Elektrobit

EB GUIDE TF

User manual

Version 6.5.1.137731

Elektrobit Automotive GmbH
Am Wolfsmantel 46
D-91058 Erlangen
GERMANY

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

Legal notice

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

1.1.1. 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 structures
- Integrate customer specific EB GUIDE GTF extensions and HMI applications
- Carry out settings to ensure system integrity within EB GUIDE Studio and on the target device
- 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 framework used and its restrictions
- The generating mechanism that ensures compatibility of an EB GUIDE model and the target system

1.1.2. 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 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

1.1.3. 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 features, EB GUIDE Studio extensions or EB GUIDE GTF extensions
- Write code for additional modules for the following items:
  - Existing EB GUIDE TF modules such as widgets or the shaders
  - Existing EB GUIDE Studio extensions such as additional toolbar buttons

Extension developers have the profound knowledge of the following:

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

1.2. Structure of user documentation

The information is structured as follows:

- Background information

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

- How-to-instruction

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

- Tutorial

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

- Reference

  References provide detailed technological parameters and tables.

- Demonstration

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

1.3. Typography and style conventions

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

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

WARNING 	**Source and kind of problem** What can happen to the software? What are the consequences of the problem? How does the user avoid the problem?

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

NOTE **Important information**

Gives important information on a subject.

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

TIP **Helpful hints**

Gives helpful hints

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

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

All default text is written in Arial Regular font.

Font	Description	Example
Arial italics	to emphasize new or important terms	The *basic building blocks* of a configuration are module configurations.
Arial boldface	for GUI elements and keyboard keys	1. In the **Project** drop-down list box, select Project_A. 2. Press the **Enter** key.
Monospaced font (Courier)	for file names, directory names and chapter names	Put your script in the `function_name\abcdirectory`.
Monospaced font (Courier)	for user input, code, and file directories	`CC_FILES_TO_BUILD =(PROJECT_PATH)\source\network\can_node.c CC_FILES_TO_BUILD += $(PROJECT_PATH)\source\network\can_config.c` The module calls the `BswM_Dcm_RequestSessionMode()` function. For the project name, enter `Project_Test`.
Square brackets [ ]	to denote optional parameters; for command syntax with optional parameters	`insertBefore [<opt>]`
Curly brackets {}	to denote mandatory parameters; for command syntax with mandatory parameters	`insertBefore {<file>}`

Font	Description	Example
Three dots …	to indicate further parameters; for command syntax with multiple parameters	`insertBefore [<opt>…]`
A vertical bar |	to indicate all available parameters; for command syntax in which you select one of the available parameters	`allowinvalidmarkup {on|off}`

This is a step-by-step instruction

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

Prerequisite:

- This line lists the prerequisites to the instructions.

Step 1
An instruction to complete the task.

Step 2
An instruction to complete the task.

Step 3
An instruction to complete the task.

1.4. Naming conventions

In EB GUIDE documentation the following directory names are used:

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

  For example:

  ```
  C:\Program Files\Elektrobit\EB GUIDE Studio 6.5
  ```

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

  For example:

  ```
  C:\Program Files\Elektrobit\EB GUIDE Studio 6.5\platform\win32
  ```

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

  For example:

```
C:\Users\[user name]\Documents\EB GUIDE 6.5\projects\
```

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

2. Safe and correct use

2.1. Intended use

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

2.2. Possible misuse

WARNING **Possible misuse and liability**

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

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

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

3. Support

EB GUIDE support is available in the following ways.

- For community edition:

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

- For enterprise edition:

  Contact us according to your support contract.

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

4. Introduction to EB GUIDE

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

4.1. The EB GUIDE product line

The EB GUIDE product line comprises the following software parts:

- EB GUIDE Studio
- EB GUIDE TF

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

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

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

4.2. EB GUIDE Studio

4.2.1. Modeling HMI behavior

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

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

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

4.2.2. Modeling HMI appearance

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

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

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

4.2.3. Handling data

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

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

Event system
: Events are temporary triggers. Events can be sent to both parties to signal that something specific happens.

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

4.2.4. Simulating the EB GUIDE model

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

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

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

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

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

4.2.5. Exporting the EB GUIDE model

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

4.3. EB GUIDE TF

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

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

- EB GUIDE GTF

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

  EB GUIDE Speech Target Framework is the run-time environment executing speech functionality in the HMI.

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

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

EB GUIDE TF uses the following platform abstractions:

- OS abstraction

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

- GL abstraction

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

- Audio abstraction

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

5. Background information

5.1. Overview of EB GUIDE TF

Within the EB GUIDE product line, EB GUIDE GTF is responsible for the following:

- EB GUIDE GTF executes the content of an exported EB GUIDE model.
- EB GUIDE GTF renders the previews of the views and widgets within EB GUIDE Studio.

Extending EB GUIDE Studio with custom widgets and features also means extending EB GUIDE GTF.

EB GUIDE Studio provides input to EB GUIDE GTF in form of the binary model description files that were generated during export. EB GUIDE GTF runs the EB GUIDE model for the simulation or runs the exported EB GUIDE model on a dedicated target platform. A target platform for EB GUIDE GTF is typically defined by concrete target architecture, operating system and graphics API.

You can connect EB GUIDE Monitor to EB GUIDE GTF which allows observation, manipulation and testing of an EB GUIDE model running on EB GUIDE GTF.

5.1.1. Architecture of EB GUIDE GTF

EB GUIDE GTF is based on a microkernel architecture. This means EB GUIDE GTF consists of a core with base functionality and plugins that enable the execution of EB GUIDE models.

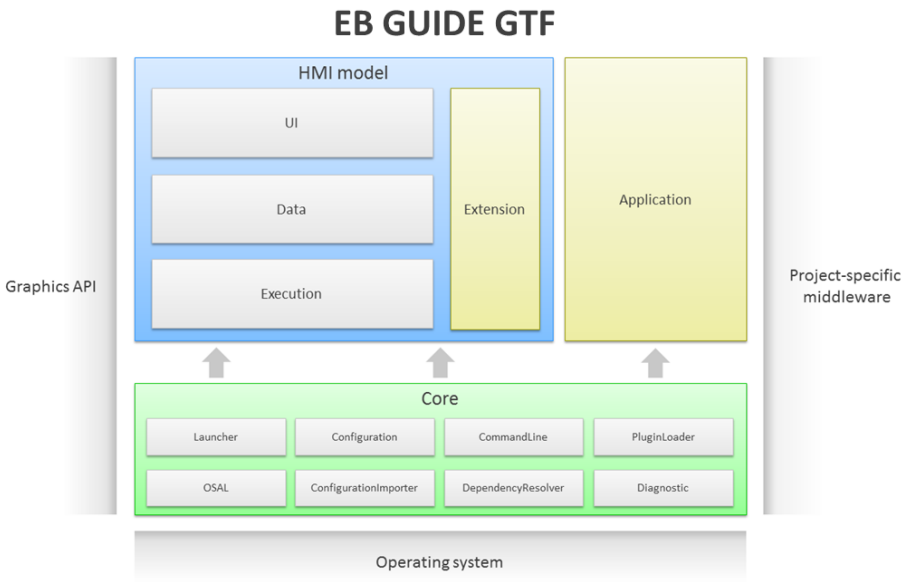


Figure 5.1. EB GUIDE GTF overview

EB GUIDE GTF is embedded in an environment containing an operating system, a graphics API and project-specific middleware.

The core includes the following:

- Abstraction of the dedicated operating system that is called Operating System Abstraction Layer (OSAL)
- The `Launcher` interface that launches EB GUIDE GTF
- `DependencyResolver`, a dependency resolving mechanism
- The `PluginLoader` interface that loads plugins
- The `Configuration` interface that accesses configuration items
- The `ConfigImporter` interface that loads configuration files
- The `CommandLine` interface that parses given command line parameters
- Interfaces for diagnostic purposes

Applications are plugins for EB GUIDE GTF. This means the core loads applications at the start-up before the start of an EB GUIDE model is initiated. Applications are bound to the global accessible API and cannot directly access EB GUIDE model content.

Extensions are plugins for an EB GUIDE model. Extensions are initiated during the EB GUIDE model start-up. Extensions can access model-specific content directly and extend the EB GUIDE model content, for example using widgets, features or EB GUIDE script functions.

The HMI model in this context is the summary of all components that are necessary to run an exported EB GUIDE model. This includes components that manage the user interface, the data and the execution of the dynamic behavior of an EB GUIDE model.

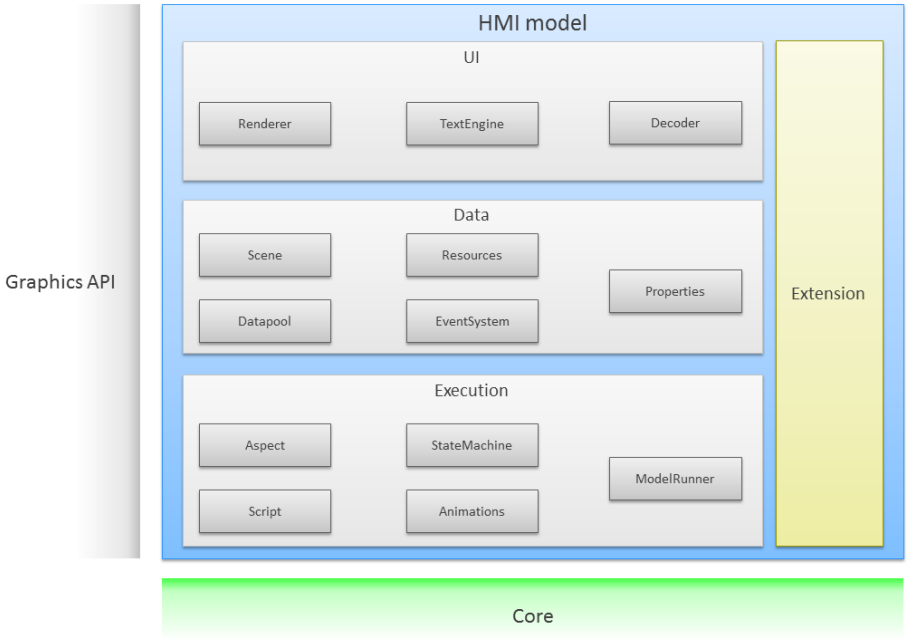


Figure 5.2. HMI model components

The execution components take care of the dynamic behavior of an EB GUIDE model. This means the execution of the following:

- State machine
- EB GUIDE scripts that are part of the datapool, state machine or widget tree
- Animations that describe property changes within the widget tree
- Parts of an EB GUIDE model that are described using languages and skins
- EB GUIDE models for which the `ModelRunner` is responsible

The data components take care of data definition and management.

EB GUIDE GTF considers the following items as data items of an EB GUIDE model:

- Datapool items
- Events that are managed using `EventSystem`
- Resources
- Scenes built up from views and widget trees
- Widget properties

The user interface components take care of functions required to support graphical user interfaces. These components enable EB GUIDE GTF to perform the following:

- The `TextEngine` component supports processing and rendering of internationalized and localized texts.
- The `Decoder` component loads and decodes various types of resources, for example images as PNG, JPEG.
- The `Renderer` component creates the visual representation using the available EB GUIDE GTF graphics abstraction layer.

5.1.2. Life cycle

EB GUIDE TF has two different life cycle concepts:

- The core life cycle drives the start-up of the framework. It provides basic functionality for communication, extension, integration, platform abstraction, tracing, and configuration.
- The EB GUIDE model life cycle drives the start-up of an EB GUIDE model. It provides functionality for model elements, model processing, model diagnosis, and scene rendering.

The EB GUIDE GTF core is responsible for preparing core plugins to run EB GUIDE models.

5.1.2.1. Core life cycle

The core life cycle starts when calling the main interface of EB GUIDE GTF that is part of `Launcher`. This call blocks for the whole life cycle of EB GUIDE GTF.

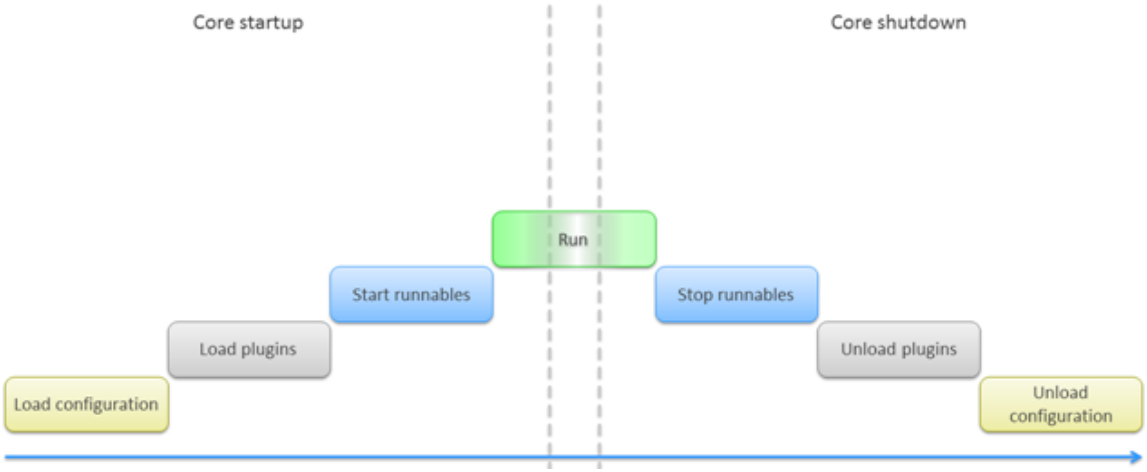


Figure 5.3. EB GUIDE GTF life cycle

The life cycle can be split into three phases.

1. Core start-up phase
2. Run phase
3. Shutdown phase

The core start-up phase is split into the following:

1. Load of the configuration file for the platform
2. Load of the plugins that are configured for the core
3. Start of the `Runnable` interface implementations

For more information, see section 16.7, “`platform.json` configuration file”.

The `Runnable` interface is a part of the `Launcher` component and allows the participation in the core and EB GUIDE model life cycles.

The core shutdown phase is the opposite of the start-up phase and includes the following:

1. Stop of the `Runnable` interface implementations
2. Unload of the plugins
3. Unload of the configuration file for the platform

In between the start-up and shutdown phases, EB GUIDE GTF is running. This means that the EB GUIDE model life cycle is driven here and will stay active up until the core shuts down. For more information, see section 16.4.3, “EB GUIDE Script standard library”.

5.1.2.2. EB GUIDE model life cycle

Each loaded EB GUIDE model has its own life cycle. The life cycle consists of four successive predefined stages and custom stages, see figure below.

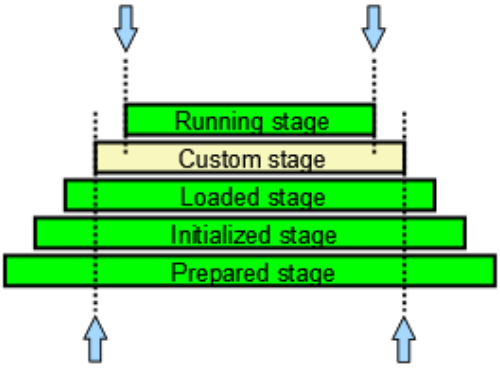


Figure 5.4. Life cycle of an EB GUIDE model with predefined and custom stages

The EB GUIDE TF SDK defines four predefined stages:

- `Prepared` stage: Announcement of widget descriptions etc.
- `Initialized` stage: Information about configurations
- `Loaded` stage: Registration of previously announced script functions and preparation for running the state machine
- `Running` stage: Decoding of resources and the state machine is running

Specific actions are executed during each predefined stage. When the life cycle of the EB GUIDE model enters the next stage, all executed actions of the previous stage are lost.

Each predefined stage has an intermediary internal stage, where the same actions are executed as in the current stage. Thus the executed actions are not lost in the internal stage when the model life cycle enters a new stage. For example in the internal `Running` stage the EB GUIDE model is already running and the actions are already executed. The internal stages function as a backup and therefore the EB GUIDE model cannot fail.

You can add a custom stage at any point during the model life cycle. To see how to add a custom stage, visit our website https://www.elektrobit.com/ebguide/learn/resources/ and go to the section **Downloads**. Click on **SDK examples**, download the file `EB_GUIDE_6_sdk_examples` and have a look at the `StagesPlugin` example.

5.1.3. Deployment 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 EB GUIDE GTF extensions, for example widgets which are not part of EB GUIDE TF.

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

- `GtfStartup.exe`

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

- `GtfCore`

  Shared library which contains all mandatory modules for each GUI project based on EB GUIDE Studio and EB GUIDE TF.

- `GtfRuntime`

  Shared library which contains all mandatory functionality for EB GUIDE GTF based projects.

- `GtfGui`

  Shared library that is responsible for text-/resource handling and is required for EB GUIDE GTF based GUI projects.

- `GtfGuiOpenGLES20`

  Shared library which contains the OpenGL ES 2.0 API renderer implementation.

- `GtfGuiOpenGLES3`

  Shared library which contains the OpenGL ES 3.0 API renderer implementation.

- `GtfGuiDirectX11`

  Shared library which contains the DirectX 11 API renderer implementation.

- `GtfService`

  Shared library which is required to establish TCP connection to an EB GUIDE TF instance, used by for example EB GUIDE Monitor or `GtfIpc`.

- `GtfIpc`

  Shared library which extends GtfService and provides possibility for the inter-process communication (IPC).

- `GtfFileOutput`

Shared library which is used by the trace logging system for file output.

- `GtfProfilingOutput`

  Shared library which is used by the trace logging system for output of profiling data.

- `GtfStderrOutput`

  Shared library which is used by the trace logging system for output to the standard streams.

- `GtfVsDebugOutput`

  Shared library which is used by the trace logging system for output inside of the developement IDE.

5.1.4. The `GtfStartup.exe` executable file

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

5.1.4.1. Command line options

The following table lists command line options available in EB GUIDE GTF for `GtfStartup.exe` and explains their meaning. Undefined commands are ignored.

The general syntax for a command line is as follows: `GtfStartup.exe <option>`

Table 5.1. Command line options

Option	Description
`--version`	Optional parameter. Displays the EB GUIDE GTF version.
`--romfs $ROMFS_FILE_PATH $ROMFS_ROOT_FOLDER`	Optional parameter. In order to run, EB GUIDE GTF needs a file system. Therefore, if the target device has no file system available, RomFS file system (RomFS) is used. If specified, the given RomFS is loaded. `$ROMFS_FILE_PATH` is the path to RomFS container file and `$ROMFS_ROOT_FOLDER` is the root location in the RomFS file system.
`$MODEL_JSON_PATH`	Path to the `model.json` configuration file You can specify either of the following:

Option	Description
	► Only the directory where the file is stored. In this case the file with the name `model.json` is used. ► Only the file name, if the file is stored in the directory in which you called `GtfStartup.exe`. ► The directory where the file is stored and the file name, if the file name is not `model.json`.
`$PLATFROM_JSON_PATH`	Path to the `platform.json` configuration file You can specify either of the following: ► Only the directory where the file is stored. In this case the file with the name `platform.json` is used. ► Only the file name, if the file is stored in the directory in which you called `GtfStartup.exe`. ► The directory where the file is stored and the file name, if the file name is not `platform.json`.
`> <output_file.txt>`	Redirects the output messages to a `.txt` file. The output file is created in the directory in which `GtfStartup.exe` was called.
`> <output_file.txt> 2>&1`	Redirects error messages to a `.txt` file. The output file is created in the directory in which `GtfStartup.exe` was called.

Note that the output messages are not displayed after you entered a command line option. To see the output messages, use the > redirect option.

Example 5.1.
Usage of command line options

To start an EB GUIDE model with the specified configuration files `model.json` and `platform.json`, enter the following:

```
GtfStartup.exe $MODEL_JSON_PATH $PLATFROM_JSON_PATH
```

Note the order: first you specify the path to `model.json`, second you specify the path to `platform.json`.

5.2. EB GUIDE GTF core modules

5.2.1. `Diagnostic` module

The `Diagnostic` module offers the possibility of logging messages from an application using the `Log` interface.

You can retrieve an `Log` instance from the `DependencyResolver` module. Make sure that the retrieved handle is valid. A handle is a managed pointer to the specified data type.

Afterwards the following 2 methods can be called:

- `void vprint(LogLevel level, const char * format, va_list ap)`
- `void print(LogLevel level, const char * format, ...)`

Depending on the message, different log levels can be selected:

- LogError
- LogWarning
- LogInfo

The `Diagnostic` module is responsible for providing a logging interface so than an application can print it's own messages. These messages can be seen either in the EB GUIDE Monitor or in a separate log file.

For instructions how to use the `Diagnostic` module, see chapter 8, “Using the `Diagnostic` module“.

5.2.2. `Configuration` module

The `Configuration` module provides the way of sharing data between modules of a EB GUIDE TF instance. A configuration item is a name value pair. The name must be unique.

The values are of the following types: `BooleanScalar`, `IntegerScalar`, `FloatScalar`, `StringScalar`, `BooleanList`, `IntegerList`, `FloatList`, and `StringList`.

The key naming structure is `namespace.namespace.[...]`, see the following example: `gtf.core.pluginstoload`

For instructions how to use the `Configuration` module, see chapter 7, “Using the `Configuration` module“.

5.2.3. `DependencyResolver` module

The module `DependencyResolver` has the purpose of managing a hierarchy of interfaces, which can be built-in or provided by EB GUIDE GTF plugins.

This module enforces loose coupling between module and is thread safe.

5.2.3.1. Interfaces

All interfaces are found in the namespace `gtf::dependencyresolver`.

The interfaces of the `DependencyResolver` module are:

- `DependencyContainer`: Main interface, used to register and retrieve instances
- `DependencyObject`: Helper class for adding data to a catalog
- `Interface`: Parent for all interfaces that are registered in `DependencyContainer`

For instructions how to use the `DependencyResolver` module, see chapter 9, “Using the `DependencyResolver` module“.

5.3. 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:

- Event system
- Datapool

The advantages of the event system are that no sent information is lost and all events are processed in a defined order. The disadvantage is that high frequency communication may overcharge the processing in communication partner.

The advantages of datapool are that the latest information is always available and high frequency updates cannot overcharge the processing in communication partner. But the disadvantage is that intermediate updates may get lost.

TIP

Purpose 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.

5.3.1. Event system

An event system provides an asynchronous communication mechanism based on events. Events are delivered in exactly the same order they were sent. An event is added to all event queues subscribed to this event. Events can transport values, for example integers, strings or resources.

An event queue is a facility used by a receiver to control, when to process the arrived events. An event queue may limit the number of unprocessed events. New events are ignored by this queue until the number of unprocessed events does not exceed the limit anymore.

EB GUIDE TF uses multiple event systems for different purposes. The most important one for application developers is the global event system. It provides communication between applications and EB GUIDE models. Another event system is used for local communication within each EB GUIDE model.

NOTE

Limited event transition between global and local event systems

Only events of user-specific event groups transferred between global event system and local event systems and vice versa.

In the global event system, string-based identifiers are used. They are created from the EB GUIDE model during export and are unique for all models.

In the local event system, numeric identifiers are used. They are also created from the EB GUIDE model during export, but are unique in one model only.

Both types of identifiers are exported to C++ header files and can be used by application developers to interact with an EB GUIDE model.

5.3.1.1. Event publication

The following steps show the general procedure of event publication:

1. Create a new event at the event system or at an event queue and provide the identifiers for event and group.
2. Optionally add parameter values.

3. Send the event to the event system or to the event queue.

   To identify the sender, you can provide an optional send parameter. This may be useful if you want to identify your own events.

After you send an event, the event is read-only. You cannot change parameter values anymore.

NOTE **The event API is not thread-safe**

Do not access one event from concurrent threads before it was sent, because the API is not thread-safe. After you send an event, the event is read-only and can be used by different threads.

5.3.1.2. Event receipt

The following steps show the general procedure of event receipt:

1. Create an event queue.
2. Register an invoking callback method at the event queue.
3. To subscribe to required events, provide a callback method for specific events, groups of events or all events.
4. After a new event arrives, the invoker callback method is executed. This method should invoke the worker task. To dispatch and process events which are currently in the event queue, use method `dispatch()` within the task. After dispatching the event queue is empty.

NOTE **Ensure asynchronous event processing**

It is not allowed to dispatch and process events directly in the registered callback method. Event dispatching 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.

NOTE **The event queue API is not thread-safe**

Do not access one event queue instance from concurrent threads, because the API is not thread-safe.

Only synchronization between event system and event queue provides thread-safety. Different threads must use different event queue instances.

5.3.2. 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 or image lists.

At EB GUIDE TF runtime each application and each EB GUIDE model is working on its own datapool instance. A datapool instance is empty by default. The owner of a datapool can load items and their initial values from a description file. This datapool description file is part of the exported EB GUIDE model.

EB GUIDE TF synchronizes the values of all datapool items that are shared by different datapool instances. Explicit calls to the API methods of the datapool control the synchronization.

NOTE **The datapool API is not thread-safe**

Do not access one datapool instance from concurrent threads, because the API is not thread-safe. The synchronization between datapool instances provides thread safety. Different datapool instances can be used in different threads.

The common use case is that one datapool user modifies a datapool item and one or more users read the changed values.

NOTE **Possible race conditions on competing modifications in different datapool instances**

If one datapool item is modified in several datapool instances, race conditions may occur.

Datapool items can change as follows:

- Datapool items use the API of this datapool instance.
- The datapool items change during the synchronization of the changes done in other datapool instances.

NOTE **No order of change notifications**

The change notifications do not depend on the order of modifications.

5.3.2.1. Identifiers of datapool items

The datapool API supports two types of datapool item identifiers: a string-based identifier, and a numeric one. During the export of the EB GUIDE model, datapool item identifiers are created and written to a C++ header file. The string-based identifier is unique for all EB GUIDE models, the numeric identifier is unique in one EB GUIDE model only.

5.3.2.2. Synchronisation of datapool items

The following steps show the general procedure for synchronisation of datapool items:

- Register an invoke callback method.
- The API method `commit()` provides all changed values of one datapool instance to all other datapool instances.
- All datapool instances sharing a sub-set of the changed datapool items are invoked by calling the registered invoke callback method.
- The API method `update()` applies the latest available values to the specific datapool instance only.

`commit()` and `update()` affect all changed datapool items at once.

NOTE **Possible loss of intermediate values**

The datapool provides access to the latest available value only.

NOTE **Ensure asynchronous processing of changed datapool items**

It is not allowed to process updates or notifications directly in the registered invoke callback method. Updates and notifications must be processed asynchronously, even if the whole system runs in the same thread.

The callback method must only invoke an asynchronous worker task.

5.3.2.3. Windowed lists

Lists support two operating modes. The default operating mode is based on one data element per list element.

The second operating mode provides the concept of windowed list. Windowed lists are useful to save system resources in case of very big list with thousand or millions of elements, or in cases where access to the data source is very slow.

List 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. Some operations are not supported for windowed lists and will fail, for example insert, append or remove.

An application can switch between both operation modes.

In a typical use case the EB GUIDE model tells the application about the needed parts of a list by using events or datapool items. The application creates windows that cover these parts and provides the requested data for these parts only. To avoid missing data in the EB GUIDE model, define a pre-loading strategy.

5.4. Extensions to EB GUIDE TF

5.4.1. Model elements

In EB GUIDE Studio it is possible to add model elements. Each model element needs a descriptor that is added to EB GUIDE TF. 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 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 WidgetFeature-Descriptor.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.

5.4.1.1. 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.

5.4.1.2. 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.

5.4.2. Widgets

The widget tree is composed of a generic class called `gtf::widgetset::Widgetmodel`, which is implemented with `gtf::propertycontainer::PropertyContainer`.

`gtf::propertycontainer::PropertyContainer` wraps an array of properties and a type ID.

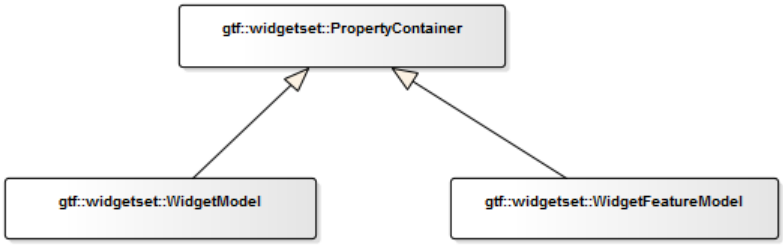


Figure 5.5. Classes that form the generic widget tree

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 using a simple range check.

The renderer scans the type ID information of widgets and properties.

The `gtf::widgetset::Widgetmodel` adds the following:

- Array of child widgets
- Parent pointer
- Optional: pointer to the widget instance
- Array of widget features

- Pointer to cached renderer data to the `gtf::propertycontainer::PropertyContainer`

5.4.3. EB GUIDE Script functions

You 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.

5.4.3.1. 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.

5.4.3.2. 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.

5.4.4. Rendering widgets

Widgets in EB GUIDE GTF can provide their own rendering or drawing routines to generate a visual representation of the widget on the screen. The EB GUIDE TF SDK provides the `WidgetRenderer` interface through which EB GUIDE TF draws the widget.

The following chapters provide information that have to be taken into account in order to implement custom rendering routines for your widget.

To see how to create such customized widgets, visit our website www.elektrobit.com/ebguide and download the file `sdk_examples`. Start with the `TriangleWidget` example that demonstrates rendering a simple widget and hit testing for touch. To learn how to implement a custom offscreen effect shader, refer to the `RippleEffect` example.

5.4.4.1. Rendering

A single instance of the `WidgetRenderer` handles all instances of the widget it hast to render. Within the `WidgetRenderer`, a single instance of a widget is represented by a `RenderObject`.

To draw the `RenderObject`, EB GUIDE TF performs three passes:

1. Traversal pass

   A traversal pass instructs the `WidgetRenderer` to load any rendering relevant data and transformations. Additionally, the widget must propagate rendering relevant information to child widgets. For example, this step may create a new GPU framebuffer to pass to the child widgets. Using this technique, you can realize offscreen effects.

2. Update pass

   An update pass requests GPU buffers, textures, or other GPU related data structures to be updated. This step may rely on interfaces and functions that load image resources directly into textures.

3. Render pass

   Last is the draw or render pass. The `WidgetRenderer` is instructed to assemble one or more draw commands. A draw command is a collection of GPU state and drawing resources. In order to guarantee correct and efficient drawing orders, the `WidgetRenderer` must schedule this draw command for drawing

through the `GraphicsContext` interface. Within the draw command, the drawing algorithm gains access to the `GraphicsExecutionContext` through which you setup the draw state according to your rendering algorithm.

5.4.4.2. Touch handling

Touch handling or hit testing is another important aspect that your widget must consider if it shall react on touch events. EB GUIDE TF requests a `TouchObject` interface from the `RenderObject` that represents your widget in the `WidgetRenderer`. The `TouchObject` interface has to implement the intersection method that tests a screen position against your widget.

5.5. Simulation with EB GUIDE Monitor

When simulating the EB GUIDE model, 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.

For more information on EB GUIDE Monitor and for instructions, see EB GUIDE Studio manual.

5.5.1. Communication with EB GUIDE TF

EB GUIDE Monitor communicates with a running EB GUIDE TF instance through a TCP/IP connection. The connection is implemented in the `GtfService` plugin for the target framework.

5.6. Android APK

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

5.6.1. 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.

Table 5.2. Minimal requirements

Architecture	ARMv7
Platform	EB GUIDE TF: Android 5.0 (API Level 21)

5.6.2. Features of the EB GUIDE TF APK

Table 5.3. Features of the EB GUIDE TF APK

Feature	**Description**
Life cycle management	EB GUIDE TF supports Android life cycle management.
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.

5.6.3. Description of the EB GUIDE TF APK files

- `EB GUIDE Launcher.apk`

  The EB GUIDE Launcher starts EB GUIDE TF and displays the exported EB GUIDE model.

  If you start the EB GUIDE Launcher it displays the exported EB GUIDE model that was selected last by EB GUIDE Model Chooser.

- `EB GUIDE Model Chooser.apk`

  The EB GUIDE Model Chooser provides a user interface to select an exported EB GUIDE model that is executed on the Android device.

  By selecting an exported EB GUIDE model, the EB GUIDE Launcher is started with the corresponding model.

  Clicking the **Info** button displays the directory where exported EB GUIDE models are stored for the 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 5.6.5, “Directory for EB GUIDE models”.

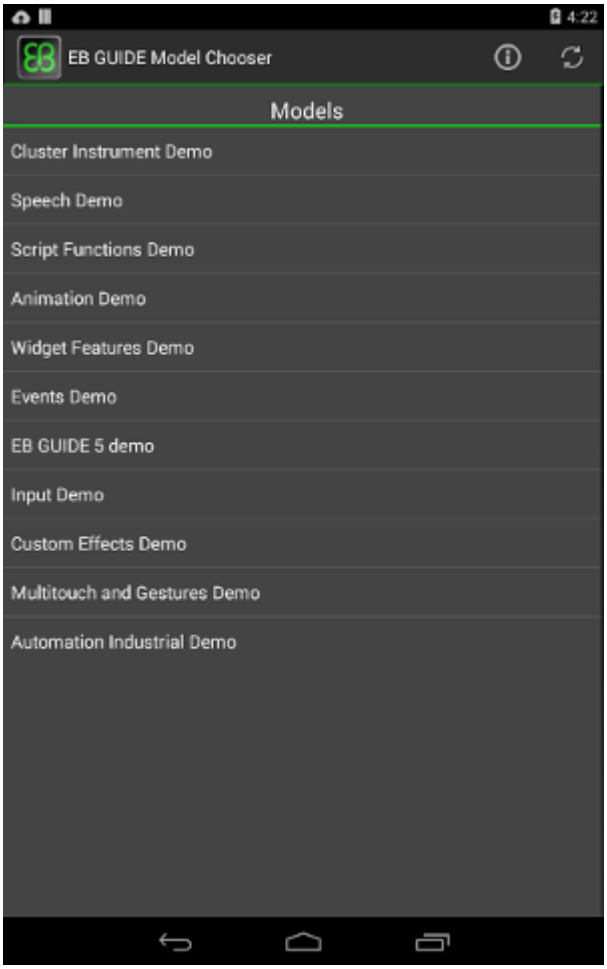


Figure 5.6. EB GUIDE Model Chooser

5.6.3.1. 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 GTF extensions, 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.

5.6.3.2. 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.

5.6.4. Android life cycle management

The Android life cycle management is an optimization implemented by the Android operating system. If an application moves to the background, Android releases all graphics resources and makes the resources available for the application that moves to the foreground. An application is responsible for recreating the resources when it moves to the foreground.

5.6.5. Directory for EB GUIDE models

EB GUIDE models are stored in the `com.elektrobit.guide_model_chooser/files` directory that is located on the primary external storage directory. Application-related files are stored there permanently. One directory is required per EB GUIDE model.

NOTE

`com.elektrobit.guide_model_chooser/files` directory

After you installed EB GUIDE Model Chooser and EB GUIDE Launcher the `com.elektrobit.guide_model_chooser/files` directory does not yet exist. The directory is created when you start EB GUIDE Model Chooser for the first time. Alternatively, you can also add the directory manually.

The EB GUIDE Model Chooser searches only in the primary external storage directory. Usually Android devices have their primary external storage on a portion of the internal storage. Make sure you copy the files to the correct place.

Examples for primary external storage directory:

- For a Samsung Galaxy S3 device with Android 5.0 that is connected to a PC with Windows 7, the directory is `Computer\GT-I9300\Phone\Android\data\com.elektrobit.guide_model_chooser\files`.
- For a Nexus 7 device with Android 5.0 that is connected to a PC with Windows 7, the directory 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 directory for the EB GUIDE TF configuration file `model.json`. The parent directory for each start-up configuration is displayed as the EB GUIDE model name.

5.6.6. 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.

EB GUIDE supports the developer by providing events that indicate layout changes in the visible screen area.

Example 5.2.
Examples for layout handling

- When a mobile device is rotated, the GUI has to adapt according to the rotation.
- When a virtual keyboard is displayed on the screen, the GUI has to adapt to the new element.

5.7. Integration

5.7.1. 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 reference the `libcpp.so` library into the EB GUIDE TF process, because the EB GUIDE TF process uses libraries that reference 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.

5.7.2. 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.

5.7.3. 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.

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.

To create a RomFS container, refer to chapter 15, “Creating a read-only file system (RomFS) container“.

5.8. Programming concept

5.8.1. 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 5.8.2, “Functors” explains the usage and behavior of `GtfFunctorX`.

5.8.2. 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.

5.8.2.1. 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.

5.8.2.2. `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.

5.8.2.3. 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.

5.9. Inter-process communication

The `GtfIpc` plugin extends the EB GUIDE GTF with the inter-process communication (IPC) mechanism that connects multiple EB GUIDE GTF instances and exchanges information, for example datapool items and events.

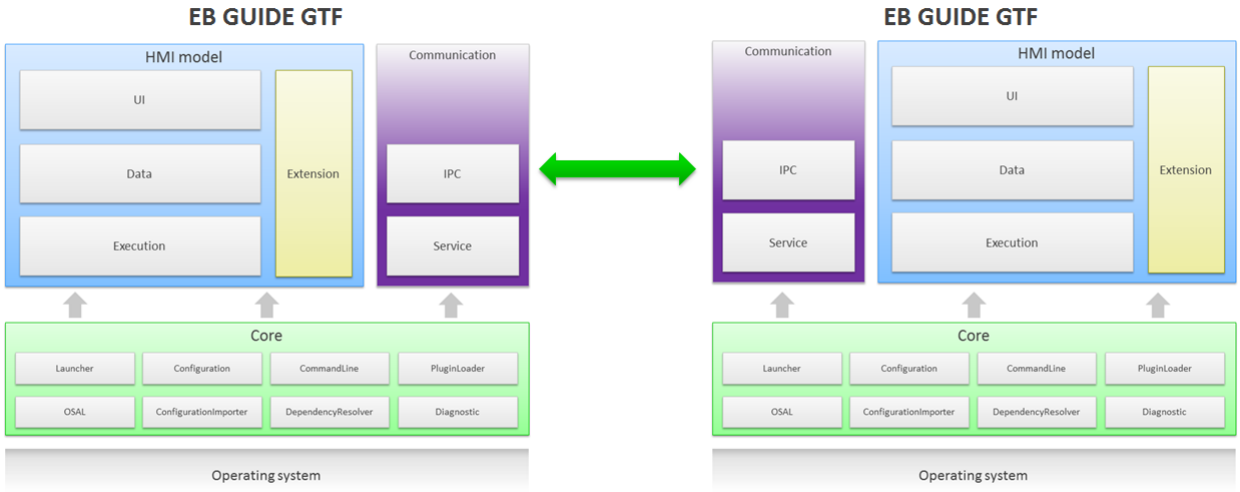


Figure 5.7. Connecting EB GUIDE GTF instances with IPC

The IPC has a server-client architecture. The implementation supports several clients. Clients and server can be restarted. If a server is restarted, it will be initialized with the default datapool item values and update the clients with the same values.

The server instance is responsible for the following:

- Connecting clients
- Mastering datapool data

Once the IPC network is established, the server forwards the changes in the exchange information to the other clients.

Example 5.3.

An IPC network is created by connecting a server and several clients. All these connected nodes contain the current time information in a datapool item. If a client updates the datapool item and therefore the time, all the other nodes present in the IPC network will get updated with the new value.

By default, all datapool items and events contained in a nodes' model are shared. You can configure the items which are shared as follows:

- Set the `"gtf.datapool.descriptionfile"` configuration item.

  If the configuration item is set, the IPC initializes its datapool with the file set in this configuration item.
- Set the start-up parameter that refers to the `model.json` path.

Data exchange within the IPC has the default datapool and event system behavior. For more information see section 5.3.2, “Datapool” and section 5.3.1, “Event system”.

In case of datapool, if one node changes a datapool item five times, other nodes will only get the latest updated value of the datapool item.

In case of events, if one node fires an event e.g. five times, other nodes will have the event triggered also five times.

The following tables show the examples for connection modes configuration:

NOTE

Structure of configuration items

Basic structure of an item is as follows: `<configuration_item> : <value>`

String values must be enclosed with `" "`.

Values in the tables below are just example values. Adapt the values according to your projects' needs.

Broadcast

Table 5.4. Server configuration

Configuration item and value	Description
`"gtf.ipc.role": "server"`	The role in the IPC network
`"gtf.ipc.discovery.mode": "broadcast"`	The connection mode
`"gtf.ipc.discovery.network": "255.255.-255.255"`	The broadcast network address
`"gtf.ipc.discovery.port": 4712`	The broadcast port

Table 5.5. Client configuration

Configuration item and value	Description
`"gtf.ipc.role": "client"`	The role in the IPC network
`"gtf.ipc.discovery.mode": "broadcast"`	The connection mode
`"gtf.ipc.discovery.network": "255.255.-255.255"`	The broadcast network address
`"gtf.ipc.discovery.port": 4712`	The broadcast port
`"gtf.ipc.client.timeout": 5000`	The time interval in milliseconds between the connection retry attempts of the client

Multicast

Table 5.6. Server configuration

Configuration item and value	Description
`"gtf.ipc.role": "server"`	The role in the IPC network
`"gtf.ipc.discovery.mode": "multicast"`	The connection mode
`"gtf.ipc.discovery.network": "230.0.0.-1"`	The multicast network address
`"gtf.ipc.discovery.port": 4712`	The multicast port

Table 5.7. Client configuration

Configuration item and value	Description
`"gtf.ipc.role": "client"`	The role in the IPC network
`"gtf.ipc.discovery.mode": "multicast"`	The connection mode
`"gtf.ipc.discovery.network": "230.0.0.-1"`	The multicast network address
`"gtf.ipc.discovery.port": 4712`	The multicast port
`"gtf.ipc.client.timeout": 5000`	The time interval in milliseconds between the connection retry attempts of the client

Direct

Table 5.8. Server configuration

Configuration item and value	Description
`"gtf.ipc.role": "server"`	The role in the IPC network
`"gtf.ipc.discovery.mode": "direct"`	The connection mode
`"gtf.ipc.discovery.network": "127.0.0.-1"`	Localhost
`"gtf.servicemapper.port": 60001`	If the configuration is generated, the port might already be set. You need to set the value only if it does not yet exist.

Table 5.9. Client configuration

Configuration item and value	Description
`"gtf.ipc.role": "client"`	The role in the IPC network
`"gtf.ipc.discovery.mode": "direct"`	The connection mode
`"gtf.ipc.discovery.network": "x.x.x.x"`	The network address of the server
`"gtf.ipc.discovery.port": 60001`	The port should match the `"gtf.servicemapper.port"` value of the server

Configuration item and value	Description
`"gtf.ipc.client.timeout": 5000`	The time interval in milliseconds between the connection retry attempts of the client

For more information on `platform.json`, see section 16.7, “`platform.json` configuration file”.

6. Executing an EB GUIDE model on target framework

6.1. Configuring profiles

EB GUIDE Studio offers the possibility to create different profiles for an EB GUIDE model. In profiles you set up a configuration for a specific target platform, for example Win32, Linux or QNX. During export the EB GUIDE TF start-up configuration file `model.json` is generated from profiles.

You use profiles to do the following:

- Add custom configuration items
- Configure internal and user-defined libraries to load
- Configure a scene
- Configure a renderer

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

For details on how to configure profiles, see *Configuring profiles* in the EB GUIDE Studio manual.

6.2. Exporting an EB GUIDE model

For details on how to export an EB GUIDE model, see *Exporting an EB GUIDE model* in the EB GUIDE Studio manual.

6.3. Configuring the system start

Configuring the system start

Prerequisite:

- Profiles are configured according to the project requirements.
- An EB GUIDE model is exported.

Step 1
Copy the EB GUIDE TF version for your platform to your target platform. This includes the executable file and all EB GUIDE TF plugins that are required by your `model.json` and `platform.json` configuration.

Step 2
Copy the exported EB GUIDE model. Make sure the paths in `model.json` refer to the EB GUIDE model's files and its relative paths are correct.

Step 3
Start EB GUIDE TF on the target device.

For details on how to start EB GUIDE GTF from the command line, see section 5.1.4.1, “Command line options”.

You have configured EB GUIDE TF for your target platform.

7. Using the Configuration module

The Configuration module is responsible for building, retrieving and adding elements to a map that contains configuration items. The main interface of the Configuration module is the Settings interface. An instance can be retrieved from the DependencyResolver module. Make sure that the retrieved handle is valid. A handle is a managed pointer to the specified data type.

NOTE

Thread safety of the Configuration module

Adding and retrieving operations of the Configuration module are thread-safe.

7.1. Retrieving an item from a Configuration module

Retrieving an item from a Configuration module

Prerequisite:

- You know the data type of the configuration item that you want to retrieve, for example StringScalar.

Step 1
Declare a handle that represents a managed pointer to a value.

Example: `configuration::StringScalarHandle value;`

Step 2
Store in this variable the result of the call to getItem where you specify the key as a parameter. When you call getItem, the template parameter must be the same type as the handle.

Example: `value = settings->getItem<gtf::configuration::StringScalar>(key);`

If the item does not exist in the map, getItem returns null. If the key is not found in the current map then it is searched for recursively in all its ancestors.

7.2. Adding a scalar item to a Configuration module

Adding a scalar item to a Configuration module

Step 1
To retrieve a builder from the Settings interface, use getBuilder.

Example: `configuration::StringScalarBuilderHandle stringBuilder = settings->getBuilder<configuration::StringScalarBuilder>();`

Step 2
Specify the value of the scalar item.

Example: `stringBuilder->set(model.getString().getUTF8());`

Step 3
To add the scalar item to the map, use addItem.

Example: `settings->addItem(stringBuilder->build(), key);`

If the key already exists or the handle is null, addItem returns false.

7.3. Adding a list item to the Configuration module

Adding a list item to the Configuration module

Step 1
To retrieve a Builder interface from the Settings interface, use getBuilder.

Example: `configuration::FloatListBuilderHandle listBuilder = settings->getBuilder<configuration::FloatListBuilder>();`

Step 2
To add values to the list, use add.

Example: `FloatListBuilder->add(1.0f); listBuilder->add(2.0f); FloatListBuilder->add(3.0f);`

Step 3
To add the item to the map, use addItem.

Example: `settings->addItem(listBuilder ->build(), key);`

If the key already exists or the handle is null, addItem returns false.

7.4. Creating a path value using the Configuration module

Create a path value using the Configuration module

Prerequisite:

- A StringScalar item is added to the Configuration module. For more information, see section 7.2, "Adding a scalar item to a Configuration module".

Step 1
Declare a handle to a PathValue variable.

Example: `configuration::PathValueHandle pathHandle;`

Step 2
Retrieve the previously added StringScalar as a PathValue.

Example: `pathHandle = settings->getItem<configuration::PathValue> ("stringItemName");`

Step 3
Specify the base value for the path as a const char*.

Example: `pathHandle->setBasePath("basePath");`

Step 4
Retrieve the resulting value.

Example: `const char* stringPath = pathHandle->get();`

7.5. Adding elements of a .json file to the Configuration module

Adding elements of a .json file to the Configuration module

The ConfigurationImporter module parses a .json file and adds it to the Configuration module.

Prerequisite:

- A valid handle for the ConfigurationImporter module is available.
- An instance of the Settings interface exists.

Step 1
To add elements of a .json file to the Configuration module, use parse.

Example: `bool result = configurationImporter->parse($JSON_PATH);`

If the file is missing or invalid, the method returns false.

8. Using the Diagnostic module

8.1. Logging messages

Logging messages

Prerequisite:

- You wrote an EB GUIDE GTF extension.

Step 1
Declare a handle that represents a managed pointer to a Log interface.

Example: `gtf::diagnostic::LogHandle logger;`

Step 2
Store in this variable the result of the get method. When calling get, the template parameter must be the Log interface.

Example:
```
gtf::diagnostic::LogHandle logger = container->get<gtf::diagnostic::Log>();
```

If the interface does not exist, get returns null.

Step 3
Use the retrieved interface to print out the message.

Example: `logger->print(diagnostic::Log::LogInfo, "Hello world!");`

8.2. Redirecting your logged messages into a file

For instructions how to redirect the logged messages into a file, see section 5.1.4.1, “Command line options”.

9. Using the DependencyResolver module

When using the DependencyResolver module, note the following:

- All interfaces that have to be registered in the container must inherit from public dependencyresolver::Interface.
- The macro GTF_DEFINE_INTERFACE_NAME(InterfaceName) must be added in the class header file.

  For example:

```
class UserInterface : public dependencyresolver::Interface
public:
 virtual void fooBar() = 0;
};
typedef gtf::smartptr::RefCountedPtr<UserInterface> UserInterfaceHandle;
GTF_DEFINE_INTERFACE_NAME(UserInterface);
```

9.1. Retrieving an item from DependecyContainer

Retrieving an item from DependecyContainer

Prerequisite:

- A handle to an item was registered to the container.

Step 1
Declare a handle variable.

Example: `UserInterfaceHandle value;`

Step 2
The call to the get method returns a handle. Store this handle in the variable.

Example: `value = container->get<UserInterface>();`

If the item does not exist in the container, get returns NULL.

NOTE

Lookup scope

If the lookup scope is not specified, by default the instances of the requested interface will be searched recursively in all its ancestors.

To search only in the local lookup scope use `get<Type>(LookupScope::Local);`.

9.2. Retrieving all instances registered to an interface

Retrieving all instances registered to an interface

Step 1
Declare an array of handles.

Example: `gtf::container::Array<UserInterfaceHandle> instances;`

Step 2
Use it to store the specified instances from the container using `getAll` method.

Example: `container->getAll(instances);`

Step 3
Use an iterator to navigate through the array.

Example:

```
for (gtf::container::Array<UserInterfacelHandle>::Iterator it = instances.begin();
    it != instances.end(); ++it)
{
//process data here...
}
```

If no items are found, `getAll` will return `FALSE` and the array will be empty.

NOTE **Lookup scope**

If the lookup scope is not specified, by default the instances of the requested interface will be searched recursively in all its ancestors.

To search only in the local lookup scope use `getAll<Type>(instances, LookupScope::Local);`.

9.3. Registering an instance at the container

Registering an instance to the container

Prerequisite:

- A valid handle to the item must exist.

Step 1
To register an instance of an interface, add the new instance to the container.

Example: `container->registerInstance<UserInterfaceHandle >(instance);`

Step 2
To register the same instance for multiple interfaces, add the same instance for each interface to the container.

Example:

```
class InterfaceA : public Interface
{
};
class InterfaceB : public InterfaceA
{
};
typedef gtf::smartptr::RefCountedPtr<InterfaceB> InterfaceBHandle;
InterfaceBHandle interfaceB= new InterfaceB();
container->registerInstance<InterfaceA>( interfaceB);
container->registerInstance<InterfaceB>( interfaceB);
```

If the item cannot be registered, `registerInstance` will return `FALSE`.

NOTE It is not possible to register the same instance with the same interface twice.

Registering an instance using `create` function

Step 1
Create a function that returns a handle to the instance you want to register.

Example:

```
static dependencyresolver::UserInterfaceHandle createInstance
(dependencyresolver::DependencyContainerHandle & container)
{
    if (!container.valid())
    {
        return dependencyresolver::InterfaceHandle();
    }
    UserInterfaceHandle instance = new UserInterfaceImplementation();
    return instance;
}
```

Step 2
Create a catalog instance.

Example: `gtf::dependecyresolver::DependencyCatalog catalog;`

Step 3
Extend the catalog with the previous function:
`Dependencyresolver::extendCatalog<UserInterface>(catalog, createInstance);`

NOTE

DependencyObjects

A catalog is an array of DependencyObjects.

For more information see API documentation module DependencyResolver.

9.4. Unregister an instance

Unregister an instance

Prerequisite:

- A handle to an item was registered to the container.

Step 1
Unregister the instance from the container.

Example: `container->unregisterInstance<UserInterface>(instance);`

If the item cannot be unregistered, `registerInstance` will return `FALSE`.

9.5. Registering a catalog

The following instruction shows you how to add the content of a catalog to a container.

Registering a catalog

Step 1
Declare a catalog.

Example: `Example: gtf::dependecyresolver::DependencyCatalog catalog;.`

Step 2
Register the catalog in the container.

Example: `container->registerCatalog(catalog);`

If the call to `registerCatalog` fails, it will return `FALSE`.

9.6. Unregistering a catalog

Unregistering a catalog

Step 1
Unregister the catalog from the container.

Example: `container->unregisterCatalog(catalog);`

If the call to `unregisterCatalog` fails, it will return `FALSE`.

9.7. Creating a container

Creating a container

Prerequisite:

- A handle to an item was registered to the container.

Step 1
Create a container.

Example: `DependencyContainerHandle childContainer = container->createContainer();`

`retrieve` and `retrieveAll` are used internally by `get` and `getAll`.

10. Using the EB GUIDE TF plugin mechanism

10.1. Creating an EB GUIDE TF plugin

Two types of EB GUIDE TF plugins exist: extensions, and applications. The following instructions are valid for both types.

Creating an EB GUIDE TF plugin

The following section explains the general workflow for integrating EB GUIDE TF into your build system on Windows platforms. Find the instructions for each step in the sections below.

Step 1
Adjust the `model.json` to load the EB GUIDE TF plugin. For details see section 10.2, “Adding an EB GUIDE TF plugin”.

Step 2
Write an EB GUIDE TF plugin. For details see section 10.3, “Writing an EB GUIDE TF plugin”.

Step 3
Copy the resulting `.dll` file. For details see section 10.4, “Copying the resulting `.dll` file”.

Step 4
Start the simulation with `GtfStartup.exe`. For details see section 10.5, “Starting the simulation with `GtfStartup.exe`”.

10.2. Adding an EB GUIDE TF plugin

NOTE

Adding an EB GUIDE TF plugin

To add an EB GUIDE TF plugin, you can do either of the following:

- Add the EB GUIDE TF plugin to an EB GUIDE model in EB GUIDE Studio.
- Modify the `model.json` file of an exported EB GUIDE model.

Adding an EB GUIDE TF plugin to an EB GUIDE model

In order to use a newly written EB GUIDE TF plugin such as a widget or a widget feature in an EB GUIDE model, it is necessary to add the EB GUIDE TF plugin to the EB GUIDE model.

Prerequisite:

- An EB GUIDE TF plugin exists.

Step 1
Compile the EB GUIDE TF plugin.

Step 2
Copy the resulting file to the `$GUIDE_PROJECT_PATH/resources` directory.

Step 3
Start EB GUIDE Studio and open an EB GUIDE Studio project.

Step 4
Go to the project center and click **Configure** > **Profiles**.

Step 5
Select the required profile.

Step 6
Open the **Model** tab.

Step 7
To load your EB GUIDE TF plugin, integrate the following code into the `model.json`:

```
{
 "gtf": {
     "model": {
      "pluginstoload": [
             "resources/myplugin"
         ]
     }
 }
}
```

`myplugin` is the name of the example EB GUIDE TF plugin.

Step 8
Save the project and close EB GUIDE Studio.

Step 9
Open the project again.

You have added an EB GUIDE TF plugin to an EB GUIDE model.

Step 10
Export the EB GUIDE model.

Adding an EB GUIDE TF plugin without EB GUIDE Studio

The following instruction shows you how to modify the `model.json` file directly so that it loads an EB GUIDE TF plugin.

Prerequisite:

- An EB GUIDE model is exported.

Step 1
Navigate to the exported EB GUIDE model.

Step 2
Open the `model.json` file with a text editor.

Step 3
To load your EB GUIDE TF plugin, include the following code:

```
{
 "gtf": {
     "model": {
      "pluginstoload": [
             "myplugin"
         ]
     }
 }
}
```

`myplugin` is the name of the example EB GUIDE TF plugin.

Step 4
Save the `model.json` file.

10.3. Writing an EB GUIDE TF plugin

Writing an EB GUIDE TF plugin

To enable your EB GUIDE TF plugin to react on datapool and event updates it is necessary to include the corresponding files.

In order to access an example, visit our website https://www.elektrobit.com/ebguide/learn/resources/. Download the file `sdk_examples` and open the `BasicPlugin` example.

Prerequisite:

- An EB GUIDE model is exported.
- The `model.json` file is adapted.
- A new directory is created, for example `C:\plugin\myplugin`.

Step 1
Navigate to the directory you prepared for the plugin, for example `C:\plugin\myplugin`.

Step 2
Create a file named `myplugin.cpp`.

Step 3
Open the `myplugin.cpp` file and write an EB GUIDE TF plugin.

Find a description of all relevant classes and methods in the EB GUIDE TF API.

Step 4
Compile `myplugin.cpp`.

The result is a `.dll` file `myplugin.dll`.

10.4. Copying the resulting `.dll` file

Copying the resulting `.dll` file

Prerequisite:

- An EB GUIDE model is exported.
- The `model.json` file is adapted.
- A new directory is created, for example `C:\plugin\myplugin`.

- A compiled EB GUIDE TF plugin is created.

Step 1
Navigate to the directory where you saved the `myplugin.dll` file, for example `C:\plugin\myplugin`.

Step 2
Copy `C:\plugin\myplugin` to the directory where you exported the EB GUIDE model, for example `C:\projects\example_project`.

10.5. Starting the simulation with `GtfStartup.exe`

Starting the simulation with `GtfStartup.exe`

On Windows platforms you can start the simulation directly using `GtfStartup.exe`.

Prerequisite:

- An EB GUIDE model is exported.
- The `model.json` file is adapted.
- A new directory is created, for example `C:\plugin\myplugin`.
- A compiled EB GUIDE TF plugin 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 `model.json` 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.

11. Extending EB GUIDE Script with foreign functions

This section provides information about the EB GUIDE Script function example, in this case a function returning the sum of two integer values.

In order to access this example, visit our website https://www.elektrobit.com/ebguide/learn/resources/. Download the file `sdk_examples` and open the `ScriptFunction` example.

For more information about EB GUIDE Script functions, see section 5.4.3, “EB GUIDE Script functions”.

12. Adding widgets and widget features

This section provides information about EB GUIDE GTF extension examples such as widgets and widget features. The widget in the first example adds new functionality, while the second example is a widget feature which makes a rectangle change its color when focused.

NOTE

Example widgets for EB GUIDE

To see how customized widgets can be created and added to an EB GUIDE Studio project, download pre-designed widgets from our website https://www.elektrobit.com/ebguide/learn/resources/.

12.1. Example of the extended container widget

The example implements a container widget that controls the visibility of its child widgets. The `displayStatus` property defines which child widgets are displayed: all, none, or only the first.

In order to access this example, visit our website https://www.elektrobit.com/ebguide/learn/resources/. Download the file `sdk_examples` and open the `ExtendedContainerWidget` example.

12.2. Example of the widget feature for focus behavior of rectangles

The example adds a widget feature to a rectangle widget. The example widget feature is called `focusRectColor` and has a property `focusedColor`. The purpose of the widget feature is to change the background color of a rectangle to the value of the `focusedColor` property when the rectangle is focused. To enable the focus widget feature, add the widget feature **State focused**.

In order to access this example, visit our website https://www.elektrobit.com/ebguide/learn/resources/. Download the file `sdk_examples` and open the `FocusedWidgetFeature` example.

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

For background information on Android APK see section 5.6, “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.

13.1. 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 executes 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 `$GTF_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 more information, see EB GUIDE Studio manual.

Step 5
Copy the whole directory that was exported by EB GUIDE Studio to your Android device. For information where to store the EB GUIDE models see section 5.6.5, “Directory for EB GUIDE 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.

13.2. Creating your own Android APK using the template

Creating your own Android APK using the template

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

Step 2
Add the library `$GTF_INSTALL_PATH/platform/android/bin/GtfBridge.jar` to the Eclipse or IntelliJ workspace.

Step 3
Optional: To change the location of the EB GUIDE model and the libraries, edit the implementation of the template `TemplateActivity.java`.

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

Step 4
In `$GUIDE_INSTALL_PATH/projects/code/apk/AndroidAppTemplate` create the folder structure `libs/armeabi`.

Step 5
Copy the Android SDK binaries delivered with EB GUIDE GTF to the directory `$GTF_INSTALL_PATH/platform/android/apk/libs/armeabi`.

Step 6
In `$GUIDE_INSTALL_PATH/projects/code/apk/AndroidAppTemplate` create the folder `assets`.

Step 7
Copy the file `platform.json` delivered with EB GUIDE GTF to the directory `$GTF_INSTALL_PATH/platform/android/apk/assets`.

Step 8
Copy an EB GUIDE model to the default external file directory of the application. The default directory implemented in the template activity is `/data/android/com.elektrobit.gtf_android_template.package`.

Step 9
Deploy and launch the application in Eclipse or IntellJ on the target device or use an Android virtual device (AVD).

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

13.3. 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 13.2, “Creating your own Android APK using the template”. You can integrate additional EB GUIDE GTF extensions that are useful for a project. Save the additional EB GUIDE GTF extensions 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 5.6.3.1, “Released APK and custom APK”.

Creating your own Android APK from scratch

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

Step 2
Add the library `$GTF_INSTALL_PATH/platform/android/bin/GtfBridge.jar` to your Eclipse or IntelliJ workspace.

Step 3
In the project workspace, create a directory `libs/armeabi` and copy the `.so` files from `$GTF_INSTALL_PATH` into the directory.

Step 4
In the project workspace, create a directory `assets` and copy the file `platform.json` from `$GTF_INSTALL_PATH` into the directory.

Step 5
Add the import `com.elektrobit.gtf.android.GtfActivity`.

Step 6
Create an activity that extends `GtfActivity`.

Step 7
Adapt the following methods:

Step 7.1
To set the model path, call the method `protected String getModelPath()`.

Return either `getStandardModelPath()` for the default path, or a string with the path of the EB GUIDE model files.

Step 7.2
To load additional libraries, add their names to the string array, which is the return value of `protected String[] getAdditionalNativeLibs()`. If you do not use any additional libraries, return `null`.

Step 8
In the manifest, modify or add the following code:

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

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

<uses-sdk android:minSdkVersion="21"/>
```

Step 9
Create a keystore file for a release build.

Step 10
Create the release build with Ant tooling 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 for a separation of functionality

Place the activity 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.

14. 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.

The memory report is printed to the standard output streams. On Microsoft Windows operating systems, outputs to console are deactivated. To see the stream outputs, pipe the streams into files. EB GUIDE Monitor will also show the memory report in the **Logger** component. On other operating systems that are POSIX compatible (like Linux) the report is shown on console.

Configuring a memory report

You configure a memory report by adding a configuration message to the `model.json` 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
Select the **Platform** tab.

Step 4
Enter the following code:

```
{
 "gtf": {
     "diagnostic": {
      "memory": {
             "interval":5000
         }
     }
 }
}
```

You configured the memory report to be printed with an interval of 5 seconds.

15. 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 RomFsCreate in the `tools\RomFsTools` sub-directory of your EB GUIDE GTF SDK directory.

Step 3
Run RomFsCreate without any parameters.

The following usage directions are displayed:

```
Invalid command line options
Usage:
RomFsCreate.exe [OPTIONS] SOURCE_DIRECTORY [IMAGE_NAME]
Options:
--create-c-file BASE_NAME:
        Creates a source (.c) file that contains the romfs data and a header (.h) file
        the files will have the names: BASE_NAME.c and BASE_NAME.h
--output-dir TARGET_DIRECTORY:
        define the location were the .romfs file will be created
--max-size N
        specify the maximum size of the container
-h or --help    display this help
```

For usage options see the list below.

You have the following options:

Create a RomFS container
`RomFsCreate.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

`RomFsCreate.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, `RomFsCreate` 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

`RomFsCreate.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

`RomFsCreate.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.

16. References

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

16.1. Android events

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

Table 16.1. Android events

Event ID	Name	Description
1	`RendererEnabled`	Is sent by the application when Android life cycle management stops or starts the renderer Parameters: ► `enabled`: If true, the renderer is enabled. If false, the renderer is set to sleep mode.
2	`setKeyboardVisibility`	Is sent by the EB GUIDE model if a virtual keyboard is intended to be shown Parameters: ► `visibility`: If true, a virtual keyboard is made visible. If false, it is invisible.
3	`onKeyboardVisibilityChanged`	Is sent by the application if a virtual keyboard is shown Parameters: ► `visibility`: If true, a virtual keyboard is visible. If false, it is invisible.
4	`onLayoutChanged`	Is sent by the application when the visible area of the screen changes Parameters (in pixels): ► `x`: The x-coordinate of the top left corner of the visible screen area

Event ID	Name	Description
		► `y`: The y-coordinate of the top left corner of the visible screen area ► `width`: The width of the visible screen area ► `height`: The height of the visible screen area

16.2. Datapool items

Table 16.2. Properties of a datapool item

Property name	Description
`Value`	The initial value of the datapool item

16.3. Data types

The following section describes data types in EB GUIDE. You can add user-defined properties and datapool items from the types listed below.

16.3.1. Mesh

Mesh defines the shape of the 3D object.

Available operations are as follows:

- assign (writable properties) (=)

It is possible to store meshes in a list. For details about lists see section 16.3.9, “List”.

16.3.2. Boolean

Boolean properties can have the values true and false.

Available operations are as follows:

- equal (==)
- not equal (!=)
- negation (!)
- and (&&)
- or (||)
- assign (writable properties) (=)

It is possible to store boolean properties in a list. For details about lists, see section 16.3.9, “List”.

16.3.3. Color

Colors are stored in the RGBA8888 format.

Example: Red without transparency is (255, 0, 0, 255).

Available operations are as follows:

- equal (==)
- not equal (!=)
- assign (writable properties) (=)

It is possible to store color properties in a list. For details about lists, see section 16.3.9, “List”.

16.3.4. Conditional script

Conditional scripts are used to react on initialization and on trigger. When you edit conditional scripts, the content area is divided into the following sections.

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

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

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

  If true, it triggers a change notification.

If false, it does not trigger a change notification.

16.3.5. Float

Float-point number data type represents a single-precision 32-bit IEEE 754 value.

Available operations are as follows:

- equal (==)
- not equal (!=)
- greater (>)
- greater or equal (>=)
- less (<)
- less or equal (<=)
- addition (+)
- subtraction (-)
- multiplication (*)
- division (/)
- assign (writable properties) (=)

It is possible to store float properties in a list. For details about lists, see section 16.3.9, “List”.

16.3.6. Font

To add a font to an EB GUIDE project, copy the font file in the following directory: `$GUIDE_PROJECT_PATH/<project name>/resources`

Available operations are as follows:

- assign (writable properties) (=)

It is possible to store font properties in a list. For details about lists, see section 16.3.9, “List”.

16.3.7. Image

To add an image to an EB GUIDE project, copy the image file in the following directory: `$GUIDE_PROJECT_PATH/<project name>/resources`

Available operations are as follows:

- assign (writable properties) (=)

It is possible to store image properties in a list. For details about lists, see section 16.3.9, “List”.

16.3.8. Integer

EB GUIDE supports signed 32-bit integers.

Available operations are as follows:

- equal (==)
- not equal (!=)
- greater (>)
- greater or equal (>=)
- less (<)
- less or equal (<=)
- addition (+)
- subtraction (-)
- multiplication (*)
- division (/)
- modulo (%)
- assign (writable properties) (=)

It is possible to store integer properties in a list. For details about lists, see section 16.3.9, “List”.

16.3.9. List

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

The following list types are available:

- Mesh list
- Boolean list
- Color list
- Float list
- Font list
- Image list
- Integer list
- String list

The following types cannot be used in lists:

- List
- Property reference
- List element reference

Available operations are as follows:

- length: (length)
- element accessor: ([])

16.3.10. String

EB GUIDE supports character strings, for example *Hello world*.

Available operations are as follows:

- equal (case sensitive) (==)
- not equal (case sensitive) (!=)
- equal (case insensitive, only in the ASCII range) (=Aa=)
- greater (>)
- greater or equal (>=)
- less (<)
- less or equal (<=)
- concatenation (+)
- assign (writable properties) (=)

It is possible to store string properties in a list. For details about lists, see section 16.3.9, “List”.

16.4. EB GUIDE Script

16.4.1. EB GUIDE Script keywords

The following is a list of reserved keywords in EB GUIDE Script. If you want to use these words as identifiers in a script, you must quote them.

Keyword	Description
`color:`	A color parameter follows, for example {0,255,255}.
`dp:`	A datapool item follows.
`l:`	A language follows.
`else`	An `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
`if`	A statement which tests a boolean expression follows. If the expression is true, the statement is executed.
`in`	Is a separator between a local variable declaration and the variable's scope of usage Is used with `match_event` and `let`.
`function`	Declares a function
`length`	The length of a property
`let`	Declares a local variable that is accessible in the scope
`list`	Declares a list type, for example an integer list
`match_event`	Checks if the current event corresponds to an expected event and declares variables like `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.

Keyword	Description
`while`	Repeats a statement as long as the condition is true

16.4.2. EB GUIDE Script operator precedence

The following is a list of the operators in EB GUIDE Script together with their precedence and associativity. Operators are listed top to bottom, in descending precedence.

Table 16.3. EB GUIDE Script operator precedence

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

16.4.3. EB GUIDE Script standard library

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

16.4.3.1. EB GUIDE Script functions A

16.4.3.1.1. `abs`

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

Table 16.4. Parameters of `abs`

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

16.4.3.1.2. `absf`

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

Table 16.5. Parameters of `absf`

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

16.4.3.1.3. `acosf`

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

Table 16.6. Parameters of `acosf`

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

16.4.3.1.4. `animation_before`

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

Table 16.7. Parameters of `animation_before`

Parameter	Type	Description
`animation`	GtfTypeRecord	The animation to manipulate
`time`	integer	The point in time
`<return>`	boolean	If true, the animation has not yet passed the point in time.

16.4.3.1.5. `animation_beyond`

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

Table 16.8. Parameters of `animation_beyond`

Parameter	Type	Description
`animation`	GtfTypeRecord	The animation to manipulate
`time`	integer	The point in time
`<return>`	boolean	If true, the animation has passed the point in time.

16.4.3.1.6. `animation_cancel`

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

Table 16.9. Parameters of `animation_cancel`

Parameter	Type	Description
`animation`	GtfTypeRecord	The animation to manipulate
`<return>`	boolean	If true, the function succeeded.

16.4.3.1.7. `animation_cancel_end`

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

Table 16.10. Parameters of `animation_cancel_end`

Parameter	Type	Description
`animation`	GtfTypeRecord	The animation to manipulate
`<return>`	boolean	If true, the function succeeded.

16.4.3.1.8. `animation_cancel_reset`

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

Table 16.11. Parameters of `animation_cancel_reset`

Parameter	Type	Description
`animation`	GtfTypeRecord	The animation to manipulate
`<return>`	boolean	If true, the function succeeded.

16.4.3.1.9. `animation_pause`

The function pauses an animation.

Table 16.12. Parameters of `animation_pause`

Parameter	Type	Description
`animation`	GtfTypeRecord	The animation to manipulate
`<return>`	boolean	If true, the function succeeded.

16.4.3.1.10. `animation_play`

The function starts or continues an animation.

Table 16.13. Parameters of `animation_play`

Parameter	Type	Description
`animation`	GtfTypeRecord	The animation to manipulate
`<return>`	boolean	If true, the animation is not running yet.

16.4.3.1.11. `animation_reverse`

The function plays an animation backwards.

Table 16.14. Parameters of `animation_reverse`

Parameter	Type	Description
`animation`	GtfTypeRecord	The animation to manipulate
`<return>`	boolean	If true, the animation is not running yet.

16.4.3.1.12. `animation_running`

The function checks if an animation is currently running.

Table 16.15. Parameters of `animation_running`

Parameter	Type	Description
`animation`	GtfTypeRecord	The animation to manipulate
`<return>`	boolean	If true, the animation is running.

16.4.3.1.13. `animation_set_time`

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

Table 16.16. Parameters of `animation_set_time`

Parameter	Type	Description
`animation`	GtfTypeRecord	The animation to manipulate
`time`	integer	time
`<return>`	boolean	If true, the function succeeded.

16.4.3.1.14. `asinf`

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

Table 16.17. Parameters of `asinf`

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

16.4.3.1.15. `atan2f`

The function calculates the principal value of the arc tangent of y/x, using the signs of the two arguments to determine the quadrant of the result.

Table 16.18. Parameters of `atan2f`

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

16.4.3.1.16. `atan2i`

The function calculates the principal value of the arc tangent of y/x, using the signs of the two arguments to determine the quadrant of the result.

Table 16.19. Parameters of `atan2i`

Parameter	Type	Description
`y`	integer	Argument y

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

16.4.3.1.17. atanf

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

Table 16.20. Parameters of atanf

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

16.4.3.2. EB GUIDE Script functions C - H

16.4.3.2.1. ceil

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

Table 16.21. Parameters of ceil

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

16.4.3.2.2. changeDynamicStateMachinePriority

The function changes the priority of a dynamic state machine.

Table 16.22. Parameters of changeDynamicStateMachinePriority

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

16.4.3.2.3. `character2unicode`

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

Table 16.23. Parameters of `character2unicode`

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

16.4.3.2.4. `clearAllDynamicStateMachines`

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

Table 16.24. Parameters of `clearAllDynamicStateMachines`

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

16.4.3.2.5. `color2string`

The function converts a color to eight hexadecimal values.

Table 16.25. Parameters of `color2string`

Parameter	Type	Description
`value`	color	The color to convert to string
`<return>`	string	The color formatted as a string of hexadecimal digits with # as prefix

NOTE

Formatting examples

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

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

16.4.3.2.6. `cosf`

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

Table 16.26. Parameters of `cosf`

Parameter	Type	Description
`x`	float	The number to return the cosine from
`<return>`	float	The return value

16.4.3.2.7. `deg2rad`

The function converts an angle from degrees to radians.

Table 16.27. Parameters of `deg2rad`

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

16.4.3.2.8. `expf`

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

Table 16.28. Parameters of `expf`

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

16.4.3.2.9. `float2string`

The function converts simple float to string.

Table 16.29. Parameters of `float2string`

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

16.4.3.2.10. `floor`

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

Table 16.30. Parameters of `floor`

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

16.4.3.2.11. `focusNext`

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

Table 16.31. Parameters of `focusNext`

Parameter	Type	Description
`<return>`	void	

16.4.3.2.12. `focusPrevious`

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

Table 16.32. Parameters of `focusPrevious`

Parameter	Type	Description
`<return>`	void	

16.4.3.2.13. `format_float`

The function formats a float value.

Table 16.33. Parameters of `format_float`

Parameter	Type	Description
`format`	string	A string of the following structure: %[flags] [width] [.precision] type ▶ flags: Optional character or characters that control output justification and output of signs, blanks, leading zeros, decimal points, and octal and hexadecimal prefixes. ▶ width: Optional decimal number that specifies the minimum number of characters that are output. ▶ precision: Optional decimal number that specifies the number of significant digits or the number of digits after the decimal-point character .

Parameter	Type	Description
		▶ type: Required conversion specifier character that determines whether the associated argument is interpreted as a character, a string, an integer, or a float number.
`useDotAsDelimiter`	boolean	Defines the delimiter sign. Possible values: ▶ true: Use a dot as delimiter. ▶ false: Use a comma as delimiter.
`value`	float	The number to format

WARNING **Adhere to printf specification for C++**

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

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

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

16.4.3.2.14. `format_int`

The function formats an integer value.

Table 16.34. Parameters of `format_int`

Parameter	Type	Description
`format`	string	A string of the following structure: %[flags] [width] [.precision] type ▶ flags: Optional character or characters that control output justification and output of signs, blanks, leading zeros, decimal points, and octal and hexadecimal prefixes. ▶ width: Optional decimal number that specifies the minimum number of characters that are output. ▶ precision: Optional decimal number that specifies the minimum number of digits that are printed. ▶ type: Required conversion specifier character that determines whether the associated argument is interpreted as a character, a string, an integer, or a float number.
`value`	int	The number to format

WARNING **Adhere to printf specification for C++**

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

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

For example, allowed types for `format_int` are `d`, `i`, `o`, `x` and `u`, and not more than one type character is allowed.

16.4.3.2.15. `getLineCount`

The function returns the number of lines of a text for a widget.

Table 16.35. Parameters of `getLineCount`

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

16.4.3.2.16. `getTextHeight`

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

Table 16.36. Parameters of `getTextHeight`

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

16.4.3.2.17. `getTextLength`

The function returns the number of characters in a text.

Table 16.37. Parameters of `getTextLength`

Parameter	Type	Description
`text`	string	The text to evaluate
`<return>`	integer	The number of characters in the text

NOTE **Escape sequences**

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

16.4.3.2.18. `getTextWidth`

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

Table 16.38. Parameters of `getTextWidth`

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

16.4.3.2.19. `has_list_window`

The function checks if the index is valid for a datapool item of type list. For windowed lists it also checks if the index is located inside at least one window.

Table 16.39. Parameters of `has_list_window`

Parameter	Type	Description
`itemId`	dp_id	The ID of the datapool item of type list
`index`	integer	The index within the datapool item
`<return>`	boolean	If true, the index within a datapool item is valid and located inside at least one window.

16.4.3.2.20. `hsba2color`

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

Table 16.40. Parameters of `hsba2color`

Parameter	Type	Description
`hue`	integer	The color value in degrees from 0 to 360
`saturation`	integer	The saturation in percent

Parameter	Type	Description
`brightness`	integer	The brightness in percent
`alpha`	integer	The alpha value between 0 (totally transparent) and 255 (opaque)
`<return>`	color	The resulting GTF color with the alpha value applied

16.4.3.3. EB GUIDE Script functions I - R

16.4.3.3.1. `int2float`

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

Table 16.41. Parameters of `int2float`

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

16.4.3.3.2. `int2string`

The function converts a simple integer to string.

Table 16.42. Parameters of `int2string`

Parameter	Type	Description
`value`	integer	The value to convert to string
`<return>`	string	The integer value, in decimal notation, converted to string

16.4.3.3.3. `isDynamicStateMachineActive`

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

Table 16.43. Parameters of `isDynamicStateMachineActive`

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

16.4.3.3.4. `language`

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

Table 16.44. Parameters of `language`

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

16.4.3.3.5. `localtime_day`

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

Table 16.45. Parameters of `localtime_day`

Parameter	Type	Description
`time`	integer	A time stamp as returned by system time
`<return>`	integer	The extracted day

16.4.3.3.6. `localtime_hour`

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

Table 16.46. Parameters of `localtime_hour`

Parameter	Type	Description
`time`	integer	A time stamp as returned by system time
`<return>`	integer	The extracted hour

16.4.3.3.7. `localtime_minute`

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

Table 16.47. Parameters of `localtime_minute`

Parameter	Type	Description
`time`	integer	A time stamp as returned by system time
`<return>`	integer	The extracted minute

16.4.3.3.8. `localtime_month`

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

Table 16.48. Parameters of `localtime_month`

Parameter	Type	Description
`time`	integer	A time stamp as returned by system time
`<return>`	integer	The extracted month

16.4.3.3.9. `localtime_second`

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

Table 16.49. Parameters of `localtime_second`

Parameter	Type	Description
`time`	integer	A time stamp as returned by system time
`<return>`	integer	The extracted second

16.4.3.3.10. `localtime_weekday`

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

Table 16.50. Parameters of `localtime_weekday`

Parameter	Type	Description
`time`	integer	A time stamp as returned by system time
`<return>`	integer	The extracted weekday

16.4.3.3.11. `localtime_year`

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

Table 16.51. Parameters of `localtime_year`

Parameter	Type	Description
`time`	integer	A time stamp as returned by system time
`<return>`	integer	The extracted year

16.4.3.3.12. `log10f`

The function returns the base 10 logarithm of x.

Table 16.52. Parameters of `log10f`

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

16.4.3.3.13. `logf`

The function returns the natural logarithm of x.

Table 16.53. Parameters of `logf`

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

16.4.3.3.14. `nearbyint`

The function rounds to nearest integer.

Table 16.54. Parameters of `nearbyint`

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

16.4.3.3.15. `popDynamicStateMachine`

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

Table 16.55. Parameters of `popDynamicStateMachine`

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

16.4.3.3.16. `powf`

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

Table 16.56. Parameters of `powf`

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

16.4.3.3.17. `pushDynamicStateMachine`

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

Table 16.57. 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

16.4.3.3.18. `rad2deg`

The function converts an angle form radians to degree.

Table 16.58. Parameters of `rad2deg`

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

16.4.3.3.19. `rand`

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

Table 16.59. Parameters of `rand`

Parameter	Type	Description
`<return>`	integer	A random number between 0 and 2^{31}-1

16.4.3.3.20. `shutdown`

The function requests the framework to shutdown the program.

16.4.3.3.21. `rgba2color`

The function converts from RGB color space to GTF color.

Table 16.60. 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
`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

16.4.3.3.22. `round`

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

Table 16.61. Parameters of `round`

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

16.4.3.4. EB GUIDE Script functions S - W

16.4.3.4.1. `seed_rand`

The function sets the seed of the random number generator.

Table 16.62. Parameters of `seed_rand`

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

16.4.3.4.2. `sinf`

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

Table 16.63. Parameters of `sinf`

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

16.4.3.4.3. `skin`

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

Table 16.64. Parameters of `skin`

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

16.4.3.4.4. `sqrtf`

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

Table 16.65. Parameters of `sqrtf`

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

16.4.3.4.5. `string2float`

The function converts the initial part of a string to float.

The expected form of the initial part of the string is as follows:

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

Table 16.66. Parameters of `string2float`

Parameter	Type	Description
`str`	string	The string value
`<return>`	float	The return value

16.4.3.4.6. `string2int`

The function converts the initial part of a string to integer. The result is clipped to the range from 2147483647 to -2147483648, if the input exceeds the range. If the string does not start with a number, the function returns 0.

Table 16.67. Parameters of `string2int`

Parameter	Type	Description
`str`	string	The string value
`<return>`	integer	The return value

16.4.3.4.7. `string2string`

The function formats strings.

Table 16.68. 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

16.4.3.4.8. `substring`

The function creates a substring copy of the string. Negative end indexes are supported.

Examples:

- substring("abc", 0, -1) returns "abc".
- substring("abc", 0, -2) returns "ab".

Table 16.69. Parameters of `substring`

Parameter	Type	Description
`str`	string	The input string
`startIndex`	integer	The first character index of the result string

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

16.4.3.4.9. system_time

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

Table 16.70. Parameters of system_time

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

16.4.3.4.10. system_time_ms

The function gets the current system time in milliseconds.

Table 16.71. Parameters of system_time_ms

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

16.4.3.4.11. tanf

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

Table 16.72. Parameters of tanf

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

16.4.3.4.12. trace_dp

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

Table 16.73. Parameters of trace_dp

Parameter	Type	Description
itemId	dp_id	The datapool ID of the item to trace debug information about
<return>	void	

16.4.3.4.13. `trace_string`

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

Table 16.74. Parameters of `trace_string`

Parameter	Type	Description
`str`	string	The text to trace
`<return>`	void	

16.4.3.4.14. `transformToScreenX`

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

Table 16.75. 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

16.4.3.4.15. `transformToScreenY`

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

Table 16.76. 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

16.4.3.4.16. `transformToWidgetX`

The function takes a widget and a screen coordinate as provided to the touch reactions and returns x-position in the widget-relative local coordinate system.

Table 16.77. Parameters of `transformToWidgetX`

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

16.4.3.4.17. `transformToWidgetY`

The function takes a widget and a screen coordinate as provided to the touch reactions and returns y-position in the widget-relative local coordinate system.

Table 16.78. Parameters of `transformToWidgetY`

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

16.4.3.4.18. `trunc`

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

Table 16.79. Parameters of `trunc`

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

16.4.3.4.19. `widgetGetChildCount`

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

Table 16.80. Parameters of `widgetGetChildCount`

Parameter	Type	Description
`widget`	widget	The widget of which to obtain the number of child widgets

Parameter	Type	Description
`<return>`	integer	The number of child widgets

16.5. Events

Table 16.81. Properties of an event

Property name	Description
`Name`	The name of the event
`Event ID`	A numeric value that EB GUIDE TF uses to send and receive the event
`Event group`	The name of the event group An event group has an ID that EB GUIDE TF uses to send and receive the event.

16.6. `model.json` configuration file

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

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

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

NOTE

JSON object notation

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

For an example, see section 16.6.1, “Example `model.json` in EB GUIDE Studio”.

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

Table 16.82. Common

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

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

Table 16.83. Files and paths

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

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

Table 16.84. Cores

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

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

Table 16.85. Scenes

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

Configuration item	Type	Description	Default value
		▶ 0: no multisampling ▶ 1: 2x multisampling ▶ 2: 4x multisampling	
`gtf.model.scenes.< scenename>.enableRemoteFramebuffer`	boolean	If `true`, the transfer of the off-screen buffer to the simulation window is enabled	`false`
`gtf.model.scenes.< scenename>.showWindowFrame`	boolean	Determines if the renderer window frame should be displayed	`true`
`gtf.model.scenes.< scenename>.showWindow`	boolean	If `true`, an additional window for simulation is opened on Windows based systems	`true`
`gtf.model.scenes.< scenename>.disableVsync`	boolean	If `true`, the vertical synchronization for the renderer is disabled.	`false`
`gtf.model.scenes.<scenename>.showFPS`	integer	Possible values: ▶ `0`: Do not show FPS ▶ `1`: Show FPS on the screen ▶ `2`: Show FPS on the console ▶ `3`: Show FPS on the screen and on the console	`0`
`gtf.model.scenes.<scenename>.renderer`	string	Name of the renderer to use: `DirectXRenderer` `OpenGLRenderer` or `OpenGL3Renderer`	

Table 16.86. Rendering common

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

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

Table 16.87. Renderer display extensions

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

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

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

Table 16.88. `TextEngine` configuration items

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

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

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

Table 16.89. Touch devices

Configuration item	Type	Description	Default value
gtf.model.touchDevices.< deviceName>.touchscreenType	integer	Defines the touch device type: ▶ 0: Galaxy ▶ 1: imx WVGA ▶ 2: Mouse ▶ 3: General ▶ 4: Lilliput_889GL ▶ 5: GeneralMultitouch ▶ 6: Lilliput with automatic calibration ▶ 7: GenericTouchConfiguration	3
gtf.model.touchDevices.< deviceName>.displayManagerId	integer	Specifies the scene ID for which the device is valid	0

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

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

Example 16.1.
`model.json` in EB GUIDE Studio

```
{
```

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

    ],
    "driverName":[

    ],
    "fontCache":{
      "width":512,
      "height":512,
      "age":100
```

```
        },
        "maxCacheSize":[

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

        ]
      }
    }
}
```

16.7. `platform.json` configuration file

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

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

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

NOTE

JSON object notation

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

For an example, see section 16.7.1, “Example `platform.json` in EB GUIDE Studio”.

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

Table 16.90. Platform configuration

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

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

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

Example 16.2.
`platform.json` in EB GUIDE Studio

```
{
  "gtf":{
    "core":{
      "pluginstoload":[
        "TfRuntime",
        "TfService",
        "TfGui",
        "TfGUIOpenGLES20",
        "TfGUIOpenGLES3",
        "TfGUIDirectX11"
      ]
    },
    "servicemapper":{
      "port":60000
    },
    "launcher":{
      "editmode":true
    },
    "platform":{
      "config":"<binary_folder>/platform.json"
    },
    "framework":{
      "path":"<gtf_binaries_folder>"
    },
    "diagnostic":{
      "memory":{
        "interval":0
      }
    },
    "ipc":{
      "role":"server",
      "discovery":{
        "network":"255.255.255.255",
        "port":4711,
        "mode":"broadcast"
      },
      "client":{
        "timeout":5000
      }
```

```
      }
    }
  }
```

16.8. Scenes

Table 16.91. Properties of a scene

Property name	Description
`height`	The height of the area in which the views of a haptic state machine are rendered on a target device
`width`	The width of the area in which the views of a haptic state machine are rendered on a target device
`x`	The x-offset of the area in which the views of a haptic state machine are rendered on a target device
`y`	The y-offset of the area in which the views of a haptic state machine are rendered on a target device
`visible`	If true, the state machine and its child widgets are visible.
`projectName`	The name of the EB GUIDE project
`windowCaption`	The text that is shown on the window frame
`sceneID`	The unique scene identifier which can be used, for example, for input handling
`maxFPS`	The redraw rate (FPS = Frames per second) Set to `0` for an unlimited redraw rate.
`hwLayerID`	The ID of the hardware layer on the target device's display that is mapped to the current state machine
`colorMode`	Possible values: ► 32 bit (=1): RGBA8888 ► 16 bit (=2): RGB565 ► 24 bit (=3): RGB888
`multisampling`	Possible values: ► Off (= 0): no multisampling

Property name	Description
	▶ 2x (=1): 2x multisamling ▶ 4x (=2): 4x multisampling
`enableRemoteFramebuffer`	If true, transfer of the off-screen buffer to the simulation window is enabled
`showWindowFrame`	If true, a frame is displayed on the simulation window. The frame allows the window to be grabbed and moved.
`showWindow`	If true, an additional window for simulation is opened on Windows based systems.
`disableVSync`	If true, vertical synchronization for the renderer is disabled.
`showFPS`	Possible values: ▶ 0: Do not show FPS ▶ 1: Show FPS on the screen ▶ 2: Show FPS on the console ▶ 3: Show FPS on the screen and on the console
`Renderer`	Defines a renderer for the scene. Possible values: ▶ DirectX ▶ OpenGL

TIP

Settings for multisampling

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

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

16.9. Touch screen types supported by EB GUIDE GTF

The supported types depend on the target device.

Table 16.92. Touch screen types supported by EB GUIDE GTF

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

16.10. Widgets

16.10.1. View

Table 16.93. Properties of a view

Property name	Description
`name`	The name of the widget
`height`	The height of the widget in pixels
`width`	The width of the widget in pixels
`visible`	If true, the widget and its child widgets are visible
`x`	The x-coordinate of the widget
`y`	The y-coordinate of the widget

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

Table 16.94. Properties of an entry animation

Property name	Description
`Entry animation`	If true, instances of the view template have an entry animation.
`Type`	The type of the entry animation, for example **Move in from left**, **Fade in from center** or **Show view immediately**.
`Duration`	The duration of the entry animation in milliseconds
`Delay`	The delay of the entry animation in milliseconds

Property name	Description
`Play after exit animation`	If true, the start time of the entry animation depends on the duration of a previous exit animation.

An exit animation is executed when the view is exited.

Table 16.95. Properties of an exit animation

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

16.10.2. Basic widgets

There are six basic widgets.

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

The following sections list the properties of basic widgets.

NOTE **Unique names**

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

NOTE **Negative values**

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

16.10.2.1. Container

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

Table 16.96. Properties of the container

Property name	Description
`name`	The name of the widget
`height`	The height of the widget in pixels
`width`	The width of the widget in pixels
`visible`	If true, the widget and its child widgets are visible
`x`	The x-coordinate of the widget relative to its parent widget
`y`	The y-coordinate of the widget relative to its parent widget

16.10.2.2. Ellipse

An ellipse draws a colored ellipse with the dimensions and coordinates of the widget into a view. The widget can also be used to draw a sector or an arc.

Table 16.97. Properties of the ellipse

Property name	Description
`name`	The name of the widget
`height`	The height of the widget in pixels
`width`	The width of the widget in pixels
`visible`	If true, the widget and its child widgets are visible
`x`	The x-coordinate of the widget relative to its parent widget
`y`	The y-coordinate of the widget relative to its parent widget
`fillColor`	The color that fills the ellipse
`arcWidth`	The width of the arc of the ellipse
`centralAngle`	The angle in degrees which defines a sector of the ellipse
`sectorRotation`	The angle in degrees which describes the rotation of the ellipse's sector

16.10.2.3. Image

An image places a picture into a view.

Table 16.98. Properties of the image

Property name	Description
`name`	The name of the widget

Property name	Description
`height`	The height of the widget in pixels
`width`	The width of the widget in pixels
`visible`	If true, the widget and its child widgets are visible
`x`	The x-coordinate of the widget relative to its parent widget
`y`	The y-coordinate of the widget relative to its parent widget
`image`	The image the widget displays
`horizontalAlign`	The horizontal alignment of the image file within the boundaries of the widget
`verticalAlign`	The vertical alignment of the image file within the boundaries of the widget

NOTE

Supported image file types

The available image formats depend on the implementation of the renderer. DirectX 11 and OpenGL ES version 2.0 or higher support `.png` files and `.jpg` files.

16.10.2.4. Instantiator

An instantiator creates widget instances during run-time. You can use the instantiator to model lists or tables with dynamic or static content. The child widgets of an instantiator serve as line templates for the list or table which is created during run-time. By default the instantiator only instantiates the first line template.

Table 16.99. Properties of the instantiator

Property name	Description
`name`	The name of the widget
`height`	The height of the widget in pixels
`width`	The width of the widget in pixels
`visible`	If true the widget and its child widgets are visible
`x`	The x-coordinate of the widget relative to its parent widget
`y`	The y-coordinate of the widget relative to its parent widget
`numItems`	The number of instantiated child widgets. If `numItems` is `0`, no child widgets are created.
`lineMapping`	Defines which child widget is the line template for which line, i.e. defines the order of instantiation

16.10.2.5. Label

A label places text into a view.

Table 16.100. Properties of the label

Property name	Description
`name`	The name of the widget
`height`	The height of the widget in pixels
`width`	The width of the widget in pixels
`visible`	If true, the widget and its child widgets are visible
`x`	The x-coordinate of the widget relative to its parent widget
`y`	The y-coordinate of the widget relative to its parent widget
`text`	The text the label displays. If the text does not fit into the widget area it is truncated at the end by default.
`textColor`	The color in which the text is displayed
`font`	The font in which the text is displayed
`horizontalAlign`	The horizontal alignment of the text within the boundaries of the label
`verticalAlign`	The vertical alignment of the text within the boundaries of the label

16.10.2.6. Rectangle

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

Table 16.101. Properties of the rectangle

Property name	Description
`name`	The name of the widget
`height`	The height of the widget in pixels
`width`	The width of the widget in pixels
`visible`	If true, the widget and its child widgets are visible
`x`	The x-coordinate of the widget relative to its parent widget
`y`	The y-coordinate of the widget relative to its parent widget
`fillColor`	The color that fills the rectangle

16.10.3. Animations

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

16.10.3.1. Animation

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

Table 16.102. Properties of the animation

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

16.10.3.2. Constant curves

A constant curve is a child widget of an animation. A constant curve sets a target value after a defined delay. Constant curves are available for integer, boolean, float, and color types.

Table 16.103. Properties of constant curves

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

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

16.10.3.3. Fast start curves

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

Table 16.104. Properties of fast start curves

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

16.10.3.4. Slow start curves

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

Table 16.105. Properties of slow start curves

Property name	Description
name	The name of the curve

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

16.10.3.5. Quadratic curves

A quadratic curve is a child widget of an animation. A quadratic curve periodically sets a value using a quadratic function curve. Quadratic curves are available for integer, float, and color types.

Table 16.106. Properties of quadratic curves

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

16.10.3.6. Sinus curves

A sinus curve is a child widget of an animation. A sinus curve periodically sets a value using a sinus function curve. Sinus curves are available for integer, float, and color types.

Table 16.107. Properties of sinus curves

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

16.10.3.7. Script curves

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

Table 16.108. Properties of script curves

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

16.10.3.8. Linear curves

A linear curve is a child widget of an animation. A linear curve periodically sets a value using a linear progression curve. Linear curves are available for integer, float, and color types.

Table 16.109. Properties of linear curves

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

16.10.3.9. Linear interpolation curves

A linear interpolation curve is a child widget of an animation. A linear interpolation curve widget periodically sets a value using a linear interpolation curve. Linear interpolation curves are available for integer, float, and color types.

NOTE

Linear key value interpolation curves

During import of a 3D graphic file, if the imported 3D scene has animations, linear key value interpolation integer curve and linear key value interpolation float curve are created. The underlying key-value pairs of these curves cannot be modified in EB GUIDE Studio.

Table 16.110. Properties of linear interpolation curves

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

Property name	Description
`start`	The initial value
`end`	The final value

16.10.4. 3D widgets

16.10.4.1. Scene graph

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

Table 16.111. Properties of the scene graph

Property name	Description
`visible`	If true, the widget and its child widgets are visible
`width`	The width of the widget in pixels
`height`	The height of the widget in pixels
`x`	The x-coordinate of the widget relative to its parent widget
`y`	The y-coordinate of the widget relative to its parent widget

16.10.4.2. Scene graph node

A scene graph node is a child node and is added to the scene graph or to another scene graph node. You use scene graph nodes to place 3D widgets in the 3D scene with transformation properties. The following 3D widgets can be added to the scene graph node:

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

Table 16.112. Properties of the scene graph node

Property name	Description
`visible`	If true, the widget and its child widgets are visible
`rotationX`	The rotation around the x-axis
`rotationY`	The rotation around the y-axis

Property name	Description
rotationZ	The rotation around the z-axis
scalingX	The scaling along the x-axis
scalingY	The scaling along the y-axis
scalingZ	The scaling along the z-axis
translationX	The translation along the x-axis
translationY	The translation along the y-axis
translationZ	The translation along the z-axis

16.10.4.3. Camera

A camera defines the view of the scene from a particular point of view. Use several cameras to show the scene from different points of view.

Table 16.113. Properties of the camera

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

16.10.4.4. Directional light

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

Table 16.114. Properties of the directional light

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

16.10.4.5. Material

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

Table 16.115. Properties of the material

Property name	Description
`ambient`	The color the object reflects when it is illuminated by ambient light
`diffuse`	The color the object reflects evenly in all directions when it is illuminated by pure white light
`emissive`	The self-illumination color of the object
`shininess`	The shininess factor
`specular`	The color an object with a shiny surface reflects
`opacity`	The opacity value Note that only values between `0` and `1`, as for example `0.3`, are valid.

16.10.4.6. Mesh

A mesh defines the shape of the 3D object.

Table 16.116. Properties of the mesh

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

16.10.4.7. Point light

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

Table 16.117. Properties of the point light

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

16.10.4.8. Spot light

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

Table 16.118. Properties of the spot light

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

16.11. Widget features

The following list contains a description of all widget features that are implemented, with a brief description on how to use them in an EB GUIDE model.

16.11.1. Common

16.11.1.1. Child visibility selection

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

Table 16.119. Properties of the **Child visibility selection** widget feature

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

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

16.11.1.2. Enabled

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

Table 16.120. Properties of the **Enabled** widget feature

Property name	Description
`enabled`	If true, the widget reacts on touch and press input

16.11.1.3. Focused

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

Table 16.121. Properties of the **Focused** widget feature

Property name	Description
`focusable`	Defines whether the widget receives the focus or not. Possible values: ► `not focusable (=0)` ► `only by touch (=1)` ► `only by key (=2)` ► `focusable (=3)`
`focused`	If true, the widget has focus

16.11.1.4. Multiple lines

The **Multiple lines** widget feature enables line breaks.

Restrictions:

► The **Multiple lines** widget feature is only available for the label widget.

Table 16.122. Properties of the **Multiple lines** 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.

Property name	Description
	When the `line gap` is too small (high negative value), it has no effect anymore and the text is rendered in one line. This occurs for example, when the font style is set to `PT_Sans_Narrow`, size is set to `30` and the `line gap` is defined as `-50`.
`maxLineCount`	The maximum number of visible lines. 0 = no limitation

TIP

Number of lines used

With the script function `getLineCount`, you can obtain the number of lines of the text.

For more information on this, see section 16.4.3.2.15, “`getLineCount`”.

NOTE

Character replacement

Sequences of '\\' '\\' are replaced by '\\' . Sequences of '\\' 'n' are replaced by '\n'.

If the size of the label is increased so that one line is sufficient to display the text, '\n' is replaced by ' '.

16.11.1.5. Pressed

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

Restrictions:

- Adding the **Pressed** widget feature automatically adds the **Focused** widget feature.

Table 16.123. Properties of the **Pressed** widget feature

Property name	Description
`pressed`	If true, a key is pressed while the widget is focused

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

16.11.1.6. Selected

The **Selected** widget feature adds a `selected` property to a widget. It is typically set by the application or the HMI modeler. It is not changed by any other component of the framework.

Table 16.124. Properties of the **Selected** widget feature

Property name	Description
`selected`	If true, the widget is selected

16.11.1.7. Selection group

The **Selection group** widget feature is used to model a list of radio buttons. In the list, every radio button has the **Selection group** widget feature and a unique button ID.

Use a datapool item for the `buttonValue` property. Assign the datapool item to all widgets in the radio button array.

Selecting and deselecting a widget within the button group can be done by an application that sets the `buttonValue` property. Alternatively, changes can be triggered by touch or key input as well as by adding a condition that sets the button value.

Restrictions:

- Adding the **Selection group** widget feature automatically adds the **Selected** widget feature.

Table 16.125. Properties of the **Selection group** widget feature

Property name	Description
`buttonId`	The ID that identifies a button within a button group
`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.

16.11.1.8. Spinning

The **Spinning** widget feature turns a widget into a rotary button. A widget with the **Spinning** widget feature reacts to increment and decrement events by changing an internal value. The **Spinning** widget feature can be used to create a scale, a progress bar, or a widget with a preview value.

Table 16.126. Properties of the **Spinning** 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`	The 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

Property name	Description
steps	The number of steps to calculate the increment or decrement for the currentValue property
valueWrapAround	Possible values: ► true: The currentValue property continues at the inverse border, if minValue or maxValue is exceeded. ► false: The currentValue property does not decrease/increase, if minValue or maxValue is exceeded.

16.11.1.9. Text truncation

The **Text truncation** widget feature truncates the content of the text property if it does not fit into the widget area. The widget feature enables a different truncation than the default setting trailing.

Restrictions:

► The **Text truncation** widget feature is only available for the label widget.

Table 16.127. Properties of the **Text truncation** widget feature

Property name	Description
truncationPolicy	For single-line texts, the truncationPolicy property defines the position of the truncation. Possible values: ► leading (=0): Text is replaced at the beginning of the text ► trailing (=1): Text is replaced at the end of the text For multi-line texts, the truncationPolicy property defines where text is replaced. Possible values: ► leading (=0): Lines at the beginning are replaced and text of the first visible line is truncated at the beginning of the text. ► trailing (=1) 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

16.11.1.10. Touched

The **Touched** widget feature enables a widget to react to touch input.

Table 16.128. Properties of the **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 (=0)`: Press first, then the widget reacts. Notifications of moving and releasing are only active within the widget area. ▶ `Press and grab (=1)`: Press to grab the contact. The contact remains grabbed even if it moves away from the widget area. ▶ `Press then react on contact (=3)`: 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: ▶ `Whole area (=0)`: To identify the touched widget, the renderer evaluates the widget's clipping rectangle. ▶ `Visible pixels (=1)`: 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. Note that the `Visible pixels` value has no effect on labels.

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

TIP

Performance recommendation

If performance is an important issue in your project, set the `touchBehavior` property to `Whole area (=0)`. EB GUIDE GTF evaluates `Whole area (=0)` faster than `Visible pixels (=1)`.

16.11.2. Effect

16.11.2.1. Border

The **Border** widget feature adds a configurable border to the widget. The border starts at the widget boundaries and is placed within the widget.

Restrictions:

- The widget feature is available for rectangles.

Table 16.129. 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

16.11.2.2. Coloration

The **Coloration** widget feature colors the widget and its widget subtree. It also affects transparency if the alpha value is not opaque.

Example 16.3.
Usage of the Coloration widget feature

For all colors with RGBA components between 0.0 and 1.0, the algorithm in the **Coloration** widget feature multiplies the current color values of a widget by the `colorationColor` property value. Multiplication is done per pixel and component-wise.

A semi-transparent gray colored by an opaque blue results in semi-transparent darker blue as follows:

```
(0.5, 0.5, 0.5, 0.5) * (0.0, 0.0, 1.0, 1.0) = (0.0, 0.0, 0.5, 0.5)
```

Table 16.130. Properties of the **Coloration** widget feature

Property name	Description
`colorationEnabled`	If true, coloration is used
`colorationColor`	The coloration used. Possible values: - Pure - Opaque - White

16.11.3. Focus

The **Focus** widget feature category provides the widget features relating to focus management.

16.11.3.1. Auto focus

With the **Auto focus** widget feature, the order in which child widgets are focused is pre-defined. The **Auto focus** widget feature checks the widget subtree for child widgets with the `focusable` property.

The order of the widgets in the layout is used to calculate focus order. Depending on layout orientation, the algorithm begins in the upper left or upper right corner.

Restrictions:

- The widget feature **Auto focus** automatically adds the **Focused** widget feature.

Table 16.131. 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
`focusFlow`	The behavior for focus changes within the hierarchy. Possible values: - `stop at hierarchy (=0)` - `wrap within hierarchy level (=1)` - `step up in hierarchy (=2)`
`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.

16.11.3.2. User-defined focus

The **User-defined focus** widget feature enables additional focus functionality for the widget. A widget that uses the feature manages a local focus hierarchy for its widget subtree.

Restrictions:

- The widget feature **User-defined focus** automatically adds the **Focused** widget feature.

Table 16.132. Properties of the **User-defined focus** widget feature

Property name	Description
`focusNext`	The trigger that assigns the focus to the next child widget
`focusOrder`	The `focusOrder` property makes it possible to skip child widgets when assigning focus. The ID of a child widget corresponds to its position in the subtree.

Property name	Description
	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 **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: ► `stop at hierarchy level (=0)` ► `wrap within hierarchy level (=1)` ► `step up in hierarchy (=2)`
`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

16.11.4. Gestures

16.11.4.1. Flick gesture

A quick brush of a contact over a surface

Restrictions:

► Adding the **Flick gesture** widget feature automatically adds the **Gestures** and **Touched** widget features.

Table 16.133. Properties of the **Flick gesture** widget feature

Property name	Description
`onGestureFlick`	The 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`

Property name	Description
	► `directionX`: x-part of the direction vector of the gesture ► `directionY`: y-part of the direction vector of the gesture
`flickMaxTime`	The maximal time in milliseconds the contact may stay in place for the gesture to be recognized as a flick gesture
`flickMinLength`	The minimal distance in pixels a contact has to move on the surface to be recognized as a flick gesture

16.11.4.2. Hold gesture

A hold gesture without movement

Restrictions:

- Adding the **Hold gesture** widget feature automatically adds the **Gestures** and **Touched** widget features.
- The **Hold gesture** widget feature does not trigger the **Touch lost** widget feature.

Table 16.134. Properties of the **Hold gesture** widget feature

Property name	Description
`onGestureHold`	The 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
`holdDuration`	The minimal time in milliseconds the contact must stay in place for the gesture to be recognized as a hold gesture

16.11.4.3. Long hold gesture

A long hold gesture without movement

Restrictions:

- Adding the **Long hold gesture** widget feature automatically adds the **Gestures** and **Touched** widget features.
- The **Long hold gesture** widget feature does not trigger the **Touch lost** widget feature.

Table 16.135. Properties of the **Long hold gesture** widget feature

Property name	Description
`onGestureLongHold`	The reaction that is triggered once the gesture is recognized. The reaction is triggered only once per contact: when `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 ► `y`: y-coordinate of the contact position
`longHoldDuration`	The minimal time in milliseconds the contact must stay in place for the gesture to be recognized as a long hold gesture

16.11.4.4. Path gestures

A shape drawn by one contact is matched against a set of known shapes.

Restrictions:

► Adding the **Path gesture** widget feature automatically adds the **Gestures** and **Touched** widget features.

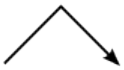
Table 16.136. Properties of the **Path gesture** widget feature

Property name	Description
`onPath`	The reaction that is triggered when the entered shape matches. The reaction is only triggered if `onPathStart` has been triggered already.
`onPathStart`	The reaction that is triggered once a contact moves beyond the minimal box (`pathMinXBox`, `pathMinYBox`.) Reaction argument: ► `gestureId`: ID of the path that was matched
`onPathNotRecognized`	The reaction that triggered when the entered shape does not match. The reaction is only triggered if `onPathStart` has been triggered already.
`pathMinXBox`	The x-coordinate of the minimal distance in pixels a contact must move so that the path gesture recognizer starts considering the input
`pathMinYBox`	The y-coordinate of the minimal distance in pixels a contact must move so that the path gesture recognizer starts considering the input

16.11.4.4.1. Gesture IDs

Gesture identifiers depend on the configuration of the path gesture recognizer. The following table shows an example configuration which is included in EB GUIDE.

Table 16.137. Path gesture samples configuration included in EB GUIDE

ID	Shape	Description
0		Roof shape left to right
1		Roof shape right to left
2		Horizontal line left to right
3		Horizontal line right to left
4		Check mark
5		Wave shape left to right
6		Wave shape right to left

16.11.4.5. Pinch gesture

Two contacts that move closer together or further apart

Restrictions:

- Adding the **Pinch gesture** widget feature automatically adds the **Gestures** and **Touched** widget features.

Table 16.138. Properties of the **Pinch gesture** widget feature

Property name	Description
`onGesturePinchStart`	The 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`	The reaction that is triggered when the pinch ratio or center point change. 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
`onGesturePinchEnd`	The reaction that is triggered once the gesture is finished. 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
`pinchThreshold`	The minimal distance in pixels each contact has to move from its initial position for the gesture to be recognized

16.11.4.6. Rotate gesture

Two contacts that move along a circle

Restrictions:

- Adding the **Rotate gesture** widget feature automatically adds the **Gestures** and **Touched** widget features.

Table 16.139. Properties of the **Rotate gesture** widget feature

Property name	Description
`onGestureRotateStart`	The reaction that is triggered once the start of the gesture is recognized
`onGestureRotateUpdate`	The reaction that is triggered when the recognized angle or center point changes
`onGestureRotateEnd`	The reaction that is triggered once the gesture is finished
`rotateThreshold`	The minimal distance in pixels each contact has to move from its initial position for the start of the gesture to be recognized

Reaction arguments for `onGestureRotateEnd`, `onGestureRotateStart`, and `onGestureRotateUpdate`:

- `angle`: Angle between the line specified by the initial position of the two involved contacts and the line specified by the current position of the two contacts. The angle is measured counter-clockwise.
- `centerX`: x-coordinate of the current center point between the two contacts
- `centerY`: y-coordinate of the current center point between the two contacts

16.11.5. Input handling

16.11.5.1. Gestures

The **Gestures** widget feature enables the widget to react on touch gestures.

Restrictions:

- Adding the **Gestures** widget feature automatically adds the **Touched** widget feature.
- The **Gestures** widget feature has no additional properties.

16.11.5.2. Key pressed

The **Key pressed** widget feature enables a widget to react on a key being pressed.

Restrictions:

- Adding the **Key pressed** widget feature automatically adds the **Pressed** and **Focused** widget features.

Table 16.140. Properties of the **Key pressed** widget feature

Property name	Description
`keyPressed`	The widget's reaction on a key being pressed

Property name	Description
	Reaction argument: ► `keyId`: The ID of the key that is processed

16.11.5.3. Key released

The **Key released** widget feature enables a widget to react on a key being released.

Restrictions:

► Adding the **Key released** widget feature automatically adds the **Pressed** and **Focused** widget features.

Table 16.141. Properties of the **Key released** widget feature

Property name	Description
`keyShortReleased`	The widget's reaction on a key being released Reaction argument: ► `keyId`: The ID of the key that is processed

16.11.5.4. Key status changed

The **Key status changed** widget feature enables a widget to react on a key being pressed or released. It defines the reaction to key input such as **short press**, **long**, **ultra long** and **continuous**.

Restrictions:

► Adding the **Key status changed** widget feature automatically adds the **Pressed** and **Focused** widget features.

Table 16.142. Properties of the **Key status changed** widget feature

Property name	Description
`keyStatusChanged`	The widget's reaction on a key being pressed or released Reaction arguments: ► `keyId`: The ID of the key that is processed ► `status`: The numeric ID of the status change

16.11.5.5. Key unicode

The **Key unicode** widget feature enables a widget to react on Unicode key input.

Restrictions:

- Adding the **Key unicode** widget feature automatically adds the **Pressed** and **Focused** widget features.

Table 16.143. Properties of the **Key unicode** widget feature

Property name	Description
`keyUnicode`	The widget's reaction on a Unicode key input Reaction argument: - `keyId`: The ID of the key that is processed

16.11.5.6. Move in

The **Move in** widget feature enables a widget to react on movement into its boundaries.

Restrictions:

- Adding the **Move in** widget feature automatically adds the **Touched** widget feature.

Table 16.144. Properties of the **Move in** widget feature

Property name	Description
`moveIn`	The widget's reaction on a movement into its boundaries Reaction arguments: - `touchId`: The ID of the touch screen the user has clicked or released - `fingerId`: The ID of the contact that moves across the widget

16.11.5.7. Move out

The **Move out** widget feature enables a widget to react on movement out of its boundaries.

Restrictions:

- Adding the **Move out** widget feature automatically adds the **Touched** widget feature.

Table 16.145. Properties of the **Move out** widget feature

Property name	Description
`moveOut`	The widget's reaction on a movement out of its boundaries Reaction arguments:

Property name	Description
	▶ `touchId`: The ID of the touch screen the user has clicked or released ▶ `fingerId`: The ID of the contact that moves across the widget

16.11.5.8. Move over

The **Move over** widget feature enables a widget to react on movement within its boundaries.

Restrictions:

▶ Adding the **Move over** widget feature automatically adds the **Touched** widget feature.

Table 16.146. Properties of the **Move over** widget feature

Property name	Description
`moveOver`	The widget's reaction on a movement within its boundaries Reaction arguments: ▶ `touchId`: The ID of the touch screen the user has clicked or released ▶ `fingerId`: The ID of the contact that moves across the widget

16.11.5.9. Moveable

The **Moveable** widget feature enables a widget to be moved by touch.

Restrictions:

▶ Adding the **Moveable** widget feature automatically adds the **Touched** and **Touch moved** widget features.

Table 16.147. Properties of the **Moveable** widget feature

Property name	Description
`moveDirection`	The direction into which the widget moves. Possible values: ▶ `horizontal (=0)` ▶ `vertical (=1)` ▶ `free (=2)`

16.11.5.10. Rotary

The **Rotary** widget feature enables a widget to react on being rotated.

Restrictions:

- Adding the **Rotary** widget feature automatically adds the **Focused** widget feature.

Table 16.148. Properties of the **Rotary** widget feature

Property name	Description
`rotaryReaction`	The widget's reaction on being rotated. If true, 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

16.11.5.11. Touch lost

The **Touch lost** widget feature enables a widget to react on a lost touch contact.

A contact can disappear when it is part of a gesture or leaves the touch screen without releasing. In these cases the `touchShortReleased` reaction is not executed.

Restrictions:

- Adding the **Touch lost** widget feature automatically adds the **Touched** widget feature.

Table 16.149. Properties of the **Touch lost** widget feature

Property name	Description
`onTouchGrabLost`	The widget's reaction on a lost touch contact Reaction arguments: - `touchId`: The ID of the touch screen the user has clicked or released - `fingerId`: The ID of the contact that moves across the widget

16.11.5.12. Touch move

The **Touch move** widget feature enables a widget to react on being touched and moved.

Restrictions:

- Adding the **Touch move** widget feature automatically adds the **Touched** widget feature.

Table 16.150. Properties of the **Touch move** widget feature

Property name	Description
`touchMoved`	The widget's reaction on being touched and moved Reaction arguments: ► `touchId`: The ID of the touch screen the user has clicked or released ► `fingerId`: The ID of the contact that moves across the widget

16.11.5.13. Touch pressed

The **Touch pressed** widget feature enables a widget to react on being pressed.

Restrictions:

► Adding the **Touch pressed** widget feature automatically adds the **Touched** widget feature.

Table 16.151. Properties of the **Touch pressed** widget feature

Property name	Description
`touchPressed`	The widget's reaction on being pressed Reaction arguments: ► `touchId`: The ID of the touch screen the user has clicked or released ► `fingerId`: The ID of the contact that moves across the widget

16.11.5.14. Touch released

The **Touch released** widget feature enables a widget to react on being released.

Restrictions:

► Adding the **Touch released** widget feature automatically adds the **Touched** widget feature.

Table 16.152. Properties of the **Touch released** widget feature

Property name	Description
`touchShortReleased`	The widget's reaction on being released Reaction arguments: ► `touchId`: The ID of the touch screen the user has clicked or released ► `fingerId`: The ID of the contact that moves across the widget

16.11.5.15. Touch status changed

The **Touch status changed** widget feature enables a widget to react on changes of its touch status.

Restrictions:

- Adding the **Touch status changed** widget feature automatically adds the **Touched** widget feature.

Table 16.153. Properties of the **Touch status changed** widget feature

Property name	Description

`touchStatusChanged`	The widget's reaction on changes of its touch status
	
	Reaction arguments:
	
	- `touchId`: The ID of the touch screen the user has clicked or released
	- `touchStatus`: The ID of the type of touch
	
	Possible values:
	
	  - `0`: new contact
	  - `1`: touch press
	  - `2`: touch move
	  - `3`: touch released
	  - `4`: movement without touch
	  - `5`: touch gone
	  - `6`: any status change
	- `fingerId`: The ID of the contact that moves across the widget

16.11.6. Layout

16.11.6.1. Absolute layout

The **Absolute layout** widget feature of a parent widget defines the position and size of the child widgets. Invisible child widgets are ignored. The added widget feature properties consist of integer lists. Each list element is mapped to one child widget.

Restrictions:

- The **Absolute layout** widget feature excludes the following widget features:
  - **Box layout**

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

Table 16.154. Properties of the **Absolute layout** widget feature

Property name	Description
`itemLeftOffset`	An integer list that stores the offset from the left border for the child widgets. Each list element is mapped to a child widget.
`itemTopOffset`	An integer list that stores the offset from the top border for the child widgets. Each list element is mapped to a child widget.
`itemRightOffset`	An integer list that stores the offset from the right border for the child widgets. Each list element is mapped to a child widget.
`itemBottomOffset`	An integer list that stores the offset from the bottom border for the child widgets. Each list element is mapped to a child widget.

16.11.6.2. Box layout

The **Box layout** widget feature defines position and size of each child widget.

Position and size properties of child widgets are set by the parent widget. Invisible child widgets are ignored in the calculation.

Restrictions:

- The **Box layout** widget feature excludes the following widget features:
  - **Absolute layout**
  - **Flow layout**
  - **Grid layout**
  - **List layout**

Table 16.155. Properties of the **Box layout** widget feature

Property name	Description
`gap`	The space between two child widgets, depending on the layout direction
`layoutDirection`	The direction in which the list elements i.e. the child widgets are positioned.

16.11.6.3. Flow layout

The **Flow layout** widget feature defines position and size of each child widget.

Position and size properties of child widgets are set by the parent widget. Invisible child widgets are ignored in the calculation.

Restrictions:

- The **Flow layout** widget feature excludes the following widget features:
  - **Absolute layout**
  - **Box layout**
  - **Grid layout**
  - **List layout**

Table 16.156. Properties of the **Flow layout** widget feature

Property name	Description
`horizontalGap`	The horizontal space between two child widgets
`verticalGap`	The vertical space between two child widgets
`layoutDirection`	The direction in which the list elements i.e. the child widgets are positioned.
`horizontalChildAlign`	The horizontal alignment of child widgets
`verticalChildAlign`	The vertical alignment of child widgets - `center (=0)`: The child widget is placed in the center. - `top (=1)`: The child widget is placed at the top - `bottom (=2)`: The child widget is placed at the bottom.

16.11.6.4. Grid layout

The **Grid layout** widget feature defines position and size of each child widget.

Position and size properties of child widgets are set by the parent widget. Invisible child widgets are ignored in the calculation.

Restrictions:

- The **Grid layout** widget feature excludes the following widget features:
  - **Absolute layout**
  - **Box layout**
  - **Flow layout**
  - **List layout**

Table 16.157. Properties of the **Grid layout** widget feature

Property name	Description
`horizontalGap`	The horizontal space between two child widgets
`verticalGap`	The vertical space between two child widgets
`numRows`	Defines the number of rows
`numColumns`	Defines the number of columns

16.11.6.5. Layout margins

The **Layout margins** widget feature adds configurable margins to a widget that uses the **Flow layout**, **Absolute layout**, **Box layout**, or **Grid layout** widget feature.

Table 16.158. Properties of the **Layout margins** widget feature

Property name	Description
`leftMargin`	The margin of the left border
`topMargin`	The margin of the top border
`rightMargin`	The margin of the right border
`bottomMargin`	The margin of the bottom border

16.11.6.6. List layout

The **List layout** widget feature defines position and size of each child widget.

Position properties of child widgets and the `listIndex` property of the **List index** widget feature are set by the parent widget.

Best used in conjunction with instantiators to create the child widgets.

For details about the **List index** widget feature, see section 16.11.7.2, “List index”.

Restrictions:

- The **List layout** widget feature is intended to be used with instantiator.
- The **List layout** widget feature excludes the following widget features:
  - **Absolute layout**
  - **Box layout**
  - **Flow layout**
  - **Grid layout**

Table 16.159. Properties of the **List layout** widget feature

Property name	Description
`layoutDirection`	The direction in which the list elements i.e. the child widgets are positioned.
`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.
`firstListIndex`	The list index of the first visible list element, defined by the widget feature
`scrollIndex`	The base list index the `scrollOffset` property applies to. Scrolling starts at the list elements given in the `scrollIndex` property.
`scrollValue`	The current scroll value
`scrollValueMax`	The maximum scroll value, which is mapped to the end of the list
`scrollValueMin`	The minimum scroll value, which is mapped to the beginning of the list
`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.
`segments`	For horizontal layout direction: the number of rows For vertical layout direction: the number of columns
`listLength`	The number of list elements
`wrapAround`	Possible values: ► true: The `scrollValue` property continues at the inverse border, if `scrollValueMin` or `scrollValueMax` is exceeded. ► false: The `scrollValue` property does not decrease/increase, if `scrollValueMin` or `scrollValueMax` is exceeded.

16.11.6.7. Scale mode

The **Scale mode** widget feature defines how an image is displayed if its size differs from the size of the widget.

Restrictions:

► The **Scale mode** widget feature is only available for the widget image.

Table 16.160. Properties of the **Scale mode** widget feature

Property name	Description
`scaleMode`	The scale mode of the image. Possible values: ► `0 = original size` ► `1 = fit to size` ► `2 = keep aspect ratio`

16.11.7. List management

16.11.7.1. Line index

The **Line index** widget feature defines the unique position for each line of your list or table.

Restrictions:

► The **Line index** widget feature is intended to be used in combination with instantiators.

Table 16.161. Properties of the **Line index** widget feature

Property name	Description
`lineIndex`	The index of the current line in a table

16.11.7.2. List index

The **List index** widget feature defines the unique position of a widget in a list.

Restrictions:

► The **List index** widget feature is intended to be used in combination with the **List layout** widget feature.

Table 16.162. Properties of the **List index** widget feature

Property name	Description
`listIndex`	The index of the current widget in a list

16.11.7.3. Template index

The **Template index** widget feature defines the unique position of the used line template.

Restrictions:

- The **Template index** widget feature is intended to be used in combination with instantiators.

Table 16.163. Properties of the **Template index** widget feature

Property name	Description
`lineTemplateIndex`	The index of the used line template

16.11.7.4. Viewport

The **Viewport** widget feature clips oversized elements at the widget borders.

Restrictions:

- The **Viewport** widget feature is intended to be used in combination with containers or lists.
- The **Viewport** widget feature takes effect on the following model elements:
  - Child widgets of the widget you added **Viewport** to are clipped inside the dimensions of the widget.
  - The widget you added **Viewport** is clipped inside the dimensions of its parent view.

Table 16.164. Properties of the **Viewport** widget feature

Property name	Description
`xOffset`	The horizontal offset of the visible clipping within the drawn area of child widgets
`yOffset`	The vertical offset of the visible clipping within the drawn area of child widgets

16.11.8. 3D

Widget features in the **3D** category are only available for 3D widgets.

16.11.8.1. Camera viewport

The **Camera viewport** widget feature defines the camera's drawing region within the scene graph.

Restrictions:

- The **Camera viewport** widget feature is available for camera.

Table 16.165. Properties of the **Camera viewport** widget feature

Property name	Description
`viewportX`	The x-origin of the viewport within the scene graph

Property name	Description
`viewportY`	The y-origin of the viewport within the scene graph
`viewportWidth`	The viewport's width in pixels
`viewportHeight`	The viewport's height in pixels

16.11.8.2. Ambient texture

The **Ambient texture** widget feature adds extended configuration values to a material.

Restrictions:

- The **Ambient texture** widget feature is available for material.

Table 16.166. Properties of the **Ambient texture** widget feature

Property name	Description
`ambientTexture`	The file name of the texture
`ambientTextureAddressModeU`	The address mode of the texture along the u-direction. Possible values: - `repeat (=0)`: When accessed outside the texture bounds, the texture is repeated. Also known as wrap or tile - `clamp (=1)`: When accessed outside the texture bounds, the pixels at the edge of the texture are used.
`ambientTextureAddressModeV`	The address mode of the texture along the v-direction. Possible values: - `repeat (=0)`: When accessed outside the texture bounds, the texture is repeated. Also known as wrap or tile - `clamp (=1)`: When accessed outside the texture bounds, the pixels at the edge of the texture are used.
`ambientFilterMode`	The filtering mode of the texture. Possible values: - `point (=0)`: Texture is not smoothed at all. Least expensive but prone to aliasing artifacts when texture is minimized. - `linear (=1)`: Also known as bilinear filtering. Smoothens the texture when minimized to reduce aliasing artifacts. - `trilinear (=2)`: Most expensive, but yields better results than linear filtering.

16.11.8.3. Diffuse texture

The **Diffuse texture** widget feature adds extended configuration values to a material.

Restrictions:

- The **Diffuse texture** widget feature is available for material.

Table 16.167. Properties of the **Diffuse texture** widget feature

Property name	Description
`diffuseTexture`	The file name of the texture
`diffuseTextureAddressModeU`	The address mode of the texture along the u-direction. Possible values: - `repeat (=0)`: When accessed outside the texture bounds, the texture is repeated. Also known as wrap or tile - `clamp (=1)`: When accessed outside the texture bounds, the pixels at the edge of the texture are used.
`diffuseTextureAddressModeV`	The address mode of the texture along the v-direction. Possible values: - `repeat (=0)`: When accessed outside the texture bounds, the texture is repeated. Also known as wrap or tile - `clamp (=1)`: When accessed outside the texture bounds, the pixels at the edge of the texture are used.
`diffuseFilterMode`	The filtering mode of the texture. Possible values: - `point (=0)`: Texture is not smoothed at all. Least expensive but prone to aliasing artifacts when texture is minimized. - `linear (=1)`: Also known as bilinear filtering. Smoothens the texture when minimized to reduce aliasing artifacts. - `trilinear (=2)`: Most expensive, but yields better results than linear filtering.

16.11.8.4. Emissive texture

The **Emissive texture** widget feature adds extended configuration values to a material.

Restrictions:

- The **Emissive texture** widget feature is available for material.

Table 16.168. Properties of the **Emissive texture** widget feature

Property name	Description
`emissiveTexture`	The file name of the texture

Property name	Description
`emissiveTextureAddress-ModeU`	The address mode of the texture along the u-direction. Possible values: ► `repeat (=0)`: When accessed outside the texture bounds, the texture is repeated. Also known as wrap or tile ► `clamp (=1)`: When accessed outside the texture bounds, the pixels at the edge of the texture are used.
`emissiveTextureAddressMod-eV`	The address mode of the texture along the v-direction. Possible values: ► `repeat (=0)`: When accessed outside the texture bounds, the texture is repeated. Also known as wrap or tile ► `clamp (=1)`: When accessed outside the texture bounds, the pixels at the edge of the texture are used.
`emissiveFilterMode`	The filtering mode of the texture. Possible values: ► `point (=0)`: Texture is not smoothed at all. Least expensive but prone to aliasing artifacts when texture is minimized. ► `linear (=1)`: Also known as bilinear filtering. Smoothens the texture when minimized to reduce aliasing artifacts. ► `trilinear (=2)`: Most expensive, but yields better results than linear filtering.

16.11.8.5. Light map texture

The **Light map texture** widget feature adds extended configuration values to a material.

Restrictions:

► The **Light map texture** widget feature is available for material.

Table 16.169. Properties of the **Light map texture** widget feature

Property name	Description
`lightMapTexture`	The file name of the texture
`lightMapTextureAddress-ModeU`	The address mode of the texture along the u-direction. Possible values: ► `repeat (=0)`: When accessed outside the texture bounds, the texture is repeated. Also known as wrap or tile ► `clamp (=1)`: When accessed outside the texture bounds, the pixels at the edge of the texture are used.

Property name	Description
`lightMapTextureAddressModeV`	The address mode of the texture along the v-direction. Possible values: ► `repeat (=0)`: When accessed outside the texture bounds, the texture is repeated. Also known as wrap or tile ► `clamp (=1)`: When accessed outside the texture bounds, the pixels at the edge of the texture are used.
`lightMapFilterMode`	The filtering mode of the texture. Possible values: ► `point (=0)`: Texture is not smoothed at all. Least expensive but prone to aliasing artifacts when texture is minimized. ► `linear (=1)`: Also known as bilinear filtering. Smoothens the texture when minimized to reduce aliasing artifacts. ► `trilinear (=2)`: Most expensive, but yields better results than linear filtering.

16.11.8.6. Normal map texture

The **Normal map** widget feature adds extended configuration values to a material.

Restrictions:

► The **Normal map texture** widget feature is available for material.

Table 16.170. Properties of the **Normal map** widget feature

Property name	Description
`normalMapTexture`	The file name of the texture
`normalMapTextureAddressModeU`	The address mode of the texture along the u-direction. Possible values: ► `repeat (=0)`: When accessed outside the texture bounds, the texture is repeated. Also known as wrap or tile ► `clamp (=1)`: When accessed outside the texture bounds, the pixels at the edge of the texture are used.
`normalMapTextureAddressModeV`	The address mode of the texture along the v-direction. Possible values: ► `repeat (=0)`: When accessed outside the texture bounds, the texture is repeated. Also known as wrap or tile ► `clamp (=1)`: When accessed outside the texture bounds, the pixels at the edge of the texture are used.
`normalMapFilterMode`	The filtering mode of the texture. Possible values:

Property name	Description
	▶ `point (=0)`: Texture is not smoothed at all. Least expensive but prone to aliasing artifacts when texture is minimized. ▶ `linear (=1)`: Also known as bilinear filtering. Smoothens the texture when minimized to reduce aliasing artifacts. ▶ `trilinear (=2)`: Most expensive, but yields better results than linear filtering.

16.11.8.7. Opaque texture

The **Opaque texture** widget feature adds extended configuration values to a material.

Restrictions:

- The **Opaque texture** widget feature is available for material.

Table 16.171. Properties of the **Opaque texture** widget feature

Property name	Description
`opaqueTexture`	The file name of the texture
`opaqueTextureAddressModeU`	The address mode of the texture along the u-direction. Possible values: ▶ `repeat (=0)`: When accessed outside the texture bounds, the texture is repeated. Also known as wrap or tile ▶ `clamp (=1)`: When accessed outside the texture bounds, the pixels at the edge of the texture are used.
`opaqueTextureAddressModeV`	The address mode of the texture along the v-direction. Possible values: ▶ `repeat (=0)`: When accessed outside the texture bounds, the texture is repeated. Also known as wrap or tile ▶ `clamp (=1)`: When accessed outside the texture bounds, the pixels at the edge of the texture are used.
`opaqueFilterMode`	The filter mode of the texture. Possible values: ▶ `point (=0)`: Texture is not smoothed at all. Least expensive but prone to aliasing artifacts when texture is minimized. ▶ `linear (=1)`: Also known as bilinear filtering. Smoothens the texture when minimized to reduce aliasing artifacts. ▶ `trilinear (=2)`: Most expensive, but yields better results than linear filtering.

16.11.8.8. Reflection texture

The **Reflection texture** widget feature adds extended configuration values to a material.

Restrictions:

- The **Reflection texture** widget feature is available for material.

Table 16.172. Properties of the **Reflection texture** widget feature

Property name	Description
`reflectionTopTexture`	The file name of the texture
`reflectionBottomTexture`	The file name of the texture
`reflectionLeftTexture`	The file name of the texture
`reflectionRightTexture`	The file name of the texture
`reflectionFrontTexture`	The file name of the texture
`reflectionBackTexture`	The file name of the texture
`reflectionFilterMode`	The filtering mode of the texture. Possible values: - `point (=0)`: Texture is not smoothed at all. Least expensive but prone to aliasing artifacts when texture is minimized. - `linear (=1)`: Also known as bilinear filtering. Smoothens the texture when minimized to reduce aliasing artifacts. - `trilinear (=2)`: Most expensive, but yields better results than linear filtering.

NOTE

Reflection texture widget feature

EB GUIDE Studio displays the **Reflection texture** widget feature, only when an image file is selected for all of the following properties:

- `reflectionTopTexture`
- `reflectionBottomTexture`
- `reflectionLeftTexture`
- `reflectionRightTexture`
- `reflectionFrontTexture`
- `reflectionBackTexture`

The image files must have the same size.

16.11.8.9. Specular texture

The **Specular texture** widget feature adds extended configuration values to a material.

Restrictions:

- The **Specular texture** widget feature is available for material.

Table 16.173. Properties of the **Specular texture** widget feature

Property name	Description
`specularTexture`	The file name of the texture
`specularTextureAddressModeU`	The address mode of the texture along the u-direction. Possible values: - `repeat (=0)`: When accessed outside the texture bounds, the texture is repeated. Also known as wrap or tile - `clamp (=1)`: When accessed outside the texture bounds, the pixels at the edge of the texture are used.
`specularTextureAddressModeV`	The address mode of the texture along the v-direction. Possible values: - `repeat (=0)`: When accessed outside the texture bounds, the texture is repeated. Also known as wrap or tile - `clamp (=1)`: When accessed outside the texture bounds, the pixels at the edge of the texture are used.
`specularFilterMode`	The filtering mode of the texture. Possible values: - `point (=0)`: Texture is not smoothed at all. Least expensive but prone to aliasing artifacts when texture is minimized. - `linear (=1)`: Also known as bilinear filtering. Smoothens the texture when minimized to reduce aliasing artifacts. - `trilinear (=2)`: Most expensive, but yields better results than linear filtering.

16.11.9. Transformation

The widget features of the category **Transformation** modify location, form, and size of widgets.

The order in which transformations are executed is equal to the order in the widget tree. If multiple transformations are applied to one widget at the same widget tree hierarchy level, the order is as follows:

1. Translation
2. Shearing
3. Scaling

4. Rotation around z-axis
5. Rotation around y-axis
6. Rotation around x-axis

16.11.9.1. Pivot

The **Pivot** widget feature defines the pivot point of transformations which are applied to the widget. If no pivot point is configured, the default pivot point is at (0.0, 0.0, 0.0).

Restrictions:

- Adding the **Pivot** widget feature automatically adds the **Rotation**, **Scaling** and **Shearing** widget features.

Table 16.174. Properties of the **Pivot** widget feature

Property name	Description
`pivotX`	The pivot point on the x-axis relative to parent widget
`pivotY`	The pivot point on the y-axis relative to parent widget
`pivotZ`	The pivot point on the z-axis relative to parent widget if widget is a scene graph

16.11.9.2. Rotation

The **Rotation** widget feature is used to rotate the widget and its subtree.

Table 16.175. Properties of the **Rotation** widget feature

Property name	Description
`rotationEnabled`	Defines whether rotation is used or not
`rotationAngleX`	The rotation angle on the x-axis. This property only affects scene graph.
`rotationAngleY`	The rotation angle on the y-axis. This property only affects scene graph.
`rotationAngleZ`	The rotation angle on the z-axis

16.11.9.3. Scaling

The **Scaling** widget feature is used to scale the widget and its subtree.

Table 16.176. Properties of the **Scaling** widget feature

Property name	Description
`scalingEnabled`	Defines whether scaling is used or not

Property name	Description
scalingX	The scaling on the x-axis in percent
scalingY	The scaling on the y-axis in percent
scalingZ	The scaling on the z-axis in percent if widget is a scene graph

16.11.9.4. Shearing

The **Shearing** widget feature is used to distort widgets in the widget subtree.

Table 16.177. Properties of the **Shearing** widget feature

Property name	Description
shearingEnabled	Defines whether shearing is used or not
shearingXbyY	The shearing amount of x-axis by y-axis
shearingXbyZ	The shearing amount of x-axis by z-axis if widget is a scene graph
shearingYbyX	The shearing amount of y-axis by x-axis
shearingYbyZ	The shearing amount of y-axis by z-axis if widget is a scene graph
shearingZbyX	The shearing amount of z-axis by x-axis if widget is a scene graph
shearingZbyY	The shearing amount of z-axis by y-axis if widget is a scene graph

16.11.9.5. Translation

The **Translation** widget feature is used to translate the widget and its subtree. It moves widgets in x, y and z directions.

Table 16.178. Properties of the **Translation** widget feature

Property name	Description
translationEnabled	Defines whether translation is used or not
translationX	The translation on the x-axis
translationY	The translation on the y-axis
translationZ	The translation on the z-axis if widget is a scene graph

Glossary

#

3D graphic — A 3D graphic is a virtual picture of a 3D scene. A 3D scene is a collection of 3D models (meshes or shapes), materials, light sources, and cameras. Materials define the visual appearance of 3D models through colors and textures and the behavior under virtual lighting. A camera provides the view point from where a virtual picture of the 3D scene is taken.

A

API — Application programming interface

C

communication context — The communication context describes the environment in which communication occurs. Each communication context is identified by a unique numerical ID.

D

datapool — The datapool is a data cache in an EB GUIDE model that provides access to datapool items during run-time. It is used for data exchange between the application and the HMI.

datapool item — Datapool items store and exchange data. Each item in the datapool has a communication direction.

E

EB GUIDE GTF — EB GUIDE GTF is the graphics target framework of the EB GUIDE product line and is part of EB GUIDE TF. EB GUIDE GTF represents the run-time environment to execute EB GUIDE models on target devices.

EB GUIDE GTF SDK — EB GUIDE GTF SDK is the development environment contained in EB GUIDE GTF. It is a sub-set of the EB GUIDE SDK. Another sub-set is the EB GUIDE Studio SDK.

EB GUIDE model — An EB GUIDE model is the description of an HMI created with EB GUIDE Studio.

EB GUIDE product line	The EB GUIDE product line is a collection of software libraries and tools which are needed to specify an HMI model and convert the HMI model into a graphical user interface that runs on an embedded environment system.
EB GUIDE Script	EB GUIDE Script is the scripting language of the EB GUIDE product line. EB GUIDE Script enables accessing the datapool, model elements such as widgets and the state machine, and system events.
EB GUIDE SDK	EB GUIDE SDK is a product component of EB GUIDE. It is the software development kit for the EB GUIDE product line. It includes the EB GUIDE Studio SDK and the EB GUIDE GTF SDK.
EB GUIDE Studio	EB GUIDE Studio is the tool for modeling and specifying an HMI with a graphical user interfaces.
EB GUIDE Studio SDK	EB GUIDE Studio SDK is an application programming interface (API) to communicate with EB GUIDE Studio. It is a sub-set of the EB GUIDE SDK. Another sub-set is the EB GUIDE GTF SDK.
EB GUIDE TF	EB GUIDE TF is the run-time environment of the EB GUIDE product line. It consists of EB GUIDE GTF and EB GUIDE STF. It is required to run an EB GUIDE model.

G

GL	Graphical library
GUI	Graphical user interface

H

HMI	Human machine interface

L

library	A library is a set of resources used in EB GUIDE Studio. Libraries that are necessary for an EB GUIDE project are defined in the project center.

M

model element	A model element is an object within an EB GUIDE model, for example a state, a widget, or a datapool item. See Also EB GUIDE model.

O

OS	Operating system

P

profile	In the project center, a profile is a set of specifications. In a profile you define libraries, messages and scenes for your project. During export of an EB GUIDE model the data in the profile is written to the `model.json` configuration file.
project center	All project-related functions are located in the project center, for example profiles and languages.
project editor	In the project editor you model the behavior and the appearance of the human machine interface.

R

resource	A resource is a data package that is part of the EB GUIDE project. Examples for resources are fonts, images, meshes. Resources are stored outside of the EB GUIDE model, for example in files, depending on the operating system.

S

shared library	A shared library, as opposed to a static library, can be loaded when preparing a program for execution. On Windows platforms shared libraries are called dynamic link libraries and have a `.dll` file extension. On Unix systems shared libraries are called shared objects and have an `.so` file extension.
state	A state defines the status of the state machine. States and state transitions are modeled in state charts.
state machine	A state machine is a set of states, transitions between those states, and actions. A state machine describes the dynamic behavior of the system.

T

transition	A transition defines the change from one state to another. A transition is usually triggered by an event.

U

UI	User interface

V

view	A view is a graphical representation of a project-specific HMI-screen and is related to a specific state machine state. A view consists of a tree of widgets.

W

widget	A widget is a basic graphical element. Widgets are used for interaction with a graphical user interface.

Index

Symbols

A

B

C

D

L

M

N

O

P

Q

R

S

T

U

V

W