Elektrobit

EB GUIDE documentation

Version 6.0.0.97672

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

Table of Contents

1. About this documentation

1.1. Target audiences of the user documentation

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

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

The following roles exist:

- section 1.1.1, “Modelers”
- section 1.1.2, “System integrators”
- section 1.1.3, “Application developers”
- section 1.1.4, “Extension developers”

1.1.1. Modelers

Modelers use EB GUIDE Studio to create an interface between man and machine. The interface is a model called an HMI (human machine interface). 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 following tasks:

- Use widgets and view architecture to specify the graphical elements that are shown 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 the elements' behavior reacts to input from devices such as control panels or touch screens
- Define how elements receive required information from other 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 the 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

1.1.2. System integrators

System integrators make sure that all the different system parts are integrated into one complete and working system.

System integrators perform the following tasks:

- Ensure that the different project parts are executed together
- Configure required modules and file system folder structures
- Integrate customer specific framework extensions and HMI applications
- Carry out settings to ensure system integrity within EB GUIDE Studio and on the target system
- Carry responsibility for the project setup in EB GUIDE Studio, for example, create a shared workspace in projects involving different people working together on one EB GUIDE model

System integrators have the profound knowledge of the following:

- The system, including the target platform used and the restrictions of this target platform
- The generating mechanism that ensures compatibility of an EB GUIDE model and the target system

1.1.3. Application developers

Application developers write source code for HMI applications, such as a CD player or a radio. Such applications add distinct functionality to the system, for example control of hardware components.

Application developers perform the following tasks:

- Program additional functionality that is required by the system
- Write code to interface with the EB GUIDE TF, provide application data to the HMI, and provide communication with the HMI
- Consider the required communication data between the HMI model and its application
- Define datapool items and events
- Determine the flow of data between HMI model and application
- Communicate with modelers to know what data can be provided by hardware devices and how to use the different EB GUIDE GTF communication mechanisms

Application developers have the profound knowledge of the following:

- C++, to know how to compile for the existing EB GUIDE TF C++ interfaces
- All programming languages used, as applications can be written in any programming language
- The specifications and requirements of the domain

1.1.4. Extension developers

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

Extension developers perform the following tasks:

- Communicate with members of the EB GUIDE development team through chapter 3, “Support“ to find out if there are already solutions to problems
- Work on the framework and develop new plugins or features
- Write code for additional modules or replace the following items:
    - Existing EB GUIDE TF modules such as widgets or the renderer
    - Existing EB GUIDE Studio plugins such as problem checkers

Extension developers have the profound knowledge of the following:

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

1.2. Structure of user documentation

The information is structured as follows:

- Background information

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

- How-to-instruction

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

- Tutorial

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

- Reference

  References provide detailed technological parameters and tables as well as the EB GUIDE Monitor API documentation.

- Demonstration

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

1.3. Typography and style conventions

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

Convention	Item is used	Example
Arial italics	to emphasize	If your project's release version is mixed, all content types are available. It is thus called *mixed version*.
Arial boldface	for menus and submenus	Select the **Options** menu.
Arial boldface	for buttons	Select **OK**.
Arial boldface	for keyboard keys	Press the **Enter** key.
Arial boldface	for keyboard combination of keys	Press **Ctrl**+**Alt**+**Delete**.
Arial boldface	for commands	Convert the XDM file to the newer version by using the **legacy convert** command.
Monospaced font (Courier)	for file and folder names, also for chapter names	Put your script in the `function_name\abcfolder`.

Monospaced font (Courier)	for code	`CC_FILES_TO_BUILD =(PROJECT_PATH)\source\network\can_node.c`
		`CC_FILES_TO_BUILD += $(PROJECT_PATH)\source\network\can_config.c`

Monospaced font (Courier)	for function names, methods, or routines	The cos function finds the cosine of each array element. Syntax line example is `MLGetVar ML_var_name`.
Monospaced font (Courier)	for user input/indicates variable text	Enter a `three-digit prefix` in the menu line.

Convention	Item is used	Example
Square brackets [ ]	denote optional parameters; for command syntax with optional parameters	`insertBefore [<opt>]`
Curly brackets {}	denote mandatory parameters; for command syntax with mandatory parameters (in curly brackets)	`insertBefore {<file>}`
Three dots …	indicate further parameters; for command syntax	`insertBefore [<opt>…]`
Warning	to warn about danger of death or severe personal injury	**WARNING** ⚠ **This is an example for a warning** This is what a warning looks like.
Caution	to warn about danger of slight personal injury or material damage	**CAUTION** ⚠ **This is an example for a caution** This is what a caution looks like.
Notice	to give additional but not vital information on a subject	**NOTE** ⓘ **This is an example for a notice** This is what a notice looks like.
Tip	to provide helpful hints and tips	**TIP** **This is an example for a tip** This is what a tip looks like.
Example	to demonstrate or illustrate information	**Example 1.1.** **This is an example** This is what an example looks like.

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 6.0`

- 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 6.0\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 6.0\projects\`

2. Safe and correct use

2.1. Intended use

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

2.2. Possible misuse

WARNING ⚠

Possible misuse and liability

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

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

  Usage of EB GUIDE GTF in such environments is not allowed. If you are unsure about your specific application, contact EB for clarification at chapter 3, “Support“.

3. Support

EB GUIDE support is available in the following ways.

- For community edition:

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

- For enterprise edition:

  Contact us according to your support contract.

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

4. Introduction to EB GUIDE

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

4.1. The EB GUIDE product line

The EB GUIDE product line comprises the following software parts:

- EB GUIDE Studio
- EB GUIDE TF

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

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

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

4.2. EB GUIDE Studio

4.2.1. Modeling HMI behavior

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

- State machines

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

- States

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

4.2.2. Modeling HMI appearance

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

4.2.3. Handling data

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

- Events are temporary triggers. Events can be sent to both parties to signal that something specific happens.
- The datapool is an embedded database that holds all data that needs to be displayed and all other internal information. Datapool items store and exchange data.

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

4.2.4. Exporting the EB GUIDE model

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

4.3. EB GUIDE TF

The EB GUIDE TF is a set of libraries, executables, and software tools, which are required to execute an EB GUIDE model.

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

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

EB GUIDE TF uses the following platform abstractions:

- OS abstraction

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

- GL abstraction

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

5. Modeler’s manual

5.1. Overview

As a modeler you are the target audience for the following chapters. For more information, see section 1.1.1, “Modelers”.

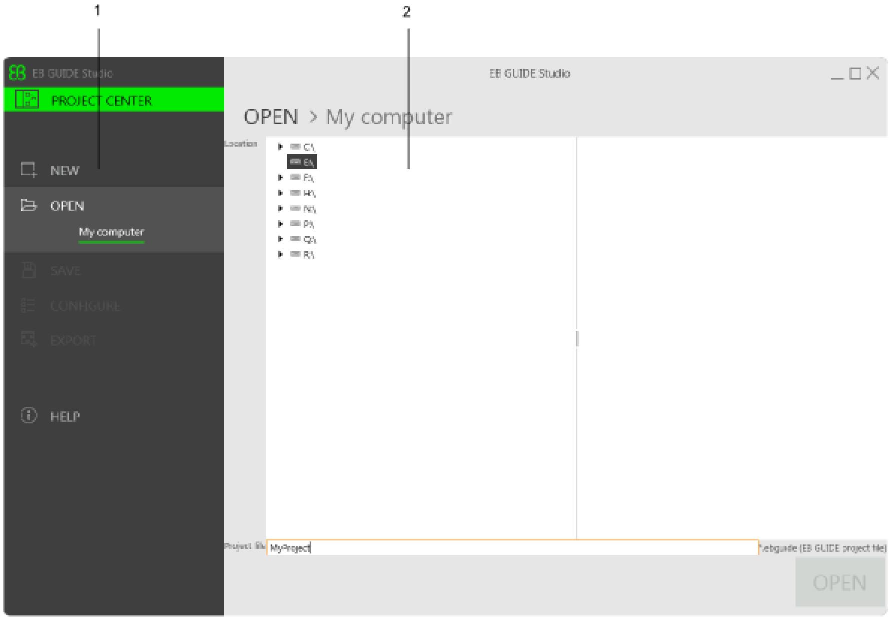
For more information on the structure of the manual, see section 1.2, “Structure of user documentation”.

5.2. 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 project options, and export projects for copying to the target device. In the project editor, you model HMI appearance and behavior.

5.2.1. 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.



Project file: MyProject *.ebguide (EB GUIDE project file)

Figure 5.1. Project center with navigation area (1) and content area (2)

5.2.1.1. Navigation area

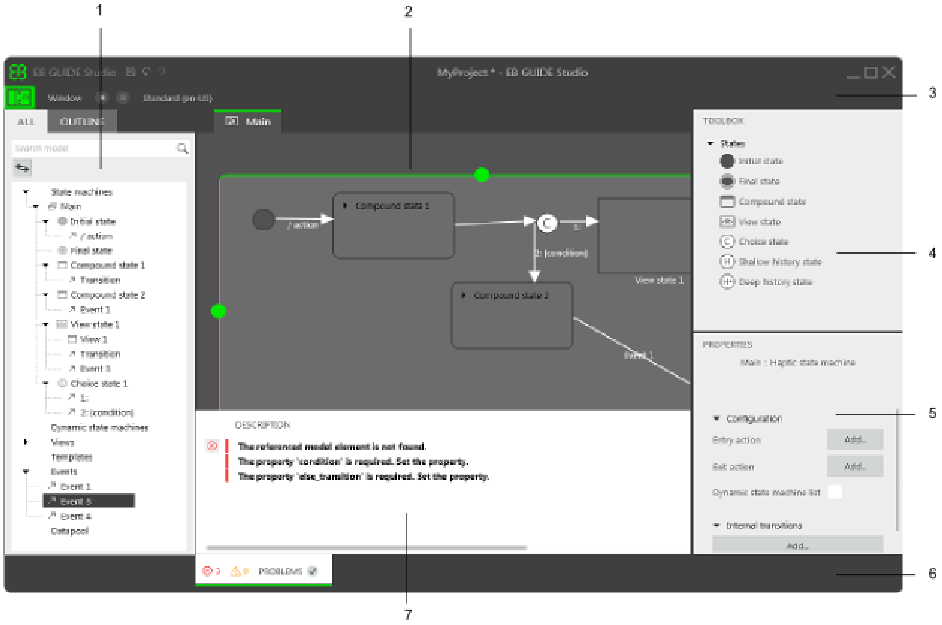
The navigation area of the project center consists of function tabs such as **CONFIGURE** or **EXPORT**. You select a tab in the navigation area and the content area displays the corresponding functions and settings.

5.2.1.2. 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.

5.2.2. 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.



The referenced model element is not found.
The property 'condition' is required. Set the property.
The property 'else_transition' is required. Set the property.

Figure 5.2. Project editor with its areas

1 Navigation area

2 Content area

3 Command area

4 Toolbox

5 Properties panel

6 Status bar

7 Problems area

5.2.2.1. Navigation area

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

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

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

  The **All** tab is also where you add events and datapool items.

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

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

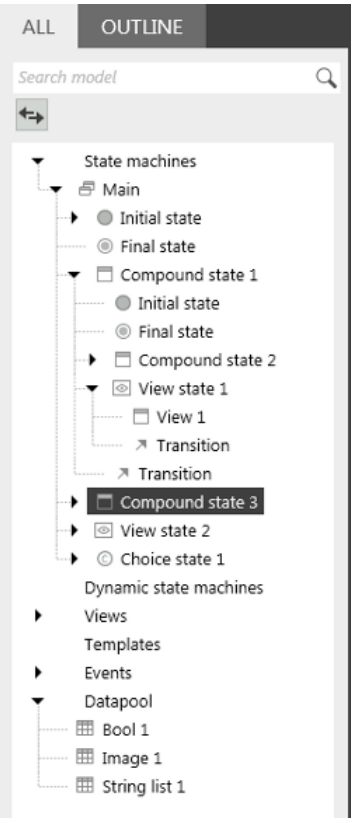


Figure 5.3. Navigation area in project editor

5.2.2.2. Content area

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

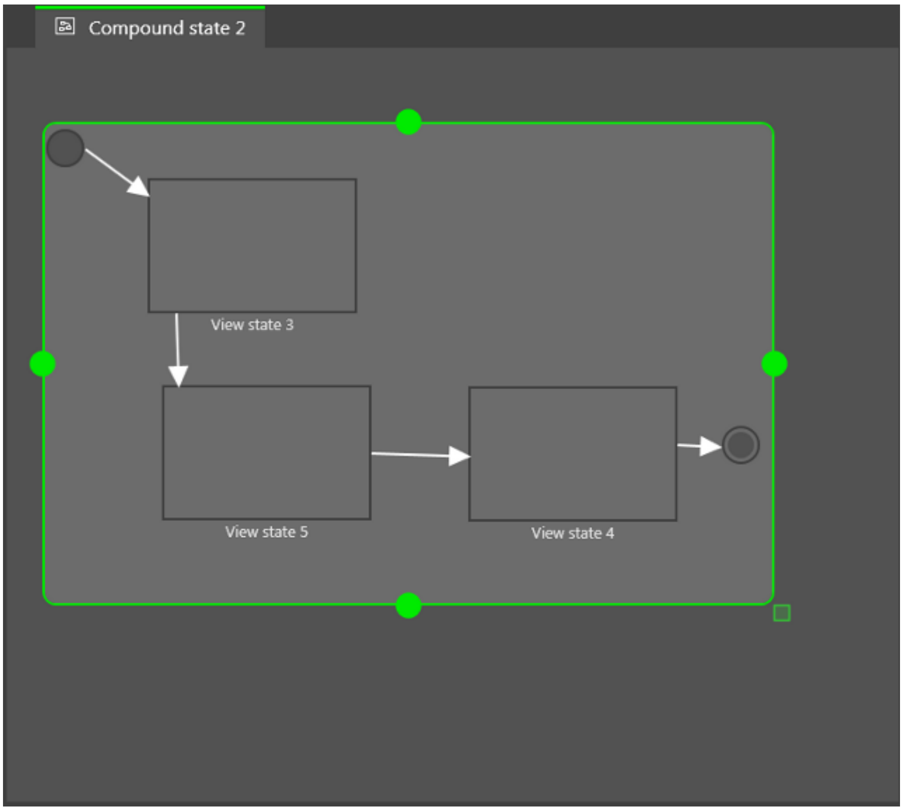


Figure 5.4. Content area in project editor

5.2.2.3. Command area

In the command area you find the ▣ button to open the project center and further menus.

5.2.2.4. Toolbox

All tools you need for modeling are available in the toolbox. Depending on the element that is displayed in the content area, the toolbox offers a different set of tools. For example, the toolbox can contain the following:

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

You drag model elements from the toolbox to the content area.

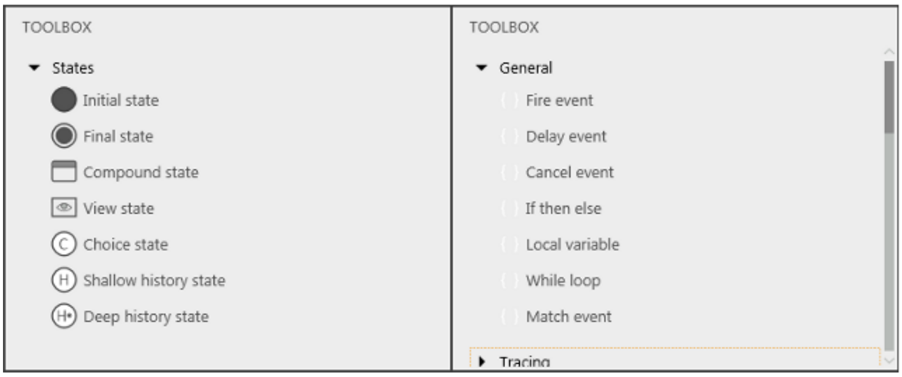
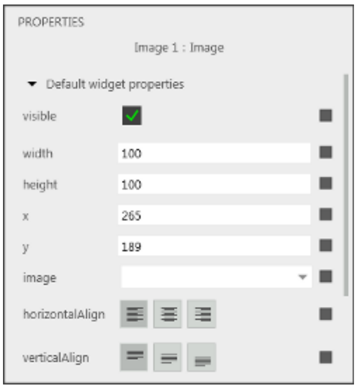


Figure 5.5. Toolbox in project editor

5.2.2.5. Properties panel

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



visible ☑
width 100
height 100
x 265
y 189
image
horizontalAlign
verticalAlign

Figure 5.6. Properties panel displaying properties of a widget

5.2.2.6. Status bar

The status bar displays status information about EB GUIDE Studio.

5.2.2.7. Problems area

The problems area displays errors and warning for the EB GUIDE model.

5.3. Tutorial: Getting started

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

5.3.1. 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.

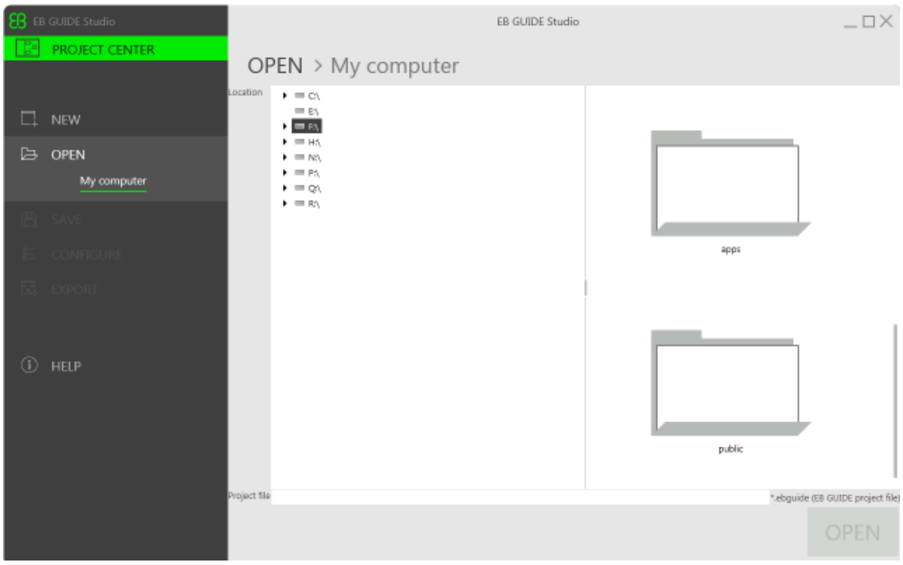


Figure 5.7. Project center

5.3.2. Creating a project

Creating a project

Prerequisite:

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

Step 1
In the navigation area, click the **NEW** tab.

Step 2
Select the `C:\temp` directory.

Step 3
Enter the project name `MyProject`.

Step 4
Click the **CREATE** button.

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.

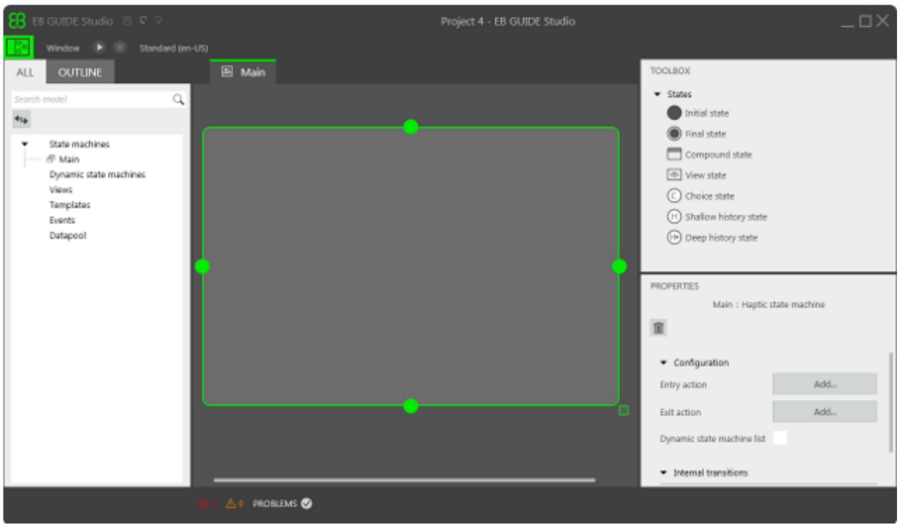


Figure 5.8. Project editor with **Main** state machine

5.3.3. 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 project editor is displayed.
- In the content area the **Main** state machine is displayed.

Step 1
Drag a view state from the toolbox to the content area.

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 to the content area.

Step 4
Drag a final state from the toolbox to the content area.

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

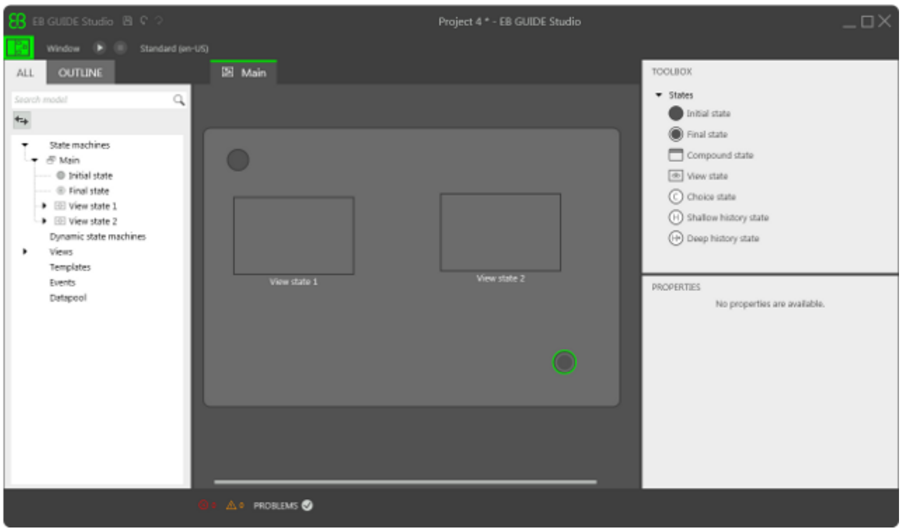


Figure 5.9. Project editor with states

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:

- In the content area, an initial state, two view states, and a final state are displayed.

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 to 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
In the **PROPERTIES** panel expand the **Trigger** combo box.

Step 8
Enter `Event 1` in the **Trigger** combo box and click **Add event**.

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

Step 9
Add a transition between **View state 2** and the final state.

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

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

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

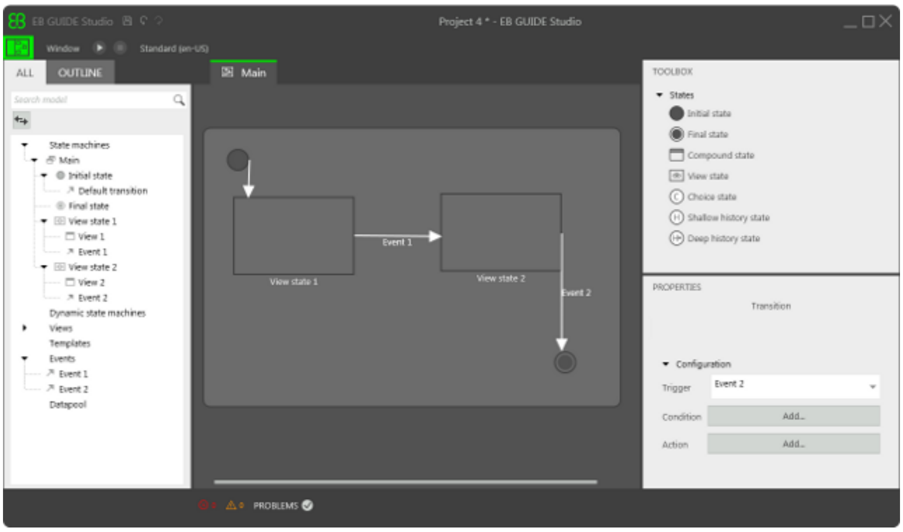


Figure 5.10. States linked by transitions with events

You have defined the behavior of a basic state machine.

5.3.4. 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:

- In navigation area the **All** tab is opened.
- **View state 1** is added.

Step 1
In the content area double-click **View state 1**.

In the content area **View 1** is displayed.

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 widget to a view. The rectangle widget reacts on user input and thus functions as a button.

Prerequisite:

- In the content area **View 1** is displayed.

Step 1
Drag a rectangle widget from the toolbox to the content area.

Step 2
In the **PROPERTIES** panel go to the **Widget features** category and click the **Add/Remove** button.

The **Widget features** dialog is displayed.

Step 3
Expand the **Common** category and select **State Touched**.

The related properties are added to the **PROPERTIES** panel.

Step 4
Expand the **Input handling** category and select **Touch Released**.

The related properties are added to the **PROPERTIES** panel.

Step 5
To close the **Widget features** dialog, click outside the dialog.

Step 6
In the **PROPERTIES** panel set the `touchPolicy` property to `Press then react`.

The rectangle widget reacts on touch input.

Step 7
Go to the `touchShortReleased` property and click the **Edit** button.

Step 8
Enter the EB GUIDE Script expression `fire_delayed 500, ev:"Event 1"()`.

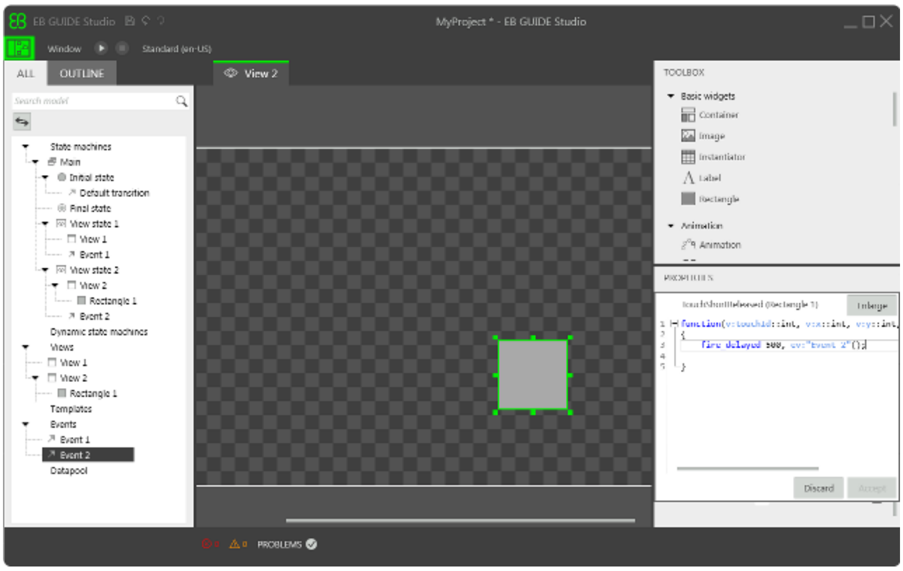
If the rectangle is touched, `Event 1` is fired after 500 milliseconds.

Step 9
Click **Accept**.

Step 10
In the **PROPERTIES** panel set the **fillColor** property to red.

Step 11
Open **View 2** and repeat steps 1 - 7.

Step 12
Enter the EB GUIDE Script expression `fire_delayed 500, ev:"Event 2"()`.



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

}
```

Figure 5.11. Widget property with an EB GUIDE Script expression

Step 13
Click **Accept**.

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

Step 14
In the **PROPERTIES** panel set the **fillColor** property to blue.

5.3.5. 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

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 ■ .

5.4. Background information

The topics in this chapter are sorted alphabetically.

5.4.1. Animations

With widget animations, you can animate widgets within a view. Each animation has one or more animation curves associated to it.

An animation curve has a target widget property and describes the time-based change of the target property. For example, there are constant curves, linear interpolation curves, or sinus curves.

Among others, widget animations 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 script functions `f:animation_play`, `f:animation_pause`, `f:animation_cancel`, etc.

TIP

Concurrent animations

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

For animation and curve properties, see section 9.7.3, "Animations".

5.4.2. Application programming interface between application and model

EB GUIDE abstracts all communication data between an application and the EB GUIDE TF in an application programming interface (API). An application may be 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 5.1.
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

5.4.3. Communication context

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

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

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

5.4.4. Datapool

5.4.4.1. 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.

5.4.4.2. Datapool items

Datapool items are used to the follwing:

- 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.

To channel communication, you use the communication context.

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

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

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

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

5.4.4.3. Windowed lists

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

NOTE

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

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

Example 5.2.
Windowed list

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

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

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

5.4.5. 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 platform, you export the EB GUIDE model and copy the resulting binary files to the target platform.

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

5.4.6. Event handling

5.4.6.1. Event system

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

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

5.4.6.2. Events

Group ID

The group IDs 0...65535 are reserved for internal use within the EB GUIDE product line. The remaining range of group IDs is available for customer-specific applications.

Event ID

If you set the property, the given numeric value defines the event ID used by EB GUIDE TF to send and receive the event.

5.4.7. Languages

Most human machine interfaces offer the possibility to display written texts in the user's preferred language. Such language management is also provided by EB GUIDE. You add and select languages for an EB GUIDE model in the project configuration.

Example 5.3.

Language support

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.

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

A numerical identifier is assigned to every language. To change a language during run-time, you require the identifier. Set a unique numerical identifier to know which identifier belongs to which language.

For more details see: section 5.6.2.1, "Adding a language".

NOTE

Support of languages
It is possible to make datapool items language dependent. A datapool item defines a value for each language. To support languages the **read-only** property has to be selected.

5.4.8. Resource management

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

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

There are three types of resource in EB GUIDE:

1. Fonts
2. Images
3. 3D graphics

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

5.4.8.1. 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`).

5.4.8.2. Images

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

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

5.4.8.2.1. SVG images

EB GUIDE supports the following SVG element types:

Basic shapes

- Rectangle
- Circle
- Ellipse
- Line
- Polyline
- Polygon
- Path: moveto, lineto, curve, closepath, cubic bézier curve, quadratic bézier curve

Painting modes

- Fill
- Stroke: solid and with dashes

Paint types

- Color
- Linear and radial gradients, including spread modes

Only the OpenVG renderer can process SVG images.

NOTE

Mandatory attributes for SVG files

For correct clipping and scaling, EB GUIDE Studio depends on the SVG attributes `width` and `height`. The size of SVG files has to match the view area.

5.4.8.2.2. 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.

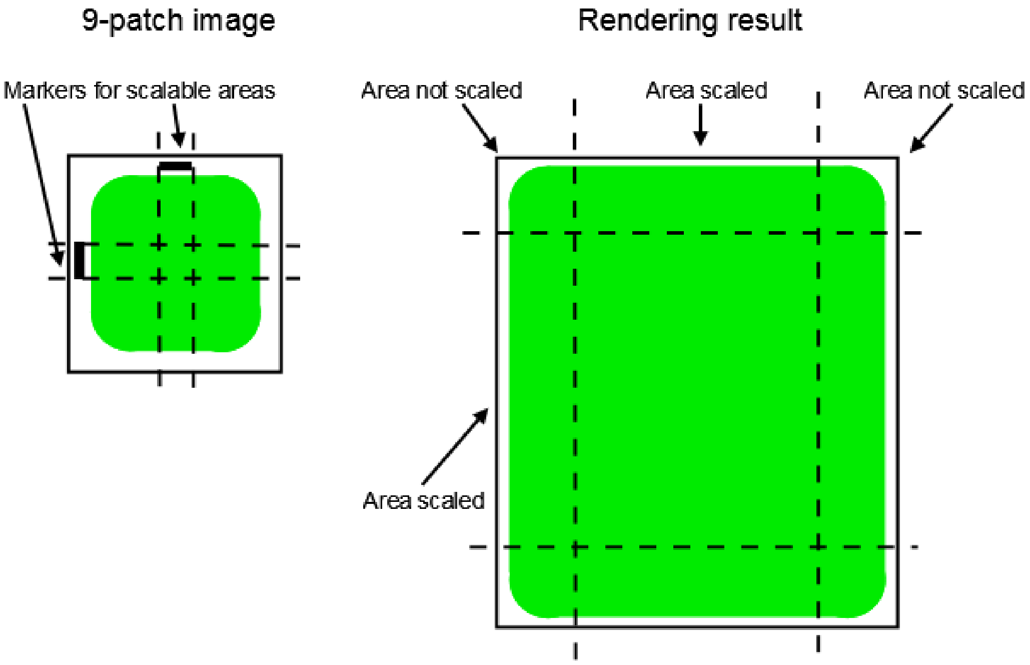


Figure 5.12. 9-patch example

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

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

5.4.8.3. 3D graphics

It is possible to display 3D graphics in EB GUIDE Studio. .

In order to use a 3D graphic in the project, add the 3D graphic to the directory `$GUIDE_PROJECT_PATH/<project name>/resources`.

3D graphics can have textures. Textures are images that are mapped to the 3D graphic. Copy any 3D graphic texture manually into a directory with a name equal to the 3D graphic file but without the file extension.

Example 5.4.

Naming of textures for 3D graphics

The 3D graphic is called `car.dae`. Place any related texture images in a directory called `car` that resides in the same directory as the file `car.dae`.

Only the OpenGL ES 2.0 and DirectX 11 renderers can display 3D graphics. For supported 3D graphic formats see section 9.7.4.1.1, “Supported 3D graphic formats”.

5.4.9. 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.

5.4.9.1. 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

5.4.9.2. 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 5.5.
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!";
```

5.4.9.3. 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 5.6.
Comments in EB GUIDE Script

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

5.4.9.4. 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 (int)
- Unicode strings (string)
- Objects with reference counting
- Type definitions to the above listed types and to the following:
  - Color (int for 32-bit RGBA value)
  - Boolean (bool)
  - IDs of different model elements: datapool items, views, state machines, pop-ups (all of int 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.

5.4.9.5. 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 5.7.
Evaluation of an integer value

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

5.4.9.6. 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 5.8.

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
```

5.4.9.7. 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 `int`.
- The logical operators or `(||)`, and `(&&)`, not `(!)` can be applied to expressions of type `bool`.
- Integers and strings can be compared with the comparison operators greater-than `(>)`, less-than `(<)`, greater-than-or-equal `(>=)`, less-than-or-equal `(<=)`.
- Every type can be compared with the equality operators `(==)` and `(!=)`.
- Strings can be concatenated with the `(+)` operator.

Example 5.9.
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?)
!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
```

5.4.9.8. 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.

5.4.9.9. 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 5.10.
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;
in
    let v:z = v:x + v:a;
    in
        dp:x = v:z;
```

5.4.9.10. 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 `bool`, 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 5.11.
Usage of the `while` loop

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

5.4.9.11. 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 `bool`.
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 can not be used to return values from scripts.

Example 5.12.
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 {}
```

5.4.9.12. 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 5.13.
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 children 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);
```

5.4.9.13. 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 5.14.
Assignment of datapool values

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

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

The following operators can be applied to the datapool items:

- The reference operator (&) can be applied to datapool items. It refers to the address of a datapool item rather than to its value. The reference operator is used in foreign function calls to pass parameters of type `dp_id`.
- The redirect-reference operator (=>) assigns a different datapool item to a datapool reference. It may only be applied to datapool references.

5.4.9.14. Widget properties

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

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

Example 5.15.
Setting widget properties

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

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

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

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

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

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

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

Example 5.16.
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;
}
```

5.4.9.15. 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 `int`.

Example 5.17.
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.

5.4.9.16. 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 5.18.
  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 5.19.
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 5.20.
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 5.21.
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;
}
```

5.4.9.17. 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 following conversion functions:

- `int2string` for simple integer-to-string conversion
- `formatInteger` for advanced integer-to-string conversion including features like fill characters and formatting in binary, hexadecimal, and decimal format

Example 5.22.
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);
```

5.4.9.18. 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 more information on the standard library, see section 9.3.3, “EB GUIDE Script standard library”.

5.4.10. Shortcuts and icons

5.4.10.1. Shortcuts

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

Table 5.1. Shortcuts

Hotkey	**Description**
Ctrl+**Y**	Redo
Ctrl+**Z**	Undo
Del	Deletes the selected widget from a view in the content area
F1	Opens the user documentation
F2	Renames the selected model element in the navigation area
Up/Down/Left/Right	Moves the selected state or widget in the content area one pixel up, down, left, or right

5.4.10.2. Icons

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

Table 5.2. Icons in the command area

Icon	**Description**
↶	Undo
↷	Redo
💾	Save
▶	Starts the simulation
■	Stops the simulation

Table 5.3. Icons in the navigation area

Icon	**Description**
👁	Indicates a template
↗	Indicates a transition
⇆	Synchronizes content area and navigation area
⊞	Adds an event, a datapool item or a state machine

Table 5.4. Icons in the **PROPERTIES** panel

Icon	Description
⬛	Indicates a local property
🟨	Indicates that a property is linked to another property
🟧	Indicates that a property is linked to a datapool item
🔵	Widget template: Indicates that a property is added to the widget template interface
🟦	Widget template instance: Indicates that a property value is equal to the value in the template

5.4.11. State machines and states

5.4.11.1. 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:

5.4.11.1.1. Haptic state machine

Haptic state machine allows the specification of GUI.

5.4.11.1.2. Logic state machine

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

5.4.11.1.3. 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

5.4.11.2. 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 the destination of a state change.

A state has the following properties:

- Entry action
- Exit action
- Internal transitions

5.4.11.2.1. 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.

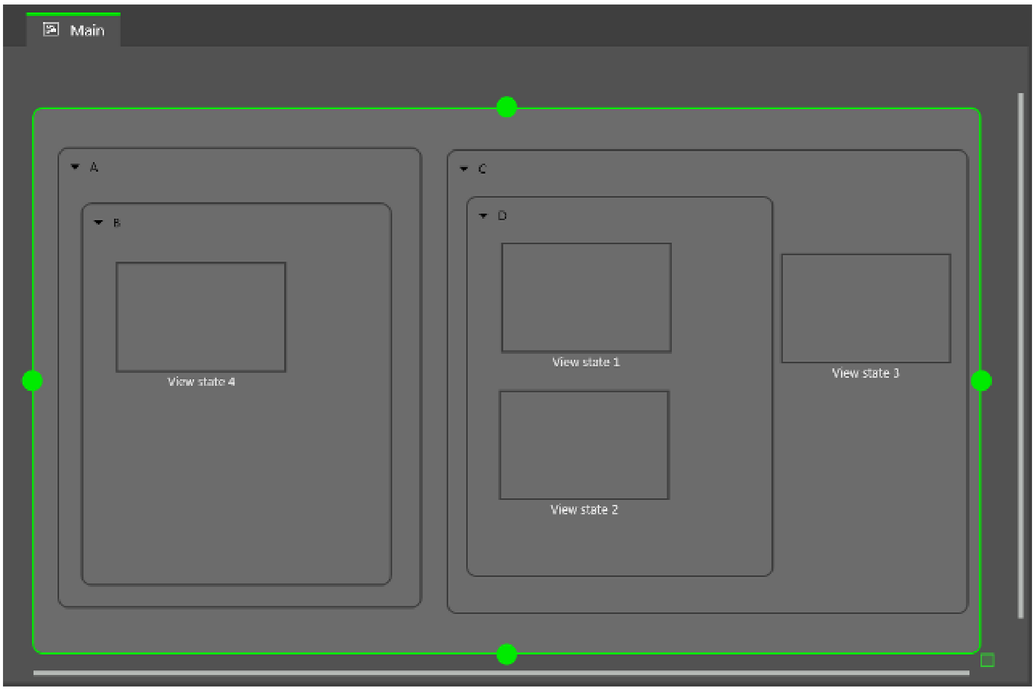


Figure 5.13. Compound states

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

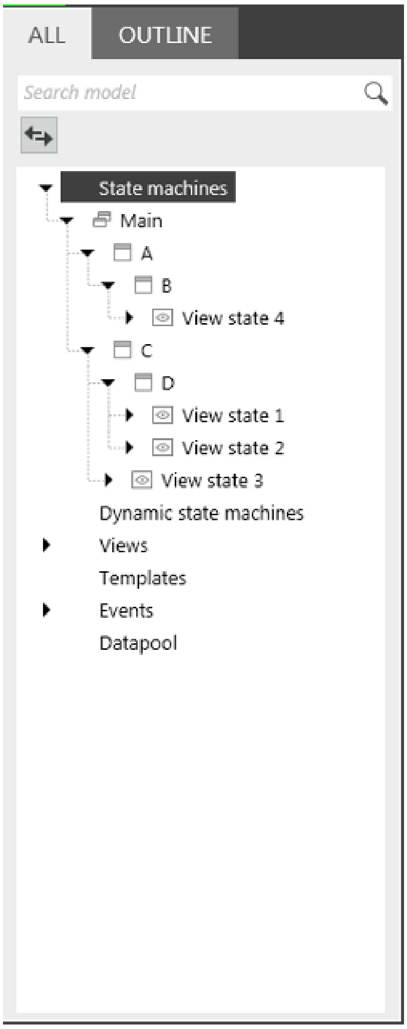


Figure 5.14. State hierarchy as a tree

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.

5.4.11.2.2. 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.

5.4.11.2.3. 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 compound state children

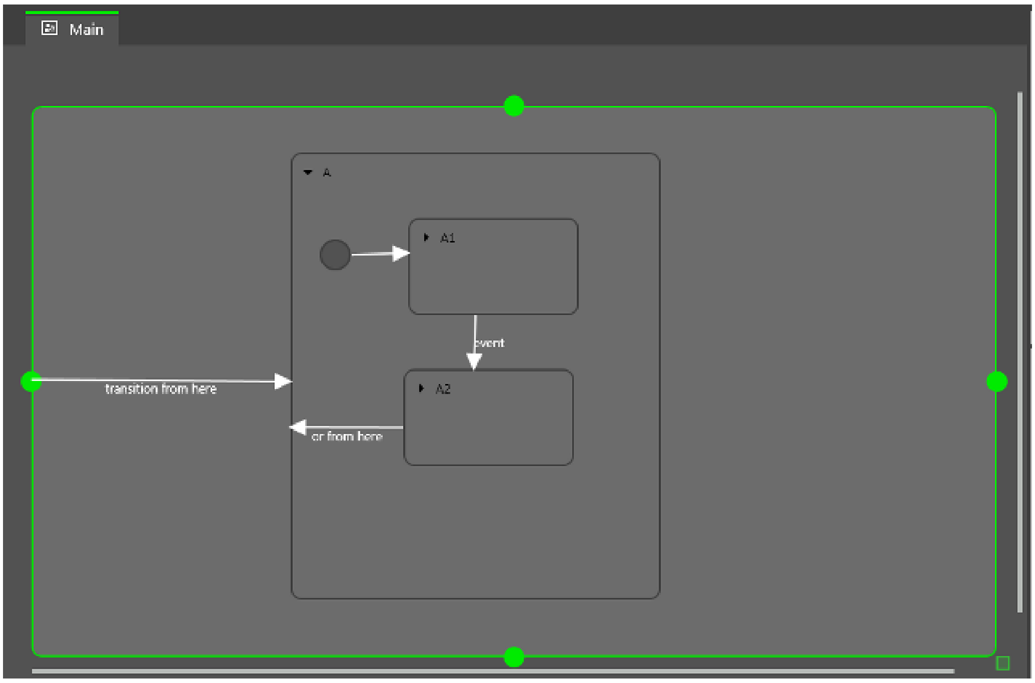


Figure 5.15. An example of an initial state

5.4.11.2.4. 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.

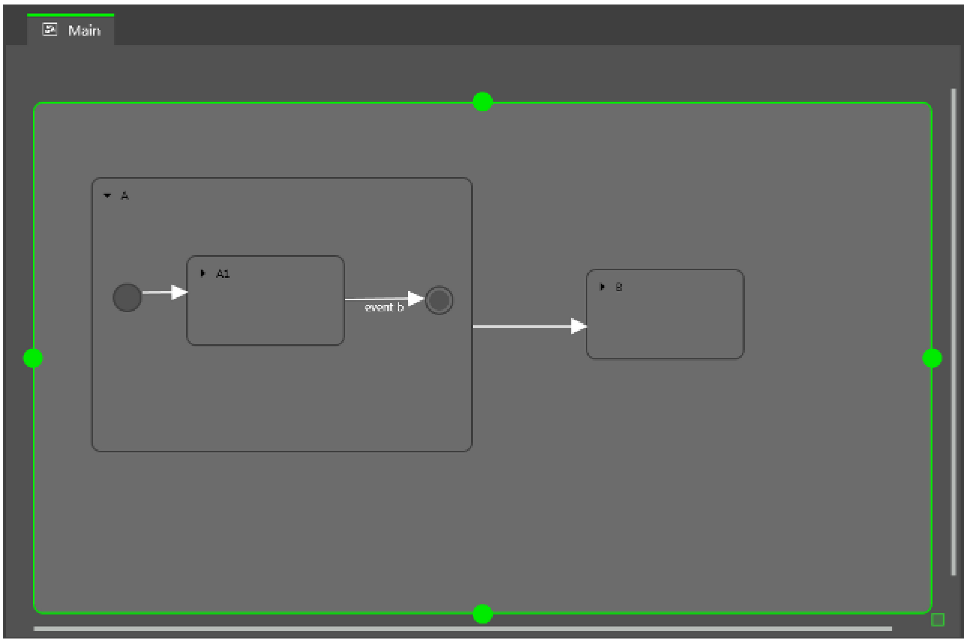


Figure 5.16. Final state usage in a compound state

5.4.11.2.5. 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.

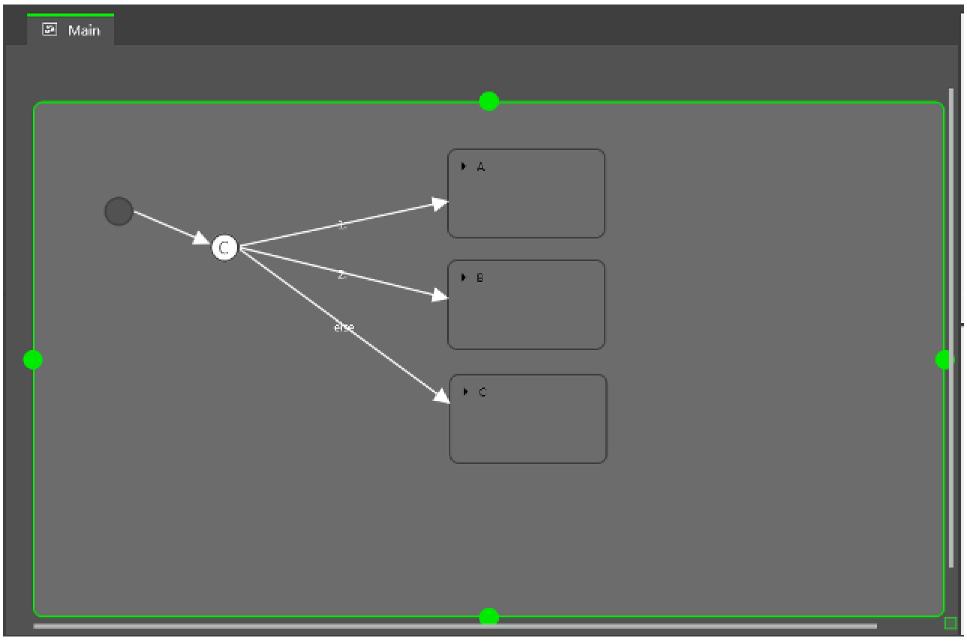


Figure 5.17. Choice state with incoming and outgoing transitions

5.4.11.2.6. 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 5.23.
Shallow history state

A shallow history state can be used as follows.

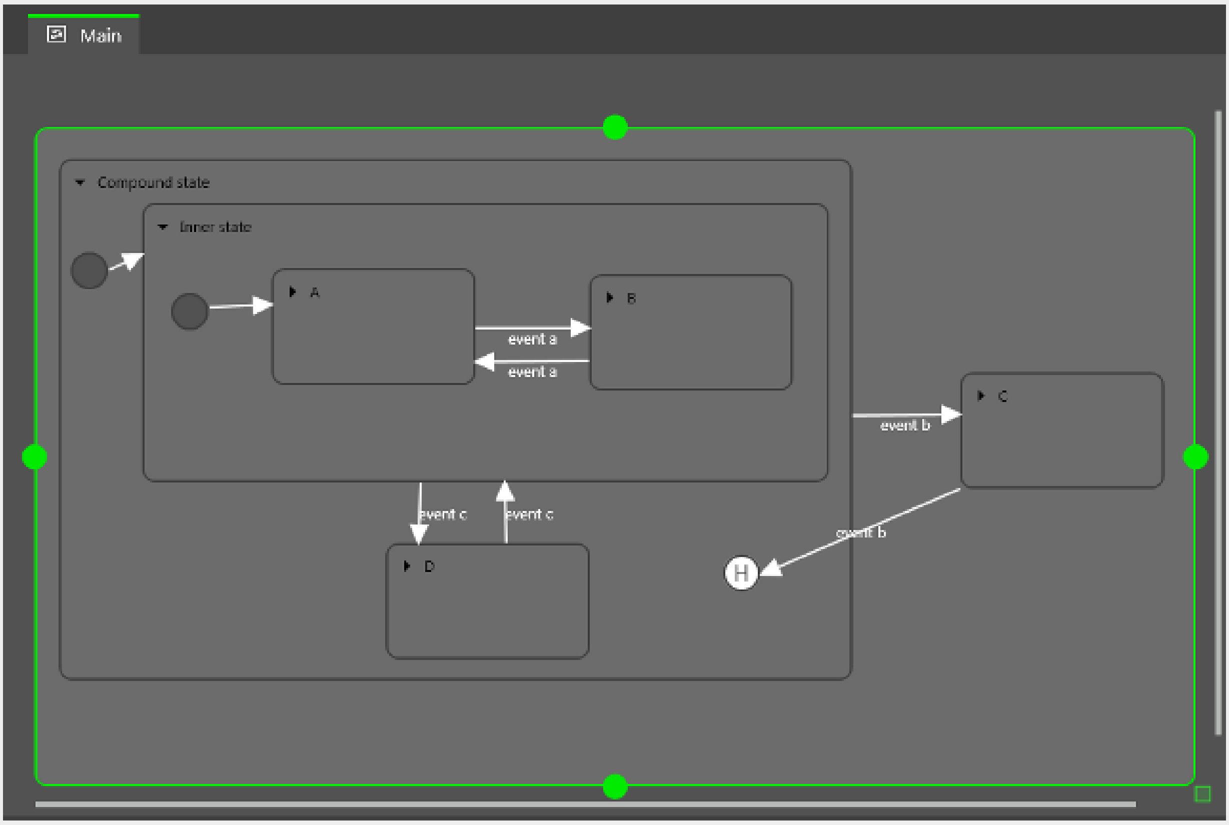


Figure 5.18. Shallow history state

- 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 5.24.

Deep history state

A deep history state can be used as follows.

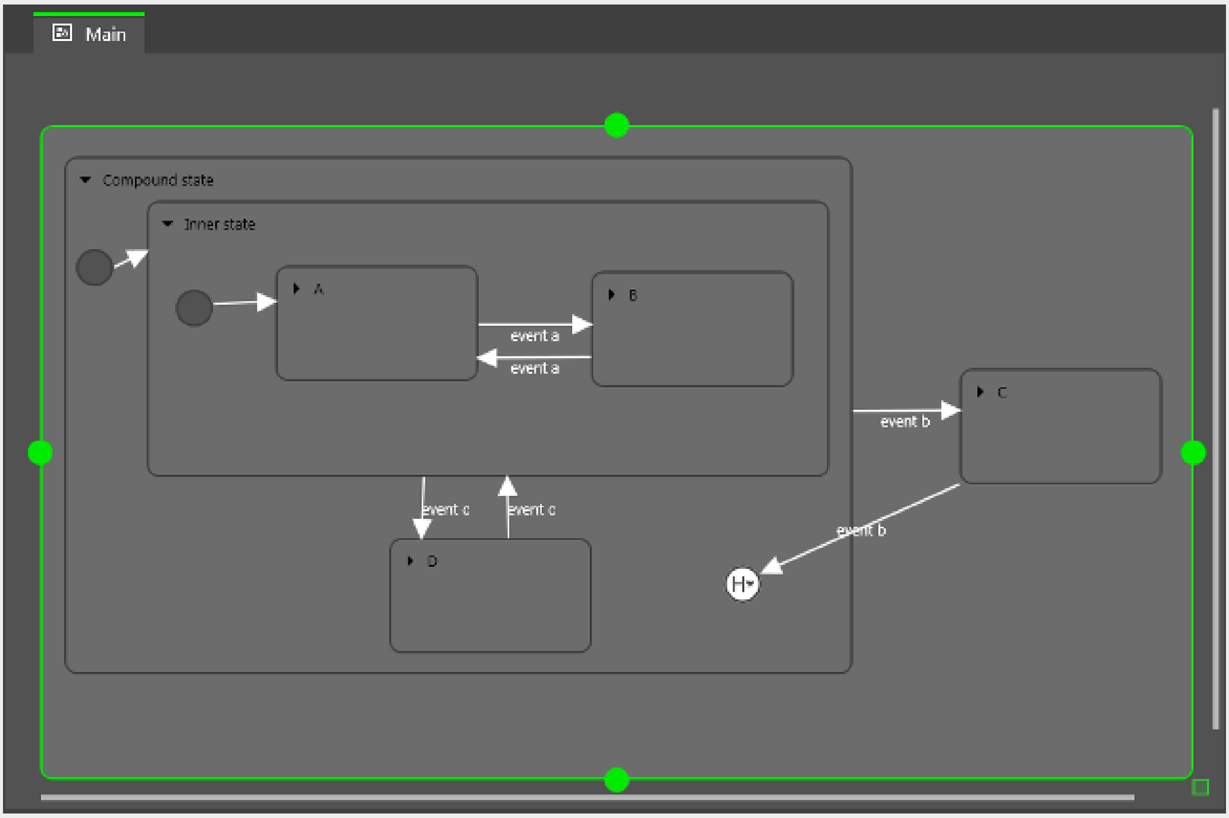


Figure 5.19. Deep history state

- 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.

5.4.11.3. 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 executed along with the transition

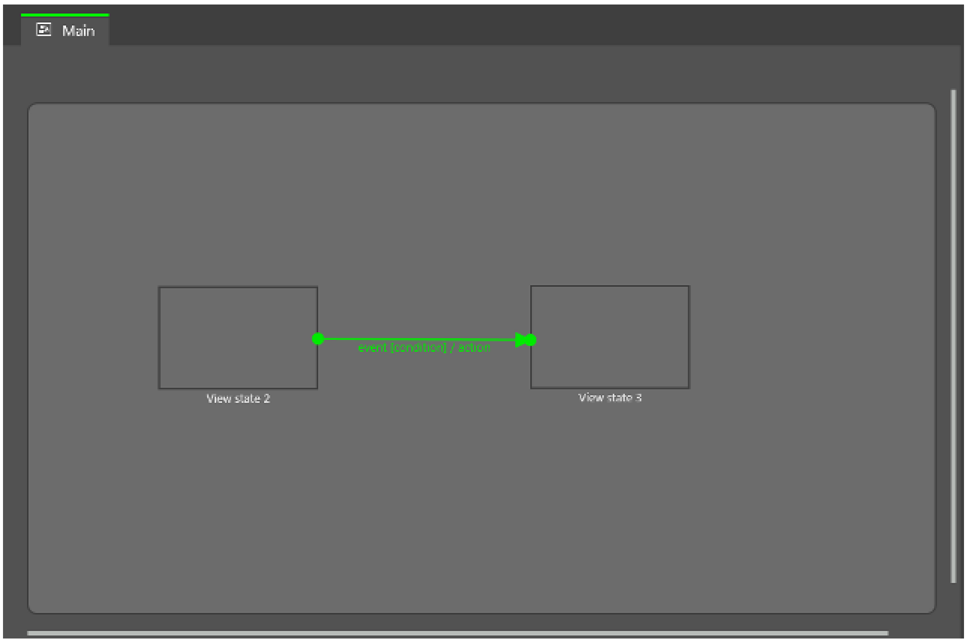


Figure 5.20. A transition

NOTE

Transitions are deterministic

It is not possible to have more than one transition 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 5.25.

Transition inheritance

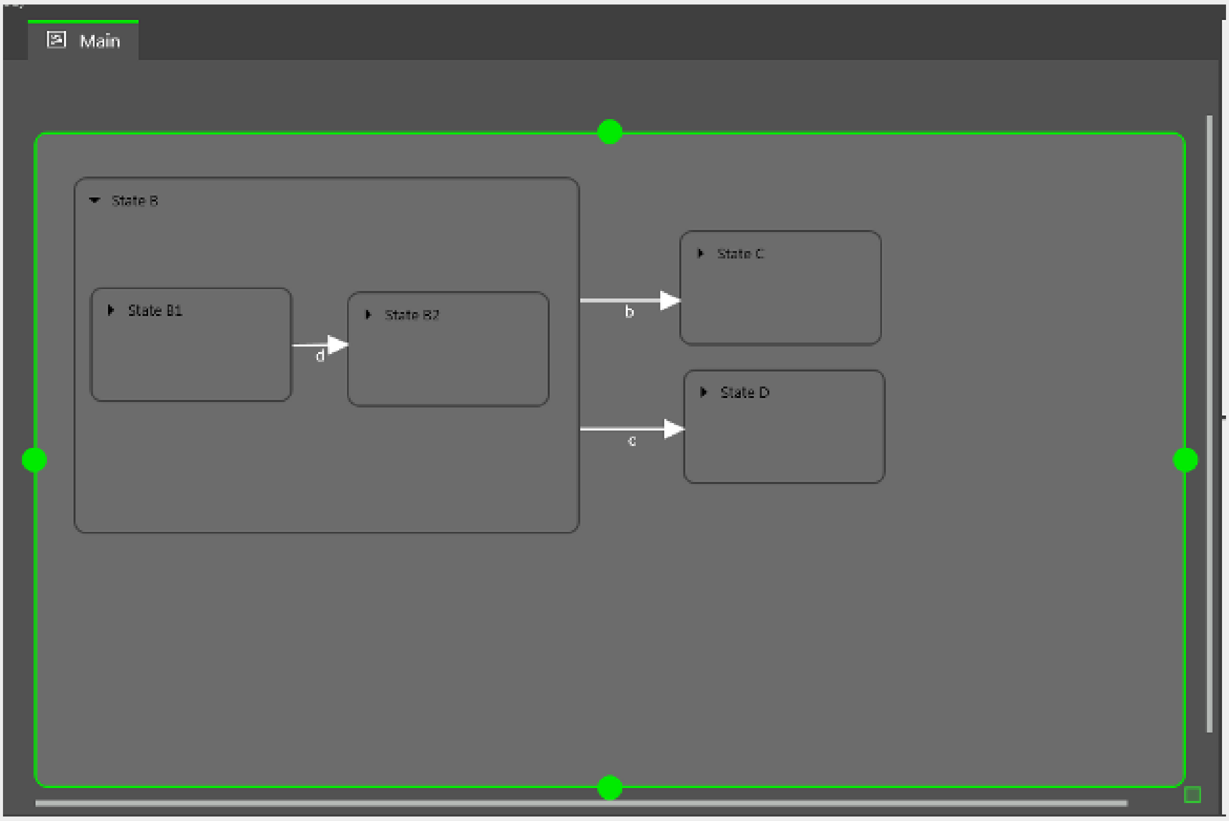


Figure 5.21. Transition inheritance

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 5.26.

Transition override

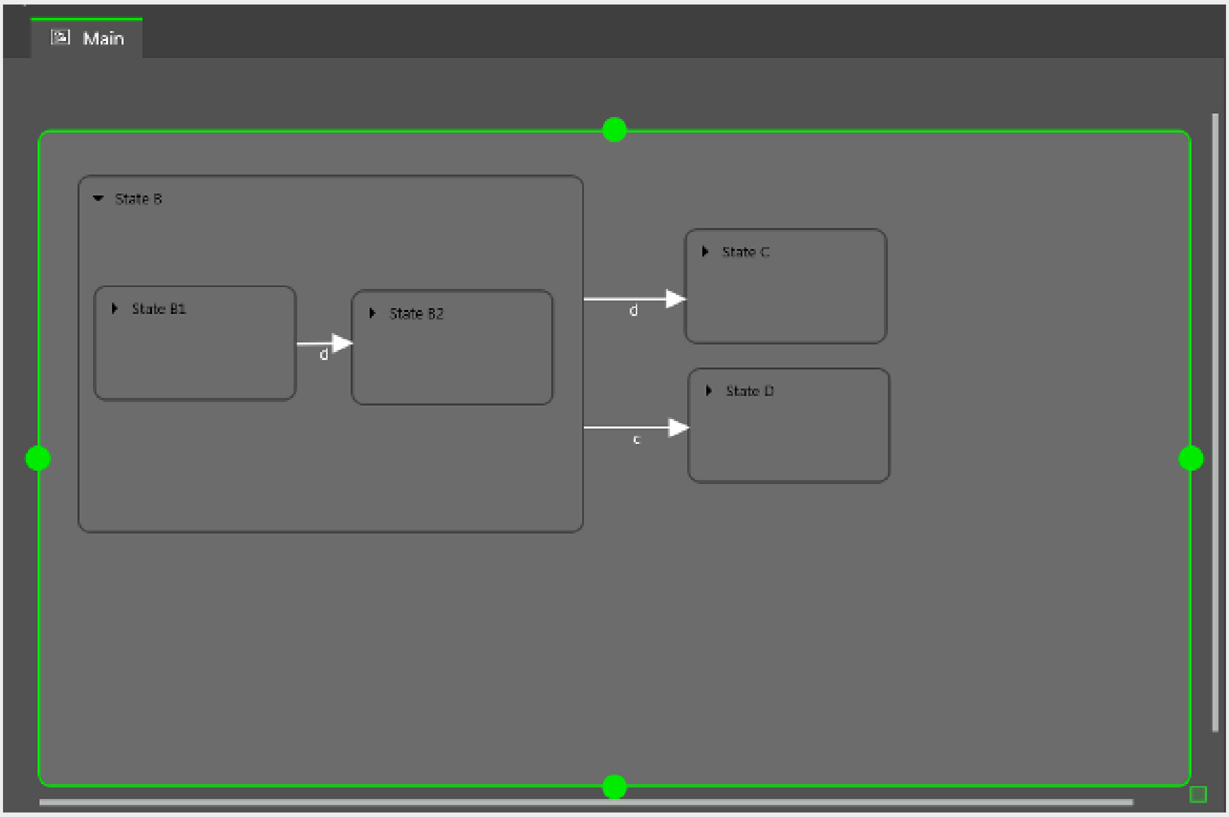


Figure 5.22. Transition override

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.

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.

5.4.11.4. 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 step ii 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 step 2 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 may 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 5.27.
Executing a transition

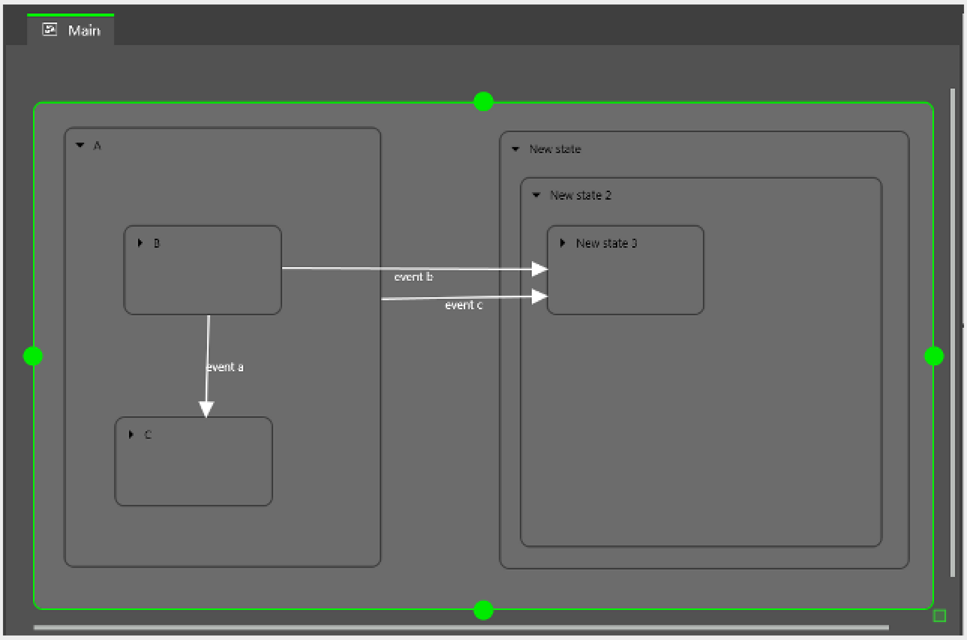


Figure 5.23. Entry/exit cascade

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 5.28.

Executing a transition

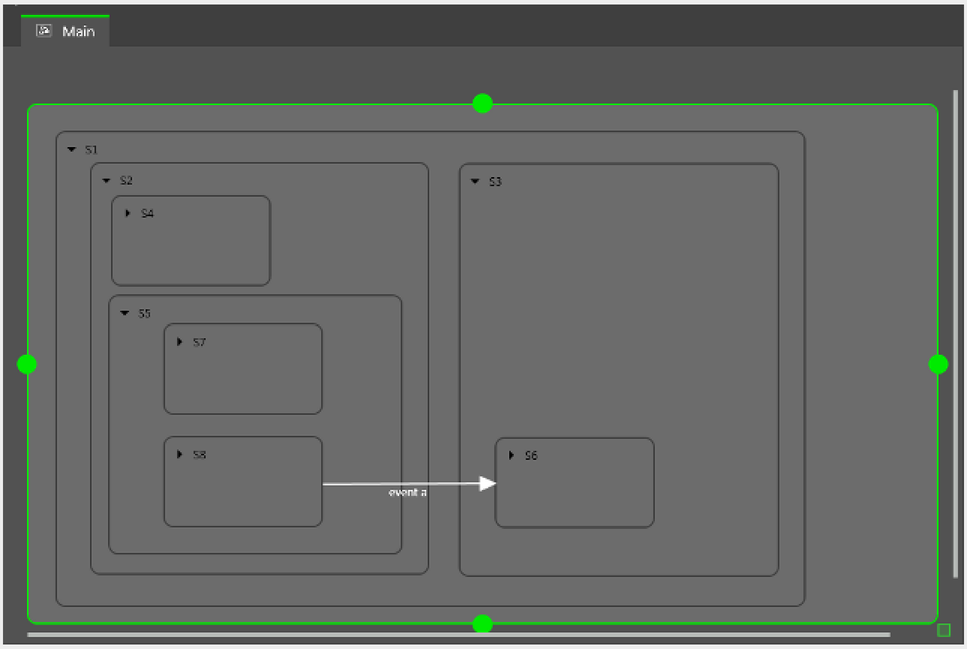


Figure 5.24. Executing a transition

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

1. State `S8` is exited.
2. State `S5` is exited.
3. State `S2` is exited.
4. State `S3` is entered.
5. State `S6` is entered.

Example 5.29.

Executing a transition

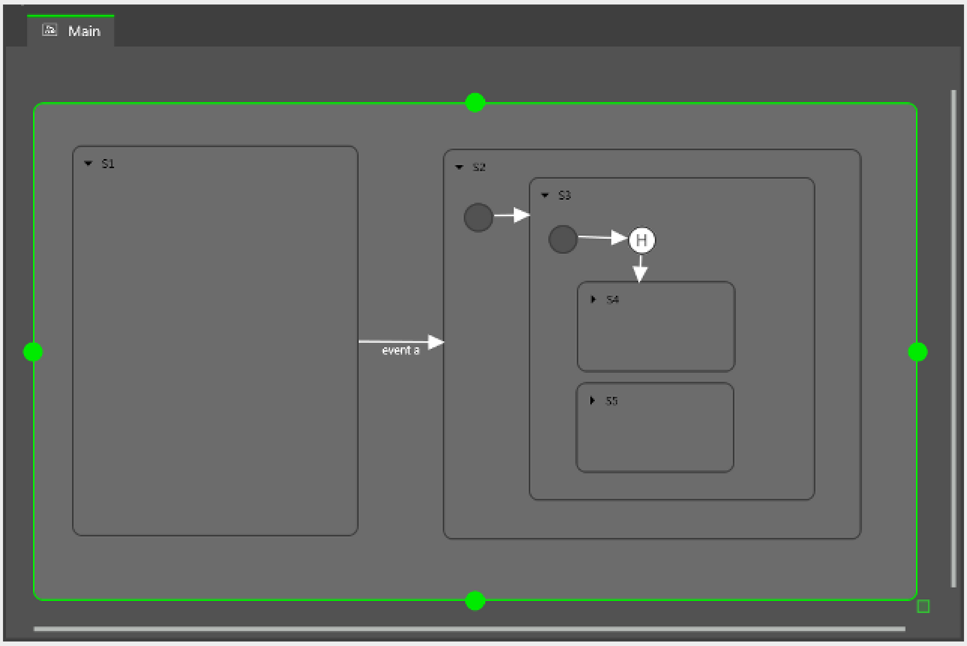


Figure 5.25. Executing a transition

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.

5.4.11.5. 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.

5.4.11.5.1. 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)

Name in EB GUIDE	Name in UML 2.5
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]

[a] EB GUIDE does not differentiate between external and local transitions.

5.4.11.5.2. Not supported elements

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

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

5.4.11.5.3. 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.
Time triggers at transitions	Concept is implemented using EB GUIDE Script (`fire_delayed`) in a state machine, a datapool item, a transition or a view.

5.4.12. 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

5.4.12.1. 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.

5.4.12.2. 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.

5.4.12.3. 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.

5.4.12.4. 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.

5.4.13. Widgets

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

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

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

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

5.4.13.1. View widget

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

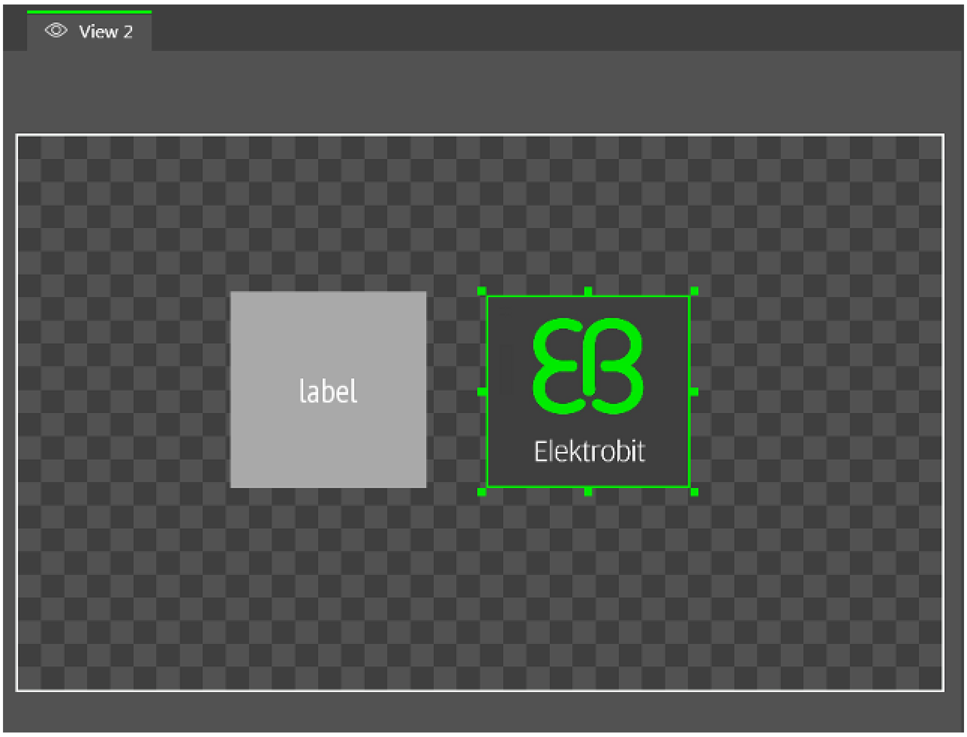


Figure 5.26. A view state that contains a rectangle, a label, and an image

5.4.13.2. Basic widgets

- Label

  A label widget places text into a view widget.

- Rectangle

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

- Image

  An image widget places a picture into the view widget. For supported file types see section 5.4.8.2, “Images”.

- Container

  A container widget holds several widgets as children and thus groups the widgets.

- Instantiator

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

5.4.13.3. Animations

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

- Animation

  An animation widget defines an animation for its parent widget. An animation widget requires at least one curve widget as a child widget.

- Constant curve bool, Constant curve color, Constant curve float, Constant curve integer

  A constant curve widget sets a target value after a defined delay.

- Fast start color, Fast start float, Fast start integer

  A fast start widget periodically sets a value that increases fast in the beginning but loses speed constantly until the end.

- Linear curve color, Linear curve float, Linear curve integer

  A linear curve widget periodically sets a value using a linear progression curve.

- Linear interpolation color, Linear interpolation float, linear interpolation integer

  A linear interpolation widget periodically sets a value using a linear interpolation curve.

- Quadratic curve color, Quadratic curve float, Quadratic curve integer

  A quadratic curve widget periodically sets a value using a quadratic function curve.

- Script curve bool, Script curve color, Script curve float, Script curve integer

A script curve widget sets a value using a curve that is described by an EB GUIDE Script.

- Sinus curve color, Sinus curve float, Sinus curve integer

  A sinus curve widget periodically sets a value using a sinus function curve.

- Slow start color, Slow start float, Slow start integer

  A slow start widget periodically sets a value that increases slowly in the beginning but rises constantly until the end.

5.4.13.4. 3D widgets

There are two kinds of 3D widgets. First, the 3D graphic widget which displays a 3D object. Second, four custom effect widgets which modify the graphical representation of a 3D graphic. Custom effects cannot be combined. Only one custom effect can be used on a 3D graphic at a time. Custom effect widgets influence a 3D graphic which is modeled as a child widget. Therefore, custom effect widgets are only regarded by the OpenGL ES 2.0 and DirectX 11 renderers.

NOTE

Supported renderers

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

3D widgets include the following:

- 3D graphic

  A 3D graphic widget places a 3D object into a view. For supported file types see section 5.4.8.3, "3D graphics".

- Material effect

  A material effect widget modifies the appearance of a 3D graphic. Placed as a parent to the 3D graphic widget, it customizes the color and shininess of the 3D graphic.

- Light effect

  A light effect widget modifies the appearance of a 3D graphic. Placed as a parent to the 3D graphic widget, it customizes the light position of the 3D graphic.

- Light and material effect

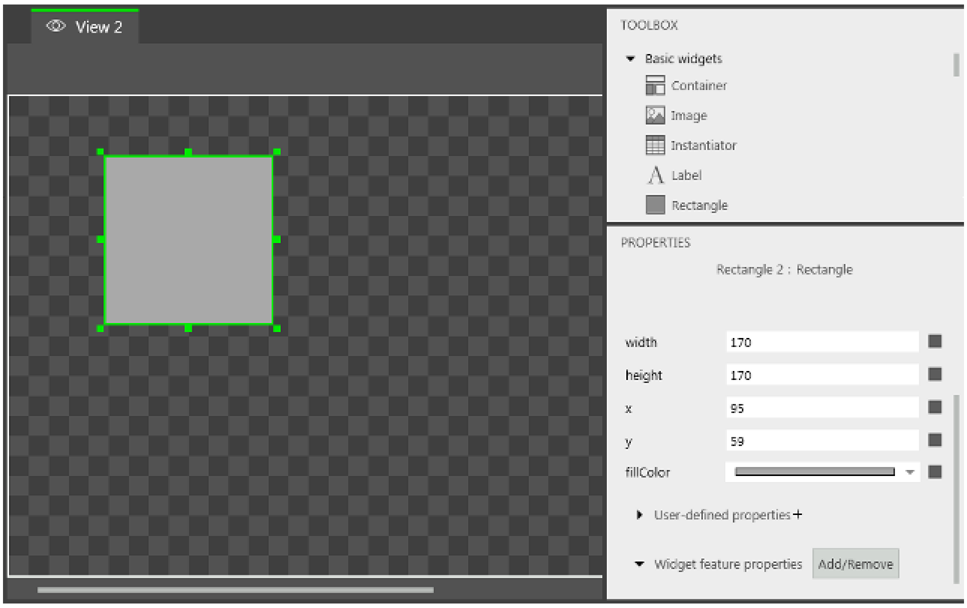
  A light and material effect widget modifies the appearance of a 3D graphic. Placed as a parent to the 3D graphic widget, it customizes the light position, the color, and the shininess of the 3D graphic.

- No lighting effect

A no lighting effect widget modifies the appearance of a 3D graphic. Placed as a parent to the 3D graphic widget, it disables 3D lighting of the 3D graphic. Disabling 3D lighting considerably speeds up rendering.

5.4.13.5. Widget properties

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



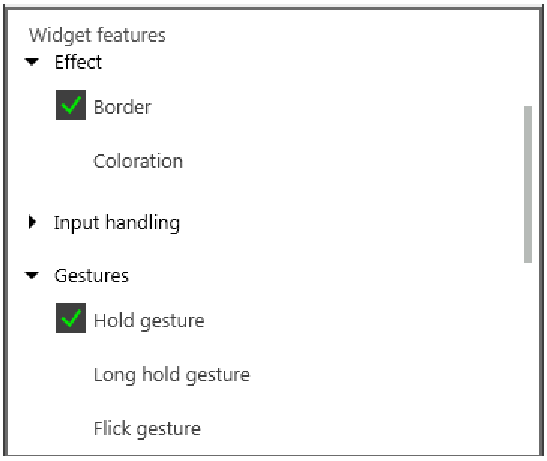
Rectangle 2 : Rectangle

width	170
height	170
x	95
y	59
fillColor	

Figure 5.27. A rectangle widget 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 chapter 9, “References“.
- User-defined widget properties are created by the modeler in addition to the default ones.
- Widget features supply widgets with additional properties. Adding a widget feature to a widget means adding one or more properties. Widget feature properties are grouped by categories.



Widget features

▾ Effect

☑ Border

Coloration

▸ Input handling

▾ Gestures

☑ Hold gesture

Long hold gesture

Flick gesture

Figure 5.28. Widget features

For example, the **Touched** widget features defines if and how a widget reacts to being touched by adding four properties: a flag `touchable`, a flag `touched`, and int values for `touch policy` and `touch behavior`.

5.4.13.6. Widget templates

A widget template allows adding a customized widget in several locations of an EB GUIDE model. You can define templates on the basis of existing widgets and then modify the template according to your needs.

Dragging a widget template to the content area means creating an instance from the widget template. Widget templates are displayed in the toolbox. Widget instances are displayed in the content area and in the navigation area.

A widget template has a template interface. The template interface specifies the properties of the template which are passed on to 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 🟦 icon.

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

Example 5.30.
Relation of the properties of a widget and its template

Let a widget `BlueSquare` be derived from a widget template `Square`. Let `Square` have a property `color`. Let the value of `color` be `red`.

- `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`.

5.5. Modelling HMI behavior

5.5.1. Modelling a state machine

5.5.1.1. Adding a state machine

Adding a state machine

Prerequisite:

- In the navigation area the **All** tab is displayed.

Step 1
In the navigation area point to **State machines**.

The ⊞ button appears.

Step 2
Click ⊞ .

A menu expands.

Step 3
Click a type for the state machine.

A new state machine of the selected type is added.

Step 4
Rename the state machine.

5.5.1.2. Defining an entry action for a state machine

Defining an entry action for a state machine

Prerequisite:

- A state machine is added to the EB GUIDE model.

Step 1
Select a state machine.

Step 2
In the **PROPERTIES** panel go to the `Entry action` property and click **Add**.

Step 3
Enter an action using EB GUIDE Script.

For background information see section 5.4.9, "Scripting language EB GUIDE Script"

Step 4
Click **Accept**.

You defined an entry action for a state machine.

5.5.1.3. Defining an exit action for a state machine

Defining an exit action for a state machine

Prerequisite:

- A state machine is added to the EB GUIDE model.

Step 1
Select a state machine.

Step 2
In the **PROPERTIES** panel go to the `Exit action` property and click **Add**.

Step 3
Enter an action using EB GUIDE Script.

For background information see section 5.4.9, "Scripting language EB GUIDE Script"

Step 4
Click **Accept**.

You defined an exit action for a state machine.

5.5.1.4. Deleting a state machine

Deleting a state machine

Prerequisite:

- In the navigation area the **All** tab is displayed.
- A state machine is added to the EB GUIDE model.

Step 1
In the navigation area right-click the state machine.

Step 2
In the context menu click **Delete**.

The state machine is deleted.

5.5.2. Modelling states

5.5.2.1. Adding a state

Adding a state

Prerequisite:

- In the content area, a state machine is displayed.

Step 1
Drag a state from the toolbox to the content area.

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.

5.5.2.2. 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:

- In the content area, a state machine is displayed.
- The state machine contains a compound state.

Step 1
In the content area click ▶ to expand the compound state.

Step 2
Drag a state from the toolbox to the compound state.

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

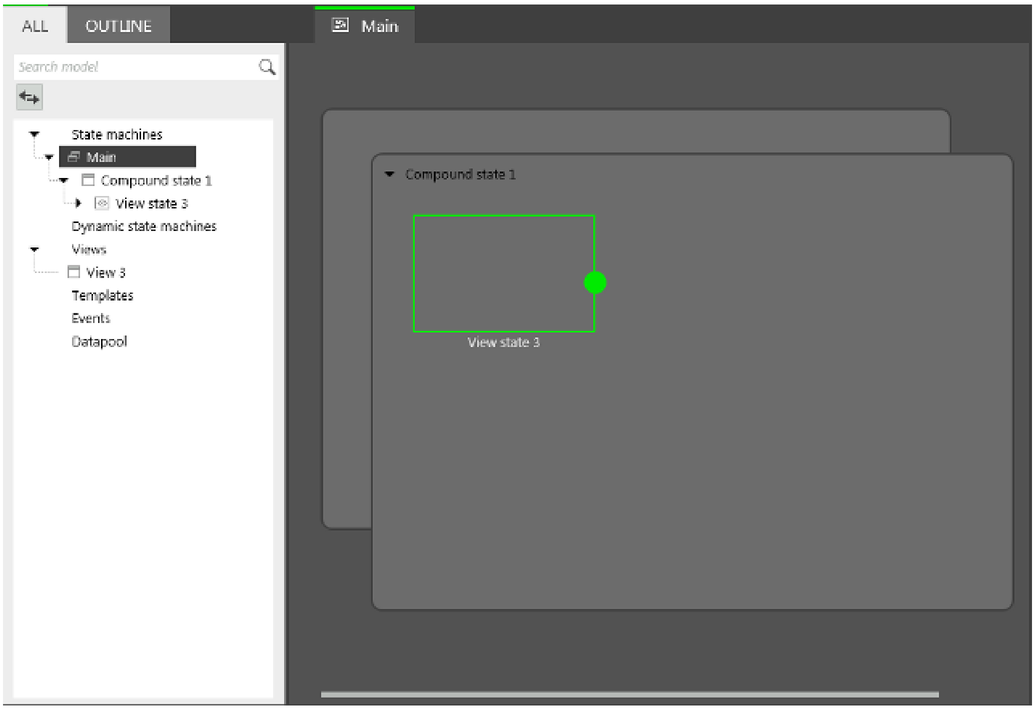


Figure 5.29. A compound state with a nested view state

5.5.2.3. Adding a choice state

Adding a choice state

Prerequisite:

- In the content area a state machine is displayed.
- The state machine contains at least two states.

Step 1
Drag a choice state from the toolbox to the content area.

Step 2
Add an outgoing transition from the choice state.

Step 3
Add a condition to the outgoing 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
Right-click the transition and 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`.

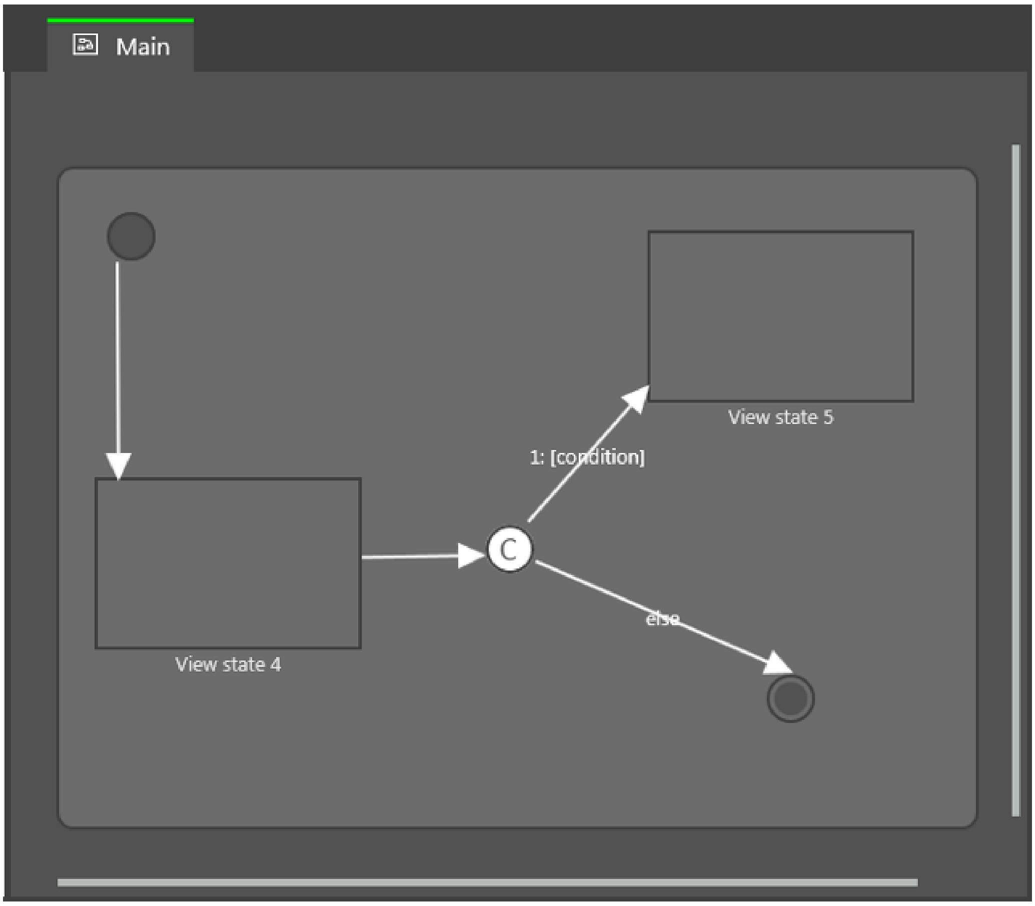


Figure 5.30. A choice state with its choice transitions

5.5.2.4. 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:

- In the content area, a state machine is displayed.
- The state machine contains a view state or a compound state.

Step 1
Select a state.

Step 2
In the **PROPERTIES** panel go to the `Entry action` property and click **Add**.

Step 3
Enter an action using EB GUIDE Script.

For background information see section 5.4.9, “Scripting language EB GUIDE Script”

Step 4
Click **Accept**.

5.5.2.5. 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:

- In the content area, a state machine is displayed.
- The state machine contains a view state or a compound state.

Step 1
Select a state.

Step 2
In the **PROPERTIES** panel go to the `Exit action` property and click **Add**.

Step 3
Enter an action using EB GUIDE Script.

For background information see section 5.4.9, “Scripting language EB GUIDE Script”

Step 4
Click **Accept**.

5.5.2.6. Deleting a model element from a state machine

Deleting a model element from a state machine

Prerequisite:

- In the content area, a state machine is displayed.

- The state machine contains at least one model element.

Step 1
In the navigation area right-click a model element.

Step 2
In the context menu click **Delete**.

The model element is deleted.

5.5.3. Connecting states through transitions

5.5.3.1. 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:

- In the content area, a state machine is displayed.
- 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 to the target state.

Step 4
When the target state is highlighted green, release the mouse button.

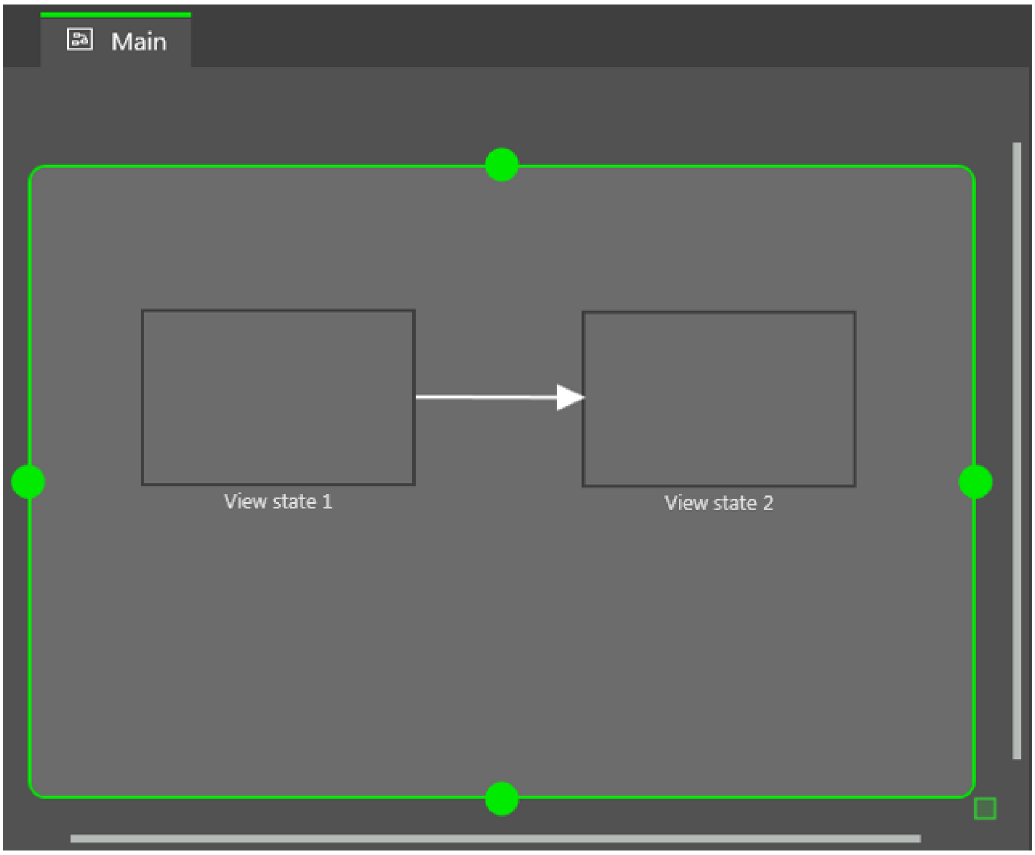


Figure 5.31. A transition

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.

5.5.3.2. Moving a transition

Moving a transition

You move a transition by moving one of its end points.

Prerequisite:

- In the content area, a state machine is displayed.

- The state machine contains at least two states.
- The states are connected by a transition.

Step 1
In the content area, select 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 to a different state.

Step 4
When the state is highlighted green, release the mouse button.

The transition is moved.

5.5.3.3. Defining a trigger for a transition

Defining a trigger for a transition

For a transition, you can define an event that triggers it.

Prerequisite:

- In the content area, a state machine is displayed.
- The state machine contains at least two states.
- The states are connected by a transition.

Step 1
Select a transition.

Step 2
In the **PROPERTIES** panel expand the **Trigger** combo box.

Step 3
Select an event.

Step 4
To create a new event, enter a name in the **Trigger** combo box and click **Add event**.

The event is added as a transition trigger.

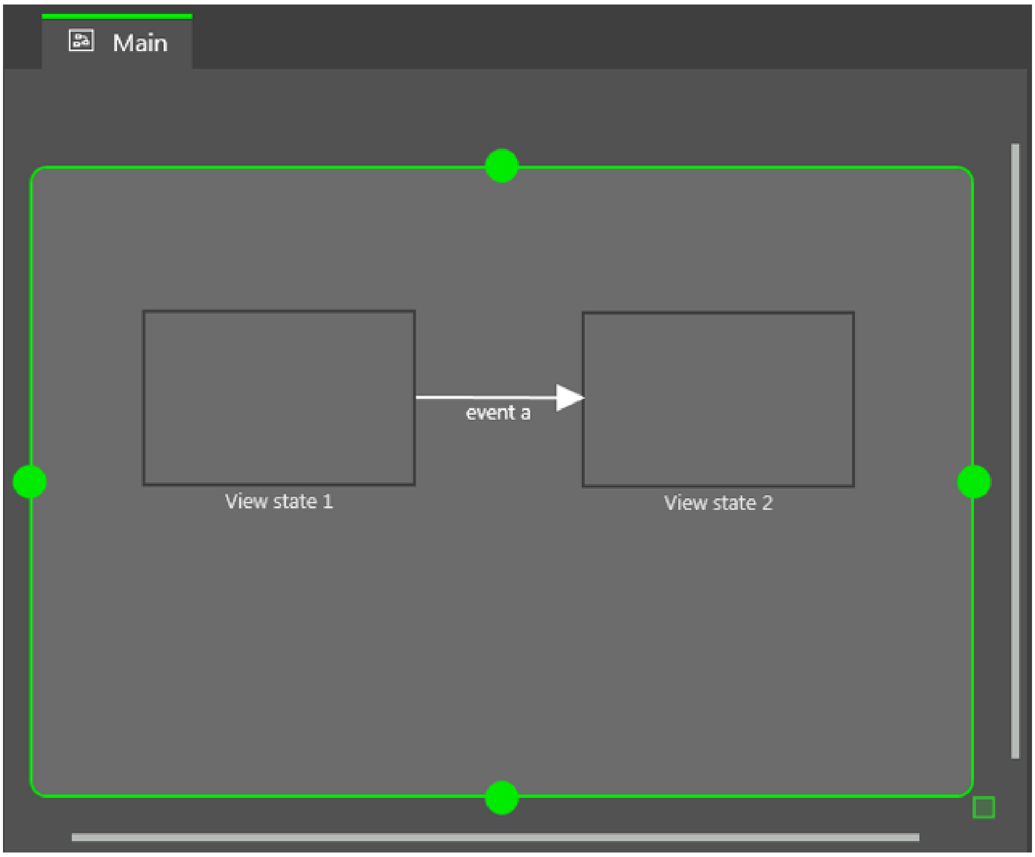


Figure 5.32. A transition with a trigger

5.5.3.4. 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:

- In the content area, a state machine is displayed.
- The 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, click **Add** in the **PROPERTIES** panel.

Step 3
Enter a condition using EB GUIDE Script.

For background information see section 5.4.9, "Scripting language EB GUIDE Script"

Step 4
Click **Accept**.

The condition is added to the transition.

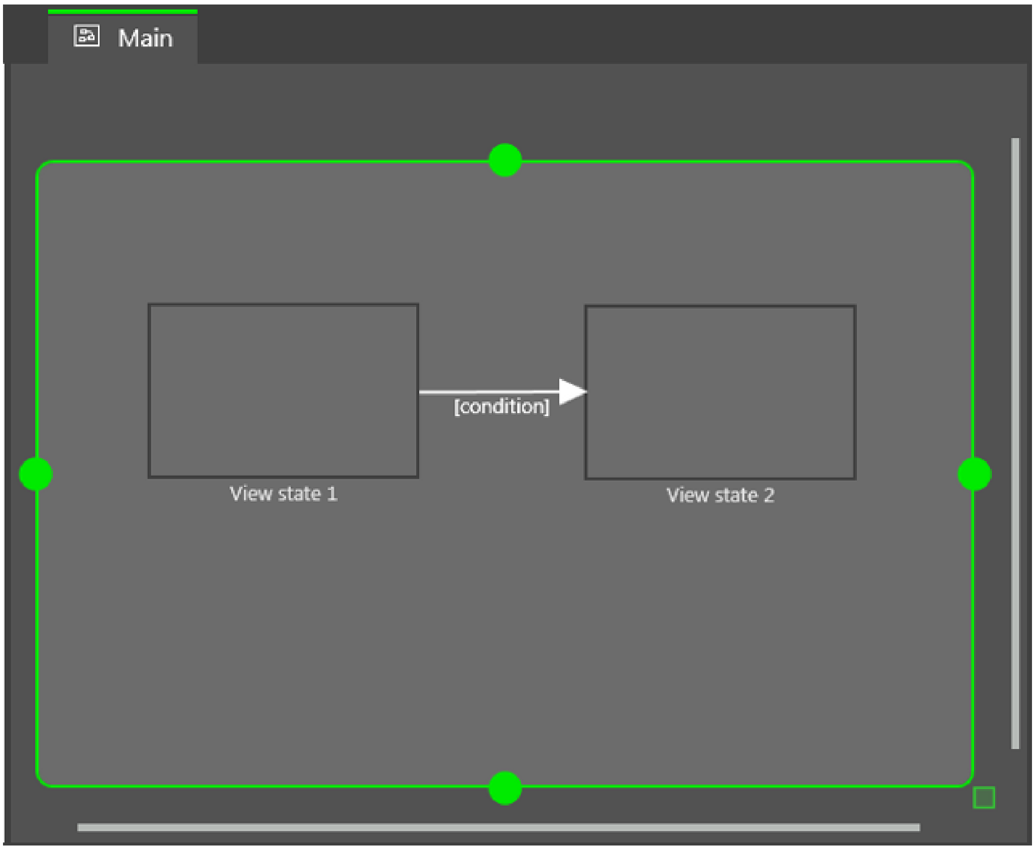


Figure 5.33. A transition with a condition

5.5.3.5. 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:

- In the content area, a state machine is displayed.

- The 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, click **Add** in the **PROPERTIES** panel.

Step 3
Enter an action using EB GUIDE Script.

For background information see section 5.4.9, “Scripting language EB GUIDE Script”

Step 4
Click **Accept**.

The action is added to the transition.

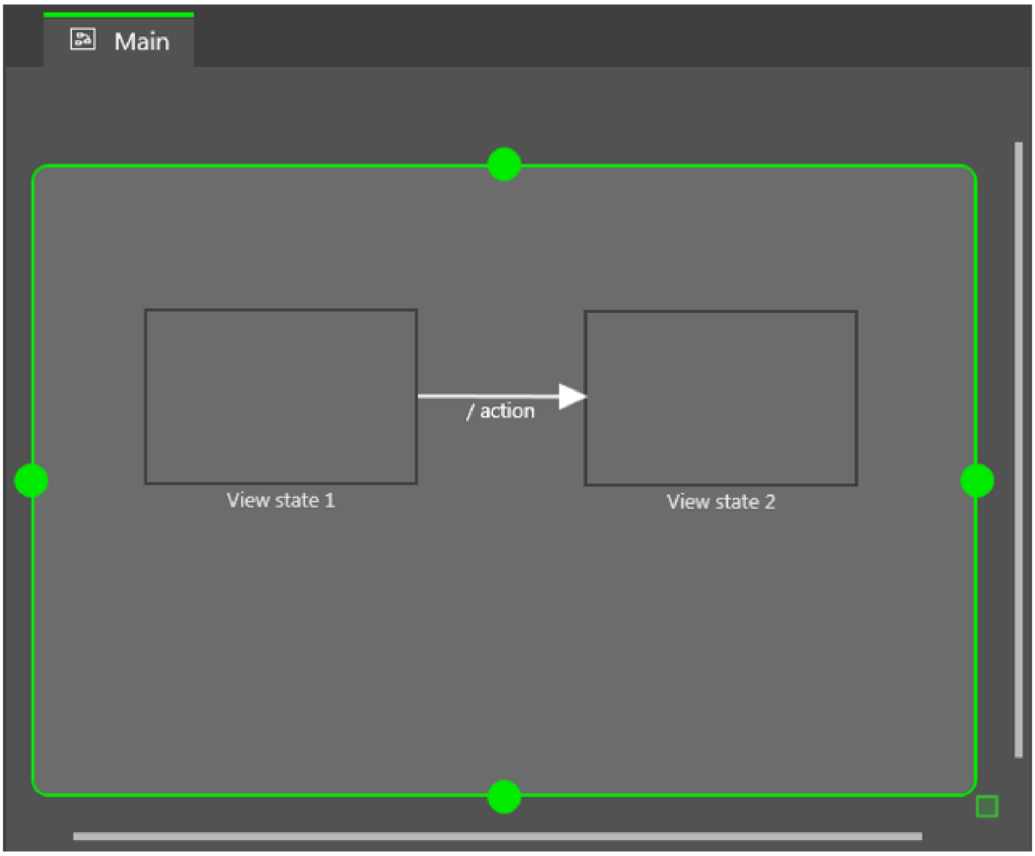


Figure 5.34. A transition with an action

5.5.3.6. Adding an internal transition to a state

Adding an internal transition to a state

Prerequisite:

- In the content area, a state machine is displayed.
- The state machine contains a state.

Step 1
In the content area, select a state.

Step 2
In the **PROPERTIES** panel, go to **Internal transitions** and click **Add**.

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

5.6. Modeling HMI appearance

5.6.1. Managing graphical elements

5.6.1.1. Adding a view

Adding a view

Prerequisite:

- In the content area a state machine is displayed.

Step 1
Drag a view state from the toolbox to the content area.

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

Step 2
Press the **F2** key and rename the view.

Step 3
Double-click the view state in the content area.

The content area displays the new view.

5.6.1.2. Adding a widget to a view

Adding a widget to a view

Prerequisite:

- In the content area a view is displayed.

Step 1
Drag a widget from the toolbox into the view.

The widget is added to the view.

5.6.1.3. Positioning a widget

Positioning a widget

Prerequisite:

- In the content area a view is displayed. The view contains a widget.

Step 1
Select a widget.

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

Step 2
To define the x coordinate of the widget enter a value in the `x` text box.

Step 3
To define the y coordinate of the widget enter a value in the `y` text box.

Step 4
Click outside the text box.

The content area displays the widget at the entered position.

TIP

Alternative approach

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

5.6.1.4. Resizing a widget

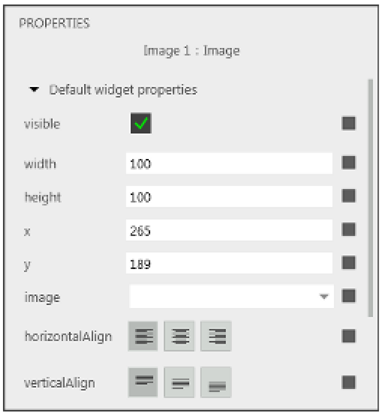
Resizing a widget

Prerequisite:

- In the content area a view is displayed. The view contains a widget.

Step 1
Select a widget.

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



PROPERTIES

Image 1 : Image

Default widget properties

visible ☑

width 100

height 100

x 265

y 189

image

horizontalAlign

verticalAlign

Figure 5.35. 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.

TIP

Alternative approach

To resize a widget by visual judgment, select the widget and drag one of its corners with the mouse.

5.6.1.5. Deleting a widget from a view

Deleting a widget from a view

Prerequisite:

- In the content area a view is displayed. The view contains a widget.

Step 1
Select a widget.

Step 2
Press the **Delete** key.

The widget is deleted from the view.

5.6.1.6. Adding an image to a view

Adding an image to a view

Prerequisite:

- An image file is located in the `$GUIDE_PROJECT_PATH\resources` directory. For supported file types see section 9.7.4.1.1, "Supported 3D graphic formats".
- In the content area a view is displayed.

Step 1
Drag an image widget from the toolbox to the view.

Step 2
In the **PROPERTIES** panel select an image from the `image` drop-down list box.

The view displays the image.

5.6.1.7. Grouping widgets

Grouping widgets

A container allows grouping widgets.

Step 1
Drag a container widget from the toolbox to 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 to the container.

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

5.6.1.8. Adding a 3D graphic to a view

Adding a 3D graphic to a view

Prerequisite:

- A 3D graphic file is located in the `$GUIDE_PROJECT_PATH\resources` directory. For supported 3D graphic formats see section 9.7.4.1.1, “Supported 3D graphic formats”.
- In the content area a view is displayed.

Step 1
Drag a 3D graphic widget from the toolbox to the view.

Step 2
In the **PROPERTIES** panel select a 3D graphic file from the `model` drop-down list box.

The view displays the 3D graphic.

5.6.1.9. Changing the font of a label

Changing the font of a label

Prerequisite:

- A TTF file is located in the `$GUIDE_PROJECT_PATH\resources` directory.
- In the content area a view is displayed. The view contains a label.

Step 1
Select the label in the view.

Step 2
In the **PROPERTIES** panel select a font from the `font` drop-down list box.

The view displays the label with the new font.

5.6.1.10. 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.

Linking widget properties is only possible in the following cases:

- Between child widgets of the same parent widget
- Between a parent widget and a child widget

Prerequisite:

- In the content area a view is displayed.
- The view contains a rectangle widget.

Step 1
In the content area select the rectangle widget.

The **PROPERTIES** panel displays the properties of the rectangle widget.

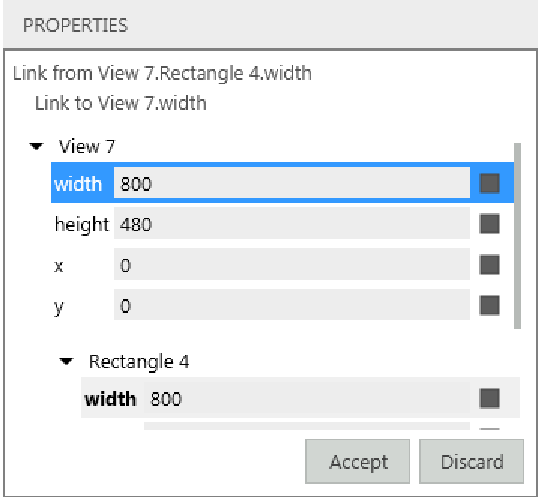
Step 2
In the **PROPERTIES** panel go to the `width` property and click the ■ icon next to the property.

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.



PROPERTIES

Link from View 7.Rectangle 4.width
Link to View 7.width

View 7
width 800
height 480
x 0
y 0

Rectangle 4
width 800

Figure 5.36. Linking between widget properties

Step 5
Click **Accept**.

The dialog closes. The 🟨 icon is displayed next to the `width` property. The icon indicates that the `width` property of the rectangle widget is now linked to the `width` property of the view widget. Whenever you change the width of the view, the width of the rectangle changes and vice versa.

5.6.1.11. 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 widget to a new datapool item.

Prerequisite:

- In the content area a view is displayed.

- The view contains an image widget.

Step 1
In the content area select the image widget.

The **PROPERTIES** panel displays the properties of the image widget.

Step 2
In the **PROPERTIES** panel go to the `image` property and click the ■ icon next to the property.

A menu expands.

Step 3
In the menu click **Add link to datapool item**.

A dialog opens.

Step 4
To add a new datapool item, enter a name in the combo box.

Step 5
Click **Add datapool item**.

A dialog opens.

Step 6
Select a type for the datapool item.

Step 7
Click **Accept**.

Step 8
If the datapool item is of a list type, enter an index in the **Value** text box.

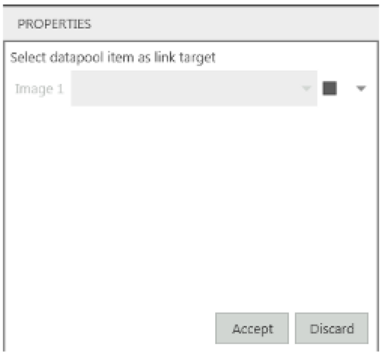


Figure 5.37. Linking to a datapool item

Step 9
Click **Accept**.

The dialog closes. The 🟧 icon is displayed next to the `image` property. The icon indicates that the `image` property is now linked to a datapool item. Whenever you change the image, the datapool item changes and vice versa.

5.6.1.12. Adding a user-defined property to a widget

Adding a user-defined property to a widget

Prerequisite:

- In the content area a view is displayed.
- A widget is added to the view.

Step 1
Select a widget in the content area.

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

Step 2
In the **PROPERTIES** panel go to the **User-defined properties** category and click the **Add** button.

A menu expands.

Step 3
In the menu select a type for the user-defined property.

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

Step 4
Rename the user-defined property.

5.6.1.13. Adding a widget feature

Adding a widget feature

Widget features supply widgets with additional properties. Adding a widget feature to a widget means adding one or more properties.

For a list of widget features grouped by categories see section 9.8, “Widget features”.

Prerequisite:

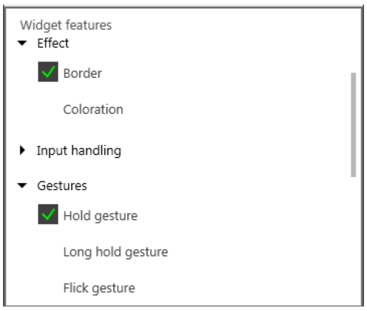
- In the content area a view is displayed.
- A widget is added to the view.

Step 1
Select a widget in the content area.

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

Step 2
In the **PROPERTIES** panel go to the **Widget features** category and click the **Add/Remove** button.

The **Widget features** dialog is displayed.



Widget features
- Effect
  - ☑ Border
  - Coloration
- Input handling
- Gestures
  - ☑ Hold gesture
  - Long hold gesture
  - Flick gesture

Figure 5.38. Widget features dialog

Step 3
Expand a category and select the widget feature you want to add.

The related properties are added to the widget and displayed in the **PROPERTIES** panel.

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 **State Touched** and **Touch Move** are added automatically.

5.6.1.14. Removing a widget feature

Removing a widget feature

Prerequisite:

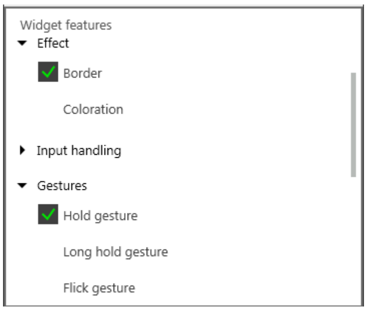
- In the content area a view is displayed.
- A widget is added to the view.
- Widget features are added to the widget.

Step 1
Select the widget in the content area.

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

Step 2
In the **PROPERTIES** panel go to the **Widget features** category and click the **Add/Remove** button.

The **Widget features** dialog is displayed.



Widget features
- Effect
  - ☑ Border
  - Coloration
- Input handling
- Gestures
  - ☑ Hold gesture
  - Long hold gesture
  - Flick gesture

Figure 5.39. Widget features dialog

Step 3
Expand a category and clear the widget feature you want to remove.

The related properties are removed from the **PROPERTIES** panel.

NOTE

Removing widget features with dependencies

Widget feature with dependencies to other widget features cannot be removed directly. Clear the parent widget feature before you clear the child widget feature.

5.6.2. Adding a language to the EB GUIDE model

To enable language support during run-time, you add languages to the project.

5.6.2.1. Adding a language

Adding a language

The first language in the list is always the default language and can not 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 ⊞ .

A language is added to the table.

Step 4
Enter a name for the language.

Step 5
Select a language from the **Language** drop-down list box.

Step 6
Select a country from the **Country** drop-down list box.

Step 7
Enter a unique numerical ID in the **Identifier** text box.

5.6.2.2. Deleting a language

Deleting a language

Prerequisite:

- At minimum two languages are added to the project.

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 the **Delete** button.

The language is deleted from the table.

5.6.3. Re-using an element

5.6.3.1. Creating a template

Creating a template

Prerequisite:

- In the content area a view is displayed.

Step 1
In the toolbox, right-click the widget you want to create a template from.

Step 2
In the context menu click **Create template**.

A template is created and opened in a new tab.

Step 3
Press the **F2** key and rename the template in the navigation area.

Step 4
Select the template in the content area.

Step 5
In the **PROPERTIES** panel edit the template's properties and define the template interface.

5.6.3.2. Defining the template interface

Defining the template interface

Prerequisite:

- In the content area, a template is displayed.

Step 1
In the content area, select a template.

Step 2
To add a property to the template interface, right-click the ■ icon next to the property. In the context menu, click **Add to template interface**.

The ● icon is displayed next to the property.

Step 3
To remove a property from the template interface, right-click the ■ icon next to the property. In the context menu, click **Remove from template interface**.

The ● icon is no longer displayed next to the property.

5.6.3.3. Using a template

Using a template

Prerequisite:

- In the content area a view is displayed.
- In the toolbox a template is available.

Step 1
Drag a widget template from the toolbox to the view.

An instance of the template is added to the view. The **PROPERTIES** panel displays the properties which belong to the template interface.

TIP

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

Step 2
In the **PROPERTIES** panel edit the properties of the template instance.

After editing a property, the 🟦 icon changes to the ■ icon.

Step 3
To reset a property value to the value of the template, right-click the ■ icon next to the property. In the context menu click **Reset to template value**.

5.6.4. Tutorial: Modelling a path gesture

Tutorial: Modelling a path gesture

For a list of widget features for gestures see section 9.8.4, "Gestures".

Prerequisite:

- In the content area a view is displayed.

Step 1
Drag a rectangle widget from the toolbox to the view.

Step 2
Drag a label widget from the toolbox to the rectangle.

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

The **PROPERTIES** panel displays the properties of the label widget.

Step 3
Select the rectangle widget.

The **PROPERTIES** panel displays the properties of the rectangle widget.

Step 4
In the **PROPERTIES** panel go to `fillColor` and select red.

Step 5
In the **PROPERTIES** panel go to the **Widget feature** category and click the **Add/Remove** button.

The **Widget features** dialog is displayed.

Step 6
Expand the **Gesture** category and select `Path gestures`.

The `State touched` widget feature is automatically selected, as it is required for the `Gesture` widget feature.

The related properties are added to the rectangle widget and displayed in the **PROPERTIES** panel.

Step 7
For the `Path gestures` widget feature edit the following properties:

Step 7.1
Next to the `onPath` property click the **Edit** button.

Step 7.2
Enter the following EB GUIDE Script expression:

```
 v:this->"Label 1".text = "recognized path gesture #"
+ f:int2string(v:gestureId);
```

Step 7.3
Next to the `onPathNotRecognized` property click the **Edit** button.

Step 7.4
Enter the following EB GUIDE Script expression:

```
v:this->"Label 1".text = "shape not recognized";
```

Step 7.5
Next to the `onPathStart` property click the **Edit** button.

Step 7.6
Enter the following EB GUIDE Script expression:

```
v:this->"Label 1".text = "path gesture start";
```

Step 7.7
In the `pathMinXBox` text box, enter `50`.

Step 7.8
In the `pathMinXBox` text box, enter `50`.

5.7. Handling data

5.7.1. Adding an event

Adding an event

Prerequisite:

- In the navigation area the **All** tab is displayed.

Step 1
In the navigation area point to **Events**.

The ⊞ button appears.

Step 2
Click ⊞ .

An event is added to the navigation area.

Step 3
Rename the event.

5.7.2. Adding a parameter to an event

Adding a parameter to an event

Prerequisite:

- In the navigation area the **All** tab is displayed.
- An event is added.

Step 1
In the navigation area click an event.

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

Step 2
In the **PROPERTIES** panel point to **Parameters**.

The ⊞ button appears.

Step 3
Click ⊞ .

A parameter is added to the event.

Step 4
Rename the parameter.

Step 5
Select a type for the parameter.

5.7.3. 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

The group IDs 0 to 65535 are reserved for internal use within the EB GUIDE product line.

Step 1
Click ▣.

The project center opens.

Step 2
In the navigation area click **CONFIGURE** > **Event groups**.

Step 3
In the content area click the **Add** button.

An event group is added to the table.

Step 4
Rename the event group.

Step 5
To change an event group ID, type a number for **ID**.

Addressing an event for EB GUIDE TF

Prerequisite:

- An event group is added.
- The project editor is displayed.
- In the navigation area the **All** tab is displayed.
- In the navigation area an event is added.

Step 1
In the navigation area click an event.

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

Step 2
In the **PROPERTIES** panel insert an ID in the **Event ID** text box.

Step 3
In the **PROPERTIES** panel select an event group from the **Event group** drop-down list box.

5.7.4. Deleting an event

Deleting an event

Prerequisite:

- In the navigation area the **All** tab is displayed.
- An event is added.

Step 1
In the navigation area right-click the event.

Step 2
In the context menu click **Delete**.

The event is deleted.

5.7.5. Adding a datapool item

Adding a datapool item

Prerequisite:

- In the navigation area the **All** tab is displayed.

Step 1
In the navigation area point to **Datapool**.

The ⊞ button appears.

Step 2
Click ⊞ .

A menu expands.

Step 3
In the menu click a type for the datapool item.

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

Step 4
Rename the datapool item.

5.7.6. Establishing external communication

To establish external communication for example between the EB GUIDE model and an application, you add communication contexts to the project.

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**.

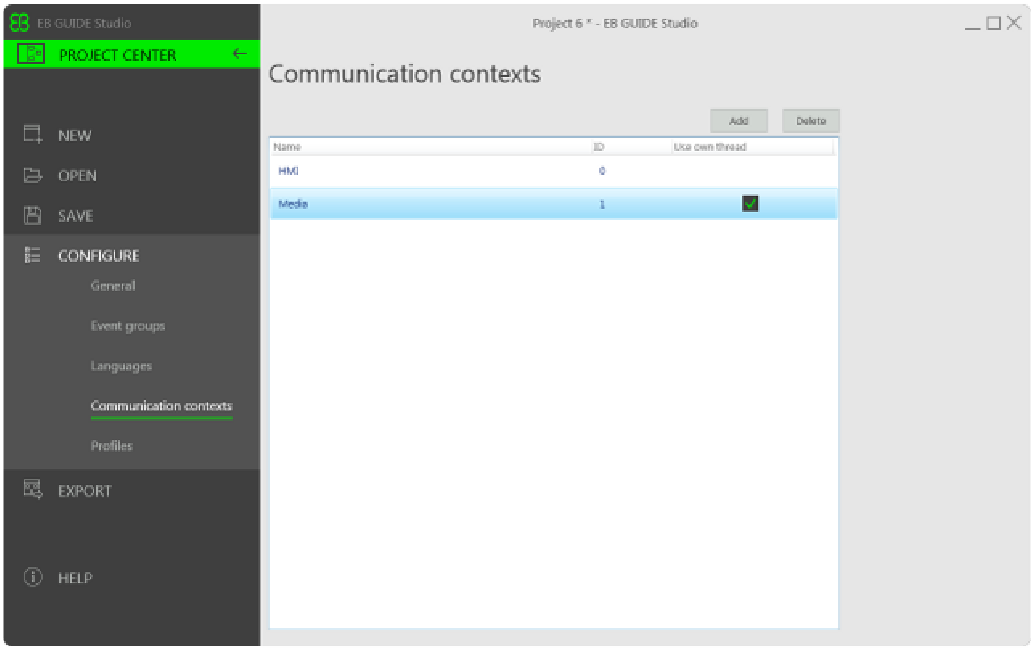
Step 3
In the content area click the **Add** button.

A communication context is added to the table.

Step 4
Rename the communication context, for example to `Media`.

Step 5
To change a communication context ID, enter a number in the **ID** text box.

Step 6
To run the communication context in an own thread, select **Use own thread**.



Communication contexts

Name	ID	Use own thread
HMI	0	
Media	1	☑

Figure 5.40. Communication context `Media`.

Using external communication in a datapool item

Prerequisite:

- At minimum two communication contexts are added to the project .

- The project editor is opened.
- In the navigation area the **All** tab is displayed.
- A datapool item is added.

Step 1
In the navigation area click the datapool item.

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

Step 2
In the **PROPERTIES** panel select a communication context from the `Reader context` drop-down list box, for example `hmi`.

Step 3
In the **PROPERTIES** panel select a different communication context from the `Writer context` drop-down list box, for example `media`.

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

In the instruction above the data is sent from `hmi` to `media`.

5.7.7. Linking between datapool items

Linking between datapool items

Prerequisite:

- A datapool item is added in the navigation area.

Step 1
In the navigation area click a datapool item.

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

Step 2
In the **PROPERTIES** panel go to the `Value` property and click the ■ icon next to the property.

A menu expands.

Step 3
In the menu, click **Add link to datapool item**.

A dialog opens.

Step 4
To add a new datapool item, enter a name in the combo box.

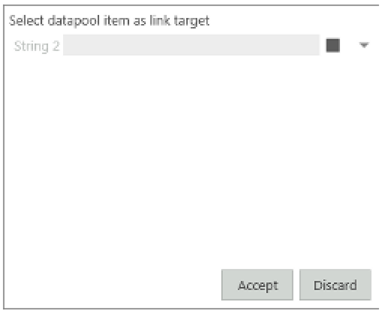
Step 5
Click **Add datapool item**.

A dialog opens.

Step 6
Select a type for the datapool item.

Step 7
Click **Accept**.

Step 8
If the datapool item is of a list type, enter an index in the **Value** text box.



Select datapool item as link target

String 2

Figure 5.41. Linking between datapool items

Step 9
Click **Accept**.

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

5.7.8. Deleting a datapool item

Deleting a datapool item

Prerequisite:

- In the navigation area the **All** tab is displayed.
- A datapool item is added.

Step 1
In the navigation area right-click the datapool item.

Step 2
In the context menu click **Delete**.

The datapool item is deleted.

5.8. Handling a project

5.8.1. Creating a project

Creating a project

Step 1
Click 🟩.

The project center opens.

Step 2
In the navigation area, click the **NEW** tab.

Step 3
Enter a project name and select a location.

Step 4
Click the **CREATE** button.

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

5.8.2. Opening a project

5.8.2.1. Opening a project from the file explorer

Opening a project from the file explorer

Prerequisite:

- An EB GUIDE Studio project already exists.

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.

5.8.2.2. Opening a project within EB GUIDE Studio

Opening a project within EB GUIDE Studio

Prerequisite:

- An EB GUIDE Studio project already exists.

Step 1
Click ▣.

The project center opens.

Step 2
In the navigation area, click the **OPEN** tab.

Step 3
Select the EB GUIDE Studio project file you would like to open. EB GUIDE Studio project files have the file extension `.ebguide`. Click the **OPEN** button.

The project opens in EB GUIDE Studio.

5.8.3. Saving a project

Saving a project

Prerequisite:

- A project is opened in EB GUIDE Studio.
- New widgets are added to the project.

Step 1
Click ▣.

The project center opens.

Step 2
In the navigation area, click the **SAVE** tab.

The project is saved.

5.8.4. Testing and improving an EB GUIDE model

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

5.8.4.1. Validating an EB GUIDE model

Validating an EB GUIDE model

In the problems area, EB GUIDE displays the following:

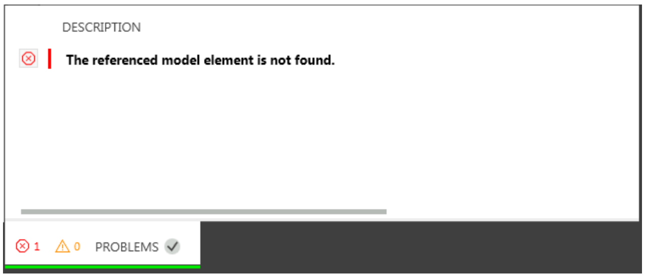
- ⊗ errors
- ⚠ warnings

Step 1
In the problems area, click the ✓ button.

The number of errors and warnings is displayed.

Step 2
Click **Problems** to expand the problems area.

A list of errors and warnings is displayed.



DESCRIPTION

⊗ The referenced model element is not found.

Figure 5.42. Problems area

Step 3
To navigate to the source of a problem, double-click the corresponding line.

The element that causes the problem is highlighted.

Step 4
Solve the problem.

Step 5
Click the **VALIDATE** button.

The problem you solved is no longer listed in the problems area.

Step 6
To collapse the problems area, click **Problems** once again.

5.8.4.2. Starting 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.

5.8.5. Exporting a project

Exporting a project

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

Step 1
Click ▣.

The project center opens.

Step 2
In the navigation area click the **EXPORT** tab.

Step 3
Select a location where to export the binary files.

Step 4
Click the **EXPORT** button.

The binary files are exported to the selected location.

6. System integrator's manual

6.1. Overview

As a system integrator you are the target audience for the following chapters. For more information, see section 1.1.2, “System integrators”.

For more information on the structure of the manual, see section 1.2, “Structure of user documentation”.

6.2. Background information

6.2.1. Android APK

The Android application package (APK) file format is used to distribute and install applications and other middleware on Android devices.

6.2.1.1. System requirements

The Android APK version that is currently released for EB GUIDE TF is designed to run on a wide range of Android devices. It therefore fulfills the minimal requirements of most devices.

Table 6.1. Minimal requirements

Architecture	ARMv7
Platform	EB GUIDE TF: Android 4.3 (API Level 18).

6.2.1.2. Features of the EB GUIDE TF APK

Table 6.2. Features of the EB GUIDE TF APK

Feature	Description
Lifecycle management	EB GUIDE TF supports Android lifecycle management, see section 9.1.1, “Android lifecycle management”

Feature	Description
Multi-touch support	EB GUIDE TF supports up to ten fingers for multi-touch. The number of supported fingers may be limited by the Android device.
Key handling	EB GUIDE TF processes 16-bit UTF key mapping codes.
Interaction with the Java API	EB GUIDE TF can be started and controlled by the Android activity. Example code and a template implementation are provided by the application. A native activity is not necessary.
Android layout handling	The exported EB GUIDE model is informed through events if the layout of the visible screen area changes. That way you can handle a virtual keyboard or changes in rotation.

6.2.1.3. Description of the EB GUIDE TF APK files

- `EB GUIDE Launcher.apk`

  `EB GUIDE Launcher.apk` is started by EB GUIDE Model Chooser. It starts EB GUIDE TF and displays the exported EB GUIDE model.

  Alternatively `EB GUIDE Launcher.apk` can be started without EB GUIDE Model Chooser. It then displays the exported EB GUIDE model that was selected last by EB GUIDE Model Chooser.

  If `EB GUIDE Launcher.apk` is not yet started by EB GUIDE Model Chooser or if the files from the exported EB GUIDE model are deleted, an message is displayed.

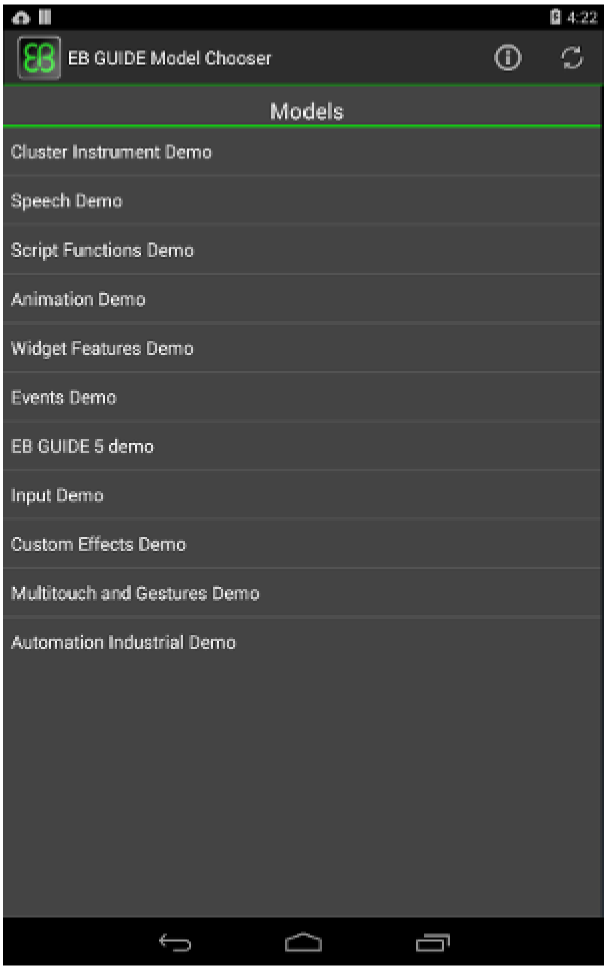
- `EB GUIDE Model Chooser.apk`

  EB GUIDE Model Chooser provides a user interface to select an exported EB GUIDE model that is executed on the Android system. By selecting an exported EB GUIDE model, `EB GUIDE Launcher.apk` is started with the corresponding model.

  Clicking the **Refresh** icon updates the exported EB GUIDE model list.

  Clicking the **Info** icon displays the directory where exported EB GUIDE models are stored for EB GUIDE Model Chooser, and a list with device-related details.

  For information about the location of the exported EB GUIDE models in the file system, see section 9.1.2, “File path for models”.



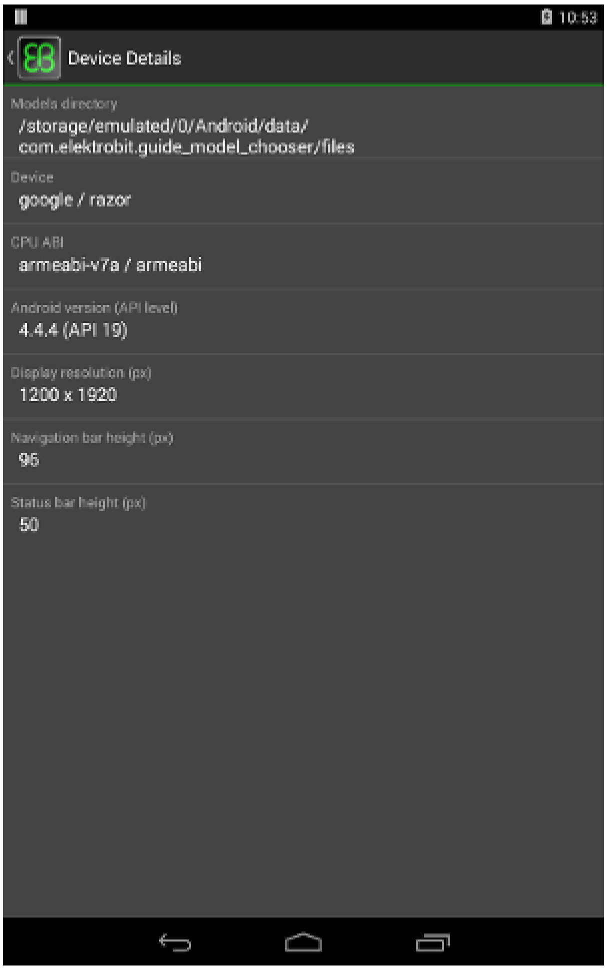
4:22

EB GUIDE Model Chooser

Models

Cluster Instrument Demo
Speech Demo
Script Functions Demo
Animation Demo
Widget Features Demo
Events Demo
EB GUIDE 5 demo
Input Demo
Custom Effects Demo
Multitouch and Gestures Demo
Automation Industrial Demo

Figure 6.1. EB GUIDE Model Chooser



10:53

Device Details

Models directory
/storage/emulated/0/Android/data/
com.elektrobit.guide_model_chooser/files

Device
google / razor

CPU ABI
armeabi-v7a / armeabi

Android version (API level)
4.4.4 (API 19)

Display resolution (px)
1200 x 1920

Navigation bar height (px)
96

Status bar height (px)
50

Figure 6.2. Device Details

6.2.1.4. Restrictions

The Android APK that is currently released for EB GUIDE TF has the following restrictions:

- The exported EB GUIDE model is informed about rotation changes and layout changes, for example an incoming virtual keyboard on the display. It is the responsibility of the exported EB GUIDE model to handle these events.
- If the system uses Android layout handling, the Android flag SOFT_INPUT_ADJUST_NOTHING must not be set in the configuration of the Android activity.

6.2.1.5. Released APK and custom APK

EB GUIDE TF is delivered and installed as an APK. Use either a pre-built released APK of a released version or create a custom version based on the delivered Android binaries and the APK template in the SDK.

The following lists help you to decide whether or not you need a custom APK.

If the following applies to your project, use the released APK:

- It contains EB GUIDE functionality or feature demonstrations with no further extensions.
- It contains project-specific extensions, for example EB GUIDE TF plugins, to be added to the exported EB GUIDE model.
- Standard access rights are sufficient. The standard access rights are read or write to the external storage of the device, network access `android.permission.INTERNET`, record audio, and modify audio settings.

If the following applies to your project, use the delivered APK template:

- You need additional access rights that are not granted by the released APK version, for example CALL_-PHONE.
- You require a customer-specific APK, for example a customer signature for APK verification or icons.
- You use Android framework features that are not accessible in the stable API of the native development kit (NDK). The NDK contains only a small subset of features and functionality which you can use with the Java API.
- You need additional Android application functionalities that require modifications to Java-related code pieces, for example activities, services, skins, intents, or compositing.

6.2.2. Application simulation

When simulating the application, the tool EB GUIDE Monitor observes and controls a running EB GUIDE model. EB GUIDE Monitor includes mechanisms for communication with the datapool, the event system, and the state machines of a running and connected EB GUIDE model.

6.2.2.1. Control panels

There is a GUI component that holds a control panel. Specify the components and layout of the control panel in XML format.

The XML file has a mandatory root element named `panel` and optional child elements. You can add the following child elements as often as required:

- `button`: A clickable button

  Child elements:

  - `bounds`: The size and location of the button
  - `text`: The text on the button
  - `onPress`: JavaScript code for the press action
  - `onRelease`: JavaScript code for the release action

- `rotary`: A rotary input

  Child elements:

  - `bounds`: The size and location of the rotary input
  - `stepSize`: The value change for each rotary step

  Attributes:

  - `displayID`: The ID of the display the rotary event is dispatched to

- `viewArea`: The area that shows the content of a display

  Child elements:

  - `bounds`: The size and location of the view area

  Attributes:

  - `touchable`: Flag which determines if touch events are dispatched to the display
  - `keyInput`: Flag which determines if key events are dispatched to the display
  - `displayID`: The ID of the display the content of which is displayed

Example 6.1.
Control panel XML file

```
<?xml version="1.0" ?>

<panel
  xmlns="http://www.elektrobit.com/gtf/monitor/view/config"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

    <button>
        <bounds>
            <x>5</x>
            <y>5</y>
            <width>80</width>
            <height>30</height>
        </bounds>

        <text>Mute</text>
        <onRelease>Events.send("VolumeMute");</onRelease>
    </button>

    <rotary displayID="65278">
        <bounds>
            <x>5</x>
            <y>120</y>
            <width>80</width>
            <height>80</height>
        </bounds>

        <stepSize>10</stepSize>
    </rotary>

    <viewArea touchable="true" keyInput="true" displayID="65278">
        <bounds>
            <x>100</x>
            <y>5</y>
            <width>480</width>
            <height>240</height>
        </bounds>
    </viewArea>
</panel>
```

Alternatively, the `Events` class allows sending key input, touch input, or rotary input events as well as events with parameters. To replace the `VolumeMute` event by the `key_phone_pressed` event the button code looks like this:

```
  <button>
        <bounds>
            <x>5</x>
            <y>5</y>
            <width>80</width>
            <height>30</height>
        </bounds>

        <text>PHONE</text>
        <onPress>Events.sendKeyPress(65278, 275);
        </onPress>
    </button>
```

To send the `key_phone_released` event as an event with parameters, the button code looks like this:

```
  <button>
        <bounds>
            <x>5</x>
            <y>5</y>
            <width>80</width>
            <height>30</height>
        </bounds>

        <text>PHONE</text>
        <onRelease>Events.send("key_phone_released",  [65278,
        275, 0, 0]);</onRelease>
    </button>
```

6.2.2.2. Application script objects

EB GUIDE Monitor uses the Mozilla JavaScript engine Rhino which is included in the Oracle JDKs. For more information on using JavaScript and Rhino see http://www.mozilla.org/rhino/.

To simplify scripting, all methods related to EB GUIDE TF are available through the following objects that are globally accessible to every JavaScript.

- Datapool
    - Reads and writes datapool items
    - Looks up IDs by name and names by ID
    - Executes functions such as `commit()` or `switchLanguage()`
- Events
    - Fires events
    - Looks up IDs by name and names by ID
    - Registers events and reacts or runs on command
- StateMachine
    - Jumps to a state
    - Looks up IDs by name and names by ID
    - Listens to state changes and executes reactions
- Test
    - Tests expressions
    - Compares values
    - Modifies the exit code when EB GUIDE Monitor is executed in bash shell mode

NOTE

Embedded JavaScript help in EB GUIDE Monitor

The JavaScript editor of EB GUIDE Monitor contains a help button that calls up a complete API documentation of these objects.

Example 6.2.
Datapool interaction with JavaScript

```
importPackage(java.lang);
importPackage(com.elektrobit.gtf.monitor.types.values);

var ctx_hmi = Datapool.getContextID("hmi");
var ctx = Datapool.getContextID("pdal1");

var artist       = Datapool.getID("artist");
var album        = Datapool.getID("album");
var tracklist    = Datapool.getID("tracklist");
var currentTrack = Datapool.getID("currentTrack");

var statusLine =  Datapool.getID("statusLine");
var userList = Datapool.getID("userList");
var currentTime = Datapool.getID("currentTime");


var tracks =
[
 "Damage, Inc",
 "The Thing That Should Not Be",
 "Welcome Home (Sanitarium)",
 "Battery",
 "Master Of Puppets",
 "Disposeable Heroes",
 "Lepper Messiah",
 "Orion"
];


// write values to properties that have the context "pdal1" as a writer context
Datapool.writeValue(ctx, artist, "Metallica");
Datapool.writeValue(ctx, album,  "Master of Puppets");

Datapool.clearList(ctx, tracklist);
for(var i = 0; i < tracks.length; ++i)
{
    Datapool.appendListItem(ctx, tracklist, tracks[i]);
}
```

```
var ref = new DPListReferenceConst(new DPReference(tracklist), new Uint32(0));
Datapool.writeValue(ctx, currentTrack, ref);

// read values from properties that have "pdal1" as a reader context
var currentTime = Datapool.readValue("pdal1", "currentTime");
var currentUserId =  Datapool.readValue("pdal1", "currentUserId");
var currentUserName = Datapool.readListItem("pdal1", "userList", currentUserId);

Datapool.writeValue(ctx, statusLine, currentUserName + " " + currentTime );

Datapool.commit(ctx);
```

Example 6.3.
Sending and receiving events with JavaScript

```
importPackage(java.lang);
importPackage(com.elektrobit.gtf.monitor.types.values);
importPackage(com.elektrobit.gtf.monitor.event.remote);

listenerImpl = {
    eventOccurred: function(e) {
        // 'e' is of type com.elektrobit.gtf.monitor.event.remote.GtfEvent.
        // See the Javadoc of this class for more detailed information on
        // how to retrieve information from the event object.
        println("Received event: "
                + e.getGroupId() + " , "
                + e.getMessageId() + " ) " );

        var params = e.getParameters();
        for(i = 0; i < params.size(); ++i) {
            param = params.get(i);
            println("Parameter " + param.getName() + ": " + param.getValue());
        }
    },
    // the equals function is important for automatic listener deregistration
    equals: function(other) {
        return true;
    }
}

Events.addListener(new GtfEventListener(listenerImpl));
Events.send("someEvent");
// Fire the event with the name 'someOtherEvent' and one integer parameter
Events.send("someOtherEvent", [new Int32(5)]);
```

```
Thread.sleep(1000);
```

6.2.2.3. Communication with the target

EB GUIDE Monitor communicates with a running EB GUIDE TF instance through a TCP/IP connection. The connection is implemented in the TCP plugin for EB GUIDE Monitor and in the *GtfService* module for the target framework. The TCP/IP connection is split into several virtual channels. Each channel is used by a different interface in the EB GUIDE Monitor application.

6.2.2.4. Command line mode

It is possible to execute EB GUIDE Monitor in command line mode. When run in command line mode no pop-up window opens up. In command line mode, EB GUIDE Monitor starts, loads its plugins, executes actions specified in the command line, and exits the program.

EB GUIDE Monitor supports the following set of command line options to control the behavior of the application from the command line.

-consoleMode
: starts EB GUIDE Monitor in command line mode.

-model <path to monitor.cfg>
: loads the EB GUIDE Monitor configuration from the specified file.

-userSettings <file name>
: stores EB GUIDE Monitor user settings and allows loading them. The name is specified without file name extension. If you do not specify it, the default value is `monitor` and thus creates file `monitor.xml`.

-plugin <plugin class name>
: enables the specified plugin on start-up. Uses the full name of the plugin class.

-connect
: requires an enabled TCP plugin. If a connection configuration is available, it automatically connects to the EB GUIDE TF process on start-up.

-tcp <host>:<post>
: requires an enabled TCP plugin. Creates a TCP connection configuration based on the specified host and port.

-script <file name>
: requires an enabled scripting plugin. Executes the script file with the file name specified.

Example 6.4.

Command line

The following command line loads a configuration file, enables the TCP and scripting plugins, connects to the target, and executes a script.

```
monitor.bat -consoleMode
            -model C:\MyModel\monitor.cfg
            -plugin com.elektrobit.gtf.monitor.tcp.TCPPlugin
            -plugin com.elektrobit.gtf.monitor.scripting.ScriptingPlugin
            -connect -tcp localhost:5456
            -script C:\MyModel\testcase.js
```

6.2.3. Configuration of touch screen devices

Depending on the target platform, it is be necessary to configure the touch screen device so that EB GUIDE TF can use it.

For the following platforms no touch screen configuration is necessary, because EB GUIDE TF relies on the environment to configure the touch screen.

- Windows PC
- Linux X11

For other platforms, you configure touch screens in the `gtfStartup.cfg` file. The corresponding entry has the following form:

```
STARTUP 0 MESSAGE 550
UINT32 touchScreenType
UINT32 sceneId
STRBUF devicePath
UINT32 touchDeviceId
UINT32 outputWidth
UINT32 outputHeight
UINT32 Xright
UINT32 Xleft
UINT32 Ybottom
UINT32 Ytop
```

The *touchScreenType* value specifies the type of the touch screen device and is an indication of whether multitouch input is supported. For a list of supported values see section 9.6, “Touch screen types supported by EB GUIDE GTF”.

The *sceneId* value links the touch screen to a given scene: input events of the device are dispatched into the scene with the given identifier.

The *devicePath* value specifies which physical device is accessed. The format of this name depends on the platform.

The remaining parameters are used to transform the raw device coordinates into the coordinates used by EB GUIDE TF. Their values must be determined during touch screen calibration.

6.2.4. EB GUIDE TF and C++ exceptions

EB GUIDE TF is designed and built without support for C++ exceptions.

If your own C++ code uses exceptions it is your responsibility to ensure that your code is combined with EB GUIDE TF and its libraries in a way that is safe for your system. Not following this rule can lead to crashes for which the root cause is difficult to find.

If your system includes C++ standard libraries, only libraries containing C++ code without exceptions are allowed to be loaded or linked into the EB GUIDE TF process. Make sure that the full dependency of all libraries is adhered to.

For example, on QNX systems you are not allowed to load libraries that link against the `libcpp.so` library into the EB GUIDE TF process, because the EB GUIDE TF process uses libraries that link against the `libcpp-ne.so` library. `libcpp.so` is a C++ standard library with exceptions, whereas `libcpp-ne.so` is a C++ standard library without exception.

6.2.5. EB GUIDE TF and POSIX signals

POSIX signals may interrupt system calls. EB GUIDE TF does not support error handling for interrupted system calls on POSIX platforms.

EB GUIDE TF does not use POSIX signals, but user applications possibly do. Therefore the following POSIX signals are blocked in all EB GUIDE TF threads:

- `SIGALRM`
- `SIGCHLD`
- `SIGUSR1`
- `SIGUSR2`

WARNING ⚠ **POSIX signals**

If EB GUIDE TF threads or user applications do not block POSIX signals while calling EB GUIDE TF API methods, POSIX signals lead to undefined EB GUIDE TF behavior.

6.2.6. Linking EB GUIDE TF statically

By default EB GUIDE TF is provided with the `GtfStartup.exe` executable file and shared object libraries for any dedicated platform. For details see section 6.2.10, “Software module structure of EB GUIDE TF”.

However, some systems do not support shared objects but require linking all software modules statically. EB GUIDE TF can be configured as a set of static libraries for these kinds of systems.

If you intend to use a static system, contact the EB GUIDE support, see chapter 3, “Support“.

6.2.7. Message handling

Messages are an asynchronous mechanism for communication between software modules in EB GUIDE TF. Messages transport up to 255 items of scalar data types, for example integers. List data types are not supported.

The EB GUIDE TF message system never drops any message but delivers all messages in exactly the order in which they were sent. However, there is no pre-defined order for delivering one message to different subscribers.

A message has a numeric message ID that addresses the subscribers. The message IDs from 0 to 65535 are reserved for internal use within EB GUIDE TF and the EB GUIDE product line. Customer-specific software modules can use and manage the remaining message ID range.

GtfMessenger is the process-internal message system provided by the GtfPluginLoader. GtfMessenger is thread-safe and available for all software modules running in EB GUIDE TF. It is mainly used for integration and management of modules, for example run level and interface management or run-time environment configuration.

For detailed information about events, refer to section 5.4.6, “Event handling”.

6.2.8. Read-only file system support

A read-only file system (RomFS) is a block-based file system. Its organization structure has less overhead than regular file systems because it has read-only access and omits access right management.

A RomFS has the purpose of overlaying the file system that is provided by an operating system (OS), for example to speed the system up.

A RomFS can also be used to run EB GUIDE TF on embedded systems without OS file system support.

EB GUIDE TF RomFS support is completely implemented in user space and does not depend on any way of the underlying OS.

EB GUIDE TF RomFS support is used as a layer between the following:

1. Interface provided by `GtfFile` and `GtfPath`
2. File system abstractions provided by the underlying OS

With a RomFS container, you are able to overlay a file system with your own container. The access to these overlay containers is completely transparent, thus you do not have to change any EB GUIDE TF modules that use the `GtfFile` and `GtfPath` interface.

NOTE

Use the RomFS container

The RomFS container is the preferred container format although GtfOSAL overlay file system support is designed to allow other container implementations, too. You may use other container implementations for example to evaluate different designs.

6.2.9. Renderers supported in EB GUIDE

The following renderers are available with EB GUIDE:

- The OpenGL ES 2.0 renderer is capable of rendering 2D and 3D widgets and achieves best results for 24-bit images. It needs a graphics processing unit that supports OpenGL ES 2.0. The OpenGL ES 2.0 renderer utilizes a dedicated hardware unit for rendering. With the OpenGL ES 2.0 renderer, it is recommended to optimize image resource files, for example using tools like 'Optipng'.
- The DirectX 11 renderer is capable of rendering 2D and 3D widgets. It needs an operating system that supports DirectX 11. It utilizes a dedicated hardware unit for rendering.
- The OpenVG 1.1 renderer is capable of rendering 2D widgets and vector graphics and achieves best results for 32-bit images. The OpenVG 1.1 renderer needs a graphics processing unit that supports OpenVG 1.1. It utilizes a dedicated hardware unit for rendering. With the OpenVG 1.1 renderer, it is recommended to optimize image resource files.

6.2.10. Software module structure of EB GUIDE TF

EB GUIDE TF consists of several software modules. Depending on the customer project some of them are essential and others are optional. You as a system integrator can add additional project specific modules, for example widget sets or applications, which are not part of EB GUIDE TF.

The default delivery of EB GUIDE TF runs on operating systems that support shared object files, for example Windows 8, Linux or QNX. EB GUIDE TF is divided into the following executable files and libraries to fit most customer projects out of the box:

- GtfStartup.exe

  The executable file which contains platform-specific start-up code and interprets the gtfStartup.cfg configuration file. GtfStartup.exe is configurable with parameters.

- GtfCommon

  Shared object library which contains

  - base classes and an abstraction of the operating system
  - memory management
  - a trace logging system
  - GtfPluginLoader

- GtfCore

  Shared object library which contains all mandatory modules for each GUI project based on EB GUIDE Studio and EB GUIDE TF. Example modules are state machine interpreter, action interpreter, and datapool.

- GtfWidgetSet

  Shared object library which uses the Basic Widget Set and is required for EB GUIDE GTF based GUI projects which are modeled with the Basic Widget Set.

- GtfWidgetSet3D

  Shared object library which uses the 3D Widget Set and is required for EB GUIDE GTF based GUI projects which are modeled with the 3D Widget Set.

- GtfWidgetSetOpenVG

  Shared object library which uses the OpenVG Widget Set and is required for EB GUIDE GTF based GUI projects which are modeled with the OpenVG Widget Set.

- GtfDisplayManager

  Shared object library that is responsible for the connection between a GtfCoreRuntime and a specific renderer.

- GtfOpenGLRenderer

  Shared object library which allows dragging views and widgets using the OpenGL ES 2.0 API.

- GtfDirectXRenderer

  Shared object library which allows dragging views and widgets using the DirectX 11 API.
- GtfOpenGLRenderer

  Shared object library which allows dragging views and widgets using the OpenGL ES 2.0 API.
- GtfOpenVGRenderer

  Shared object library which allows dragging views and widgets using the OpenVG 1.1 API.
- GtfService

  Shared object library which is required to establish TCP connections between EB GUIDE TF and EB GUIDE Studio or EB GUIDE Monitor.
- GtfIPC

  Shared object library which extends GtfService and provides inter-process communication (IPC). Implements a service running in the HMI process. Counterpart to GtfIPCClient.
- GtfIPCClient

  Shared object library which can be used by separate application processes to access the external event system and the datapool. Counterpart to GtfIPC.

6.2.10.1. Run level and interface management

You as a system integrator can influence the start-up and shutdown order of software modules by managing run levels and interfaces. To manage run levels and interfaces, you use system messages.

For example, a system message does one of the following:

- It publishes the current run level and the direction of the boot process which can be start-up, final, or shutdown.
- It is used by software modules for the publication of interface objects that the software modules provide to other software modules.

A receiver of a run level message can perform specific activities for that run level and returns the next required run level. Return values that request a run level which is already passed are ignored.

NOTE

Available run levels

Only the following run levels are guaranteed to be processed:

- Initial run levels
- Final run levels
- Requested run levels

Processing of additional run level information depends on your implementation.

An interface consists of the following:

- a data set that contains a unique identifier
- a version number
- a pointer to an object that provides the interface

A software module can typecast the object pointer after evaluating the unique identifier and the version number. In addition, module interfaces provide the information about the lowest valid run level for the interface object.

Basic framework interfaces which are provided by GtfPluginLoader are available at least until `gtf_destroyModule()` is called.

Modules must not publish their interfaces before receiving the first run level message. If the run level falls below the lowest valid level, multiple or incomplete start-up and shutdown cycles force the software modules to publish their interfaces again.

NOTE

Recommendation

It is recommended for every module to process the lowest valid run level of all used interfaces during shutdown.

The following tables show examples for run level and interface management during start-up and shutdown.

Table 6.3. Run level and interface management during start-up

Software module	Activity
GtfPluginLoader	Loads and initializes modules
Module A	Subscribes to system messages
Module B	Subscribes to system messages
GtfPluginLoader	Sends message GTF_MID_RUN_STARTUP
GtfRunlevelManager	Publishes start-up at run level 0 to all subscribers
Module A	Returns 100, the next required run level

Software module	Activity
Module B	Returns 50, the next required run level
...	...
GtfRunlevelManager	Publishes start-up at run level 50
Module A	▶ Publishes the interface with 100, the lowest valid run level ▶ Returns the current run level 50 to indicate that it is not interested in additional start-up steps
Module B	Returns 100, the next required run level
Module B	Receives the interface of module A
...	...
GtfRunlevelManager	Publishes start-up at run level 100
Module A	Returns the current run level 100 to indicate that it is not interested in additional start-up steps
Module B	▶ Starts using the interface of module A ▶ Returns 200, the next required run level
...	...
GtfRunlevelManager	Publishes start-up at run level 200
Module A	Returns the current run level 200 to indicate that it is not interested in additional start-up steps
Module B	Returns the current run level 200 to indicate that it is not interested in additional start-up steps
...	...
GtfRunlevelManager	Publishes final run level 65535 to all subscribers
Module A	Returns the current run level 65535
Module B	Returns the current run level 65535
...	...

Table 6.4. Run level and interface management during shutdown

Module	Activity
Module B	Sends message `GTF_MID_RUN_SHUTDOWN`
GtfRunlevelManager	Publishes shutdown at run level 65535 to all subscribers
Module A	Returns 99, the next required run level
Module B	Returns 100, the next required run level
...	...

Module	Activity
GtfRunlevelManager	Publishes shutdown at run level 100
Module A	Returns 99, the next required run level
Module B	▶ Stops using the interface of module A ▶ Returns the current run level 100 to indicate that it is not interested in additional shutdown steps
...	...
GtfRunlevelManager	Publishes shutdown at run level 99
Module A	Returns the current run level 99 to indicate that it is not interested in additional shutdown steps
Module B	Returns the current run level 99 to indicate that it is not interested in additional shutdown steps
...	...
GtfRunlevelManager	Publishes final run level 0 to all subscribers
Module A	▶ Unsubscribes to system messages ▶ Returns the current run level 0
Module B	▶ Unsubscribes to system messages ▶ Returns the current run level 0
...	...
GtfRunlevelManager	Triggers the complete system shutdown by sending `GTF_MID_SYSTEM_EXIT` to `GtfPluginLoader`
GtfPluginLoader	Unloads the modules and returns to the caller

Besides the return value during run level handling, a module can use the following messages to control GtfRunlevelManager:

- Trigger a system shutdown by sending a `GTF_MID_RUN_SHUTDOWN` message.
- Lock the current run level by sending a `GTF_MID_RUN_RUNLEVEL_LOCK` message. Locking the run level is useful for pending asynchronous activities during start-up or shutdown.
- Release a locked run level by sending an `GTF_MID_RUN_RUNLEVEL_UNLOCK` message. Releasing a locked run level is useful as soon as possible after the pending asynchronous activity finishes.

The current run level is locked while the number of `LOCK` messages received by GtfRunlevelManager is higher than the number of `UNLOCK` messages.

6.2.11. The gtfStartup.cfg configuration file

The gtfStartup.cfg configuration file contains rules that describe how to map signals to actions. Signals can be run level changes, actions can be module loading or sending pre-defined messages file. After evaluating the command line parameters, the GtfStartup.exe file reads the configuration file.

The configuration file is a line-oriented text file encoded with UTF8. It can be edited with any text editor that supports UTF8 character encoding. Both DOS line endings and Unix line endings are allowed. One line describes one mapping rule. Multi-line rules or multiple rules in one line are not supported. It is possible to trigger multiple actions for one signal by multiple mapping rules for the signal.

TIP

Text editors without UTF8 character encoding support

The first 128 ASCII characters (0-127) are compliant with the UTF8 standard. Thus, if the gtfStartup.cfg file does not contain any characters with Unicode code points above 127, it is possible to edit the file with text editors without UTF8 character encoding.

6.2.11.1. Mapping rule structure

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

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

Table 6.5. Escape sequences used to enter special characters

Escape sequence	Special character
\n	line feed
\r	carriage return
\\	\
\"	"
\t	TAB

6.2.11.2. Signals

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

Table 6.6. Signals

Keyword	Parameter	Description
INIT	none	Loading gtfStartup.cfg is finished.

Keyword	Parameter	Description
STARTUP	<run level>	<run level> (0...0xFFFF) is reached during start-up.

6.2.11.3. Actions

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

Table 6.7. Actions

Keyword	First parameter	Following parameters	Description
MESSAGE	<MsgID>	message parameters	Sends a message with the <MsgID> (0...0xFFFFFFFF) and message parameters specified.
LOAD (supported by INIT signal only)	FW_PATH or MODEL_PATH	<file path>	Loads a plugin file and initializes the included modules. FW_PATH means a path relative to the GtfStartup.exe executable file. MODEL_PATH means a path relative to the `gtfStartup.cfg` file. For absolute paths the key words FW_PATH and MODEL_PATH lead to the same result.
LOAD_ALL (supported by INIT signal only)	FW_PATH or MODEL_PATH	<directory path>	Loads all plugin files in the directory and initializes the included modules. FW_PATH means a path relative to the GtfStartup.exe executable file. MODEL_PATH means a path relative to the gtfStartup.cfg file. For absolute paths the

Keyword	First parameter	Following parameters	Description
			key words FW_PATH and MODEL_PATH lead to the same result.

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

Table 6.8. Message parameters

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

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

TIP

Working with message IDs

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

```
#define GTF_MID_RANGE_GTF_DISPLAY 400
```

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

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

NOTE

Predefined messages in EB GUIDE TF

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

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

6.2.11.4. Execution order of mapping rules

Mapping rules in EB GUIDE TF are executed in the following order:

1. If the gtfStartup.cfg file has been parsed successfully, the INIT signal is triggered.
2. If EB GUIDE TF has entered the run level during system start-up, the STARTUP signal is triggered.
3. If a signal contains several mapping rules, the rules are executed in the order in which they are defined in the gtfStartup.cfg file.
4. If EB GUIDE TF has entered the run level during system shutdown, the SHUTDOWN signal is triggered.

6.2.11.5. Example of a gtfStartup.cfg file

The following example is intended to show the syntax of a typical gtfStartup.cfg file. It is not intended to be copied into your project because, taken out of its context, it will not work.

Example 6.5.
gtfStartup.cfg file

```
//init - load all modules in the framework path
INIT LOAD_ALL FW_PATH ""

//init - load specific modules relatively to the model directory
INIT LOAD MODEL_PATH "MyExampleA"
INIT LOAD MODEL_PATH "MyExampleB"

// startup - runlevel 499 - configure the example module
```

```
STARTUP 499 MESSAGE 65536 STRBUF "MyExample/fileA.bin"

// shutdown - runlevel 1 - configure the example module
SHUTDOWN 1 MESSAGE 0x10002 INT32 4711 FW_PATH "MyExample/fileB.bin" STRING "Hi"
```

6.2.12. The GtfStartup.exe executable file

The GtfStartup.exe executable file provides platform-specific start-up code and interprets the gtfStartup.cfg configuration file. Additional functionality is available for specific platforms, for example command line parameter handling or detection of other EB GUIDE TF instances.

6.2.12.1. Command line parameters

If you specify one single command line parameter, it is interpreted as file path of the configuration file. If you do not provide any command line parameter, the start-up code looks for the file gtfStartup.cfg in the directory in which the GtfStartup.exe executable file is located.

- `<gtfStartupConfigurationFile>` or

  `--startup-cfg <gtfStartupConfigurationFile>`:

  Optional parameter. If specified, the file `gtfStartupConfigurationFile` is loaded and parsed as start-up configuration. Otherwise, `gtfStartup.cfg` is used by default.

- `--debug`: Optional parameter. If specified, the contents of error logs and traces are optimized for debugging without EB GUIDE Studio. Otherwise, everything is optimized for display in EB GUIDE Studio.

- `--monitor`: Optional parameter. If specified, EB GUIDE TF synchronizes start-up with EB GUIDE Monitor. The `--monitor` parameter is intended for internal EB GUIDE use-cases, for example simulation mode in EB GUIDE Studio.

- `--report`: Optional parameter. If specified, EB GUIDE TF uses a buffer to avoid missing debug messages, error logs, and traces at start-up. Losing the connection to the GtfReport service triggers a shutdown of EB GUIDE TF. The `--report` parameter is intended for internal EB GUIDE use-cases, for example simulation mode in EB GUIDE Studio.

- `--romfs <romFileSystemFile>`: Optional parameter. If specified, the given ROM file system (RomFS) is loaded. It is possible to specify multiple RomFS. Configuration files inside RomFS containers are supported. For example, the path to the `gtfStartupConfigurationFile` can refer to a file in the RomFS `romFileSystemFile`.

- `--remotefb`: Optional parameter. If specified, EB GUIDE TF waits during start-up until a remote framebuffer client connects. The `--remotefb` parameter is intended for internal EB GUIDE use-cases, for example simulation mode in EB GUIDE Studio.

- `--version`: Optional parameter. If specified, EB GUIDE TF displays the version of the run-time on shut down. The displayed version matches the version that is shown in EB GUIDE Studio

6.2.12.2. Single instance detection on Windows platforms

The Microsoft Windows concept of named events is used for optional detection of other EB GUIDE TF instances. Single instance detection works as follows.

1. Configure the message `GTF_MID_SYSTEM_SINGLE_INSTANCE_CONFIG` in the gtfStartup.cfg file to enable a named event. See GtfMessageId.h for details.
2. The first EB GUIDE TF instance is started using the gtfStartup.cfg file. The configured message enables the named event. The event checks that no instance is running yet. The instance observes the event.
3. As soon as a second EB GUIDE TF instance is started using the gtfStartup.cfg file, it triggers the named event. The first EB GUIDE TF instance detects that and sends the message `GTF_MID_SYSTEM_SECOND_INSTANCE_TRIGGER` through the `GtfMessenger`. `GTF_MID_SYSTEM_SECOND_INSTANCE_TRIGGER` can be observed by an application and used to react to the start of the second instance, for example by setting the focus to its own window.
4. If the second EB GUIDE TF instance detects that the named event already exists in another instance, it triggers the event and immediately shuts down the framework.

6.3. Configuring profiles

EB GUIDE Studio offers the possibility to create different profiles for a project. Project profiles write the EB GUIDE TF start-up configuration file gtfStartup.cfg.

You use profiles to do the following:

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

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

6.3.1. 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
Rename the profile to `MySimulation`.

Step 5
Select the radio button **Use for simulation**.

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

6.3.2. Adding a library

Adding a library

The default delivery of EB GUIDE TF runs on operating systems that support shared object files, for example Windows 8, Linux or QNX. EB GUIDE TF is divided into executable files and libraries to fit most customer projects out of the box.

For details see section 6.2.10, “Software module structure of EB GUIDE TF”.

Prerequisite:

- The project center is displayed.
- In the navigation area the tab **CONFIGURE** > **Profiles** is selected.
- A profile `MySimulation` is added.
- A library `MyLibraryA` is available in `$GUIDE_PROJECT_PATH\ressources`.

Step 1
In the content area, select the `MySimulation` profile.

Step 2
Click ▶ to expand the libraries.

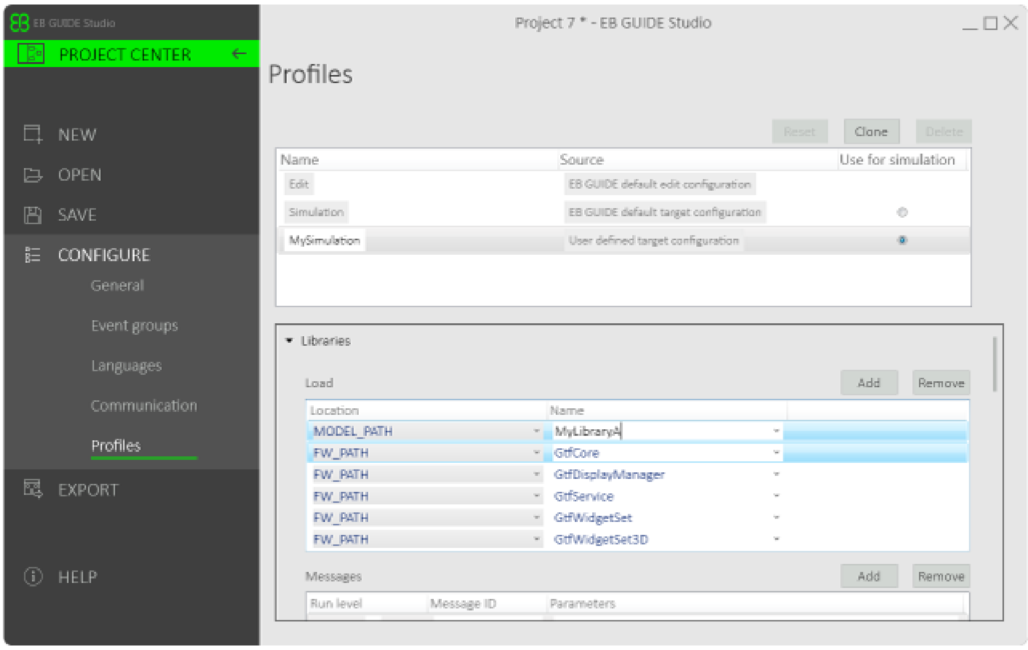
The **Load** table with all included libraries is displayed.

Step 3
Click **Add**.

A new row is added to the table.

Step 4
In the table select `MODEL_PATH` from the drop-down list box under **Location**.

Step 5
Enter `MyLibraryA` in the **Name** text box.



Name	Source	Use for simulation
Edit	EB GUIDE default edit configuration	
Simulation	EB GUIDE default target configuration	○
MySimulation	User defined target configuration	●

Location	Name
MODEL_PATH	MyLibraryA
FW_PATH	GtfCore
FW_PATH	GtfDisplayManager
FW_PATH	GtfService
FW_PATH	GtfWidgetSet
FW_PATH	GtfWidgetSet3D

Run level	Message ID	Parameters

Figure 6.3. Table of libraries

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

6.3.3. Adding messages

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

For details see section 6.2.10.1, “Run level and interface management”.

NOTE ⓘ

Predefined messages in EB GUIDE TF

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

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

Message IDs and parameters of pre-defined messages are documented in the GtfMessageId.h file.

Adding messages

Prerequisite:

- The project center is displayed.
- In the navigation area the tab **CONFIGURE** > **Profiles** is selected.

Step 1
In the content area, select a profile.

Step 2
Click ▶ to expand the libraries.

Step 3
The **Messages** table with all included libraries is displayed.

Step 4
Click **Add**.

A new row is added to the table.

Step 5
Enter `0` in **Run level** text box.

Step 6
Enter `300` in **Message ID** text box.

Step 7
Enter `UINT32 0xDEADBEAF` in the **Parameter** text box.

You added a system message.

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

6.3.4. Configuring a display

Configure each display of your EB GUIDE model as follows.

Configuring a display

Prerequisite:

- The project center is displayed.
- In the navigation area the tab **CONFIGURE** > **Profiles** is selected.

Step 1
Click ▶ to expand the scenes.

Step 2
Select a display from the **State machines** drop-down list box, for example **Main**.

Step 3
Adjust the properties for the display. For information on each property see section 9.5, “Scenes”.

6.4. Configuring the system start

6.4.1. Configuring the system start for operating systems that support shared object files

Configuring the system start for operating systems that support shared object files

Step 1
Adjust the gtfStartup.cfg file for your project. For instructions see section 6.4.2, “Configuring the gtfStartup.cfg file”.

Step 2
Export your project. For instructions see section 5.8.5, “Exporting a project”.

Step 3
Copy the following files to the target system:

- The EB GUIDE GTF version for your platform. This includes the executable file and all plugin files that are required by your gtfStartup.cfg configuration.
- Your adjusted gtfStartup.cfg.
- The exported EB GUIDE model. Make sure the paths in gtfStartup.cfg refer to the EB GUIDE model's files and its relative paths are correct.

Step 4

Start EB GUIDE GTF on the target system.

Use a command line that enables you to type the commands that are suitable for your operating system.

You have configured the operating system on the target framework.

6.4.2. Configuring the gtfStartup.cfg file

Configuring the gtfStartup.cfg file

Look up functions names listed in this instruction in the GtfMessageId.h file.

Step 1

Load plugin files:

To define the files that contain the required shared objects, add the following example messages to the gtfStartup.cfg file:

```
INIT LOAD FW_PATH "GtfModelConverter"
INIT LOAD FW_PATH "GtfCore"
INIT LOAD FW_PATH "GtfDisplayManager"
INIT LOAD FW_PATH "GtfService"
INIT LOAD FW_PATH "GtfWidgetSet"
INIT LOAD FW_PATH "GtfOpenGLRenderer"
```

Files can differ depending on the operating system.

Step 2

Configure the type manager:

To inform EB GUIDE GTF where to find the binary types file, add the following message to the gtfStartup.cfg file:

```
STARTUP 0 MESSAGE 317 MODEL_PATH "types.bin"
```

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

Step 3

Create a GtfCoreModel:

To be able to display and execute an EB GUIDE model, add the following message to the gtfStartup.cfg file:

```
STARTUP 0 MESSAGE 300 UINT32 0xDEADBEAF
```

This message (`GTF_MID_GTF_CORE_CREATE_MODEL`) makes EB GUIDE GTF create a GtfCoreModel with the ID 0xDEADBEAF. The ID has to be unique. It is used in the following steps to load the model.

Step 4
Create a GtfCoreRuntime:

To connect a GtfStateMachine to a GtfCoreModel, add the following message to the gtfStartup.cfg file:

```
STARTUP 0 MESSAGE 306 UINT32 0xDEADBEAF UINT8 0
```

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

NOTE

Context ID

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

Step 5
Create a GtfViewFactory:

To define how to retrieve view descriptions, add the following message to the gtfStartup.cfg file:

```
STARTUP 403 MESSAGE 450 UINT32 0xBAADF00D MODEL_PATH "views.bin"
```

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

Step 6
Configure the .bdf input file:

To make the GtfCoreModel load the binary declaration file of the datapool, add the following message to the gtfStartup.cfg file:

```
STARTUP 499 MESSAGE 308 UINT32 0xDEADBEAF MODEL_PATH "datapool.bdf"
```

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

Step 7
Configure the .bvf input file:

To make the GtfCoreModel load the binary value file of the datapool, add the following message to the gtfStartup.cfg file:

```
STARTUP 499 MESSAGE 309 UINT32 0xDEADBEAF MODEL_PATH "datapool.bvf"
```

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

Step 8
Configure the state machine file:

To load the binary state machine file, add the following message to the gtfStartup.cfg file:

```
STARTUP 499 MESSAGE 311 UINT32 0xDEADBEAF MODEL_PATH "model.bin"
```

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

Step 9
Enable state machines:

To enable a state machine, add the following message to the gtfStartup.cfg file:

```
STARTUP 501 MESSAGE 320 UINT32 0xDEADBEAF STRBUF "Main"
```

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

Step 10
Configure displays:

To configure a display option, add the following messages to the gtfStartup.cfg file:

```
STARTUP 0 MESSAGE 511 STRBUF "Main" STRBUF "windowCaption" STRBUF "My Model"
STARTUP 0 MESSAGE 512 STRBUF "Main" STRBUF "hwLayerId" INT32 0
STARTUP 0 MESSAGE 513 STRBUF "Main" STRBUF "showWindowFrame" UINT8 1
```

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

Step 11
Configure resource configuration file:

To load the binary resource configuration file, add the following message to the gtfStartup.cfg file:

```
STARTUP 499 MESSAGE 312 UINT32 0xDEADBEAF MODEL_PATH "resources.bin"
```

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

Step 12
Configure the debug database (optional):

If you want to run an EB GUIDE model in debug mode to receive error messages in more detail, add the following message to the gtfStartup.cfg file:

```
STARTUP 499 MESSAGE 318 UINT32 0xDEADBEAF MODEL_PATH "debug.bin"
```

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

Step 13
Configure the service mapper TCP/IP port:

To be able to use EB GUIDE Monitor or a similar client, add the following message to the gtfStartup.cfg file:

```
STARTUP 0 MESSAGE 305 UINT16 5456
```

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

Step 14
Load a RomFS container:

To load a RomFS container, add the following message to the gtfStartup.cfg file:

```
STARTUP 0 MESSAGE 701 MODEL_PATH "container.romfs"
```

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

Step 15
Configure how font files are accessed by EB GUIDE GTF (optional):

To configure how font files are accessed, add the following message to the gtfStartup.cfg file:

```
STARTUP 0 MESSAGE 510 UINT8 1
```

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

NOTE

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

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

Step 16
Disable the output of EB GUIDE Script trace functions:

To suppress the output of `f:trace_string()` and `f:trace_dp()` in EB GUIDE Script code, add the following message to the gtfStartup.cfg file:

```
STARTUP 0 MESSAGE 321 UINT32 0xDEADBEAF
```

Step 17
Debug monitoring information:

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

Step 18
Configure FreeType Cache (optional):

To configure the FreeType cache, add the following message to the gtfStartup.cfg file:

```
STARTUP 0 MESSAGE 321 UINT32 0xDEADBEAF
```

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

- `max_faces`: 0
- `max_sizes`:0
- `max_bytes`: 0 kB

Due to the way EB GUIDE GTF handles font sizes, `ft_size` objects are not cached separately from `ft_face` objects at the moment. It is recommended to use meaningful values for `max_sizes`. This behavior may change in future versions of EB GUIDE GTF.

Step 19
Configure the resource cache:

For each display ID, it is possible to configure a resource cache which caches textures. Add the following message to the gtfStartup.cfg file:

```
STARTUP 0 MESSAGE 520 UINT32 61441 UINT32 1048576
```

This message (`GTF_MID_GTF_RENDERER_TEXTURE_CACHE`) creates a resource cache for the default display ID (61441) with a size of 1048576 bytes. Add the message several times to configure more display IDs. Assigning display ID 0 configures all display IDs which are not configured otherwise to use the same resource cache.

6.5. Evaluating memory usage

Evaluating memory usage helps you to debug the system and the EB GUIDE model. During run-time, EB GUIDE GTF can continuously print information about memory that the framework manages dynamically.

Configuring a memory report

You configure a memory report by adding a configuration message to the gtfStartup.cfg configuration file.

Prerequisite:

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

Step 1
In the navigation area, click **CONFIGURE** > **Profiles**.

Step 2
Select the **Simulation** profile.

Step 3
Click ▶ to expand the libraries.

Step 4
Next to **Messages** click **Add**.

Step 5
For the new message, enter the following in the **Parameters** text box:

```
STARTUP 0 MESSAGE 12 UINT32 5000
```

The message `GTF_MID_SYSTEM_REPORT_MEMORY` activates memory reporting with a delay of 5000 ms between each report.

6.6. Creating a read-only file system (RomFS) container

Creating a read-only file system (RomFS) container

The directory you create serves as root directory in the RomFS. It is referred to as `"/"` on POSIX platforms and as `"C:\"` on Microsoft Windows platforms.

Step 1
Create a directory structure and files in a local working directory.

Step 2
Locate the command line tool GtfRomFsCreate in the `tools\GtfRomFsTools` sub-directory of your EB GUIDE GTF SDK directory.

Step 3
Run GtfRomFsCreate without any parameters.

The following usage directions are displayed:

```
Usage: GtfRomFsCreate.exe [OPTIONS] DIRECTORY [IMAGE_NAME]

Create a read only filesystem container from a directory

Options are:
```

```
--create-c-file BASE_NAME  Create a C file that contains your data,
                           and a coresponding header defining it.
                           The files created are BASE_NAME.c and BASE_NAME.h
                           IMAGE_NAME is used as the identifier of the RomFS
                           binary blob
--output-dir OUTPUT_DIR    Specify the output directory for the generated files.
                           This directory must exist.
--max-size N               Specify a maximum size for your container
-h or --help               Display this help
```

For usage options see the list below.

You have the following options:

Create a RomFS container

`GtfRomFsCreate.exe romfs_root_directory` creates the file `romfs_root_directory.romfs`. This file contains `romfs_root_directory`.

Create a RomFS container and specify the name of the resulting file

`GtfRomFsCreate.exe romfs_root_directory image` creates the file `image.romfs`. This file contains `romfs_root_directory`.

Limit the size of the resulting container

Specify `--max-size N` on the command line. If the size limit you specify is exceeded, `GtfRomFsCreate` emits an error message and stops putting files into the container. The maximum size `max-size` is defined in bytes.

Create a RomFS container and put it, ready to use, in a C-array

`GtfRomFsCreate.exe romfs_root_directory --create-c-file c_array` creates the file `romfs_root_directory.romfs`. This file contains `romfs_root_directory` .

Content is put in the file `c_array.c` as `const unsigned char romfs_root_directory[] = "...";`. `"..."` is the content of the container encoded in C hexadecimal literals.

Additionally a `c_array.h` header file is created. The header file has an `extern const unsigned char romfs_root_directory[N];` forward declaration which you can include and use in your code.

The `--max-size N` parameter is respected.

Create a RomFS container, specify the name of the resulting file and put it, ready to use, in a C-array

`GtfRomFsCreate.exe romfs_root_directory image --create-c-file c_array` creates the file `image.romfs`. This file contains `romfs_root_directory`. Content is put in file `c_array.c` as `const unsigned char romfs_root_directory[] = "...";`.

`"..."` is the content of the container encoded in C heximal literals.

Additionally a `c_array.h` header file is created. This header file has an `extern const unsigned char romfs_root_directory[N];` forward declaration, which you can include and use in your code.

The `--max-size N` parameter is respected.

6.7. Starting and connecting EB GUIDE Monitor

EB GUIDE Monitor communicates with an EB GUIDE GTF instance using a TCP/IP connection. Therefore it is necessary to configure EB GUIDE Monitor before you can use it to an EB GUIDE model.

Starting EB GUIDE Monitor

Step 1
Go to `$GUIDE_INSTALL_PATH/tools/monitor` and double-click the `monitor.bat` file.

EB GUIDE Monitor opens.

Step 2
If this is the first time you start EB GUIDE Monitor, do the following:

Step 2.1
In the **Plugins** menu, click **Plugin list...**.

A dialog opens.

Step 2.2
In the **Plugin list** dialog, select the plugins you want to use.

Step 2.3
Close the **Plugin list** dialog.

Connecting EB GUIDE Monitor

Prerequisite:

- EB GUIDE Monitor is opened.

Step 1
In the **File** menu, click **Open configuration...**.

A dialog opens.

Step 2
Open the directory that contains an exported EB GUIDE model.

Step 3
Select the `monitor.cfg` file.

Step 4
Click **Open**.

Step 5
Click **Configure connection...**.

A dialog opens.

Step 6
Enter the host address and port of the EB GUIDE GTF process you want to connect to. If EB GUIDE GTF is running on your PC, the host address is `localhost`. The port number is the service mapper port specified in gtfStartup.cfg.

Step 7
Click **OK**.

Step 8
With EB GUIDE GTF running, click **Connect** in the toolbar.

6.8. Using and creating an Android APK for EB GUIDE TF

For background information on Android APK see section 6.2.1, “Android APK”.

For more information on Android setup, APK creation or the Android toolchain, refer to the official Android documentation.

As the basic concepts and approaches known for other platforms are also valid for the Android platform, the following sections focus on the topics that are specific for Android.

6.8.1. Executing an exported EB GUIDE model on Android

Executing an exported EB GUIDE model on Android

To execute an exported EB GUIDE model on Android, you install the EB GUIDE Model Chooser and EB GUIDE Launcher. The EB GUIDE Model Chooser provides a user interface to select exported EB GUIDE models. Selecting an exported EB GUIDE model starts EB GUIDE Launcher. The EB GUIDE Launcher is the framework to execute an exported EB GUIDE model on the Android device.

Prerequisite:

- To install the two applications on the Android device, enable your system to install from a different source than the Android Play Store. On your Android device select the **Settings** > **Security** > **Unknown sources** option.

Step 1
Copy `EB Guide Launcher.apk` and `EB Guide Model Chooser.apk` from the `$GUIDE_INSTALL_PATH/platform/android/bin/` directory to your Android device or to the external storage of your Android device.

Step 2
Open a file manager and navigate to the copied files.

Step 3
Install `EB Guide Launcher.apk` and `EB Guide Model Chooser.apk`.

Step 4
Export an EB GUIDE model. For instructions see section 5.8.5, “Exporting a project”.

Step 5
Copy the whole directory that was exported by EB GUIDE Studio to your Android device. For information where to store the models see section 9.1.2, “File path for models”.

Step 6
To execute the EB GUIDE model on your Android device, open `EB Guide Model Chooser.apk` and select an EB GUIDE model from the list. The `EB Guide Launcher.apk` is started automatically with the selected EB GUIDE model.

The EB GUIDE model is executed on your Android device.

6.8.2. Creating your own Android APK using the template

Creating your own Android APK using the template

Step 1
Import the project `$GUIDE_INSTALL_PATH/platform/android/apk/GtfAndroidAppTemplate` into Eclipse or IntelliJ.

Step 2
Optional: Modify the location of the model and the native libraries by editing the implementation of the template `TemplateActivity.java`.

The template activity is the main activity of your custom application.

Step 3
Copy the Android SDK binaries delivered with EB GUIDE GTF to the directory `$GUIDE_INSTALL_PATH/platform/android/apk/libs/armeabi`.

Step 4
Copy an EB GUIDE model to the default external files directory of the application. The default location implemented in the template activity is `/data/android/com.elektrobit.gtf_android_template.package`.

Step 5
Deploy and launch your application in Eclipse or IntelliJ on your target device or using an Android virtual device (AVD).

The EB GUIDE model is now executed on your Android device. Customize the application according to your requirements.

6.8.3. Creating your own Android APK from scratch

The APK files installed with the Android SDK of EB GUIDE TF are suitable for most use cases. If they are not sufficient, use the APK template (see section 6.8, “Using and creating an Android APK for EB GUIDE TF”). You can integrate additional EB GUIDE TF plugins that are useful for a project. Save the additional plugins in the directory of the exported EB GUIDE model and include them in the start-up configuration file. All run-time dependencies are resolved by EB GUIDE TF.

For background information on the custom APK, see section 6.2.1.5, “Released APK and custom APK”.

Creating your own Android APK from scratch

Step 1
Create an Android project. Use either the Eclipse ADT plugin or create it with the provided Ant tooling.

Step 2
Create a `libs` directory where the delivered SDK binaries are stored.

Step 3
Copy only the `.so` files into the directory.

Step 4
Create an activity that derives from the `GtfNativeActivity` class.

Step 5
Optional: Add the android-dl loading code to the `onCreate` method to ease `.so` dependency handling. If no android-dl is available, add the necessary `System.loadLibrary` call for every `.so` file instead.

Step 6
Add a `System.loadLibrary` call to the `onCreate` method for `libGtfAndroid.so` and `System.loadLibrary` calls for every EB GUIDE GTF SDK `.so` file.

Alternatively load only the `GtfNativeActivity` using `AndroidDL.loadLibrary()` command.

Step 7
Specify the `GtfNativeActivity` as library name in the `EB GUIDE Launcher.apk` manifest as follows:

```
<meta-data
                        android:name="android.app.lib_name"
                        android:value="GtfNativeActivity" />
```

Refer to the `NativeActivity` example in the Android NDK installation directory.

Step 8
Make the following modifications:

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
 <uses-permission android:name="android.permission.INTERNET" />


<uses-sdk android:minSdkVersion="10"/>
```

Step 9
Create a keystore file for a release build.

Step 10
Create the release build with Ant on the command line.

TIP

Debug builds

Test and create debug builds within Eclipse. The Eclipse plugin takes care of the whole APK build process, for example debug keystore.

TIP

Sub-projects

Use sub-projects for a clean separation of functionality. Place the `GtfNativeActivity` in a master project and divide individual functionality into sub-projects. The master project references the other components as sub-project dependencies and the Android build performs the necessary integration steps.

7. Application developer's manual

7.1. Overview

As an application developer you are the target audience for the following chapters.

For more information, see section 1.1.3, “Application developers”.

For more information on the structure of the manual, see section 1.2, “Structure of user documentation”.

7.2. Interaction between HMI and applications

In most cases, the HMI interacts with at least one project specific application, for example a media player. Asynchronous communication allows better separation of software modules and helps to reduce mutual timing impacts.

To establish an asynchronous communication between the generated EB GUIDE model and the dedicated application you have the following options:

- External event system
- Datapool

TIP

Direction of communication

Events are the advised mechanism if the HMI needs to trigger asynchronous application activities, for example `play next track`.

Datapool items are the advised mechanism if the application needs to provide information asynchronously to the HMI, for example title list of a media player.

The external event system and the datapool provide three types of property access methods:

- The first group of methods expects parameters which are container objects, for example string objects. Such methods are only for internal usage by EB GUIDE TF.
- The second group of methods expects functor callback parameters. These callbacks must provide access to a stream of plain old data for example strings which are stored as null-terminated byte streams. These methods are also intended for internal usage by EB GUIDE TF, but you can use them, too.
- The third group of methods expects parameters which are stored as plain old data structures in a buffer, for example strings which are stored as null-terminated byte arrays. The third group is recommended for application developers.

7.2.1. EB GUIDE model

`GtfCoreModel` class represents a specific EB GUIDE model at run-time. It is an interface class which provides access to other associated interfaces for example datapool and external event system.

For more details of class descriptions, see the `.h` files in the `$GTF_INSTALL_PATH/include/` directory.

7.2.2. External event system

With events it is possible to transport a limited number of scalar values, for example integers. Events are not intended to transport large data types such as lists. The external event system delivers every event, even if newer events are available and the receiver has not yet fetched the out-of-date events. The receiver gets the events in exactly the order in which they were sent.

The interface methods of class `GtfExternalEventSystem` are described in the `GtfExternalEventSystem.h` file.

Class `GtfEvent` provides methods for event parameter encoding and decoding. The interface methods of class `GtfEvent` are described in the `GtfEvent.h` file.

7.2.2.1. Event receipt

The external event system creates a separate event queue for each communication context. Whenever new events arrive in a previously empty event queue, the external event system invokes the corresponding communication context by calling a registered callback method.

NOTE

Ensure asynchronous event processing

It is not allowed to fetch or process events directly in the registered callback method. Event fetching and processing has to be done asynchronously, even if the whole system runs in one and the same thread. The callback method must only invoke an asynchronous worker task.

The following steps show the general procedure of event receipt:

1. To register an invoking callback method at system start-up, use method `SetInvoker()`.
2. To subscribe the communication context to specific events, use method `Subscribe()` .
3. After a new event has arrived, `GtfExternalEventSystem` executes the callback method to invoke the worker task. To fetch and process all events which are currently in the event queue of the communication context, use method `Fetch()` within the task.

7.2.2.2. Event publication

The following steps show the general procedure of event publication. The interface methods of class `GtfEvent` are described in the `GtfEvent.h` file.

1. Create a local instance of class `GtfEvent` and provide the external event system, event group ID and event ID to the constructor method.
2. To publish the event, use method `Publish()`.

7.2.3. Datapool

The datapool provides an asynchronous communication mechanism based on datapool items. Datapool items can be of scalar, list, or project specific resource types, for example string lists, image lists.

Each datapool item is a communication channel between exactly one writing communication context and one reading communication context. Each communication context has a private sight on the datapool.

The writing communication context always has an up-to-date view on the datapool item. Updates of multiple associated datapool items become visible simultaneously in one step. The update prevents the GUI from flickering. The order of datapool item updates and the order of the resulting update notifications can differ.

Datapool items can change in two ways:

- The communication context manipulates the datapool item.
- The communication context updates its sight on the datapool to new datapool item values provided by other communication contexts.

Committing as well as updating affects all changed datapool items at once.

NOTE

Avoid competing datapool access of the same communication context

All datapool API methods that require a communication context ID as parameter are thread safe. Due to performance issues, avoid competing datapool access of one and the same communication context.

The interface methods of class `GtfDataPool` are described in `GtfDataPool.h`.

7.2.3.1. Internal and external IDs for datapool items

Numerical IDs are used to address the datapool item properties **Writer ID** and **Reader ID**. If the modeler does not set a value for the properties, an internal addressing is applied. If you use internal IDs, you minimize the addressing efforts during every API call. The internal property ID may change whenever the EB GUIDE model is changed.

Setting a value to the properties **Writer ID** and **Reader ID** causes external addressing.

Method `GetMappedID()` maps external datapool item IDs to the corresponding internal datapool item IDs at run-time.

As an application developer, you can force the usage of internal property IDs during API calls. You can force the usage by manipulating the numerical communication context ID. Use a bitwise OR operation to set the flag *eContextIdFlag_NoMapping* defined in `GtfTypesDP.h`.

7.2.3.2. Commitment of datapool items

If one communication context changes the value of a datapool item, the new value is not visible to another communication context. To provide the new values of a datapool item a writing communication context has to call method `Commit()`.

The method `Commit()` affects all datapool items that have been changed by the communication context since the previous call. A communication context can change multiple values one after another, but commits all of them at once.

An internal datapool item within HMI does not require committing because the reader and writer communication context are equal.

7.2.3.3. Update of datapool items

To retrieve changed values, a reading communication context has to call method `Update()`. The method `Update()` affects all datapool item values which have been manipulated and committed by other communication contexts since the previous call.

Whenever values change, the datapool invokes the corresponding reader communication context by calling a registered callback method. But this only happens if method `Update()` was called at least once since last invoking. Use method `SetInvoker()` to register an invoking callback method at system start-up.

NOTE

Ensure asynchronous update of datapool item value

It is not allowed to process updates or fetch notifications directly in the registered callback method. Updates must be processed asynchronously and notifications must be fetched asynchronously, even if the whole system runs in one and the same thread. The callback method must only invoke an asynchronous worker task.

7.2.3.4. Notifications on value updates for datapool items

Only the reader communication context of a datapool item can retrieve update notifications. Use the method `Fetch()` to fetch and process the notifications. The modeler can select a notification policy for datapool items by setting the property value.

NOTE

An update notification does not guarantee a changed value

An update notification only indicates that the corresponding datapool item has been written since method `Fetch()` was called.

7.2.3.5. Windowed lists

The windows of one list are identified by numeric IDs 0...255. There is no predefined number of windows for one list. The application can change the number of windows at run-time.

Method `List_SetLength()` sets the virtual length of the windowed list.

Method `List_SetWindow()` defines the position and size of the windows.

Method `List_Clear()` sets the virtual list length as well as the position and size of all windows to 0.

Access is possible only if list elements are covered by at least one window. If the window position or window size is changed, the newly covered list elements are uninitialized until the application writes the list element value for the first time. Read access fails for all uninitialized list elements.

7.2.4. The main workloop

The `GtfPluginLoader` gives access to the workloop that is driven by the EB GUIDE TF main thread. Therefore the interface class `GtfMainWorkLoop` is provided to all software modules.

The interface methods of class `GtfMainWorkLoop` are described in `GtfMainWorkLoop.h`.

To schedule task objects for execution, application developers can use the methods `PerformTask()` and `PerformTaskDelayed()`.

The interface methods of class `Task` are described in `task.h`.

NOTE

Avoid blocking or delaying tasks

The EB GUIDE TF main thread first processes the method `Execute()` and afterwards the method `Dispose()`. A blocking or delaying task will impede the thread and all other scheduled tasks. If such a task is defined to run in the main thread, it could, for example, delay or block HMI event processing.

7.2.5. Observer patterns and callbacks

To track the value of widget properties or to observe other model elements of the EB GUIDE model, EB GUIDE TF uses the observer pattern. There are implementations of the observer pattern with an observer interface class and respective registration methods, for example the `GtfStateMachineObserver`. Widget properties are observed using the functor template `GtfFunctorX` as shown in the following example:

```
pWidget->subscribe(pContext, propertyIndex, this,
    gtf_bind(&MyComp::propertyChanged, this, pWidget, propertyIndex)
        );
```

In the example the method `propertyChanged` is called, whenever the property at index `propertyIndex` changes. section 7.2.6, “Functors” explains the usage and behavior of `GtfFunctorX`.

7.2.6. Functors

A functor is a data type that stores a function or method invocation and provides an interface to call the encapsulated function or method like an ordinary function. In EB GUIDE TF a set of functor type templates and utility routines are provided to assemble function invocations. The `GtfFunctorX` templates are used to store callbacks.

The signature of the function call is encoded in the functor template. There is a separate functor template type for every possible number of parameters. In the documentation the number of parameters is denoted as a suffix `X`. The first template parameter of `GtfFunctorX` describes the type of the return value. All further template parameters define the expected parameter types of the call.

7.2.6.1. Initialization of functor templates

The functor type templates provides the following basic constructors:

`GtfFunctorX<R,Params>();`

The default constructor creates an empty functor object. It is safe to call an empty functor object. Empty functors can be tested using the negation operator.

`GtfFunctorX<R,Params>(R (*)(Params))`

This constructor expects a pointer to a global function or static class method as parameter. The passed function is then called by the function call operator.

`GtfFunctorX<R,Params>(R (Class::*)(Params), Class*)`

If you want to set a non-static method, you require an additional object pointer, for example as in the following code: `GtfFunctor0<void> example( &SomeObject::doIt, pSomeObject);`. There is also a variant of this constructor, which expects a pointer to a constant object and a method pointer of a constant method.

`GtfFunctorX<R,Params>(F const&)`

This is the catch-all constructor template, for assigning compatible and callable functor types. In the previous constructors the signature required an exact match of each element of the signature. This constructor also works for compatible functor types, for example if an GtfFunctor2<int,float,float> is initialized with a GtfFunctor2<int,double,double>. These two functor types are different but compatible, because a method that expects double parameters can be called with float parameters. The only requirement for the constructor parameter F is that its function call operator can be called using implicit conversion of the parameters denoted as Params.

A functor can also be initialized using the utility routines gtf_bind. The gtf_bind functions assemble a GtfFunctorX instance of the parameters given. The function is available in the following versions, which resemble the constructors of GtfFunctorX:

```
GtfFunctorX<R,Params> gtf_bind(R (*)(Params) );
GtfFunctorX<R,Params> gtf_bind(R (Class::*)(Params), Class *);
GtfFunctorX<R,Params> gtf_bind(R (Class::*)(Params)const, Class const *);
```

The syntax with gtf_bind is usually simpler and less verbose compared to the GtfFunctorX constructors. This is due to the template type deduction of the C++ compiler that allows omitting the template parameters.

7.2.6.2. GtfFunctorX value behavior

GtfFunctorX objects partially mimic the behavior of primitive values. They are put onto the stack and assigned. When assigned, the content of the GtfFunctorX on the right is duplicated.

GtfFunctorX objects cannot be compared. A comparison yields compile errors.

To make sure that a functor is configured during run-time, you can use it inside a boolean expression since it yields true when initialized. Calling an uninitialized functor is not harmful because an empty fall-back function is always available and is executed.

7.2.6.3. Argument binding with functor objects

When the signature of a method does not match the expected or required signature of the functor, it is possible to use the extended syntax of gtf_bind. The syntax allows you to attach values to the method call or reorder parameters in the method call.

When you attach values, the values are stored within the functor object - similar to the object pointer, which is stored inside the GtfFunctorX when the constructor is called with a method.

To refer to the arguments of the functor, call the placeholders objects _1, _2, ... _9 which have to be passed to the call of gtf_bind. The placeholder _1 refers to the first parameter, _2 to the second ...

NOTE

Possible dynamic memory usage with `gtf_bind` and placeholders

A functor object created with `gtf_bind` requires dynamic memory if the extended version of `gtf_bind` with placeholder functionality is used. `gtf_bind` copies all parameters into the functor object. The internal storage of `GtfFunctorX` is limited. The `GtfFunctorX` allocates heap memory if the storage is too small.

7.2.7. Inter-process communication

The most common way to integrate an application into the HMI is to develop an EB GUIDE GTF plugin. An EB GUIDE GTF plugin can receive C++ interface objects that give direct access to the external event system and the datapool of the HMI.

But in some use cases it may be required to run an application in a separate process. Unlike an EB GUIDE GTF plugin, such an application cannot receive C++ interfaces. In this case, an interaction between HMI and application processes requires inter-process communication (IPC).

EB GUIDE GTF includes optional libraries that provide a high level C++ interface for inter-process communication. The library `GtfIPC` implements a service running in the HMI process. The counterpart is `GtfIPCClient`, a library used in separate application processes to communicate with the HMI process.

7.2.8. Project specific EB GUIDE Script functions

An application developer can extend EB GUIDE Script by supplying functions written in C++. Such functions are called foreign functions and can be used in EB GUIDE Script to implement synchronous calls from the HMI to the application. A modeler can then use foreign functions in EB GUIDE Script programs. The typical use of foreign functions is to make features of some library written in C/C++ available to EB GUIDE Script programs. For example it is possible to use foreign functions to make C++ math library functions such as sinus or square root available to EB GUIDE Script programs.

TIP

EB GUIDE Script functions are not recommended for communication between HMI and application

The HMI thread is blocked until the called function returns. This may have massive impact on the timing of HMI activities. Therefore, keep the execution time of these functions as short as possible.

7.2.8.1. The EB GUIDE Script run-time stack

EB GUIDE Script uses a stack for the parameter and return values of a foreign function.

The stack plays a vital role in the execution of EB GUIDE Script programs. If there are too many or too few arguments for an instruction on the stack, the execution of the program is in an undefined state.

7.2.8.2. The foreign function interface

In order for the EB GUIDE Script compiler to generate calls to your foreign function, you provide information about your foreign function:

- The name of your function: what it is called in EB GUIDE Script programs.
- The number and types of the parameters of your function.
- The type of the return value of your function.

Parameters are passed via stack in a defined order. The first parameter of your function is at the very bottom of the stack, and the last parameter of your function is on top of the stack. The function has to pop its arguments in reverse order.

The foreign function calls all parameters which are defined in the function signature. The foreign function has to push the result value which is defined in the function signature, even if there are errors during the execution of the foreign function.

NOTE

The function has to preserve the integrity of the stack

You tell the compiler which parameters the function expects, and which return value it generates. The function has to behave according to that information. Take all parameters from the stack, and push a return value to the stack.

7.3. Communicating through a plugin

Communicating through a plugin

The following section explains the general workflow for integrating EB GUIDE TF into your build system. Find the instructions for each step in the sections below.

Step 1
Export an EB GUIDE model. For details see section 7.3.1, “Exporting an EB GUIDE model”.

Step 2
Adjust the `gtfStartup.cfg` to load the plugin. For details see section 7.3.2, “Adjusting the `gtfStartup.cfg` to load the plugin”.

Step 3
Copy the header files of the exported EB GUIDE model. For details see section 7.3.3, “Copying the header files of the exported EB GUIDE model”.

Step 4
Write a plugin. For details see section 7.3.4, “Writing a plugin”.

Step 4.1
Include header files in the plugin.

Step 4.2
Compile the plugin with the included header files.

Step 5
Copy the resulting DLL file. For details see section 7.3.5, “Copying the resulting DLL file”.

Step 6
Start simulation directly with `gtfStartup.exe`. For details see section 7.3.6, “Starting the simulation directly with `gtfStartup.exe` ”.

7.3.1. Exporting an EB GUIDE model

For information on how to export an EB GUIDE model, refer to section 5.8.5, “Exporting a project”.

In the following instructions `C:\projects\example_project` is used as export directory.

7.3.2. Adjusting the `gtfStartup.cfg` to load the plugin

Adjusting the `gtfStartup.cfg` to load the plugin

The following instruction shows you how to adjust the `gtfStartup.cfg` file so that it loads a plugin. Alternatively, you can add the plugin as a library to the profile of the EB GUIDE model in EB GUIDE Studio. For details see section 6.3.2, “Adding a library”.

Prerequisite:

- An EB GUIDE model is exported.

Step 1
Navigate to the exported EB GUIDE model.

Step 2
Open the `gtfStartup.cfg` file with a text editor.

Step 3
To load your plugin, include the following program code:

```
INIT LOAD MODEL_PATH "myapp"
```

`myapp` is the name of the example plugin.

7.3.3. Copying the header files of the exported EB GUIDE model

Copying the header files of the exported EB GUIDE model

EB GUIDE TF creates an events header file for each event group that is defined in the EB GUIDE model. For example, the header file `events_0xabe60.h` contains all the events the application can send and receive to interact with the EB GUIDE model. _0xabe60 represents the event group ID 704096 in hexadecimal notation.

EB GUIDE TF creates a datapool header file for each communication context defined in the EB GUIDE model. The header file `datapool_F.h` specifies the communication context with the ID 16. Your application uses the file to access datapool properties. It contains the communication context ID and the datapool item IDs you specify.

Prerequisite:

- An EB GUIDE model is exported.
- The ID of the communication context of your application is known.
- The `gtfStartup.cfg` file is adapted.

Step 1
Create an empty directory, for example `C:\application\myapp`.

Step 2
Navigate to the exported EB GUIDE model.

Step 3
Select the following files in `C:\projects\example_project`:

- ▶ The event header files, for example `events_0xabe60.h`.
- ▶ The datapool header files, for example `datapool_F.h`.

Step 4
Copy the selected files to the empty directory, for example `C:\application\myapp`.

7.3.4. Writing a plugin

Writing a plugin

To enable your plugin to react on datapool and event updates it is necessary to include the corresponding files.

Prerequisite:

- An EB GUIDE model is exported.
- The `gtfStartup.cfg` file is adapted.
- A new directory is created, for example `C:\application\myapp`.
- Header files from the exported EB GUIDE model are copied to this directory.

Step 1
Navigate to the directory where you copied the header files, for example `C:\application\myapp`.

Step 2
Create a file named `myapp.cpp`.

Step 3
Open the `myapp.cpp` file and write a plugin.

Find a description of all relevant classes and methods in the EB GUIDE GTF API.

Step 4
Define the communication context of your plugin.

Step 5
Include the datapool and event header files.

Step 6
Compile `myapp.cpp`.

The result is a DLL file `myapp.dll`. Your plugin is capable of communicating with the EB GUIDE model.

7.3.5. Copying the resulting DLL file

Copying the resulting DLL file

Prerequisite:

- An EB GUIDE model is exported.

- The `gtfStartup.cfg` file is adapted.
- A new directory is created, for example `C:\application\myapp`.
- Header files from the exported EB GUIDE model are copied to this directory.
- A compiled plugin including the header files from the exported EB GUIDE model is created.

Step 1
Navigate to the directory where you saved the `myapp.dll` file, for example `C:\application\myapp`.

Step 2
Copy `C:\application\myapp` to the directory where you exported the EB GUIDE model, for example `C:\projects\example_project`.

7.3.6. Starting the simulation directly with `gtfStartup.exe`

Starting the simulation directly with `gtfStartup.exe`

Prerequisite:

- An EB GUIDE model is exported.
- The `gtfStartup.cfg` file is adapted.
- A new directory is created, for example `C:\application\myapp`.
- Header files from the exported EB GUIDE model are copied to this directory.
- A compiled plugin including the header files from the exported EB GUIDE model is created.
- The resulting DLL file is available in the directory of the exported EB GUIDE model.

Step 1
Navigate to `$GUIDE_INSTALL_PATH\platform\win32\bin`.

Step 2
Execute `GtfStartup.exe` with the complete path to `gtfStartup.cfg` as the first argument. Enter the following command line:

```
GtfStartup.exe
                              C:\projects\example_project
```

The framework opens a window which displays the start view.

8. Extension developer's manual

8.1. Overview

As an extension developer you are the target audience for the following chapters. For more information, see section 1.1.4, “Extension developers”.

For more information on the structure of the manual, see section 1.2, “Structure of user documentation”.

8.2. Background information

8.2.1. Custom effect widgets

Custom effect widgets are widgets that are capable of modifying the drawing implementation. With custom effect widgets renderers are upgraded to open up individual drawing routines in custom shaders that you create. With custom shaders you overwrite default shaders of EB GUIDE. You apply the functionality of your own effects in custom shaders and a custom effect widget.

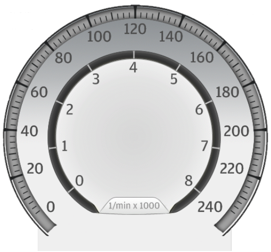


Figure 8.1. Original image drawn by default shaders

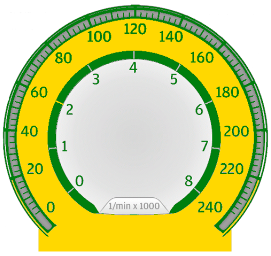


Figure 8.2. Same image, drawn by custom shaders included in a custom effect

EB GUIDE Studio uses custom effect widgets to assign its shaders to all widgets in the subtree. You specify custom effect widget properties and use them in EB GUIDE. The properties serve as uniform input parameters in your custom shaders.

Custom shaders can use the same input parameters as default shaders. If shaders require additional input parameters, you define widget properties that serve as uniform input paramaters in the shader pairs. That way, EB GUIDE Studio modelers specify input values for the shaders.

In EB GUIDE Studio, the parameters are ordinary widget properties and can be used in the same way as all widget properties in the following locations:

- Animations
- EB GUIDE Script
- Widget conditions

The OpenGL ES 2.0 and DirectX 11 renderers use default shader pairs. You can exchange the default shaders with your own implementation by using a custom effect widget. You can define a custom shader for drawing specific kinds of child objects. Thus, you are able to implement various custom effects, for example:

- Changes in coloration
- Masking effects
- Texture/image replacements in 3D graphics through widget properties in EB GUIDE Studio
- Cube/environment mapping
- Vertex distortion effects

8.2.2. Custom shaders and custom effect API

Custom shaders replace the existing shader implementation by a custom implementation. Possible reasons are, for example, to achieve better rendering results with rendering effects or to simplify rendering and get better performance. You are able to overwrite all default shaders by custom shaders.

`GtfCustomEffect` makes it possible to change the applied shaders in its subtree.

8.2.2.1. Custom input parameters: Uniforms

Define the custom input parameters for custom shaders both in the `CustomEffect` descriptor and in the `GtfCustomEffect` descriptor.

By defining custom input parameters in the `CustomEffect` descriptor, the parameters appear in EB GUIDE Studio as widget properties and can be used and changed.

The OpenGL ES 2.0 renderer is able to identify available custom input parameters that are defined in the shaders. In case of OpenGL ES 2.0, the name of the uniform in custom vertex and fragment shaders is the same as the name of the widget property in the `GtfCustomEffect` descriptor.

The DirectX 11 renderer on the other hand does not rely on the name. It puts the uniforms or constant buffers into the shader registers in ascending order. That means, HLSL shaders receive the standard constant buffers in registers *b0* until *bn* whereas *n* is the number of uniforms that are returned in the shader configuration by the custom shader. For details see GTF API documentation - "shaderUniformFlags". Custom uniforms or constant buffers are given in subsequent registers.

NOTE

Naming convention for uniform in custom shaders

Use characters in ([a-zA-Z_-][0-9a-zA-Z_-]*) for the name of the uniform in custom vertex and fragment shaders. Ensure the name differs from one of the default uniforms.

Since types in shaders are different from types in EB GUIDE, the `GtfCustomEffect` widget must provide a type for each of the custom input parameters. The `shaderCustomType` is contained in an instance of a derived class of `shaderCustomUniformBase`. It is used for types that are available in GLSL for OpenGL ES 2.0 and HLSL for DirectX 11. Therefore, the custom widget class must apply further interface methods to enable the renderer to identify the uniform parameters.

The custom widget class must apply the following interface method:

`shaderCustomUniformBase *getCustomParameter(char const * const propertyName)`

For the property name, the function returns the pointer to the instance of the custom uniform. The pointer is used for the shader uniform in the custom shader. Return value:

- If the widget property is not a custom shader input, the function returns NULL.
- If the widget property is a custom shader input, the function returns a derived shaderCustomUniform.

`PropertyName` is the name of the property in EB GUIDE which also serves as name of the uniform in the custom shader. It has to be a NULL-terminated ASCII string.

Be aware that a returned pointer must fulfill the following characteristics:

- It must point to a permanent address, not a temporary one.
- The address it points to must be unique with respect to the uniform instance. The renderer uses this instance to store current values and compares the current values to new values later on. The uniform instance must be permanent.

The return value type must be compatible to the widget property type in the `CustomEffect` descriptor. `shaderCustomType` is an enumeration mapping to the following types in the GLSL and HLSL:

8.2.2.1.1. Cube maps

It is possible to implement custom cube mapping with custom effect widgets. The EB GUIDE TF texture loader expects the cube map image in a format which includes all six images in one file. The sub-images have to be contained in the image one below the other in the order in the image below.

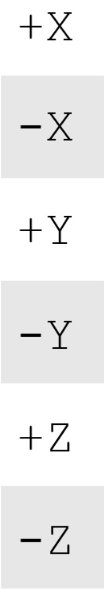


Figure 8.3. The six sides of a cube map

All sub-images have to be of the same size. This means the height of the complete cube map image has to be dividable by six.

8.2.2.1.2. Interaction of multiple GtfCustomEffect widgets

If you model multiple `GtfCustomEffect` widgets as parents of a widget, the custom shader of the nearest parent is used for drawing. If you have not configured any shader for the kind of widget in the nearest parent, the default shader is used.

A `GtfCustomEffect2` widget overwrites all custom shaders that are specified by a parent `GtfCustomEffect1`.

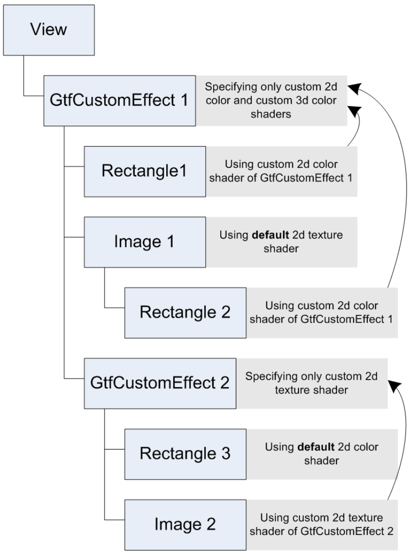

View
GtfCustomEffect 1
Specifying only custom 2d color and custom 3d color shaders
Rectangle1
Using custom 2d color shader of GtfCustomEffect 1
Image 1
Using **default** 2d texture shader
Rectangle 2
Using custom 2d color shader of GtfCustomEffect 1
GtfCustomEffect 2
Specifying only custom 2d texture shader
Rectangle 3
Using **default** 2d color shader
Image 2
Using custom 2d texture shader of GtfCustomEffect 2

Figure 8.4. Example demonstrating how multiple GtfCustomEffects interact

8.2.3. Model element descriptors

In EB GUIDE Studio it is possible to add model elements. Each model element needs a descriptor that is added to the EB GUIDE TF. The EB GUIDE TF cares about registering the additional model elements within EB GUIDE Studio. The descriptor is also known as meta information of a model element.

A component that provides such descriptors to the EB GUIDE TF is called descriptor provider. The interface methods of class `DescriptorProvider` are described in the `DescriptorProvider.h` file.

The following descriptors can be added:

- Widget descriptor

  A widget descriptor stores all information for a single widget definition. The descriptor is used to instantiate a default widget template within EB GUIDE Studio.

  The interface methods of class `WidgetDescriptor` are described in the `WidgetDescriptor.h` file.

- Widget feature descriptor

  A widget feature descriptor stores all information for a single widget feature definition. The descriptor is used to instantiate a widget feature within EB GUIDE Studio.

  The interface methods of class `WidgetFeatureDescriptor` are described in the `WidgetFeatureDescriptor.h` file.

- Action descriptor for functions in EB GUIDE Script

  An action descriptor is used to define functions in EB GUIDE Script.

The interface methods of class `ActionDescriptor` are described in the `ActionDescriptor.h` file.

To add the descriptors above, you use a property descriptor. The components catch the published property descriptor and use its information. For example, view factories use widget and widget feature information for the creation of every widget tree which is displayed.

8.2.3.1. Property descriptor

A property descriptor stores all information for a widget property. It is also used to describe the parameters within EB GUIDE Script functions.

The interface methods of class `PropertyDescriptor` are described in the `PropertyDescriptor.h` file.

8.2.3.2. Property constant descriptor

A property constant descriptor defines a name for a concrete property value. The constants are used as enumerations within EB GUIDE Studio.

For example, the integer property **alignment** can have the constants **left**, **center**, or **right**, where. And **left** stands for the value `1`, **center** stands for the value `0` and **right** stands for the value `2`.

The interface methods of class `PropertyConstantDescriptor` are described in the `PropertyDescriptor.h` file.

8.2.4. Renderer

A renderer is responsible for drawing scenes on the EB GUIDE GTF. Beside drawing, the renderer is responsible for touch input and object picking. The reason why the renderer performs object picking is that only the renderer knows at which position on the screen widgets appear.

To fulfill the tasks, the renderer uses the following interfaces:

- OpenGL ES 2.0
- DirectX 11

The OpenGL ES 2.0 and DirectX 11 renderers use pairs of fragment and vertex shaders to draw objects. These shader pairs are little programs on the graphics processing unit that are executed during rendering.

8.2.5. Shaders

Shaders affect the final look of objects. Different shader pairs are needed to render different kinds of objects. For example, an image widget requires a shader pair that supports textured objects, while a rectangle widget requires a shader pair that does not use any texture.

8.2.5.1. Shading languages

Shading languages are used to program the GPU rendering pipeline. With shaders, customized effects can be used.

The following shading languages are supported by the renderers:

- OpenGL ES 2.0 uses OpenGL ES Shading Language (GLSL).
- DirectX 11 uses High Level Shading Language (HLSL)

8.2.5.2. Input and output parameters

In GLSL and HLSL, shaders use different kinds of input and output parameters. The following parameters exist:

- Uniforms (constant buffers in HLSL):

  Input parameters that are constant for the whole drawing call
- Attributes:

  Input parameters that are constant for one vertex
- Varyings:

  Output parameters of vertex shaders and input parameters of fragment shaders. Varyings are data that is computed in vertex shaders and transferred from vertex shader to fragment shader for each vertex.

To make custom shaders capable of drawing widgets, custom shaders have to provide default input parameters.

8.2.5.3. Default shaders

There are two possible render targets in the OpenGL ES 2.0 and DirectX 11 renderers. Thus, two different shaders are used as follows:

- The screen shader, used when drawing to the screen respectively to the EB GUIDE Studio remote framebuffer
- The touch shader, used when drawing to the touch off-screen buffer

8.2.5.4. 2D and 3D default shaders

An HLSL (*.fx) shader consists of both vertex and fragment shaders. However, the vertex shader has to reside in a function named `VS` and the fragment shader has to reside in a function named `PS`. PS (Pixel Shader) is the name of the fragment shader in DirectX 11. The parameters of both the 2D and 3D default shaders are listed in the files located in the `$GUIDE_INSTALL_PATH\shaders` directory.

8.2.5.5. Touch shaders

Touch shaders are responsible for touch evaluation. Correct functionality is assured if the vertex shader transforms the vertices the same way as the non-touch variant does. The fragment shader that is used for touch input sets the resulting color to `a_color` for all fragments the user may successfully touch. Set all other fragments to complete transparency.

8.2.6. Widget set

The widget tree is composed of a generic class called `GtfWidgetModel`, which is implemented with `GtfPropertyContainer`. `GtfPropertyContainer` wraps an array of properties and a type ID.

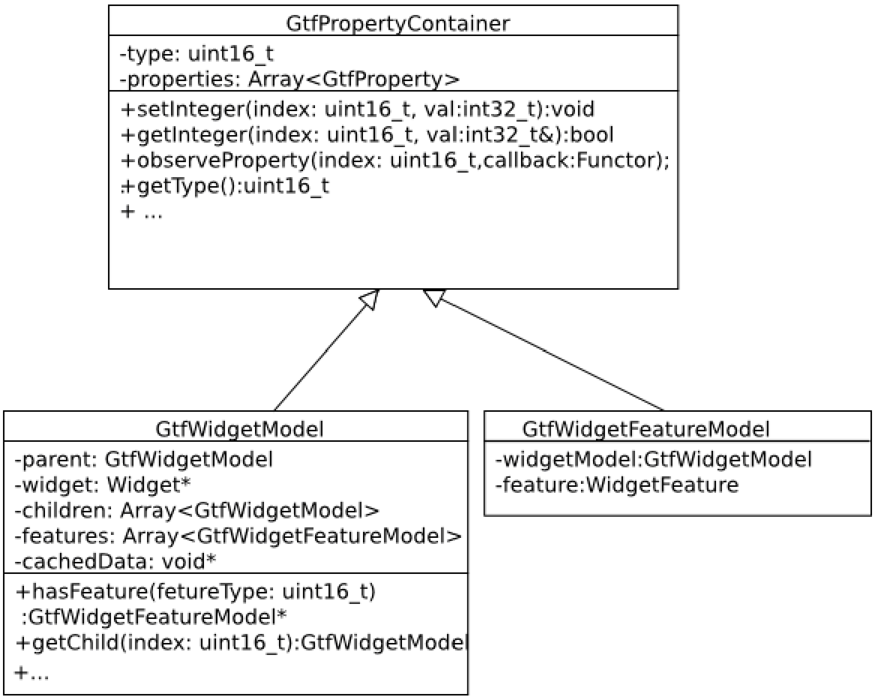

GtfPropertyContainer
-type: uint16_t
-properties: Array<GtfProperty>
+setInteger(index: uint16_t, val:int32_t):void
+getInteger(index: uint16_t, val:int32_t&):bool
+observeProperty(index: uint16_t,callback:Functor);
+getType():uint16_t
+ ...
GtfWidgetModel
-parent: GtfWidgetModel
-widget: Widget*
-children: Array<GtfWidgetModel>
-features: Array<GtfWidgetFeatureModel>
-cachedData: void*
+hasFeature(fetureType: uint16_t)
:GtfWidgetFeatureModel*
+getChild(index: uint16_t):GtfWidgetModel
+...
GtfWidgetFeatureModel
-widgetModel:GtfWidgetModel
-feature:WidgetFeature

Figure 8.5. Classes that form the generic widget tree

Type IDs are assigned during the export of the EB GUIDE model. The type ID numbering scheme allows the framework components to perform type checks in the widget tree in constant time – with a simple range check.

The renderer scans the type ID information of widgets and properties.

The GtfWidgetModel adds the following:

- An array of child widgets
- A parent pointer
- Optional: A pointer to the widget instance
- An array of widget features
- A pointer to cached renderer data to the GtfPropertyContainer

9. References

The following chapter provides you with API documentation as well as lists and tables for example parameters, properties, identifiers, etc.

9.1. Android

9.1.1. Android lifecycle management

The Android lifecycle management is an optimization implemented by the Android system. If an application is moved to the background, Android releases all graphics resources like surfaces, textures and vertex buffers to have the resources available for the application which is currently on screen. It is in the responsibility of the application to recreate the resources when the application is switched to the foreground again.

9.1.2. File path for models

EB GUIDE models are stored in the `com.elektrobit.guide_model_chooser/files` folder that is located on the primary file system. Application-related files are stored there permanently. One folder is required per EB GUIDE model.

Examples:

- For a Samsung Galaxy S3 device with Android 4.3 that is connected to a PC with Windows 7, the path is `Computer\GT-I9300\Phone\Android\data\com.elektrobit.guide_model_chooser\files`.
- For a Nexus 7 device with Android 4.4 that is connected to a PC with Windows 7, the path is `Computer\Nexus 7\Internal storage\Android\data\com.elektrobit.guide_model_chooser\files`.

On start-up or refresh, EB GUIDE Model Chooser recursively scans the folder for EB GUIDE TF configuration file `gtfStartup.cfg`. The parent folder for each start-up configuration is displayed as the model name.

9.1.3. Android layout handling

Android is designed for mobile devices. On a mobile device, some characteristics concerning the layout of the visible screen area need to be considered. Examples:

- When a mobile device is rotated, the graphical user interface of the smartphone has to adapt according to the rotation.
- When a virtual keyboard is displayed on the screen of an Android device, the graphical user interface has to adapt to the new element.

EB GUIDE supports the developer by providing events that indicate layout changes in the visible screen area.

9.1.4. Android Events

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

Table 9.1. Events

Event ID	Name	Description	Parameters
1	`RendererEnabled`	Sent by the application when Android lifecycle management stops or starts the renderer	enabled: `true` if the renderer was enabled, `false` if the renderer was set to sleep mode
2	`setKeyboardVisibility`	Sent by the EB GUIDE model if a virtual keyboard is intended to be shown	visibility: `true` if the virtual keyboard is visible, `false` if it is invisible
3	`onKeyboardVisibility-Changed`	Sent by the application if the keyboard is intended to be shown	visibility: `true` if the virtual keyboard is visible, `false` if invisible
4	`onLayoutChanged`	Sent by the application when the visible area of the screen changes	x: x-coordinate of the top left corner of the visible screen area y: y-coordinate of the top left corner of the visible screen area width: width of the visible screen area height: height of the visible screen area

9.2. Datapool items

Table 9.2. Properties of a datapool item

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

9.3. EB GUIDE Script

9.3.1. EB GUIDE Script keywords

The following is a list of reserved keywords in EB GUIDE Script. If you want to use these words as identifiers in a script, you must quote them.

Keyword	Description
`color:`	A color parameter follows, for example {0,255,255}.
`dp:`	A datapool item follows.
`else`	An `if` condition is completed. The following block is executed as an alternative.
`ev:`	An event follows.
`f:`	A user-defined function follows.
`false`	A boolean literal value
`fire`	Fires an event
`font:`	A font resource follows, for example {PT Sans,12}.
`if`	A statement which tests a boolean expression. If the expression is true, the statement is executed.
`image:`	An image resource follows.
`in`	A separator between a local variable declaration and the variable's scope of usage. Used with `match_event` and `let`.
`function`	Declares a function
`length`	Length of a property
`let`	Declares a local variable that is accessible in the scope
`list`	Declares a list type, for example an integer list
`match_event`	Checks if the current event corresponds to an expected event and declares variables like `let`
`popup_stack`	The dynamic state machine list which defines the priority of dynamic state machines
`sm:`	A state machine follows
`true`	A boolean literal value
`unit`	A value of type void
`v:`	A local variable follows.
`view:`	A view follows.
`while`	Repeats a statement as long as the condition is true

9.3.2. EB GUIDE Script operator precedence

The following is a list of the operators in EB GUIDE Script together with their associativity. Entries later in the list have higher precedence.

Table 9.3. EB GUIDE Script operator precedence

Operator	Associativity
(;)	left
(,)	right
(+=), (-=), (=), (=>)	right
(||)	left
(&&)	left
(!=), (==), (=Aa=)	left
(<), (>), (<=), (>=)	left
(+), (-)	left
(*), (/), (%)	left
length	none
(!), (&)	right
(::)	left
(.)	none
(->)	left
([])	none
(()), ({})	none

9.3.3. EB GUIDE Script standard library

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

9.3.3.1. EB GUIDE Script functions A

9.3.3.1.1. `abs`

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

Table 9.4. Parameters of `abs`

Argument	Type	Description
x	integer	Parameter
<return>	integer	Return value

9.3.3.1.2. `absf`

The function returns the absolute value of the floating point number x.

Table 9.5. Parameters of `absf`

Parameter	Type	Description
x	floating	Argument
<return>	floating	Return value

9.3.3.1.3. `acosf`

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

Table 9.6. Parameters of `acosf`

Parameter	Type	Description
x	floating	Argument
<return>	floating	Return value

9.3.3.1.4. `animation_before`

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

Table 9.7. Parameters of `animation_before`

Parameter	Type	Description
animation	GtfTypeRecord	The animation to manipulate
time	integer	point in time
<return>	boolean	True on success, false otherwise

9.3.3.1.5. `animation_beyond`

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

Table 9.8. Parameters of `animation_beyond`

Parameter	Type	Description
animation	GtfTypeRecord	The animation to manipulate
time	integer	point in time
<return>	boolean	True on success, false otherwise

9.3.3.1.6. `animation_cancel`

The function cancels an animation, leaving manipulated properties in the current state.

Table 9.9. Parameters of `animation_cancel`

Parameter	Type	Description
animation	GtfTypeRecord	The animation to manipulate
<return>	boolean	True on success, false otherwise

9.3.3.1.7. `animation_cancel_end`

The function cancels an animation and sets manipulated properties to the end state, as far as possible.

Table 9.10. Parameters of `animation_cancel_end`

Parameter	Type	Description
animation	GtfTypeRecord	The animation to manipulate
<return>	boolean	True on success, false otherwise

9.3.3.1.8. `animation_cancel_reset`

The function cancels an animation and resets changed properties to the initial state, as far as possible.

Table 9.11. Parameters of `animation_cancel_reset`

Parameter	Type	Description
animation	GtfTypeRecord	The animation to manipulate
<return>	boolean	True on success, false otherwise

9.3.3.1.9. `animation_pause`

The function pauses an animation.

Table 9.12. Parameters of `animation_pause`

Parameter	Type	Description
animation	GtfTypeRecord	The animation to manipulate
<return>	boolean	True on success, false otherwise

9.3.3.1.10. animation_play

The function starts or continues an animation.

Table 9.13. Parameters of `animation_play`

Parameter	Type	Description
animation	GtfTypeRecord	The animation to manipulate
<return>	boolean	False if the animation is already running, true otherwise

9.3.3.1.11. animation_reverse

The function plays an animation backwards.

Table 9.14. Parameters of `animation_reverse`

Parameter	Type	Description
animation	GtfTypeRecord	The animation to manipulate
<return>	boolean	False if the animation is already running, true otherwise

9.3.3.1.12. animation_running

The function determines whether an animation is currently running.

Table 9.15. Parameters of `animation_running`

Parameter	Type	Description
animation	GtfTypeRecord	The animation to manipulate
<return>	boolean	True if the animation is running

9.3.3.1.13. animation_set_time

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

Table 9.16. Parameters of `animation_set_time`

Parameter	Type	Description
animation	GtfTypeRecord	The animation to manipulate
time	integer	time
<return>	boolean	True on success, false otherwise

9.3.3.1.14. `asinf`

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

Table 9.17. Parameters of `asinf`

Parameter	Type	Description
x	floating	Argument
<return>	floating	Return value

9.3.3.1.15. `assign_language_ids`

The function fills a datapool item of type integer list with the IDs of all defined languages.

Table 9.18. Parameters of `assign_language_ids`

Parameter	Type	Description
dstItemId	dp_id	The ID of a datapool item in which to store the language IDs. The datapool item must be a list of unsigned integers.
<return>	void	

9.3.3.1.16. `assign_language_labels`

The function fills a datapool item of type string list with the labels of all defined languages.

Table 9.19. Parameters of `assign_language_labels`

Parameter	Type	Description
dstItemId	dp_id	The ID of a datapool item in which to store the language labels. The datapool item must be a string list.
<return>	void	

9.3.3.1.17. `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 9.20. Parameters of `atan2f`

Parameter	Type	Description
y	floating	Argument y
x	floating	Argument x
<return>	floating	Return value

9.3.3.1.18. `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 9.21. Parameters of `atan2i`

Parameter	Type	Description
y	integer	Argument y
x	integer	Argument x
<return>	floating	Return value

9.3.3.1.19. `atanf`

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

Table 9.22. Parameters of `atanf`

Parameter	Type	Description
x	floating	Argument
<return>	floating	Return value

9.3.3.2. EB GUIDE Script functions C - H

9.3.3.2.1. `ceil`

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

Table 9.23. Parameters of `ceil`

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

9.3.3.2.2. `changeDynamicStateMachinePriority`

The function changes the priority of a dynamic state machine.

Table 9.24. 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

9.3.3.2.3. `character2unicode`

The function returns the Unicode value of a character. The Unicode value of the first character found in the string is returned. In case of errors 0 is returned.

Table 9.25. Parameters of `character2unicode`

Parameter	Type	Description
str	string	The string with the input character
<return>	integer	The character as Unicode

9.3.3.2.4. `clearAllDynamicStateMachines`

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

Table 9.26. Parameters of `clearAllDynamicStateMachines`

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

9.3.3.2.5. `color2string`

The function prints a color as eight hexadecimal values.

Table 9.27. Parameters of `color2string`

Parameter	Type	Description
value	color	The color to convert to string
<return>	string	The color formatted as string of hexadecimal digits with # as prefix

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

Examples:

- An opaque pure red color is converted to "#ff0000ff".
- A semi-transparent pure green color is converted to "#00ff007f".

9.3.3.2.6. `cosf`

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

Table 9.28. Parameters of `cosf`

Parameter	Type	Description
x	floating	Argument
<return>	floating	Return value

9.3.3.2.7. `deg2rad`

The function converts an angle form degree to radians.

Table 9.29. Parameters of `deg2rad`

Parameter	Type	Description
x	floating	Argument
<return>	floating	Return value

9.3.3.2.8. `expf`

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

Table 9.30. Parameters of `expf`

Parameter	Type	Description
x	floating	Argument
<return>	floating	Return value

9.3.3.2.9. `float2string`

The function converts simple floating to string.

Table 9.31. Parameters of `float2string`

Parameter	Type	Description
value	floating	The value to convert to string

Parameter	Type	Description
<return>	string	The floating value, formatted as string

9.3.3.2.10. `floor`

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

Table 9.32. Parameters of `floor`

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

9.3.3.2.11. `focusNext`

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

Table 9.33. Parameters of `focusNext`

Parameter	Type	Description
<return>	void	

9.3.3.2.12. `focusPrevious`

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

Table 9.34. Parameters of `focusPrevious`

Parameter	Type	Description
<return>	void	

9.3.3.2.13. `formatFloat`

The function converts advanced floating to string.

Table 9.35. Parameters of `formatFloat`

Parameter	Type	Description
minStrLen	integer	Minimum length of the result string
maxStrLen	integer	Maximum length of the result string
minPrecision	integer	Minimum number of decimal places

Parameter	Type	Description
maxPrecision	integer	Maximum number of decimal places
showAbsoluteValue	boolean	If **value** is negative, the formatter negates it before formatting, thus turning it positive.
alwaysShowSign	boolean	Forces the formatter to show the sign for both negative and positive values
roundingMode	integer	The rounding mode that the formatter uses to ensure the maximum length of the result string. Possible values: ▶ 0: trunc ▶ 1: round
fillStyle	integer	The character to use to ensure the minimum length of the result string. Possible values: ▶ 0: fills with blanks ▶ 1: fills with zeros
value	floating	The number to format
<return>	string	The formatted string according to the options

9.3.3.2.14. `formatInteger`

The function converts advanced integer to string.

Table 9.36. Parameters of `formatInteger`

Parameter	Type	Description
minStrLen	integer	Minimum length of the result string
maxStrLen	integer	Maximum length of the result string
showAbsoluteValue	boolean	If **value** is negative, the formatter negates it before formatting, thus turning it positive.
alwaysShowSign	boolean	Forces the formatter to show the sign for both negative and positive values
fillStyle	integer	The character to use to ensure the minimum length of the result string. Possible values: ▶ 0: fills with blanks ▶ 1: fills with zeros
base	integer	Possible values:

Parameter	Type	Description
		▶ 2: binary ▶ 10: decimal ▶ 16: hexadecimal
value	integer	The number to format
<return>	string	The formatted string according to the options

9.3.3.2.15. `getTextHeight`

The function obtains the height of a text regarding its font resource.

Table 9.37. Parameters of `getTextHeight`

Parameter	Type	Description
text	string	The text to evaluate
font	font	The font to evaluate the text size
<return>	integer	The height of the text

9.3.3.2.16. `getTextLength`

The function obtains the number of characters in a text.

Table 9.38. Parameters of `getTextLength`

Parameter	Type	Description
text	string	The text which to evaluate
<return>	integer	Number of characters in the text

9.3.3.2.17. `getTextWidth`

The function obtains the width of a text regarding its font resource.

Table 9.39. Parameters of `getTextWidth`

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

9.3.3.2.18. `has_list_window`

The function checks if the index is valid for a global list property. For windowed list properties it also checks if the index is located inside at least one window.

Table 9.40. Parameters of `has_list_window`

Parameter	Type	Description
itemId	dp_id	Datapool ID of the global list property
index	integer	Index within the global list property
<return>	boolean	True if the index within a global list property is valid and located inside at least one window, false otherwise

9.3.3.2.19. `hsba2color`

The function converts from HSB/HSV color space to GTF color.

Table 9.41. Parameters of `hsba2color`

Parameter	Type	Description
hue	integer	The color value in degrees from 0 to 360
saturation	integer	The saturation in percent
brightness	integer	The brightness in percent
alpha	integer	The alpha value, ranging from 0 (totally transparent) to 255 (opaque)
<return>	color	The color converted from HSB color space to GTF color, with the alpha value applied

9.3.3.3. EB GUIDE Script functions I - R

9.3.3.3.1. `int2float`

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

Table 9.42. Parameters of `int2float`

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

9.3.3.3.2. `int2string`

The function converts a simple integer to string.

Table 9.43. 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

9.3.3.3.3. `language`

The function switches the language of all datapool items, independent of group membership.

Table 9.44. Parameters of `language`

Parameter	Type	Description
languageId	integer	The ID of the language to switch to
<return>	void	

9.3.3.3.4. `language_of_group`

The function switches the language of a group of datapool items.

Table 9.45. Parameters of `language_of_group`

Parameter	Type	Description
groupId	integer	The ID of the group of datapool items for which to switch the language
languageId	integer	The ID of the language to switch to
<return>	void	

9.3.3.3.5. `localtime_day`

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

Table 9.46. Parameters of `localtime_day`

Parameter	Type	Description
time	integer	A time stamp as returned by system_time
<return>	integer	The extracted information

9.3.3.3.6. `localtime_hour`

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

Table 9.47. Parameters of `localtime_hour`

Parameter	Type	Description
time	integer	A time stamp as returned by system_time
<return>	integer	The extracted hour

9.3.3.3.7. `localtime_minute`

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

Table 9.48. Parameters of `localtime_minute`

Parameter	Type	Description
time	integer	A time stamp as returned by system_time
<return>	integer	The extracted minute

9.3.3.3.8. `localtime_month`

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

Table 9.49. Parameters of `localtime_month`

Parameter	Type	Description
time	integer	A time stamp as returned by system_time
<return>	integer	The extracted month

9.3.3.3.9. `localtime_second`

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

Table 9.50. Parameters of `localtime_second`

Parameter	Type	Description
time	integer	A time stamp as returned by system_time
<return>	integer	The extracted second

9.3.3.3.10. `localtime_weekday`

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

Table 9.51. Parameters of `localtime_weekday`

Parameter	Type	Description
time	integer	A time stamp as returned by system_time
<return>	integer	The extracted weekday

9.3.3.3.11. `localtime_year`

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

Table 9.52. Parameters of `localtime_year`

Parameter	Type	Description
time	integer	A time stamp as returned by system_time
<return>	integer	The extracted year

9.3.3.3.12. `log10f`

The function returns the base 10 logarithm of x.

Table 9.53. Parameters of log10f

Parameter	Type	Description
x	floating	Argument
<return>	floating	Return value

9.3.3.3.13. `logf`

The function returns the natural logarithm of x.

Table 9.54. Parameters of `logf`

Parameter	Type	Description
x	floating	Argument
<return>	floating	Return value

9.3.3.3.14. `nearbyint`

The function rounds to nearest integer.

Table 9.55. Parameters of `nearbyint`

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

9.3.3.3.15. `popDynamicStateMachine`

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

Table 9.56. Parameters of `popDynamicStateMachine`

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

9.3.3.3.16. `powf`

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

Table 9.57. Parameters of powf

Parameter	Type	Description
x	floating	Argument x
y	floating	Argument y
<return>	floating	Return value.

9.3.3.3.17. `pushDynamicStateMachine`

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

Table 9.58. Parameters of `pushDynamicStateMachine`

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

9.3.3.3.18. `rad2deg`

The function converts an angle form radians to degree.

Table 9.59. Parameters of `rad2deg`

Parameter	Type	Description
x	floating	Argument
<return>	floating	Return value

9.3.3.3.19. `rand`

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

Table 9.60. Parameters of `rand`

Parameter	Type	Description
<return>	integer	A random number between -2^{31} and 2^{31}-1

9.3.3.3.20. `request_runlevel`

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

Table 9.61. Parameters of `request_runlevel`

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

9.3.3.3.21. `rgba2color`

The function converts from RGB color space to GTF color.

Table 9.62. Parameters of `rgba2color`

Parameter	Type	Description
red	integer	The red color coordinate, ranging from 0 to 255
green	integer	The green color coordinate, ranging from 0 to 255
blue	integer	The blue color coordinate, ranging from 0 to 255

Parameter	Type	Description
alpha	integer	The alpha value, ranging from 0 (totally transparent) to 255 (opaque)
<return>	color	The color converted from RGB color space to GTF color, with the alpha value applied

9.3.3.3.22. `round`

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

Table 9.63. Parameters of `round`

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

9.3.3.4. EB GUIDE Script functions S - W

9.3.3.4.1. `seed_rand`

The function sets the seed of the random number generator.

Table 9.64. Parameters of `seed_rand`

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

9.3.3.4.2. `sinf`

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

Table 9.65. Parameters of `sinf`

Parameter	Type	Description
x	floating	Argument
<return>	floating	Return value

9.3.3.4.3. `sqrtf`

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

Table 9.66. Parameters of `sqrtf`

Parameter	Type	Description
x	floating	Argument
<return>	floating	Return value

9.3.3.4.4. `string2float`

The function converts the initial part of the string to floating.

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 9.67. Parameters of `string2float`

Parameter	Type	Description
str	string	The string value
<return>	floating	Return value

9.3.3.4.5. `string2int`

The function converts the initial part of the 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 9.68. Parameters of `string2int`

Parameter	Type	Description
str	string	The string value
<return>	integer	Return value

9.3.3.4.6. `string2string`

The function formats strings.

Table 9.69. Parameters of `string2string`

Parameter	Type	Description
str	string	The string to format
len	integer	The maximum length of the string
<return>	string	The language string

9.3.3.4.7. `substring`

The function creates a substring copy of the string. Negative end indexes are supported.

Examples:

- substring("abc", 0, -1) returns "abc".
- substring("abc", 0, -2) returns "ab".

Table 9.70. Parameters of `substring`

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

9.3.3.4.8. `system_time`

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

Table 9.71. Parameters of `system_time`

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

9.3.3.4.9. `system_time_ms`

The function gets the current system time in milliseconds.

Table 9.72. Parameters of `system_time_ms`

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

9.3.3.4.10. `tanf`

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

Table 9.73. Parameters of `tanf`

Parameter	Type	Description
x	floating	Argument
<return>	floating	Return value

9.3.3.4.11. `trace_dp`

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

Table 9.74. Parameters of >`trace_dp`

Parameter	Type	Description
itemId	dp_id	Datapool ID of the item to trace debug information about
<return>	void	

9.3.3.4.12. `trace_string`

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

Table 9.75. Parameters of `trace_string`

Parameter	Type	Description
str	string	The text to trace
<return>	void	

9.3.3.4.13. `transformToScreenX`

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

Table 9.76. Parameters of `transformToScreenX`

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

9.3.3.4.14. `transformToScreenY`

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

Table 9.77. Parameters of `transformToScreenY`

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

9.3.3.4.15. `transformToWidgetX`

The function takes a widget and a screen coordinate as provided to the touch reactions and returns x position in the widget-relative local coordinate system.

Table 9.78. Parameters of `transformToWidgetX`

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

9.3.3.4.16. `transformToWidgetY`

The function takes a widget and a screen coordinate as provided to the touch reactions and returns y position in the widget-relative local coordinate system.

Table 9.79. Parameters of `transformToWidgetY`

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

9.3.3.4.17. `trunc`

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

Table 9.80. Parameters of `trunc`

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

9.3.3.4.18. `widgetGetChildCount`

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

Table 9.81. Parameters of `widgetGetChildCount`

Parameter	Type	Description
widget	widget	The widget of which to obtain the number of children.
<return>	integer	The number of child widgets

9.4. Events

Table 9.82. 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	Name of the event group An event group has a corresponding ID that EB GUIDE TF uses to send and receive the event.

9.5. Scenes

Table 9.83. 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 children are visible.
projectName	The name of the project
windowCaption	The text shown on the window frame
sceneID	A unique scene identifier which can be used for example for input handling
maxFPS	FPS = Frames per second Limits the redraw rate to the value you set 0: unlimited
hwLayerID	Maps the value for the rendering of the current state machine to a hardware layer of a target display
colorMode	The color depth which the renderer uses ▶ 1: 32 bit ▶ 2: 16 bit
multisampling	0: no multisampling is used 1: 2x multisampling is used 2: 4x multisampling is used
enableRemoteFramebuffer	Enables transfer of the off-screen buffer to the simulation window
showWindowFrame	Puts a frame on the simulation window that allows the window to be grabbed and moved
showWindow	Opens an additional window for simulation on Windows based systems
disableVSync	Disables vertical synchronization for the renderer

Property name	Description
Renderer	Defines a renderer for the scene ▶ DirectX ▶ OpenVG ▶ OpenGL ES

9.6. Touch screen types supported by EB GUIDE GTF

The actual types supported depend on target platform.

Table 9.84. 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

9.7. Widgets

9.7.1. View widget

The view widget has the following properties.

Table 9.85. Properties of the view widget

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	Flag which determines if the widget and its children are visible
x	The x coordinate of the widget
y	The y coordinate of the widget

9.7.2. Basic widgets

There are five basic widgets.

- Label
- Image
- Rectangle
- Container
- Instantiator

The following sections list the properties of basic widgets.

NOTE

Unique names

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

9.7.2.1. Label

Table 9.86. Properties of the label widget

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	Flag which determines if the widget and its children 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

Property name	Description
text	The text the label displays
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.

9.7.2.2. Rectangle

Table 9.87. Properties of the rectangle widget

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	Flag which determines if the widget and its children 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

9.7.2.3. Image

Table 9.88. Properties of the image widget

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	Flag which determines if the widget and its children 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 within the boundaries of the image widget
verticalAlign	The vertical alignment of the image within the boundaries of the image widget

NOTE

Default renderer supports PNG and JPEG

The available image formats depend on the implementation of the renderer used. The default renderer only supports PNG files and JPEG files.

9.7.2.4. Container

Table 9.89. Properties of the container widget

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	Flag which determines if the widget and its children 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

9.7.2.5. Instantiator

Table 9.90. Properties of the instantiator widget

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	Flag which determines if the widget and its children are visible
x	The x coordinate of the widget relative to its parent widget
y	The y coordinate of the widget relative to its parent widget
numItems	The number of instantiated child elements
lineMapping	Defines which child is the template for which line

9.7.3. Animations

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

9.7.3.1. Animation

Table 9.91. Properties of the animation widget

Property name	Description
name	The name of the animation
alternating	Defines if the animation is executed repeatedly
repeat	Number of repetitions, `0` for infinite number
enabled	Defines if the animation is executed
scale	Factor by which the animation time is multiplied
onPause	Reaction that is executed when the animation is paused. Parameter: Current animation time.
onPlay	Reaction that is executed when the animation is started or continued. Parameters: Start time and play direction (`true` for forwards, `false` for backwards).
onTerminate	Reaction that is executed when the animation completes. First parameter: animation time. Second parameter: reason for the termination, encoded as follows. ▶ 0: Animation is completed. ▶ 1: Animation is cancelled, triggered by `f:animation_cancel` . ▶ 2: Widget is destroyed due to view transition. ▶ 3: Animation jumps to its last step, triggered by `f:animation_cancel_end`. ▶ 4: Animation jumps to its first step and is then canceled, triggered by `f:animation_cancel_reset`.

9.7.3.2. Constant curves

Constant curve widgets are available for int, bool, float, and color types.

Table 9.92. Properties of constant curve widgets

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

Property name	Description
repeat	The number of repetitions
target	The target property the resulting value is assigned to
value	The resulting constant value

9.7.3.3. Fast start curves

Fast start curve widgets are available for int, float, and color types.

Table 9.93. Properties of fast start curve widgets

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

9.7.3.4. Slow start curves

Slow start curve widgets are available for int, float, and color types.

Table 9.94. Properties of slow start curve widgets

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

Property name	Description
repeat	The number of repetitions
target	The target property the resulting value is assigned to
start	The initial value
end	The final value

9.7.3.5. Quadratic curves

Quadratic curve widgets are available for int, float, and color types.

Table 9.95. Properties of quadratic curve widgets

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

9.7.3.6. Sinus curves

Sinus curve widgets are available for int, float, and color types.

Table 9.96. Properties of sinus curve widgets

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

Property name	Description
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 degrees
frequency	The frequency of the curve in hertz

9.7.3.7. Script curves

Script curve widgets are available for int, bool, float, and color types.

Table 9.97. Properties of script curve widgets

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

9.7.3.8. Linear curves

Linear curve widgets are available for int, float, and color types.

Table 9.98. Properties of linear curve widgets

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

Property name	Description
duration	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

9.7.3.9. Linear interpolation curves

Linear interpolation curve widgets are available for int, float, and color types.

Table 9.99. Properties of linear interpolation curve widgets

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

9.7.4. 3D widgets

In the 3D category, there is the 3D graphic widget and four custom effect widgets.

- 3D graphic
- Material effect
- Light effect
- Light and material effect

- No lighting effect

The following sections list the properties of 3D widgets.

9.7.4.1. 3D graphic

Table 9.100. Properties of the 3D graphic widget

Property name	Description
3D graphic	The 3D graphic file to be displayed

9.7.4.1.1. Supported 3D graphic formats

Only the OpenGL ES 2.0 and DirectX 11 renderers can display 3D graphics. Supported 3D graphic formats are as follows:

- Collada (`.dae`)
- Blender 3D (`.blend`)
- 3ds Max 3DS (`.3ds`)
- 3ds Max ASE (`.ase`)
- Wavefront Object (`.obj`)
- Industry Foundation Classes (IFC/Step) (`.ifc`)
- XGL (`.xgl`,`.zgl`)
- Stanford Polygon Library (`.ply`)
- LightWave (`.lwo`)
- LightWave Scene (`.lws`)
- Modo (`.lxo`)
- Stereolithography (`.stl`)
- DirectX X (`.x`)
- AC3D (`.ac`)
- Milkshape 3D (`.ms3d`)
- Ogre XML (`.mesh.xml`)
- Irrlicht Mesh (`.irrmesh`)
- Quake I (`.mdl`)
- Quake II (`.md2`)

- Quake III (.md3)
- Doom 3 (.md5*)
- BlitzBasic 3D (.b3d)
- Quick3D (.q3d,.q3s)
- Neutral File Format (.nff)
- Sense8 WorldToolKit (.nff)
- Object File Format (.off)
- PovRAY Raw (.raw)
- Terragen Terrain (.ter)
- 3D GameStudio (3DGS) (.mdl)
- 3D GameStudio Terrain (3DGS) (.hmp)
- Izware Nendo (.ndo)

3D graphic formats with limited support are as follows:

- AutoCAD DXF (.dxf)
- TrueSpace (.cob,.scn)
- Irrlicht Scene (.irr)
- Return to Castle Wolfenstein (.mdc)
- Valve Model (.smd,.vta)
- Starcraft II M3 (.m3)
- Unreal (.3d)

9.7.4.2. Light effect

Table 9.101. Properties of the light effect widget

Property name	Description
name	The name of the effect
enabled	Defines if the effect is in use
lightPos	Defines the position of the light for child 3D graphic widgets. A vector of three floats.

9.7.4.3. Material effect

Table 9.102. Properties of the material effect widget

Property name	Description
name	The name of the effect
enabled	Defines if the effect is in use
ambientColor	Defines the ambient color of child 3D graphic widgets. Three values for red, green, and blue range from 0.0 to 1.0.
diffuseColor	Defines the diffuse color of child 3D graphic widgets. Four values for red, green, blue, and alpha range from 0.0 to 1.0.
specularColor	Defines the specular color of child 3D graphic widgets. Three values for red, green, and blue range from 0.0 to 1.0.
specularShininess	Defines the specular shininess of child 3D graphic widgets.

9.7.4.4. Light and material effect

Table 9.103. Properties of the light and material effect widget

Property name	Description
name	The name of the effect
enabled	Defines if the effect is in use
ambientColor	Defines the ambient color of child 3D graphic widgets. Three values for red, green, and blue range from 0.0 to 1.0.
diffuseColor	Defines the diffuse color of child 3D graphic widgets. Four values for red, green, blue, and alpha range from 0.0 to 1.0.
specularColor	Defines the specular color of child 3D graphic widgets. Three values for red, green, and blue range from 0.0 to 1.0.
specularShininess	Defines the specular shininess of child 3D graphic widgets.
lightPos	Defines the position of the light for child 3D graphic widgets. A vector of three floats.

9.7.4.5. No lighting effect

Table 9.104. Properties of the no lighting effect widget

Property name	Description
name	The name of the effect
enabled	Defines if the effect is in use

9.8. 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.

9.8.1. Common

9.8.1.1. Virtual layer

The **Virtual layer** widget feature defines that a widget is bound to a layer. During run-time, the layer is mapped to a real hardware layer.

Table 9.105. Properties of the **Virtual layer** widget feature

Property name	Description
layerId	During run-time, the layer influences rendering order for all widgets with this widget feature. Possible values range from 0 for the lowest layer to 5 for the top layer.

NOTE

Layer assignments are static

Layer assignments are static and cannot be changed during run-time.

9.8.1.2. Text truncation

The **Text truncation** widget feature truncates the content of the **text** property if it does not fit into the widget area.

Table 9.106. Properties of the **Text truncation** widget feature

Property name	Description
truncationPolicy	For single-line texts, **truncationPolicy** defines the position of the truncation. Possible values: ▶ Leading: Text is replaced at the beginning of the text. ▶ Trailing: Text is replaced at the end of the text. For multi-line texts, **truncationPolicy** defines where text is replaced. Possible values:

Property name	Description
	▶ Leading: Lines at the beginning are replaced and text of the first visible line is truncated at the beginning of the text. ▶ Trailing: 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

NOTE

Labels with bi-directional texts

Text that contains two text directions, right-to-left and left-to-right, is called bi-directional text. In case of bi-directional text, the truncation symbol is added with respect to the text, but not to its formatting. This means:

- If you use `leading` the truncation symbol is added to the left-hand side of the first visible line.
- If you use `trailing` the truncation symbol is added to the right-hand side of the last visible line.

9.8.1.3. Toggle button

The **Toggle button** widget feature changes a button into a toggle button. The appearance of toggle buttons does not change on pressing or releasing.

The **Toggle button** widget feature has no additional properties.

9.8.1.4. State enabled

The **State enabled** widget feature adds an **enabled** property to a widget.

Table 9.107. Properties of the **State enabled** widget feature

Property name	Description
enabled	If true, the widget reacts on touch and press input

9.8.1.5. State selected

The **State 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 9.108. Properties of the **State selected** widget feature

Property name	Description
selected	Returns true if the widget is selected

9.8.1.6. State focused

The **State focused** widget feature enables a widget to have input focus.

Table 9.109. Properties of the **State focused** widget feature

Property name	Description
focusable	Defines whether the widget receives the focus or not. Possible values: ▶ 0: not focusable ▶ 1: focusable only by touch ▶ 2: focusable only by key ▶ 3: focusable
focused	If true, the widget has focus

9.8.1.7. State touched

The **State touched** widget feature enables a widget to react to touch input.

Table 9.110. Properties of the **State touched** widget feature

Property name	Description
touchable	If true, the widget reacts on touch input
touched	If true, the widget is currently touched
touchPolicy	Defines how to handle touch and movement that crosses widget boundaries. Possible values: ▶ Press then react: Press first, then the widget reacts. Notifications of moving and releasing are only active within the widget area. ▶ Press and grab: Press to grab the contact. The contact remains grabbed even if it moves away from the widget area. ▶ Press then react on contact: Even if the contact enters the pressed state outside the widget boundaries, the subsequent move and release events are delivered to the widget.
touchBehavior	Defines touch evaluation. Possible values:

Property name	Description
	▶ 0: Whole area To identify the touched widget, the renderer evaluates the widget's clipping rectangle. ▶ 1: Visible pixels To identify the touched widget, the renderer evaluates the widget the touched pixel belongs to. Transparent pixels in an image with alpha transparency or pixels inside letters such as in O or A are not touchable.

Combining the **State touched** widget feature with the **Touch pressed** widget feature allows modelling a push button.

TIP

Performance recommendation:

If performance is an important issue in your project set the **touchBehavior** property to **Whole area**. EB GUIDE GTF evaluates **Whole area** faster than **Visible pixels**.

9.8.1.8. State pressed

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

Table 9.111. Properties of the **State pressed** widget feature

Property name	Description
pressed	True if a key is pressed while the widget is focussed

Combining the **State touched** widget feature with the **Touch pressed** widget feature allows modelling a push button.

9.8.1.9. Multi-state

The **Multi-state** widget feature handles the visibility of child widgets. Only the content of one child is visible at a time.

Table 9.112. Properties of the **Multi-state** widget feature

Property name	Description
containerIndex	Index of the children of the parent widget

Property name	Description
containerMapping	If a mapping is set, each child of the container is re-addressed by its appropriate value in `containerMapping`. If a mapping is not set, undefined or if the length does not match the number of children in the container, the mapping is not used. Instead, the order of widgets in the widget tree is used as their index. The topmost child has index 0, next index 1 etc.

9.8.1.10. Multi-line

The **Multi-line** widget feature enables line breaks for a label widget.

Table 9.113. Properties of the **Multi-line** widget feature

Property name	Description
lineGap	The size of the gap between the lines. A negative value decreases the gap, a positive value increases the gap.
lineSeparators	Defines at what letter in the line a break is to be made once the line is full.
maxLineCount	The number of visible lines

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 ' '.

9.8.1.11. Button group

The **Button group** widget feature is used to model an array of radio buttons. In an array, every radio button has the **Button group** widget feature and a unique button ID.

Use a datapool item for the **buttonValue** property. Assign the datapool item to all widgets in the radio button array.

Selecting and deselecting a widget within the button group can be done by an external application that sets the **buttonValue** property. Alternatively, changes can be triggered by touch or key input as well as by adding a condition that sets the button value.

Table 9.114. Properties of the **Button group** widget feature

Property name	Description
buttonId	The ID that identifies a button within a button group

Property name	Description
buttonValue	The current value of a button. If this value matches the **buttonId**, the button is selected.
selected	Evaluates if **buttonID** and **buttonValue** are identical. If true, the button is selected.

9.8.1.12. Rotary button

The **Rotary button** widget feature turns a widget into a rotary button. A widget with the **Rotary button** widget feature reacts to increment and decrement events by changing an internal value. The **Rotary button** widget feature can be used to create a scale, a progress bar, or a widget with a preview value.

Table 9.115. Properties of the **Rotary button** widget feature

Property name	Description
currentValue	The current rotary value
maxValue	The maximum value for the **currentValue** property
minValue	The minimum value for the **currentValue** property
incValueTrigger	If true, the **currentValue** property is incremented by 1
incValueReaction	Reaction to an incrementation of the **currentValue** property
decValueTrigger	If true, the current value is decremented by 1
decValueReaction	Reaction to a decrementation of the **currentValue** property
steps	The number of steps to calculate the increment or decrement for the **currentValue** property
valueWrapAround	Possile values: ▶ true: **currentValue** continues at the inverse border, if **minValue** or `maxValue` is exceeded. ▶ false: **currentValue** does not decrease/increase if `minValue` or `maxValue` is exceeded.

9.8.2. Focus

9.8.2.1. User-defined focus

The **User-defined focus** widget feature enables additional focus functionality for the widget. A widget that uses the feature manages a local focus hierarchy for its widget subtree.

Table 9.116. Properties of the **User-defined focus** widget feature

Property name	Description
focusNext	The trigger that assigns the focus to the next child widget
focusOrder	Focus order makes it possible to skip child widgets when assigning focus. The ID of a child widget corresponds to its position in the subtree. Child widgets that are not focusable are skipped by default. Order in which the child widgets are focused: ▶ defined: User-defined widget order is used. ▶ not defined: Default widget order is used instead. Each child widget requires the **State focused** widget feature, otherwise widgets are ignored for focus handling. Example: focusOrder=1|0|2 means the second widget receives focus first, then the first widget receives focus, and finally the third widget.
focusPrevious	The trigger that assigns the focus to the previous child
focusFlow	The behavior for focus changes within the hierarchy. Possible values: ▶ 0: stop at hierarchy level ▶ 1: wrap within hierarchy level ▶ 2: step up in hierarchy
focusedIndex	The index defines the position of the child widget in the **focusOrder** list. If the widget is not focusable, the child next in the list is used.
initFocus	The index of the focused child widget at initialization

9.8.2.2. Auto focus

With the **Auto focus** widget feature, the order in which child widgets are focused is pre-defined. Focusable child widgets cannot be skipped. A widget with the **Auto focus** widget feature manages a local focus hierarchy for its widget subtree. The Auto focus widget feature checks the widget subtree for child widgets with the **focusable** property.

The order of the widgets in the layout is used to calculate focus order. Depending on layout orientation, the algorithm begins in the upper left or upper right corner.

Table 9.117. Properties of the **Auto focus** widget feature

Property name	Description
focusNext	The condition on which the focus index is incremented
focusPrevious	The condition on which the focus index is decremented.

Property name	Description
focusFlow	The behavior for focus changes within the hierarchy. Possible values: ▶ 0: stop at hierarchy level ▶ 1: wrap within hierarchy level ▶ 2: step up in hierarchy
focusedIndex	The index of the currently focused child widget as the n-th child widget which is focusable
initFocus	The index defines the focused child widget at initialization. If the widget is not focusable, the next focusable child is used.

9.8.3. Input handling

9.8.3.1. Move over

The **Move over** widget feature enables a widget to react on movement within its boundaries.

Property name	Description
moveOver	The widget's reaction on a movement within its boundaries

9.8.3.2. Move out

The **Move out** widget feature enables a widget to react on movement out of its boundaries.

Property name	Description
moveOut	The widget's reaction on a movement out of its boundaries

9.8.3.3. Move in

The **Move in** widget feature enables a widget to react on movement into its boundaries.

Property name	Description
moveIn	The widget's reaction on a movement into its boundaries

9.8.3.4. Touch pressed

The **Touch pressed** widget feature enables a widget to react on being pressed.

Property name	Description
touchPressed	The widget's reaction on being pressed

9.8.3.5. Touch released

The **Touch released** widget feature enables a widget to react on being released.

Property name	Description
touchShortReleased	The widget's reaction on being released

9.8.3.6. Touch grab lost

The **Touch grab 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.

Property name	Description
onTouchGrabLost	The reaction on a lost touch contact

9.8.3.7. Touch status changed

The **Touch status changed** widget feature enables a widget to react on changes of its touch status.

Property name	Description
touchStatusChanged	The widget's reaction on changes of its touch status

9.8.3.8. Touch move

The **Touch move** widget feature enables a widget to react on being touched and moved.

Property name	Description
touchMoved	The widget's reaction on being touched and moved

9.8.3.9. Gestures

The **Gestures** widget feature enables the widget to react on touch gestures.

The **Gestures** widget feature has no additional properties.

9.8.3.10. Key pressed

The **Key pressed** widget feature enables a widget to react on a key being pressed.

Property name	Description
keyPressed	The widget's reaction on a key being pressed. Reaction argument: ▶ keyId: returns true, if the widget reacts on the incoming key event

9.8.3.11. Key Unicode

The **Key Unicode** widget feature enables a widget to react on Unicode key input.

Property name	Description
keyUnicode	The widget's reaction on a Unicode key input. Reaction argument: ▶ keyId: returns true, if the widget reacts on the incoming key event

9.8.3.12. Key released

The **Key released** widget feature enables a widget to react on a key being released.

Property name	Description
keyShortReleased	The widget's reaction on a key being released. Reaction argument: ▶ keyId: returns true, if the widget reacts on the incoming key event

9.8.3.13. Key status changed

The **Key status changed** widget feature enables a widget to react on a key being pressed or released. It defines the reaction to key input such as **short press**, **long**, **ultra long** and **continuous**.

Property name	**Description**
keyLongPressed	The widget's reaction on a key being pressed or released. Reaction argument: ▶ keyId: returns true, if the widget reacts on the incoming key event

9.8.3.14. Rotary

The **Rotary** widget feature enables a widget to react on being rotated.

Property name	**Description**
rotaryReaction	The widget's reaction on being rotated. Returns true, if the widget reacts on an incoming rotary event. Reaction arguments: ▶ rotaryId: integer ID ▶ increment: number of units the rotary input shifts when the incoming event is sent

9.8.3.15. Moveable

The **Moveable** widget feature enables a widget to ne moved by touch.

Property name	**Description**
moveDirection	The direction into which the widget moves. Possible values: ▶ 0: free ▶ 1: horizontal ▶ 2: vertical

9.8.4. Gestures

9.8.4.1. Hold gesture

A hold gesture without movement

NOTE The **Hold gesture** widget feature does not trigger the **Touch grab lost** widget feature.

Table 9.118. Properties of the **Hold gesture** widget feature

Property name	Description
holdDuration	Minimal time in milliseconds the contact must stay in place for the gesture to be recognized as a hold gesture
onGestureHold	Reaction that is triggered once the gesture is recognized. The reaction is triggered only once per contact: when **holdDuration** is expired and the contact still is in a small boundary box around the initial touch position. Reaction arguments: ▶ x: X coordinate of the contact position ▶ y: Y coordinate of the contact position

9.8.4.2. Long hold gesture

A long hold gesture without movement

NOTE The **Long hold gesture** widget feature does not trigger the **Touch grab lost** widget feature.

Table 9.119. Properties of the **Long hold gesture** widget feature

Property name	Description
longHoldDuration	Minimal time in milliseconds the contact must stay in place for the gesture to be recognized as a long hold gesture
onGestureLongHold	Reaction that is triggered once the gesture is recognized. The reaction is triggered only once per contact: when **longHoldDuration** has expired and the contact still is in a small boundary box around the initial touch position. Reaction arguments: ▶ x: X coordinate of the contact position

Property name	Description
	▶ y: Y coordinate of the contact position

9.8.4.3. Flick gesture

A quick brush of a contact over a surface

Table 9.120. Properties of the **Flick gesture** widget feature

Property name	Description
flickMaxTime	Maximal time in milliseconds the contact may stay in place for the gesture to be recognized as a flick gesture.
onGestureFlick	Reaction that is triggered once the gesture is recognized. Reaction arguments: ▶ speed: relative speed of the flick gesture Speed in pixels/ms divided by flickMinLength/flickMaxTime ▶ directionX: X part of the direction vector of the gesture ▶ directionY: Y part of the direction vector of the gesture
flickMinLength	Minimal distance in pixels a contact has to move on the surface to be recognized as a flick gesture

9.8.4.4. Pinch gesture

Two contacts that move closer together or further apart

Table 9.121. Properties of the **Pinch gesture** widget feature

Property name	Description
onGesturePinchStart	Reaction that is triggered once the start of the gesture is recognized. Reaction arguments: ▶ ratio: Current contact distance to initial contact distance ratio ▶ centerX: X coordinate of the current center point between the two contacts ▶ centerY: Y coordinate of the current center point between the two contacts
onGesturePinchUpdate	Reaction that is triggered when the pinch ratio or center point change
onGesturePinchEnd	Reaction that is triggered once the gesture is finished
pinchThreshold	Minimal distance in pixels each contact has to move from its initial position for the gesture to be recognized. Reaction arguments:

Property name	Description
	▶ Angle: Angle between the line specified by the initial position of the two 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

9.8.4.5. Rotate gesture

Two contacts that move along a circle

Table 9.122. Properties of the **Rotate gesture** widget feature

Property name	Description
onGestureRotateStart	Reaction that is triggered once the start of the gesture is recognized
onGestureRotateUpdate	Reaction that is triggered when the recognized angle or center point changes
onGestureRotateEnd	Reaction that is triggered once the gesture is finished
rotateThreshold	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, 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

9.8.4.6. Path gestures

A shape drawn by one contact is matched against a set of known shapes.

Table 9.123. Properties of the **Path gesture** widget feature

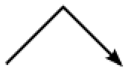
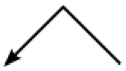
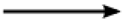
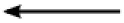
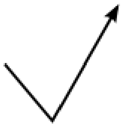
Property name	Description
onPathStart	Reaction that is triggered once a contact moves beyond the minimal box (**pathMinXBox**, **pathMinXBox**). Reaction argument: ▶ gestureId: ID of the path that was matched
onPathNotRecognized	Reaction that triggered when the entered shape does not match. The reaction is only triggered if **onPathStart** has been triggered already.

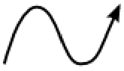
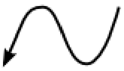
Property name	Description
onPath	Reaction that is triggered when the entered shape matches. The reaction is only triggered if **onPathStart** has been triggered already.
pathMinXBox	X coordinate of the minimal distance in pixels a contact must move so that the path gesture recognizer starts considering the input
pathMinYBox	Y coordinate of the minimal distance in pixels a contact must move so that the path gesture recognizer starts considering the input

9.8.4.6.1. 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 9.124. Path gesture samples configuration included in EB GUIDE

ID	Shape	Description
0		Roof shape left to right
1		Roof shape right to left
2		Horizontal line left to right
3		Horizontal line right to left
4		Check mark

ID	Shape	Description
5		Wave shape left to right
6		Wave shape right to left

9.8.5. Effects

9.8.5.1. 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.

Table 9.125. Properties of the **Border** widget feature

Property name	Description
borderThickness	The thickness of the border in pixels
borderColor	The color that is used to render the border
borderStyle	The style that is used to render the border

9.8.5.2. Coloration

The **Coloration** widget feature colors the widget and its widget subtree. It also affects transparency if the alpha value is not opaque.

Example 9.1.
Usage of the Coloration widget feature

For all colors with RGBA components between 0.0 and 1.0, the algorithm in the **Coloration** widget feature multiplies the current color values of a widget by the `colorationColor` property value. Multiplication is done per pixel and component-wise.

A semi-transparent gray colored by an opaque blue results in semi-transparent darker blue as follows:

```
(0.5, 0.5, 0.5, 0.5) * (0.0, 0.0, 1.0, 1.0) = (0.0, 0.0, 0.5, 0.5)
```

Table 9.126. Properties of the **Coloration** widget feature

Property name	Description
colorationEnabled	If true, coloration is used
colorationColor	The coloration used. Possible values: ▶ Pure ▶ Opaque ▶ White

9.8.6. Layout

9.8.6.1. Absolute layout

The **Absolute 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.

Property name	Description
itemLeftOffset	An integer list that stores the offset from the left border for all child widgets
itemRightOffset	An integer list that stores the offset from the right border for all child widgets
itemTopOffset	An integer list that stores the offset from the top border for all child widgets
itemBottomOffset	An integer list that stores the offset from the bottom border for all child widgets

9.8.6.2. 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.

Name	Description
horizontalChildAlign	The horizontal alignment of child widgets
verticalChildAlign	The vertical alignment of child widgets
layoutDirection	The direction in which the widget is positioned
horizontalGap	The horizontal space between two child widgets
verticalGap	The vertical space between two child widgets

9.8.6.3. 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.

Name	Description
numColumns	The horizontal space between two child widgets
numRows	The vertical space between two child widgets
horizontalGap	The horizontal space between two child widgets
verticalGap	The vertical space between two child widgets

9.8.6.4. 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.

Name	Description
gap	The space between two child widgets, depending on the layout direction
layoutDirection	The direction in which the widget is positioned

9.8.6.5. 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 instantiator widgets to create the child widgets.

Name	Description
layoutDirection	The direction in which the widget is positioned
segments	For horizontal layout direction: the number of rows For vertical layout direction: the number of columns
firstListIndex	The list index of the first visible list item, defined by the widget feature
listLength	The number of list items
scrollValueMin	The minimum scroll value, which is mapped to the beginning of the list
scrollValueMax	The maximum scroll value, which is mapped to the end of the list
scrollValue	The current scroll value
scrollIndex	The base list index the **scrollOffset** property applies to. Scrolling starts at the list item given in the **scrollIndex** property.
scrollOffset	The amount of pixels to scroll the list
scrollOffsetRebase	If the **scrollOffsetRebase** property changes, the current **scrollOffset** is translated to **scrollIndex**. The remaining offset is written to the **scrollOffset** property.
bounceValue	The **bounceValue** property is zero as long as the **scrollOffset** 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 **bounceValue** is added to **scrollOffset**, the scroll position is back in range.
bounceValueMax	The maximum value which **scrollOffset** can move outside the valid scroll range. **scrollOffset** is truncated if the user tries to scroll further.

9.8.6.6. Layout margins

The **Layout margins** widget feature adds configurable margins to a widget that uses the **Flow layout** or the **Absolute layout** widget feature.

Name	Description
topMargin	Margin of the top border
bottomMargin	Margin of the bottom border
leftMargin	Margin of the left border

Name	Description
rightMargin	Margin of the right border

9.8.7. List management

9.8.7.1. List index

The **List index** widget feature adds a list index property to a widget. It is intended to be used in combination with the **List layout** widget feature.

Table 9.127. Properties of the **List index** widget feature

Name	Description
listIndex	The index of the current widget in a list

9.8.7.2. Line index

The **Line index** widget feature adds a line index property to a widget. It is intended to be used in combination with tables.

Table 9.128. Properties of the **Line index** widget feature

Name	Description
lineIndex	The index of the current line in a table

9.8.7.3. Line template index

The **Line template index** widget feature adds a line template index property to a widget. It is intended to be used in combination with instantiator widgets.

Table 9.129. Properties of the **Line template index** widget feature

Name	Description
lineTemplateIndex	The index of the used line template

9.8.7.4. View port

The **View port** widget feature clips oversized elements at the widget borders. It is intended to be used in combination with container widgets or lists.

Table 9.130. Properties of the **View port** 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

9.8.8. Transformations

Transformations modify location, form, and size of widgets.

The order in which transformations are executed is equal to the order in the widget tree. If multiple transformations are applied to one widget at the same widget tree hierarchy level, the order is as follows:

1. Translation
2. Shearing
3. Scaling
4. Rotation around z-axis
5. Rotation around y-axis
6. Rotation around x-axis

9.8.8.1. Translation

The **Translation** widget feature is used to translate the widget and its subtree. It moves widgets in x, y and z directions.

Table 9.131. Properties of the **Translation** widget feature

Property name	Description
translationEnabled	Defines whether translation is used or not
translationX	Translation on the x-axis
translationY	Translation on the y-axis
translationZ	Translation on the z-axis if widget is a 3D graphic

9.8.8.2. Rotation

The **Rotation** widget feature is used to rotate the widget and its subtree

Table 9.132. Properties of the **Rotation** widget feature

Property name	Description
rotationEnabled	Defines whether rotation is used or not
rotationAngleX	Rotation angle on the x-axis
rotationAngleY	Rotation angle on the y-axis
rotationAngleZ	Rotation angle on the z-axis if widget is a 3D graphic

9.8.8.3. Scaling

The **Scaling** widget feature is used to scale the widget and its subtree

Table 9.133. Properties of the **Scaling** widget feature

Property name	Description
scalingEnabled	Defines whether scaling is used or not
scalingX	Scaling on the x-axis in percent
scalingY	Scaling on the y-axis in percent
scalingZ	Scaling on the z-axis in percent if widget is a 3D graphic

9.8.8.4. Shearing

The **Shearing** widget feature is used to distort widgets in the widget subtree.

Table 9.134. Properties of the **Shearing** widget feature

Property name	Description
shearingEnabled	Defines whether shearing is used or not
shearingXbyY	Shearing amount of x-axis by y-axis
shearingXbyZ	Shearing amount of x-axis by z-axis if widget is a 3D graphic
shearingYbyX	Shearing amount of y-axis by x-axis
shearingYbyZ	Shearing amount of y-axis by z-axis if widget is a 3D graphic
shearingZbyX	Shearing amount of z-axis by x-axis if widget is 3D.
shearingZbyY	Shearing amount of z-axis by y-axis if widget is a 3D graphic

9.8.8.5. Pivot

The **Pivot** widget feature defines the pivot point of transformations which are applied to the widget. If no pivot point is configured, the default pivot point is at (0.0, 0.0, 0.0).

Table 9.135. Properties of the **Pivot** widget feature

Property name	**Description**
pivotX	Pivot point on the x-axis relative to parent widget
pivotY	Pivot point on the y-axis relative to parent widget
pivotZ	Pivot point on the z-axis relative to parent widget if widget is a 3D graphic

10. Installation

10.1. Background information

10.1.1. 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 8 systems, you require administrator rights.

10.1.2. System requirements

Observe the following settings:

Table 10.1. 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 8
Screen resolution	Usage of 2 separate monitors with 1600 x 1200 pixels
Software	Microsoft .NET Framework 4.5.1.

	DirectX 11

Table 10.2. Recommended settings for EB GUIDE SDK

Target platform compiler	Microsoft Visual Studio 2013 or newer
File integration	CMake

10.2. Downloading from EB Command

EB Command is the server from which you are going to download the EB GUIDE product line software.

NOTE

Activate your account

After ordering a product, you receive a mail from sales department. Click the link in the email. Follow the steps to create an account as directed in the email and in the browser, then proceed to log in.

Downloading from EB Command

Prerequisite:

- Your user account is activated.

Step 1
Open a browser and go to https://command.elektrobit.com/command/mod_perl/login.pl.

The EB Command front page opens.

Step 2
To change the language, toggle the language in the lower left corner of the screen.

Step 3
Type in your alias, which is your user name.

Step 4
Type in your password and click the **Login** button.

The main page opens.

Step 5
Select a project, for example `EB GUIDE Studio`. The project overview opens.

Step 6
Select the distribution container in the version you want to download, for example `EB GUIDE Studio Core 6.x`. An overview of all downloadable items open.

Step 7
Select the **Actions** check box beside the file you want to download.

Step 8
Click **Download Selection**.

TIP

Downloading multiple files

If you select multiple files for download, a download package is generated. You are prompted to save the file `CommandDownload<date>.zip` to your local system.

The download starts. To log out from EB Command, click the **Logout** button.

10.3. 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.

10.4. Troubleshooting the installation

10.4.1. Renderer errors

If you are unable to use the renderer in EB GUIDE Studio, use the *Renderer test* in the **Start menu** to evaluate problems. Also make sure that you are using the current driver for your graphics card.

For DirectX 11 renderers, make sure that you have installed the DirectX 11 that is available at Microsoft's website.

10.5. 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

A

animation

An animation is a set of time-dependent functions. These functions can be used to change datapool items and widget properties or to send events. These functions are described either by an animation curve or using EB GUIDE Script.

There are two types of animations: widget animations and view transition animations.
See Also widget animation, view transition animation.

API

Application programming interface

C

communication context

The communication context describes the environment in which communication occurs. Each communication context is identified by a unique numerical ID.

D

datapool

The datapool is a data cache in an EB GUIDE model that provides access to datapool items during run-time. It is used for data exchange between the application and the HMI.

datapool item

Datapool items store and exchange data. Each item in the datapool has a communication direction.

E

EB GUIDE GTF

EB GUIDE GTF is the graphics target framework of the EB GUIDE product line and is part of the EB GUIDE TF. EB GUIDE GTF represents the run-time environment to execute EB GUIDE models on target platforms.

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 HMI model 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

global property

See datapool item.

GUI

Graphical user interface

H

HMI

Human machine interface

M

model element

A model element is an object within an EB GUIDE model, for example a state, a widget, or a datapool item.
See Also EB GUIDE model.

P

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.
property	A property is a name-value pair. The name is used as identifier, the value contains data.

R

resource	A resource is a data package that is part of the EB GUIDE model. Examples for resources are fonts, images, 3D-objects. Resources are stored outside of the EB GUIDE model, for example in files, depending on the operating system.

S

state	A state defines the status of the state machine. States and state transitions are modeled in state diagrams.
state machine	A state machine is a set of states, transitions between those states, and actions.

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 model element with one of the following functions: ▶ Graphical representation ▶ Logical behavior ▶ Data

- Any combination of the three above

Index