

Table 12.64. Parameters of `skin`

Parameter	Type	Description
<code>skin</code>	<code>skinType</code>	The skin to switch to, for example <code>f:skin(s:Standard)</code>
<code><return></code>	<code>void</code>	

12.4.3.4.4. `sqrtof`

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

Table 12.65. Parameters of `sqrtof`

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

12.4.3.4.5. `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><return></code>	float	The return value

12.4.3.4.6. `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><return></code>	integer	The return value

12.4.3.4.7. `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><return></code>	string	The language string

12.4.3.4.8. `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
<code>startIndex</code>	integer	The first character index of the result string

Parameter	Type	Description
endIndex	integer	The first character index that is not part of the result
<return>	string	The language string

12.4.3.4.9. `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

12.4.3.4.10. `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

12.4.3.4.11. `tanf`

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

Table 12.72. Parameters of `tanf`

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

12.4.3.4.12. `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
itemId	dp_id	The datapool ID of the item to trace debug information about
<return>	void	

12.4.3.4.13. `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>	string	The text to trace
<code><return></code>	void	

12.4.3.4.14. `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>	widget	The widget to which the coordinates are relative
<code>localX</code>	integer	The x-position of the local coordinate
<code>localY</code>	integer	The y-position of the local coordinate
<code><return></code>	integer	The x-position of the screen coordinate

12.4.3.4.15. `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>	widget	The widget to which the coordinates are relative
<code>localX</code>	integer	The x-position of the local coordinate
<code>localY</code>	integer	The y-position of the local coordinate
<code><return></code>	integer	The y-position of the screen coordinate

12.4.3.4.16. `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
<code>widget</code>	<code>widget</code>	The widget to which the coordinates are relative
<code>screenX</code>	<code>integer</code>	The x-position of the screen coordinate
<code>screenY</code>	<code>integer</code>	The y-position of the screen coordinate
<code><return></code>	<code>integer</code>	The x-position of the local coordinate

12.4.3.4.17. `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
<code>widget</code>	<code>widget</code>	The widget to which the coordinates are relative
<code>screenX</code>	<code>integer</code>	The x-position of the screen coordinate
<code>screenY</code>	<code>integer</code>	The y-position of the screen coordinate
<code><return></code>	<code>integer</code>	The y-position of the local coordinate

12.4.3.4.18. `trunc`

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

Table 12.79. Parameters of `trunc`

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

12.4.3.4.19. `widgetGetChildCount`

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

Table 12.80. Parameters of `widgetGetChildCount`

Parameter	Type	Description
<code>widget</code>	<code>widget</code>	The widget of which to obtain the number of child widgets

Parameter	Type	Description
<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. model.json configuration file

The `model.json` is an EB GUIDE TF configuration file that contains configuration items which are relevant for a single EB GUIDE model.

The `model.json` file is a part of the exported EB GUIDE model.

The following table is used as documentation for all default configuration parameters.

NOTE



JSON object notation

If you configure `model.json` in EB GUIDE Studio, use the JSON object notation.

For an example, see [section 12.6.1, “Example model.json in EB GUIDE Studio”](#).

For more information about JSON format, see <http://www.json.org>.

Table 12.82. Common

Configuration item	Type	Description	Default value
<code>gtf.eventsystem.maxQueue</code>	integer	Maximum size of the event queues	0
<code>gtf.model.traces</code>	boolean	Enables the tracing of the <code>f:trace_string</code> script function	true
<code>gtf.model.identifier</code>	string	Unique identifier of the EB GUIDE mod-	empty

Configuration item	Type	Description	Default value
		el (equal to the EB GUIDE Studio project UUID)	
<code>gtf.model.identifier.short</code>	integer	Short identifier of the EB GUIDE model	0xdeadbeaf

Table 12.83. Files and paths

Configuration item	Type	Description	Default value
<code>gtf.model.path</code>	string	Path to the EB GUIDE model	<gtf_binaries_folder>
<code>gtf.model.config</code>	string	Full path to the EB GUIDE model configuration	<gtf.model.path>/model.json
<code>gtf.datapool.descriptionFile</code>	string	Name of the datapool description file	datapool.gtf
<code>gtf.model.files.sm</code>	string	Name of the state machine description file	model.bin
<code>gtf.model.files.rm</code>	string	Name of the resources description file	resources.bin
<code>gtf.model.files.views</code>	string	Name of the view description file	views.bin
<code>gtf.model.files.types</code>	string	Name of the type description file	types.bin
<code>gtf.model.pluginstoload</code>	string list	Names of EB GUIDE model plugins to load	empty string list
<code>gtf.eventsystem.mapFile</code>	string	Name of the event system mapping file	eventMap.gtf

The option `gtf.model.coreNames` is a string list that contains the names of all configured cores. The following table contains configuration items for every core.

Table 12.84. Cores

Configuration item	Type	Description	Default value
<code>gtf.model.cores.<corename>.own-Thread</code>	boolean	Specifies if the core uses an own thread to run	false
<code>gtf.model.cores.<corename>.id</code>	integer	Specifies the core context identifier	0

The option `gtf.model.sceneNames` is a string list that contains the names of all configured scenes. For every scene, the configuration items in the following table are found.

Table 12.85. Scenes

Configuration item	Type	Description	Default value
<code>gtf.model.scenes.<scenename>.visible</code>	boolean	Determines the visibility of the scene	true
<code>gtf.model.scenes.<scenename>.width</code>	integer	Width of the scene	800
<code>gtf.model.scenes.<scenename>.-height</code>	integer	Height of the scene	480
<code>gtf.model.scenes.<scenename>.x</code>	integer	Coordinates of the scene's starting point	0
<code>gtf.model.scenes.<scenename>.y</code>	integer	Coordinates of the scene's starting point	0
<code>gtf.model.scenes.<scenename>.projectName</code>	string	Name of the working project	
<code>gtf.model.scenes.<scenename>.windowCaption</code>	string	Displayed window name text	
<code>gtf.model.scenes.<scenename>.-sceneId</code>	integer	Identifier for the scene	0
<code>gtf.model.scenes.<scenename>.maxFPS</code>	integer	The redraw rate (FPS = Frames per second). Set to 0 for an unlimited redraw rate.	60
<code>gtf.model.scenes.< scenename>.hwLayerId</code>	integer	Specifies the core context identifier	0
<code>gtf.model.scenes.< scenename>.colorMode</code>	integer	Specifies the color mode: ▶ 1: 32 bit (RGBA8888) ▶ 2: 16 bit (RGB565) ▶ 3: 24 bit (RGB888)	1
<code>gtf.model.scenes.< scenename>.multisampling</code>	integer	Specifies the multisampling of the scene	0

Configuration item	Type	Description	Default value
		<ul style="list-style-type: none"> ▶ 0: no multisampling ▶ 1: 2x multisampling ▶ 2: 4x multisampling 	
<code>gtf.model.scenes.< scenename>.enableRemoteFramebuffer</code>	boolean	If <code>true</code> , the transfer of the off-screen buffer to the simulation window is enabled	false
<code>gtf.model.scenes.< scene-name>.showWindowFrame</code>	boolean	Determines if the renderer window frame should be displayed	true
<code>gtf.model.scenes.< scene-name>.showWindow</code>	boolean	If <code>true</code> , an additional window for simulation is opened on Windows based systems	true
<code>gtf.model.scenes.< scenename>.disableVsync</code>	boolean	If <code>true</code> , the vertical synchronization for the renderer is disabled.	false
<code>gtf.model.scenes.<scenename>.-showFPS</code>	integer	Possible values: <ul style="list-style-type: none"> ▶ 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 	0
<code>gtf.model.scenes.<scenename>.renderer</code>	string	Name of the renderer to use: <code>DirectXRenderer</code> , <code>OpenGLRenderer</code> or <code>OpenGL3Renderer</code>	

Table 12.86. Rendering common

Configuration item	Type	Description	Default value
<code>gtf.model.fontCache.width</code>	integer	Width of the font cache atlas texture	512
<code>gtf.model.fontCache.height</code>	integer	Height of the font cache atlas texture	512
<code>gtf.model.fontCache.age</code>	integer	Maximum allowed age before the refresh operation of the font cache has to be done	100
<code>gtf.model.traversalStackSize</code>	integer	The renderers traversal stack size in bytes	32768

The configuration items in the following table belong together. This means that the renderer expects that the same amount of items is in all three lists. The entry with an index in one list belongs to the entries with the same index in other lists.

Table 12.87. Renderer display extensions

Configuration item	Type	Description	Default value
<code>gtf.model.displayId</code>	integer list	Identifiers of the scenes	
<code>gtf.model.maxCacheSize</code>	integer list	Maximum texture caches for the scenes	
<code>gtf.model.driverName</code>	string list	OS specific driver names for the scenes (e.g. <code>/dev/fb0</code>)	

The configuration items in the following table are used to configure the `TextEngine` component. `TextEngine` is based on the FreeType third-party library. The following parameters are passed to the FreeType implementation. For more information about FreeType, see https://www.freetype.org/freetype2/docs/reference/ft2-cache_subsystem.html.

Due to the way EB GUIDE TF handles font sizes, `ft_size` objects are not cached separately from `ft_face` objects. Consider that the values for `max_sizes` can be limited by the hardware of your target platform.

Table 12.88. `TextEngine` configuration items

Configuration item	Type	Description	Default value
<code>gtf.model.textengine.replacementGlyph</code>	integer list	Unicode character that should be used in case the dedicated	0

Configuration item	Type	Description	Default value
		font character is not found in the current font	
<code>gtf.model.textengine.maxFaces</code>	integer list	Maximum amount of cached font faces	0
<code>gtf.model.textengine.maxSizes</code>	integer list	Maximum amount of cached font sizes	0
<code>gtf.model.textengine.maxBytes</code>	integer list	Maximum amount of memory in bytes that can be used for caches	0

The option `gtf.model.touchDevicesNames` is a string list containing the names of all configured touch devices. For every touch device the configuration items listed in the following table are available.

Table 12.89. Touch devices

Configuration item	Type	Description	Default value
<code>gtf.model.touchDevices.< device-Name>.touchscreenType</code>	integer	Defines the touch device type: <ul style="list-style-type: none"> ▶ 0: Galaxy ▶ 1: imx WVGA ▶ 2: Mouse ▶ 3: General ▶ 4: Lilliput_889GL ▶ 5: GeneralMultitouch ▶ 6: Lilliput with automatic calibration ▶ 7: Generic-TouchConfiguration 	3
<code>gtf.model.touchDevices.< device-Name>.displayManagerId</code>	integer	Specifies the scene ID for which the device is valid	0

Configuration item	Type	Description	Default value
<code>gtf.model.touchDevices.< device-Name>.touchId</code>	integer	Specifies the ID of the device	0
<code>gtf.model.touchDevices.< device-Name>.minimalDistanceToMove</code>	integer	Threshold for reacting on touch position changes	0
<code>gtf.model.touchDevices.< device-Name>.touchMoveRepeatTimeout</code>	integer	Delay between touch position change notifications	0
<code>gtf.model.touchDevices.< device-Name>.width</code>	integer	Width of the touchable device area	0
<code>gtf.model.touchDevices.< device-Name>.height</code>	integer	Height of the touchable device area	0
<code>gtf.model.touchDevices.< device-Name>.x_high</code>	integer	Maximum horizontal resolution extend of the touchable device area	0
<code>gtf.model.touchDevices.< device-Name>.y_high</code>	integer	Maximum vertical resolution extend of the touchable device area	0
<code>gtf.model.touchDevices.< device-Name>.x_low</code>	integer	Minimal horizontal resolution extend of the touchable device area	0
<code>gtf.model.touchDevices.< device-Name>.y_low</code>	integer	Minimal vertical resolution extend of the touchable device area	0
<code>gtf.model.touchDevices.< device-Name>.devicePath</code>	string	Name of the driver used for touch (e.g. /dev/input0)	

12.6.1. Example `model.json` in EB GUIDE Studio



Example 12.1. `model.json` in EB GUIDE Studio

```
{
```



```
"gtf":{
  "datapool":{
    "descriptionFile":"datapool.gtf"
  },
  "eventsystem":{
    "maxQueue":0,
    "mapFile":"eventMap.gtf"
  },
  "model":{
    "coreNames":[
      "<core_1>"
    ],
    "cores":{
      "<core_1>":{
        "ownThread":false,
        "id":0
      }
    },
    "touchDevicesNames":[
      "<device_1>"
    ],
    "touchDevices":{
      "<device_1>":{
        "touchscreenType":3,
        "displayManagerId":0,
        "touchId":0,
        "minimalDistanceToMove":0,
        "touchMoveRepeatTimeout":0,
        "width":0,
        "height":0,
        "x_high":0,
        "y_high":0,
        "x_low":0,
        "y_low":0,
        "devicePath":""
      }
    },
    "displayId":[
    ],
    "driverName":[
    ],
    "fontCache":{
      "width":512,
      "height":512,
      "age":100
    }
  }
}
```

```
    },
    "maxCacheSize": [

    ],
    "sceneNames": [
        "<scene_1>"
    ],
    "scenes": {
        "<scene_1>": {
            "visible": true,
            "width": 800,
            "height": 480,
            "x": 0,
            "y": 0,
            "projectName": "<project_x>",
            "windowCaption": "<Displayed window name text>",
            "sceneId": 0,
            "maxFPS": 60,
            "hwLayerId": 0,
            "colorMode": 1,
            "multisampling": 0,
            "enableRemoteFramebuffer": false,
            "showWindowFrame": true,
            "showWindow": true,
            "disableVsync": false,
            "showFPS": 0,
            "renderer": "DirectXRenderer"
        }
    },
    "traces": true,
    "traversalStackSize": 32768,
    "identifier": "",
    "path": "<gtf_binaries_folder>",
    "config": "<gtf.model.path>/model.json",
    "files": {
        "sm": "model.bin",
        "rm": "resources.bin",
        "views": "views.bin",
        "types": "types.bin"
    },
    "pluginstoload": [

    ]
}
}
```

12.7. platform.json configuration file

The `platform.json` is an EB GUIDE TF configuration file which contains common and platform dependent items.

The `platform.json` file is a part of the exported EB GUIDE model.

The following table is used as documentation for all default configuration parameters.

NOTE



JSON object notation

If you configure `platform.json` within EB GUIDE Studio, use the JSON object notation.

For an example, see [section 12.7.1, “Example platform.json in EB GUIDE Studio”](#).

For more information about JSON format, see <http://www.json.org>.

Table 12.90. Platform configuration

Configuration item	Type	Description	Default value
<code>gtf.servicemapper.port</code>	integer	Connection port for the services (e.g. EB GUIDE Monitor)	60000
<code>gtf.core.pluginstoload</code>	string list	List of core plugins that should be loaded (relative to binary folder or absolute path)	None
<code>gtf.launcher.editmode</code>	boolean	Defines if EB GUIDE TF is running in EB GUIDE Studio. This is a read-only item.	false
<code>gtf.platform.config</code>	string	Full path to the <code>platform.json</code> file. This is a read-only item.	<binary_folder>/platform.json
<code>gtf.framework.path</code>	string	Path to the <code>GtfS-tartup</code> executable. This is a read-only item.	<gtf_binaries_folder>
<code>gtf.diagnostic.memory.interval</code>	integer	Specifies the time interval for the memory	0

Configuration item	Type	Description	Default value
		diagnostic. If value is 0 the diagnostic is deactivated.	
<code>gtf.ipc.role</code>	string	The role of the IPC node. Possible values are <code>server</code> or <code>client</code>	<code>server</code>
<code>gtf.ipc.discovery.network</code>	string	The IPv4 network address which will be used for the server-client discovery mechanism. In case of direct connection, this represents the servers' network address.	<code>255.255.255.255</code>
<code>gtf.ipc.discovery.port</code>	integer	The network port which will be used for the server-client discovery mechanism. In case of direct connection, this has to be equal to the item <code>gtf.servicemap-per.port</code> from the server configuration.	<code>4711</code>
<code>gtf.ipc.discovery.mode</code>	string	The discovery mode used for connecting the server and the clients. Possible options are: " <code>broadcast</code> ", " <code>multicast</code> " and " <code>direct</code> ".	<code>broadcast</code>
<code>gtf.ipc.client.timeout</code>	integer	Retry period of the client connection to the server, expressed in milliseconds.	<code>5000</code>

12.7.1. Example `platform.json` in EB GUIDE Studio



Example 12.2. `platform.json` in EB GUIDE Studio

```
{
  "gtf":{
    "core":{
      "pluginstoload":[
        "TfRuntime",
        "TfService",
        "TfGui",
        "TfGUIOpenGLS20",
        "TfGUIOpenGLS3",
        "TfGUIDirectX11"
      ]
    },
    "servicemapper":{
      "port":60000
    },
    "launcher":{
      "editmode":true
    },
    "platform":{
      "config":"<binary_folder>/platform.json"
    },
    "framework":{
      "path":"<gtf_binaries_folder>"
    },
    "diagnostic":{
      "memory":{
        "interval":0
      }
    },
    "ipc":{
      "role":"server",
      "discovery":{
        "network":"255.255.255.255",
        "port":4711,
        "mode":"broadcast"
      },
      "client":{
        "timeout":5000
      }
    }
  }
}
```

```
}  
}  
}
```

12.8. Scenes

Table 12.91. Properties of a scene

Property name	Description
height	The height of the area in which the views of a haptic state machine are rendered on a target device
width	The width of the area in which the views of a haptic state machine are rendered on a target device
x	The x-offset of the area in which the views of a haptic state machine are rendered on a target device
y	The y-offset of the area in which the views of a haptic state machine are rendered on a target device
visible	If true, the state machine and its child widgets are visible.
projectName	The name of the EB GUIDE project
windowCaption	The text that is shown on the window frame
sceneID	The unique scene identifier which can be used, for example, for input handling
maxFPS	The redraw rate (FPS = Frames per second) Set to 0 for an unlimited redraw rate.
hwLayerID	The ID of the hardware layer on the target device's display that is mapped to the current state machine
colorMode	Possible values: <ul style="list-style-type: none"> ▶ 32 bit (=1): RGBA8888 ▶ 16 bit (=2): RGB565 ▶ 24 bit (=3): RGB888
multisampling	Possible values: <ul style="list-style-type: none"> ▶ Off (= 0): no multisampling

Property name	Description
	<ul style="list-style-type: none">▶ 2x (=1): 2x multisampling▶ 4x (=2): 4x multisampling
<code>enableRemoteFramebuffer</code>	If true, transfer of the off-screen buffer to the simulation window is enabled
<code>showWindowFrame</code>	If true, a frame is displayed on the simulation window. The frame allows the window to be grabbed and moved.
<code>showWindow</code>	If true, an additional window for simulation is opened on Windows based systems.
<code>disableVSync</code>	If true, vertical synchronization for the renderer is disabled.
<code>showFPS</code>	Possible values: <ul style="list-style-type: none">▶ 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
<code>Renderer</code>	Defines a renderer for the scene. Possible values: <ul style="list-style-type: none">▶ 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.

12.9. Touch screen types supported by EB GUIDE GTF

The supported types depend on the target device.

Table 12.92. 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

12.10. Widgets

12.10.1. View

Table 12.93. Properties of a view

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
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 12.94. 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 Move in from left , Fade in from center or Show view immediately .
Duration	The duration of the entry animation in milliseconds
Delay	The delay of the entry animation in milliseconds

Property name	Description
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 12.95. 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 Move out to top , Fade out to center or Hide view immediately .
Duration	The duration of the exit animation in milliseconds
Delay	The delay of the exit animation in milliseconds

12.10.2. Basic widgets

There are six basic widgets.

- ▶ Container
- ▶ Ellipse
- ▶ Image
- ▶ Instantiator
- ▶ Label
- ▶ Rectangle

The following sections list the properties of basic widgets.

NOTE



Unique names

Use unique names for two widgets with the same parent widget.

NOTE



Negative values

Do not use negative values for `height` and `width` properties. EB GUIDE Studio treats negative values as 0, this means the respective widget will not be depicted.

12.10.2.1. Container

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

Table 12.96. 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

12.10.2.2. Ellipse

An ellipse draws a colored ellipse with the dimensions and coordinates of the widget into a view. The widget can also be used to draw a sector or an arc.

Table 12.97. Properties of the ellipse

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 ellipse
arcWidth	The width of the arc of the ellipse
centralAngle	The angle in degrees which defines a sector of the ellipse
sectorRotation	The angle in degrees which describes the rotation of the ellipse's sector

12.10.2.3. Image

An image places a picture into a view.

Table 12.98. Properties of the image

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 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 version 2.0 or higher support .png files and .jpg files.

12.10.2.4. Instantiator

An instantiator creates widget instances during run-time. You can use the instantiator to model lists or tables with dynamic or static content. The child widgets of an instantiator serve as line templates for the list or table which is created during run-time. By default the instantiator only instantiates the first line template.

Table 12.99. 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 widgets. If <code>numItems</code> is 0, no child widgets are created.
lineMapping	Defines which child widget is the line template for which line, i.e. defines the order of instantiation

12.10.2.5. Label

A label places text into a view.

Table 12.100. Properties of the label

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
text	The text the label displays. If the text does not fit into the widget area it is truncated at the end by default.
textColor	The color in which the text is displayed
font	The font in which the text is displayed
horizontalAlign	The horizontal alignment of the text within the boundaries of the label
verticalAlign	The vertical alignment of the text within the boundaries of the label

12.10.2.6. Rectangle

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

Table 12.101. 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

12.10.3. Animations

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

12.10.3.1. Animation

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

Table 12.102. 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	<p>The reaction that is executed when the animation completes. First parameter: Animation time. Second parameter: Reason for the termination, encoded as follows:</p> <ul style="list-style-type: none"> ▶ 0: Animation is completed ▶ 1: Animation is cancelled, triggered by <code>f:animation_cancel</code> ▶ 2: Widget is destroyed due to view transition ▶ 3: Animation jumps to its last step, triggered by <code>f:animation_cancel_end</code> ▶ 4: Animation jumps to its first step and is then canceled, triggered by <code>f:animation_cancel_reset</code>

12.10.3.2. Constant curves

A constant curve is a child widget of an animation. A constant curve sets a target value after a defined delay. Constant curves are available for integer, boolean, float, and color types.

Table 12.103. 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

Property name	Description
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
value	The resulting constant value

12.10.3.3. Fast start curves

A fast start curve is a child widget of an animation. 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 12.104. 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
start	The initial value
end	The final value

12.10.3.4. Slow start curves

A slow start curve is a child widget of an animation. 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 12.105. Properties of slow start curves

Property name	Description
name	The name of the curve

Property name	Description
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

12.10.3.5. Quadratic curves

A quadratic curve is a child widget of an animation. A quadratic curve periodically sets a value using a quadratic function curve. Quadratic curves are available for integer, float, and color types.

Table 12.106. 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

12.10.3.6. Sinus curves

A sinus curve is a child widget of an animation. A sinus curve periodically sets a value using a sinus function curve. Sinus curves are available for integer, float, and color types.

Table 12.107. 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

12.10.3.7. Script curves

A script curve is a child widget of an animation. 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 12.108. 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

12.10.3.8. Linear curves

A linear curve is a child widget of an animation. A linear curve periodically sets a value using a linear progression curve. Linear curves are available for integer, float, and color types.

Table 12.109. 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

12.10.3.9. Linear interpolation curves

A linear interpolation curve is a child widget of an animation. A linear interpolation curve widget periodically sets a value using a linear interpolation curve. Linear interpolation curves are available for integer, float, and color types.

NOTE



Linear key value interpolation curves

During import of a 3D graphic file, if the imported 3D scene has animations, linear key value interpolation integer curve and linear key value interpolation float curve are created. The underlying key-value pairs of these curves cannot be modified in EB GUIDE Studio.

Table 12.110. 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

Property name	Description
start	The initial value
end	The final value

12.10.4. 3D widgets

12.10.4.1. Scene graph

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

Table 12.111. 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

12.10.4.2. 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 12.112. 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

Property name	Description
rotationZ	The rotation around the z-axis
scalingX	The scaling along the x-axis
scalingY	The scaling along the y-axis
scalingZ	The scaling along the z-axis
translationX	The translation along the x-axis
translationY	The translation along the y-axis
translationZ	The translation along the z-axis

12.10.4.3. 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 12.113. 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

12.10.4.4. Directional light

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

Table 12.114. 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

12.10.4.5. Material

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

Table 12.115. 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
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.

12.10.4.6. Mesh

A mesh defines the shape of the 3D object.

Table 12.116. 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

12.10.4.7. Point light

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

Table 12.117. 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

12.10.4.8. Spot light

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

Table 12.118. 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

12.11. Widget features

The following list contains a description of all widget features that are implemented, with a brief description on how to use them in an EB GUIDE model.

12.11.1. Common

12.11.1.1. 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 12.119. Properties of the **Child visibility selection** widget feature

Property name	Description
containerIndex	The index of the child widgets of the parent widget
containerMapping	<p>If a mapping is set, each child of the container is re-addressed by its appropriate value in <code>containerMapping</code>.</p> <p>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</p>

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

12.11.1.2. Enabled

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

Table 12.120. Properties of the **Enabled** widget feature

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

12.11.1.3. Focused

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

Table 12.121. 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">▶ <code>not focusable (=0)</code>▶ <code>only by touch (=1)</code>▶ <code>only by key (=2)</code>▶ <code>focusable (=3)</code>
<code>focused</code>	If true, the widget has focus

12.11.1.4. Multiple lines

The **Multiple lines** widget feature enables line breaks.

Restrictions:

- ▶ The **Multiple lines** widget feature is only available for the label widget.

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.

Property name	Description
	When the <code>line gap</code> is too small (high negative value), it has no effect anymore and the text is rendered in one line. This occurs for example, when the font style is set to <code>PT_Sans_Narrow</code> , size is set to 30 and the <code>line gap</code> is defined as -50.
<code>maxLineCount</code>	The maximum number of visible lines. 0 = no limitation

TIP



Number of lines used

With the script function `getLineCount`, you can obtain the number of lines of the text.

For more information on this, see [section 12.4.3.2.15, “getLineCount”](#).

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 `' '`.

12.11.1.5. Pressed

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

Restrictions:

- ▶ Adding the **Pressed** widget feature automatically adds the **Focused** widget feature.

Table 12.123. 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.

12.11.1.6. 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 12.124. Properties of the **Selected** widget feature

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

12.11.1.7. Selection group

The **Selection group** widget feature is used to model a list of radio buttons. In the list, every radio button has the **Selection group** widget feature and a unique button ID.

Use a datapool item for the `buttonValue` property. Assign the datapool item to all widgets in the radio button array.

Selecting and deselecting a widget within the button group can be done by an application that sets the `buttonValue` property. Alternatively, changes can be triggered by touch or key input as well as by adding a condition that sets the button value.

Restrictions:

- ▶ Adding the **Selection group** widget feature automatically adds the **Selected** widget feature.

Table 12.125. 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.

12.11.1.8. 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 12.126. 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
<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

Property name	Description
steps	The number of steps to calculate the increment or decrement for the <code>currentValue</code> property
valueWrapAround	Possible values: <ul style="list-style-type: none">▶ <code>true</code>: The <code>currentValue</code> property continues at the inverse border, if <code>minValue</code> or <code>maxValue</code> is exceeded.▶ <code>false</code>: The <code>currentValue</code> property does not decrease/increase, if <code>minValue</code> or <code>maxValue</code> is exceeded.

12.11.1.9. 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 feature enables a different truncation than the default setting `trailing`.

Restrictions:

- ▶ The **Text truncation** widget feature is only available for the label widget.

Table 12.127. Properties of the **Text truncation** widget feature

Property name	Description
truncationPolicy	<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">▶ <code>leading (=0)</code>: Text is replaced at the beginning of the text▶ <code>trailing (=1)</code>: Text is replaced at the end of the text <p>For multi-line texts, the <code>truncationPolicy</code> property defines where text is replaced. Possible values:</p> <ul style="list-style-type: none">▶ <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.▶ <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.
truncationSymbol	The string that is shown instead of the replaced text part

12.11.1.10. Touched

The **Touched** widget feature enables a widget to react to touch input.

Table 12.128. 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. Possible values:</p> <ul style="list-style-type: none"> ▶ <code>Press then react (=0)</code>: Press first, then the widget reacts. Notifications of moving and releasing are only active within the widget area. ▶ <code>Press and grab (=1)</code>: Press to grab the contact. The contact remains grabbed even if it moves away from the widget area. ▶ <code>Press then react on contact (=3)</code>: Even if the contact enters the pressed state outside the widget boundaries, the subsequent move and release events are delivered to the widget.
<code>touchBehavior</code>	<p>Defines touch evaluation. Possible values:</p> <ul style="list-style-type: none"> ▶ <code>Whole area (=0)</code>: To identify the touched widget, the renderer evaluates the widget's clipping rectangle. ▶ <code>Visible pixels (=1)</code>: To identify the touched widget, the renderer evaluates the widget the touched pixel belongs to. <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 (=0)`. EB GUIDE GTF evaluates `Whole area (=0)` faster than `Visible pixels (=1)`.

12.11.2. Effect

12.11.2.1. Border

The **Border** widget feature adds a configurable border to the widget. The border starts at the widget boundaries and is placed within the widget.

Restrictions:

- ▶ The widget feature is available for rectangles.

Table 12.129. 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

12.11.2.2. Coloration

The **Coloration** widget feature colors the widget and its widget subtree. It also affects transparency if the alpha value is not opaque.



Example 12.3. Usage of the Coloration widget feature

For all colors with RGBA components between 0.0 and 1.0, the algorithm in the **Coloration** widget feature multiplies the current color values of a widget by the `colorationColor` property value. Multiplication is done per pixel and component-wise.

A semi-transparent gray colored by an opaque blue results in semi-transparent darker blue as follows:

$$(0.5, 0.5, 0.5, 0.5) * (0.0, 0.0, 1.0, 1.0) = (0.0, 0.0, 0.5, 0.5)$$

Table 12.130. 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">▶ Pure▶ Opaque▶ White

12.11.3. Focus

The **Focus** widget feature category provides the widget features relating to focus management.

12.11.3.1. Auto focus

With the **Auto focus** widget feature, the order in which child widgets are focused is pre-defined. The **Auto focus** widget feature checks the widget subtree for child widgets with the `focusable` property.

The order of the widgets in the layout is used to calculate focus order. Depending on layout orientation, the algorithm begins in the upper left or upper right corner.

Restrictions:

- ▶ The widget feature **Auto focus** automatically adds the **Focused** widget feature.

Table 12.131. 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">▶ <code>stop at hierarchy (=0)</code>▶ <code>wrap within hierarchy level (=1)</code>▶ <code>step up in hierarchy (=2)</code>
<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.

12.11.3.2. User-defined focus

The **User-defined focus** widget feature enables additional focus functionality for the widget. A widget that uses the feature manages a local focus hierarchy for its widget subtree.

Restrictions:

- ▶ The widget feature **User-defined focus** automatically adds the **Focused** widget feature.

Table 12.132. 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.

Property name	Description
	<p>Child widgets that are not focusable are skipped by default. Order in which the child widgets are focused:</p> <ul style="list-style-type: none"> ▶ defined: User-defined widget order is used ▶ not defined: Default widget order is used instead <p>Each child widget requires the Focused 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.</p>
<code>focusPrevious</code>	The trigger that assigns the focus to the previous child
<code>focusFlow</code>	<p>The behavior for focus changes within the hierarchy. Possible values:</p> <ul style="list-style-type: none"> ▶ <code>stop at hierarchy level (=0)</code> ▶ <code>wrap within hierarchy level (=1)</code> ▶ <code>step up in hierarchy (=2)</code>
<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

12.11.4. Gestures

12.11.4.1. Flick gesture

A quick brush of a contact over a surface

Restrictions:

- ▶ Adding the **Flick gesture** widget feature automatically adds the **Gestures** and **Touched** widget features.

Table 12.133. 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"> ▶ <code>speed</code>: relative speed of the flick gesture <p>Speed in pixels/ms divided by <code>flickMinLength/flickMaxTime</code></p>

Property name	Description
	<ul style="list-style-type: none">▶ <code>directionX</code>: x-part of the direction vector of the gesture▶ <code>directionY</code>: y-part of the direction vector of the gesture
<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

12.11.4.2. Hold gesture

A hold gesture without movement

Restrictions:

- ▶ Adding the **Hold gesture** widget feature automatically adds the **Gestures** and **Touched** widget features.
- ▶ The **Hold gesture** widget feature does not trigger the **Touch lost** widget feature.

Table 12.134. 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">▶ <code>x</code>: x-coordinate of the contact position▶ <code>y</code>: y-coordinate of the contact position
<code>holdDuration</code>	The minimal time in milliseconds the contact must stay in place for the gesture to be recognized as a hold gesture

12.11.4.3. Long hold gesture

A long hold gesture without movement

Restrictions:

- ▶ Adding the **Long hold gesture** widget feature automatically adds the **Gestures** and **Touched** widget features.
- ▶ The **Long hold gesture** widget feature does not trigger the **Touch lost** widget feature.

Table 12.135. Properties of the **Long hold gesture** widget feature

Property name	Description
<code>onGestureLongHold</code>	<p>The reaction that is triggered once the gesture is recognized. The reaction is triggered only once per contact: when <code>longHoldDuration</code> has expired and the contact still is in a small boundary box around the initial touch position.</p> <p>Reaction arguments:</p> <ul style="list-style-type: none">▶ <code>x</code>: x-coordinate of the contact position▶ <code>y</code>: y-coordinate of the contact position
<code>longHoldDuration</code>	The minimal time in milliseconds the contact must stay in place for the gesture to be recognized as a long hold gesture

12.11.4.4. Path gestures

A shape drawn by one contact is matched against a set of known shapes.

Restrictions:

- ▶ Adding the **Path gesture** widget feature automatically adds the **Gestures** and **Touched** widget features.

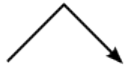






Table 12.136. Properties of the **Path gesture** widget feature

Property name	Description
<code>onPath</code>	The reaction that is triggered when the entered shape matches. The reaction is only triggered if <code>onPathStart</code> has been triggered already.
<code>onPathStart</code>	<p>The reaction that is triggered once a contact moves beyond the minimal box (<code>pathMinXBox</code>, <code>pathMinYBox</code>.) Reaction argument:</p> <ul style="list-style-type: none">▶ <code>gestureId</code>: ID of the path that was matched
<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

12.11.4.4.1. Gesture IDs

Gesture identifiers depend on the configuration of the path gesture recognizer. The following table shows an example configuration which is included in EB GUIDE.

Table 12.137. 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
5		Wave shape left to right
6		Wave shape right to left

12.11.4.5. Pinch gesture

Two contacts that move closer together or further apart

Restrictions:

- ▶ Adding the **Pinch gesture** widget feature automatically adds the **Gestures** and **Touched** widget features.

Table 12.138. Properties of the **Pinch gesture** widget feature

Property name	Description
<code>onGesturePinchStart</code>	<p>The reaction that is triggered once the start of the gesture is recognized. Reaction arguments:</p> <ul style="list-style-type: none"> ▶ <code>ratio</code>: Current contact distance to initial contact distance ratio ▶ <code>centerX</code>: x-coordinate of the current center point between the two contacts ▶ <code>centerY</code>: y-coordinate of the current center point between the two contacts
<code>onGesturePinchUpdate</code>	<p>The reaction that is triggered when the pinch ratio or center point change. Reaction arguments:</p> <ul style="list-style-type: none"> ▶ <code>ratio</code>: Current contact distance to initial contact distance ratio ▶ <code>centerX</code>: x-coordinate of the current center point between the two contacts ▶ <code>centerY</code>: y-coordinate of the current center point between the two contacts
<code>onGesturePinchEnd</code>	<p>The reaction that is triggered once the gesture is finished. Reaction arguments:</p> <ul style="list-style-type: none"> ▶ <code>ratio</code>: Current contact distance to initial contact distance ratio ▶ <code>centerX</code>: x-coordinate of the current center point between the two contacts ▶ <code>centerY</code>: y-coordinate of the current center point between the two contacts
<code>pinchThreshold</code>	<p>The minimal distance in pixels each contact has to move from its initial position for the gesture to be recognized</p>

12.11.4.6. Rotate gesture

Two contacts that move along a circle

Restrictions:

- ▶ Adding the **Rotate gesture** widget feature automatically adds the **Gestures** and **Touched** widget features.

Table 12.139. Properties of the **Rotate gesture** widget feature

Property name	Description
<code>onGestureRotateStart</code>	The reaction that is triggered once the start of the gesture is recognized
<code>onGestureRotateUpdate</code>	The reaction that is triggered when the recognized angle or center point changes
<code>onGestureRotateEnd</code>	The reaction that is triggered once the gesture is finished
<code>rotateThreshold</code>	The minimal distance in pixels each contact has to move from its initial position for the start of the gesture to be recognized

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

12.11.5. Input handling

12.11.5.1. Gestures

The **Gestures** widget feature enables the widget to react on touch gestures.

Restrictions:

- ▶ Adding the **Gestures** widget feature automatically adds the **Touched** widget feature.
- ▶ The **Gestures** widget feature has no additional properties.

12.11.5.2. Key pressed

The **Key pressed** widget feature enables a widget to react on a key being pressed.

Restrictions:

- ▶ Adding the **Key pressed** widget feature automatically adds the **Pressed** and **Focused** widget features.

Table 12.140. Properties of the **Key pressed** widget feature

Property name	Description
<code>keyPressed</code>	The widget's reaction on a key being pressed

Property name	Description
	Reaction argument: <ul style="list-style-type: none">▶ <code>keyId</code>: The ID of the key that is processed

12.11.5.3. Key released

The **Key released** widget feature enables a widget to react on a key being released.

Restrictions:

- ▶ Adding the **Key released** widget feature automatically adds the **Pressed** and **Focused** widget features.

Table 12.141. Properties of the **Key released** widget feature

Property name	Description
<code>keyShortReleased</code>	The widget's reaction on a key being released Reaction argument: <ul style="list-style-type: none">▶ <code>keyId</code>: The ID of the key that is processed

12.11.5.4. Key status changed

The **Key status changed** widget feature enables a widget to react on a key being pressed or released. It defines the reaction to key input such as **short press**, **long**, **ultra long** and **continuous**.

Restrictions:

- ▶ Adding the **Key status changed** widget feature automatically adds the **Pressed** and **Focused** widget features.

Table 12.142. 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">▶ <code>keyId</code>: The ID of the key that is processed▶ <code>status</code>: The numeric ID of the status change

12.11.5.5. Key unicode

The **Key unicode** widget feature enables a widget to react on Unicode key input.

Restrictions:

- ▶ Adding the **Key unicode** widget feature automatically adds the **Pressed** and **Focused** widget features.

Table 12.143. Properties of the **Key unicode** widget feature

Property name	Description
keyUnicode	The widget's reaction on a Unicode key input Reaction argument: <ul style="list-style-type: none">▶ <code>keyId</code>: The ID of the key that is processed

12.11.5.6. Move in

The **Move in** widget feature enables a widget to react on movement into its boundaries.

Restrictions:

- ▶ Adding the **Move in** widget feature automatically adds the **Touched** widget feature.

Table 12.144. 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">▶ <code>touchId</code>: The ID of the touch screen the user has clicked or released▶ <code>fingerId</code>: The ID of the contact that moves across the widget

12.11.5.7. Move out

The **Move out** widget feature enables a widget to react on movement out of its boundaries.

Restrictions:

- ▶ Adding the **Move out** widget feature automatically adds the **Touched** widget feature.

Table 12.145. Properties of the **Move out** widget feature

Property name	Description
moveOut	The widget's reaction on a movement out of its boundaries Reaction arguments:

Property name	Description
	<ul style="list-style-type: none">▶ <code>touchId</code>: The ID of the touch screen the user has clicked or released▶ <code>fingerId</code>: The ID of the contact that moves across the widget

12.11.5.8. Move over

The **Move over** widget feature enables a widget to react on movement within its boundaries.

Restrictions:

- ▶ Adding the **Move over** widget feature automatically adds the **Touched** widget feature.

Table 12.146. Properties of the **Move over** widget feature

Property name	Description
<code>moveOver</code>	<p>The widget's reaction on a movement within its boundaries</p> <p>Reaction arguments:</p> <ul style="list-style-type: none">▶ <code>touchId</code>: The ID of the touch screen the user has clicked or released▶ <code>fingerId</code>: The ID of the contact that moves across the widget

12.11.5.9. Moveable

The **Moveable** widget feature enables a widget to be moved by touch.

Restrictions:

- ▶ Adding the **Moveable** widget feature automatically adds the **Touched** and **Touch moved** widget features.

Table 12.147. Properties of the **Moveable** widget feature

Property name	Description
<code>moveDirection</code>	<p>The direction into which the widget moves. Possible values:</p> <ul style="list-style-type: none">▶ <code>horizontal</code> (=0)▶ <code>vertical</code> (=1)▶ <code>free</code> (=2)

12.11.5.10. Rotary

The **Rotary** widget feature enables a widget to react on being rotated.

Restrictions:

- ▶ Adding the **Rotary** widget feature automatically adds the **Focused** widget feature.

Table 12.148. Properties of the **Rotary** widget feature

Property name	Description
<code>rotaryReaction</code>	<p>The widget's reaction on being rotated. If true, the widget reacts on an incoming rotary event.</p> <p>Reaction arguments:</p> <ul style="list-style-type: none">▶ <code>rotaryId</code>: integer ID▶ <code>increment</code>: number of units the rotary input shifts when the incoming event is sent

12.11.5.11. Touch lost

The **Touch lost** widget feature enables a widget to react on a lost touch contact.

A contact can disappear when it is part of a gesture or leaves the touch screen without releasing. In these cases the `touchShortReleased` reaction is not executed.

Restrictions:

- ▶ Adding the **Touch lost** widget feature automatically adds the **Touched** widget feature.

Table 12.149. Properties of the **Touch lost** widget feature

Property name	Description
<code>onTouchGrabLost</code>	<p>The widget's reaction on a lost touch contact</p> <p>Reaction arguments:</p> <ul style="list-style-type: none">▶ <code>touchId</code>: The ID of the touch screen the user has clicked or released▶ <code>fingerId</code>: The ID of the contact that moves across the widget

12.11.5.12. Touch move

The **Touch move** widget feature enables a widget to react on being touched and moved.

Restrictions:

- ▶ Adding the **Touch move** widget feature automatically adds the **Touched** widget feature.

Table 12.150. 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">▶ touchId: The ID of the touch screen the user has clicked or released▶ fingerId: The ID of the contact that moves across the widget

12.11.5.13. Touch pressed

The **Touch pressed** widget feature enables a widget to react on being pressed.

Restrictions:

- ▶ Adding the **Touch pressed** widget feature automatically adds the **Touched** widget feature.

Table 12.151. Properties of the **Touch pressed** widget feature

Property name	Description
touchPressed	<p>The widget's reaction on being pressed</p> <p>Reaction arguments:</p> <ul style="list-style-type: none">▶ touchId: The ID of the touch screen the user has clicked or released▶ fingerId: The ID of the contact that moves across the widget

12.11.5.14. Touch released

The **Touch released** widget feature enables a widget to react on being released.

Restrictions:

- ▶ Adding the **Touch released** widget feature automatically adds the **Touched** widget feature.

Table 12.152. Properties of the **Touch released** widget feature

Property name	Description
touchShortReleased	<p>The widget's reaction on being released</p> <p>Reaction arguments:</p> <ul style="list-style-type: none">▶ touchId: The ID of the touch screen the user has clicked or released▶ fingerId: The ID of the contact that moves across the widget

12.11.5.15. Touch status changed

The **Touch status changed** widget feature enables a widget to react on changes of its touch status.

Restrictions:

- ▶ Adding the **Touch status changed** widget feature automatically adds the **Touched** widget feature.

Table 12.153. Properties of the **Touch status changed** widget feature

Property name	Description
<code>touchStatusChanged</code>	<p>The widget's reaction on changes of its touch status</p> <p>Reaction arguments:</p> <ul style="list-style-type: none">▶ <code>touchId</code>: The ID of the touch screen the user has clicked or released▶ <code>touchStatus</code>: The ID of the type of touch <p>Possible values:</p> <ul style="list-style-type: none">▶ 0: new contact▶ 1: touch press▶ 2: touch move▶ 3: touch released▶ 4: movement without touch▶ 5: touch gone▶ 6: any status change <ul style="list-style-type: none">▶ <code>fingerId</code>: The ID of the contact that moves across the widget

12.11.6. Layout

12.11.6.1. Absolute layout

The **Absolute layout** widget feature of a parent widget defines the position and size of the child widgets. Invisible child widgets are ignored. The added widget feature properties consist of integer lists. Each list element is mapped to one child widget.

Restrictions:

- ▶ The **Absolute layout** widget feature excludes the following widget features:
 - ▶ **Box layout**

- ▶ **Flow layout**
- ▶ **Grid layout**
- ▶ **List layout**

Table 12.154. 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.

12.11.6.2. Box layout

The **Box layout** widget feature defines position and size of each child widget.

Position and size properties of child widgets are set by the parent widget. Invisible child widgets are ignored in the calculation.

Restrictions:

- ▶ The **Box layout** widget feature excludes the following widget features:
 - ▶ **Absolute layout**
 - ▶ **Flow layout**
 - ▶ **Grid layout**
 - ▶ **List layout**

Table 12.155. 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.

12.11.6.3. Flow layout

The **Flow layout** widget feature defines position and size of each child widget.

Position and size properties of child widgets are set by the parent widget. Invisible child widgets are ignored in the calculation.

Restrictions:

- ▶ The **Flow layout** widget feature excludes the following widget features:
 - ▶ **Absolute layout**
 - ▶ **Box layout**
 - ▶ **Grid layout**
 - ▶ **List layout**

Table 12.156. Properties of the **Flow layout** widget feature

Property name	Description
horizontalGap	The horizontal space between two child widgets
verticalGap	The vertical space between two child widgets
layoutDirection	The direction in which the list elements i.e. the child widgets are positioned.
horizontalChildAlign	The horizontal alignment of child widgets
verticalChildAlign	The vertical alignment of child widgets <ul style="list-style-type: none">▶ <code>center (=0)</code>: The child widget is placed in the center.▶ <code>top (=1)</code>: The child widget is placed at the top▶ <code>bottom (=2)</code>: The child widget is placed at the bottom.

12.11.6.4. Grid layout

The **Grid layout** widget feature defines position and size of each child widget.

Position and size properties of child widgets are set by the parent widget. Invisible child widgets are ignored in the calculation.

Restrictions:

- ▶ The **Grid layout** widget feature excludes the following widget features:
 - ▶ **Absolute layout**
 - ▶ **Box layout**
 - ▶ **Flow layout**
 - ▶ **List layout**

Table 12.157. Properties of the **Grid layout** widget feature

Property name	Description
horizontalGap	The horizontal space between two child widgets
verticalGap	The vertical space between two child widgets
numRows	Defines the number of rows
numColumns	Defines the number of columns

12.11.6.5. Layout margins

The **Layout margins** widget feature adds configurable margins to a widget that uses the **Flow layout**, **Absolute layout**, **Box layout**, or **Grid layout** widget feature.

Table 12.158. Properties of the **Layout margins** widget feature

Property name	Description
leftMargin	The margin of the left border
topMargin	The margin of the top border
rightMargin	The margin of the right border
bottomMargin	The margin of the bottom border

12.11.6.6. 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, see [section 12.11.7.2, “List index”](#).

Restrictions:

- ▶ The **List layout** widget feature is intended to be used with instantiator.
- ▶ The **List layout** widget feature excludes the following widget features:
 - ▶ **Absolute layout**
 - ▶ **Box layout**
 - ▶ **Flow layout**
 - ▶ **Grid layout**

Table 12.159. 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
<code>listLength</code>	The number of list elements
<code>wrapAround</code>	Possible values: <ul style="list-style-type: none"> ▶ true: The <code>scrollValue</code> property continues at the inverse border, if <code>scrollValueMin</code> or <code>scrollValueMax</code> is exceeded. ▶ false: The <code>scrollValue</code> property does not decrease/increase, if <code>scrollValueMin</code> or <code>scrollValueMax</code> is exceeded.

12.11.6.7. Scale mode

The **Scale mode** widget feature defines how an image is displayed if its size differs from the size of the widget.

Restrictions:

- ▶ The **Scale mode** widget feature is only available for the widget image.

Table 12.160. 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">▶ 0 = original size▶ 1 = fit to size▶ 2 = keep aspect ratio

12.11.7. List management

12.11.7.1. Line index

The **Line index** widget feature defines the unique position for each line of your list or table.

Restrictions:

- ▶ The **Line index** widget feature is intended to be used in combination with instantiators.

Table 12.161. Properties of the **Line index** widget feature

Property name	Description
lineIndex	The index of the current line in a table

12.11.7.2. List index

The **List index** widget feature defines the unique position of a widget in a list.

Restrictions:

- ▶ The **List index** widget feature is intended to be used in combination with the **List layout** widget feature.

Table 12.162. Properties of the **List index** widget feature

Property name	Description
listIndex	The index of the current widget in a list

12.11.7.3. Template index

The **Template index** widget feature defines the unique position of the used line template.

Restrictions:

- ▶ The **Template index** widget feature is intended to be used in combination with instantiators.

Table 12.163. Properties of the **Template index** widget feature

Property name	Description
lineTemplateIndex	The index of the used line template

12.11.7.4. Viewport

The **Viewport** widget feature clips oversized elements at the widget borders.

Restrictions:

- ▶ The **Viewport** widget feature is intended to be used in combination with containers or lists.
- ▶ The **Viewport** widget feature takes effect on the following model elements:
 - ▶ Child widgets of the widget you added **Viewport** to are clipped inside the dimensions of the widget.
 - ▶ The widget you added **Viewport** is clipped inside the dimensions of its parent view.

Table 12.164. Properties of the **Viewport** widget feature

Property name	Description
xOffset	The horizontal offset of the visible clipping within the drawn area of child widgets
yOffset	The vertical offset of the visible clipping within the drawn area of child widgets

12.11.8. 3D

Widget features in the **3D** category are only available for 3D widgets.

12.11.8.1. Camera viewport

The **Camera viewport** widget feature defines the camera's drawing region within the scene graph.

Restrictions:

- ▶ The **Camera viewport** widget feature is available for camera.

Table 12.165. Properties of the **Camera viewport** widget feature

Property name	Description
viewportX	The x-origin of the viewport within the scene graph

Property name	Description
viewportY	The y-origin of the viewport within the scene graph
viewportWidth	The viewport's width in pixels
viewportHeight	The viewport's height in pixels

12.11.8.2. Ambient texture

The **Ambient texture** widget feature adds extended configuration values to a material.

Restrictions:

- ▶ The **Ambient texture** widget feature is available for material.

Table 12.166. 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. Possible values: <ul style="list-style-type: none"> ▶ <code>repeat (=0)</code>: When accessed outside the texture bounds, the texture is repeated. Also known as wrap or tile ▶ <code>clamp (=1)</code>: When accessed outside the texture bounds, the pixels at the edge of the texture are used.
ambientTextureAddressModeV	The address mode of the texture along the v-direction. Possible values: <ul style="list-style-type: none"> ▶ <code>repeat (=0)</code>: When accessed outside the texture bounds, the texture is repeated. Also known as wrap or tile ▶ <code>clamp (=1)</code>: When accessed outside the texture bounds, the pixels at the edge of the texture are used.
ambientFilterMode	The filtering mode of the texture. Possible values: <ul style="list-style-type: none"> ▶ <code>point (=0)</code>: Texture is not smoothed at all. Least expensive but prone to aliasing artifacts when texture is minimized. ▶ <code>linear (=1)</code>: Also known as bilinear filtering. Smoothens the texture when minimized to reduce aliasing artifacts. ▶ <code>trilinear (=2)</code>: Most expensive, but yields better results than linear filtering.

12.11.8.3. Diffuse texture

The **Diffuse texture** widget feature adds extended configuration values to a material.

Restrictions:

- ▶ The **Diffuse texture** widget feature is available for material.

Table 12.167. Properties of the **Diffuse texture** widget feature

Property name	Description
<code>diffuseTexture</code>	The file name of the texture
<code>diffuseTextureAddressModeU</code>	<p>The address mode of the texture along the u-direction. Possible values:</p> <ul style="list-style-type: none"> ▶ <code>repeat (=0)</code>: When accessed outside the texture bounds, the texture is repeated. Also known as wrap or tile ▶ <code>clamp (=1)</code>: When accessed outside the texture bounds, the pixels at the edge of the texture are used.
<code>diffuseTextureAddressModeV</code>	<p>The address mode of the texture along the v-direction. Possible values:</p> <ul style="list-style-type: none"> ▶ <code>repeat (=0)</code>: When accessed outside the texture bounds, the texture is repeated. Also known as wrap or tile ▶ <code>clamp (=1)</code>: When accessed outside the texture bounds, the pixels at the edge of the texture are used.
<code>diffuseFilterMode</code>	<p>The filtering mode of the texture. Possible values:</p> <ul style="list-style-type: none"> ▶ <code>point (=0)</code>: Texture is not smoothed at all. Least expensive but prone to aliasing artifacts when texture is minimized. ▶ <code>linear (=1)</code>: Also known as bilinear filtering. Smoothens the texture when minimized to reduce aliasing artifacts. ▶ <code>trilinear (=2)</code>: Most expensive, but yields better results than linear filtering.

12.11.8.4. Emissive texture

The **Emissive texture** widget feature adds extended configuration values to a material.

Restrictions:

- ▶ The **Emissive texture** widget feature is available for material.

Table 12.168. Properties of the **Emissive texture** widget feature

Property name	Description
<code>emissiveTexture</code>	The file name of the texture

Property name	Description
<code>emissiveTextureAddress-ModeU</code>	The address mode of the texture along the u-direction. Possible values: <ul style="list-style-type: none"> ▶ <code>repeat (=0)</code>: When accessed outside the texture bounds, the texture is repeated. Also known as wrap or tile ▶ <code>clamp (=1)</code>: When accessed outside the texture bounds, the pixels at the edge of the texture are used.
<code>emissiveTextureAddressModeV</code>	The address mode of the texture along the v-direction. Possible values: <ul style="list-style-type: none"> ▶ <code>repeat (=0)</code>: When accessed outside the texture bounds, the texture is repeated. Also known as wrap or tile ▶ <code>clamp (=1)</code>: When accessed outside the texture bounds, the pixels at the edge of the texture are used.
<code>emissiveFilterMode</code>	The filtering mode of the texture. Possible values: <ul style="list-style-type: none"> ▶ <code>point (=0)</code>: Texture is not smoothed at all. Least expensive but prone to aliasing artifacts when texture is minimized. ▶ <code>linear (=1)</code>: Also known as bilinear filtering. Smoothens the texture when minimized to reduce aliasing artifacts. ▶ <code>trilinear (=2)</code>: Most expensive, but yields better results than linear filtering.

12.11.8.5. Light map texture

The **Light map texture** widget feature adds extended configuration values to a material.

Restrictions:

- ▶ The **Light map texture** widget feature is available for material.

Table 12.169. Properties of the **Light map texture** widget feature

Property name	Description
<code>lightMapTexture</code>	The file name of the texture
<code>lightMapTextureAddress-ModeU</code>	The address mode of the texture along the u-direction. Possible values: <ul style="list-style-type: none"> ▶ <code>repeat (=0)</code>: When accessed outside the texture bounds, the texture is repeated. Also known as wrap or tile ▶ <code>clamp (=1)</code>: When accessed outside the texture bounds, the pixels at the edge of the texture are used.

Property name	Description
lightMapTextureAddressModeV	The address mode of the texture along the v-direction. Possible values: <ul style="list-style-type: none"> ▶ <code>repeat (=0)</code>: When accessed outside the texture bounds, the texture is repeated. Also known as wrap or tile ▶ <code>clamp (=1)</code>: When accessed outside the texture bounds, the pixels at the edge of the texture are used.
lightMapFilterMode	The filtering mode of the texture. Possible values: <ul style="list-style-type: none"> ▶ <code>point (=0)</code>: Texture is not smoothed at all. Least expensive but prone to aliasing artifacts when texture is minimized. ▶ <code>linear (=1)</code>: Also known as bilinear filtering. Smoothens the texture when minimized to reduce aliasing artifacts. ▶ <code>trilinear (=2)</code>: Most expensive, but yields better results than linear filtering.

12.11.8.6. Normal map texture

The **Normal map** widget feature adds extended configuration values to a material.

Restrictions:

- ▶ The **Normal map texture** widget feature is available for material.

Table 12.170. Properties of the **Normal map** widget feature

Property name	Description
normalMapTexture	The file name of the texture
normalMapTextureAddressModeU	The address mode of the texture along the u-direction. Possible values: <ul style="list-style-type: none"> ▶ <code>repeat (=0)</code>: When accessed outside the texture bounds, the texture is repeated. Also known as wrap or tile ▶ <code>clamp (=1)</code>: When accessed outside the texture bounds, the pixels at the edge of the texture are used.
normalMapTextureAddressModeV	The address mode of the texture along the v-direction. Possible values: <ul style="list-style-type: none"> ▶ <code>repeat (=0)</code>: When accessed outside the texture bounds, the texture is repeated. Also known as wrap or tile ▶ <code>clamp (=1)</code>: When accessed outside the texture bounds, the pixels at the edge of the texture are used.
normalMapFilterMode	The filtering mode of the texture. Possible values:

Property name	Description
	<ul style="list-style-type: none"> ▶ <code>point (=0)</code>: Texture is not smoothed at all. Least expensive but prone to aliasing artifacts when texture is minimized. ▶ <code>linear (=1)</code>: Also known as bilinear filtering. Smoothens the texture when minimized to reduce aliasing artifacts. ▶ <code>trilinear (=2)</code>: Most expensive, but yields better results than linear filtering.

12.11.8.7. Opaque texture

The **Opaque texture** widget feature adds extended configuration values to a material.

Restrictions:

- ▶ The **Opaque texture** widget feature is available for material.

Table 12.171. Properties of the **Opaque texture** widget feature

Property name	Description
<code>opaqueTexture</code>	The file name of the texture
<code>opaqueTextureAddressModeU</code>	<p>The address mode of the texture along the u-direction. Possible values:</p> <ul style="list-style-type: none"> ▶ <code>repeat (=0)</code>: When accessed outside the texture bounds, the texture is repeated. Also known as wrap or tile ▶ <code>clamp (=1)</code>: When accessed outside the texture bounds, the pixels at the edge of the texture are used.
<code>opaqueTextureAddressModeV</code>	<p>The address mode of the texture along the v-direction. Possible values:</p> <ul style="list-style-type: none"> ▶ <code>repeat (=0)</code>: When accessed outside the texture bounds, the texture is repeated. Also known as wrap or tile ▶ <code>clamp (=1)</code>: When accessed outside the texture bounds, the pixels at the edge of the texture are used.
<code>opaqueFilterMode</code>	<p>The filter mode of the texture. Possible values:</p> <ul style="list-style-type: none"> ▶ <code>point (=0)</code>: Texture is not smoothed at all. Least expensive but prone to aliasing artifacts when texture is minimized. ▶ <code>linear (=1)</code>: Also known as bilinear filtering. Smoothens the texture when minimized to reduce aliasing artifacts. ▶ <code>trilinear (=2)</code>: Most expensive, but yields better results than linear filtering.

12.11.8.8. Reflection texture

The **Reflection texture** widget feature adds extended configuration values to a material.

Restrictions:

- ▶ The **Reflection texture** widget feature is available for material.

Table 12.172. Properties of the **Reflection texture** widget feature

Property name	Description
<code>reflectionTopTexture</code>	The file name of the texture
<code>reflectionBottomTexture</code>	The file name of the texture
<code>reflectionLeftTexture</code>	The file name of the texture
<code>reflectionRightTexture</code>	The file name of the texture
<code>reflectionFrontTexture</code>	The file name of the texture
<code>reflectionBackTexture</code>	The file name of the texture
<code>reflectionFilterMode</code>	The filtering mode of the texture. Possible values: <ul style="list-style-type: none">▶ <code>point (=0)</code>: Texture is not smoothed at all. Least expensive but prone to aliasing artifacts when texture is minimized.▶ <code>linear (=1)</code>: Also known as bilinear filtering. Smoothens the texture when minimized to reduce aliasing artifacts.▶ <code>trilinear (=2)</code>: Most expensive, but yields better results than linear filtering.

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.

12.11.8.9. Specular texture

The **Specular texture** widget feature adds extended configuration values to a material.

Restrictions:

- ▶ The **Specular texture** widget feature is available for material.

Table 12.173. Properties of the **Specular texture** widget feature

Property name	Description
<code>specularTexture</code>	The file name of the texture
<code>specularTextureAddressModeU</code>	<p>The address mode of the texture along the u-direction. Possible values:</p> <ul style="list-style-type: none"> ▶ <code>repeat (=0)</code>: When accessed outside the texture bounds, the texture is repeated. Also known as wrap or tile ▶ <code>clamp (=1)</code>: When accessed outside the texture bounds, the pixels at the edge of the texture are used.
<code>specularTextureAddressModeV</code>	<p>The address mode of the texture along the v-direction. Possible values:</p> <ul style="list-style-type: none"> ▶ <code>repeat (=0)</code>: When accessed outside the texture bounds, the texture is repeated. Also known as wrap or tile ▶ <code>clamp (=1)</code>: When accessed outside the texture bounds, the pixels at the edge of the texture are used.
<code>specularFilterMode</code>	<p>The filtering mode of the texture. Possible values:</p> <ul style="list-style-type: none"> ▶ <code>point (=0)</code>: Texture is not smoothed at all. Least expensive but prone to aliasing artifacts when texture is minimized. ▶ <code>linear (=1)</code>: Also known as bilinear filtering. Smoothens the texture when minimized to reduce aliasing artifacts. ▶ <code>trilinear (=2)</code>: Most expensive, but yields better results than linear filtering.

12.11.9. Transformation

The widget features of the category **Transformation** modify location, form, and size of widgets.

The order in which transformations are executed is equal to the order in the widget tree. If multiple transformations are applied to one widget at the same widget tree hierarchy level, the order is as follows:

1. Translation
2. Shearing
3. Scaling

4. Rotation around z-axis
5. Rotation around y-axis
6. Rotation around x-axis

12.11.9.1. Pivot

The **Pivot** widget feature defines the pivot point of transformations which are applied to the widget. If no pivot point is configured, the default pivot point is at (0.0, 0.0, 0.0).

Restrictions:

- Adding the **Pivot** widget feature automatically adds the **Rotation**, **Scaling** and **Shearing** widget features.

Table 12.174. Properties of the **Pivot** widget feature

Property name	Description
<code>pivotX</code>	The pivot point on the x-axis relative to parent widget
<code>pivotY</code>	The pivot point on the y-axis relative to parent widget
<code>pivotZ</code>	The pivot point on the z-axis relative to parent widget if widget is a scene graph

12.11.9.2. Rotation

The **Rotation** widget feature is used to rotate the widget and its subtree.

Table 12.175. Properties of the **Rotation** widget feature

Property name	Description
<code>rotationEnabled</code>	Defines whether rotation is used or not
<code>rotationAngleX</code>	The rotation angle on the x-axis. This property only affects scene graph.
<code>rotationAngleY</code>	The rotation angle on the y-axis. This property only affects scene graph.
<code>rotationAngleZ</code>	The rotation angle on the z-axis

12.11.9.3. Scaling

The **Scaling** widget feature is used to scale the widget and its subtree.

Table 12.176. Properties of the **Scaling** widget feature

Property name	Description
<code>scalingEnabled</code>	Defines whether scaling is used or not

Property name	Description
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

12.11.9.4. Shearing

The **Shearing** widget feature is used to distort widgets in the widget subtree.

Table 12.177. 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

12.11.9.5. Translation

The **Translation** widget feature is used to translate the widget and its subtree. It moves widgets in x, y and z directions.

Table 12.178. Properties of the **Translation** widget feature

Property name	Description
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 EB GUIDE

To download the community edition of EB GUIDE, go to <https://www.elektrobit.com/ebguide/try-eb-guide/> and follow the instructions.

To download the enterprise edition of EB GUIDE, go to EB Command.

NOTE



Activate your account

After ordering a product, you receive an email from sales department. Click the link in the email. Follow the steps to create an account as directed in the email and in the browser, then proceed to log in.

EB Command is the server from which you are going to download the EB GUIDE product line software. For the instructions on how to download from EB Command, see <https://www.elektrobit.com/support/download-ing-from-eb-command/>.

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 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	<p>A model element is an object within an EB GUIDE model, for example a state, a widget, or a datapool item.</p> <p>See Also EB GUIDE model.</p>
---------------	--

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 `model.json` 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

- .psd file format, 126
- 3D graphic, 33, 55, 210, 312
 - add, 125
 - import, 210
 - mesh, 55
 - supported formats, 33, 55
- 3D object, 33
- 3D widget, 55, 96
- 3D widgets, 33
 - reference, 270

A

- absolute layout
 - reference, 292
- action
 - entry action, 104
 - exit action, 104
 - transition, 115
- ambient texture
 - reference, 299
- animation, 35, 96, 144, 146, 203
 - entry animation, 36, 146
 - exit animation, 36, 146
 - reference, 264
- API, 312 (see application programming interface)
- application programming interface, 36
- auto focus
 - reference, 280
- auto-hide, 45

B

- basic widget, 96
- basic widgets
 - reference, 261
- boolean
 - data type, 217
- boolean list
 - data type, 218

- border
 - reference, 278
- box layout
 - reference, 293
- button
 - user interface, 71

C

- camera
 - reference, 271
- camera viewport
 - reference, 298
- child visibility selection
 - reference, 273
- choice state, 107
- color
 - data type, 218
- coloration
 - reference, 279
- command area
 - project editor, 44
- command line, 71, 163, 170, 177
- communication context, 37, 155, 312
- component
 - docking, 45
 - undocking, 45
- compound state, 106
- condition
 - transition, 113
- conditional script
 - data type, 218
- configuration file, 246, 255
- configure
 - display, 174
- console (see command line)
- constant curve
 - reference, 265
- container
 - add, 122
 - reference, 261
- content area
 - project center, 38

project editor, 42

copy

datapool item, 154

event, 151

D

data type

boolean, 217

boolean list, 218

color, 218

conditional script, 218

float, 219

font, 219

image, 219

integer, 220

list, 220

mesh, 217

mesh list, 217

string, 221

datapool, 48, 312

datapool item, 48, 154, 312

add, 153

change, 164

copy, 154

export, 175

import, 176

language support, 207

link, 156

list, 154

paste, 154

reference, 217

windowed list, 49

diffuse texture

reference, 299

directional light

reference, 271

display

configure, 174

docking

component, 45

dynamic state machine

add, 103, 178

E

EB GUIDE extension, 51

EB GUIDE GTF, 312

EB GUIDE GTF SDK, 312

EB GUIDE model, 49, 312

model element, 49

EB GUIDE Monitor, 46, 163, 163, 164, 164, 165, 168

command line, 169

datapool component, 164

datapool item, 164

event, 164

events component, 164

scripting component, 165

tabs, 46

EB GUIDE product line, 312

EB GUIDE project, 49

EB GUIDE Script, 55, 155, 312

comment, 57

datapool access, 64

event, 67

expression, 58

foreign function call, 63

identifier, 56

if-then-else, 62

l-value, 60

list, 66

local variable, 60

namespace, 56

r-value, 60

scripted value, 69

standard library, 69

string formatting, 68

tutorial, 186

types, 57

while loop, 61

widget property, 65

EB GUIDE SDK, 312

EB GUIDE Studio, 312

EB GUIDE Studio SDK, 312

EB GUIDE TF, 312

effect

widget feature, 278

- ellipse
 - reference, 262
- emissive texture
 - reference, 300
- enabled
 - reference, 274
- entry action, 108
 - state machine, 104
- entry animation, 146
 - reference, 260
- event, 50, 66
 - add, 151
 - copy, 151
 - fire, 163
 - paste, 151
 - reference, 246
- event system, 50
- exit action, 109
 - state machine, 104
- exit animation, 146
 - reference, 261
- export, 169
 - language-dependent text, 175

F

- fast start curve
 - reference, 266
- finger ID, 94
- flick gesture
 - reference, 281
- float
 - data type, 219
- flow layout
 - reference, 293
- focused
 - reference, 274
- font, 53
 - data type, 219

G

- gesture, 93
 - non-path gesture, 93

- path gesture, 93
 - reference, 281, 286
- gesture ID
 - reference, 283
- GL, 313
- grid layout
 - reference, 294
- GUI, 313

H

- HMI, 313
- hold gesture
 - reference, 282

I

- icon
 - user interface, 71
- image
 - 9-patch, 54
 - add, 119
 - data type, 220
 - reference, 262
 - supported formats, 54
- import
 - language-dependent text, 176
- instantiator, 196
 - add, 123
 - line template, 123, 263
 - reference, 263
- integer
 - data type, 220
- internal transition, 116

K

- key pressed
 - reference, 286
- key released
 - reference, 287
- key status changed
 - reference, 287
- key unicode
 - reference, 287

L

- label, 121
 - add, 121
 - font, 121, 122
 - reference, 263
- Language
 - change, 207
- language-dependent text, 207
 - export, 175
 - import, 176
- layout margins
 - reference, 295
- library, 313
 - add, 172
- light map texture
 - reference, 301
- line index
 - reference, 297
- linear curve, 268
- linear interpolation curve, 269
- linear interpolation integer, 203
- link
 - datapool item, 157
 - widget property, 129, 131
- list, 154
 - create, 196
 - data type, 220
- list index
 - reference, 297
- list layout
 - reference, 295
- long hold gesture
 - reference, 282

M

- material
 - reference, 271
- mesh, 55
 - data type, 217
 - reference, 272
- mesh list
 - data type, 217

- model element, 49, 313
 - delete, 110
- model.json, 246
- move in
 - reference, 288
- move out
 - reference, 288
- move over
 - reference, 289
- moveable
 - reference, 289
- multi-touch input, 94
- multiple lines
 - reference, 274
- multisampling, 259

N

- navigation area
 - project center, 38
- navigation component
 - project editor, 39
- normal map texture
 - reference, 302

O

- opaque texture
 - reference, 303
- OS, 314

P

- paste
 - datapool item, 154
 - event, 151
- path gesture, 193
 - reference, 283, 283
- pinch gesture
 - reference, 284
- pivot
 - reference, 306
- platform.json, 255
- point light
 - reference, 272

- pressed
 - references, 275
- problems component, 161
 - project editor, 45
- profile, 171, 314
 - clone, 171
- project center, 37, 314
 - content area, 38
 - navigation area, 38
- project editor, 38, 314
 - command area, 44
 - content area, 42
 - navigation component, 39
 - problems component, 45
 - toolbox, 41
 - toolbox component, 41
- properties component
 - command area, 42
 - project editor, 42

Q

- quadratic curve
 - reference, 267

R

- reader application, 37
- rectangle
 - reference, 264
- reflection texture
 - reference, 304
- rename global, 160
- renderer
 - configure, 174
- resource, 314
 - .psd file format, 55
 - 3D graphic, 55
 - font, 53
 - image, 54
 - mesh, 55
- resource management, 53
- rotary
 - reference, 289

- rotate gesture
 - reference, 285
- rotation
 - reference, 306

S

- scale mode
 - reference, 296
- scaling
 - reference, 306
- scene configuration
 - reference, 258
- scene graph, 33, 55, 125, 210
 - add, 125
 - reference, 270
 - texture, 210
- scene graph node
 - reference, 270
- script curve, 268
- scripted value, 69, 155
- selected
 - reference, 275
- selection group
 - reference, 276
- shared library, 314
- shearing
 - reference, 307
- shortcut
 - user interface, 71
- simulation, 163
- sinus curve
 - reference, 267
- skin
 - add, 141
 - delete, 141
 - support, 53
 - switch, 141
- slow start curve
 - reference, 266
- specular texture
 - reference, 304
- spinning

- reference, 276
- spot light
 - reference, 273
- state, 75, 105, 106, 182, 314
 - choice state, 79
 - compound state, 75
 - entry action, 108
 - exit action, 109
 - final state, 78
 - history state, 80
 - initial state, 77
 - transition, 110
 - view state, 77
- state machine, 74, 314
 - add, 103
 - comparison to UML, 91
 - delete, 105
 - dynamic state machine, 74
 - execution of state machine, 87
 - haptic state machine, 74
 - include state machine, 74, 92
 - logic state machine, 74
 - state, 75
 - transition, 83
 - UML 2.5 notation, 91
- string
 - data type, 221

T

- template
 - create, 147
 - delete, 149
 - use, 149
- template index
 - reference, 297
- template interface, 148
 - add property, 148
 - remove property, 148
- text truncation
 - reference, 277
- todo
 - EB GUIDE Script, 57

- toolbox
 - project editor, 41
- toolbox component
 - project editor, 41
- touch gesture (see gesture)
- touch input (see gesture)
- touch lost
 - reference, 290
- touch move
 - reference, 290
- touch pressed
 - reference, 291
- touch released
 - reference, 291
- touch status changed
 - reference, 292
- touched
 - reference, 277
- transition, 83, 110, 314
 - action, 114
 - add, 110
 - condition, 113
 - internal, 116
 - move, 111
 - trigger, 112
- translation
 - reference, 307
- trigger
 - transition, 112

U

- UI, 314
- undocking
 - component, 45
- user-defined focus
 - reference, 280
- user-defined property, 133

V

- view, 95, 315
 - add, 117
 - reference, 260

view template
 reference, 260, 260

viewport
 reference, 298

W

widget, 95, 315

- 3D widget, 96
- add, 118
- animation, 96
- basic, 96
- delete, 127
- group, 122
- position, 127
- resize, 128

widget feature, 97, 98, 99

- add, 136
- path gesture, 193
- remove, 138

widget property, 97

- add, 133
- default property, 98
- EB GUIDE Script, 65
- link to datapool item, 131
- link to widget property, 129
- user-defined property, 98, 133
- widget feature property, 98
- widget template, 98

widget template, 98, 147, 150

widget template interface, 98

windowed list

- datapool item, 49

writer application, 37