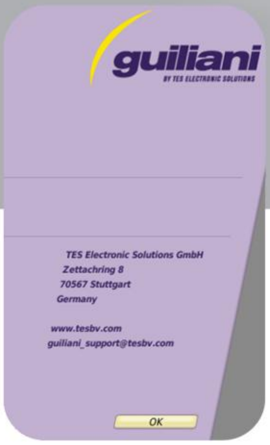
Attribute	Value
XPos	0.000000
YPos	0.000000
Width	240.000000
Height	320.000000
ObjectID	NO_HANDLE
Focussable	☐
Invisible	☐
GrayedOut	☐
Disabled	☑
OverriddenNeighbors	☐
BehaviourClassID	DUMMY_BEHAVIOUR
LayouterClassID	DUMMY_LAYOUTER

object handle	visible
NO_HANDLE	☑
MOVIE	☑

GUILIANI STREAMING EDITOR (GSE)
USER MANUAL



TES Electronic Solutions GmbH
Zettachring 8
70567 Stuttgart
Germany

www.tesbv.com
guiliani_support@tesbv.com

HOW TO...

- Start a new project
- Build a Graphical User Interface
- Run a simulation of a project
- Use standard Widgets and customize them
- Manage Images, Fonts, Sounds, Texts
- Create multiple language applications

YOUR PARTNER IN INNOVATION ™

TES™
Electronic Solutions

Contents

0. Disclaimer

TES Electronic Solutions GmbH
Hanauer Landstraße 328-330
60314 Frankfurt am Main
Germany

Phone +49 (0)69 1 53 92 12 29
Fax +49 (0)69 1 53 92 12 15

support@guiliani.de

www.guiliani.de

1. Introduction

Welcome 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 that was specifically designed for use on embedded hardware.

Profit from a mature software that has been used successfully for years in tens of thousands of devices already and start evaluating GUILIANI now!

1.1. What is Guiliani?

Guiliani is the abbreviation of: **G**raphical **U**ser **I**nterface, **L**ight and **I**nnovative with **Ani**mations.

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. Once an application has been developed, it can run natively on supported target platforms. When using Guiliani the usual development workflow is to design the application on a PC and target a set of embedded platforms for production release, like for example a PocketPC running WinCE or a Smartphone running Linux,.

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, ranging from cost-optimized to high-end hardware platforms.

1.2. What is Guiliani Streaming Editor (GSE)?

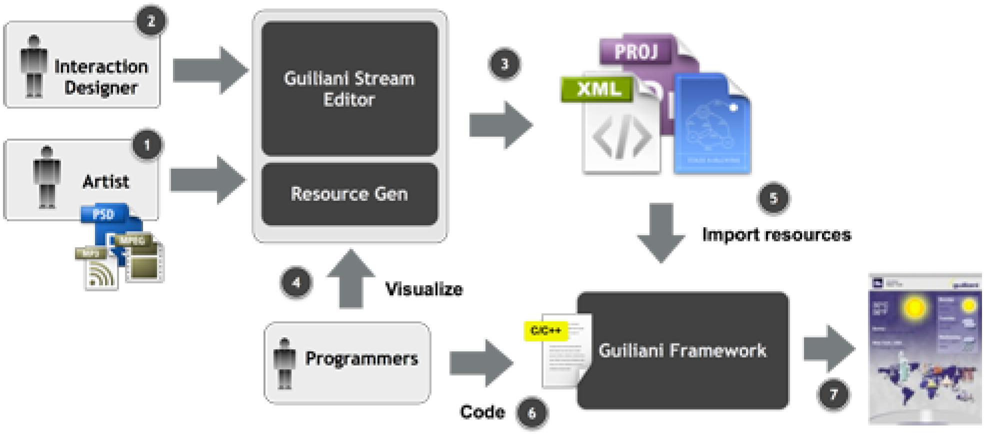
The streaming editor is a comprehensive tool for Artist, Interaction Designers and Developers to create and develop intuitive user interface with the ability to rapid prototype and deploy on embedded targets. The streaming editor is built using Guiliani and supports all features that are integrated into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

The streaming editor approaches GUI development in a very easy way. The SDK includes the editor itself with its WYSIWYG capabilities, a run-time environment to simulate the developed user interface, a resource generator for exporting the generated resources and the Guiliani C++ HMI Framework for application and event binding.

1.2.1. Main Features of Guiliani Streaming Editor

- Support of standard widgets, including property editor
- Resource Manager to manage Images, Sounds and Movies
- Language support
- Fonts
- Built-in Animations
- Object hierarchy tree browser
- Workspace management
- Export Engine
- Guiliani runtime engine for multi-platform GUI simulation
- Font support through Freetype, Bitmap Fonts or Cleartype
- Interaction editor (e.g. Switch dialogs, focus, animations)
- Text layouter
- Drag & Drop
- Export functionality
- Expandability (e.g. custom animations, custom widgets)

1.2.2. *The workflow with Guiliani*



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 XML & Resources files (.h, .lng, .png, jpeg).
4. Programmer can visualize the GUI interaction using these resources and Guiliani the 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

1.3. An introduction to the principles of Guiliani

This chapter shall give a brief introduction into 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.

1.3.1. *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 from it, such as:

- Which objects there are
- The Position and size of these objects
- Images used to visualize the GUI
- Texts to be displayed on Buttons, Textfields etc.
- CommandObjects assigned to objects
- Attributes specific to a certain object (e.g. value range of a Slider)
- 3D scenes, including 3D Objects (camera, mesh, light)
- 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 a 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. The editor’s output is in fact a set of streaming files in XML (or binary) syntax, which include all the information specified by the UI Designer during the process of building the GUI within GSE.
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. Whenever you hit “Run Simulation” within the editor, or execute your final application on the target system, it will launch the StreamRuntime, set up the Guiliani framework and read your UI designs from the streaming file(s) that were previously exported from GSE. 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.

1.3.2. Resource Management

A GUI usually consists of a variety of resources such as images, fonts and sounds which it uses to visualize itself. An 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 and 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.

1.3.3. Command Objects

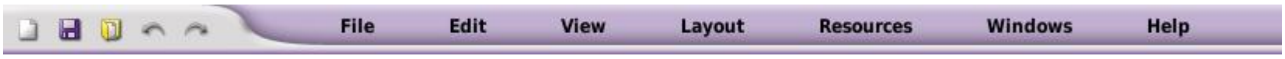
Command objects are used within GUILIANI to represent a certain behaviour in reaction to user input. For example clicking on a button, or moving a slider may execute a command object. You may see a command object as a predefined action (or chain of actions) that will be executed if a certain condition 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. 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.

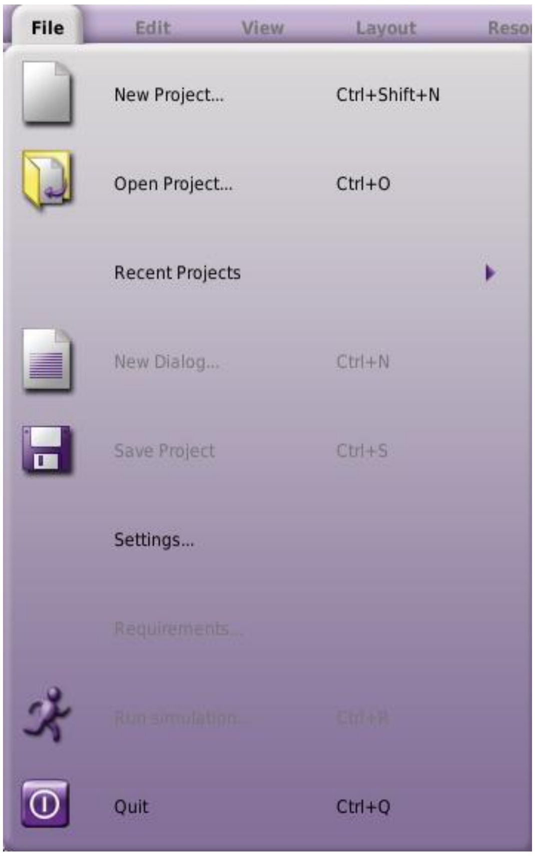


Picture 1: menu bar

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".

2.1. The File menu

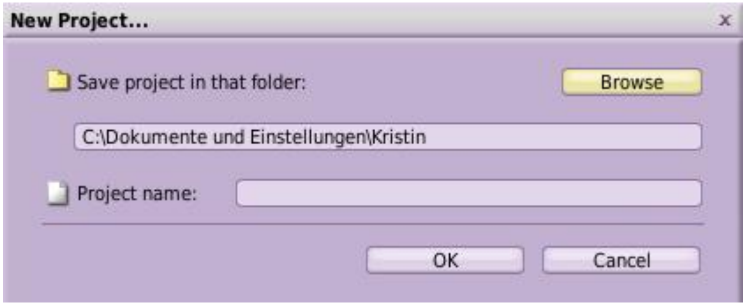
The "File" menu allows you to [create new projects](#), save existing ones, add new dialogs, handle settings, manage requirements and simulate the current GUI.



Picture 2: the file menu

2.1.1. 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 HMI Editor will create a subfolder with the name of the project in the specified directory.



Save project in that folder:

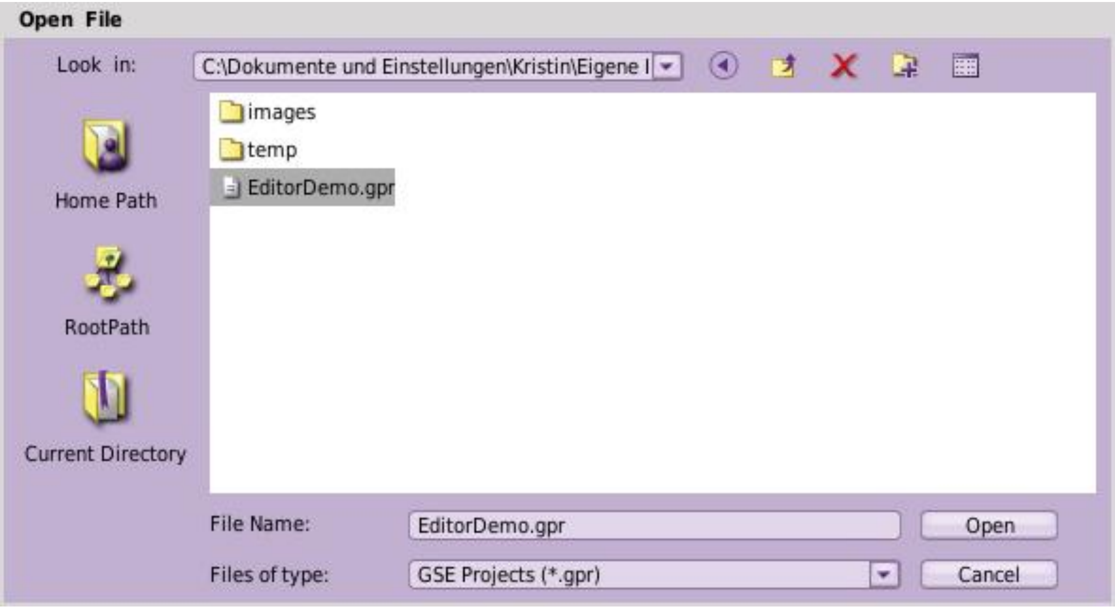
C:\Dokumente und Einstellungen\Kristin

Project name:

Picture 3: new project window

2.1.2. Open Project

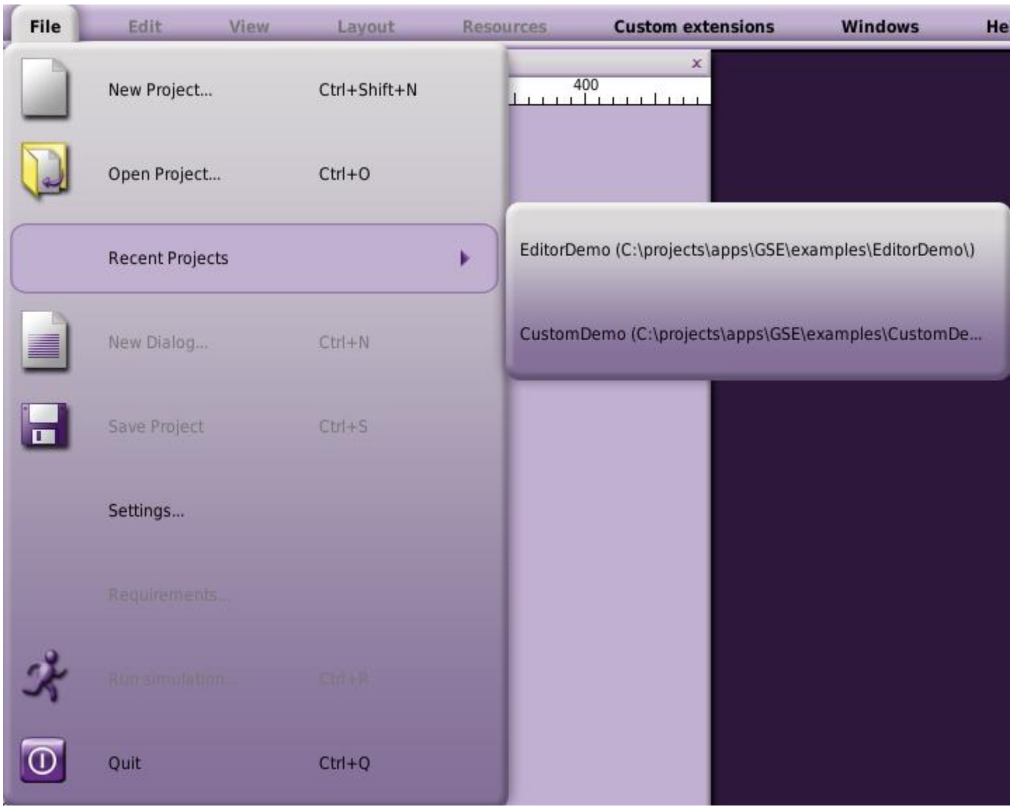
Use this option to open an already existing project. Select a Guiliani project file (".gpr") from disk. The corresponding project, including dialogs, resources, languages etc. will be loaded.



Picture 4: open Guiliani-Project window

2.1.3. Recent Projects

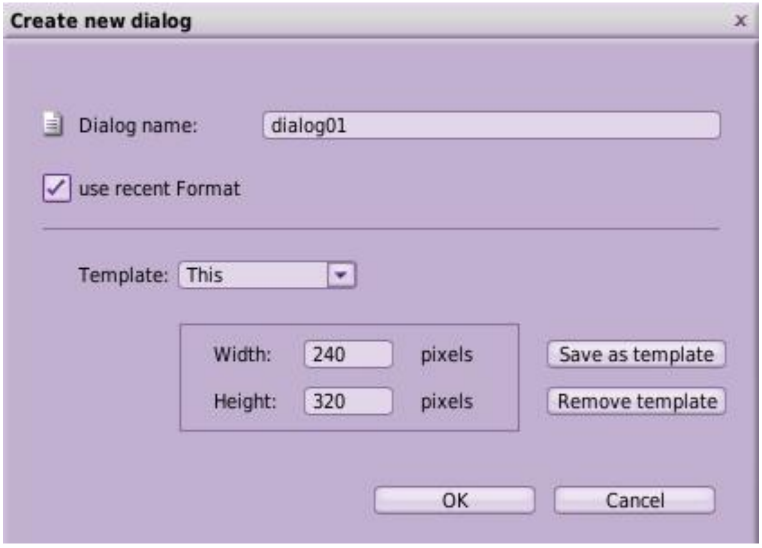
Use this option to open a previously used project. Up to 6 projects will be shown here.



Picture 5: recent projects window

2.1.4. 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.



Create new dialog

Dialog name: dialog01

☑ use recent Format

Template: This

Width: 240 pixels

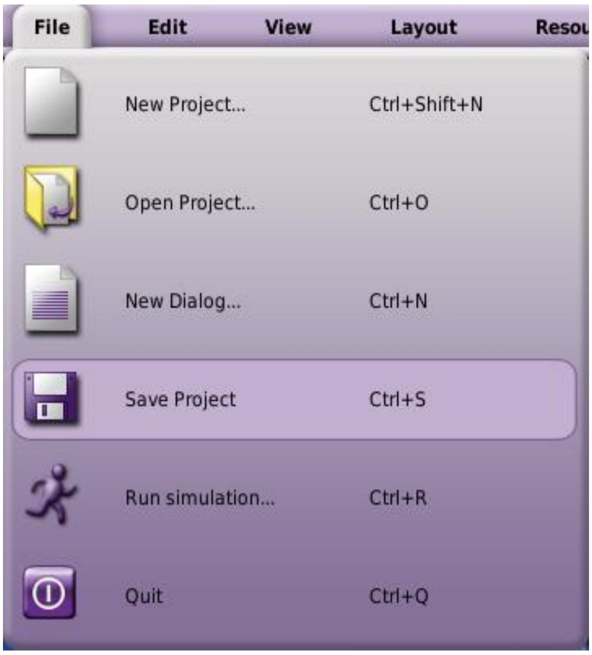
Height: 320 pixels

Picture 6: create new dialog window

2.1.5. 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:
FontProperties.xml
ImageProperties.xml
ObjectHandles.xml
OtherResourceProperties.xml
Requirements.xml
SoundProperties.xml
TextProperties.xml
[ProjectName].gpr
[ProjectName]_dialogs.xml

Additionally there are xml files for each of your dialogs and the corresponding folders:
fonts (containing all the fonts you added)
images (containing all the images you added)
save (auto-save information – for internal use only)
temp (temporary files created when you simulate your project)



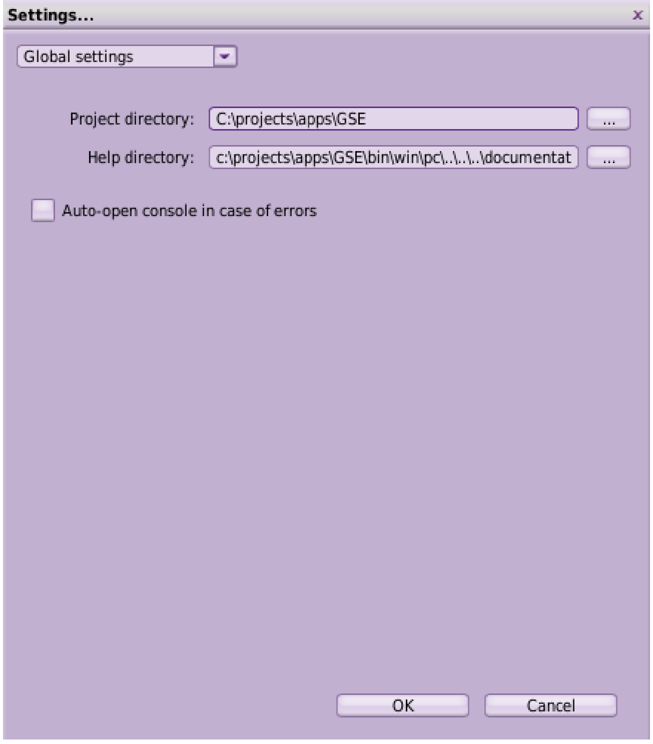
Picture 7: save project

2.1.6. Settings

You can set different values for the project you are working on and general GSE settings.

2.1.6.1. Global settings

Here you can define the directories for your projects and for the GSE help file.
Additionally you can specify whether to auto-open the console in case of errors (see Chapter 8 The console window). The console will give you more detailed information on occurring errors, by displaying any log-output that is received from GSE, GUILIANI or its widgets.



Global settings

Project directory: C:\projects\apps\GSE

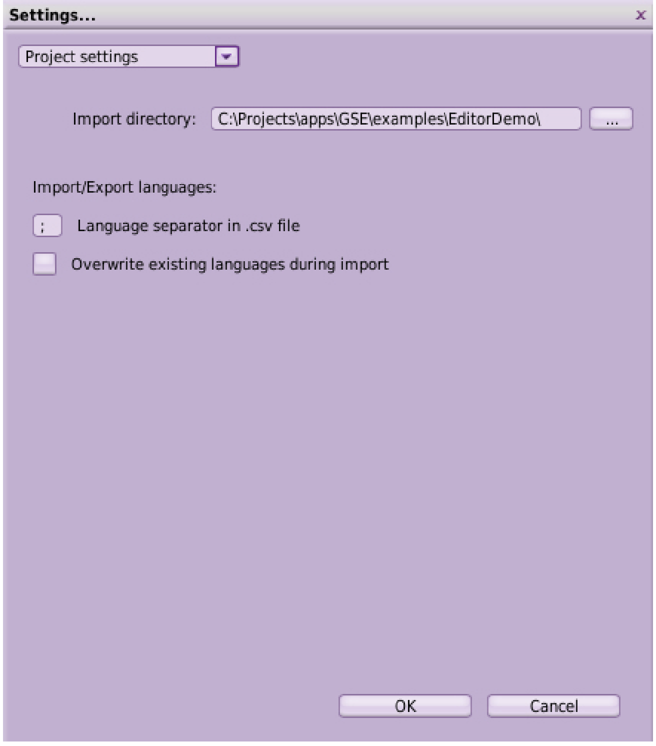
Help directory: c:\projects\apps\GSE\bin\win\pc\..\..\..\documentat

☐ Auto-open console in case of errors

Picture 8: global settings

2.1.6.2. Project settings

The project settings defines the path to dialogs which can be imported to the current project.
For the import and export of language files, the language separator of the .csv file can be adjusted (typically “;”). The Overwrite option enables the overwriting of existing language files. If overwriting is not enabled then existing languages will be overwritten by the imported languages with the same name.



Project settings

Import directory: C:\Projects\apps\GSE\examples\EditorDemo\

Import/Export languages:

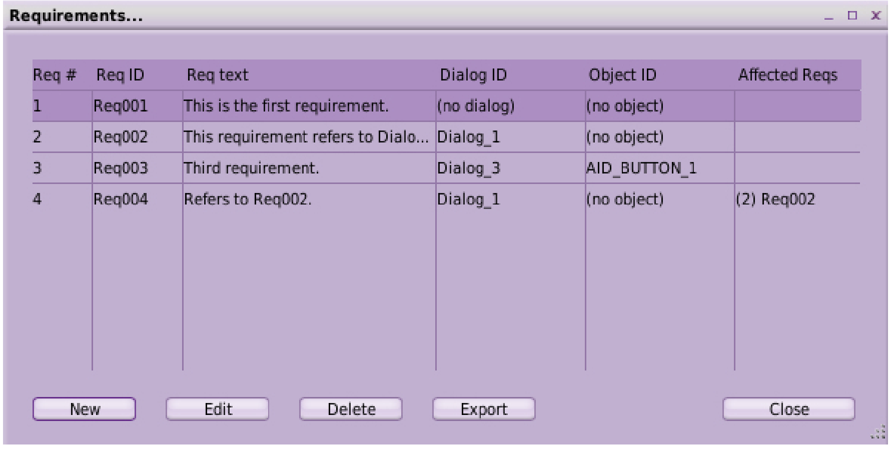
; Language separator in .csv file

☐ Overwrite existing languages during import

Picture 9: project settings

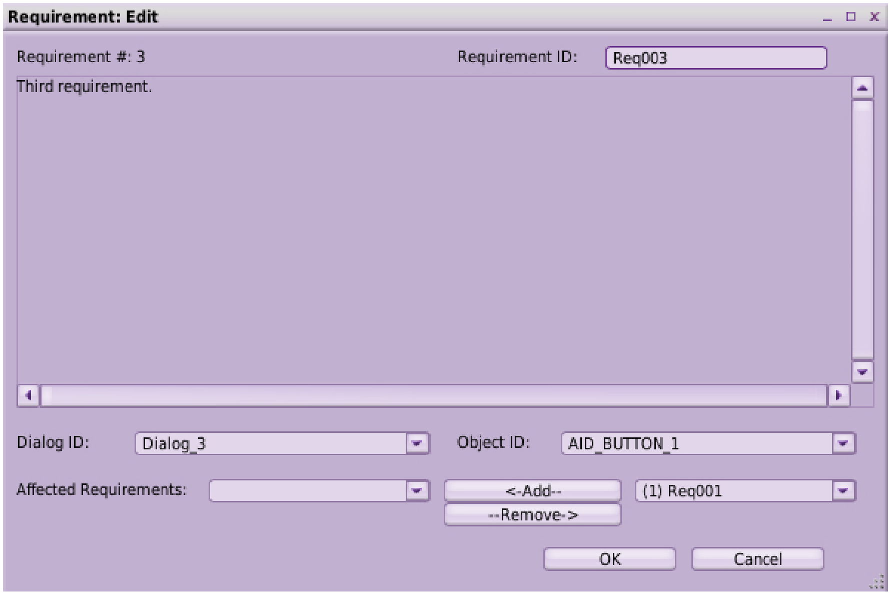
2.1.7. Requirements

The “Requirments” option 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.



Req #	Req ID	Req text	Dialog ID	Object ID	Affected Reqs
1	Req001	This is the first requirement.	(no dialog)	(no object)	
2	Req002	This requirement refers to Dialo...	Dialog_1	(no object)	
3	Req003	Third requirement.	Dialog_3	AID_BUTTON_1	
4	Req004	Refers to Req002.	Dialog_1	(no object)	(2) Req002

Picture 10: requirements window



Requirement #: 3

Requirement ID: Req003

Third requirement.

Dialog ID: Dialog_3

Object ID: AID_BUTTON_1

Affected Requirements:

(1) Req001

Picture 11: edit requirements window

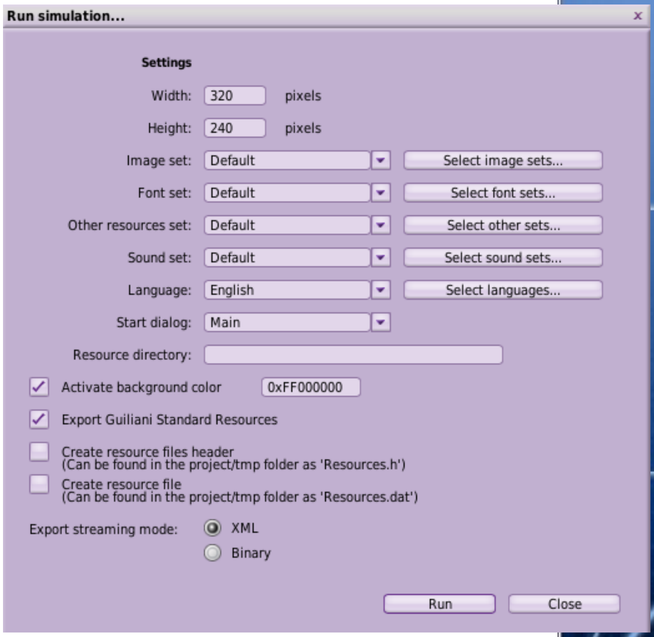
2.1.8. Run simulation

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). This application runs independently of GSE itself and offers a lifelike 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.

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 here.

In case your GUI does not have a fully opaque image as a background, you may activate a definable background color, which Guiliani will use as a background for the simulation.

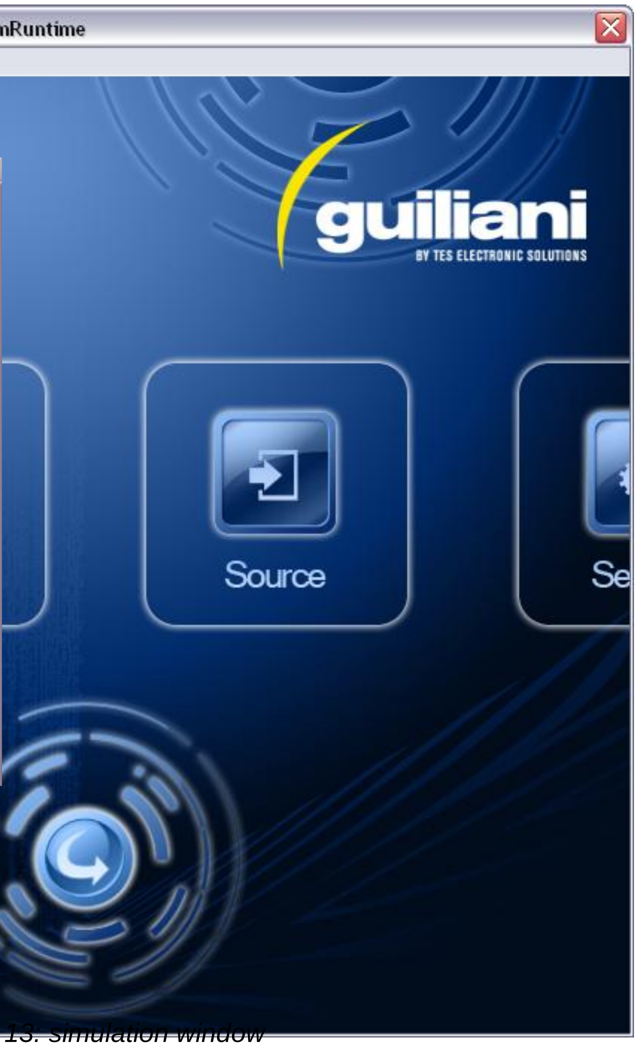


Run simulation...

Settings

Width: 320 pixels
Height: 240 pixels
Image set: Default
Font set: Default
Other resources set: Default
Sound set: Default
Language: English
Start dialog: Main
Resource directory:
☑ Activate background color 0xFF000000
☑ Export Guiliani Standard Resources
☐ Create resource files header (Can be found in the project/tmp folder as 'Resources.h')
☐ Create resource file (Can be found in the project/tmp folder as 'Resources.dat')
Export streaming mode: ● XML ○ Binary

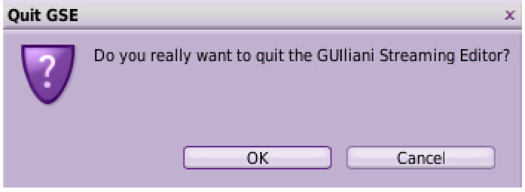
Picture 12: run simulation window



Picture 13: simulation window

2.1.9. Quit

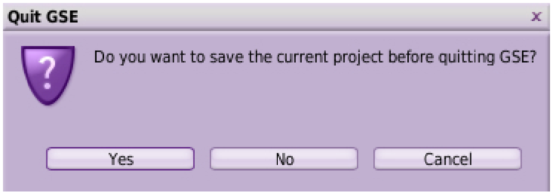
This quits the HMI Editor. If there are any un-saved changes in your GUI, you will be asked to save your project.



Quit GSE

Do you really want to quit the GUIliani Streaming Editor?

Picture 14: quit dialog



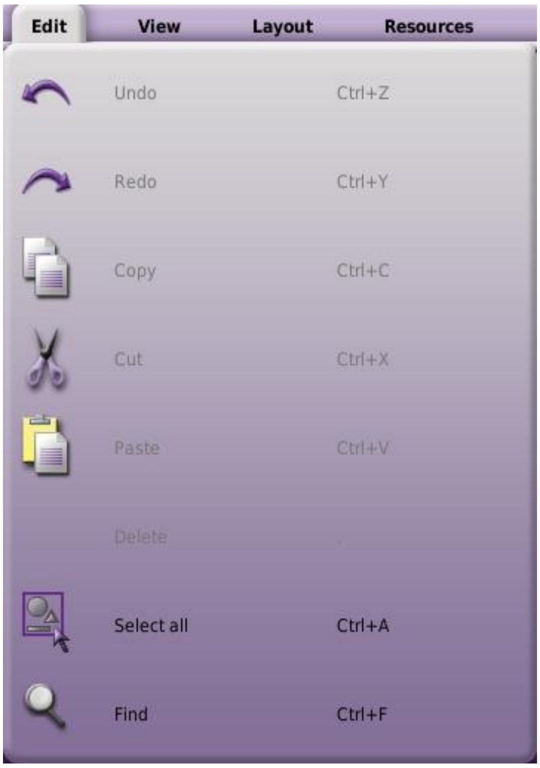
Quit GSE

Do you want to save the current project before quitting GSE?

Picture 15: quit and save dialog

2.2. The Edit menu

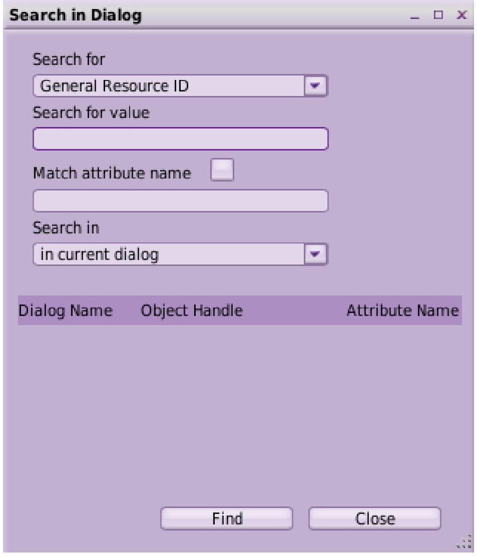
The edit menu offers you cut, copy & paste functionality. You may select one or several objects (e.g. by clicking them with the “CTRL” key pressed) and cut, copy & paste them by selecting the corresponding menu option. Note that you can also use this to copy objects between different dialogs.



Picture 16: edit menu

2.2.1. 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 ObjectID.



Search in Dialog

Search for: General Resource ID

Search for value

Match attribute name ☐

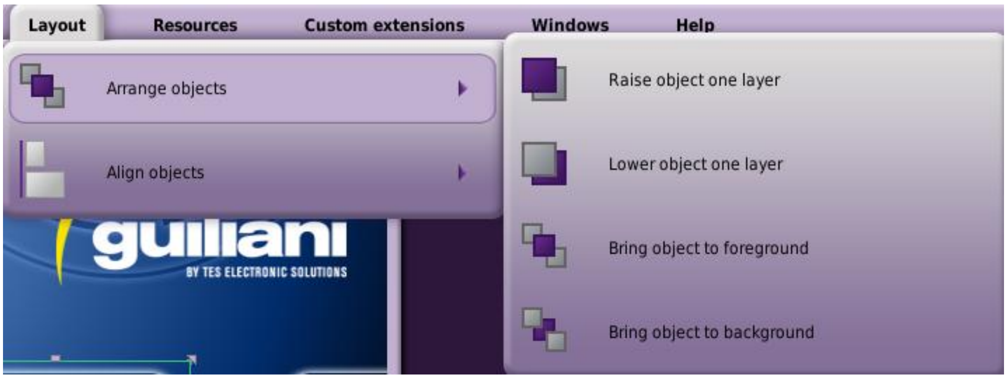
Search in: in current dialog

Dialog Name	Object Handle	Attribute Name

Picture 17: search in dialog

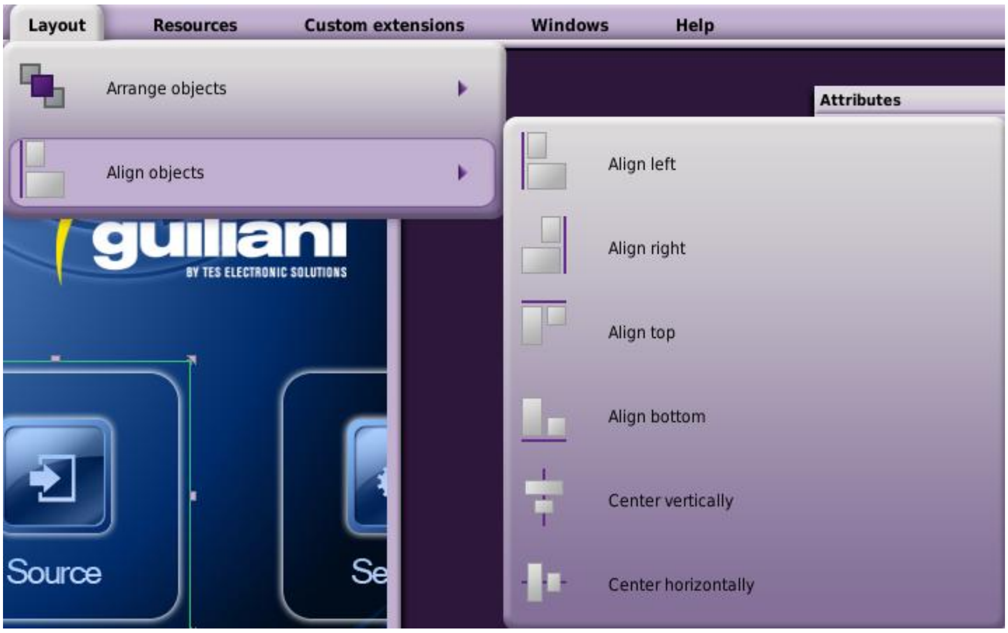
2.3. The Layout menu

The layout menu supports you to arrange GUI objects within a dialog.
“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.



Picture 18: layout menu - arrange objects

“Align objects” simplifies the arrangement of several objects in the GUI, by aligning them in one of the pre-defined manners. Note that for the center alignment options the object you have selected *first* serves as the reference point for the following objects, whereas for the edge alignment options the edge that lies farthest out in the selected direction serves as the reference. That means if you selected five objects, and then chose “Align left”, four objects will be aligned with the first left-most object’s left border.

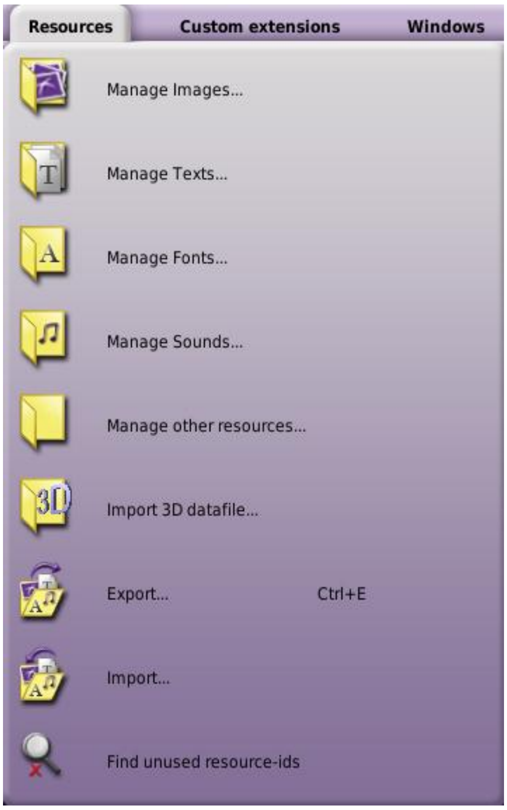


Picture 19: layout menu - align objects

2.4. The Resources menu

Use the resources menu to manage all resources of your GUI, including images, internationalized text, fonts and sounds. Each of those are handled in a dedicated sub-screen.
Here you can export your project or import dialogs, too.

NOTE: If you want to search for a certain resource, you can do it by using “Find unused resource-Ids”.



Picture 20: resources menu

2.4.1. *Manage Images*

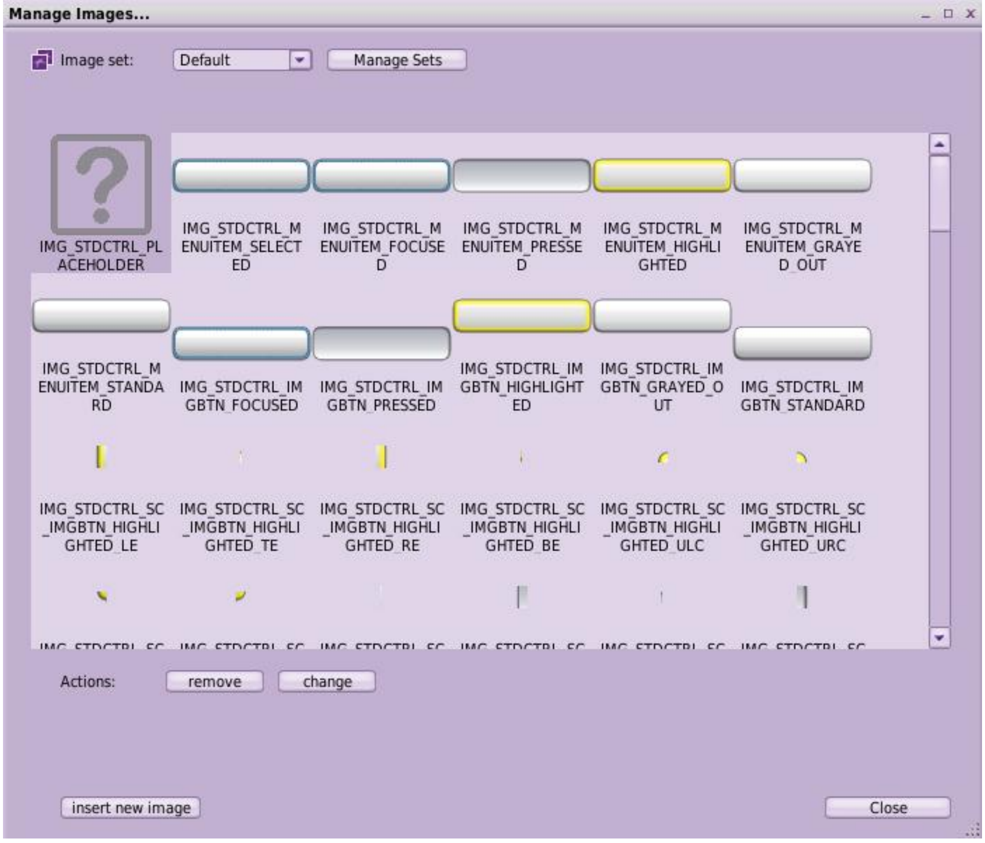
Guiliani is referencing images by an ID, which serves as a symbolic name for the given image. This has two major advantages:

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

The “Manage images” screen does allow you to import images of various formats (PNG, JPG and BMP) into the Guiliani streaming editor and to associate them 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.

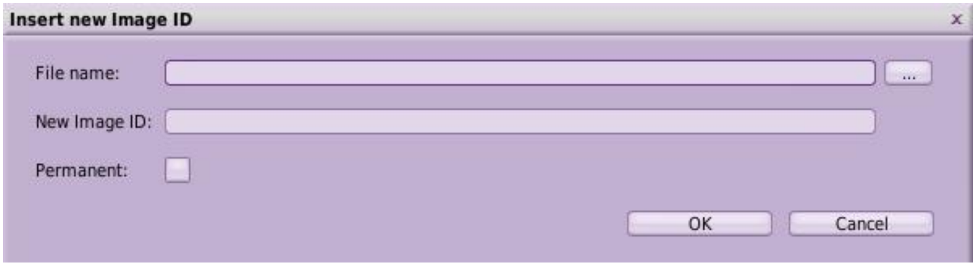
NOTE: To switch between different image sets the button “Manage Sets” (2.4.6) opens a new dialog where you can create, delete or rename them. All sets share the same IDs, but they may map different images onto them.

The “permanent” checkbox allows you to fine-tune the runtime behaviour of your GUI. Typically, images in GUILIANI will be loaded


Manage Images...
Image set:
Default
Manage Sets
IMG_STDCTRL_PLACEHOLDER
IMG_STDCTRL_MENUITEM_SELECTED
IMG_STDCTRL_MENUITEM_FOCUSED
IMG_STDCTRL_MENUITEM_PRESSED
IMG_STDCTRL_MENUITEM_HIGHLIGHTED
IMG_STDCTRL_MENUITEM_GRAYED_OUT
IMG_STDCTRL_MENUITEM_STANDARD
IMG_STDCTRL_IMGBTN_FOCUSED
IMG_STDCTRL_IMGBTN_PRESSED
IMG_STDCTRL_IMGBTN_HIGHLIGHTED
IMG_STDCTRL_IMGBTN_GRAYED_OUT
IMG_STDCTRL_IMGBTN_STANDARD
IMG_STDCTRL_SC_IMGBTN_HIGHLIGHTED_LE
IMG_STDCTRL_SC_IMGBTN_HIGHLIGHTED_TE
IMG_STDCTRL_SC_IMGBTN_HIGHLIGHTED_RE
IMG_STDCTRL_SC_IMGBTN_HIGHLIGHTED_BE
IMG_STDCTRL_SC_IMGBTN_HIGHLIGHTED_ULC
IMG_STDCTRL_SC_IMGBTN_HIGHLIGHTED_URC
Actions:
remove
change
insert new image
Close

Picture 21: manage images

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



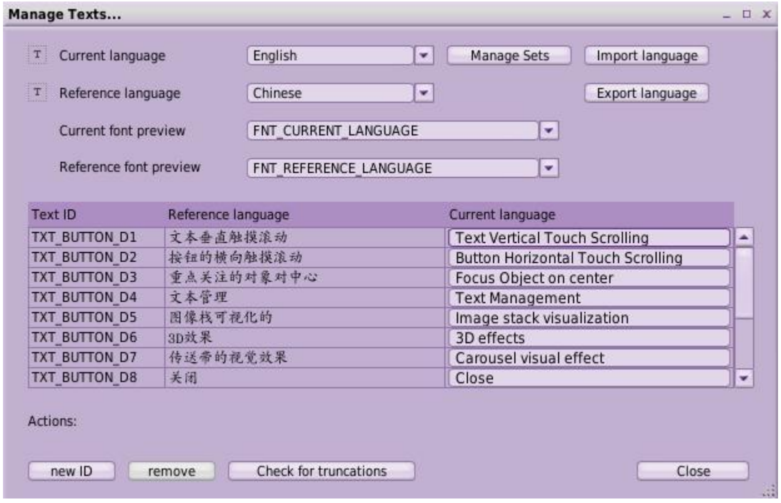
Picture 23: new image window

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

2.4.2. Manage Texts

While it is possible to use string literals within Guiliani, it is desirable to use Text IDs instead. This enables you to easily migrate from a single language to a multi-language GUI by simply translating the strings without recompilation of your application.

In the “Manage Texts…” window, you will find four dropdown boxes to select a language, a reference language, a font preview and a reference font preview. The “Language for comparison” (reference language) simplifies the translation process. The “Current language” is set visible in your GUI. The “Current font preview” provides you the opportunity to preview your texts in available font styles. The “Reference font preview” supports you to compare your texts in different font styles.



Current language: English

Reference language: Chinese

Current font preview: FNT_CURRENT_LANGUAGE

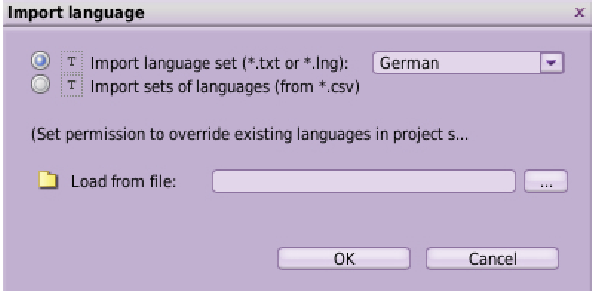
Reference font preview: FNT_REFERENCE_LANGUAGE

Text ID	Reference language	Current language
TXT_BUTTON_D1	文本垂直触摸滚动	Text Vertical Touch Scrolling
TXT_BUTTON_D2	按钮的横向触摸滚动	Button Horizontal Touch Scrolling
TXT_BUTTON_D3	重点关注的对象对中心	Focus Object on center
TXT_BUTTON_D4	文本管理	Text Management
TXT_BUTTON_D5	图像栈可视化的	Image stack visualization
TXT_BUTTON_D6	3D效果	3D effects
TXT_BUTTON_D7	传送带的视觉效果	Carousel visual effect
TXT_BUTTON_D8	关闭	Close

Picture 24: text management window

NOTE: To switch between different text sets the button “Manage Sets” (2.4.6) opens a new dialog where you can create, delete or rename them. All sets share the same IDs, but they may map different texts onto them.

NOTE: The dropdown boxes “Current font preview” and “Reference font preview” only contains fonts that were created as described in section 2.4.3.



● Import language set (*.txt or *.lng): German

○ Import sets of languages (from *.csv)

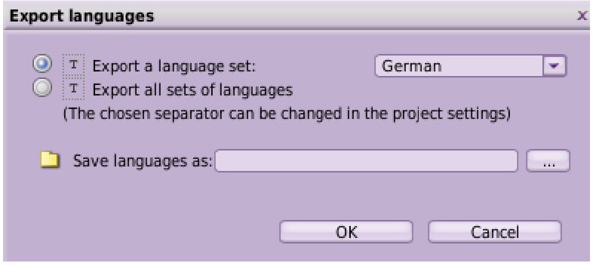
(Set permission to override existing languages in project s...

Picture 25: import languages window

Enter your Text in the text input field of the section “current language” and press Enter.

Add Text IDs by pressing the “new ID” button.
To delete, create or rename a text set, click the “Manage Sets” button.

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.



● Export a language set: German
○ Export all sets of languages
(The chosen separator can be changed in the project settings)
Save languages as:

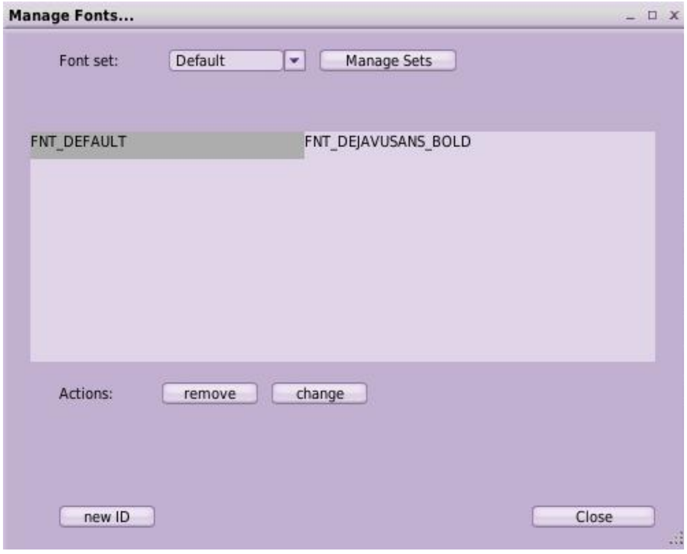
Picture 27: export languages window

2.4.3. Manage 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.

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: To switch between different font sets the button “Manage Sets” (2.4.6) opens a new dialog where you can create, delete or rename them. All sets share the same IDs, but they may map different fonts onto them.

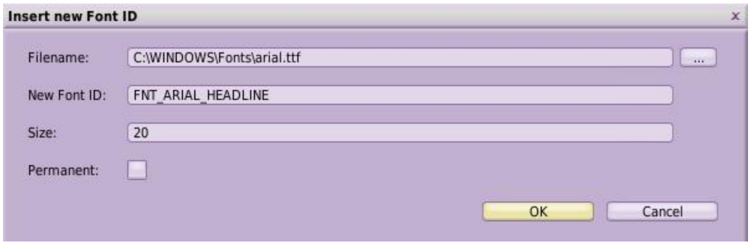


Font set: Default

FNT_DEFAULT	FNT_DEJAVUSANS_BOLD

Picture 28: manage fonts window

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).
(Please refer to [1.4.1](#) for an explanation of the “permanent” attribute)



Filename: C:\WINDOWS\Fonts\arial.ttf
New Font ID: FNT_ARIAL_HEADLINE
Size: 20
Permanent: ☐

Picture 29: insert new Font ID window

Create, rename or delete Font IDs with the “Manage Sets” button.

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.

2.4.4. Manage Sounds

Sound IDs are managed in the manage 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 has got a play button, too. Here you browse the sound file, name its ID and choose between permanent and not permanent.

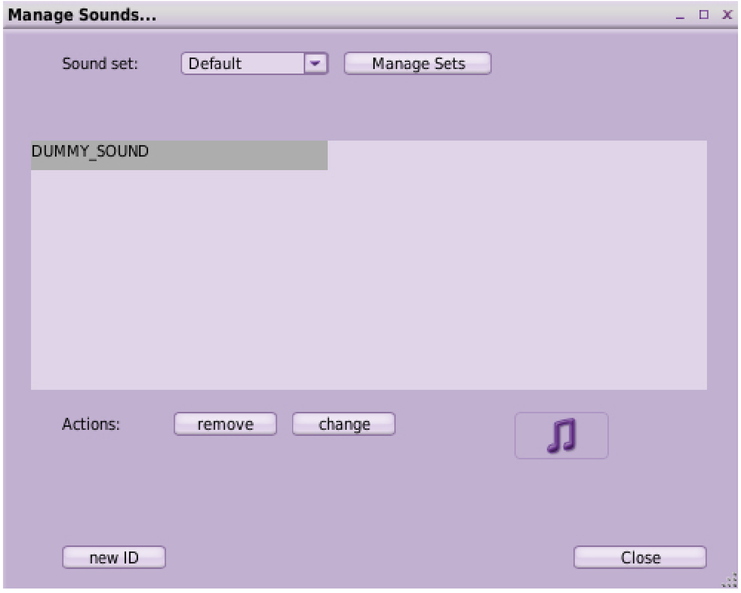
(Please refer to 1.4.1 for an explanation of the “permanent” attribute)

NOTE: To switch between different text sets the button “Manage Sets” (2.4.6) opens a new dialog where you can create, delete or rename them. All sets share the same IDs, but they may map different texts onto them.

Changing the sound scheme of your GUI is done by changing the “Sound set” in the manage sounds window.

Additionally you can create, rename and delete sound sets by clicking on the “Manage Sets” button.

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



Sound set: Default

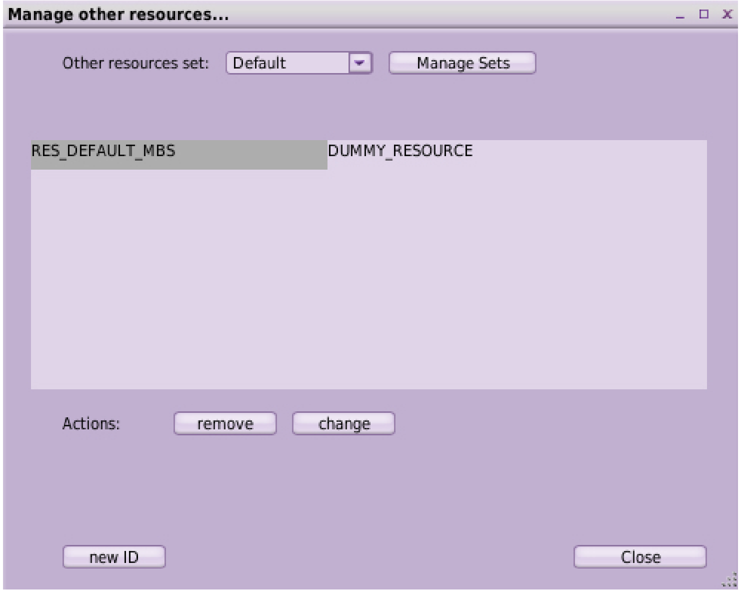
DUMMY_SOUND

Picture 30: manage sounds window

2.4.5. Manage other resources

Within the „Manage other resources“ dialog you can manage your own independent resource files, which you can handle the same way as the above mentioned resources and which you can deliver with your application workspace. Like images, fonts and sounds, they are assigned to a resource ID, but do not contain specific content.

NOTE: To switch between different resource sets the button “Manage Sets” (2.4.6) opens a new dialog where you can create, delete or rename them. All sets share the same IDs, but they may map different texts onto them.



Other resources set: Default

RES_DEFAULT_MBS DUMMY_RESOURCE

Picture 31: manage other resources window

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

- Keep track of your project's resource files in distributed development environments.
- Automatic export of other resources to the project streaming target directory.
- Application wide access to 'other 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

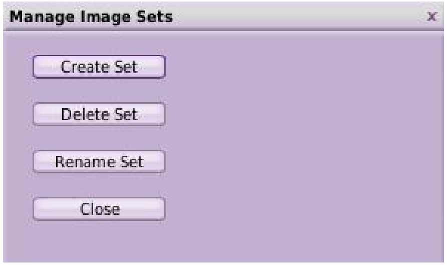
resource files of the workspace project after export.

For a new resource ID, or to change or remove an existing one, just click the corresponding button.

2.4.6. Manage Sets

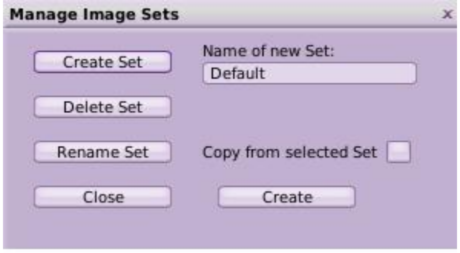
The manage sets window allows you, as its own name says, to manage the different sets your GUIliani application might use (i.e. Image, Text, Font and Other resources)

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.



Picture 32: Manage … sets

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, a check button which marking will make visible a list of possible sets to copy the information from into the newly created set, and a button named “Create”, that will create the new set once you have introduced a valid name.

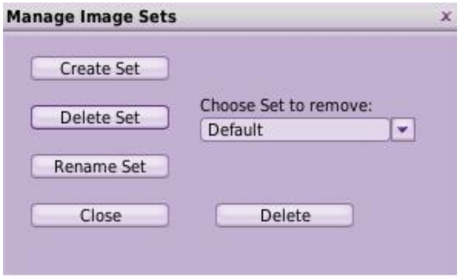


Name of new Set: Default

Copy from selected Set ☐

Picture 33: Create … set

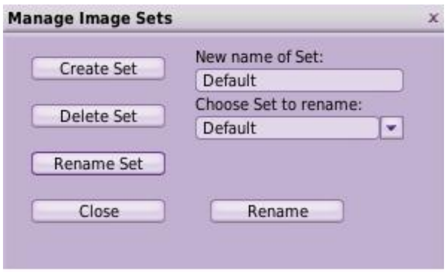
The next option of manage sets is to delete a set. That process begins by clicking the option “Delete Set”. Then, a list of existing sets will be visible, for you to choose which of them will be deleted. This deletion is produced after you click the button “Delete”.



Choose Set to remove: Default

Picture 34: Delete … set

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.



New name of Set: Default

Choose Set to rename: Default

Picture 35: Rename … set

2.4.7. *Import 3D data file*

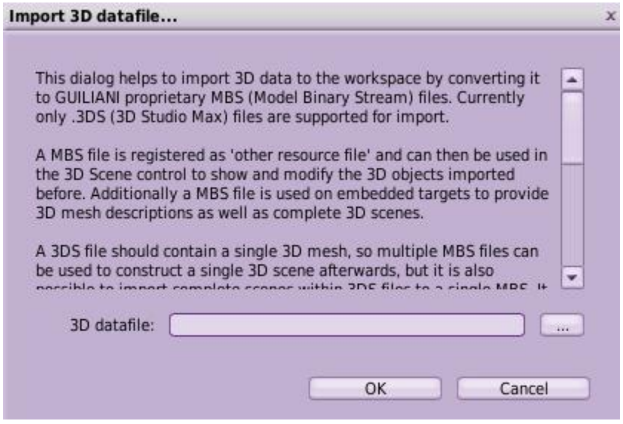
Please note: Import of 3D models is only available for targets based on OpenGL(ES) !

Import proprietary 3D description files (3Ds, Collada, etc.) to the GSE workspace using the „Import 3D datafile“ window.
The 3D scene control is made up of several 3D objects, including 3D models. A model source is needed, containing the model geometry, vertices and surface descriptions. The 3D scene uses a Guiliani proprietary format storing the 3D models, the so called Model Binary Stream (MBS). It contains all relevant information to draw a 3D model.
The MBS format is very small, optimized for performance oriented embedded systems.

Two steps are necessary to set up a 3D model, so a 3D scene object can use it.

- Convert a proprietary 3D datafile to the MBS format
  First the 'Import 3D data file' wizard requests a 3D input file. Currently only '.3Ds' files are supported by the converter. The converter reads all 3D data from the input file, passes it to a temporary (invisible) 3D scene and creates a MBS file out of it.
- Register MBS files to the workspace project
  The second dialog is up to register this generated MBS file to the workspace. Set a resource ID, which will be assigned to the registered MBS file. This ID is used by the 3D scene control to access the MBS file.

After a MBS file is registered to the workspace project, it can be assigned to and combined within several 3D scene objects.



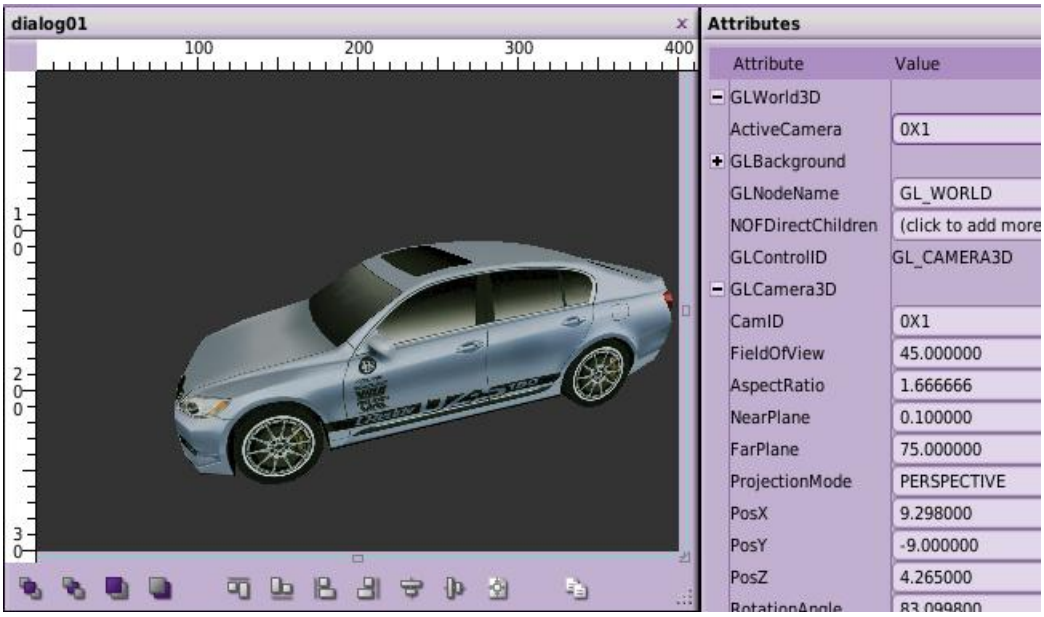
This dialog helps to import 3D data to the workspace by converting it to GUILIANI proprietary MBS (Model Binary Stream) files. Currently only .3DS (3D Studio Max) files are supported for import.

A MBS file is registered as 'other resource file' and can then be used in the 3D Scene control to show and modify the 3D objects imported before. Additionally a MBS file is used on embedded targets to provide 3D mesh descriptions as well as complete 3D scenes.

A 3DS file should contain a single 3D mesh, so multiple MBS files can be used to construct a single 3D scene afterwards, but it is also

3D datafile:

Picture 36: import 3D datafile window



Attribute	Value
GLWorld3D	
ActiveCamera	0X1
GLBackground	
GLNodeName	GL_WORLD
NOFDirectChildren	(click to add more
GLControlID	GL_CAMERA3D
GLCamera3D	
CamID	0X1
FieldOfView	45.000000
AspectRatio	1.666666
NearPlane	0.100000
FarPlane	75.000000
ProjectionMode	PERSPECTIVE
PosX	9.298000
PosY	-9.000000
PosZ	4.265000
RotationAngle	83.099800

Picture 37: example 3D scene

2.4.8. 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.

The width and height fields will be forwarded to your target system's StreamRuntime, where they will typically be used as the application's initial size.

The various set names define which resources will be loaded as defaults when starting the application. Note that you can use the "select XYZ sets…" buttons to exclude certain sets from the export, if you wish.

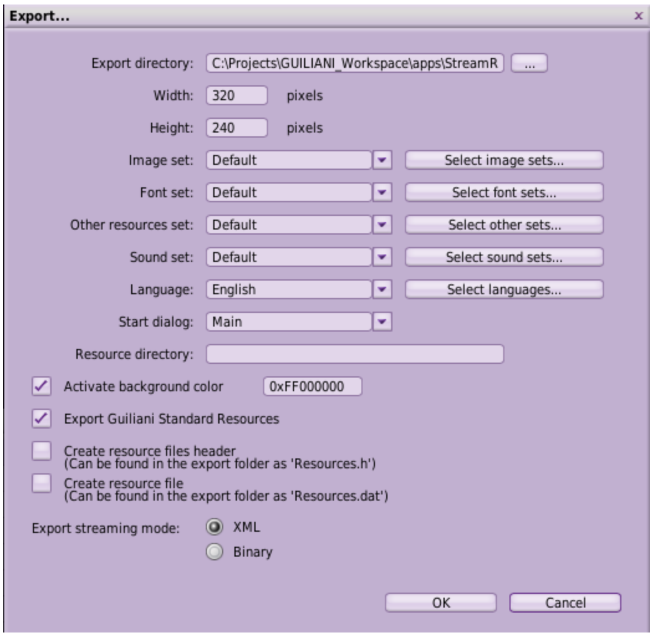
The resource directory is an optional path which will be used as a prefix when the StreamRuntime searches for resources on the target system. This is useful if the resources will not reside in their standard location right next to the executable.

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.

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.

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.

Ultimately, the "Streaming Mode" Radiobuttons let you specify whether you wish to export XML or binary streaming files.



Export directory: C:\Projects\GUILIANI_Workspace\apps\StreamR
Width: 320 pixels
Height: 240 pixels
Image set: Default
Font set: Default
Other resources set: Default
Sound set: Default
Language: English
Start dialog: Main
Resource directory:
☑ Activate background color 0xFF000000
☑ Export Guiliani Standard Resources
☐ Create resource files header
(Can be found in the export folder as 'Resources.h')
☐ Create resource file
(Can be found in the export folder as 'Resources.dat')
Export streaming mode: ● XML
○ Binary

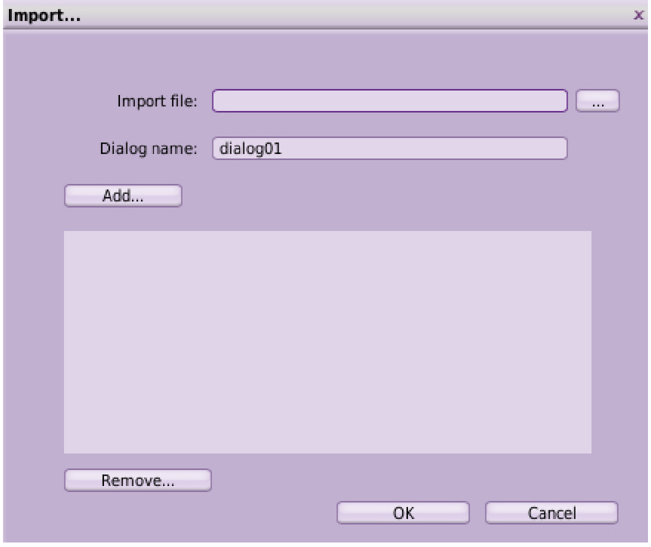
Picture 38: export window

2.4.9. Import

The import wizard is used to import Guiliani dialogs to 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. If all dialogs to import are valid, the “prepare resources'” dialog should be filled automatically. 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.

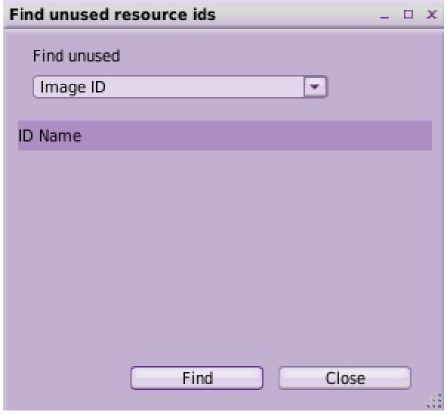


Import file:

Dialog name: dialog01

Picture 39: import window

2.4.10. *Find unused resource-Ids*



Picture 40: find unused resource ids window

A simple but very useful tool to identify resources, which are not used in any dialog of the workspace project. Use “Find unused resource ids” to keep the project clean of resource residues.
First select a resource type to inspect, using the combo box in the first line, afterwards press 'Find'. Unused resources will be presented in a table, so that they can be easily deleted by clicking the red X next to the entries.
Note: Depending on the project size the search procedure can take up to several minutes.

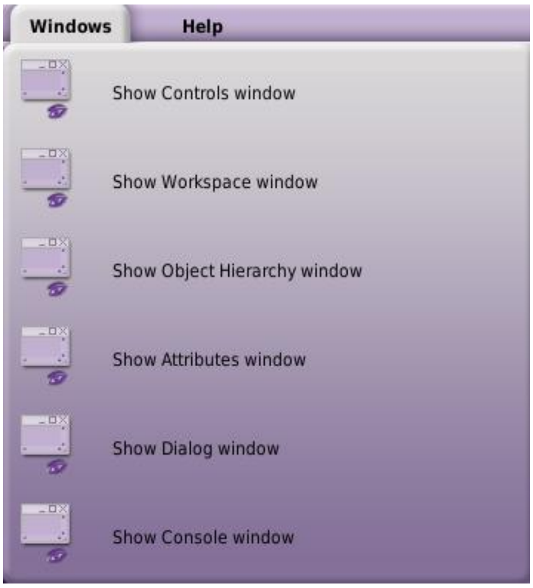
2.5. Custom extension

The custom extension dialog is a simple source code generator. It creates C++ source files out of templates and registers them as custom extension to the GSE and StreamRuntime SDK project.
A recompilation of the GSE is needed, when using this feature.

For more information and an example, please refer to the document “custom_extensions.pdf”.

2.6. 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 them, use the windows menu.

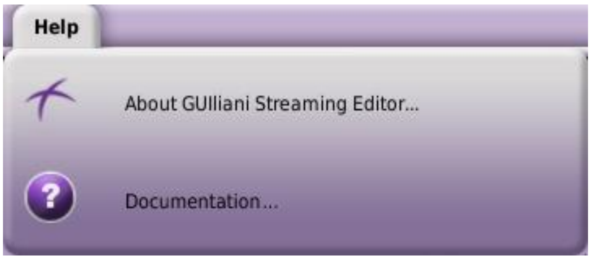


Picture 41: windows menu

2.7. The Help menu

You find all important information about the editor in the help menu.
Choose “Documentation…” to open the Guiliani Documentation containing information about the editor’s widgets. Alternatively, pressing F1 opens the documentation context-sensitively, depending on the window which you are currently in. This is particularly useful when you are looking further information on an object which you are currently editing inside the attribute view.
The “About” will open a window containing the editor version, copyright and contact information.

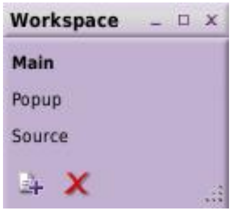
NOTE: *In some rare cases, the opening of the Help File is leading to an error message like “Navigation to the webpage was cancelled”. This is due to a Windows security mechanism and can be fixed by right-clicking on the CHM file, selecting “Properties” and clicking the “Unblock” button.*



Picture 42: help menu

3. The Workspace window

The Workspace window lists all dialogs of your project. The highlighted dialog is currently visible in the Dialog window. You can switch the dialog by selecting one of the listed dialogs. Add a new dialog with the “+” icon or delete the selected dialog with the “x” icon (at the window’s bottom).



Picture 43: Workspace window

4. The Dialog window

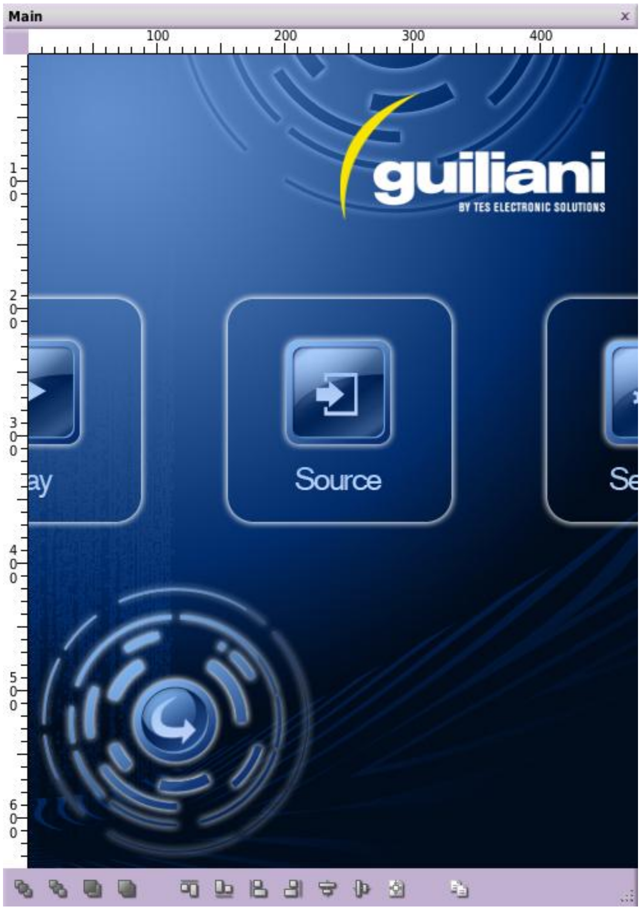
Use the Dialog window to arrange objects on the screen. Drag and drop them into groups or out of group objects or resize them (use <CTRL> to select multiple objects).

At the window’s bottom, you find shortcut icons of the edit and layout main menu:

- Arrange object’s Z order
- Align objects

The dialog’s content is docked at the window’s left corner. By expanding the Dialog window’s size, you will not influence the dialog’s size itself. But you may reveal objects which were off screen and can edit them now.

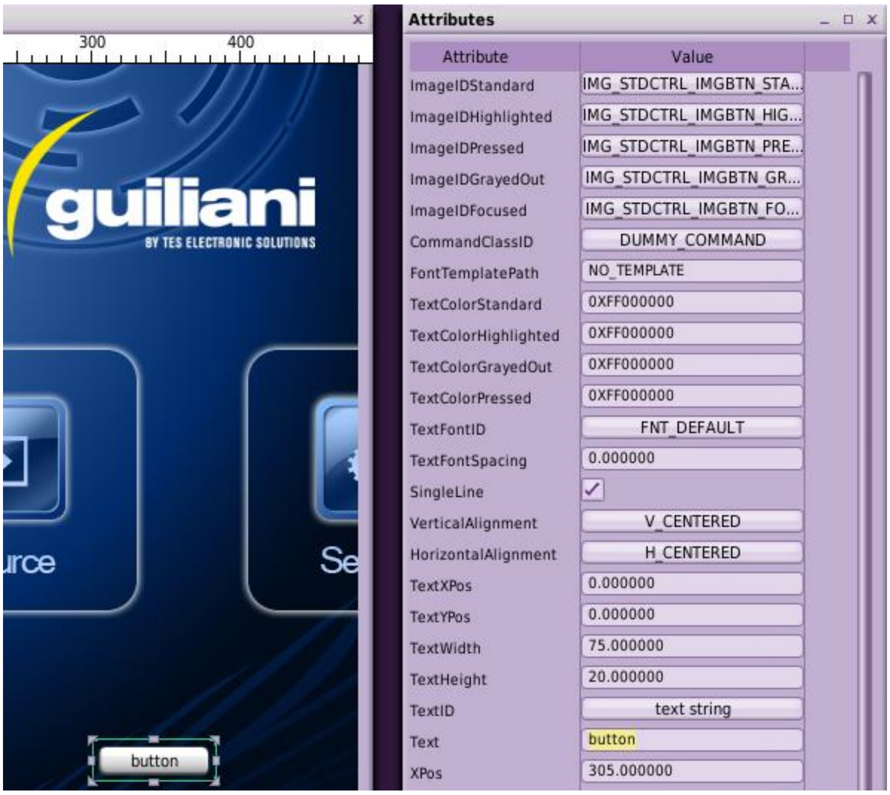
If you want to resize the dialog, select the root of the tree (first entry) in the hierarchy view and edit the width and height variables in the Attributes window.



Picture 44: Dialog window

5. The Attributes window

Edit objects in the Attributes window. 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.



Attribute	Value
ImageIDStandard	IMG_STDCTRL_IMGBTN_STA...
ImageIDHighlighted	IMG_STDCTRL_IMGBTN_HIG...
ImageIDPressed	IMG_STDCTRL_IMGBTN_PRE...
ImageIDGrayedOut	IMG_STDCTRL_IMGBTN_GR...
ImageIDFocused	IMG_STDCTRL_IMGBTN_FO...
CommandClassID	DUMMY_COMMAND
FontTemplatePath	NO_TEMPLATE
TextColorStandard	0XFF000000
TextColorHighlighted	0XFF000000
TextColorGrayedOut	0XFF000000
TextColorPressed	0XFF000000
TextFontID	FNT_DEFAULT
TextFontSpacing	0.000000
SingleLine	☑
VerticalAlignment	V_CENTERED
HorizontalAlignment	H_CENTERED
TextXPos	0.000000
TextYPos	0.000000
TextWidth	75.000000
TextHeight	20.000000
TextID	text string
Text	button
XPos	305.000000

Picture 45: Attributes window. Exp: edit specific object-texts in the text input field if you do not wish to use a Text ID.

5.1. Images

Objects may have five different states (for example the button). These include: standard, highlighted, pressed, grayed 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 the Manage Images window.

5.2. Texts

Objects with texts have a text string which can be edited in the “Text” input field of the Attributes window. (see picture below). 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 neither be repositioned nor resized. You can thus freely position a text inside it’s 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.

5.3. Fonts

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

5.4. Commands

Commands encapsulate specific actions, so they can easily be reused qnd 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. Switchin screens).
They can be directly assigned to certain widget 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 it gets clicked. 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 behaviour 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 that can be used to create a 'quit application' button. Another command that Guiliani offers by default is the *load dialog* command which loads another XML file at runtime. This can be used for creating 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 be chained to execute more than one action at once. To do this, look for the attribute 'AdditionalCmdCount' in the attributes list and click the button 'Add more' next to it.

5.5. 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 deeper information on this concept.

A powerful aid when creating your GUI will be the behaviours related to hotkey-handling and command execution, these shall be explained now.

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

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

The *Hotkeys* behaviour 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 are mapped to object IDs: When a defined key press is detected, the mapped object is searched and, if it was found, is '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.

5.6. Basic object specific attributes

5.6.1. 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. Mind that Guiliani can position objects sub-pixel accurate, so X or YPos inputs like 0.5 are possible.

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 – look for the object that is one level up in the hierarchy view).

5.6.2. 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. .

5.6.3. Focusable, grayed out, 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 the keyboard / cursor-keys.

A grayed 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 also not react to any events.

5.6.4. BehaviourClassID

Choose a 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.

5.6.5. 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 layouters, for example for automatically arranging child objects in a list container. Please refer to the GUILIANI documentation for details.

5.6.6. *Object ID*

Object IDs are symbolic names. Assign an object ID to your objects to find it 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 behaviours in order to reference specific objects inside the UI.

6. The Object Hierarchy window

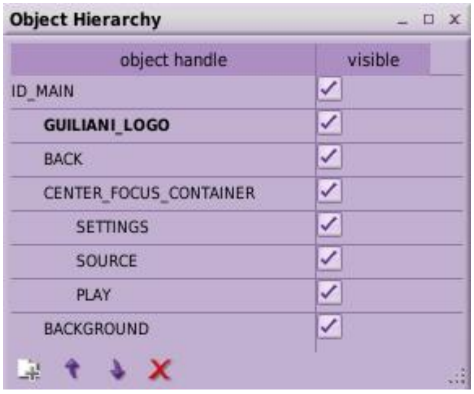
The Object Hierarchy window displays all GUI objects contained within the selected dialog including their hierarchical structure. Objects that cannot be seen on the Dialog window because they are below other GUI elements can be selected in the Object Hierarchy window instead.

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 on the Dialog window.

Delete objects by clicking on the delete icon.

You cannot change names (object IDs) of GUI-objects and group names in this window, but in the [Attributes window](#).

Declutter the workbench by temporarily hiding GUI objects. Do this by clicking on the visibility check boxes.

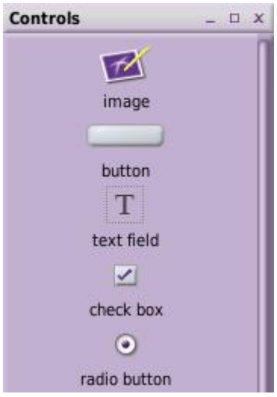


object handle	visible
ID_MAIN	☑
GUILIANI_LOGO	☑
BACK	☑
CENTER_FOCUS_CONTAINER	☑
SETTINGS	☑
SOURCE	☑
PLAY	☑
BACKGROUND	☑

Picture 46: Object hierarchy window

7. 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 Dialog window.



Picture 47: Controls window

7.1. Image

After clicking on the image icon in the Controls window, a dummy image will appear in the left corner of your GUI. Now you can edit your new image in the Attributes window.

The Guiliani Standard Image has one Image ID which by default is a placeholder image.

By selecting an Image ID in the Attributes window, the imaged is stretched to the size of the widget. If you do not wish this to happen, deselect the Check box StretchBlit in the Attributes Window. This leads to the image being drawn with its original size and being centered in the widget rectangle.

The Alpha Channel influences the transparency of your image. The value 0 means that this image is invisible, 255 means it is fully visible.

7.2. Button

Buttons have five Image IDs, one for each state (standard, grayed out, highlighted, focused, and pressed).

A button is an ‘active’ widget that can be focused and clicked by default. A command can be attached to it to provide a functionality.

7.3. Text field

Use text fields for displaying texts – both internationalized or hard-coded - inside your GUI.

7.4. Check box and radio button

Check boxes and radio buttons have ten Image IDs for the different states standard, grayed out, highlighted, focused and pressed, each in selected and unselected status.

7.5. Elements button

A scaled standard button looks distorted. An elements button can be scaled in all directions without losing its look. One status of an elements button consists of nine images: Four corners, four edges and an image for the center. Like the standard button, the elements button has five states (standard, grayed out, highlighted, focused and pressed). This means you will need 45 images for drawing an elements button.

7.6. 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).

7.7. Horizontal and vertical Slider

A slider has these images: A background image and a knob which can be normal, highlighted and pressed.

Change the slider's orientation from horizontal to vertical or vice versa by selecting the appropriate slider SLD_HORIZONTAL or SLD_VERTICAL.

In the BackgroundMargin input field you can shorten the length of the background image without influencing the span in which the knob can be moved.

By default, the knob can be moved in the span which is as long as the background image. If you would like to reduce the span-size, edit the width input field of the horizontal slider or the height input field of the vertical slider.

To change the sliders starting point (from left to right or from up to down), switch from BASE_AT_MINIMUM to BASE_AT_MAXIMUM.

Edit the value range in the MinValue and MaxValue input field and influence the StepSize as desired. You may choose 100 as a MaxValue and StepSize 10 for example if you want to display 100 percent and the slider showing every 10 percent of it.

Value represents the current value within the defined range. It does thus also define the position of the knob within the slider.

7.8. Progress bar

The progress bar is quite similar to the slider's attributes. It has less images, one for the background, the other one for the bar itself.

The bar can be positioned relative to the left corner of the background image (edit the BarX and BarY value). The width and height of the bar image can be changed, too (BarWidth and BarHeight).

The progress bar has two ProgressBarTypes.
Choose one of the two Loop modes.

Edit the value range in the MinValue and MaxValue input field and influence the StepSize just as you wish. You may choose 100 as a MaxValue and StepSize 10 for example, if you want to display 100 percent and the progress bar showing every 10 percent of it.

The ProgressBar can show a timed animation. The TimeSlice attribute influences the delay between animation steps. It defaults to 300 milliseconds.

You can change the fill direction to right-to-left with the ReverseFill check box. Change it to bottom-top or vice versa with the DrawVertical check box.

7.9. Input field

The input field consists of an editable text and a background object, the parent of the text. The background object has standard attributes such as Position or height only. To edit the attributes of the text input, select it in the Object Hierarchy window or in the Dialog window.

The input field can be used for entering passwords (PasswordMode).

Decide if the input may be numbers only or if all characters shall be allowed (AcceptedCharSet).

Choose colors for the text states (standard, highlighted, pressed, grayed out) and for the text background highlight (SelectionColor). Set the Font ID, the Font Spacing and the text alignment (top, left, right, bottom, center).

Note: This input only supports single line input. The SingleLine check box should always be selected.

Position the text within the background object with XPos and YPos or within its border. Do this by changing the TextXPos and TextYPos value. If the TextHeight has the same value as the Height value, the text will be positioned in the height’s middle.

7.10. 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.
Pleas 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.

7.11. 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.

7.12. Composite object

Composite objects serve as containers which you should use to group other object in. 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)

7.13. Reposition composite

A reposition composite object is a composite object which 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.

7.14. 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 which 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.

7.15. 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.

7.16. 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.

7.17. Animated image

Add as many images to the animated image icon as you wish. Choose the FrameDelay, which is the time after which the next image is displayed. Select the check box Repeat, if you want the animated image to run in a loop.

7.18. 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.)

7.19. 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.

7.20. 3D scene

The 3D Scene control consists of a collection of 3D objects, that builds a 3D world. The 3D scene is the bridge between a 3D world and the Guiliani 2D render surface.
When using the 3D scene, it creates a default world, which is a colored cube, a camera and a light object. Afterwards the scene can be modified on your needs using the attribute window. 3D objects can be added, removed and modified.
To add 3D models to the world, you need to add 3D MBS files to the workspace, which contains 3D models (see section 'Import 3D data file' for details).

7.21. 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).

8. 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.

All in all 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 same information can also be found in the .log file generated by GSE).

9. Tool tips

You can find tool tips when you rest the mouse pointer above an icon (e.g. in the dialog window). This will give you a short information about what the function is used for

10. Contacts

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

support@guiliani.de

Phone +49 (0)69 1 53 92 12 29
Fax +49 (0)69 1 53 92 12 15

www.guiliani.de

This project is supported by the Federal Ministry of Economics and Technology by a resolution of the German Parliament.