

GSE UserManual

Product:	Guiliani Streaming Editor (GSE)
Release version:	2.4
Release date:	April 14, 2021

Table of contents

Disclaimer	5
1.1 System Requirements	7
1.2 Documentation conventions	7
2 Introduction	8
2.1 What Is Guiliani?	8
2.2 What Is The Guiliani Streaming Editor (GSE)?	8
2.2.1 Main Features of Guiliani Streaming Editor	9
2.2.2 The Guiliani Workflow	9
2.3 An introduction to the principles of Guiliani.....	10
2.3.1 Streaming and the StreamRuntime.....	10
2.3.2 Resource Management	11
2.3.3 Skinning (Image-Sets, Property-Sets)	11
2.3.4 Multi-Language (Language-Sets, Font-Sets)	11
2.3.5 Command Objects	12
2.3.6 Behaviour Objects	12
2.3.7 Difference between Command and Behaviour	12
2.3.8 DataPool	13
2.3.9 Nine Patch	13
2.3.10 Export Format	14
2.3.11 Tool Tips	17
2.4 StreamRuntime Application	18
2.4.1 Overview	18
2.4.2 Directory structure.....	18
2.4.3 Command-Line Options	19
2.4.4 CMake Configuration.....	19

2.4.5	How to set up resource-file or resource-header.....	20
2.4.6	How to take screenshots	21
3	The Menu Bar.....	22
3.1	The File Menu	23
3.2	New Project	24
3.2.1	Open Project.....	24
3.2.2	Recent Projects	25
3.2.3	Save Project.....	26
3.2.4	Save Project As	27
3.2.5	New Dialog	28
3.2.6	Settings	29
3.2.7	Run Simulation.....	35
3.2.8	Quit.....	38
3.3	The Edit Menu	39
3.3.1	Undo / Redo	40
3.3.2	Copy / Cut / Paste / Delete	40
3.3.3	Select All	40
3.3.4	Find.....	41
3.4	The View Menu	41
3.5	The Layout Menu	43
3.5.1	Arrange Objects.....	43
3.5.2	Align Objects.....	44
3.6	The Resources Menu	46
3.6.1	Manage	47
3.6.2	Import	60
3.6.3	Import PSD-File	61
3.6.4	Export.....	67
3.6.5	Manage Sets	69
3.7	Custom Extensions	71
3.7.1	Create Custom Extension	71
3.7.2	Generate Custom StandardFactory.....	72
3.8	The Windows Menu	73
3.9	The Help Menu.....	74

4	The Dialog-List-Window	75
5	The Workspace-Window.....	76
5.1	Moving and resizing objects.....	76
5.2	Guidelines.....	77
5.3	Tools.....	78
6	The Attributes Window.....	79
6.1	Images.....	79
6.2	Texts.....	80
6.3	Fonts.....	80
6.4	Commands.....	80
6.5	Behaviours.....	81
6.6	Basic Object Specific Attributes.....	82
6.6.1	Position.....	82
6.6.2	Width and Height.....	82
6.6.3	Focusable, GrayedOut, Disabled and Invisible.....	82
6.6.4	BehaviourClassID.....	83
6.6.5	LayouterClassID.....	83
6.6.6	Object ID.....	83
6.6.7	Colors.....	85
7	The Object Hierarchy Window.....	86
7.1	Selecting objects.....	86
7.2	Change hierarchical order of objects.....	86
7.3	Edit objects.....	86
7.4	Visibility of objects.....	87
7.5	Zoom in/out.....	87
7.6	Expand or collapse object tree.....	88
7.7	Additional Object-Information.....	88
8	The Controls Window.....	89
8.1	Images/Primitives.....	90
8.1.1	Geometry Object.....	90
8.1.2	Image*.....	90
8.1.3	Animated Image.....	90
8.1.4	Image Stack.....	90

8.2	Standard	91
8.2.1	Text Field	91
8.2.2	Scrolling Text Field	91
8.2.3	Input Field*	91
8.2.4	Combo Box*	91
8.3	Buttons	92
8.3.1	Button*	92
8.3.2	Icon Button*	92
8.3.3	Blend Button*	92
8.3.4	Check Box*	92
8.3.5	Radio Button*	92
8.4	Slider and Bars	93
8.4.1	Horizontal and Vertical Slider/Scrollbar*	93
8.4.2	Progress Bar*	93
8.4.3	Circular Slider*	93
8.4.4	Knob	93
8.4.5	Segment Bar	93
8.4.6	Range Slider	93
8.5	Container	94
8.5.1	CompositeObject	94
8.5.2	Radio Button Group	94
8.5.3	Reposition Composite	94
8.5.4	Center Focus Container	94
8.5.5	Scroll View*	95
8.5.6	Touch Scroll View	95
8.5.7	Carousel	95
8.6	Advanced	96
8.6.1	Gauge	96
8.6.2	Wheel	96
8.6.3	Keyboard	96
8.6.4	Calendar	96
8.6.5	Chart	96
8.6.6	Clock	96

8.7	Custom Extensions	97
8.7.1	Example Control	97
9	Animation Window	98
9.1	Animation Chain List	100
9.2	Animation List	100
9.2.1	Timeline Navigation.....	100
9.2.2	Animation Control.....	102
9.2.3	Animation Timeline	104
9.3	Animation Attributes	106
9.3.1	General Attributes	106
9.3.2	AnimationClassID	107
9.3.3	GUIAnimationSize.....	107
9.3.4	GUIAnimationStdGuiObject.....	107
9.3.5	GUIAnimationTrigger.....	107
10	Debugging and Trouble Shooting	108
10.1	The Console Window	108
10.2	Dealing with corrupted XML Files	108
10.3	Debugging StreamRuntime	109
11	Contacts.....	110

Disclaimer

The information in this document is subject to change without notice and does not represent a commitment on the part of TES Electronic Solutions GmbH. The software described by this document is subject to a License Agreement and may not be copied to other media. No part of this publication may be copied, reproduced or otherwise transmitted or recorded, for any purpose, without prior written permission by TES Electronic Solutions GmbH, hereinafter referred to as TES. “TES Electronic Solutions”, “Guiliani” and associated logos are (registered) trademarks of TES Electronic Solutions GmbH.

Windows, Windows Vista and DirectSound are registered trademarks of Microsoft Corporation in the United States and/or other countries.

All other trademarks are the property of their respective owners and use of them does not imply any affiliation with or endorsement by them.

Document authored by: Guiliani Team

support@guiliani.de

www.guiliani.de

1.1 System Requirements

Your computer system needs to meet certain requirements in order to run GSE.

Any system capable of running Windows 7 or higher or Ubuntu Linux 14.04 or higher should be sufficient to run the GSE.

A graphics-card that supports OpenGL 2.1 or higher is also required.

Supported Operating Systems:


- Windows 7, Windows 8 or Windows 10 (latest Service Pack)
- Ubuntu Linux 14.04


1.2 Documentation conventions

Whenever you can use keys from your computer's keyboard, these will be displayed in square brackets (e.g., "To run your project press [Ctrl] + [r].").


Menu commands or file path used in this document will be shown in *italic*.

Text that appears in the software on controls will be printed in **bold and blue**.


 Whenever the reader of this document has to do something in his project, the text will start with this triangle.

 Results will be shown using this arrow.

In this document, we use icons whenever we will warn the user or will give him additional or important information.

 The speech bubble icon will show additional helpful information.

 Whenever a text begins with an exclamation mark icon, it contains important information that is essential for the current chapter.

 A warning sign icon signals serious issues and potential risks that require your full attention.

2 Introduction

We welcome you to Guiliani and the Guiliani Streaming Editor. Guiliani is your solution for graphical user interfaces (GUI) on embedded systems, which combines the comfort of a PC based development tool-chain with the benefits of a highly optimized software framework specifically designed for use on embedded hardware.

Profit from a mature software that has been used for years in tens of millions of devices already and start using Guiliani now!

2.1 What Is Guiliani?

Guiliani is the abbreviation of **G**raphical **U**ser **I**nterface, **L**ight and **I**nnovative with **A**nimations.

Guiliani is a C++ software framework enabling creation of visually appealing, platform independent GUI's for desktop, mobile and embedded systems.

Guiliani adopts the philosophy of “write once, compile & run” on target hardware. Run a once developed application natively on all supported target platforms. When using Guiliani the usual development workflow is simple:

1. Design the application on a PC
2. Target a set of embedded platforms for production release (for example a Linux-based platform using OpenGL-accelerated graphics, and a Windows Embedded Compact system based on pure software-rendering)
3. Compile and run your application in the desktop simulation or directly on the target.

Guiliani features very high quality visual appearance using sub-pixel rendering, including advanced functionality such as alpha blending and anti-aliasing, making optimum use of the underlying graphics API. This enables development of appealing GUIs for applications running on a wide range of embedded devices, spanning from cost-optimized to high-end hardware platforms.

2.2 What Is The Guiliani Streaming Editor (GSE)?

The streaming editor is a comprehensive tool for Artists, Interaction Designers and Developers to create and develop intuitive user interface with the ability to rapid prototype and deploy on embedded targets. The streaming editor was built using Guiliani and supports all features that are integrated into Guiliani such as high portability, support of different OS (e.g. Windows, Linux), support for multiple languages etc.

In addition, the editor brings some extra features for efficient GUI development like WYSIWYG capabilities, a run-time environment to simulate the developed user interface, a resource generator

for exporting the generated resources, the Guiliani C++ HMI Framework for application and event binding and many more.

The streaming editor is included in the SDK. This allows an instant development start with Guiliani and its GSE.

2.2.1 Main Features of Guiliani Streaming Editor

- Support of standard widgets, including property editor
- Resource Manager to manage Images, Fonts, Sounds and other Resources
- Multi-Language support (switchable during Runtime)
- Object hierarchy tree browser
- Workspace management
- Export Engine (including several export-formats)
- Guiliani runtime engine for multi-platform GUI simulation
- Font support through Freetype, Bitmap Fonts or Cleartype
- Interaction editor (e.g. switch dialogs, toggle object-states)
- Text layouting
- Drag & Drop of Widgets
- Expandability (e.g. custom, custom widgets)
- DataPool-Editor
- Built-in Requirements-Editor
- Fast Color-selection using Color-Picker

2.2.2 The Guiliani Workflow

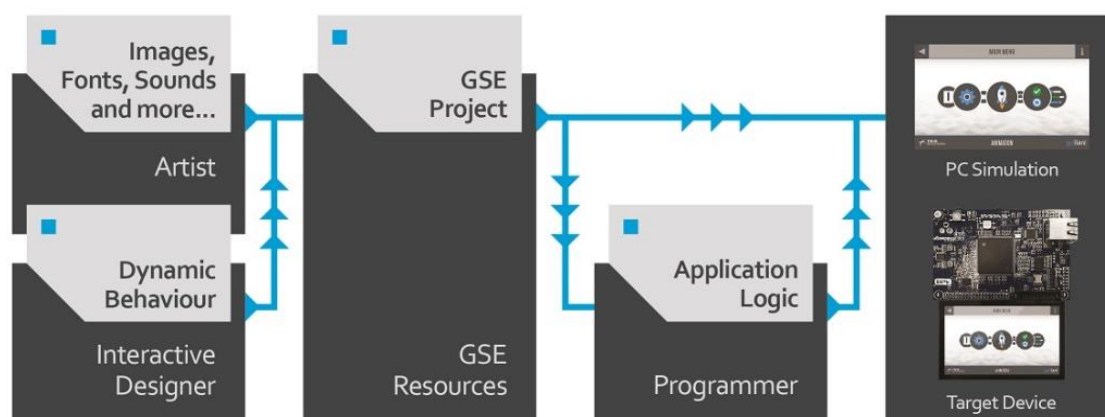


Fig 1 - The Guiliani Workflow

1. Artist creates HMI graphics (e.g. bitmap graphics for GUI background or buttons).
2. Interaction designer stitches together the event based application flow.
3. GSE's resource generator creates a XML or binary file and copied Resources files (.h, .lng, .png, jpeg).
4. Programmer can visualize the GUI interaction using these resources and the Guiliani run-time executable.
5. The produced resources can be exported to an embedded target and visualized using the run-time port (StreamRuntime.exe).
6. Developer programs the application logic and binds data to the GUI
7. Resulting GUI and application on the target

2.3 An introduction to the principles of Guiliani

This chapter gives a brief introduction to the core concepts of Guiliani. If you are interested in learning more about them and the technical concepts involved in using them, please refer to the Guiliani documentation.

2.3.1 Streaming and the StreamRuntime

Streaming means the reconstruction of a GUI at runtime from a descriptive file. This file can be of arbitrary format - usually this will be an XML description (for human readability) - or proprietary binary file (for optimized performance).

Such a streaming file contains all required information to construct the GUI:

- The objects used in the dialogs in a hierarchical way
- The Position and size of these objects
- References to resources used in the GUI (e.g. images, texts, etc.)
- Dynamic behaviour assigned to objects (e.g. Commands)
- Attributes specific to a certain object (e.g. value range and preset of a Slider)
- And many more...

This information must be present within the streaming file, but the way in which it is represented is variable. Therefore, the file may have any format - and in fact, it is not limited to a physical file somewhere on local data storage, but could as well be read from a streaming source such as a network connection.

Another core concept of streaming is that each streamable object (i.e. each object within the GUI) is capable of reading and writing itself from/to a stream. In other words, the object itself knows which attributes it must read/write in order to completely (re)store its current state. This has the advantage that the overall complexity of the system is greatly reduced, since each object encapsulates its own streaming details within its own code.

Streaming is therefore the key concept for GSE itself. In fact, the editor stores its projects as a set of streaming files in XML (or binary) syntax. These files include all the information specified by the UI Designer during the process of building the GUI within GSE and have the same content as the files used on target.

The StreamRuntime is a basic Guiliani application, whose sole purpose is to initialize the Guiliani framework for the given target platform, and to read the files generated by GSE. Whether you hit “Run Simulation” within the editor or execute your final application on the target system, it will launch the StreamRuntime. This sets up the Guiliani framework and reads your UI designs from the streaming file(s). Of course, you are free to extend the plain standard StreamRuntime with application-code for your needs – its purpose is merely to serve as a slim entry point for getting your user interface up and running.

2.3.2 Resource Management

A GUI usually consists of a variety of resources such as images, fonts and sounds, which it uses to visualize itself. Efficient management of these resources is required to optimize the memory and CPU usage of your application. This is critical especially on embedded systems, where system resources are typically very limited.

Guiliani enables you to share resources among various objects within the GUI. Further, it allows you to tune the way in which they are loaded / unloaded so that it best matches your target setup. Usually this will be a trade-off between loading-times and amount of memory usage.

The core of this principle is that all resources will be referenced via abstract identifiers (Resource IDs) that serve as an additional level of indirection between concrete files on the file-system and their usage within the GUI.

2.3.3 Skinning (Image-Sets, Property-Sets)

Your GUI can easily support different looks by using image- and property-sets. This enables you to switch the look simply by switching the sets during design and even during runtime.

Image-Sets will contain different versions of the images used in your GUI while the Property-sets can have different colours which are used for texts and other elements.

2.3.4 Multi-Language (Language-Sets, Font-Sets)

To use different languages in your GUI you specify one text-set for each language and define the translations for each text. So you can switch between different languages during design and even during runtime.

Note: to use languages which are not based on the Latin alphabet, e.g. Asian or east-European languages, a special font containing these special glyphs will be needed to correctly display texts in these languages. If you want to switch to one of these languages in your GUI you will also need to switch the font. So it is advised to have a separate font-set for each of these languages.

2.3.5 Command Objects

Command objects represent a certain behaviour within Guiliani; in reaction to user input. For instance clicking on a button, or moving a slider may execute a command object. You may see a command object as a predefined action (or chain of actions) that will be executed if a certain condition is fulfilled. The command object's Do() method describes this action.

A typical use case for command objects is the communication between the user interface and the underlying application logic. By deploying commands, you will gain three major advantages:

- Commands are thread safe. They will be serialized and executed in the GUI's thread context.
- They decouple GUI and application development. You can start developing the GUI even while the underlying application logic is not available - and vice versa. The typical work flow would be to start off with empty command-stubs whose Do() methods only include debug messages and attach the actual application-binding commands once they are available.
- Commands simplify automated testing. You can test the application-API through the respective command-objects without the GUI.

2.3.6 Behaviour Objects

The methods of CGUIBehaviour define Guiliani's standard event handling slots. These slots will be called by CGUIEventHandler automatically in response to specific user actions.

Thus if a control needs special handling for the associated events it can re-implement the corresponding slots with the desired functionality. When inheriting customized controls from Guiliani's standard set of widgets, it is advisable to call the base-class implementation of the overridden method. Failure to do so may break the base classes default behaviour.

2.3.7 Difference between Command and Behaviour

With behaviours you can override existing slots from Guiliani that will be executed immediately. Do not invoke long-running tasks inside a Guiliani-behaviour or your application will not be responsive anymore!

With commands, you can add additionally functions to Guiliani that are executed after the normal process under conditions that will be defined by you. It is also possible to divide long-running tasks up into several sub-tasks processed by the same command to get a more responsive application.

2.3.8 DataPool

The "DataPool Module" is providing on-the-fly data updates of controls. Using the DataPool simplifies the task of linking objects within the GUI to external data sources. The DataPool will then automatically take care of updating observers of an entry within the DataPool whenever it changes its value. DataPool-Connectors can connect to data from single variables, arrays, to databases and other places where data is stored.

The DataPool is also useful when synchronizing data between the application (e.g. sensor-data) and the GUI.

2.3.9 Nine Patch

A Nine Patch is a smart way of scaling up bitmaps without the usual quality and performance losses. The images are scaled by subdividing the source image into nine sections and blitting the border sections un-stretched, while only stretching the centre-section.

This is particularly useful for mostly rectangular images in use-cases such as a scaled button, a progress bar or a handle of a scrollbar.

The Nine Patch is defined by the width/height of its top, bottom, left and right borders. These are given in pixels and relative to the image's top/left, and bottom/right corners. Typically, you will wish to set these values so that they encapsulate the areas of your source-image that shall remain in their original size, such as the edges or drop shadows of a button. The following image illustrates this concept.

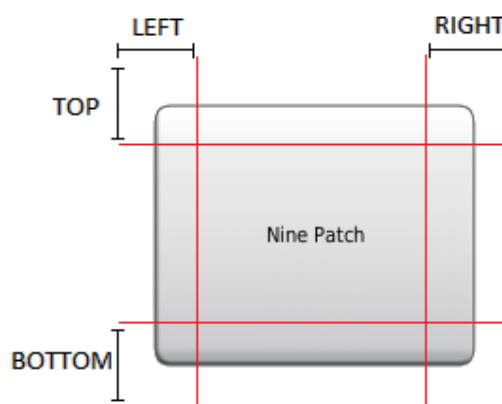


Fig 2- Nine Patch

2.3.10 Export Format

Resources used within your GUI can have several formats in which the original file will be converted on export. This offers you the possibility to optimize your resources for loading-time, memory-consumption, etc.

You can either choose to use the default export-format – which is set in the “export-settings” of the project - or set a specific export-format on a per-resource base.

More information regarding Export settings is provided in section 3.5.7

Note: when resources are added to the project they initially use the “default” export-format.

Note: Guiliani-resources are always exported using the default-format and cannot be altered.

2.3.10.1 Export-formats for images:

native:

this is the current format of the image (e.g. PNG) and during export the image is not converted into any other format

- pros: the file remains unchanged during export
- cons: if the file is a compressed format it consumes RAM during runtime to display it

RAW:

this is a special format which exports the uncompressed content of the image.

- pros: less RAM-consumption because the actual content of the image remains in ROM and is blitted from there
- cons: uses more ROM compared to the compressed format of the original image

RAW565

this is a special format which exports the uncompressed content of the image without alpha channel and reduced color channels.

- pros: even less RAM consumption than RAW because only 16bits are used and still actual content of the image remains in ROM and is blitted from there
- cons: no transparency can be used, only 65k colors.

RLE:

this is a special format which combines less RAM-consumption with fast blitting.

- pros: can be blitted directly from ROM and uses only little RAM. due to the RLE-compression the blitting is faster than the RAW-format
- cons: due to RLE-format images can only be stretched or shrunk to an integer multiple (e.g. 1/2x or 2x) also rotation is not possible

BLU:

this format is optimized to be used with BLU blitting unit on very small devices.

2.3.10.2 Export-formats for fonts:

native:

this is the current format of the font (e.g. TTF). On export it will not be converted

GlyphLib:

This format is recommended when using the GlyphLib-font engine

GLYPHLIB_A4

this format uses 4bit per pixel as alpha channel for every exported glyph.

- pros: Uses only the half ROM space compared to 8bit format.
- cons: Might cause aliasing at glyph edges when using big font sizes. Depending on font quality there can be viewable artefacts.

GLYPHLIB_A8

this format uses 8bit per pixel as alpha channel for every exported glyph.

- pros: No aliasing.
- cons: Uses double ROM space compared to 4bit format.:

2.3.11 Tool Tips

The GSE provides tooltips which give you more detailed information on specific actions. Tooltips are displayed when the mouse-cursor rests for a short time over an object or icon (e.g. the “Bring object to foreground” button in the dialog window).

Tool tip can be enabled in settings window by checking the “Show Tooltips” checkbox. (see Fig 3)

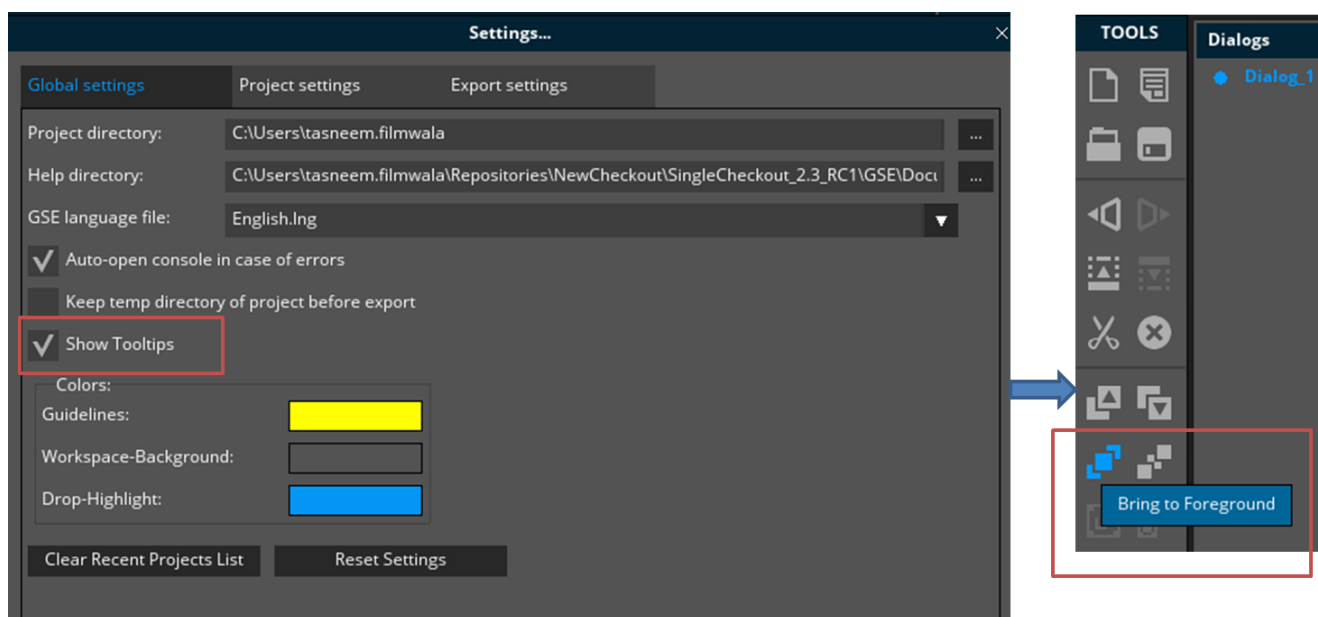


Fig 3 - Tool Tip

2.4 StreamRuntime Application

2.4.1 Overview

The StreamRuntime is the application that runs the GUI created in the GSE. This serves the GSE as a simulator for executing and testing the current project and on the target system as the basis for the target application.

2.4.2 Directory structure

Directory structure and file description of StreamRuntime application.

Directory/File	Description
CMake	CMake configuration
Common	Common files over different Guiliani applications
Include	Project specific include files
Resources	Icons for iOS and Windows
Source	Project specific source files
CMakeLists.txt	CMake project file

Directory structure of StreamRuntime

File	Description
Platform/*/StreamRuntime*.c cpp h]	Program entry points (main function) for different platforms
Common/[Include Source]/Platform/<PLATFORM>/StreamRuntimeStartup.c cpp h]	Target specific initialization of wrappers and configurations
Common/[Include Source]/Platform/win/pc/StreamRuntime.cpp	Windows specific initialization of wrappers and configurations
Common/[Include Source]/StreamRuntimeConfig.c cpp h]	Loads project configuration.
Common/[Include Source]/StreamRuntimeGUI.c cpp h]	Loads GUI based on exported files from GSE project

Files in Common directory of StreamRuntime

File	Description
CustomExtension	In this folder all custom extensions generated by GSE will be placed
GUIConfig/User*Resource.h	Header files containing the Resource IDs exported from GSE project
MyGUI_SR.c cpp h]	Main class of the application for adding specific content

Files in include and source directory of StreamRuntime

File	Description
GUIConfig.cpp	This file contains constants which hold the count of global properties, image resources, font resources, text resources, etc

File in GSE/Share directory

2.4.3 Command-Line Options

The StreamRuntime application also offers the possibility to pass different options via command line. In the following list you can see the options and their function.

Option	Description
--help	Print options overview.
--config <filename>	Specify configuration file to use.
--screenwidth <width>	Specify width of screen.
--screenheight <height>	Specify height of screen.
--keyboard <keyboard-device>	Specify keyboard device to use (e.g. "/dev/input/keyboard").
--touch <touch-device>	Specify touch-device to use (e.g. "/dev/input/touchscreen").
--remote <remote-device>	Specify remote-device to use (e.g. "/dev/input/mouse").
--touchsizeX <size>	Specify horizontal size of touch-device (negative value inverts axis).
--touchsizeY <size>	Specify vertical size of touch-device (negative value inverts axis).
--touchoffsetX <offset>	Specify horizontal offset of touch.
--touchoffsetY <offset>	Specify vertical offset of touch.
--resourcefile <resource-file>	Specify resource-file to use.
--resourcepath <resource-path>	Specify path for resources.
--license <license>	Specify license-key as a string.
--licensefile <license-file>	Specify license-file containing key.
--enablescreenshots	Enable taking screenshots with CTRL+SHIFT+S

2.4.4 CMake Configuration

The GSE builds its StreamRuntime automatically during compilation. The StreamRuntime that is being built (e.g. GuilianiDemo) can be selected in the CMake configuration of the GSE.

The StreamRuntime must be built separately for the target systems. In the following the important parameters of the CMake configuration are listed.

Variable	Description
PATH_EGAC	Path to the "eGaC" directory of the SDK.

PATH_GUILIANI	Path to the "Guiliani" directory of the SDK.
PATH_LIBS	Path to the "External" directory which contains the external libraries of the SDK.

Necessary path variable for CMake configuration

The other variables can be customized and vary depending on the platform for which the application is built. Amongst other things, the graphic and font wrapper used can be selected here.

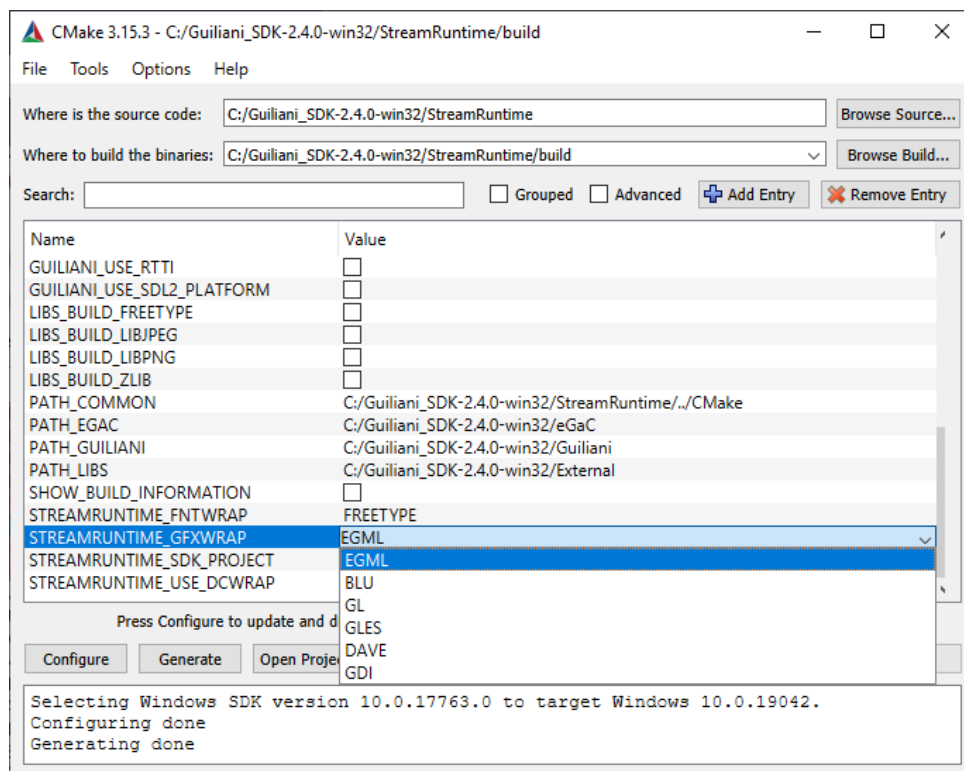


Fig 4- StreamRuntime CMake Configuration select GFX-Wrapper

2.4.5 How to set up resource-file or resource-header

Two other ways besides the exported GSE project (e.g. temp folder) is using a resource-file or resource-header.

The use of the resource-file can be done either by command-line option as well as be activated in the code.

```
GETRESHANDLER.SetFileOpenOrder(CGUIResourceFileHandler::RESOURCE_FILE_FIRST);
GETRESHANDLER.SetResourceFile("Resources.dat");
```

Codeline in "StreamRuntime.cpp" to enable and set priority of resource-file

The use of the resource-header can be done only in the code.

```
GETRESHANDLER.SetFileOpenOrder(CGUIResourceFileHandler::RESOURCE_FILE_FIRST);  
GETRESHANDLER.SetResourceData(GUIResourceData);
```

Codeline in "StreamRuntime.cpp" to enable and set priority of resource-header

!!! If you use the resource header: Don't forget to copy header file and re-compile. !!!

For both, a combination with other resources is possible and the order can be prioritized.

2.4.6 How to take screenshots

When running the StreamRuntime you can take screenshots using a specific keyboard shortcut. This can be activated via the command-line (`--enableScreenshots`) which uses the shortcut <CTRL>+<SHIFT>+S or by inserting a line of code.

To activate this feature by code the following line can be copied into the file "MyGUI_SR.cpp".

```
EnableScreenshot("Screenshot", GK_S, CGUIEvent::GKM_CONTROL);
```

Codeline to enable in File "MyGUI_SR.cpp" to enable Screenshot feature

The first parameter is used to set the prefix for the screenshot filename. The screenshots are automatically numbered (e.g. "Screenshot_0.bmp", "Screenshot_1.bmp" ...) and stored in the simulator folder as Windows BMP files.

Numbering starts at 0 when the StreamRuntime is started and will count up. If there is already an existing file with this name the next unused number will be used.

3 The Menu Bar

The menu bar at the top of the screen is your direct access to most of the editor's features, reaching from workspace management, via editing of resources related to the project, to simulation and export of the GUI. (see Fig 5)



Fig 5 - Menu Bar

A short-cut icon list is provided on the left side of the menu bar to speed-up the access to frequently used features “New Project”, “Save Project”, “Open Project”, “Undo” and “Redo”.

3.1 The File Menu

The “File” menu allows you to [create new projects](#), save existing ones, add new dialogs, handle settings, manage requirements and simulate the current GUI.(see Fig 6)

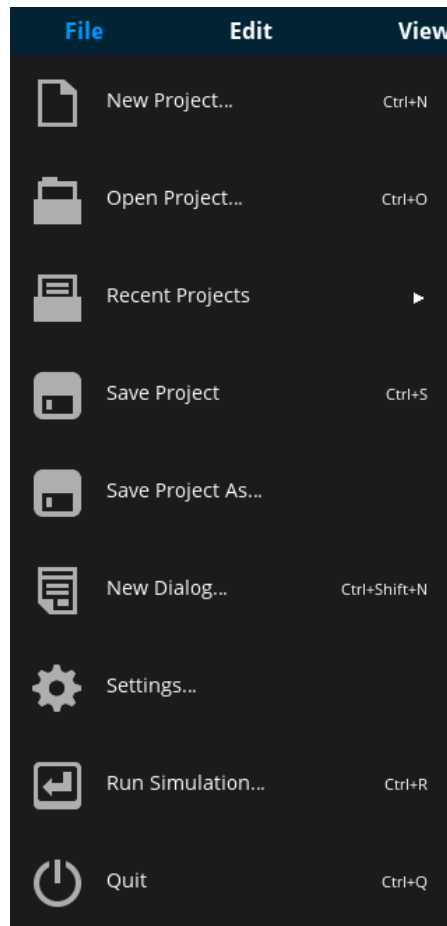


Fig 6- File Menu

3.2 New Project

The “New Project” Menu is creating a new project. Enter a name for your new project and specify a folder on your disk where all project-relevant data will be stored.

The Editor will create a subfolder with the name of the project in the specified directory (see Fig 7)

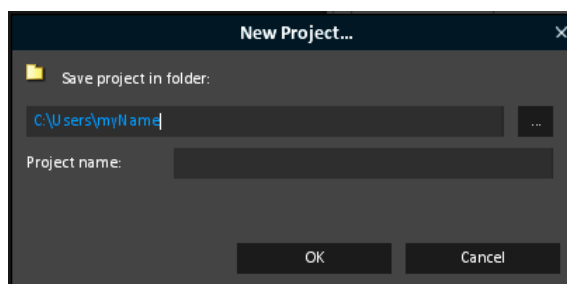


Fig 7 - New Project

3.2.1 Open Project

Use this option to open an already existing project. Select a Guiliani project file (“.gpr”) from disk. The corresponding project, including dialogs, resources, languages etc. will be loaded (see Fig 8)

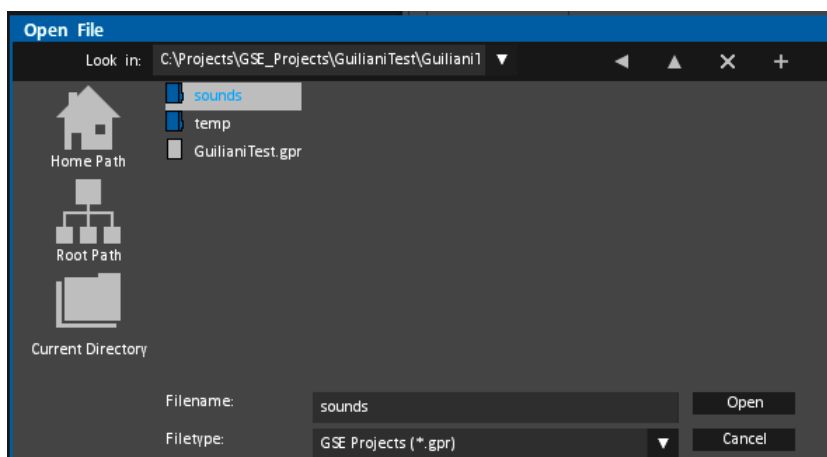


Fig 8- Open Project

3.2.2 Recent Projects

Use this fold out menu to open a previously used project. Up to 10 projects will be listed here. (see Fig 9)

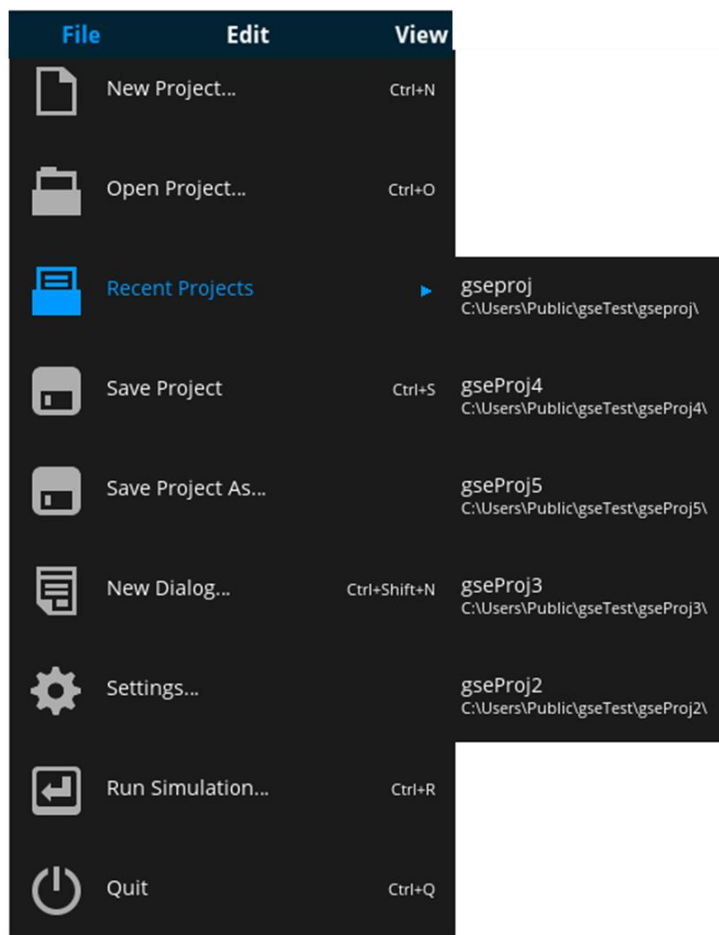


Fig 9 - Recent Projects

3.2.3 Save Project

Use this option to save the current project to disk. The streaming editor will store all relevant information of the project, such as:

- DataPoolProperties.xml
- FontProperties.xml
- GeneralResourceProperties.xml
- ImageProperties.xml
- ObjectHandles.xml
- Properties.xml
- Requirements.xml
- SoundProperties.xml
- TextProperties.xml
- [ProjectName].gpr
- [ProjectName]_dialogs.xml

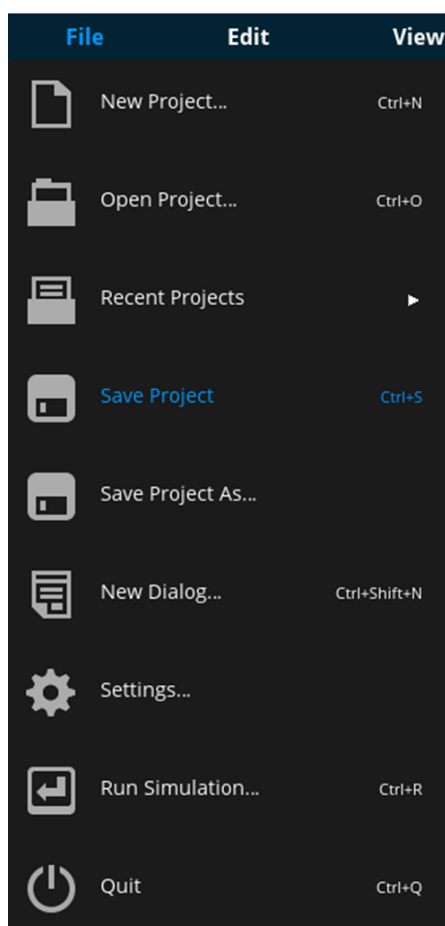


Fig 10- Save Project

Additionally there are xml files for each of your dialogs and the corresponding folders, which contain all resources used by the project: such as *fonts*, *images*, *sounds*, *general resources* and *temp* (temporary files created when you simulate your project)

3.2.4 Save Project As

This option is used to save the current project in a different directory and with a different name. After clicking on “Save Project As...” a window appears in which you can enter the new location and project name (see Fig 11).

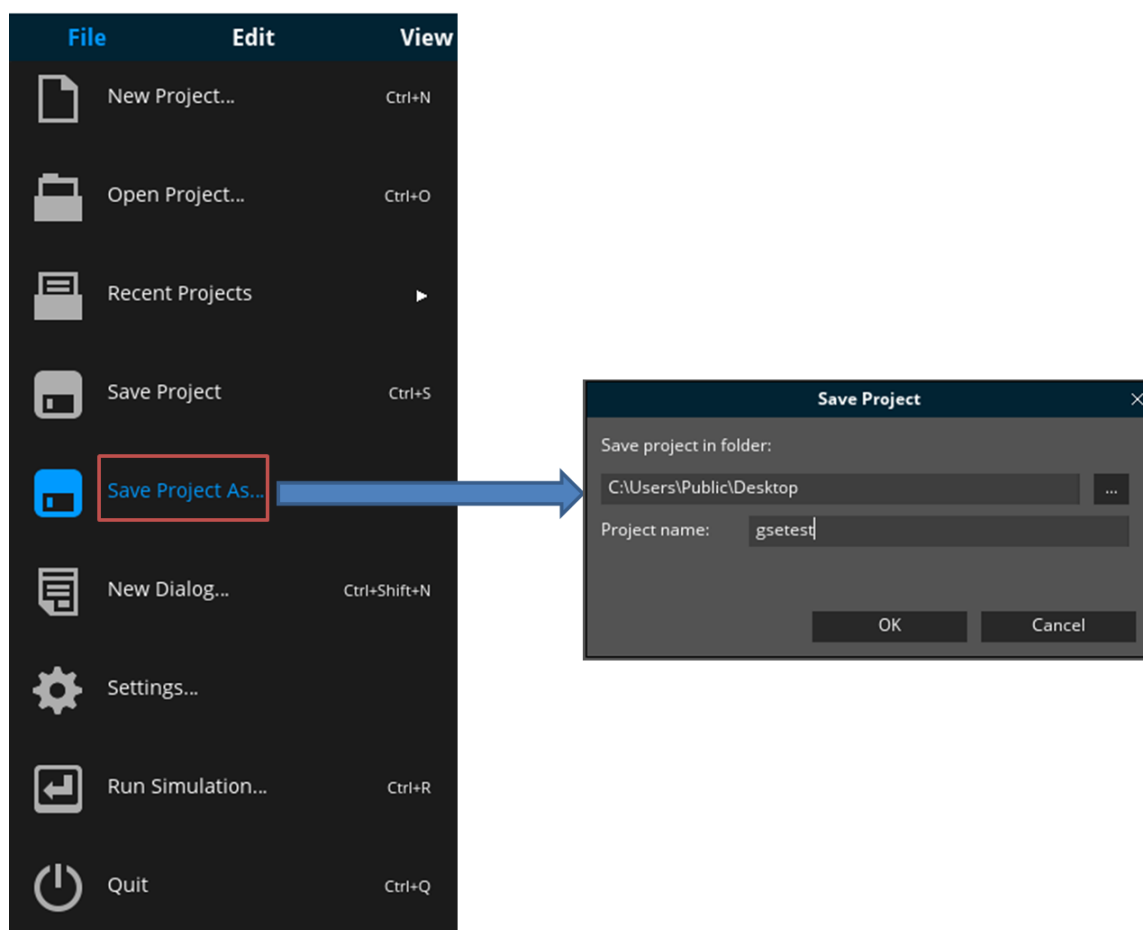


Fig 11- Save Project as

3.2.5 New Dialog

Use this option to add a new dialog to the currently opened project. You will be asked to supply a name for this dialog (this will also be used as the name of the Guiliani streaming file in which this dialog gets stored).

You can also define the size of this dialog in pixels. This for example allows you to create non-full screen dialogs, which are used as Popup-windows (see Fig 12).

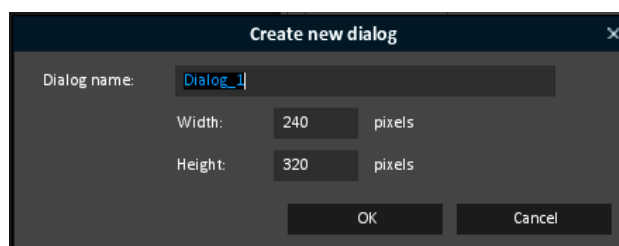


Fig 12 - New Dialog

3.2.6 Settings

The general GSE settings, project-specific settings and export-configurations can be set using settings option under file menu.

3.2.6.1 Global Settings

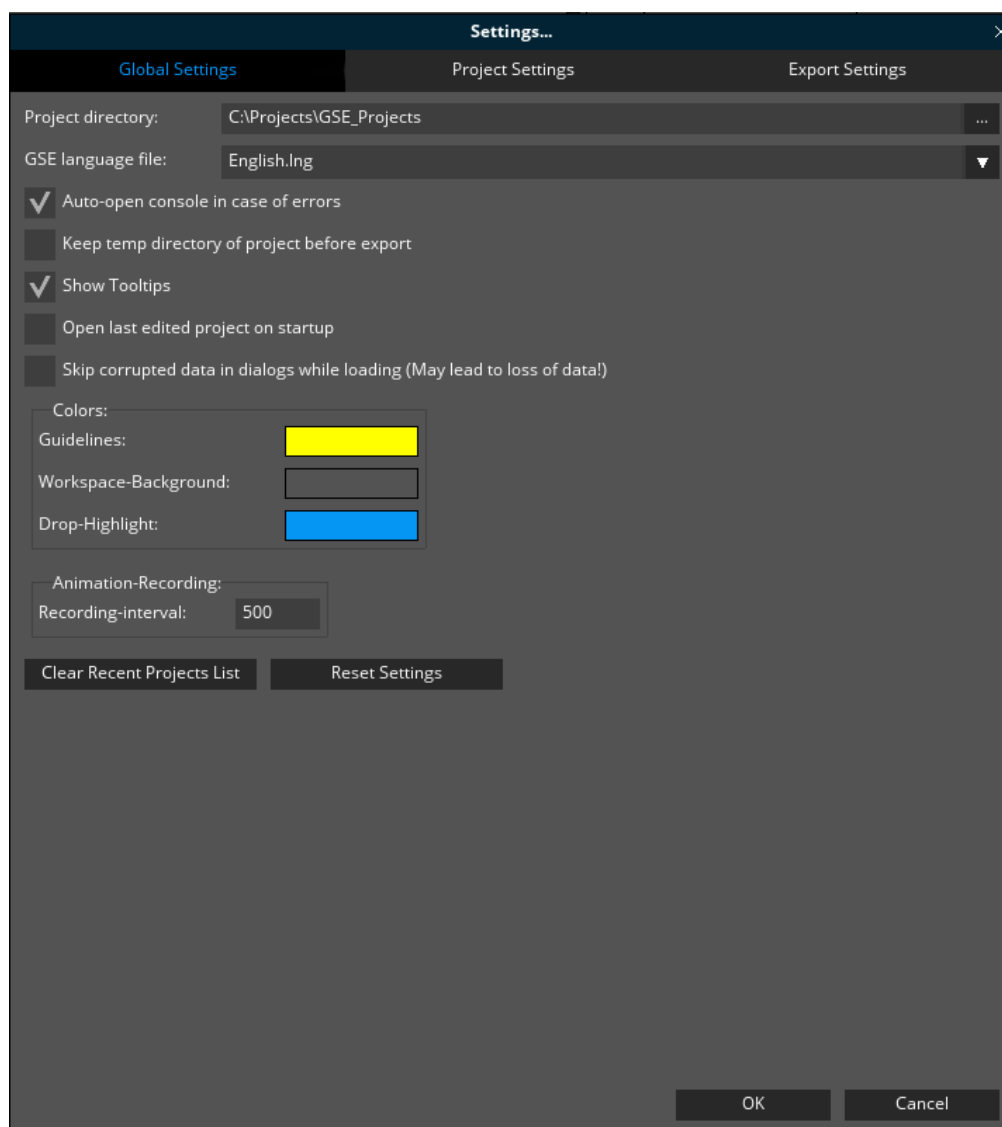


Fig 13 - Global Settings

- **Project directory:** here you can define the directories where your projects will be created by default (this is the initial location shown in the “new project”-dialog)
- **GSE language file:** Select the language of the GSE (English or German).
- **Auto-open console in case of errors:** if this is selected the error-console will be shown – even if not currently open – if an error is reported

- **Keep temp directory of project before export:** when using “run simulation” a temporary folder is created storing all necessary files. This folder is deleted by default. If this setting is selected, it will not be deleted before export
- **Show Tooltips:** if this setting is selected, tooltips containing more detailed information on various actions are displayed
- **Open last edited project on startup:** if this setting is selected the last edited project, if any, is automatically opened on editor-startup
- **Skip corrupted data in dialogs:** if this setting is active, the GSE will try to skip corrupted content in dialogs when loading a project. After the project was loaded, a summary displaying all affected dialogs is shown. **Warning: this might lead to loss of data when saving!**
- **Colors:**
Define colors for
 - **Guidelines:** Guidelines help in guiding controls when aligning near other controls
 - **Workspace-Background:** The background of the GSE workspace is set to the color specified under this setting
 - **Drop-Highlight:** When the control is dragged inside the project dialog, the Workspace background is changed to the color defined under “Drop-Highlight” setting and is changed back to background color after dropping it.
- **Animation-Recordings:** in this section you can define the recording-interval
 - **Recording-interval:** this indicates a time-interval in milliseconds in which an animation will be recorded.
- **Clear Recent Project List:** This button clears the recent projects from “Recent Projects” present under file menu.
- **Reset Settings:** This button resets the current setting to default settings.

3.2.6.2 Project Settings

These settings only affect the currently opened project.

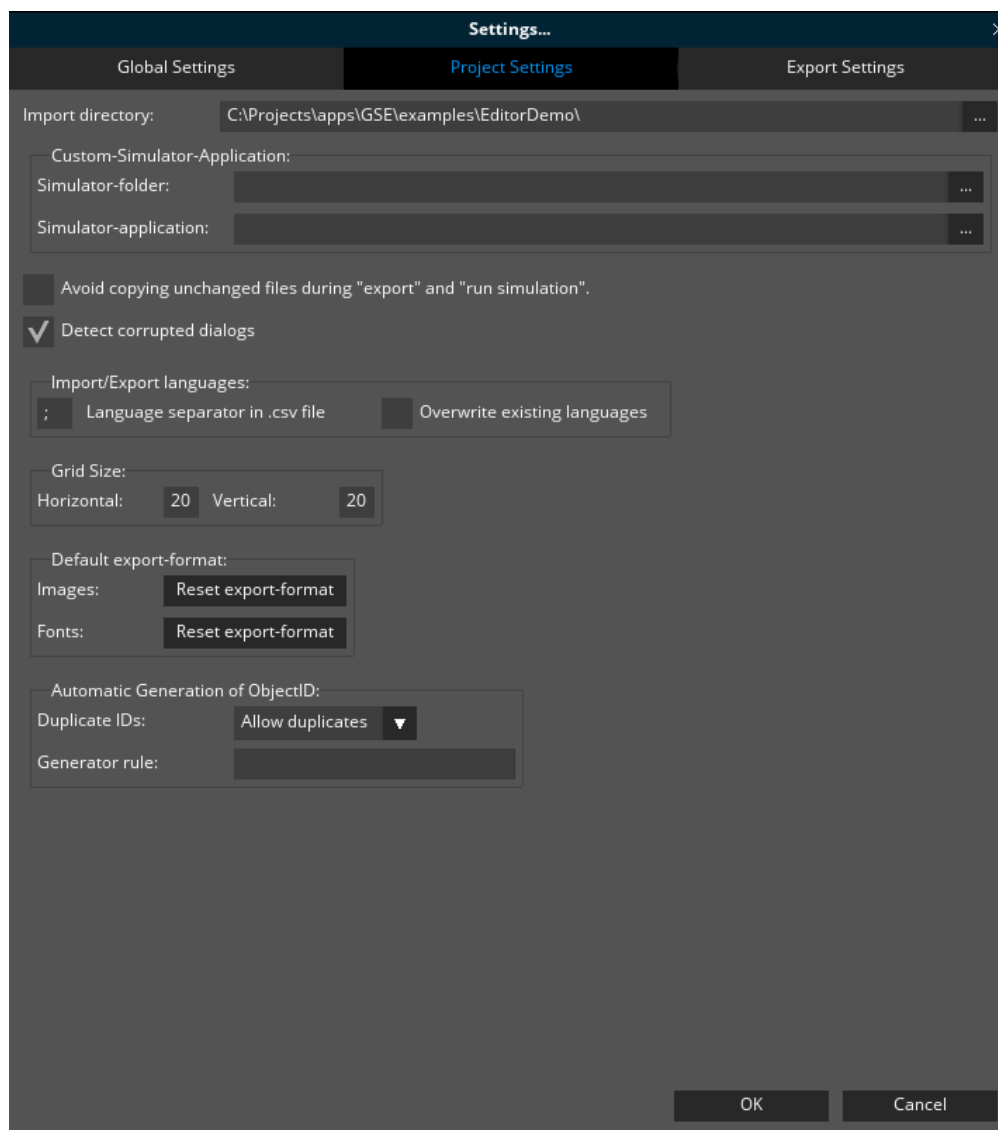


Fig 14 - Project Settings

- **Import Directory:** The “Import Directory” specifies the folder from where the editor should import new dialogs into the project.
- **Custom Simulator Application**
When the project should be launched with a custom simulator-application, it can be specified here:
 - **Simulator source-folder:** When header-files are overwritten during export (in case overwriting is active), they are placed inside the directory provided under

“Simulator-source folder”. Also generated “CustomExtensions” will be placed into a sub-folder “Source/CustomExtension” or “Include/CustomExtension” inside this folder.

- **Simulator-application:** The executable which is started for simulating the project after export is done can be specified here.
- **Avoid copying unchanged files...:** If this setting is active, files which do not have changed since the last export will not be copied. This can be used if the contents of the export-folder are under version control or the timestamp is relevant.
- **Detect corrupted dialogs:** This setting should always be active to avoid opening corrupted projects without noticing.
- **Import/Export Languages:** For the [import](#) and [export](#) of language files, the language separator of the .csv file can be adjusted (typically “;”) by specifying the separator under option “Language separator in .csv file”. The Overwrite option enables the overwriting of existing language files and is enabled using checkbox “Overwrite existing languages”. If overwriting is enabled then existing languages will be overwritten by the imported languages with the same name.
- **Grid Size:** The vertical and horizontal spacing for the grid is provided under “Horizontal” and “Vertical” field. When the grid is toggled using “Toggle Grid” option from view menu then the provided horizontal and vertical grid dimensions are applied for the project in the workspace.
- **Default export-formats:** In case the images and fonts are set to a different format other than “default” then clicking on the “Reset export-format” button resets the images and fonts back to the default export format.
- **Automatic Generation of Object ID:** The generation of object ID’s is done based upon the rule defined under “Generator rule”. The valid rule is “<Prefix>_%CLASS%_%INDEX%”. For example, if the generator rule is defined as “AID_%CLASS%_%INDEX% “, and an image is placed inside the dialog, then the generated object ID is AID_IMAGE_1. The drop down under “Duplicate ID’s specify actions for the object ID name when an object is copied and pasted inside the dialog. If “Allow Duplicate ID’s is selected from the drop down then copying a control and pasting it inside the dialog will name the new control with the same name. If “Rename” is selected from the drop down, then instead of duplicating the object ID, the ID’s are renamed. If “Set NO_HANDLE” is selected then the copied object gets the object ID as “NO_HANDLE” in the attribute window and can be changed to some other ID.

3.2.6.3 Export Settings

Export settings can be used if the same project needs to be deployed to various environments with different resolutions and resources.

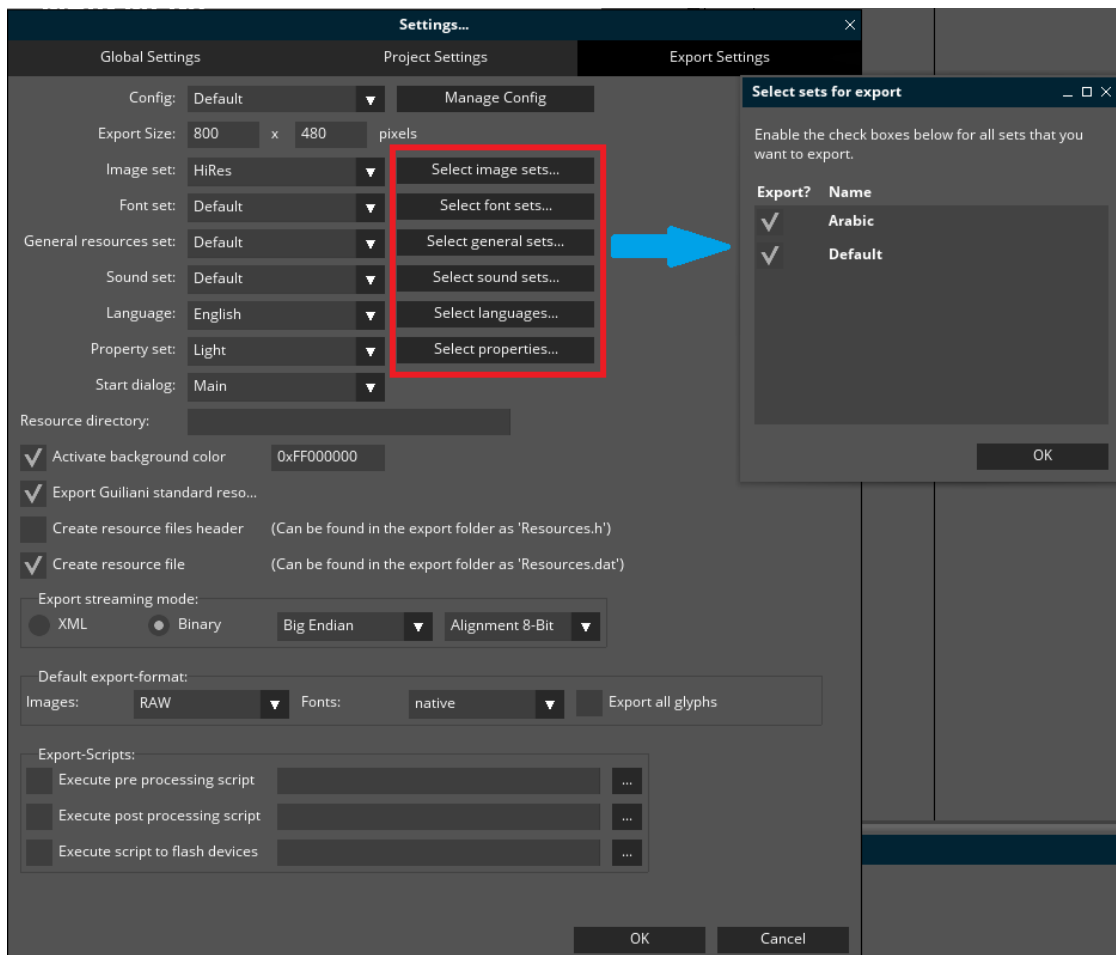


Fig 15 - Export and Select Sets

- **Config:** This dropdown-list allows selecting between different configurations for the project. New export-configurations can be created using “Manage Config”. The following settings are combined into the selected export-configuration and are switched when selecting another export-configuration from the dropdown-list.
- **Export Size:** This specifies the size the StreamRuntime will start
- **Sets:** The various set names define which resources will be loaded as defaults when starting the application. The settings include set for “Image set”, “Font set”, “General resources set”, “Sound set”, “Language” and “Property set”. By default “Default” set is selected. There is also a possibility to select sets other than Default by clicking the “Select button” for the above mentioned sets. Figure 14 shows the window for selecting sets.

- **Start Dialog:** The dialog which should appear when the application gets started is specified under “Start Dialog” field.
- **Activate Background Color:** You may wish to activate a default background-color for your application, in case you do not have a default full-screen background image in each of your screens. The background-color of the application can be activated and set using this setting.
- **Resource directory:** The “Resource directory” is an optional path, which will be used as a prefix when the StreamRuntime searches for resources on the target system. This is useful if the resources will not reside in their standard location right next to the executable.
- **Export Guiliani standard resources:** If you do not use any of Guiliani’s standard resources, because you have your own designs, it is recommended to uncheck the “Export Guiliani Standard resources” checkbox, so that they will not be exported and take up more memory.
- **Create resource files header:** The “Create resource header / Create resource file” checkboxes offer you the possibility to automatically create a Resource File from all resources, which you chose to export. This is particularly useful for systems without a file-system, where you will compile all resources right into the StreamRuntime executable.
- **Default export-format:** Resources without special setting are exported in the format specified under “Default export-format”. The export format supported for images and fonts are specified in section 2.3.9.
For Fonts which are exported as GlyphLib you can specify if all glyphs of the font should be exported or only the first 255.
- **Export Streaming Mode:** The “Streaming Mode” radio buttons let you specify whether you wish to export XML or binary streaming files. When using binary-files endianness and alignment are set in the “Settings”-dialog.
- **Export Scripts:** There are input fields to provide scripts to be run if desired. Depending on the OS running the GSE, you will need shell scripts or batch file. To understand the scripting capabilities it is necessary to know that the export can be split into two phases. First, the export copies all needed data. In a second step, it works on the gathered data like packing it. This leads to three starting points for the scripts: Before the export starts, before it begins to pack and after it is finished. There are several possibilities that arise from using one or more scripts. In a pre-processing script you might want to update some repositories to ensure the usage of the newest resources. The intermediate script can be used for some expert cases, like converting graphics to platform-specific formats, before the export packs. Finally, a script can be used to copy the export to a special destination, store a release or flash a target, immediately after exporting. For more convenience, the checkboxes enable/disable script execution.

3.2.7 Run Simulation

Use this option to run a simulation of the current GUI. This will perform a temporary export of your GUI and immediately execute it in a stand-alone Guiliani application (called StreamRuntime)

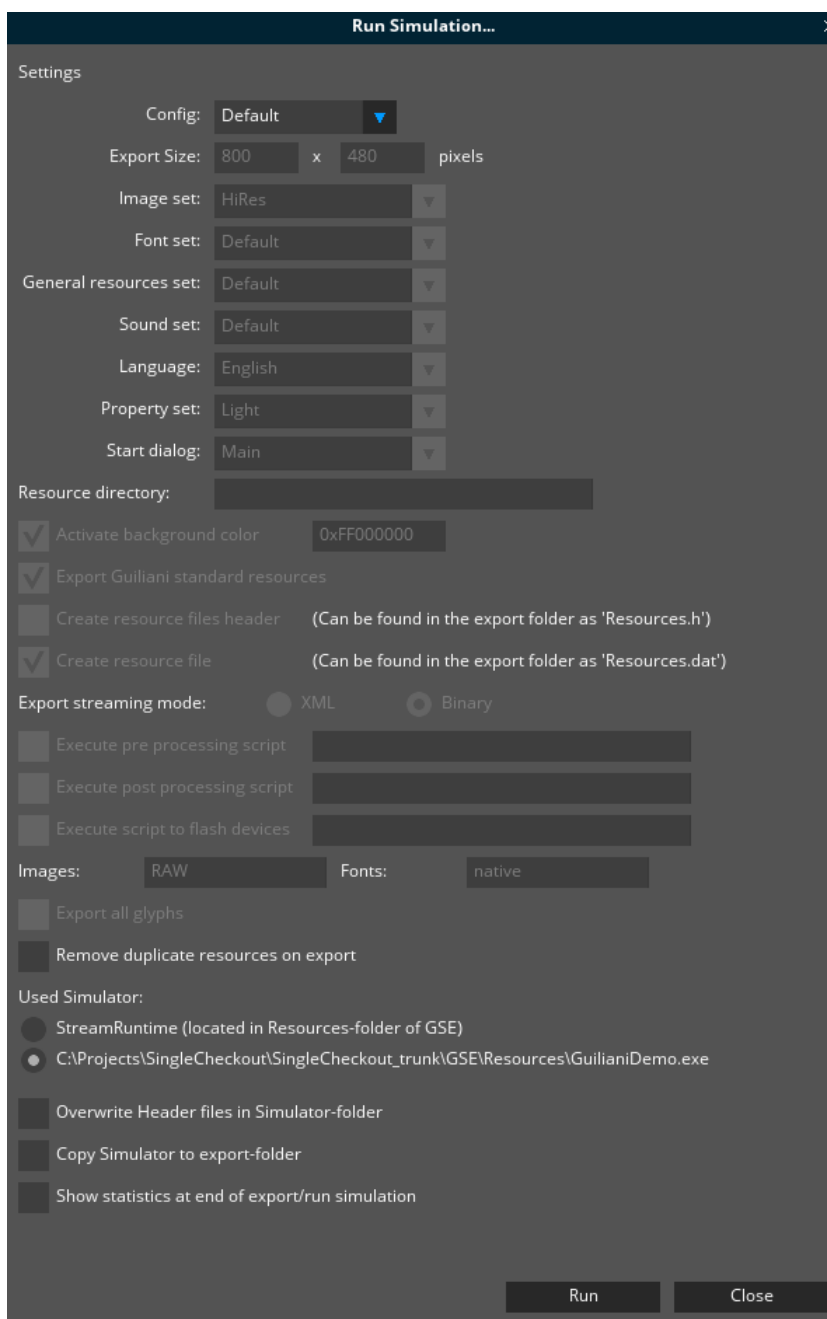


Fig 16 - Run Simulation window

In the run simulation window you can select the export-configuration which should be used to export the resources. Additionally to these settings, implicitly set via the Export-Configuration, you can specify the following:

- **Overwrite Header-files in Simulator-folder:** this will place the files named UserXXXResource.h directly into the subfolder “Include/GUIConfig” of the folder specified in the project-settings for the application
- **Copy Simulator to export-folder:** use this setting to copy the chosen Simulator-application into the export-folder. After that it can easily packed and sent over to a colleague, since it contains all necessary files
- **Used Simulator (StreamRuntime / Custom):** with this setting you can choose which Simulator should be used to start the Simulation after all files have been exported. “StreamRuntime” means the built-in Simulator of the GSE, which was shipped with the SDK and “Custom” starts the Simulator-application specified in the project-settings
- **Show statistics at end of export/run simulation:** after export is done this settings will show a short summary of used memory and compares input and output. This can be helpful to determine if the chosen export-format for images reduces the size of the exported files
- **Remove duplicate resources on export:** if this setting is active, files containing the same content are detected and avoided during export. If a duplicate file is detected, it is removed and its reference updated to the original file

This application runs independently of GSE itself and offers a preview of your application, very similar to what it will look like on the final target platform.

You are free to choose the resolution for the simulation window and to select the resource sets that will initially be used for the simulation.

You may also choose which dialog is loaded initially within the simulation.



Fig 17 - Simulator

By default, the simulation will search for the GUI's resources right next to the runtime executable. If your setup requires the resources to reside in a different location, you may supply a path to this resource directory.

3.2.8 Quit

This will end the GSE. If there are any un-saved changes in your GUI, you will be asked to save your project (see Fig 18).

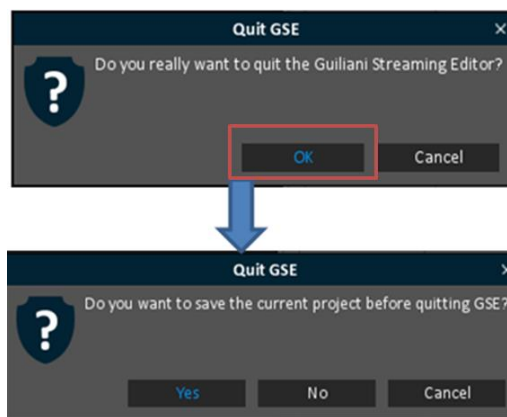


Fig 18 - Quit And Save

3.3 The Edit Menu

The edit menu offers you cut and copy & paste functionality. You may select one or several objects (e.g. by clicking them with the “CTRL” key pressed) and cut or copy & paste them by selecting the corresponding menu option. Note that you can also use this to copy objects between different dialogs (see Fig 19).

Note: if you have more than one instance of the GSE running you can also copy objects between these instances. Be aware that references to resources which cannot be resolved are set to the DUMMY-resources.

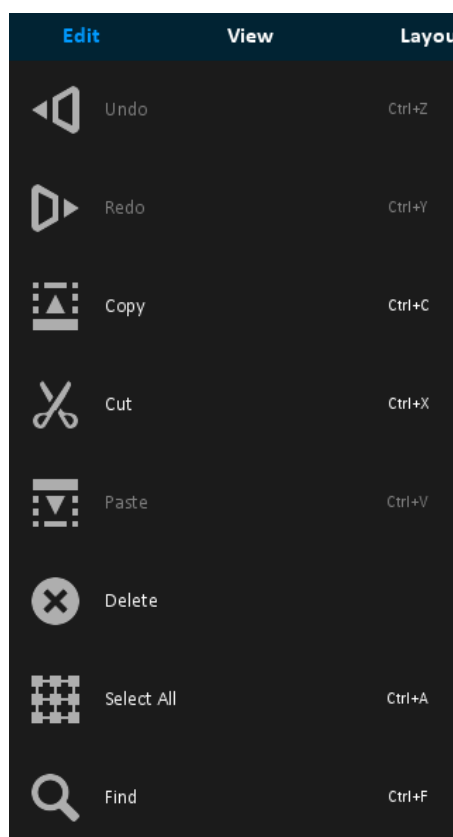


Fig 19 - Edit Menu

3.3.1 Undo / Redo

All actions done in the Dialog-, Hierarchy- and Workspace-Window are recorded and can be later un- or redone. These actions include:

- New Dialog / Rename Dialog / Delete Dialog
- Add / Remove Objects in Dialog
- Re-Order or Re-Arrange of Objects
- Move / Resize of Objects

3.3.2 Copy / Cut / Paste / Delete

These functions only work on the selected objects in the Workspace-window.

3.3.3 Select All

This function will select all objects currently displayed in the Workspace-window.

3.3.4 Find

Selecting find opens the “Search in dialog” dialog, where you can search your project for a variety of criteria. E.g., you can search for specific text strings, for uses of a specific resource-identifier, or for an object with a given Object ID (see Fig 20).

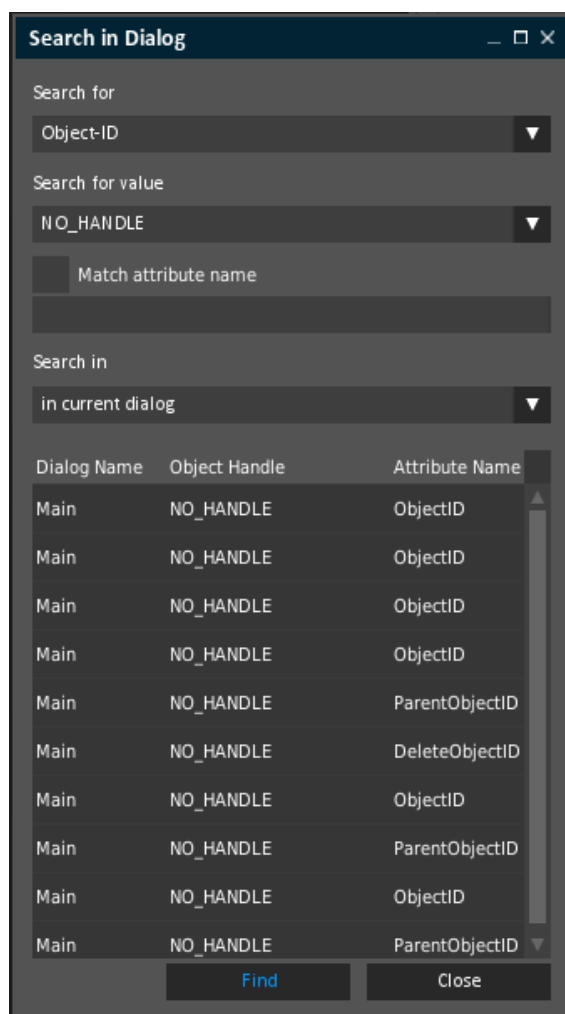


Fig 20- Find

3.4 The View Menu

With “Toggle Grid” you can insert a grid or remove the grid if chosen before. The grid serves as a visual aid and will automatically snap any moved objects inside the preview window to positions that are multiples of the grid size. The grid size can be changed under File → Settings → Project settings.

With “Toggle Guide Lines”, you can enable/disable the yellow guidelines. These yellow lines appear when a control aligns at the other control within the workspace.

With “Snap to guidelines”, the control element halts at the yellow lines, when it could align itself at the other object. This will work only when “Toggle Guide Lines” are enabled.

Toggle Guide Lines and Snap to Guidelines together guides the control to align itself at other control.

With “Toggle Coords/Size”, you can get the coordinates of the control displayed on small white box at the corner. On disabling this option the corner box disappears (see Fig 21).

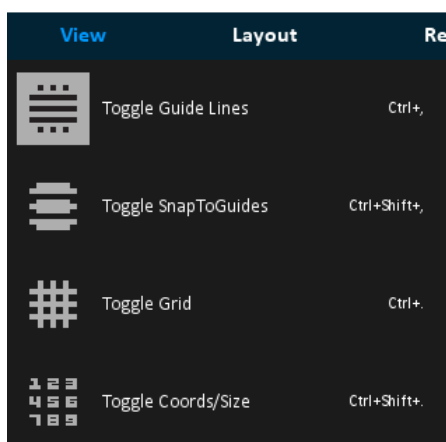


Fig 21 - View Menu

3.5 The Layout Menu

The layout menu supports you in arranging GUI objects within a dialog. The options present under layout menu can be seen in Fig 22 and the description that follows below gives better understanding:

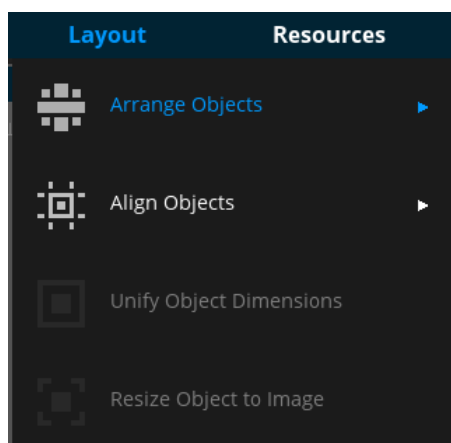


Fig 22- Layout Menu

3.5.1 Arrange Objects

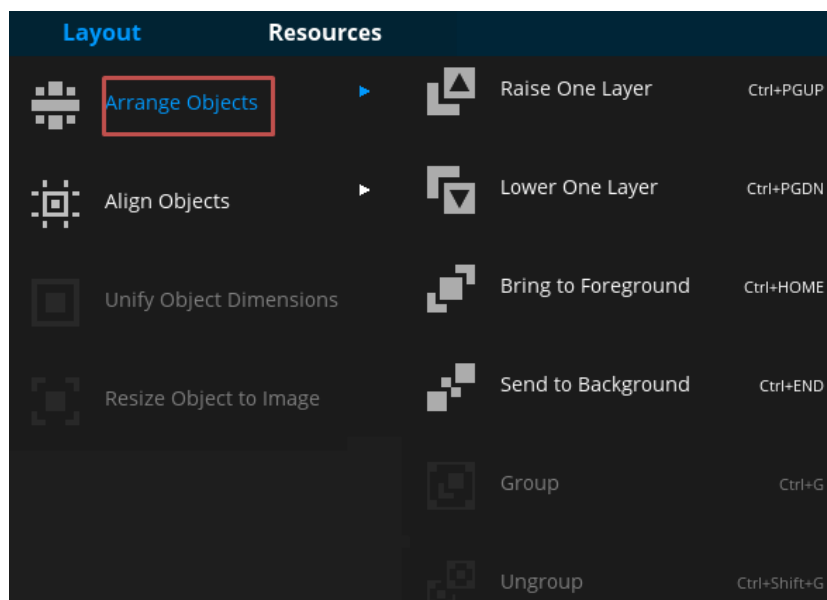


Fig 23 - Layout Menu / Arrange Objects

Arrange objects lets you modify the hierarchy of objects in the GUI-tree, by specifying the order in which they are drawn. You may move one or several objects to the front or background and will immediately see the visual effect in the dialog window and the object hierarchy (see Fig 23).

The sub menus under “Arrange objects” are described as follows:

- “Raise One Layer” puts the control one level up in the object hierarchy.
- “Lower One Layer” puts the control one level down in the object hierarchy.
- “Bring to foreground” brings the control to the topmost in the object hierarchy.
- “Send to Background” brings the control to the lowermost in the object hierarchy.
- “Group”: will pack all selected objects into a new enclosing CompositeObject
- “Ungroup”: will remove the selected object from a CompositeObject and place them on the same level. If the CompositeObject is empty after this action it is removed.

3.5.2 Align Objects

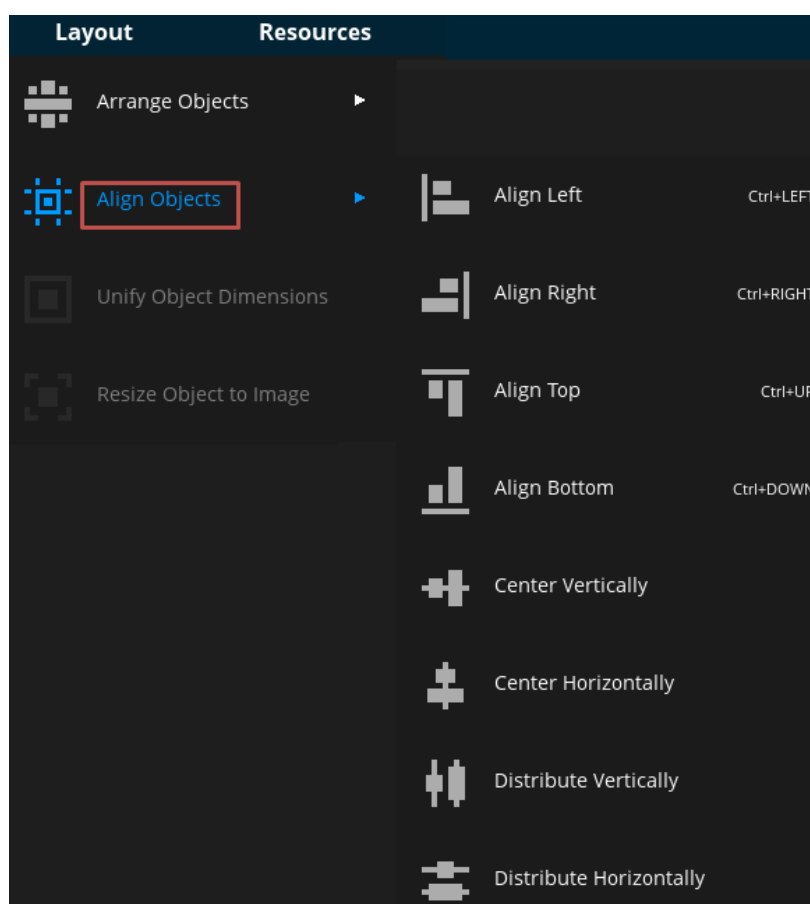


Fig 24 - Layout Menu / Align Objects

“**Align objects**” simplifies the arrangement of several objects in the GUI, by aligning them in one of the pre-defined manners. The first selected object always defines the reference point of the alignment. Fig 24 shows all the possible alignments of the objects.

Another provided functionality is the even distribution of several objects. The objects are arranged between the two outmost objects, either horizontally or vertically.

Unify object dimensions will adapt the dimensions of the selected objects to the dimensions of the first selected one.

Resize Objects to Image: Resize object to dimensions of its first image will resize the object to the original dimensions of the first image in its list of attributes. This is useful to prevent objects from being accidently stretched.

3.6 The Resources Menu

Use the resources menu to manage all resources of your GUI, including images, internationalized text, fonts and sounds. Each of those resource types is handled in a dedicated sub-screen. All resources, properties and requirements are handled under “Manage” sub menu (see Fig 25). In addition, you can export your project or import dialogs too using resource menu. It is also possible to import files from Adobe Photoshop directly into the GSE.

Note: If you want to search for resources in your project which are not referenced by any control, you can do it by using “Find unused resource-Ids” present under “Manage”. The following subsection explains the sub menus under Manage, the import and the export submenu.

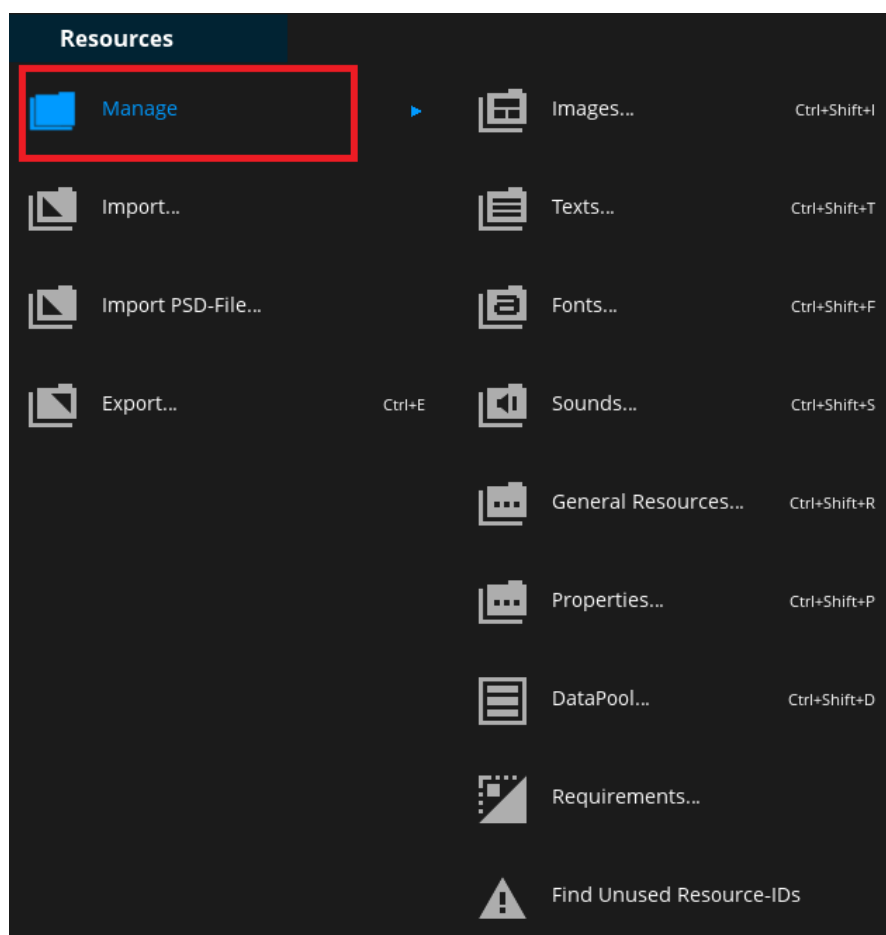


Fig 25 - Resources Menu

3.6.1 Manage

This option handles different resources such as “Images, Texts, Fonts, Sound, General Resources, Properties, DataPool, Requirements and Find Unused Resource IDs.”

3.6.1.1 Images

Guiliani is referencing images by an ID, which serves as a symbolic name for the given image. This has two major advantages:

- You can simply change the currently used image behind an ID, thus greatly simplifying skinning for a GUI.
- Resources can easily be shared among various objects in the GUI and will still only occupy memory once.

Fig 26 shows “Manage Images”-dialog listing all images-resources included in the project.

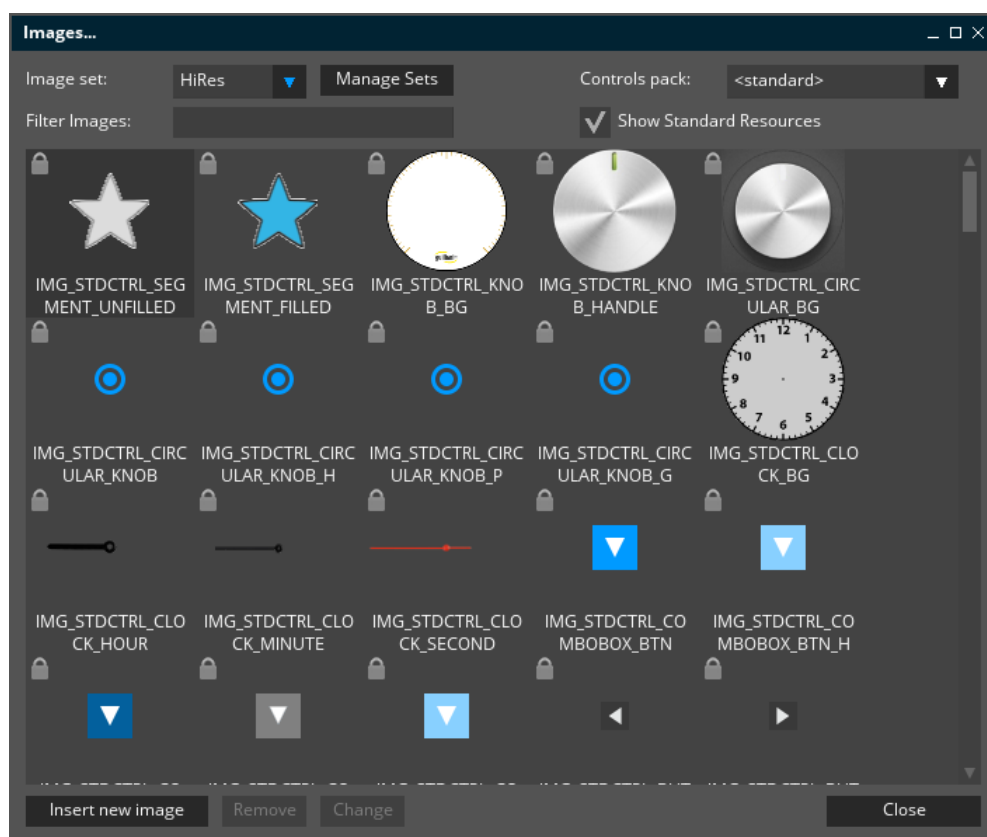


Fig 26 - Manage Images

Various image-formats can be imported into the project (e.g. PNG, JPG, BMP) associated with an ID.

You can do so by clicking the button “insert new image”, browsing your disk for the desired image and assigning an ID to it. Similarly one can remove or change an image using “Remove” or “Change” button present at the bottom

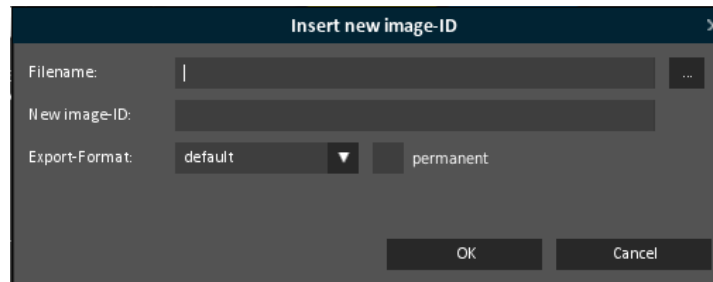


Fig 27 - Insert New Image ID

Note: If you are using image formats that are not supported on the target system (e.g. WebP), you should **not** use “native” as the export-format and you have to choose another one.

Image-Sets:

The drop down menu “Image set” lets you switch between different image sets. You can click on the button “

Manage Sets” to create, delete or rename a set.

All sets share the same IDs, but they may map different images onto them.

Permanent Resources:

In the “Insert new image-ID” dialog the “permanent” checkbox allows you to fine-tune the runtime behaviour of your GUI. Fig 27 shows the window for inserting new images. By default, resources in Guiliani will be loaded when required, and unloaded again once they are not used anymore. This reduces memory-consumption but may result in additional loading times for the respective bitmaps. Also Memory-Fragmentation might occur.

Marking an image as “permanent” prevents it from being automatically unloaded ever again once it is loaded. This will speed up your application (in particular screen switches) but at the cost of increased memory usage.

Export-Formats:

Also the “Export-Format” can be set to “default”, “native”, “RLE” ,”RLE_DAVE” ,“RAW” or “BLU ” to use different memory-consumption-strategies.

Once an image is shown within the “Manage images” screen, it can be assigned to an object within the current GUI.

Note: Guiliani-resources cannot be changed or removed; they all have a lock-symbol next to their IDs.

3.6.1.2 Texts

While it is possible to use string literals within Guiliani, it is desirable to use Text IDs instead. This enables you an easy migration from a single language to a multi-language GUI by simply translating the strings without recompilation of your application. Fig 28 shows the window allowing you to manage different text settings.

Select language text from drop down list of section “Current language” and press Enter. Add Text IDs by pressing the “new ID” button. Fig 29 shows the window for inserting new text ID. To delete, create or rename a text set, click the “Manage Sets” button.as seen at the bottom in Fig 28.

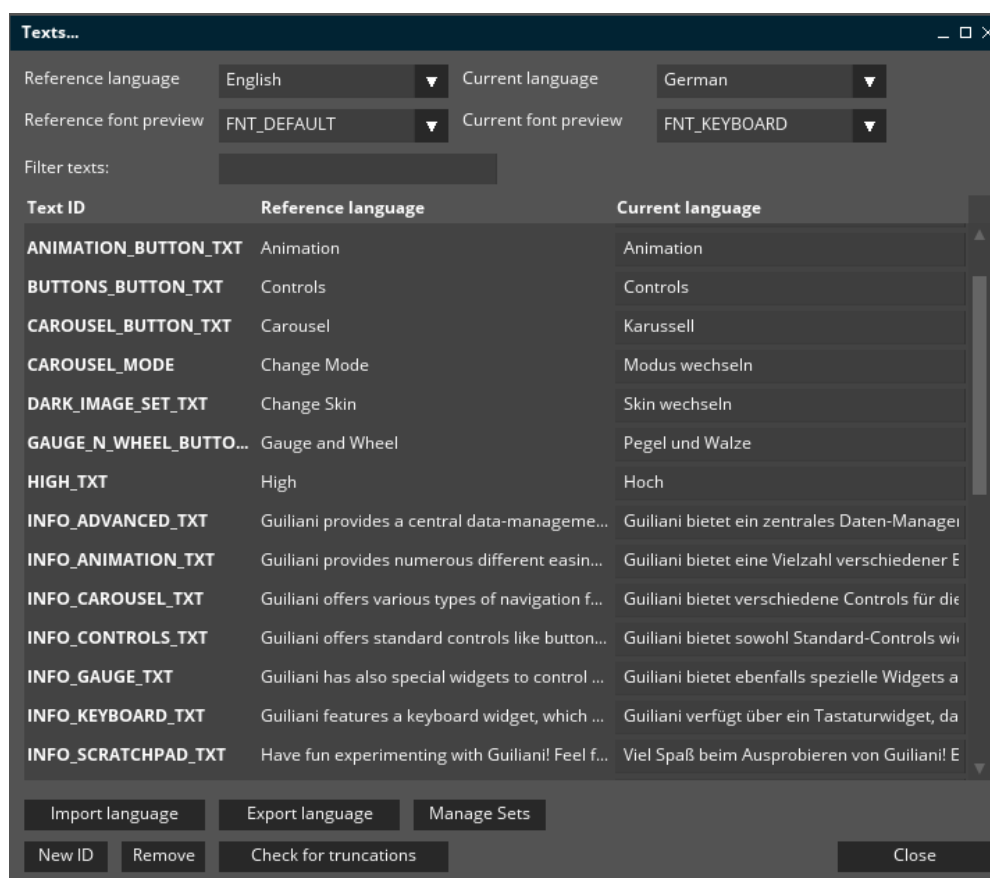


Fig 28 - Manage Texts

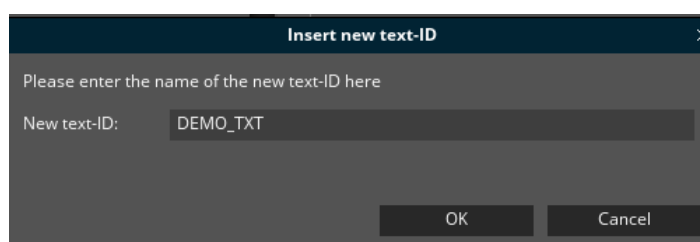


Fig 29 - Insert New Text ID

Text-sets associate Strings in a given language with Text IDs.

For example, the Text-ID “TXT_EQUALIZER” in the Text Set ‘German’ associates to “Klangeinstellung” while in the Text-set ‘English’ it maps to “Equalizer”.

Additionally you can export and import one or all of your language sets by clicking “Import language” and “Export Language” present at the bottom of “Texts..” On clicking these buttons Import Language and Export Language dialog windows are opened respectively (see Fig 30 and Fig 31).

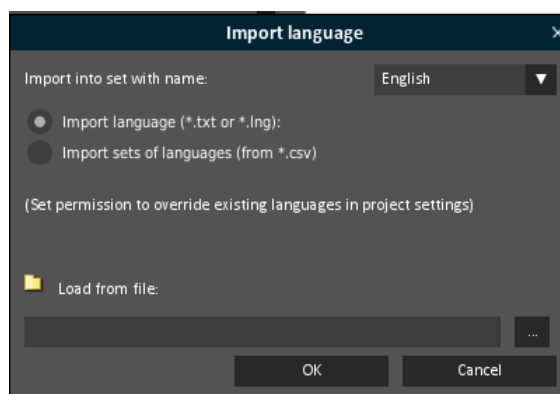


Fig 30 - Import Language

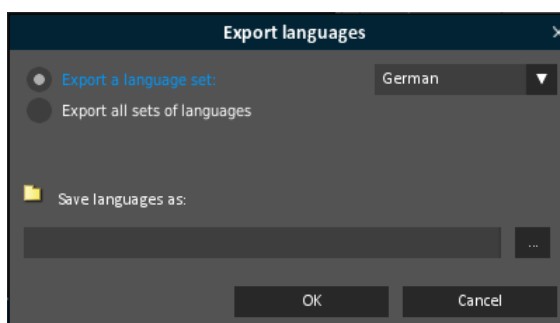


Fig 31 - Export Language

Note: Importing already existing languages only works when “Overwrite existing languages during import” under “File → Settings → Project settings” is set. Otherwise data can only be imported for new language names either by entering a new language name in the “Import into set with name” drop down menu or by importing a .csv-file with a non-existing language name column.

When exporting languages, output files are overwritten without warning. The path where the language file has to be exported is provided in the text field “Save languages as”

With the button “check for truncations”, you can easily check all of your texts if they will fit correctly into the objects. But only texts that are not entered directly in the object can be checked, i.e. those entered using “Manage Texts”. Fig 32 shows the window for checking text truncations.

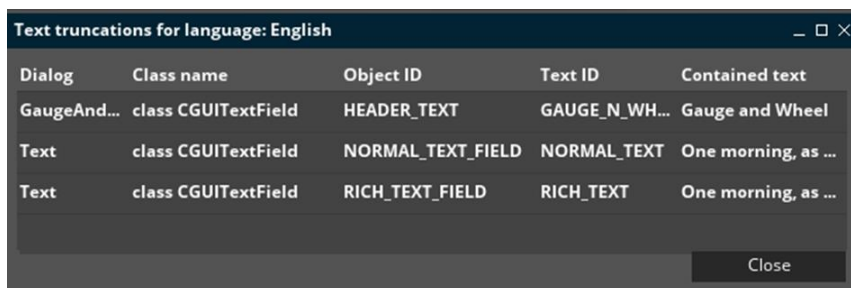


Fig 32 - Check for truncations

Note: This will work for saved dialogs only, so you should either save your project manually or just click onto the “Yes” button if you will be asked for saving your project before searching for truncations.

3.6.1.3 Fonts

Fonts in Guiliani are referenced via “Font IDs”. Each font ID represents one specific font, with a defined font-face and size.

For example, the font ID “FNT_ARIAL_HEADLINE” could have the font size 20 and the font face Arial.

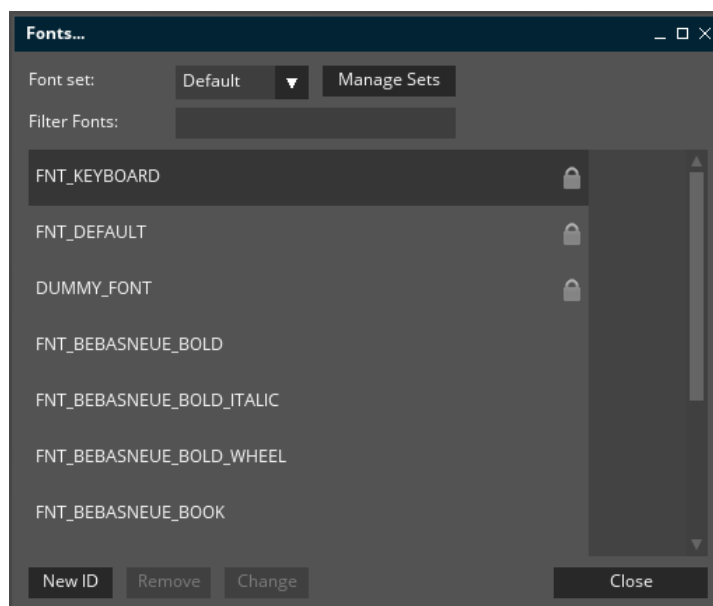


Fig 33 - Manage Fonts

By assigning this font ID to every headline of your GUI, the GUI design can easily be changed by switching the Font set of the project. The active “Font set” is set visible in your GUI.

Note: You can click onto the button “

Manage Sets” to manage different font sets (see Fig 33). A new dialog will be opened where you can create, delete or rename a set. All sets share the same IDs, but they may map different fonts onto them.

Add new Font IDs by pressing the “new ID” button. The “insert new Font ID” window is opened and you can enter or browse for the font file. The Font ID will be set automatically. Enter the font size in pixels (see Fig 34).

If you would like to change the size or source file, press the “change” button.

For deleting a selected font, click onto the “remove” button. (The default Guiliani resource fonts cannot be deleted)

NOTE: The size of a text is set in this font menu. To have a small and a large text you need to create two fonts-resources.



Fig 34 - Insert New Font ID

3.6.1.4 Sounds

Sound IDs are managed in the “Sounds..” window. Here you will find the play button for preview listening, a “new ID” button, a remove and a change button. If you press the “new ID” button, the “insert new ID” window will open, which does have a play button, too.

Here you browse the sound file name; it’s ID and chooses between permanent and not permanent.

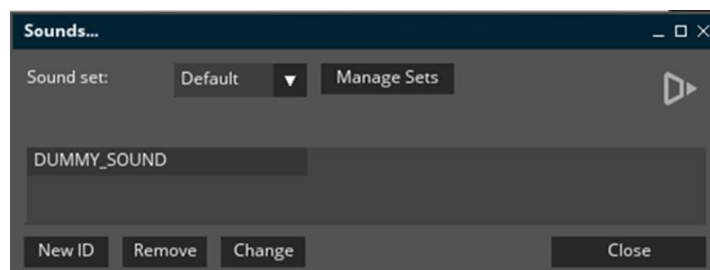


Fig 35 - Manage Sounds

Changing the sound scheme of your GUI is done by changing the “Sound set” in the drop down menu. In the “Manage Sets” dialog, you can create, delete or rename the sets. All sets share the same IDs, but they may map different sounds onto them.

To delete a sound, select it and click onto the “remove” button.

3.6.1.5 General Resources

Within the “General Resources...” dialog you can manage your own independent resource IDs. Like images, fonts and sounds, they are assigned to a resource ID, but do not contain specific content. You need to provide your own controls to use these resources, but you are free to embed 3D models or videos into the GSE workflow just like the fixed type resources. They will be exported and imported just like the fixed type resources.

The advantages of using general resources for tracking your resource files are:

- Keep track of your project's resource files in distributed development environments.
- Automatic exports of other resources to the project streaming target directory.
- Application wide access to 'general resource' files using the Guiliani integrated resource manager, i.e. by a unique resource ID.
- Possible encapsulation of other resources to the Guiliani resource file, which contains all resource files of the workspace project after export.
-

For a new resource ID, or to change or remove an existing one, just click the corresponding button (see Fig 36).

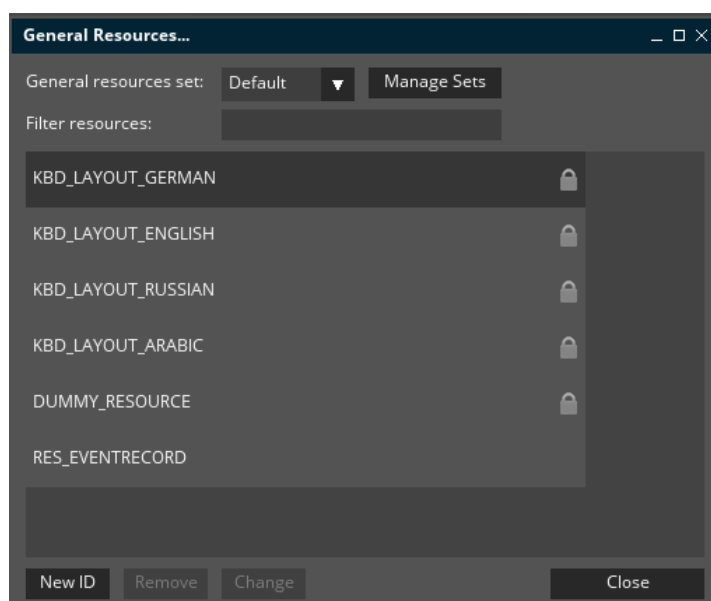


Fig 36 - Manage General Resources

3.6.1.6 Properties

Within the properties dialog you can find different Guiliani properties which can be used for different control attributes. Each property has a type and a value.

The property `GUI_PROP_DEFAULT_BACKGROUND_COLOR` has the type as hex and value as `0xff000000` (black colour) and denotes the colour. Thus one can use this property if the above specified colour is desired for a particular control attribute. In order to use properties for a given attribute, you need to click “P” provided near the desired attribute field and select the desired matching property for that field. Fig 37 shows the property dialog

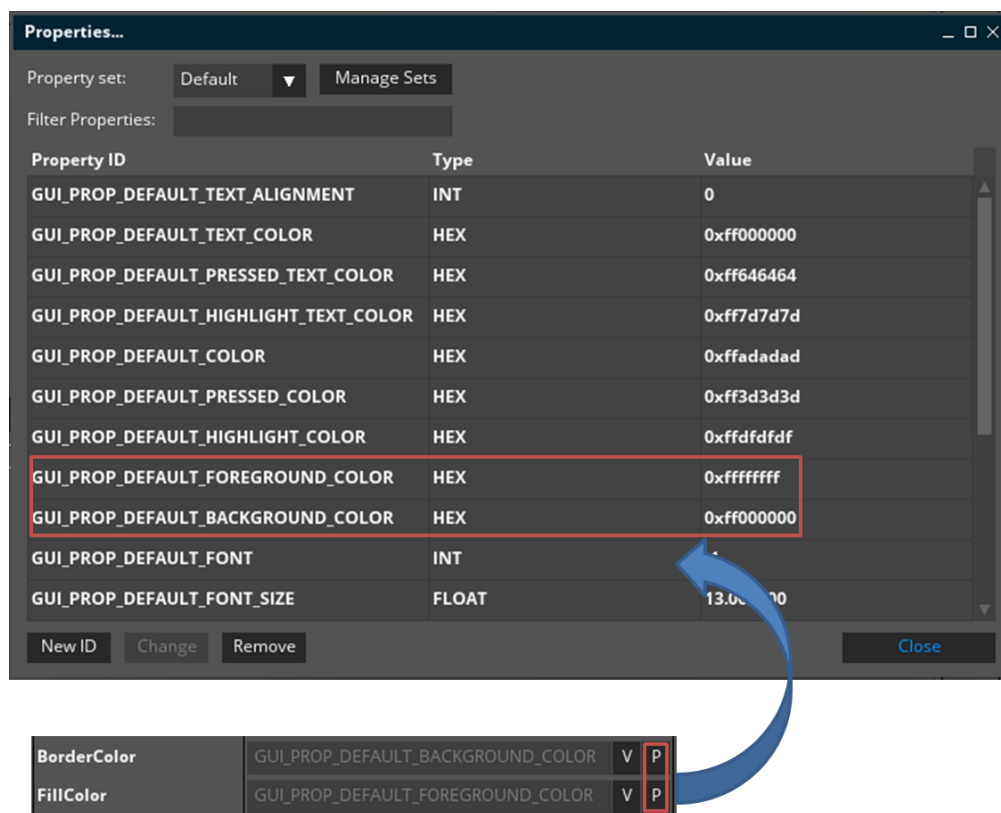


Fig 37 - Properties

Alternatively new Properties can be added by clicking the “New ID” button which is present at the bottom of the Properties dialog. In the “Insert new Property ID dialog”, you need to select the property type from the drop down and based on the type add the appropriate value for the property (see Fig 38).

The supported property types are:

- Bool
- Byte
- UByte
- Short
- UShort

- Int
- UInt
- Float
- String
- Hex

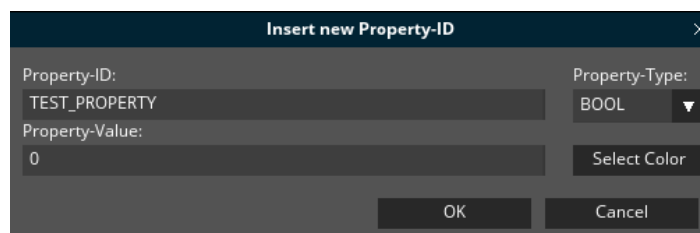


Fig 38 - Insert New Property ID

By clicking “**Change**” button in the properties dialog, value and type of a property can be changed. Also the name of the property can be altered (see Fig 39).

Note: The default Guiliani Properties cannot be modified.

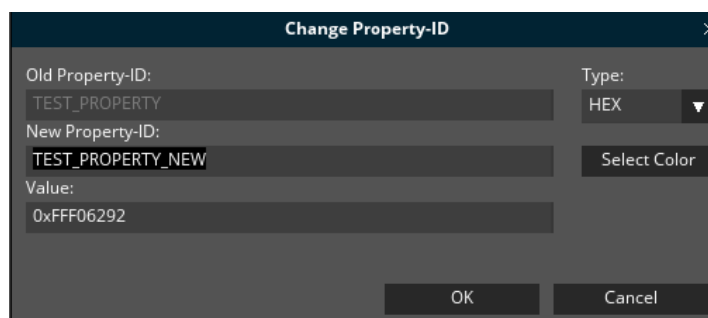


Fig 39 - Change Property

In order to delete a property from the “Properties” dialog, simply select the property and click on “**Remove**” button. The “Remove Property” window appears which asks for the deletion of the property, click OK and the property is removed from the dialog (see Fig 40).

Note: The default Guiliani properties cannot be removed.

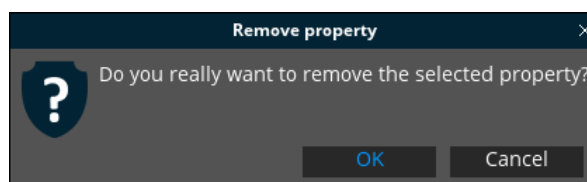


Fig 40 - Remove Property

Note: To change the value of a property which will be used for colors, the button “Select Color” might be helpful. It opens the “Color-Selection”-dialog to easily specify the desired color-value.

3.6.1.7 DataPool

DataPool-objects contained in the project can be configured with this dialog (see Fig 41). On the left side all existing DataPool-objects are listed, while on the right side the detailed configuration is displayed.

To add or remove DataPool-objects the buttons below the list can be used. Newly created DataPool-objects will have a default name “DATAPOOL_IDX” where X will be an ascending number.

For the configuration the following information can be specified:

- Name: the name which is shown in the list and also is used to access this object from the application
- Description: a description of the contents of the DataPool-object
- Dimensions: the number of rows and columns to hold data
- Value: according to the dimension of the DataPool-object values can be specified
- Connected Observers: all objects, which should be notified, when the value changes

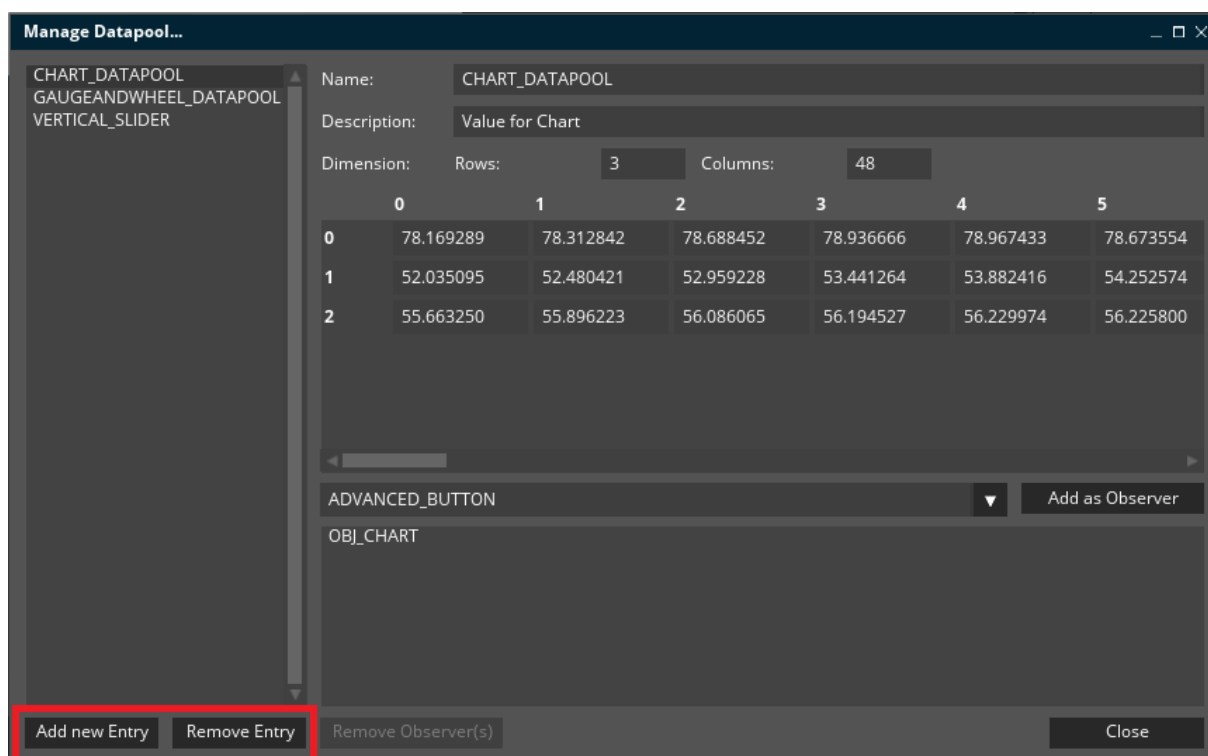


Fig 41 - Manage DataPool

Observers can be added by selecting the desired Object-ID from the combobox and clicking on “Add as Observer”. To remove one or more registered observers, simply select the according entries and click on “Remove Observer(s)”.

Please note that changes on the configuration are immediately saved. There is no undo-function for changes in this dialog. This especially concerns changes of the dimension of a DataPool-object. Also note that decreasing the number of rows/columns will lead to data-loss in the rows/column exceeding this new dimension.

3.6.1.8 Requirements

The Requirements-dialog enables you to track and assign project requirements. Using this dialog, you can specify requirements for the current project and assign them to dialogs and objects or mark dependencies to other requirements. Additionally, you can export the list of requirements into a “.csv” file for use in dedicated requirements tools.

Creation of a new requirement, editing of an existing requirement and deleting the existing requirements is also possible (see Fig 42).

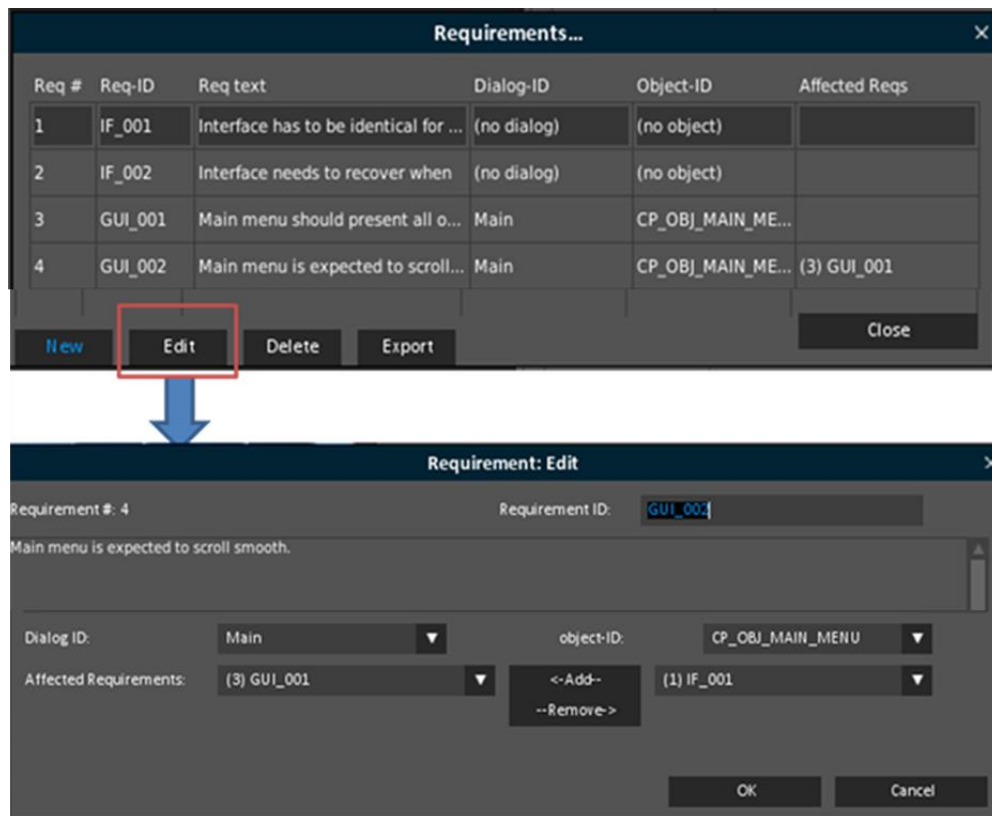


Fig 42 - Requirements

3.6.1.9 Find Unused Resource-IDs

This dialog is a simple but very useful tool to identify resources, which are currently not used throughout the project (see Fig 43).

For objects this will be in most cases auto-generated IDs, which have been changed afterwards. Also the IDs of removed object will be listed.

When it comes to Resources, the dialog displays all resources (e.g. images, fonts, etc.) which are currently not used. So maybe it is safe to remove them from the project to save space.

First select a resource-type to inspect, using the combobox in the first line, afterwards press “Find”.

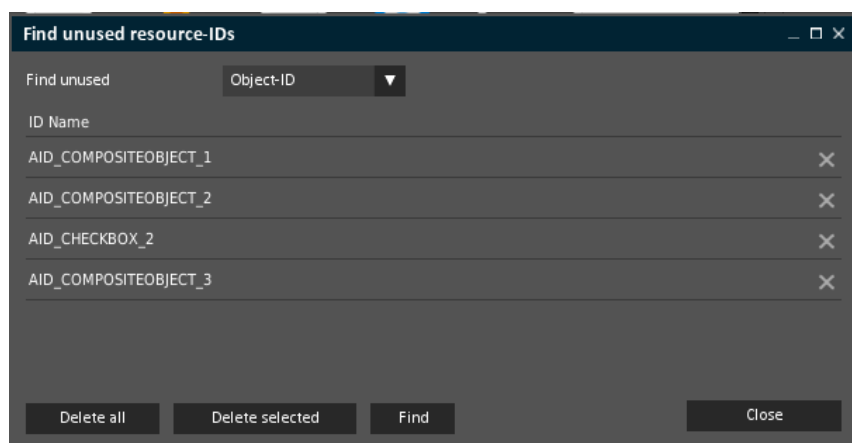


Fig 43 - Find unused resource-IDs

The table displays a list of unused resources according to the set type and they are ready for deletion by clicking the “X” next to each of the entries. To delete all displayed resources click on the button “Delete all”.

NOTE: Depending on the project size, the search procedure can take up to several minutes.

3.6.2 Import

The import wizard is used to import Guiliani dialogs into the current workspace. This allows you to import GUIs that were created from manually implemented Guiliani applications, or that have been generated through conversion from other file formats.

To import one or more dialogs out of another GSE project, pass the dialog .xml files to the “Import” dialog. After clicking the “OK” button the “prepare resources” dialog appears (see Fig 44). If all dialogs to import are valid, the “prepare resources” dialog fills itself. Afterwards the import process starts.

In case of errors during the import, all resources, which can be loaded, will appear. For all other resources the user will be informed about the error.

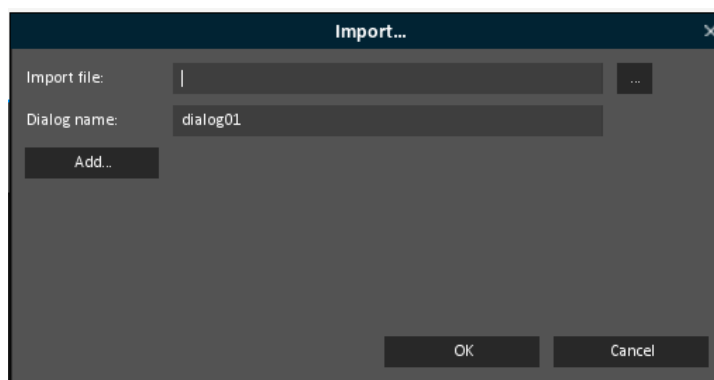


Fig 44 - Import

3.6.3 Import PSD-File

You can directly import PSD-Files into the GSE instead of placing all controls together into the dialogs. The imported dialogs will have the exact same content as in Photoshop and only need some dynamic behaviour added to them.

In the opening dialog you can choose the file you want to import. In the next dialog you can choose the dialogs to import and specify some options for the importer.

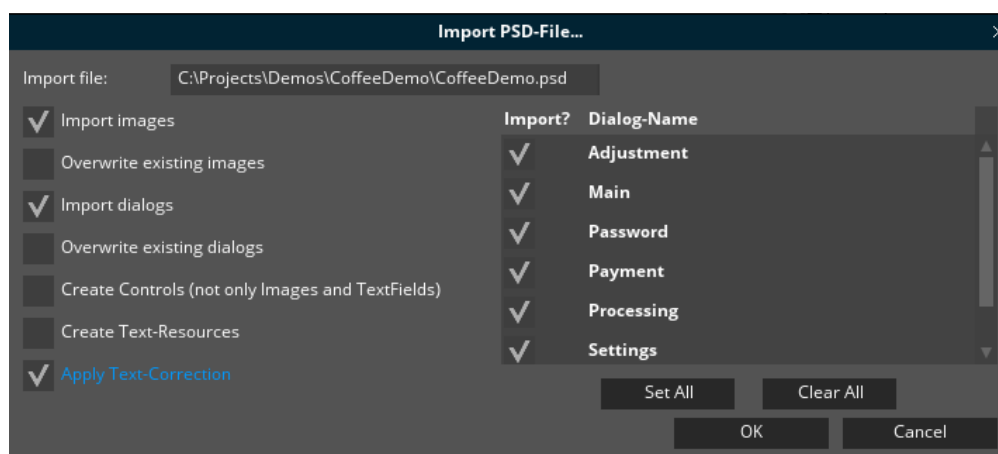


Fig 45 – Import PSD-File

These options will control the import-process and what you will get from the imported data:

- **Import images:** this will import all available images, but will not overwrite already existing images. If option “import dialogs” is active, only images from selected dialogs will be imported. **Note:** Images which are not in use anymore will not be removed automatically. Use “Resources -> Manage -> Find unused Resources” to remove these.
- **Overwrite existing images:** if this option is selected, also already existing images will be imported and overwritten. If you only want to import new images, keep this option unchecked.
- **Import dialogs:** this will create dialogs for all selected dialogs in the list to the left. **Note:** Text-layers are extracted as TextFields. Dimension of the TextField depends on necessary size to display the whole text in Photoshop. **Note:** To import the used true type fonts, they must be located in the same directory as the PSD file.
- **Overwrite existing Dialogs:** if this option is active dialogs which already exist in the project will be overwritten. If you only want to add new dialogs to the project, keep this option unchecked.
- **Create Controls:** If this is checked Guiliani-controls will be created according to the definition in the PSD-File. See the next paragraph on more details. If this option is unchecked the layers are imported as Images or TextFields. **Note:** Groups without any layers are skipped and not imported.
- **Create Text-Resources:** use this option to automatically create TextIds of the imported texts. This will simplify the process of internationalization.

Note: if the same text is used twice in the PSD-File the same TextID will be referenced after import and thus a change of its text will result in both texts changing.

- **Apply Text-Correction:** Since the GSE and Photoshop use different Font-rendering-engines some correction may be necessary to produce the best results when importing text-layers. If this option is checked all created TextFields will be shifted upwards while the distance depends on the used font and font-size. If this option is inactive the TextFields will be created on the same position where the frame of the Text-Tool in Photoshop was used.

3.6.3.1 Importing a PSD-File

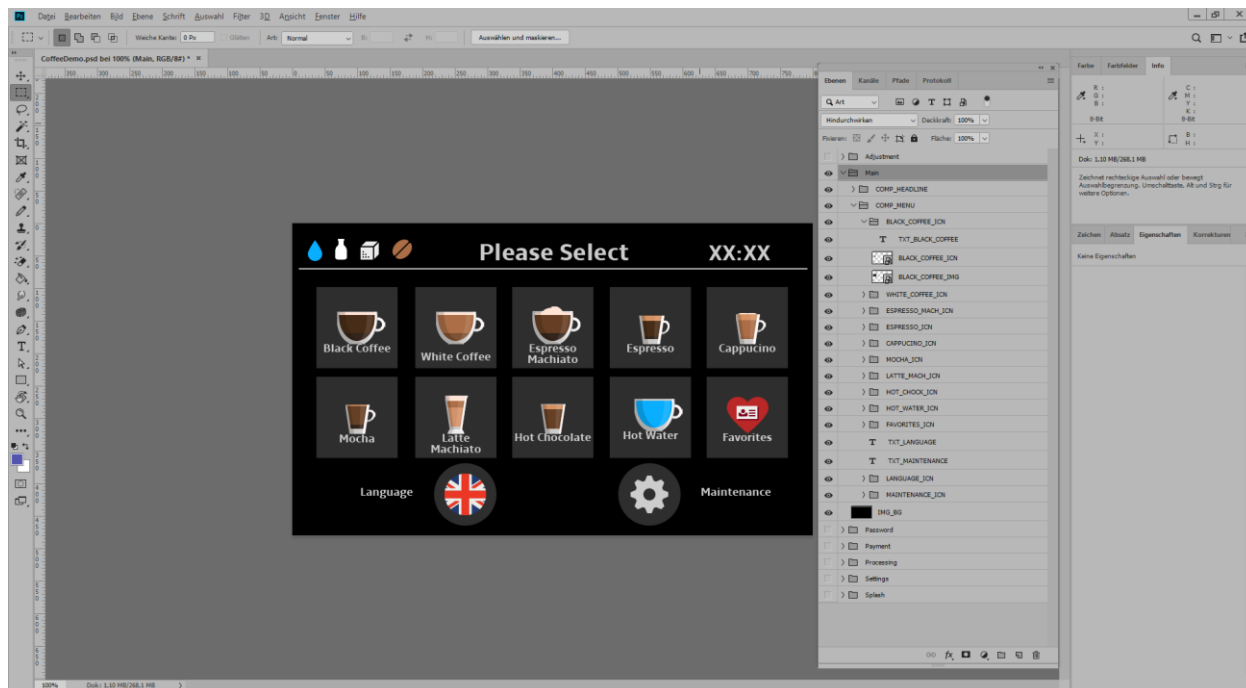


Fig 46 – Demo-project in Photoshop

Before you can import Guiliani-controls from a PSD-File you will have to do some preparations. Toplevel-folders are imported as dialogs with the same name.

Layers/Folders contained in these folders will be imported as Guiliani-Controls. What controls are created depends on their names, or specifically their suffix separated with an “_” character.

Note: Layers on the first level of the PSD-Hierarchy will not be imported.

If you don’t set a suffix for a folder it will be imported as a general container for grouping its contained objects. Layers without a recognized suffix will either be imported as an Image or as a TextField (if option “Extract Texts” is active)

Let's have a look on a typical layer-hierarchy to create a GUI and import Guiliani-controls.

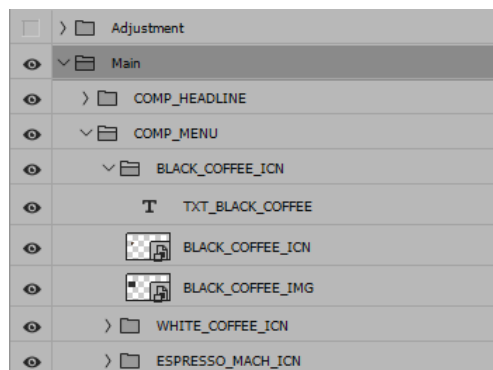


Fig 47 – Typical layer-hierarchy

In this example we have two toplevel-folders named “Adjustment” and “Main”. These will be imported as separate dialogs into the project.

In the folder “Main” we can see additional folders/layers which will be imported as the child-elements of the dialog.

The folders “COMP_HEADLINE” and “COMP_MENU” are imported as CompositeObject to group elements, because they do not have a special suffix.

The folder “BLACK_COFFEE_ICN” will be imported as a Guiliani-control, since the suffix “ICN” tells the importer to create an IconButton of this folder (see table in 3.6.3.4 for more supported suffixes for Guiliani-controls).

3.6.3.2 Set the Visuals of imported Guiliani-controls

To modify the visual appearance of a control, the layers contained in the folder will be used. To specify which part of the control should be set, the layers also need specific suffixes to their names.

Note: if a folder will be converted into Guiliani-control, all nested folders will be; only layers as direct children will determine the visuals of the control.

Let's get back to the layer-hierarchy seen before and examine the layer named “BLACK_COFFEE_ICN”. As we know this suffix will mark the folder to be imported as an IconButton.

For a list of available suffixes see the table in paragraph 3.6.3.4.

In our example the folder “BLACK_COFFEE_ICN” has several different layers which will determine the visuals for the IconButton:

- TXT_BLACK_COFFEE will be imported as the text written on the button
- BLACK_COFFEE_ICN represents the icon which will be placed on the button

- BLACK_COFFEE_IMG is the background-image for the button

Note: If there is more than one text-layer inside a folder, which will be converted to a Guiliani-control, one the first will be taken and the rest ignored.

Although the text-layer does not have a supported suffix (like _TXT) it will be imported as the text for the icon-button (as long as the option “Extract Texts” is active).

After the Main-dialog has been imported it will look something like this

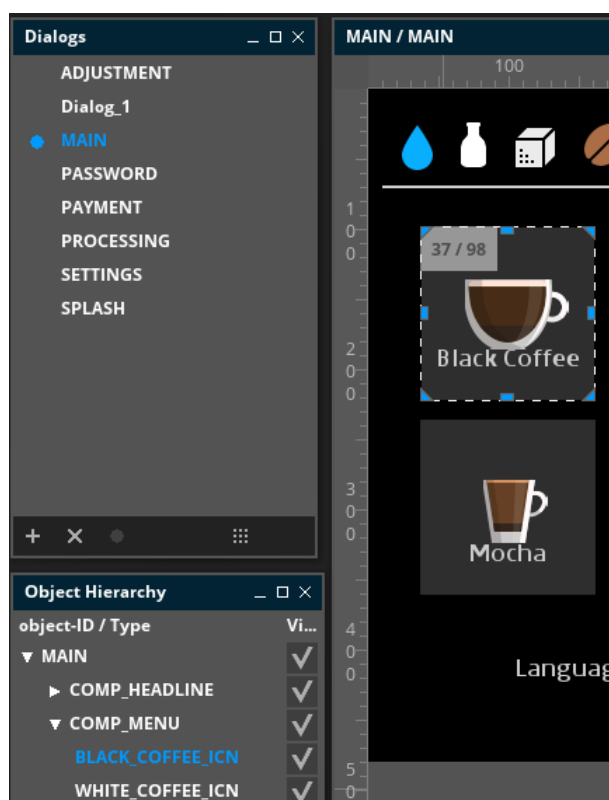


Fig 48 – Imported dialog from the PSD

3.6.3.3 Folder-Hierachy

Note: since Photoshop CS6 you can have a nearly unlimited folder-structure with a depth greater than 5. But if you want to maintain compatibility with older version you should not use more than 5 folder-levels.

3.6.3.4 Supported suffixes for layers

Suffixes for folders and corresponding Guiliani-control:

IMG	Image
BTN	Button
ANI	AnimatedImage
TXT	TextField
INP	InputField
ICN	IconButton
CHK	CheckBox
RAD	RadioButton
SLD	Slider
PRG	ProgressBar

Suffixes for visuals of the imported control:

IMG	Background-image for Buttons, IconButton, InputFields (will be used for all states)
IMGS, IMGH, IMGP, IMGG, IMGF	Background-image for the standard, highlighted, pressed, grayed-out or focussed state
ICN	Icon for IconButton (will be used for all states)
ICNS, ICNH, ICNP, ICNG, ICNF	Icon used for standard, highlighted, pressed, grayed-out or focussed state
IMGU	Image used when unchecked for CheckBoxes and RadioButtons (will be used for all states)
IMGUS, IMGUH, IMGUP, IMGUG, IMGUF	Images used when unchecked (used for standard, highlighted, pressed, grayed-out or focused state)
IMGC	Image used when checked for CheckBoxes and RadioButtons (will be used for all states)
IMGCS, IMGCH, IMGCP, IMGCG, IMGCF	Images used when checked (used for standard, highlighted, pressed, grayed-out or focused state)
IMGB, IMGF	Images for background and foreground of a ProgressBar
HND	Image for the handle of a slider (will be used for all states)
HNDS, HNDH, HNDP, HNDG	Images for the handle of a slider (used for standard, highlighted, pressed and grayed-out state)

3.6.4 Export

An export will copy all files related to your project (Images, fonts, language-files, XML files etc.) into a directory of your choice, from where you can copy them onto your target device (see figure 44).

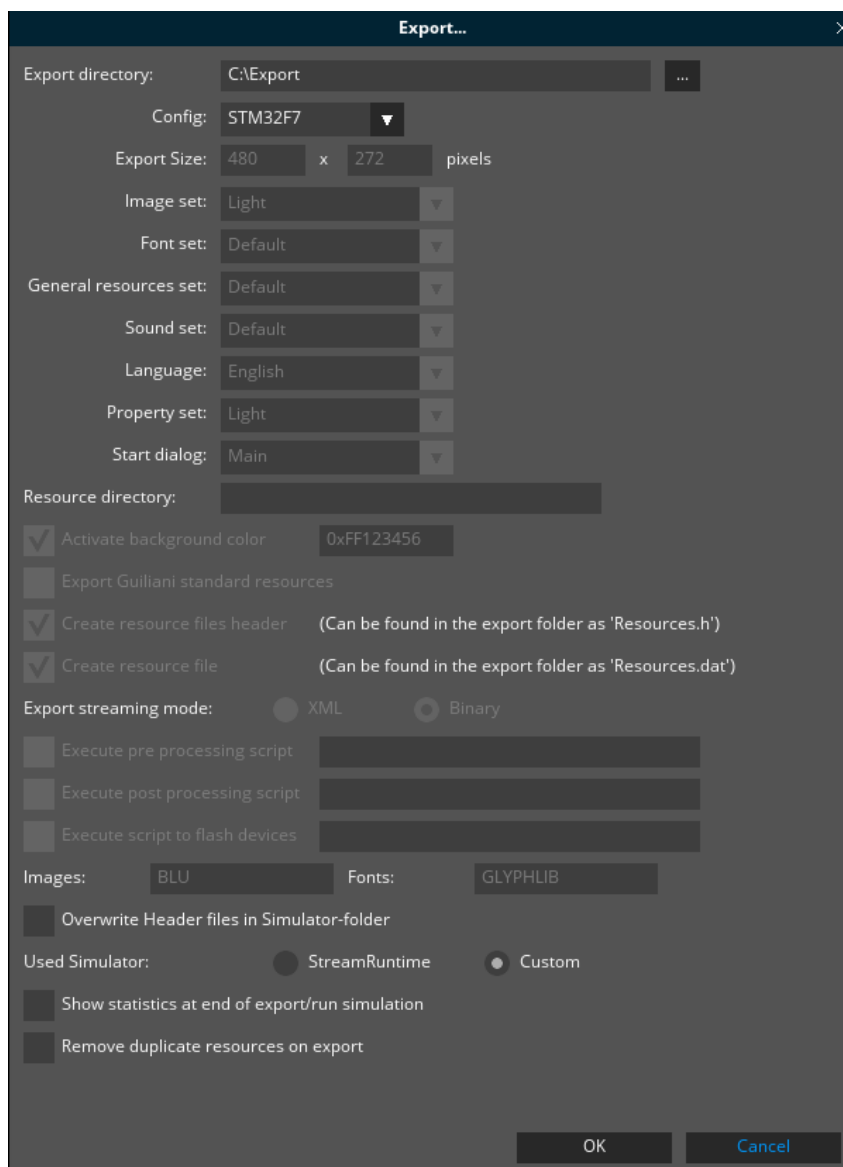


Fig 49 - Export

“Export Directory” option provides the path of the folder where the project needs to be exported. To select another path click on “...”.

The greyed-out settings are displayed according to the selected export-configuration. These settings can be edited via “Settings”-dialog in the tab “Export Settings” as explained in section 3.2.6.3

When object-ids referenced within the StreamRuntime-application have been changed, the option “overwrite Headerfiles in Simulator folder” should be used to overwrite the header-files inside the StreamRuntime source-folder to update all IDs properly on export/run simulation. Otherwise deleted resource-Ids will cause an incorrect order of the IDs used inside the application.

The “Used Simulator” option allows selecting whether the project is simulated using “custom” simulator or the “StreamRuntime” simulator.

The “Images” and “Fonts” option specifies the format in which images and fonts are used in the application.

When activating the “Show statistics” a small dialog is shown after export where all resources and their total sizes are shown. When using different export-formats the values of input and output can differ and thus the overall percentage of increase/decrease is shown.

The “Remove duplicate resources on export” option can be used to remove files which are identical in content but different in name and would thus consume more memory as really needed. If files with identical contents are found during export the first file is taken as the original and all following copies are replaced by references to this original, i.e. only one file is actually exported and the other Resource-IDs just point to the same file.

This option does not affect General Resources and operates only within the same resource-set, so different image-skins may have different duplicates.

A practical example would be to use a set of dummy-images during GUI-design each with a different name and ID, but same content. And now part by part the images are replaced with their actual correct image-file. Till this point the export with the activated option will help saving memory on the target-device during evaluation. When the GUI is finished all files are now replaced and the option will not filter out any duplicates and could be de-activated.

3.6.5 Manage Sets

The “manage sets”-window allows you to manage the different sets your application might use (i.e. Image, Text, Font, Sounds, General resources and properties).

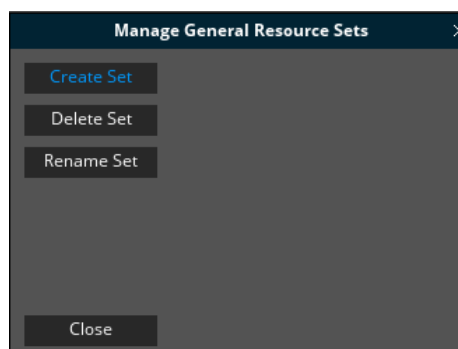


Fig 50 - Manage XXX Sets

The main view has four buttons, one for each of the options you have for managing a set, and a close button. The possible options in the managing of a set are: Create set, Delete set and Rename Set (see Fig 50).

When you click “**Create Set**”, a list of elements is made visible. Those are, a text field for introducing the name of a new set and a check button, if marked it will make visible a list of possible sets to copy the information from into the newly created set. At the bottom, a button named “Create” is visible; it will create the new set once you have introduced a valid name (see Fig 51).

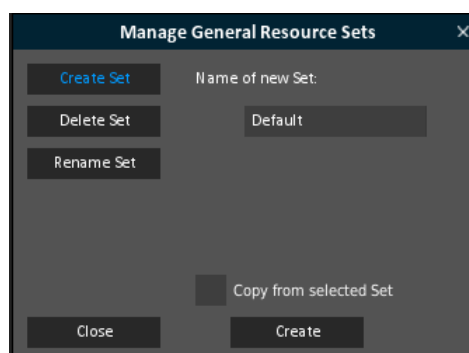


Fig 51 - Manage XXX Sets - Create

The next option of manage sets is to delete a set. That process begins by clicking the option “**Delete Set**”. Then, a list of existing sets will be visible, for you to choose which of them to delete. This deletion is executed when you click the button “Delete” (see Fig 52).

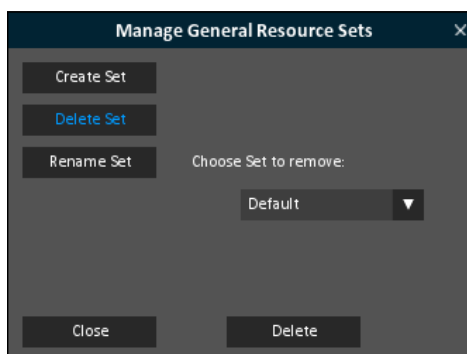


Fig 52 - Manage XXX Sets - Delete

If you click “**Rename Set**”, the list of visible elements shown contains a text field for writing the new name of the set, a list of sets from where you have to choose which set you’re going to rename, and a button named “Rename”, that will rename the chosen set after a valid name has been introduced (see Fig 53).

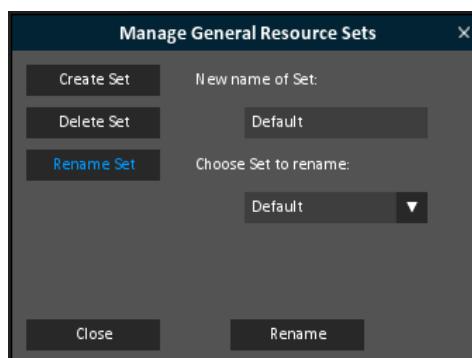


Fig 53 - Manage XXX Sets - Rename

3.7 Custom Extensions

In this menu a new Custom-Extension or “Custom StandardFactory” can be created.

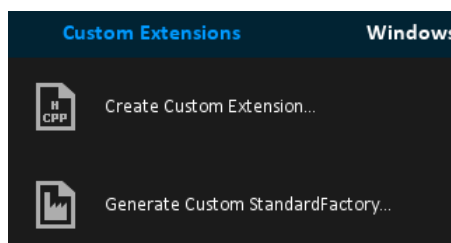


Fig 54 - Custom Extension

3.7.1 Create Custom Extension

A Custom-Extension enhances the variety of build-in Guiliani-parts (e.g. Controls, Commands, etc.) with customized elements the user can define. These Custom-Extensions can then be used directly inside the GSE as if they were built-in parts of Guiliani.

The custom extension wizard generates class stubs for customized controls, commands, behaviours and layouters and registers them as custom extensions to the GSE and StreamRuntime projects.

Note: For more information and an example, please refer to the document “Custom Extensions.pdf”.

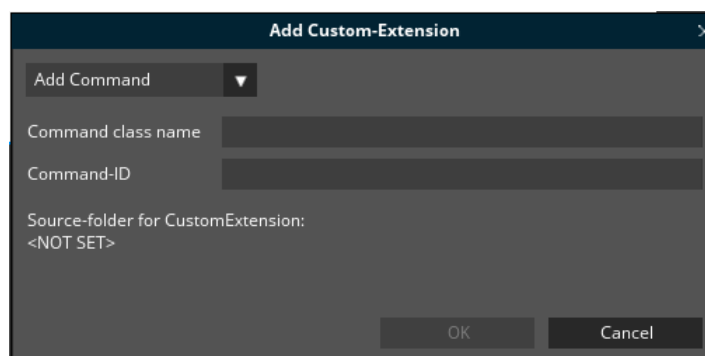


Fig 55 – Add Custom-Extension

The Combobox can be used to select the type of Custom-Extension which should be created. The possible values are:

- Command
- Control
- Behaviour
- Layouter
- TextType

XXX-Class name: in this field the name of the Custom-Extension is entered as it should appear in the GSE. This name can have spaces and special characters.

XXX-ID: this field is filled automatically when a class-name is entered and all unsuitable characters are converted into underscores “_”, since this ID needs to be an acceptable C++-identifier. All characters are converted into uppercase.

Source-Folder for Custom-Extension: this shows the folder where the GSE searches for the subfolders “Source/CustomExtension” and “Include/CustomExtension” to place the newly generated source- and header-files. This folder is set in the project-settings.

Note: Please select the main-folder of a valid StreamRuntime (e.g. “C:\Projects\Guiliani_SDK\StreamRuntime”) so the GSE finds the correct folders.

3.7.2 Generate Custom StandardFactory

The “Generate Custom StandardFactory” wizard helps to create your own custom standard factory. This factory will only contain entries for controls, commands, etc. that are used in the current project. This enables the linker to remove unused code for unused parts from the application-binary.

Note: If controls which are not generated into the Custom-Standard-Factory are used by the application code, these will be added automatically by the linker.

3.8 The Windows Menu

All editor windows (Controls window, Workspace window, Object Hierarchy window, Attributes window and Dialog window) can be manually moved around or can be hidden. To reopen specific windows, use the windows menu (see Fig 56).

The option “Auto-Arrange Windows” will open all available windows and properly arrange them within the currently available space of the editor.

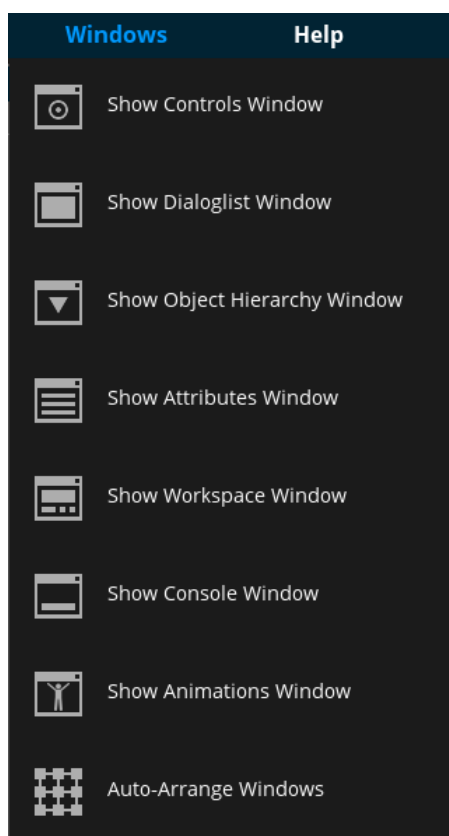


Fig 56 - Windows Menu

3.9 The Help Menu

Important information about the editor is present in the help menu.

The “About” will open a window containing the editor version, copyright and contact information. Also the currently used number of resources and their respective limits, if set, are shown.

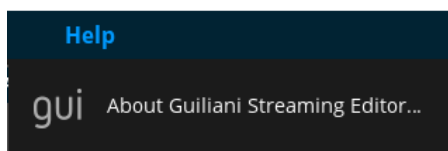


Fig 57 - Help Menu



Fig 58 - About Dialog

4 The Dialog-List-Window

The “Dialog-List” window lists all dialogs of your project. The highlighted dialog is currently visible in the Dialog window.

You can switch the dialog currently edited in the Dialog-Editor by selecting one of the listed dialogs.

At the bottom of the window there are several buttons to operate with the dialogs which are referred as short cut buttons in Fig 59. Following text describe the use of these buttons:

- To add a new dialog use the “+” icon.
- To delete the selected dialog use the “x” icon (at the window’s bottom).
- The third icon from the left denoted with a circular dot enables you to set the selected dialog as first dialog when the project is opened using recent projects or open project option under file menu.
- Switch between text list and thumbnail display with the icon on the right.

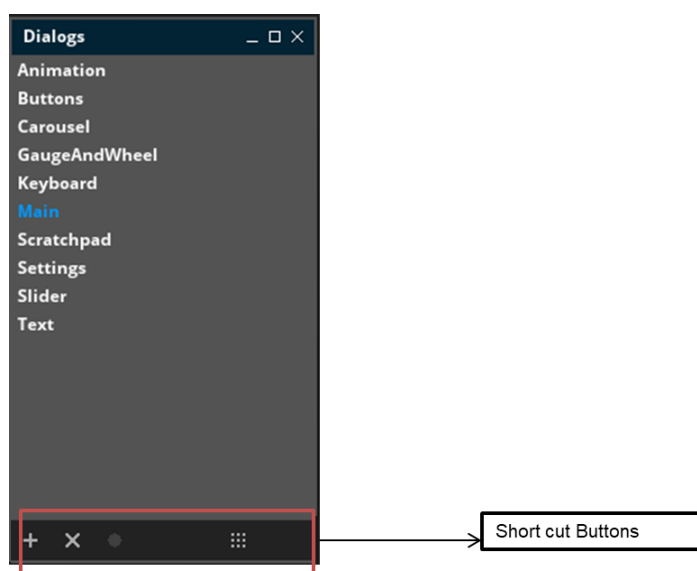


Fig 59 - Dialog List Window

5 The Workspace-Window

Use the Workspace window to arrange objects on the screen.

Drag and drop them into or out of containers or resize them (use <CTRL> to select multiple objects).

If an object is selected all of its attributes are listed in Selecting objects

You can select objects by clicking on them. A selected object is displayed with an outlined animating rectangle. To de-select an object, just press the CTRL-key and click on it.

When you want to select multiple objects you can click on each of them holding down the CTRL-key or hold down the SHIFT-key and draw a frame. This will select all objects which are contained inside the drawn frame.

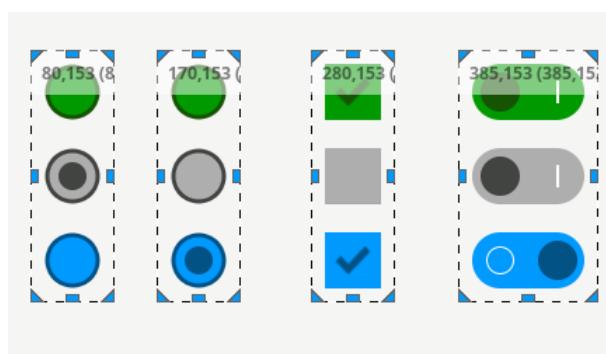


Fig 60 – Selected Objects inside Workspace-Window

5.1 Moving and resizing objects

To move an object, just select it and move the mouse while the left button is pressed. While moving the object the numbers which are displayed in the upper/left region tell you the current position of the object – first absolute, then relative to its parent in brackets – and the size.

When you want to resize an object, just select it and drag one of the triangular- or rectangular-shaped and coloured points around the object-frame.

Note: If the selected object has a Layouter attached, it might not be placed or have the size as expected after the mouse-button is released, since the Layouter will override position/size according to its settings.

In that case it is recommended to deactivate the Layouter, move the object and set a new Layouter.



Fig 61 - Moving/Resizing an object

5.2 Guidelines

The artist positions the objects in the dialog. Guidelines assist here by snapping the objects to a common edge. The snapping feature is provided between the moved object and each other object for the left, right, upper and lower edge, as well as horizontal and vertical center line. The guidelines can be toggled on/off in the tools window at the extreme left corner of GSE or in the view menu at the top of GSE.



Fig 62 - Assisting Guidelines

6 The Attributes Window

The attribute window allows the user to edit the object properties. Add object specific Image-, Text-, Sound- or Font IDs, commands, layouters or behaviours. Assign an ID to an object to identify it in [The Object Hierarchy Window](#) and reference it from other objects through commands, behaviours etc. (see Fig 64).



Fig 64 - Attribute Window

6.1 Images

Objects may have five different states (for example the button). These include normal, highlighted, pressed, greyed out and focused. In the Attributes window, you can add the state-dependent image to the object by clicking on the appropriate image ID button and selecting the desired image in [Manage Images](#) window.

6.2 Texts

Objects with texts have a text string that can be edited in the “Text” input field of the Attributes window. For internationalized Text, use the Text ID-Button and select a Text ID. If you do not have a Text ID yet, click on Manage Texts in the dropdown box, which will open the [Manage Texts](#) window. Here you can define Text IDs and translate your texts in different languages.

The position of an object’s text is per default fixed in the left corner. If you resize the object, its text remains as is, it will neither reposition nor resize. You can thus freely position a text inside its associated object by changing its TextXPos / TextYPos and TextWidth/TextHeight attributes. Additionally you can set the texts alignment by changing the VerticalAlignment- or HorizontalAlignment attributes.

6.3 Fonts

While you can make changes to the text colors in the Attributes window itself, you will need the [Manage Fonts](#) for changing the font face or the font size. Press the TextFontID button in the attributes window to select the object’s Font ID.

6.4 Commands

Commands encapsulate specific actions. In this way they can easily be reused and attached to objects within the GUI. Commands can cover calls to functions of the underlying application log (e.g. Start Playback of an MP3 File) or trigger actions inside the GUI itself (e.g. Switching screens).

They can be directly assigned to certain widgets that support them, for instance buttons or input fields. Commands are typically executed by the widget when certain conditions are met or specific events occur. For instance, the button executes its command when clicked on. The input field executes its command when the user finishes input by pressing ENTER.

If you wish to execute commands from widgets that do not have ready-to-use Command-slots, you do so by using the SingleCmdBehaviour. This predefined standard behavior enables you to execute commands for any object in reaction to various events (e.g. Clicking, Dragging, Getting the focus...).

There are predefined commands that you can use, like the Quit command, it provides a simple way to create a ‘quit application’ button. Another command that Guiliani offers by default is the load dialog command, it loads another XML file at runtime. This is useful to create screen switches or pop-ups. For real applications, you can extend this list with your own application specific commands and comfortably use them within GSE.

Commands can link to another command to execute more than one action at once, when building a command chain. To do this, look for the attribute ‘AdditionalCmdCount’ in the attributes list and click the button ‘Add more’ next to it.

6.5 Behaviours

Behaviours are used for adding functionality to Guiliani’s event slots. Each widget has numerous event slots that are called by the framework when specific events occur, like key presses, mouse clicks, mouse drags and so on. Please refer to Guiliani’s documentation for in detail information on this concept.

A powerful aid when creating your GUI will be the behaviors related to hotkey-handling and command execution, these are explained now. The attributes related to this can be seen in Fig 65.

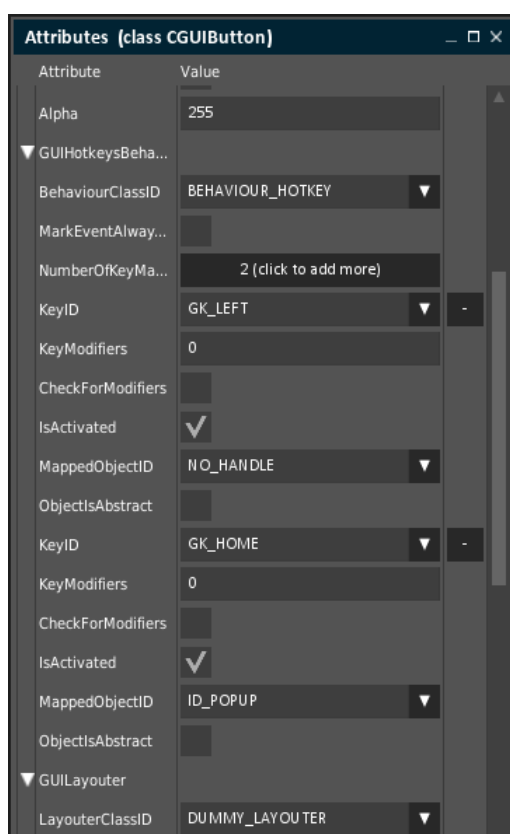


Fig 65 - GUI Hotkey Behaviour

Behaviours and commands can be tied together by using two special behaviors, the MultiCommand and SingleCommand behaviours.

Select the multiple commands behavior for assigning commands to each of the event slots, or use the single command behavior to choose one event slot only and attach a command to it.

The Hotkeys behavior reacts to key presses by the user. It is triggered whenever a key press occurs and is not handled in any way, for instance by a widget. Keys map to object IDs: When a defined key press is detected, the mapped object is searched and, if it was found, then ‘clicked’. In other words, you can use this behaviour to simulate clicking on objects through keyboard keys.

The Hotkeys behaviour should be added to a parent object (the root of an object group). Add the ASCII key code in the KeyContent text input field. You can remove the key mapping by pressing the X-icon on the right side. Map the behaviour to an Object ID by selecting it through the appropriate button. If the user presses the key, the mapped object will react like it was clicked, for example, a check box will switch from selected to unselected when the key is pressed.

6.6 Basic Object Specific Attributes

Each GUIObject has some basic attributes. These are explained below.

6.6.1 Position

Every object has an X and Y Pixel-Position in your GUI. If you add a new control to your GUI it will be displayed in the left corner (zero-point) of your GUI, meaning its coordinates will be X=0 and Y=0. Guiliani has the capability to position objects sub-pixel accurate, so XPos or YPos can have inputs like 0.5.

Positions are relative to object’s parents, so an object that seems to be located in the middle of a dialog does not necessarily have coordinates that are close to the dialog’s center. Instead, the coordinates express an offset to the object’s parent (the container that contains this object, for this look for the object that is one level up in the hierarchy view).

6.6.2 Width and Height

An object’s width or height is given in pixels and can be changed in the Attributes window, or by dragging the corners of the object in the preview window.

6.6.3 Focusable, GrayedOut, Disabled and Invisible

A focusable object can receive the focus in the running application. This is necessary if the object is supposed to be selectable via keyboard / cursor-keys.

A greyed-out or disabled object will not react to user input in the running application, so it cannot be clicked and will not be focusable either.

An invisible object will not be displayed in your application and will therefore not react to any events.

6.6.4 BehaviourClassID

Choose behaviour by clicking the button and selecting a behaviour class ID from the list. The chosen behaviour's specific attributes become visible in the attribute list in the rows following the class ID. See also [Commands](#) and [Behaviours](#).

6.6.5 LayouterClassID

Layouters are used to automatically influence the position and/or size of child objects within a composite object. For instance, the LAYOUTER_ANCHOR can be used to 'fix' the edges of a widget to its parent. Try this by creating a composite object, putting an image inside it, assigning the LAYOUTER_ANCHOR to the image, activating the "AnchorBottom" and "AnchorRight" check boxes and then resizing the composite object. You will see how the distance of the image's bottom and right edges to the respective parent's edges are always kept the same, moving the image around as you resize its parent.

There are further more layouters available, for example for automatically arranging child objects in a list container. Please refer to the Guiliani documentation for details.

6.6.6 Object ID

Object IDs are symbolic names. Assign an object ID to your objects to find them in the Object Hierarchy window. Do this by adding a new Object ID in the Attributes window.

Note that Object IDs are also used by many commands and behaviors in order to reference specific objects inside the UI. Renaming an object will not automatically update these references and might cause unexpected behavior.

The option "search object-id" can be used to search for object-ids meeting certain specific requirements like objects in the current dialog or child-objects of the currently selected object.

The search dialog appears when “Find Object Id” is clicked which comes as an option in drop down list for Object ID attribute in attribute window (see Fig 66). Similarly selecting “new object-ID” from the drop down allows the user to add the desired object ID name for the object (see Fig 67).

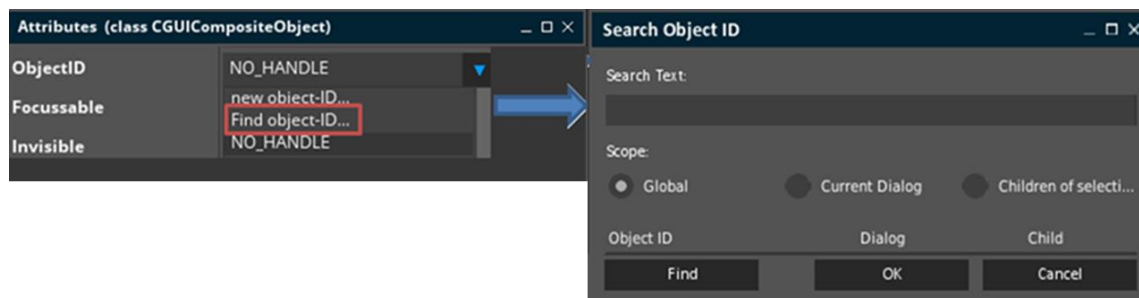


Fig 66 - Search object-id

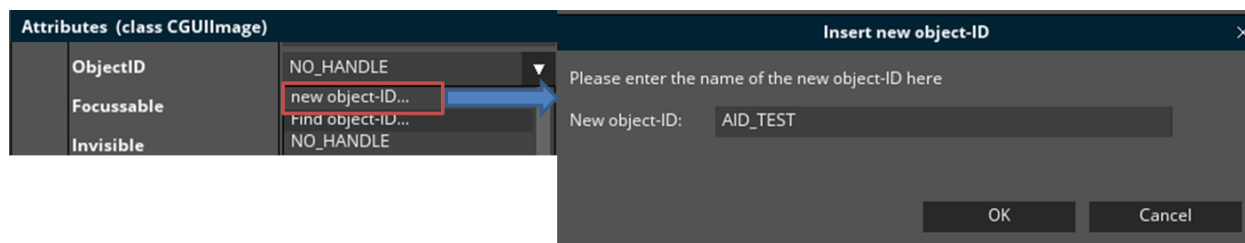


Fig 67 - Insert new object-id

6.6.7 Colors

If an object has a colour which can be changed (e.g. TextColor or BackgroundColor) the color-selection dialog can be used to easily pick a color or choose from various favorite colors.

The dialog is opened by clicking on the button displaying the hexadecimal color-code next to the attribute-name.

Favorite colours can be added by using the button “Add to Favorites”. These colours are saved with the project and can be selected from the upper list.

Note: If the list of favourites is full the first added colour will be replaced by the newly added.

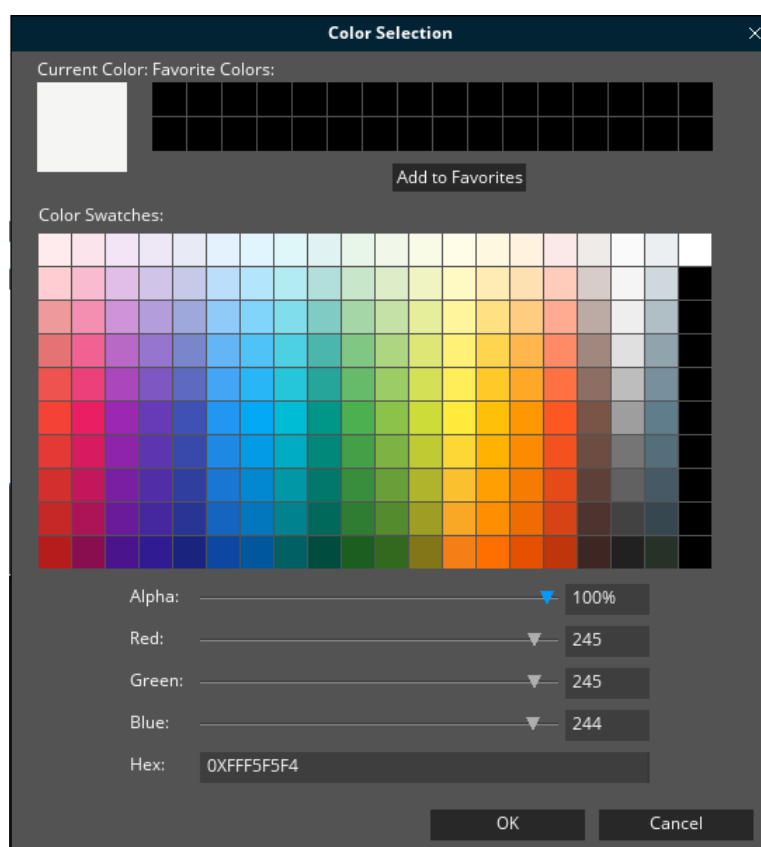


Fig 68 - Color Selection

For some attributes you can also choose between setting a static colour-value or a property to be used as a colour. When using properties you can easily define your own colour-schemes and switch them during runtime.

7 The Object Hierarchy Window

The Object Hierarchy window displays all GUI objects contained within the selected dialog including their hierarchical structure. Objects, that cannot be seen on the Dialog window because they are below other GUI elements, can be selected in the Object Hierarchy window instead.

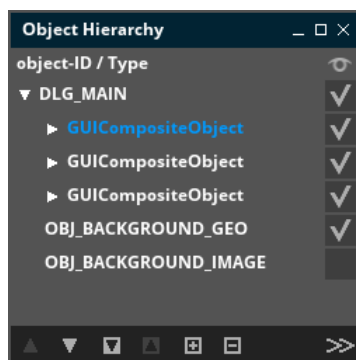


Fig 69 - Object Hierarchy

7.1 Selecting objects

If an object is selected in the object hierarchy, it is selected in the workspace-window and its attributes are displayed in the Attributes Window. It is possible to select more than one object but the display of attributes is supported for only one selected object and the attributes window remains empty if more are selected.

7.2 Change hierarchical order of objects

For changing the hierarchical order of a GUI Element, use the arrow icons at the bottom of the window. Mind that the only possibility to extract a group's sub-object is to drag and drop it out of the group into another container or the Dialog window.

7.3 Edit objects

Within the Hierarchy window it is possible to copy, cut, paste and delete objects.

To copy, cut or delete objects, they must be selected beforehand. A single selection can be made by selecting an object with the mouse.

A multiple selection is also possible by mouse with the help of the Ctrl or Shift key as usual. The desired action can then be selected either via the menu under the item "Edit" or with the usual key combinations.

When inserting an object, it is inserted at the topmost position in the active container. If more than one object is selected when inserting, the insertion is made in the uppermost object.

7.4 Visibility of objects

Toggle the visibility of an object by clicking on its visibility check box (eye symbol).

7.5 Zoom in/out

If you have many objects in the current dialog, you can “zoom-in” to a specific container by double-clicking on it (either in the Hierarchy-window or the Workspace-window) or by using the appropriate buttons in the toolbar. This way you will not be distracted by other objects.

To zoom back to the whole dialog double-click again or use the toolbar-buttons.

Note: Some composite-objects (e.g. ScrollView or Carousel) may contain content which is larger than the area covered by the object itself.

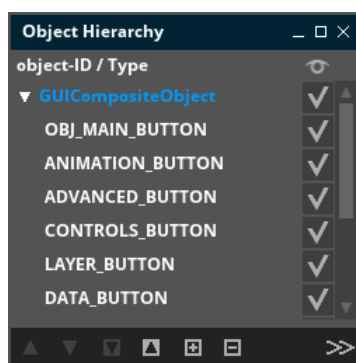


Fig 70 - Object Hierarchy (Zoomed in)

7.6 Expand or collapse object tree

In addition to zooming in on individual objects, the Hierarchy-Window also offers the option of expanding or collapsing the entire object tree. This is done with the + and - buttons in the lower button bar of the window.

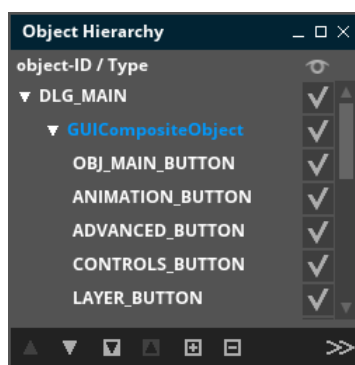


Fig 71 - Object Hierarchy (Expand tree)

7.7 Additional Object-Information

With the ">>>"-button in the lower button bar of the window you can show or hide the Additional Object-Information. This provides a quick overview on objects using commands, behaviours or layouts. However, these can only be changed in the Attributes window.

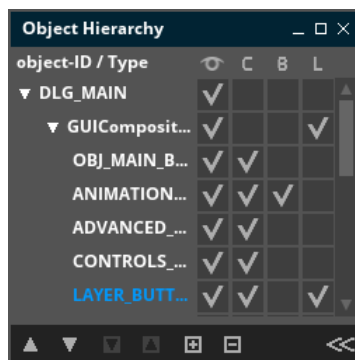


Fig 72 - Object Hierarchy (Additional Object-Information)

8 The Controls Window

You will find all Guiliani Standard Controls in the Controls window. If you click on one of the icons, a new instance of that control will appear in the upper left corner of the current dialog, or the currently selected composite object.

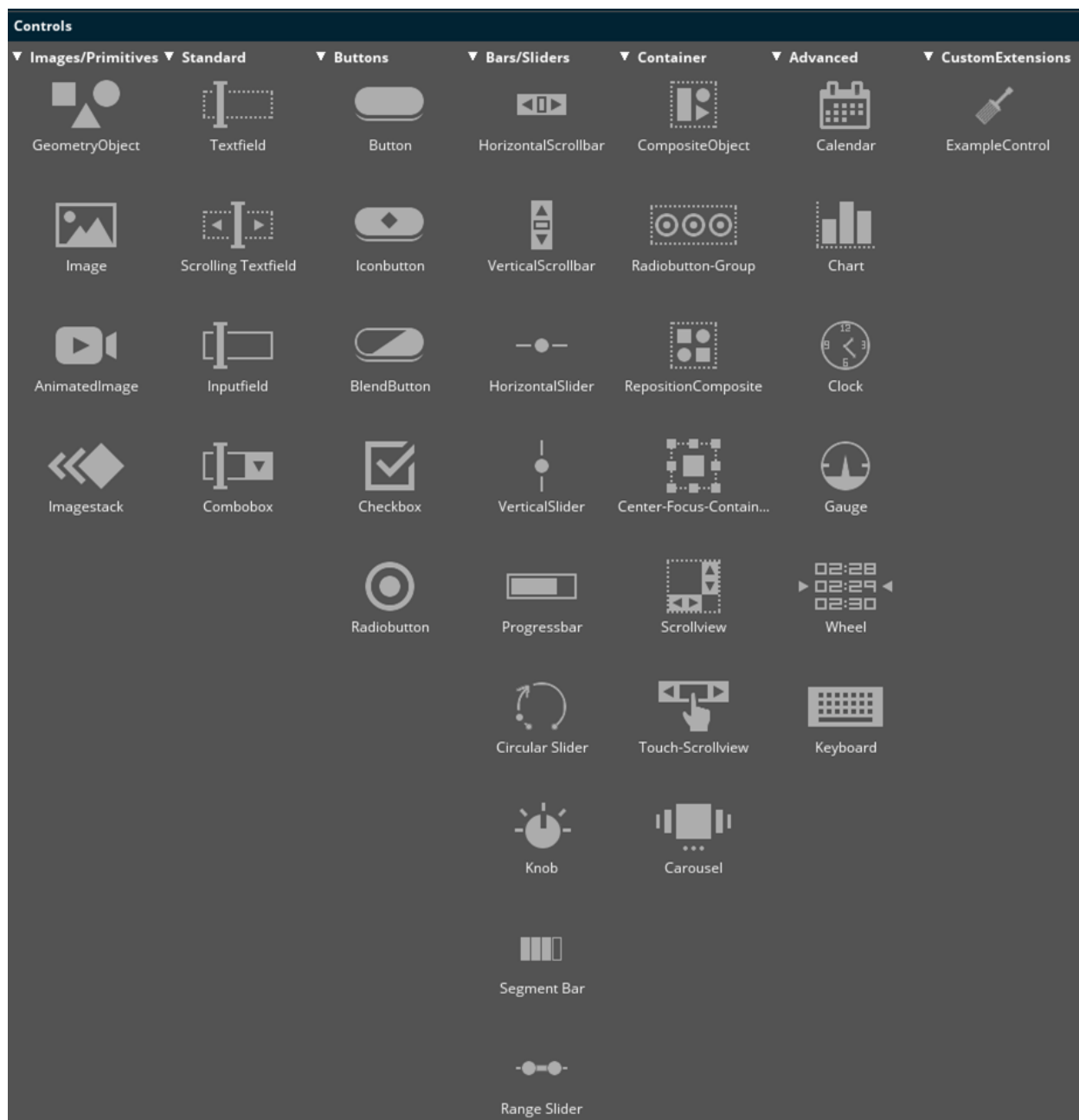


Fig 73 - Controls

A detailed description of every single attribute of each control can be found in the document “GSE Control Attributes”

Many Controls need image IDs to be displayed. There are up to five states that need to be provided with image IDs: standard, grayed out, highlighted, focused, and pressed. Controls that need one or more image are marked with an * after the name of the control in the following list.

8.1 Images/Primitives

In this group all controls representing images or graphical primitives can be found.

8.1.1 Geometry Object

The Geometry Object offers a simple way to add geometrical primitives, such as lines or rectangles, to your user-interface without writing customized drawing code.

8.1.2 Image*

The Image control in the control window is used to get the visually appealing images. By clicking on this control the placeholder for the image appears inside the workspace window and the user can add the required image in the placeholder from the image set.

8.1.3 Animated Image

With this control, the user can add multiple images and get the images displayed one after the other to produce the animation effect.

8.1.4 Image Stack

Image stack will be used for showing pseudo 3D transitions: one image fades out into the background while getting smaller; another one comes in from the front, getting smaller and fading in. The opposite direction works similarly: the images appear to come from the back to the front while fading. In order to see the effect you must run a simulation of the GUI, focus the image stack object and cycle through it using the PageUp / PageDown keys. (Of course, it is possible to map these control keys onto other inputs for the target hardware.).

8.2 Standard

This group contains all controls which are found in a standard GUI-application for selecting or displaying information.

8.2.1 Text Field

Use text fields for displaying texts inside your GUI. Either the text can be set hard coded or via text ID, we recommend the latter one, so the GUI is ready for internationalization. It is adjusted by selecting an appropriate font ID and colors for the different states.

For some extra convenience, the text field provides the option to set a background image.

8.2.2 Scrolling Text Field

With this control, text can be scrolled from the right to the left or vice versa, as well as from top to bottom or vice versa. In addition, you can define the scrolling speed.

8.2.3 Input Field*

The Input field consists of an editable text. The user can enter the desired text when this control is used in the application. The background image can also be set for input field using GSE attribute window.

8.2.4 Combo Box*

The ComboBox offers a Drop-down list of entries from which users can make a selection. Advanced features include the possibility to search for entries by typing substrings into the header and to add new entries at runtime in the same way.

Please be aware that even though you can create and edit the ComboBox inside GSE, you will currently not be able to fill it from there. Nevertheless, you can access the ComboBox from source-code via its ObjectID and dynamically fill it at runtime.

8.3 Buttons

In this group all sorts of buttons can be found.

8.3.1 Button*

Buttons are one of the main control elements. They trigger some action, like open, export, ok or cancel.

Buttons allow commands and behaviors, which makes their usage more flexible.

The buttons can be altered in their visual appearance by providing image IDs for the different states. It has all five states that are mentioned above. For each of the states a different image ID can be set.

8.3.2 Icon Button*

The icon button is a standard button with an additional image as overlay whose position can be changed independently. Both the main button image and the icon have five states (standard, grayed out, highlighted, focused and pressed).

8.3.3 Blend Button*

The Blend Button is a specialization of the standard Button, which softly blends between the various state images instead of simply switching them. Be aware that alpha-blending can be time consuming on low end platforms, in particular if blending is done using a software-renderer

8.3.4 Check Box*

Check boxes provide a binary option, either check or not. They are useful for checklists and optional choices. A full-customized checkbox has 10 images due to five states when checked and the same five states when unchecked.

Check boxes allow action methods, commands and behaviors. In this way their usage is more flexible.

They can be customized by setting a text that is displayed inside the check box. Image and text can be aligned independently to satisfy individual needs. The checkbox layout option provides different layout possibilities for the checkbox.

8.3.5 Radio Button*

Radio buttons present a choice between several mutual exclusive options. Therefore, they come in groups, from which exactly one is checked at a time.

Note that Radio buttons **HAVE** to be attached to Radio button groups in order to work correctly.

8.4 Slider and Bars

8.4.1 Horizontal and Vertical Slider/Scrollbar*

GSE control window offers horizontal and vertical sliders and scrollbars, with a background image and a knob. The knob inside the slider and scroll bar can be moved from one end to other based on the span size set by the user. Slider allows the user to select a value or range from a fixed set of options. The slider uses a knob to control a variable such as volume on a radio or brightness of a screen.

Scroll Bar present in control window can act as a slider which the user manipulates to set a value. The sliders and the scrollbars can be set with either horizontal or vertical orientation.

8.4.2 Progress Bar*

Sometimes a task running within a program might take a while to complete, using progress bar, user can track the progress of the task that is, how long a task might take and how much work has already been done. The attributes of progress bar are quite similar to the ones provided for slider.

8.4.3 Circular Slider*

The attributes of circular slider are very same to the horizontal/vertical slider and scrollbar. The difference being it has small circular slider moving in circular direction.

8.4.4 Knob

The Knob has similar attributes as in circular slider. The only difference being that unlike circular slider, the whole circular disc can be rotated.

8.4.5 Segment Bar

The Segment Bar can be used to display ratings or similar content. You can choose an inactive and an active image which will used to display the currently set value of the bar.

8.4.6 Range Slider

The range slider is a two-ended slider whose start- and ending-point can be moved independently to define a range. Additionally the complete range can be moved using the middle of the control.

8.5 Container

Here you will find all container-controls available for grouping or dynamic placement and interaction.

8.5.1 CompositeObject

CompositeObjects serve as containers, which you should use to group other objects. This has several advantages:

- Moving the container will move all child objects, as well
- Resizing the container can resize the children (when using Layouters)
- Marking a container as Invisible will render its children invisible, too
- You can use the container to clip its content. (That means everything contained inside the Composite object, but positioned outside its dimensions will not be visible).

8.5.2 Radio Button Group

The radio button group is a specific composite object in which you can align radio buttons. In a radio button group, only one radio button can be selected. All others are unselected, even if all radio buttons have the selected checkbox in the Attributes window selected.

8.5.3 Reposition Composite

A reposition composite object is a composite object that will automatically reposition its child objects. You can align all children at the top, bottom, left or right of the composite and define the space between the child objects and the space to the composite-border. You can even have a composite object in a reposition composite and vice versa.

Note: The repositioning effect will only take place when resizing the container. Therefore, drag its lower right corner to see the effect.

8.5.4 Center Focus Container

The center focus container changes its own position with an animation so that the focused child object is centered on a specific point (try it; it is much easier to see than to describe). You can define this “center point” with the CenterX and CenterY coordinates that are given relative to the Center Focus Container’s parent.

8.5.5 Scroll View*

A scroll view object is a specific composite object that represents a view window onto a larger scrollable area contained therein. This is useful when you need to make a larger amount of information or widgets accessible on limited display space.

Take a look at the Object Hierarchy window. You will see that the scroll view object consists of several objects. Two of these are a vertical and a horizontal scroll bar. An additional composite object keeps the actual content that shall be visible in the scroll view window. As long as the content is smaller than the scroll view window, the scroll bars are set to invisible. If the content's height (or width) increases, the scroll bars will be automatically visible so that you can scroll the content. The scroll bar's visibility policy may also be changed so that they are always visible (see The Object Hierarchy Window for information on how to "zoom in" on the content of Scroll Views). All objects that should be displayed in the scroll view object must be child objects of the "ScrolledContainer".

8.5.6 Touch Scroll View

The "touch scroll view" is a specialized scroll view that is optimized for touch screens and supports scrolling by dragging as well as kinetic effects

When dragging, the content of the scroll view follows the finger/mouse. Besides that, it is possible to activate kinetic scrolling which will trigger a scroll animation after the drag that slowly fades out with time. The animation speed depends on the drag speed, which means that quick fling gestures will result in faster scrolling. Optionally it is possible to activate a bounce back factor, which will invert the animation direction when the edge of the scroll view is reached.

8.5.7 Carousel

Objects you add to the carousel will automatically be positioned in a 3D rotating wheel. The focused child is always moved to the 'front' of the carousel with an animation. The carousel's tilt angle and radius can be set with the respective attributes. The radius should be large enough to allow children to be positioned far enough apart from each other. However, keep it small enough to avoid clipping of the children at the carousel's edges. The size of the children cannot be changed.

8.6 Advanced

The advanced controls includes gauge, wheel, keyboard, calendar, chart and clock

8.6.1 Gauge

The gauge control is used to visualize values on a meter using a needle. The visualization can be customized by either supplying images, for both the background and the needle, or by drawing the needle with a line of customizable color and length. The usage of the gauge control is very easy, as only minimal value, maximal value and maximal rotation have to be defined. The current position of the needle will be calculated automatically through the given value. The "gauge" control can be used in a wide range of use-cases e.g. to present an analog clock or a speedometer.

8.6.2 Wheel

The wheel-control can be used to display different numeric entries in a horizontal or vertical way. An entry is selected by dragging the wheel to this entry. Font and color can be adjusted for normal entries as well as for the selected entry.

8.6.3 Keyboard

The on-screen keyboard is mainly used for entering text into input fields when you don't have a physical keyboard available on the target.

8.6.4 Calendar

The calendar displays the currently selected date. You can change the date by clicking on a different day, swipe vertically for a different month or horizontally for a different year.

8.6.5 Chart

The chart control is used for displaying continuous, discrete data. You can either have points, lines or bars to display the various datapoints and also specify the colors used for representation. A DataPool-object can be linked to this control to display its data.

8.6.6 Clock

This control is used to display an analogue clock. The handles can either be images or lines which can have shadows.

8.7 Custom Extensions

Here all available Custom-Extensions of the Control-type are listed.

8.7.1 Example Control

This control is an example CGUIObject implementation that draws a rectangle with configurable border width and configurable colors. This is available as source code and you can use it as an example for creating your own controls (see [Custom extension](#)).

9 Animation Window

To create dynamically changing objects and combine these into animations the GSE can greatly speed-up your work.

Using the built-in animation editor you can easily define animations working on several objects and which can be started on runtime by using a special command.

If the animation-editor is not shown inside the GSE you can open it using the window menu: as shown in Fig 74

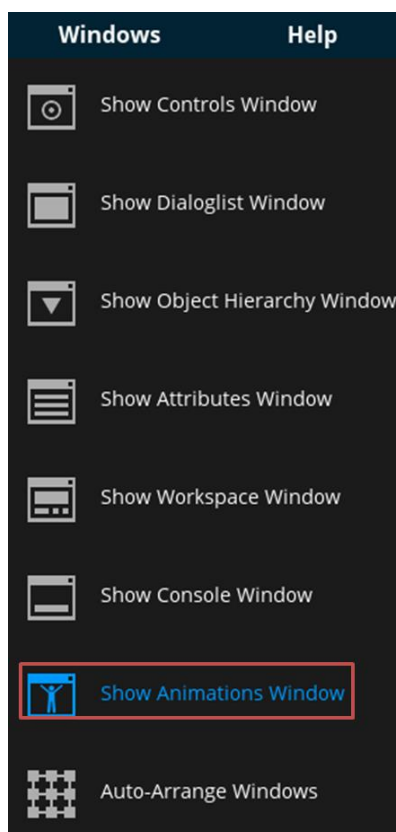


Fig 74- Open Animation Window

The animation editor is divided into two parts. The left side lists all available animation chains (see Animation Chain List). The right side shows the contents of the selected animation chain. This is a list of animations shown in a timeline. The timeline has an additional control element on the top, which is called timeline navigation (see section Timeline Navigation).

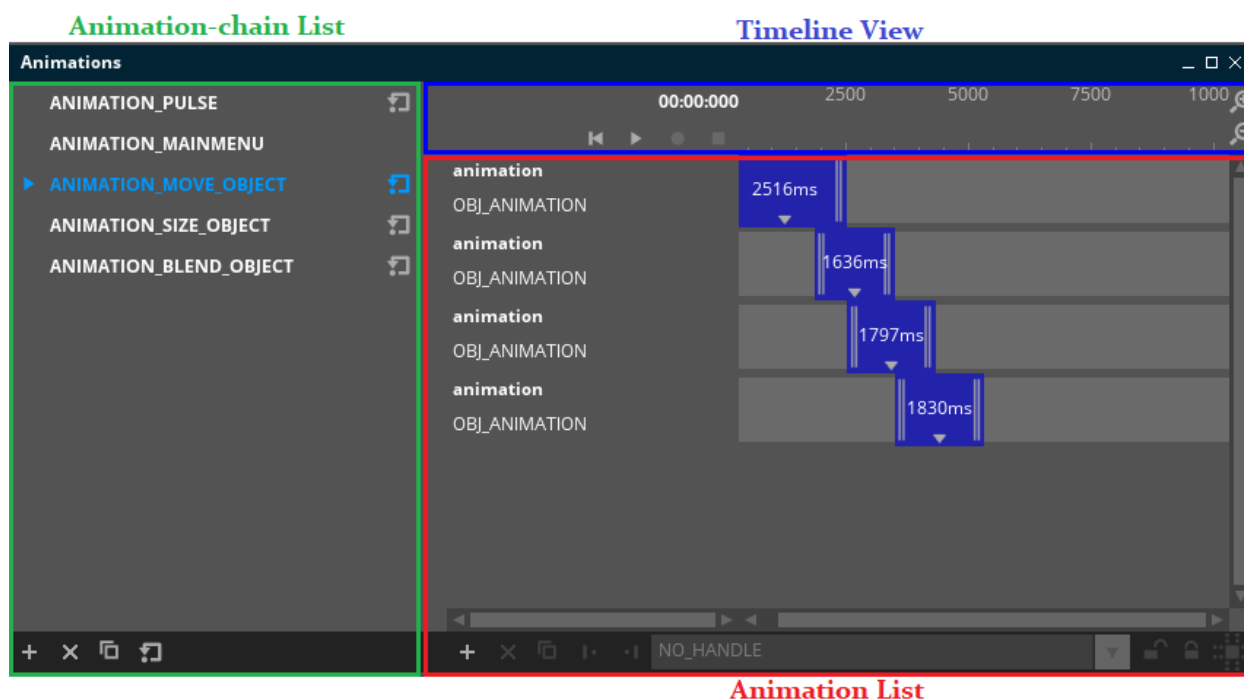



Fig 75 - Animations Window

In general you can copy (CTRL + C), paste (CTRL + V) or delete (X-button or DEL) selected Entries and add a new one with the plus (+)-button at the bottom. You can change the name of an animation-(chain) by double clicking on it. By pressing the square button you can create a copy of the selected entries.

9.1 Animation Chain List

Each entry in this list represents an animation-chain.

When you select an entry by clicking on it or drag and drop it to a new position, the animation chain opens. The currently opened animation-chain is marked with the  symbol and the contained animations are listed on the right side in the Animation List. You can close an open animation-chain by holding the CTRL key and clicking on it.





These buttons are at the bottom of the list



-button: add a new entry



-button: delete an entry or selected entries

An animation-chain is either cyclic or not. A cyclic animation-chain is marked with the  symbol on the right of its name. You can control this behaviour by pressing the  -button at the bottom of the animation-chain list. If an animation-chain is cyclic an additional symbol will appear on the right of its name.

The id of an animation-chain can be changed by double clicking on it. An id string is restricted to a set of characters to form a valid C-identifier (no blanks or special characters a allowed).

9.2 Animation List

The animation list consists of two vertically arranged parts. The top part of the container offers navigational elements for the timeline. The bottom part shows the list of animations contained in the currently selected animation-chain.

9.2.1 Timeline Navigation








The timeline navigation offers you information of the currently selected animation's state and controls to manipulate it.

The timeline-ruler displays the time in milliseconds and the current position of the animation state with the cursor. By dragging the timeline-ruler you can horizontally scroll the window and with the mouse wheel or with the zoom-in- or zoom-out-buttons you can specify its visible range.

The timeline-cursor can also be dragged if it shows three vertical lines. You can also change the position of the cursor by clicking on a position on the timeline (Note: it is only possible to change the position of the cursor if the animation is stopped).



Fig 76 - Timeline Ruler

-  This button activates the recording mode. When moving the connected object in the workspace-window each state is recorded as an AnimationStdGUIObject with linear easing according to the set recording-interval (see 3.2.6.1 Global Settings).
-  Play all animations in real time
-  Pause play- or record-mode
-  Stop record or play mode and jump to the beginning
-  Jump to the beginning
-  Zoom in on the middle of the Timeline area respectively reduce the timeline navigation-range
-  Zoom out from the middle of the Timeline View area respectively increase the timeline Navigation-range

9.2.2 Animation Control

Each entry in the Animation List represents an animation in your project. If the list is too long you can navigate through it using your mouse scroll wheel or the vertical scrollbar.

9.2.2.1 Select entries

You can select an animation by clicking on it.

By using CTRL-click you can select multiple entries. A SHIFT-click will select a range from the currently selected entry to the one which was clicked.

9.2.2.2 Change order of entries

If you click on an entry and hold the mouse button you are able to drag the selected entries to a different position. By releasing the mouse button the entry is dropped to the position that is indicated by a white shadow.

9.2.2.3 Start- and Target-State

You can store the state of an object in the animation as its start- or target-values. Make sure that the animation is connected to an object and that the object has a state that you want to store.



Accept the current state (position, dimensions, etc.) of the animated object as the **start** values of the animation.



Accept the current state (position, dimensions, etc.) of the animated object as the **target** values of the animation.

9.2.2.4 Link Object to Animation

An animation has to be linked to an object in order to change the object's state during the animation. To modify this connection you have several options.

1. Connection control (at the bottom of the list)



Connect the currently selected animation(s) to the object-id which is shown in the drop-down-list on the left. This object-id is changed by selecting objects in the workspace-window.



Disconnect the selected animation(s) from the object, i.e. connect the selected animation(s) to the "NO_HANDLE" Object

2. Using Drop-down menu at the bottom of the window: directly connects the currently selected object-id to the selected animation(s).

3. Using the attribute “AnimatedObject” of the animation in the Attribute-Window. This only affects the sole selected animation.

9.2.2.5 Snap to Grid



Toggle timeline snapping. If this mode is active the moving or resizing of animation-bars will only be done in steps indicated by the division-lines in the timeline ruler, thus snapping to a specific time-base (e.g. 100 ms). If this mode is inactive the moving or resizing will be done arbitrarily.

9.2.3 Animation Timeline

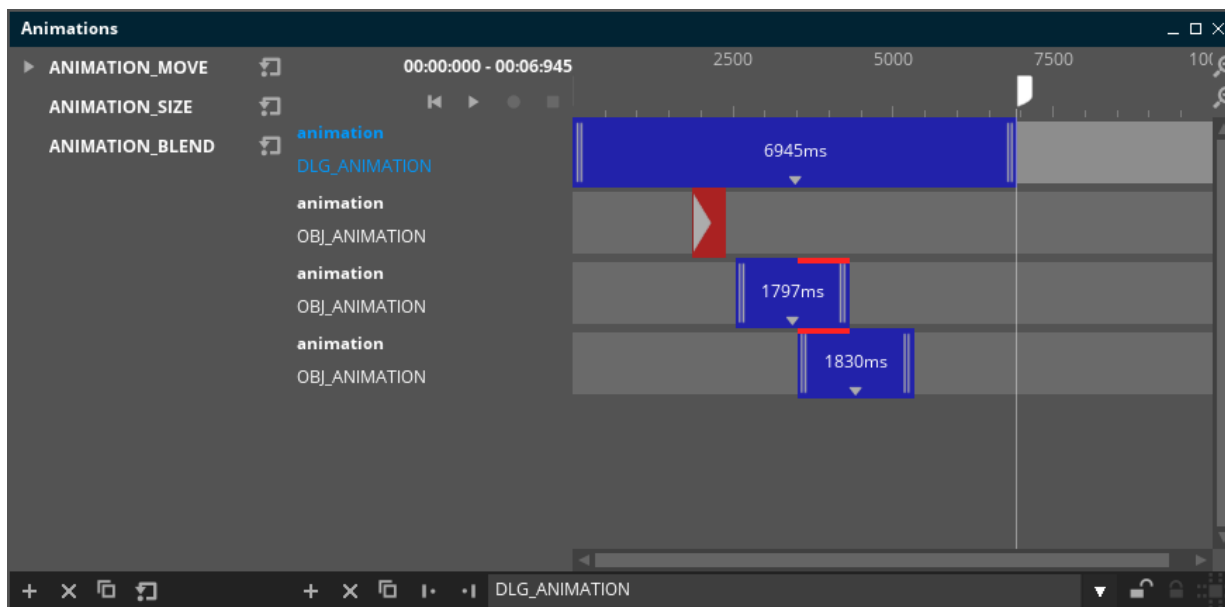


Fig 77 - Animation Bars

Inside an animation-timeline different elements can be present: A blue bar represents the time-span in which the animation is active and a red bar a trigger.

Both bars can be moved by dragging to a new location inside the timeline.

Additionally the **blue bar** can be resized using the areas on both sides marked with two vertical lines. The duration of the time span is shown in the middle in milliseconds.

9.2.3.1 Move / Resize multiple Animations

If several animations are selected, moving or resizing one will affect all selected animations with the same amount and in the same direction. If one animation reaches the size 0 during resize no further resizing in that direction will be possible.

9.2.3.2 Detailed information

Some animation-types offer additional information. You can see them by clicking on the arrow-down button at the centre of the bar (see Fig. 69). By clicking a second time on the button the information is hidden.

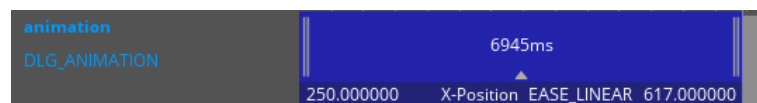


Fig 78 – Expanded Animation Bar

The drop down menu shows information about the animated attributes. Each line stands for an attribute which is modified by the animation.

You can see start value, attribute-type, easing-type and target-value.

Depending on the bar-size and the currently set zoom-factor of the timeline some information might not be visible. You can see and edit all the shown attributes inside the attribute window.

9.2.3.3 Trigger-Animations

The **red bar with a triangle inside** shows a trigger on the time-line. The position of its left side is the moment when it will be triggered. This type of animation can be used to trigger commands at a specified moment in time in synchronization with the animation (e.g. playing a sound according to a moving object).

9.2.3.4 Collisions of animations

The **vertical red lines** mark a collision of two or more animations working on the same attribute and animating the same object (i.e. having two `CGUIAnimationSize` on the same object and overlapping periods will display a collision with a red bar).

9.3 Animation Attributes

9.3.1 General Attributes

All animations have these attributes in common. If a specific type is selected more attributes will appear:

AnimationClassID	Here you can specify the type of the animation (see below)
StepTime	The duration between the iterations the animation is updated.
Duration	The length of the timespan in which the animation takes place
AnimatedObject	The object which is modified by the animation

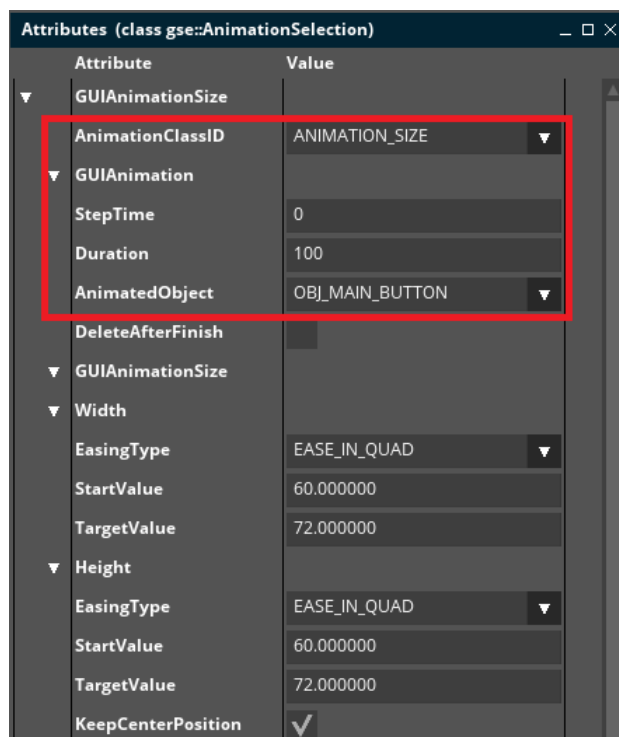


Fig 79 - AttributeView with animation attributes

9.3.2 AnimationClassID

9.3.2.1 GUIAnimationAttribute

This animation is a lightweight object. It manipulates only one attribute of the animated object. You have the choice between: **Width**, **Height**, **X Position**, **Y Position** or **Alpha**.

9.3.2.2 GUIAnimationBlinking

The blinking animation hides and shows the animated object. You can configure the duration in which the object is shown with **OnTime** and hidden with **OffTime**.

9.3.2.3 GUIAnimationMove

With this animation you can manipulate the position of the animated object in the 2d space.

9.3.2.4 GUIAnimationMoveInOut

This animation stores the position of the animated object. It animates a movement from bottom, top, left or right side of the screen to the stored position. The motion is out of screen if the **MoveOutOfScreen** attribute is active.

9.3.3 GUIAnimationSize

This animation modifies the width and the height of the object. If **KeepCenterPosition** is active, it also modifies the X- and Y-Position so that the centre position stays the same.

9.3.4 GUIAnimationStdGuiObject

This animation modifies the common attributes of an object, like **Width**, **Height**, **X**, **Y** or **Alpha**. An attribute is modified only if it is enabled.

9.3.5 GUIAnimationTrigger

This type of animation is a special case. It does not use the inherited attributes **StepTime**, **Duration** and **AnimatedObject** of the general animation instead it triggers the execution of one or more commands which have been attached to it.

10 Debugging and Trouble Shooting

This chapter contains advice on how to debug an application that is built using GSE.

10.1 The Console Window

The console window contains all the log output generated by GSE, Guiliani or its widgets.

For developers the console is very useful, since it can contain valuable information on your newly developed custom widget that simply refuses to do what it is supposed to do. It will for instance let you know if you are trying to access an illegal image resource, or if you are trying to find an ObjectID, which does not exist.

For users of GSE that are non-developers the console can still prove helpful, for example when for some reason one of the project's XML files got corrupted. In this case the console will yield information telling you which file was corrupted, what GSE expected to read, what it actually did read, and likely also the line-number within that file.

Overall, it is recommended to have a look at the console whenever something goes wrong and you need more details on the cause of the problem.

NOTE: The GSE writes the same information into its .log file.

10.2 Dealing with corrupted XML Files

If you are opening a project and a MessageBox pops up, telling you that a file is corrupted, then this is usually caused by an erroneous XML-File, or by a version conflict. In such a case, open the console window to find out more about the cause of the issue.

Usually the output will point you to the file causing the problem, and likely even to the precise line number.

Have a look at the following example log output:

```
ERROR: GUICompositeObject::ReadFromStream: Incompatible class version! Current version is 3, read version is 42.
ERROR: CGUIFactoryManager::LoadDialogFromFile: Caught streaming control exception. Problem encountered while
streaming file Unknown, user file (Line 6)
ERROR: DialogManager::CreateDialogViaFactory: Dialog
C:\Projects\GUILIANI_Workspace\apps\GSE_clean\examples\EditorDemo\240x320\Main.xml is corrupt.
ERROR: DialogManager::CreateDialogViaFactory: Caught an exception while loading dialog from file.
```

You can see that the problem was a version conflict. The first line tells us that the file contained GUICompositeObject of version 42, while the version of Giuliani/GSE works with a GUICompositeObject of version 3. The next line points us the line-number within the xml file where the problem was encountered (Line 6). The third line finally gives the name of the corrupted file, which is "Main.xml" in this case.

Now that you know the root of the problem, you can either fix it manually in the XML file, or contact the person from which you received the file and ask for an update.

10.3 Debugging StreamRuntime

The executable of your StreamRuntime-Application typically resides in the subdirectory /temp of your GSE-Project's directory. How to debug it, depends heavily on your IDE. For VisualStudio we can recommend the following two approaches:

Approach 1:

1. Set your StreamRuntime application as the "StartUp Project" by right clicking on it.
2. Set the application's working directory (Project Properties->Debugging->Working Directory) to the directory which contains the project's resources (e.g. the GSE-Project's /temp directory)
3. Start Debugging

or

Approach 2:

1. Start the application directly from GSE via "File->Run Simulation..."
2. Use VisualStudio's "Tools->Attach to Process" to attach the debugger to your application (typically named StreamRuntime.exe)

11 Contacts

If you have any questions on Guiliani or the GSE, please feel free to contact:

support@guiliani.de

www.guiliani.de