guiliani.de

GuilianiDemo Exposed

A short walkthrough to explain GuilianiDemo and how it works

Product:	Guiliani Technical Showcase (GuilianiDemo)
Release version:	2.5
Release date:	November 24, 2022

Table of contents

1 Introduction / Intended audience

This manual explains how GuilianiDemo works. It lists all screens and the actions related to that screens. And it describes how everything is done on a very high level. When reading the code-parts of this tutorial you should be able to understand the basic principles of object-oriented-programming.
Note: We are explaining only how the methods connect the GUI and the application and what parts of Guiliani are involved.

2 Helpful resources

There are basically two main resources which will help to enhance your knowledge about Guiliani and how are things done in the GuilianiDemo:
- GSE Manual (optional: GSE Control Attributes)
- Guiliani API Documentation

If you encounter any questions which are not answered in this tutorial please refer to any of these resources.

3 View the project

In order to examine the project in more detail you will need to open it in the GSE as well as the corresponding source-files in your text-editor or IDE. Please find more detailed description on how to do this in the documentation of the GSE and the used IDE.

4 Overview

GuilianiDemo has multiple screens (screen-filling dialogs) where each will demonstrate a sub-part of Guiliani's feature-rich portfolio.
Every screen has an info-button in the upper-right corner to show/hide a description of the contents of the current screen.
Sometimes the screen will have multiple buttons at the bottom of the screen. These can be used to navigate inside the screen to different sub-pages, where different features are demonstrated.

The screens are:

- Dynamics: this screen demonstrates basic dynamic visual-features of Guiliani like easing or animation as well as playing videos
- Controls: basic controls like buttons and sliders
- Advanced: more advanced controls like knobs, calendar and gauge
- Layer: demonstration of hardware-layers
- Data: this screen shows visualization of data in different controls
- Container: various container
- Text: demonstration of text-input via onscreen-keyboard and text-rendering (plain, rich) and dynamic resizing.
- Settings: set the language, the skin and other parameters of the demo
- Scratchpad: here you can place your controls and try out the different things Guiliani is offering

5 General Explanations

5.1 Parts of a Guiliani-Application

Every application using Guiliani will consist of the visual description (properties, dialogs and resources) created in the GSE and the code using the Guiliani-API to communicate with the GUI and create dynamic behaviour. The code will include the pre-built libraries and some start up-code additionally to your business logic. The starting point for your application-code will be the file *MyGUI_SR.cpp* located in *<APP>/Source*-folder.

5.2 Dynamic behaviour

For many purposes there are built-in dynamics (***Behaviours*** and ***Commands***) you can use directly in the GSE-project without writing any line of source-code. That can be moving and/or resizing objects, changing visibility or transparency.
The most often used way in GuilianiDemo to process events from the GUI in the application-code is the ***CallAPI***-command which can execute all sorts of things directly in the GUI-thread. Guiliani will call the *CMyGUI::DoCallAPI()* method with the strings API and Parameter you have specified in the GSE.
If you are unsure what is executed when interacting with an object, just click on that object in the GSE and examine the attached dynamics in the attribute-window.
For more information about the internals of the executed ***Behaviour*** and/or ***Command*** please refer to the Guiliani API Documentation.

5.3 Creating Applications with multiple screens

When your application consists of multiple screens which can be shown depending on various actions, you can use Dialog-Transitions to move from one dialog to another. There are several settings for the visual transition made during the dialog-switch as well.

In general the following procedure is done:

- Load the new dialog, it will be invisible by default
- Transition to the new dialog using blend, crossfade, dissolve, push, …
- Delete the old dialog

To prevent data-loss it is advised to save all relevant data from the former dialog **BEFORE** the transition is started. This can easily be done by using a ***CallAPI***-command beforehand.
After the new dialog is shown there may be additional initialization needed for ***TextFields***, ***ComboBoxes***, etc. The new dialog will be loaded and shown as it was designed in the GSE. If you want to run initialization on any object in the new dialog, you can do this now by using ***CallAPI*** again.

GuilianiDemo uses three commands which are executed when switching to a new dialog. These commands are executed from the menu located in *<APP>/Source/Menu.cpp*.

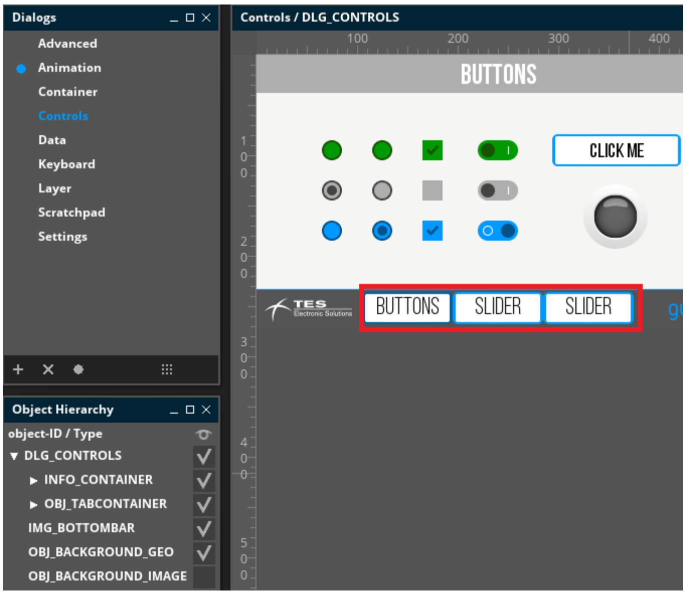
- ***CallAPI*** (“*SetTransition*” and “*<Dialog-Name>*”)
- ***DialogTransition***
- ***CallAPI*** (“*<Dialog-Name>*”)

Here the first ***CallAPI*** will set up the transition to the dialog. This can be selected in the Settings-dialog and is persisted in the application code.
After that the transition to the Animation-dialog is performed.
When the Animation-dialog is finally shown the second ***CallAPI***-command will set up all things needed to interact with the controls according to the logic of the dialog. This will be explained in more detail in the code for each dialog.

5.4 Dialogs with sub-pages

When having a dialog with more data than can be shown on the display it may be sensible to create sub-pages and partition the data into these.

For example the Control-dialog has several sub-pages (Buttons, Slider1 and Slider2) where most of the available basic control-types in Guiliani are presented. To have all these controls in one dialog will be very difficult on small screens (e.g. 480x272 pixels).



So we created sub-pages (red) containing a small part of the controls which can be dynamically shown by the buttons on the bottom of the screen. Each of these sub-pages is a ***TabItem*** inside the Client-area of a ***TabContainer***.

This construct is used in most of the dialogs.

5.5 Dialogs and their source-files

In GuilianiDemo each dialog, except for the Main-dialog and the ScratchPad, has a separate source-file which handles all necessary things like initialization and interaction with the GUI. This follows the object-oriented principle of Model-View-Control, where our class is the Control-part and the GUI is the View. The Model-part is often integrated into the Control-part, but when its size is increasing a separate class might be a good idea.

The source-files are in the *<APP>/Source*-folder and named after the dialog they control.
Each of these *DemoXXX.cpp*-files contains a class for the dialog. This class inherits the header-only class *DemoBase* which is an abstract class defining the methods *Init()* and *DeInit()*. This means that every derived class must implement these methods before you can create an instance of that particular class.
Also it has an empty implementation for the method *HandleCallAPI()* which will be called from the application when there are ***CallAPI***-executions for a specific dialog.

This is done in *CMyGUI::DoCallAPI()* depending on the name of the dialog given by the parameter *kAPI*.

In *CMyGUI::DoCallAPI()* the second ***CallAPI*** after the transition to the dialog has finished. It creates a new instance of the controller-class connected to the dialog-name and calls the method *DemoXXX::Init()* of that class.
Here the class will execute special actions to get specific objects (sliders, buttons, etc.) which are needed for interaction with the dialog as well as setting up more advanced things like ***Observers*** or create a timer.
The method *DemoXXX::DeInit()* may be a central place to clean up your class (release memory, delete objects, etc.) when it is going to be deleted. So call this method from the destructor.

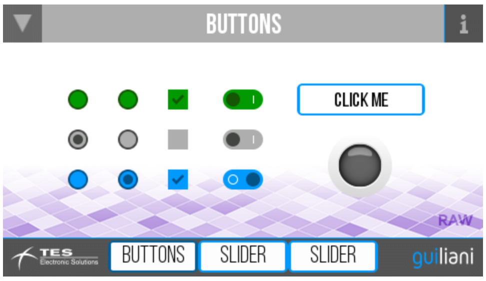
5.6 Shared functionality

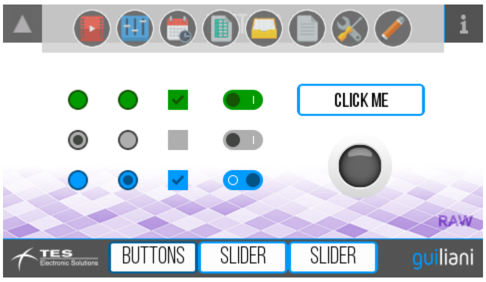
Two main-functionalities are shared between all dialogs in GuilianiDemo. So they can be used in every dialog, but implemented only in one place.

5.6.1 Menu

You will notice a small triangle pointing down in the top-middle of the screen. This is the main-menu which will expand/collapse by clicking on the button.
Inside the menu you can choose one from the available dialogs and directly jump to it.

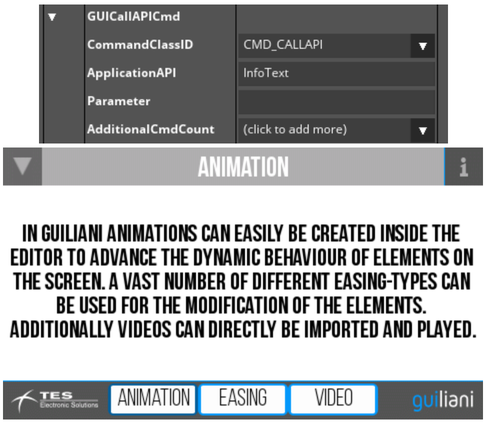
This menu is completely made in code and is located in the files *<APP>/Source/Menu.cpp* and *<APP>/Include/Menu.h*. Essentially all the menu makes is moving down or up the container when clicking on the button and executing a bunch of commands when clicking on one of the dialog-icons.





5.6.2 Info-Button

The info-button will involve some more logic as it applies several animations to the INFO_TEXT and TEXT_BACKDROP located in the container INFO_CONTAINER in each dialog.



The info-button in each dialog is attached to CallAPI with parameter "InfoText" which will call the method *CMyGUI::AnimateInfoTxt()*, where the needed objects are searched in the currently active dialog and prepared for the animation.

The first click opens the info-field and the second one closes it again.

The animations are a move- and a size-animation for the background which are started at the same time with the same duration and will inform a ***BoxObserver***, which has been registered, when the state of the animation is changed, e.g. playing, stopped, deleted.

If the move- and size-animations have finished the ***BoxObserver*** will create a ***TextAnimation*** which will alpha-blend the text from transparent to opaque.

This construct is a demonstration of how to connect different animations and how the observer for an animation works.

6 The screens explained

Now every screen will be described using the GSE-project as well as the code. First we will have a look at how the screens look like and which parts are there. After that we will go through the code to look at the Guiliani-APIs which are used.

If any function is unclear please refer to the official Guiliani API Documentation.

6.1 Main

6.1.1 Code

Source-File: *<APP>/Source/MyGUI_SR.cpp*

When starting the application most of the code is used to set up things needed for the application (*CMyGUI::CMyGUI()*). The method *InitDialog()* is used to handle specific requests via *DoCallAPI()* and to keep track which dialog is currently shown.

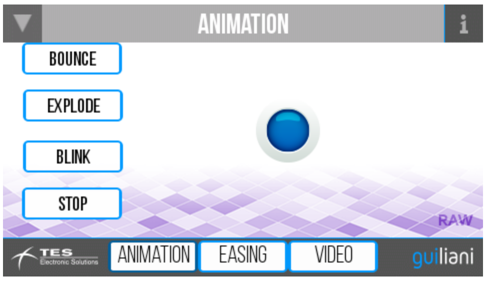
6.2 Dynamics

In this dialog there are three sub-screens which are shown / hidden with the button at the bottom of the screen. The sub-screens will show a simple animation, the different types of easing for dynamics and a random video-file.

6.2.1 GSE-project

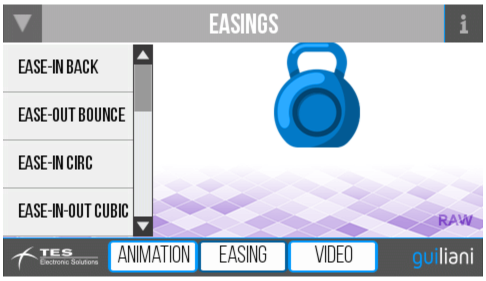
Have a look at the different sub-dialogs in this screen. Each sub-dialog has an own top-bar to display the correct headline and to show the back- and info-button.

6.2.1.1 Animation

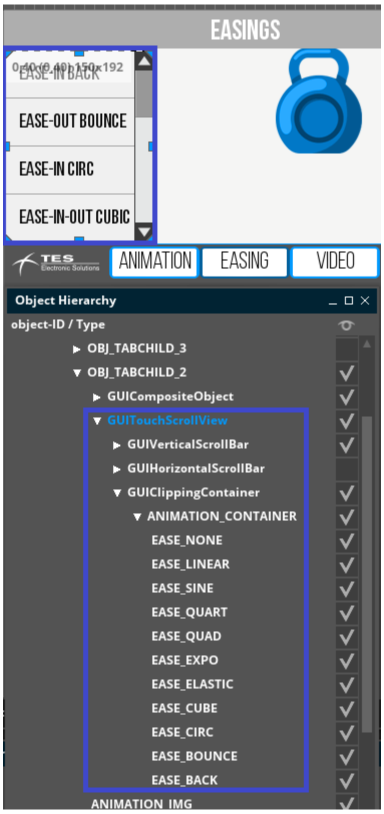


The sub-dialog Animation will show four buttons to control the animation of an image.
Each button has a command attached to start or stop a recorded animation. No special code is needed for this part to work.

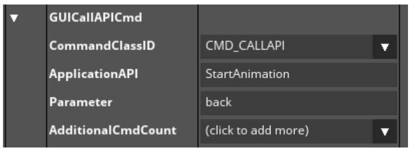
6.2.1.2 Easing



Here we have a list of the various easing-types available in Guiliani for any dynamic visuals, i.e. animations and an object which will move down towards the bottom-bar using the selected easing.

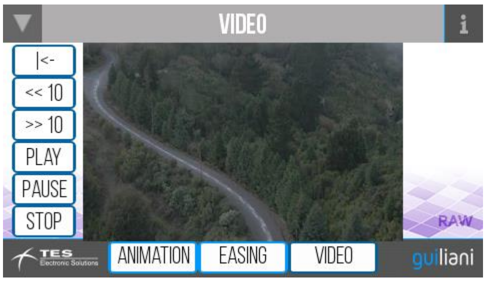


Here is the list, which is actually a ***TouchScrollView*** you can swipe with your fingers vertically to navigate to the desired list-item.



Each element in the list is a button triggering ***CallAPI*** with parameter “StartAnimation” and the type of easing to be used for the animated object.

6.2.1.3 Video



This sub-dialog shows the video-file, which is randomly picked when the dialog is loaded and additional buttons which are used to control the video, e.g. play, stop, rewind. All the buttons are attached to ***CallAPI***-commands which execute the various actions.

6.2.2 Code

Source-File: *<APP>/Source/DemoAnimation.cpp*

The method *Init()* will collect all necessary objects, select the video-file and calculate the ending position of the handle in the Easing-sub-dialog.
Additionally it will create an ***AnimationObserver*** which will react when the animations in the Easing-sub-dialog start or end to grey out the buttons in the ***TouchScrollView***.

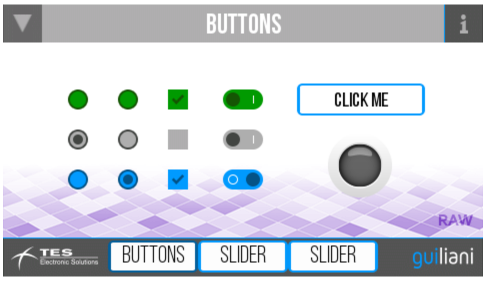
HandleCallAPI() is the heart of the controller and will receive a message every time CallAPI is triggered by a control in the GUI, i.e. when the buttons in the ***TouchScrollView*** of the Easing-sub-dialog or the buttons for the video-playback are clicked.

6.3 Controls

This screen presents most of the basic controls in Guiliani and how they can interact with the application.

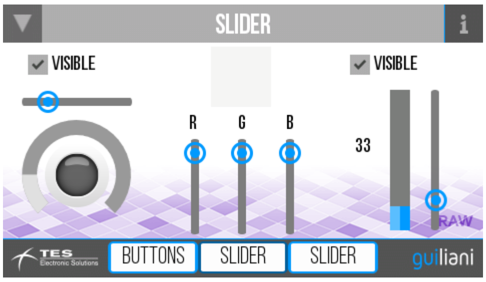
6.3.1 GSE-project

6.3.1.1 Buttons

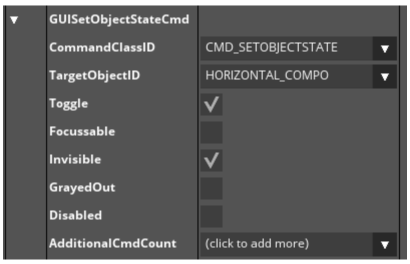


In this sub-dialog there is only one ***CallAPI*** which shows a ***MessageBox*** when clicking on the button in the right half.

6.3.1.2 Slider1



Here we have three groups of controls which demonstrate the use of Sliders.

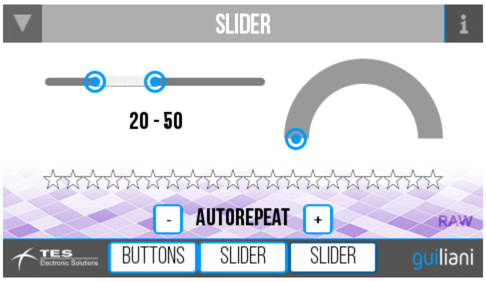


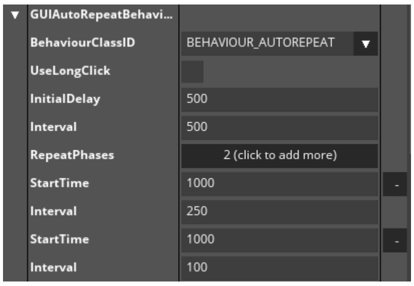
The groups on the left and on the right have a ***CheckBox*** above them which uses ***SetObjectState*** to control the visibility of the containers surrounding the controls below.
Every time the Checkbox is clicked the ***Behaviour*** will toggle the Invisible-state of the container.
Note: If a ***CompositeObject*** is invisible all contained children are invisible as well.

The group on the right, consisting of a ***TextField***, a ***ProgressBar*** and a ***Slider***, has a connection via a ***DataPool***-entry.
That means that when the value of one of these objects changes all other objects which are observing this ***DataPool***-entry.
To open the dialog and see which DataPool-entries are currently defined in the project and which objects are observing these DataPool-entries use “Resources → Manage → DataPool”.

6.3.1.3 Slider2

In this sub-dialog there are CallAPI for the buttons with “-“ and “+” to decrease and increase the value of the segment-bar. Also there is an ***AutoRepeat***-Behaviour attached to these two buttons. The rest, e.g. setting up the observers for the various controls is done in the code.





The ***AutoRepeat***-behaviour can be used to execute a command attached to the control several times in a specific period.

6.3.2 Code

Source-file: *<APP>/Source/DemoControls.cpp*.

In the method *Init()* all interesting objects are searched.

6.3.2.1 Dynamic DataPool-Entries

The ***RadialSlider*** and the ***SegmentBar*** have been connected to a dynamic DataPool-entry during application-start up (see *MyGUI_SR.cpp*). The value of the ***SegmentBar*** is changed via the method *CMyGUI::DoCallAPI()*, which is called by the buttons with “-“ and “+”.

6.3.2.2 Observing objects

The first paragraph will get the objects in the left group, where the ***Slider*** is horizontally aligned, and create a class *ObsvProgressImg*. This class is derived from ***CGUIObserver*** and will be informed of all changes of an object.
In its constructor the class will register itself as a value-observer for the ***Slider*** it has received as one of its parameters.

If an observer should be able to react on a change of the observed object is needs to implement the method *OnNotification()* which has the following parameters:

- current value of the observed object
- the object which has been changed (in case the observer observes more than one object)
- the x- and y-coordinates if the object is connected to a ***DataPool***-entry.

In this specific *OnNotification()* the received value is set for the connected ***ProgressBar*** and changes according to the value the image which is shown below the slider.

The sliders in the middle group are connected to the ***GeometryObject*** above them via the class *ObsvGeometry*.

The right group does not need any initialization since the ***DataPool***-entry will handle all things internally.

The ***RangeSlider*** and the ***TextField*** in the third sub-dialog are a bit different. Here the value-observer for the ***RangeSlider*** is the controller itself, which means that it needs to implement also an *OnNotification()*-method to receive updates of the value. In this case both values of the ***RangeSlider*** are acquired and put as a text for the ***TextField***.

6.3.2.3 Persistence

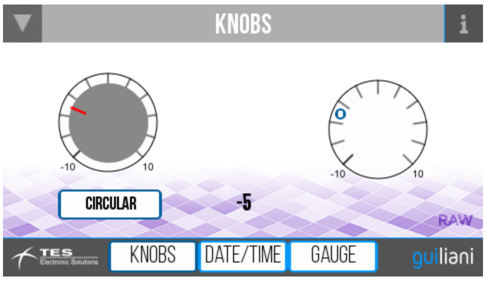
Since there is data in this dialog which needs to be persisted over the life-cycle of the dialog (namely the color of the ***GeometryObject*** set via the Sliders), we have a setter- and getter-method for the color-value. These two methods are called via *CMyGUI::DoCallAPI()* when the controller has been created or before it is deleted.

6.4 Advanced

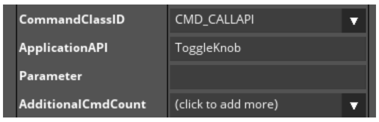
This dialog shows some more advanced controls of Guiliani.

6.4.1 GSE

6.4.1.1 Knobs

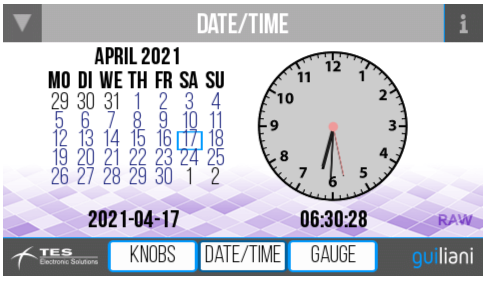


Here we have two knob-like controls which are connected to each other and a ***TextField***.



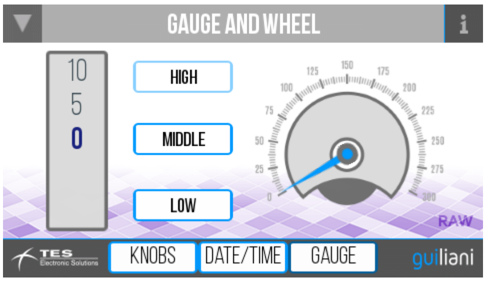
The button below the left ***Knob*** is attached to CallAPI with parameter "*ToggleKnob*" which will toggle how the value of the ***Knob*** will be controlled. This can either be radial or axial and corresponds to the attribute "*AxisControl*" of the ***Knob***.

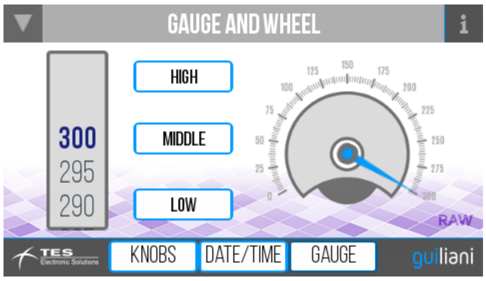
6.4.1.2 Date/Time



In this sub-dialog a calendar and a clock are displayed. There are no dynamics attached to the controls.

6.4.1.3 Gauge





This dialog demonstrates the ***Wheel***- and ***Gauge***-controls.

The buttons in the middle are attached to ***Commands*** which will set the value of the ***Wheel*** accordingly. ***Wheel*** and ***Gauge*** are connected via a ***DataPool***-entry.

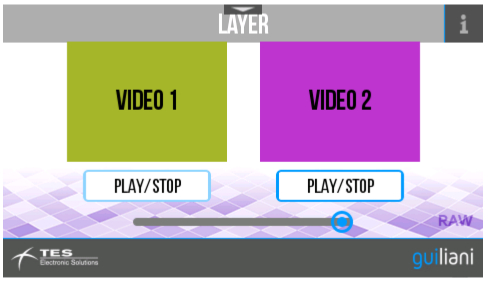
6.4.2 Code

Source-file: *<APP>/Source/DemoAdvanced.cpp*

Most of the initialization-code is similar to the Controls-dialog. It will get some objects and set specific values for their attributes. Additionally it will connect them together via ***Observers***. These are used to display the currently selected date as well as the current time, which is set by the clock.
HandleCallAPI() is used for toggling the way the Knob is controlled and each button in the Gauge-subdialog to define to which value the animation will set the value.

GetTime() will be called when the controller will be deleted to persist the current time used for the ***Clock***.

6.5 Layer



6.5.1 GSE-project

The only interesting parts are the ***LayerContainer*** which are assigned to the layers 1 and 2. If the target-platform is supporting hardware-layers the first button will start / stop dynamic changing of the content of the layer. The slider is used to change the transparency of the ***LayerContainer***.

6.5.2 Code

Source-file: *<APP>/Source/DemoLayer.cpp*

In the method *Init()* the relevant controls are collected which is in this case only the Slider to control the transparency of the layer.
Additionally there is some initialisation for the DC-Wrapper done which provides the interface to the hardware-layers.

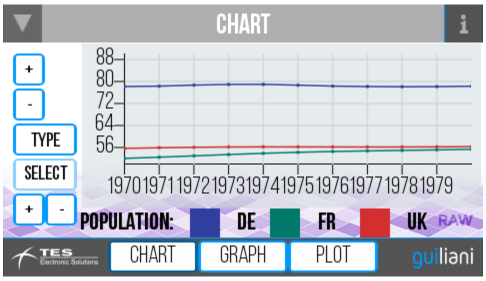
At last a timer with an interval of 100 milliseconds is started. This is used to change the background-colour for every available layer in the method *DoAnimate()*.
The method *OnNotification()* is used to receive changes from the Slider and to change the transparency-value for the layers.

Finally the method *HandleCallAPI()* receives the clicks on the button to start or stop the changing of the layer-content.

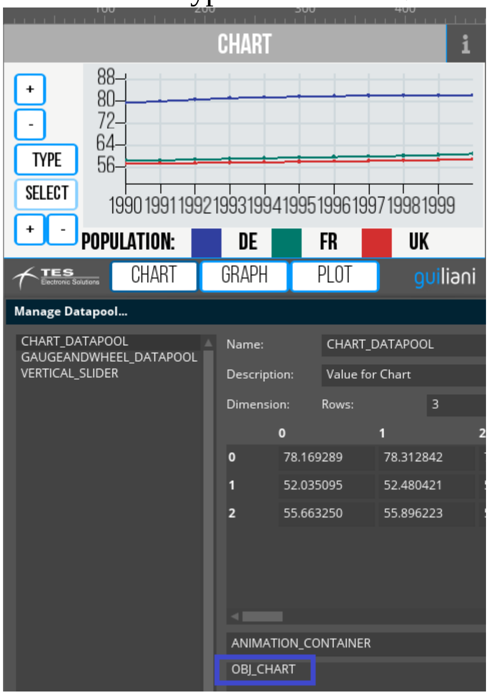
6.6 Data

6.6.1 GSE-projects

6.6.1.1 Chart



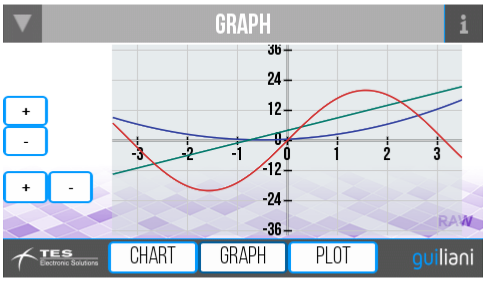
This sub-dialog contains a ***Chart*** which can display discrete values of multiple data-series like in some spreadsheet-program. The buttons on the left are all attached to ***CallAPI*** and control the zoom and type of the ***Chart***.



The ***Chart*** is connected via a ***DataPool***-entry. Since ***DataPool***-entries can have not only one value, but also two-dimensional arrays of value, this ***DataPool***-entry has multiple values.
Each row of values will form a data-series for the ***Chart*** and will be displayed in a different color. The values of the columns in each row are colored identically.

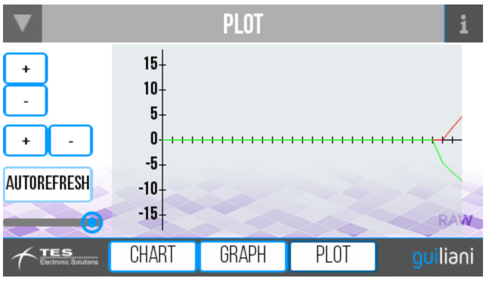
Note: you can use any type of value for a cell in the DataPool-entry. One could be a string, another one an integer and a third a float.

6.6.1.2 Graph



This sub-dialog contains a ***Graph*** which displays three different functions. The buttons can be used to zoom in or out the x- and y-axis via ***CallAPI***.

6.6.1.3 Plot



This sub-dialog shows a ***Plot*** which displays dynamic data simulated by the application. The buttons are used to zoom in or out the x- and y-axis via CallAPI. The button "*AutoRefresh*" will toggle the auto-refresh for the ***Plot*** and the slider the period of the refresh.

6.6.2 Code

Source-file: *<APP>/Source/DemoData.cpp*

The method In*it*() searches for the needed controls which are used for dynamics. The ***Chart*** will be set to a specific range and jumps to the beginning of the sequence. For the ***Graph*** three different functions are created and assigned to it. And finally for controlling the AutoRefresh for the ***Plot*** the ***Slider*** is hooked up with an ***Observer***.
There is also a timer with 500 milliseconds created for adding data to the Plot on a periodically basis.

The method *OnNotification()* will set the interval for the refresh of the Plot.

HandleCallAPI() will deal with all zooming and changing the type and the selection-mode for the Chart. *DoAnimate()* will just generate two new values and add these to the ***Plot***.

6.7 Container

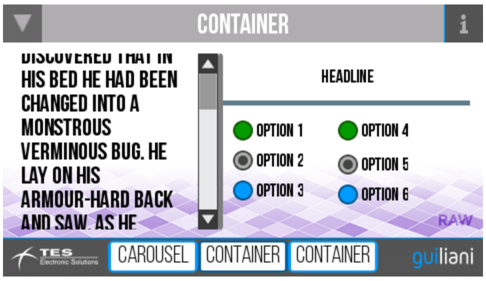
6.7.1 GSE-project

6.7.1.1 Carousel

This dialog demonstrates the ***Carousel*** which can display several objects simultaneously.

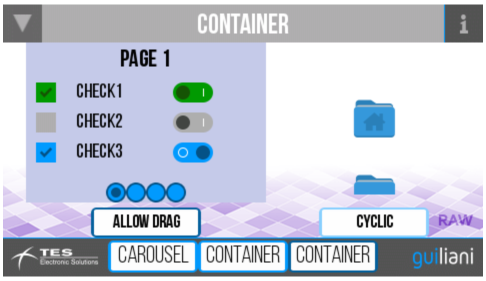
The ***Sliders*** will control attributes like tilting-angle or width of the ***Carousel***. The ***CheckBox*** toggles the flow-mode which will display the objects inside the ***Carousel*** in a horseshoe shaped list instead of a projected circle.

6.7.1.2 Container1



In this sub-dialog the screen is divided into three parts with a large ***TextField*** which can be swiped, a static ***TextField*** and several ***RadioButtons***. Each of these parts can be resized by dragging the splitter-handle.

6.7.1.3 Container2



This sub-dialog shows two container-types: ***PageContainer*** and ***WheelContainer*** which contain several elements.

To switch to another page in the ***PageContainer*** it can either be dragged horizontally or the buttons at the bottom can be clicked. The ***CheckBox*** “AllowDrag” toggles if the ***PageContainer*** can be dragged.
The elements in the ***WheelContainer*** can be cycled via vertical drag. The ***CheckBox*** “Cycle” will toggle cyclic display.
Both ***CheckBoxes*** have a ***CallAPI*** attached to them.

6.7.2 Code

Source-file: *<APP>/Source/DemoContainer.cpp*

In the code there are three ***Observers*** for the ***Carousel***, for controlling the Tilt, the Radius and the number of elements.

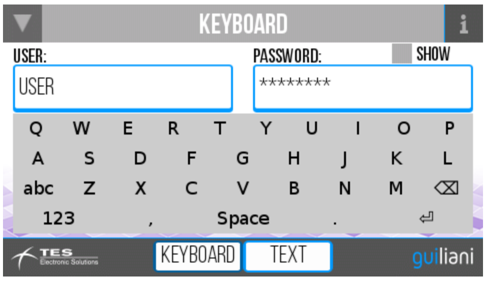
In the method *Init()* as usual all necessary controls are searched and connected to their Observers.

HandleCallAPI() will do the change of the flow-mode for the ***Carousel*** and toggle the dragging and cyclic mode for the other two container.

6.8 Text

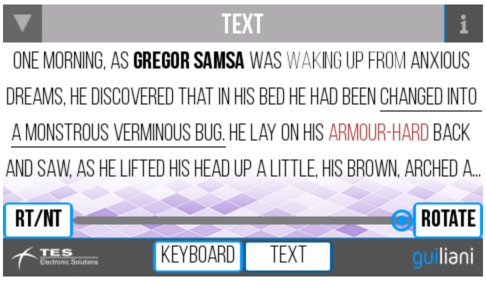
6.8.1 GSE-project

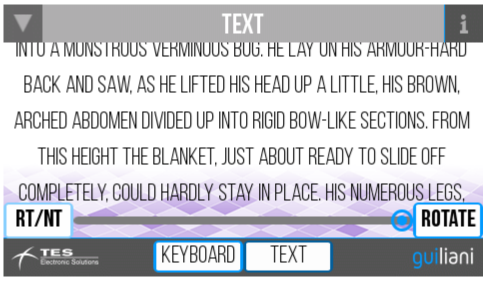
6.8.1.1 Keyboard



This dialog shows the ***Keyboard*** and two ***InputFields***, where one can be obfuscated like a password.

6.8.1.2 Text





This dialog is very simple as it only shows a large text – either as a ***TextField*** or as a ***RichTextField*** – and a Slider which controls the width of the shown text.

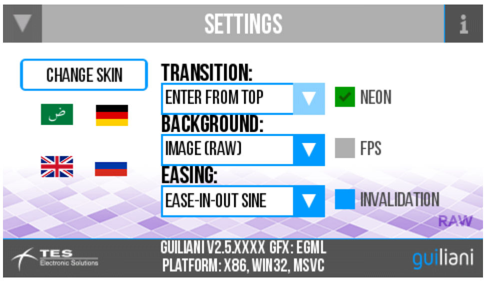
Except for the buttons at the bottom of the dialog there are no dynamics.

6.8.2 Code

Source-file: *<APP>/Source/DemoText.cpp*

In the method *Init()* only the ***Observer*** for the ***Slider*** and the ***TextFields*** is created. This ***Observer*** will simply adjust the width of the ***TextFields*** to the value of the ***Slider***.

6.9 Settings



In this dialog there are some settings for the GuilianiDemo which can alter the visuals and dynamics of the application.

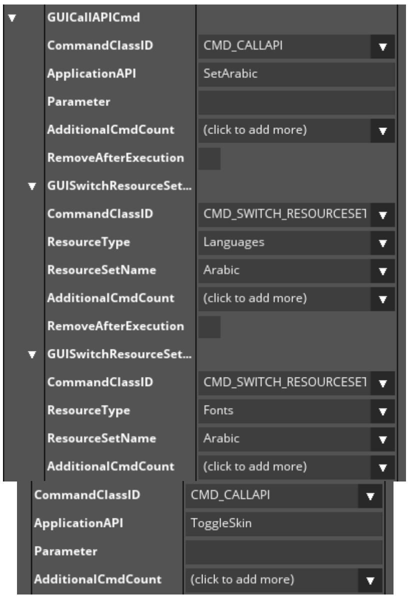
The buttons on the left with little flags will switch the currently active language. Click on the corresponding flag and the texts will immediately change. When you click on the flag for Arabic it will also switch the font-set and the text-direction to display the texts correctly.

You can select different behavior from several ***ComboBoxes*** in the middle:

- Transition: this specifies the kind of transition which is made when navigating from the main-dialog to another dialog. The transition back to the main-dialog is always “Push-from-Bottom” and cannot be changed
- Background: here you can choose the background-image which should be displayed or switch to a ***GeometryObject*** whose color can be set in the Slider1-subdialog of the Controls-dialog
- Easing: this will set the easing used in the Gauge-dialog when the ***Wheel*** and ***Gauge*** animate to a new value

When clicking on the button “Change Skin” you can switch between the two different image-sets (Light, Dark).
Note: if only one image-set has been exported this button will not work at all.
The Button “Toggle Neon” can be used to toggle NEON-optimization, when using eGML-software rasterizer.
“Toggle FPS” will show/hide onscreen FPS and “Toggle INV” will show/hide rectangles where invalidated areas are.

6.9.1 GSE-project

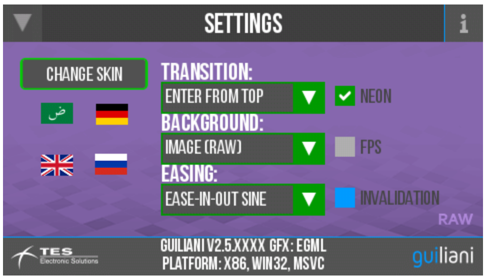


The buttons on the left are used to switch the language of the application.
First a ***CallAPI*** is executed and after that the resource-sets for texts and fonts are switched to the appropriate value.

The button on the right is used to toggle the currently used image-set via ***CallAPI***

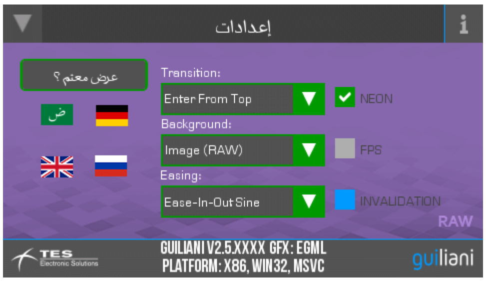
6.9.1.1 Change Skin

To change the skinning of GuilianiDemo a CallAPICmd is used which will toggle the currently used image-set and property-set. The property-set is also needed, because the text on buttons might become unreadable when the images for this button are changed. By changing the colours via the property-set solves this problem.



6.9.1.2 Change Language

By clicking on one of the flags you can change the language used in the GuilianiDemo to this language. When clicking on the Arabic flag we also switch to another font-set, since the fonts we use for the other languages do not contain the glyphs for Arabic.



6.9.2 Code

Source-file: *<APP>/Source/DemoSettings.cpp*

The method *Init()* is mainly used to fill the contents of the ***ComboBoxes*** for the various settings.
It also updates the text at the bottom of the dialog displaying the currently used version of Guiliani as well as the used ***Graphics-Wrapper*** and Platform.

In *HandleCallAPI()* the selected values will be persisted in the main-class when the controller will be deleted.

ChangeBackground() will show the selected type of background and the correct image.

6.10 Scratchpad

This dialog can be used to play around with Guiliani. Have Fun.