



life.augmented

# UI Development Fundamentals

This presentation is available on [support.touchgfx.com](https://support.touchgfx.com)

# Agenda

1 Introduction

2 Software architecture

3 Working with TouchGFX Designer

4 Designer user guide

5 UI components

6 Working with TouchGFX in User Code

With this presentation, you get a fundamental knowledge about TouchGFX UI Development. You will learn:

- The basics of the software architecture
- Getting started working with the Designer and its UI components
- Discovering TouchGFX Engine features

## Further reading

- Go to the TouchGFX documentation site :

<http://support.touchgfx.com/>

- Slides in this workshop will refer to relevant documentation pages. Links will be in the lower corner of the slides
- A good place to start reading following this workshop is:

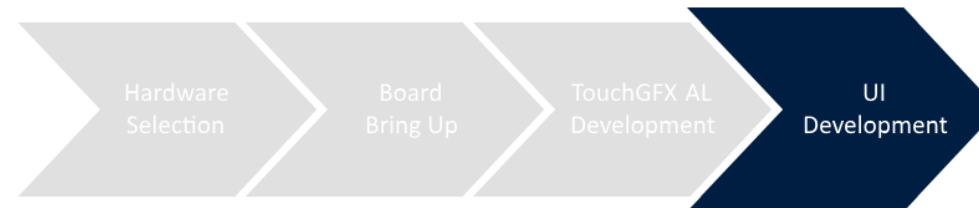
[UI Development Introduction](#)

## TouchGFX Development

### Main components:



### Main activities:



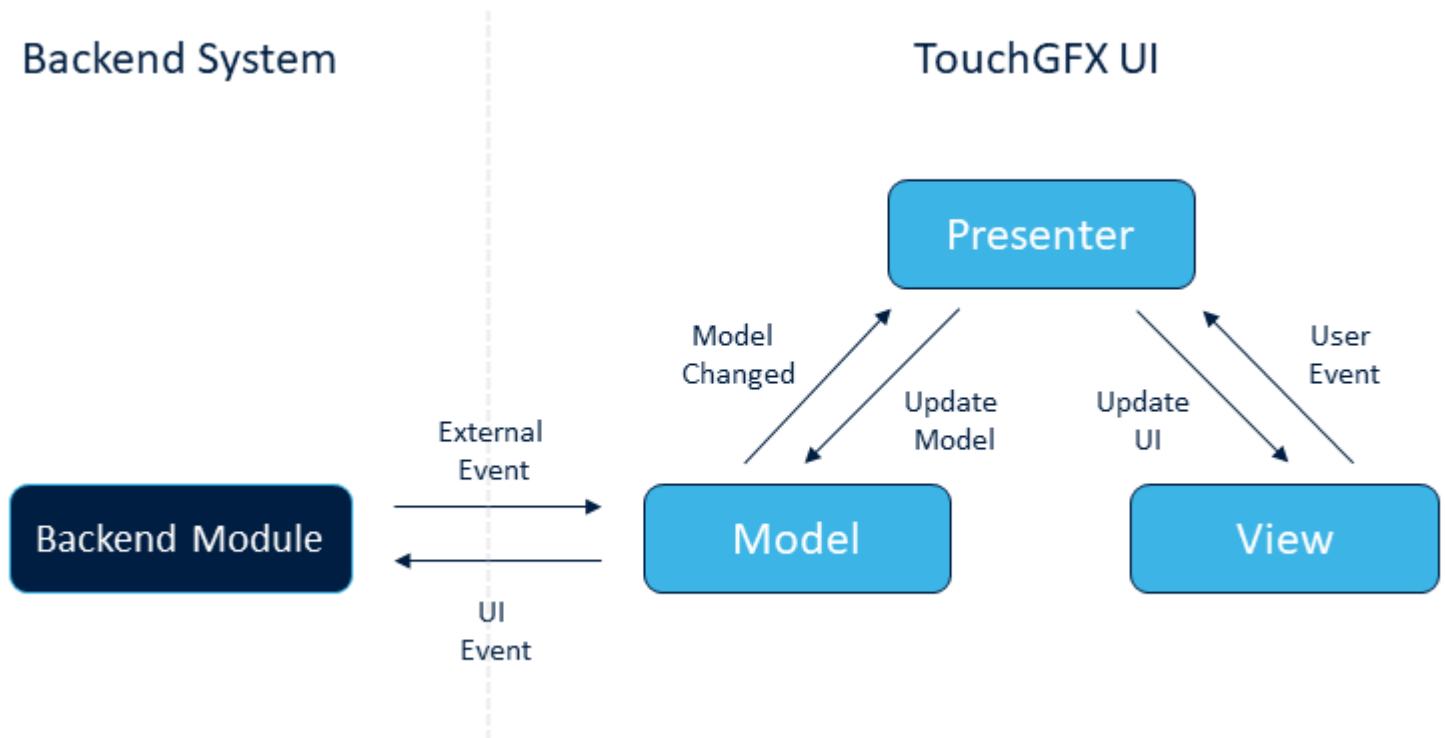
In this presentation, we will focus on the UI development activity

We go deeper into the different activities in the other presentations

# Software architecture

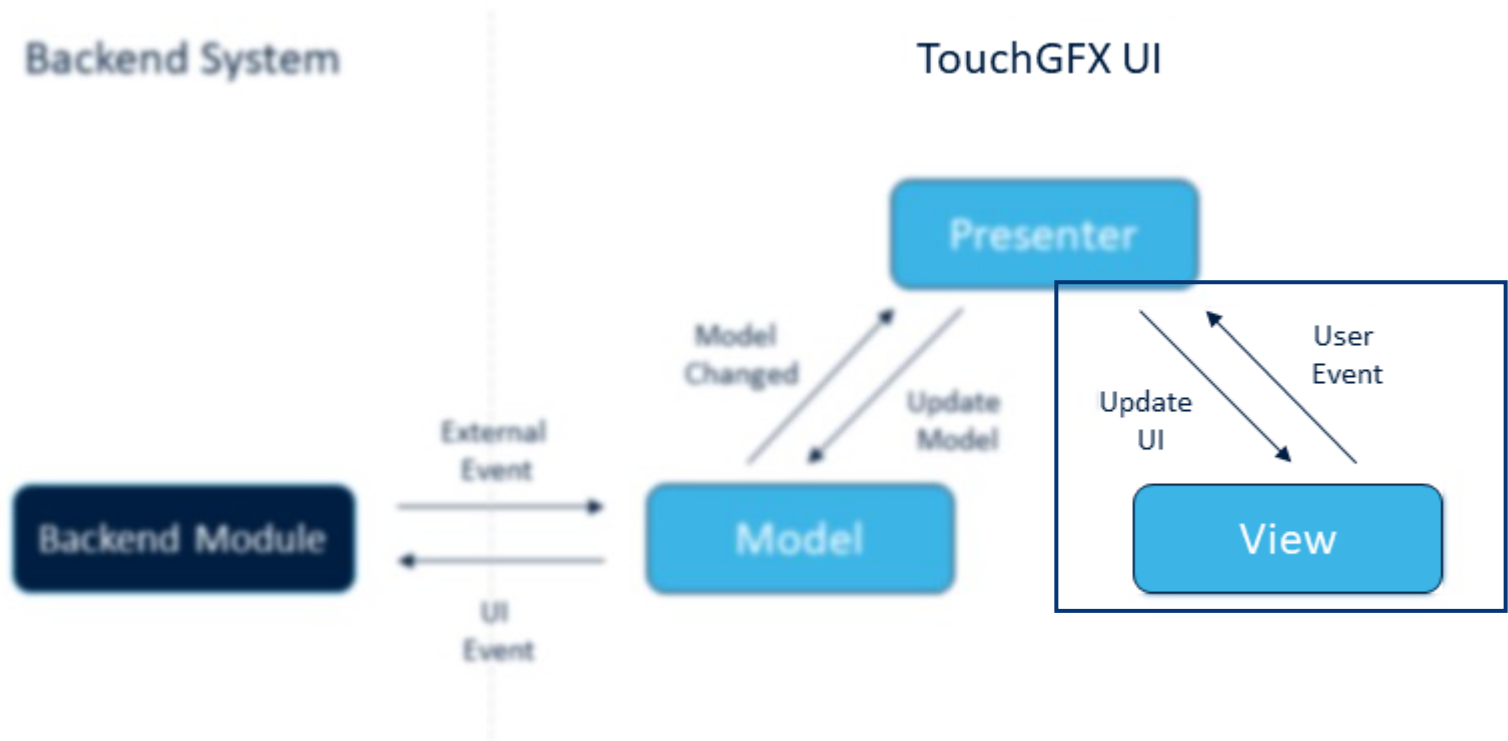
# Software architecture

- Model-View-Presenter (MVP)
  - View
  - Model
  - Presenter
- Benefits
  - Separation
  - Unit testing
  - Decoupled code



# Software architecture

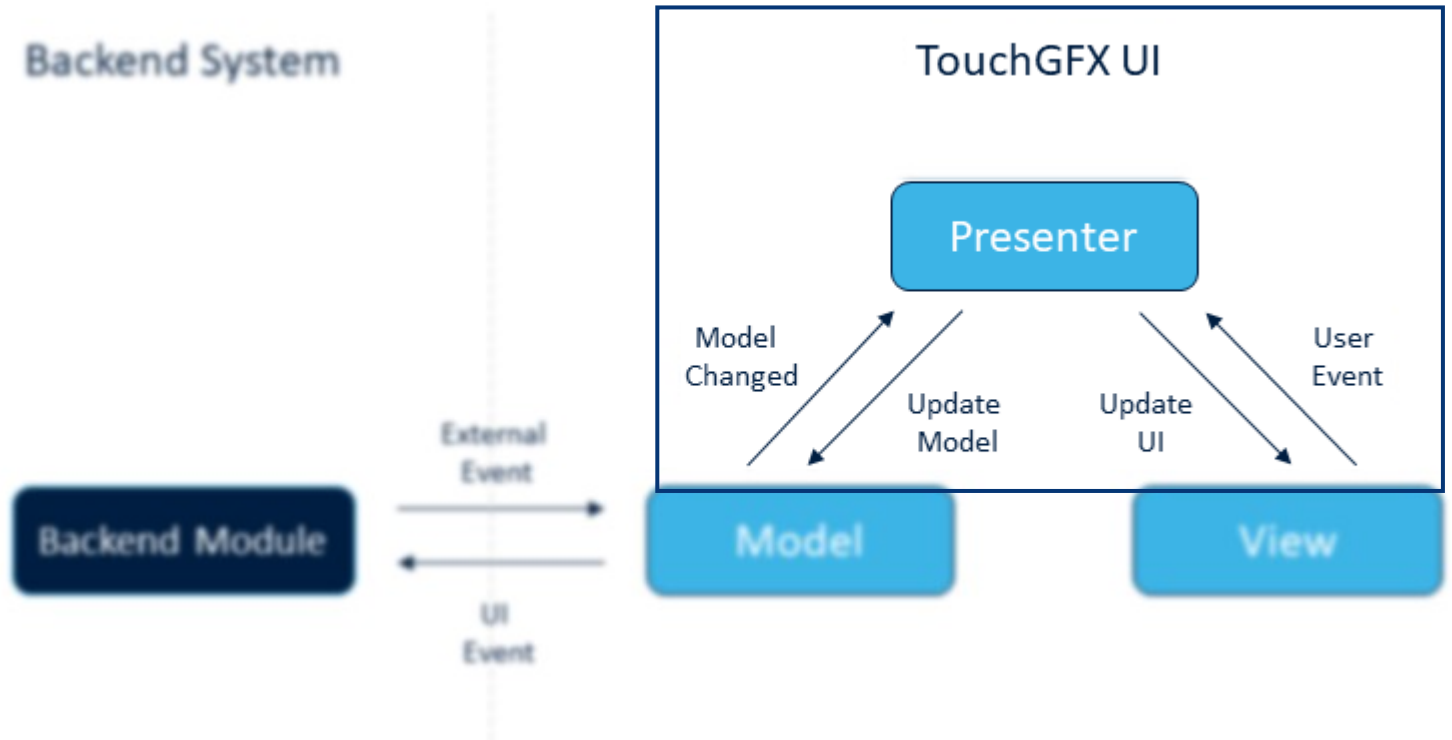
- View
  - Creation of UI elements
  - UI element manipulation
  - “Reception” of user input
  - Sending of user input to Presenter
  - Update of UI based on data from Presenter





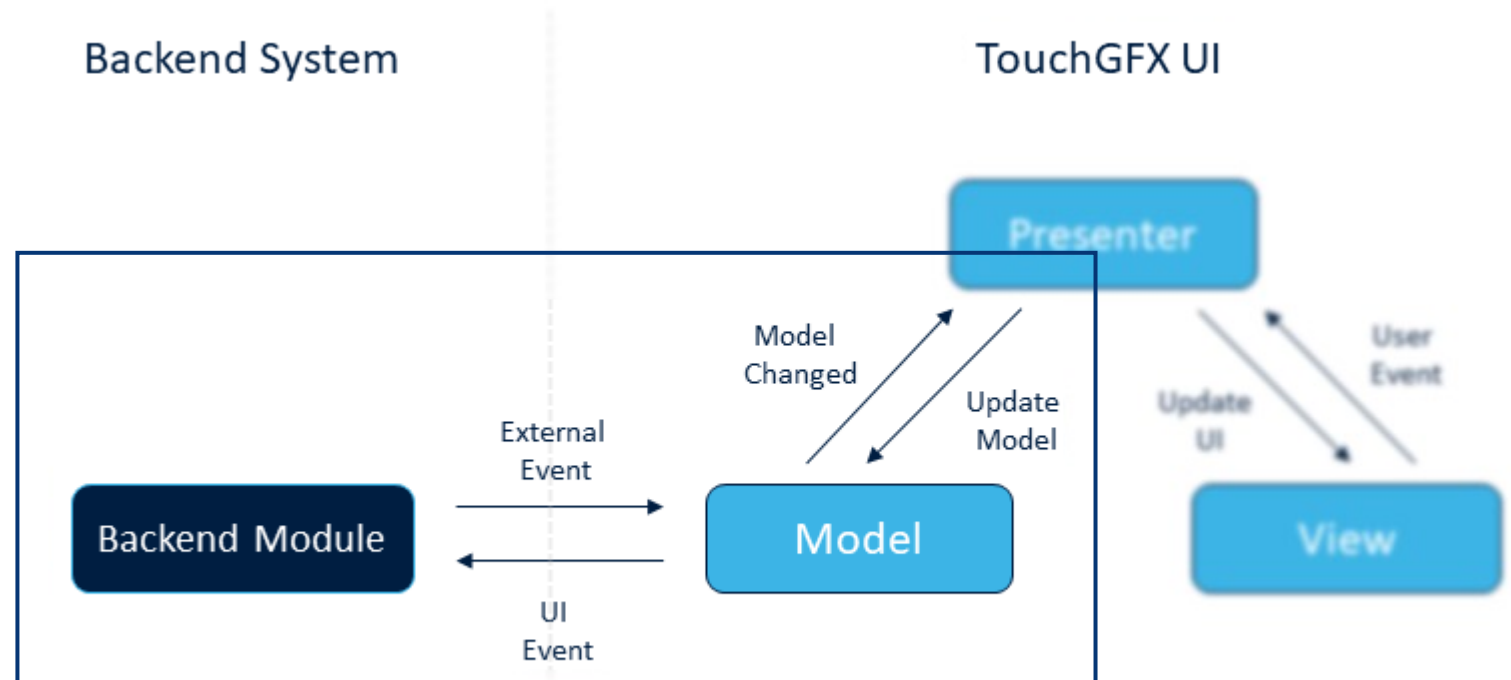
# Software architecture

- Presenter
  - Mediator between Model and View
    - Receives data from the Model and prepares it for the View
    - Receives input from the View and sends it to the Model
  - Business logic for UI
  - No knowledge about UI implementation



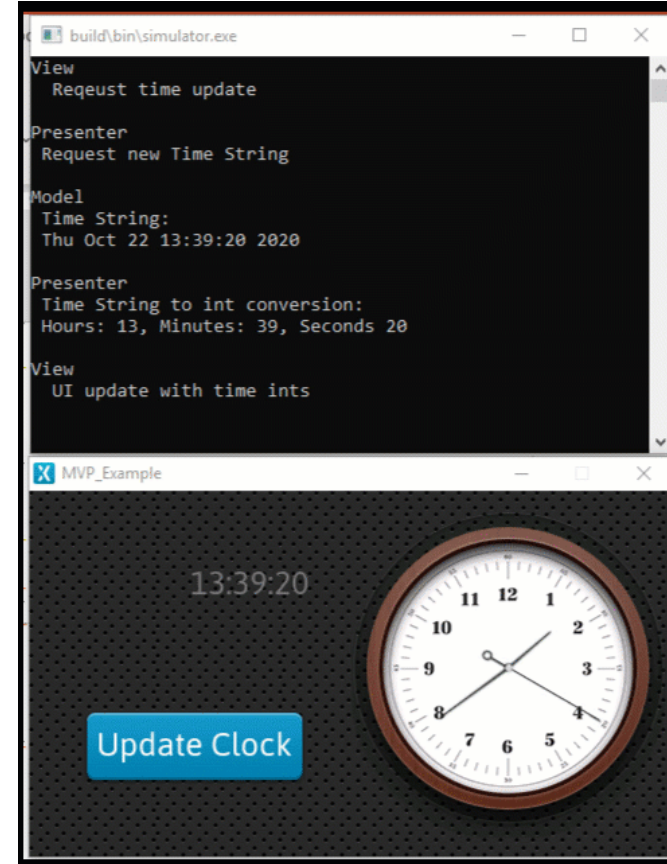
# Software architecture

- Model
  - Communicates with the backend
  - Stores data
  - Passes data to the Presenter



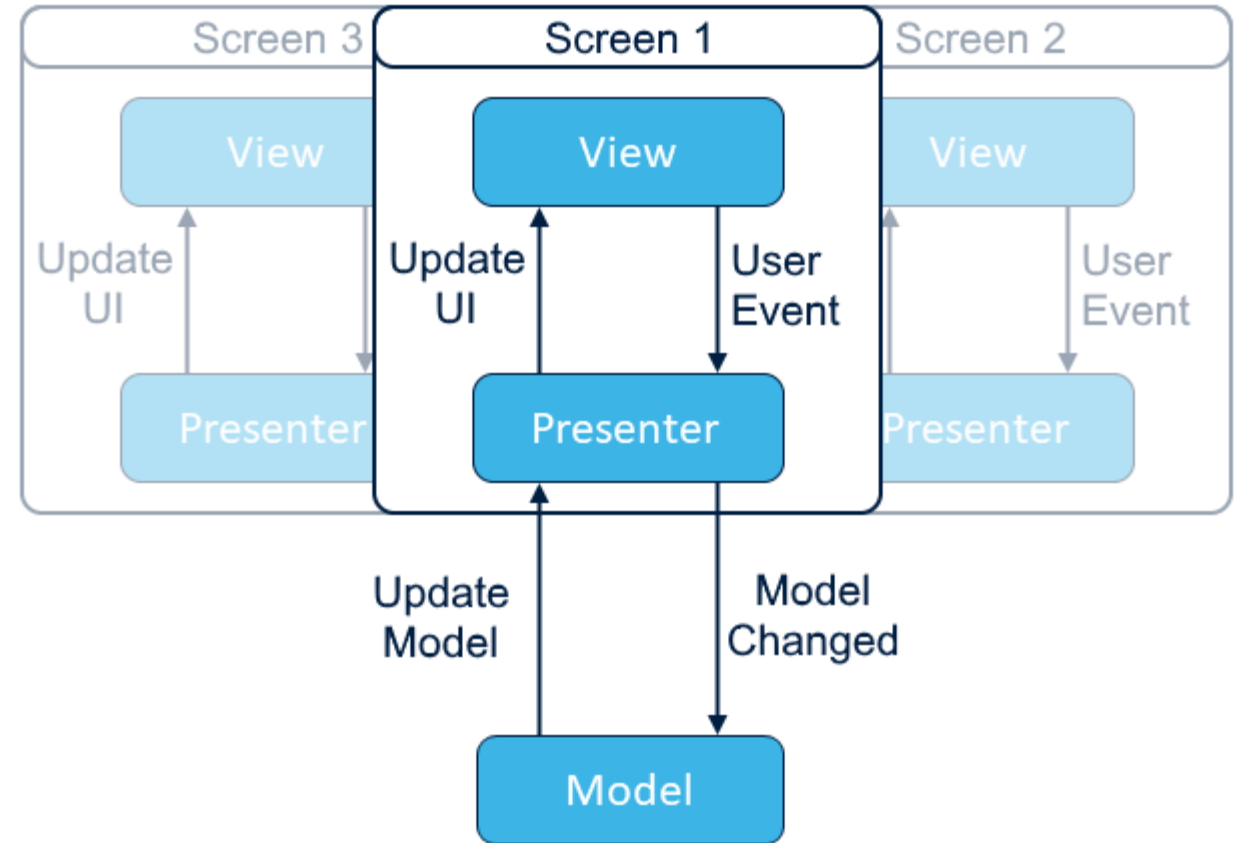
# Software architecture

- MVP – Example
  - When update is pressed, a get time request is sent to the Model via the Presenter
  - The Model sends time data to the Presenter
  - The Presenter converts the data into something the Clock widget can use and sends it to the View
  - The View updates the Clocks with the new data



# Software architecture

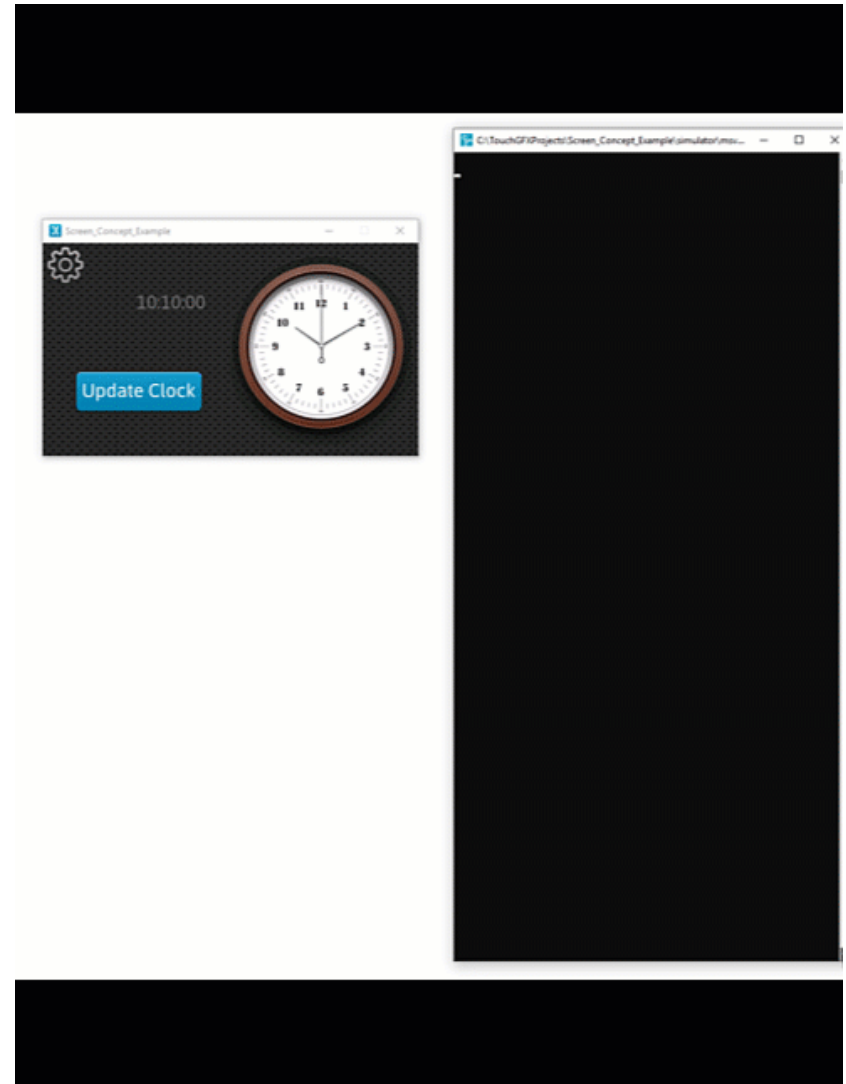
- The Screen Concept
  - Application consists of a number of “Screens”
  - Logical grouping of UI and UI logic
  - One View class and one Presenter class
  - One active screen at the time
  - Minimizes the amount of RAM-memory



*Illustration of the screen concept with screen 1 as the active screen*

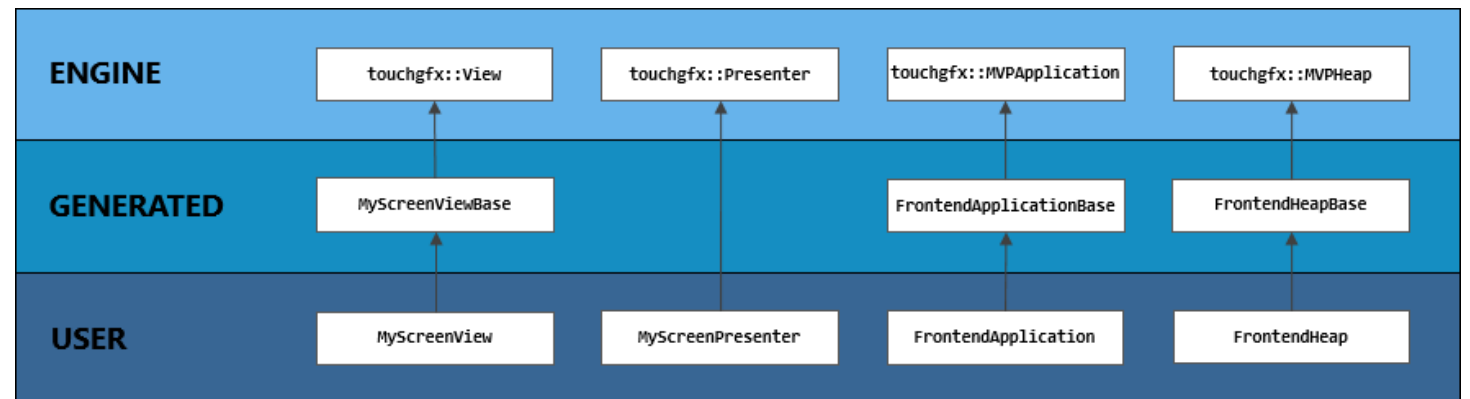
# Software architecture

- Screen Concept: Example
  - When selecting clock mode, in the “settings” screen, the selected mode is stored in the model
  - When pressing update, the Clock also retrieves the clock mode from the model



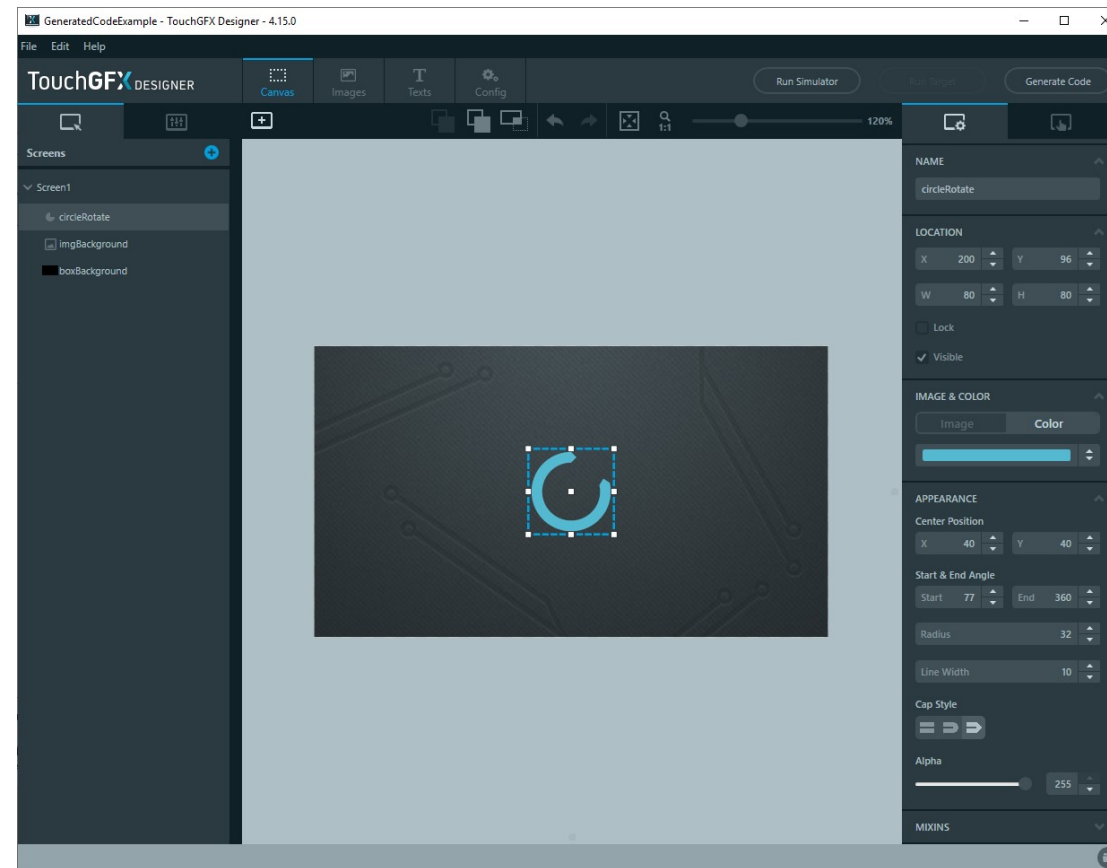
# Software architecture

- Generated Code vs. User Code
  - Engine
    - Provided by TouchGFX
  - Generated
    - Generated by the Designer multiple times
    - Code for the widgets added in the Designer
    - Regenerated often
  - User
    - Generated by the Designer once
    - Created for handwritten code



# Software architecture

- Rotating Circle Example: The Designer



- Rotating Circle Example: The generated code

```
1  /***** THIS FILE IS GENERATED BY TOUCHGFX DESIGNER, DO NOT MODIFY *****/
2  /***** THIS FILE IS GENERATED BY TOUCHGFX DESIGNER, DO NOT MODIFY *****/
3  /***** THIS FILE IS GENERATED BY TOUCHGFX DESIGNER, DO NOT MODIFY *****/
4  #include <gui_generated/screen1_screen/Screen1ViewBase.hpp>
5  #include <touchgfx/Color.hpp>
6  #include "BitmapDatabase.hpp"
7
8  Screen1ViewBase::Screen1ViewBase()
9  {
10
11     touchgfx::CanvasWidgetRenderer::setupBuffer(canvasBuffer, CANVAS_BUFFER_SIZE);
12
13     __background.setPosition(0, 0, 480, 272);
14     __background.setColor(touchgfx::Color::getColorFrom24BitRGB(0, 0, 0));
15
16     imgBackground.setXY(0, 0);
17     imgBackground.setBitmap(touchgfx::Bitmap(BITMAP_BG_ID));
18
19     circleRotate.setPosition(200, 96, 80, 80);
20     circleRotate.setCenter(40, 40);
21     circleRotate.setRadius(32);
22     circleRotate.setLineWidth(10);
23     circleRotate.setArc(77, 360);
24     circleRotate.setCapPrecision(90);
25     circleRotatePainter.setColor(touchgfx::Color::getColorFrom24BitRGB(85, 185, 209));
26     circleRotate.setPainter(circleRotatePainter);
27
28     add(__background);
29     add(imgBackground);
30     add(circleRotate);
31 }
32
33 void Screen1ViewBase::setupScreen()
34 {
35
36 }
```

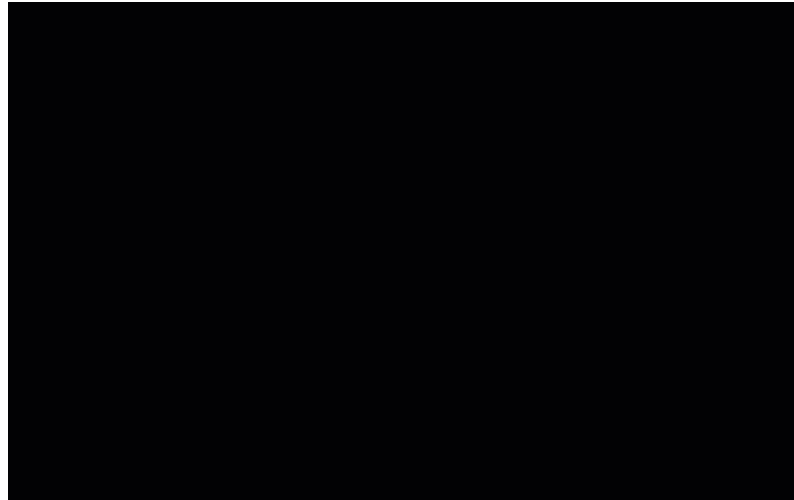


- Rotating Circle Example: The User Code

```
1  #include <gui/screen1_screen/Screen1View.hpp>
2
3  Screen1View::Screen1View()
4  {
5  }
6
7  void Screen1View::setupScreen()
8  {
9  }
10
11 void Screen1View::tearDownScreen()
12 {
13 }
14
15 void Screen1View::handleTickEvent()
16 {
17     // sets the new angle to rotate circleRotate //
18     circleRotate.invalidate();
19
20     circleRotate.setArc(circleRotate.getArcStart() + 1, circleRotate.getArcEnd() + 1);
21
22     circleRotate.invalidate();
23 }
```

# Software architecture

- Rotating Circle Example: The Demo

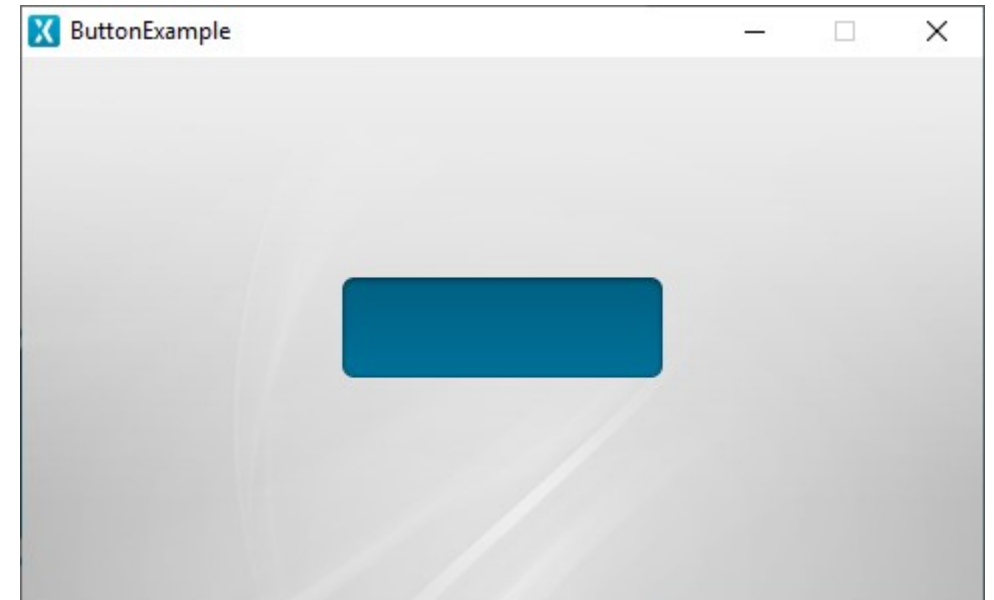


# Working with TouchGFX Designer



A Widget is an abstract definition of something that can be drawn on the screen and be interacted with

- Numerous standard widgets available
  - Customizable from Designer and Code
  - The adding order determines the display order
- Possible to create custom widgets



*A Button widget with an Image widget as a background*

# Containers

A Container is a component that contains child nodes

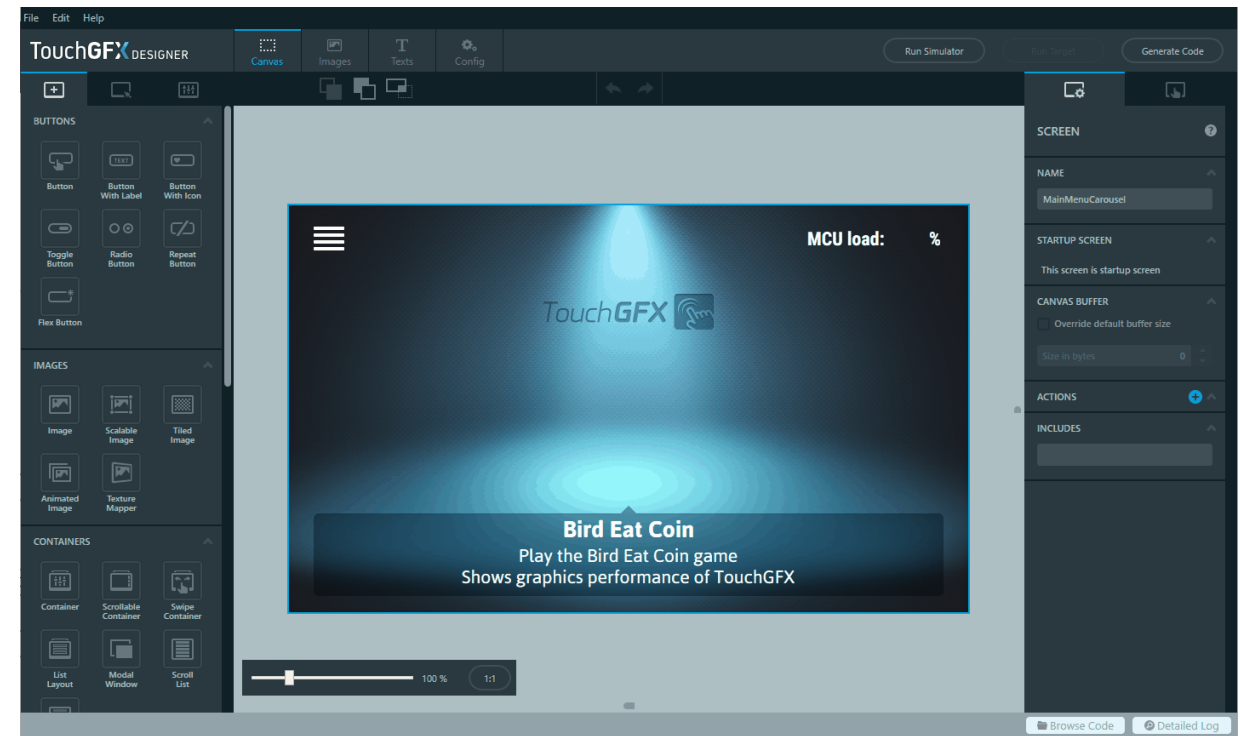
- Children can be widgets or other containers
- Position of widgets defined relatively to the parents
- Widgets are added to a container by dragging widgets into the container in the tree view
- Containers act as viewports
  - Only the parts of the children that intersect with the geometry of the container are visible



*A container acting as a view port*

## TouchGFX includes a simulator for UI testing

- Building and running on target can be time-consuming
- You can run the simulator in 3 ways by use of:
  - TouchGFX Designer
  - TouchGFX Environment
  - Visual Studio
- Option to create User code that only runs when using the simulator



*Launching the simulator from TouchGFX Designer*

A TouchGFX application can be compiled for the PC simulator or for the target hardware

- PC simulator: Projects compiled using GCC or Visual Studio
- Hardware target: Projects compiled with the active toolchain
  - Active toolchain set in CubeMX

Flashing projects on STM32-based boards can be done with different toolchains.

- Building generates a binary (.hex or .elf)
  - Flash with STlink or STM32Cube Programmer
- Running on target is done using GCC
  - Command can be overridden in configurations



# Debugging

A TouchGFX project is a set of C++ files generated by TouchGFX Designer, TouchGFX Generator and written by developers

- Can be debugged as any other C++ application
- Target debugging with an IDE
  - Performance testing
- Simulator Debugging
  - Faster process than on target
- Using the DebugPrinter
  - Prints debug messages on the display

# Getting started

Using UI templates is a good way to start a project or to understand how to work with widgets and TouchGFX

- Widget showcasing
- Complete demos
- Can be used as source of inspiration

Online applications are out-of-the box applications dedicated to an ST evaluation kit

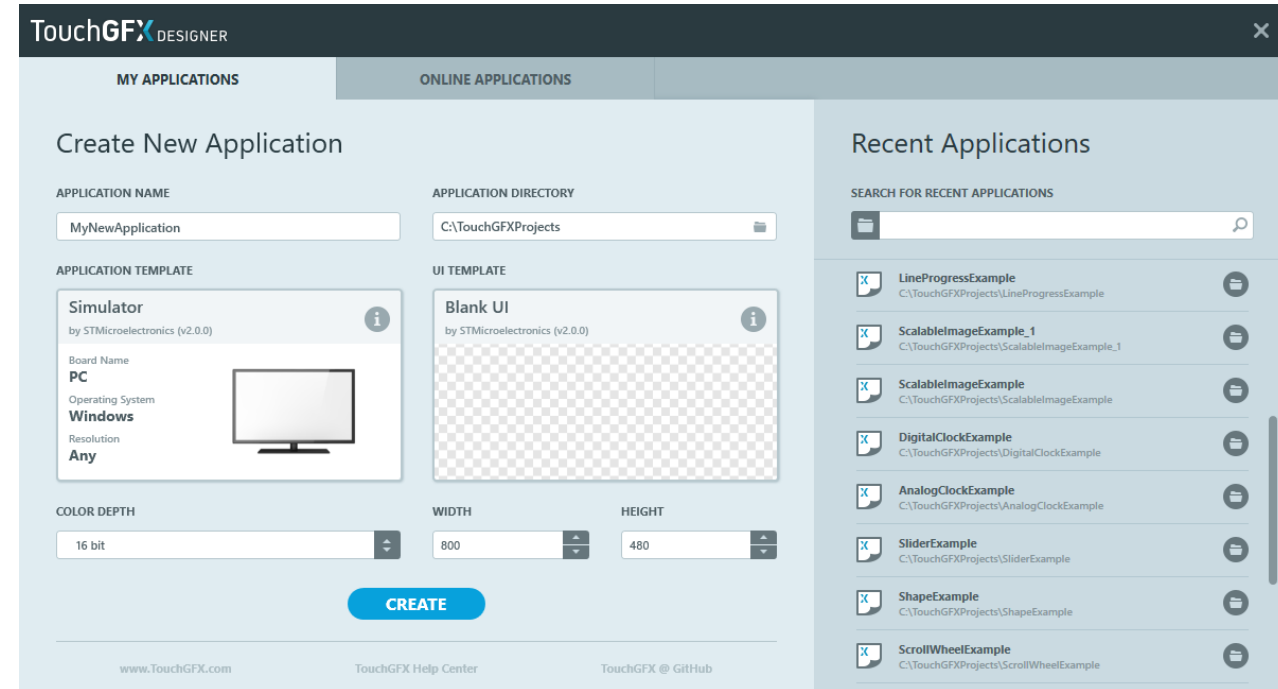
- More complex than UI templates
- Explore different features of the TouchGFX framework
- Can include sample integration with hardware

# Designer user guide

# Startup Window

This is where new projects can be configured and created

- Possibility to work with the Simulator or with an Application or UI template

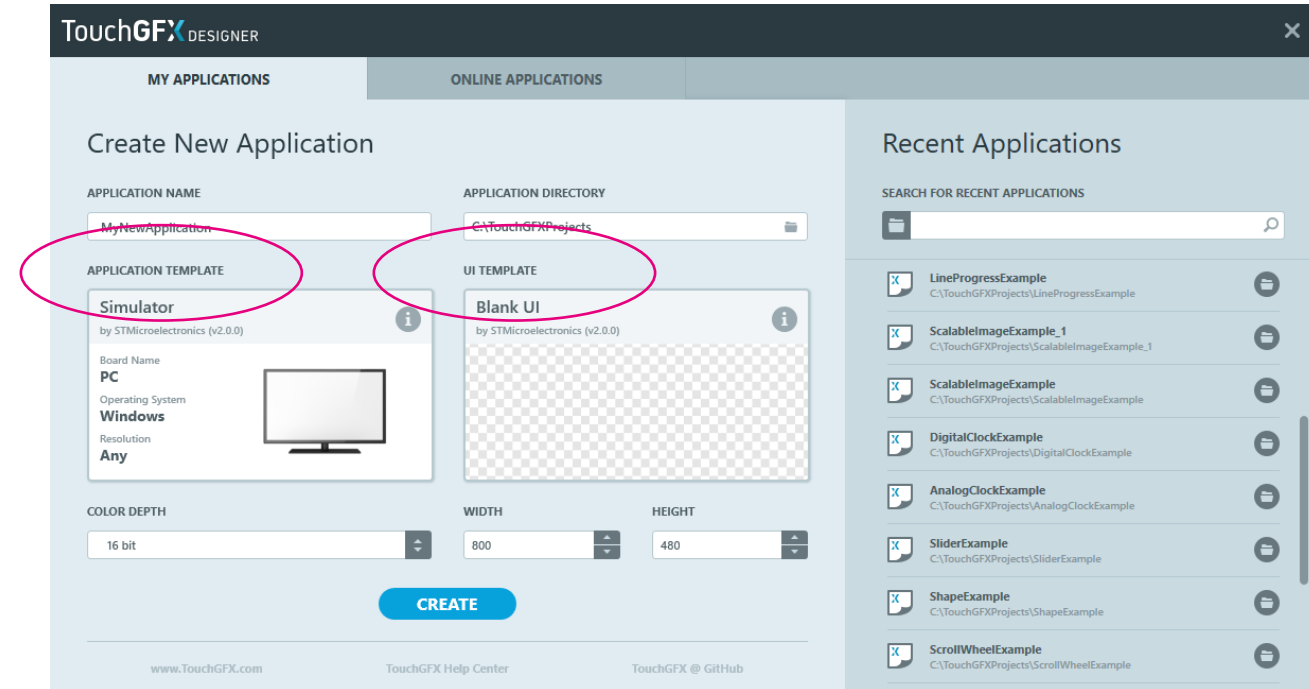


*Startup window when opening TouchGFX Designer*

# Startup Window

This is where new projects can be configured and created

- Possibility to work with the Simulator or with an Application or UI template

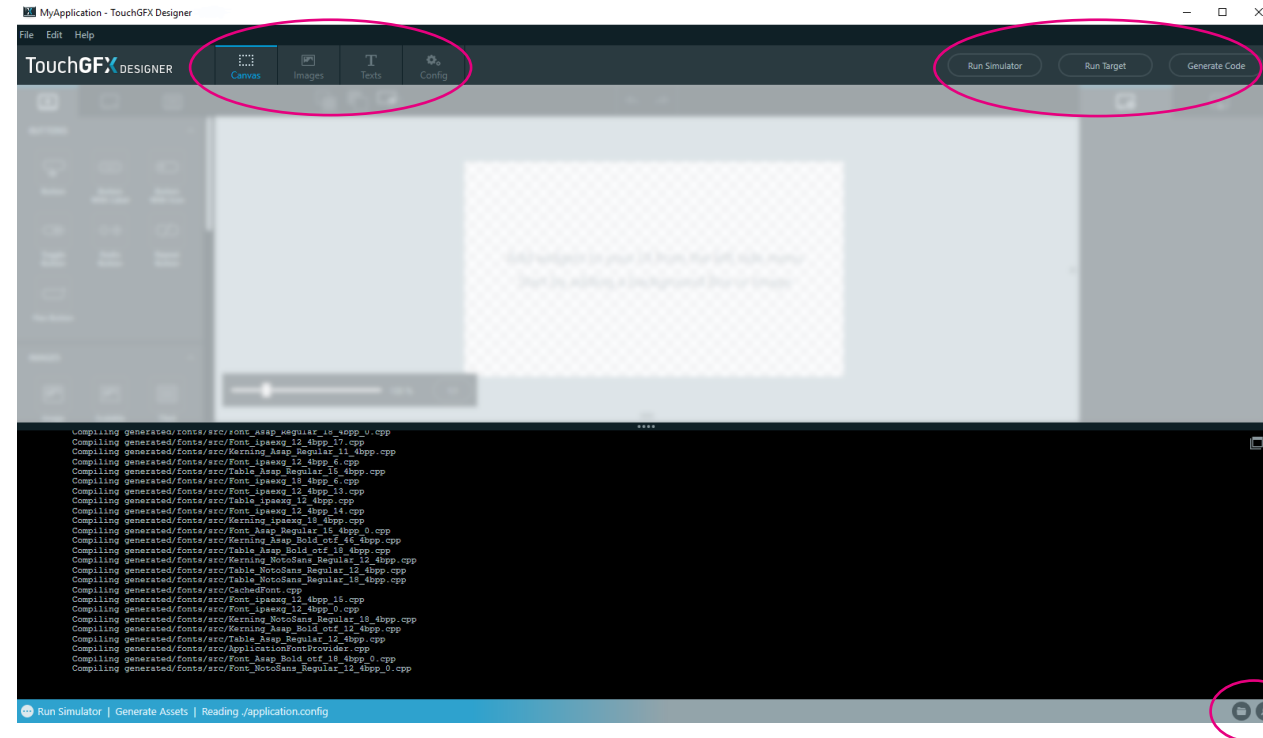


Startup window when opening TouchGFX Designer

# Main Window

Contains a Navigation Bar, Command Buttons, Notification Bar, and a Detailed log

- Navigation bar
  - *Canvas*: drag and drop application building
  - *Images*: management of images used
  - *Text*: management of text and typographies
  - *Config*: configuration of project settings

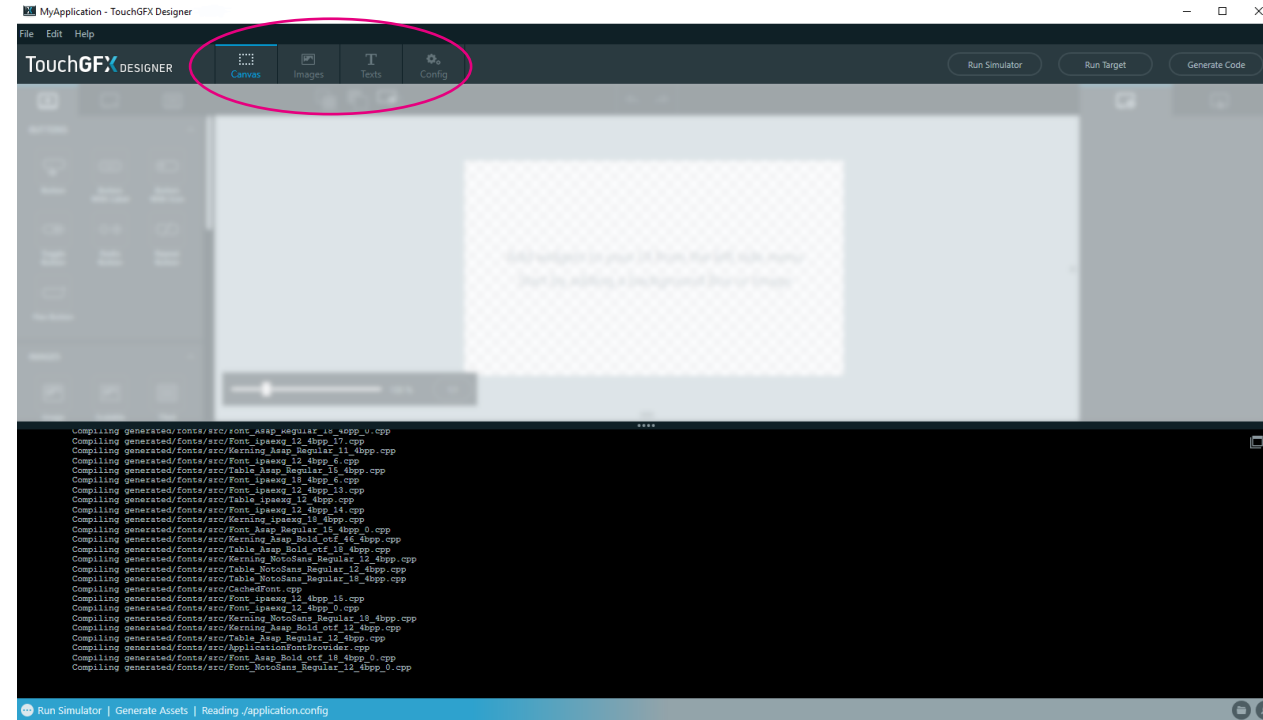


The main window options

# Main Window

Contains a Navigation Bar, Command Buttons, Notification Bar, and a Detailed log

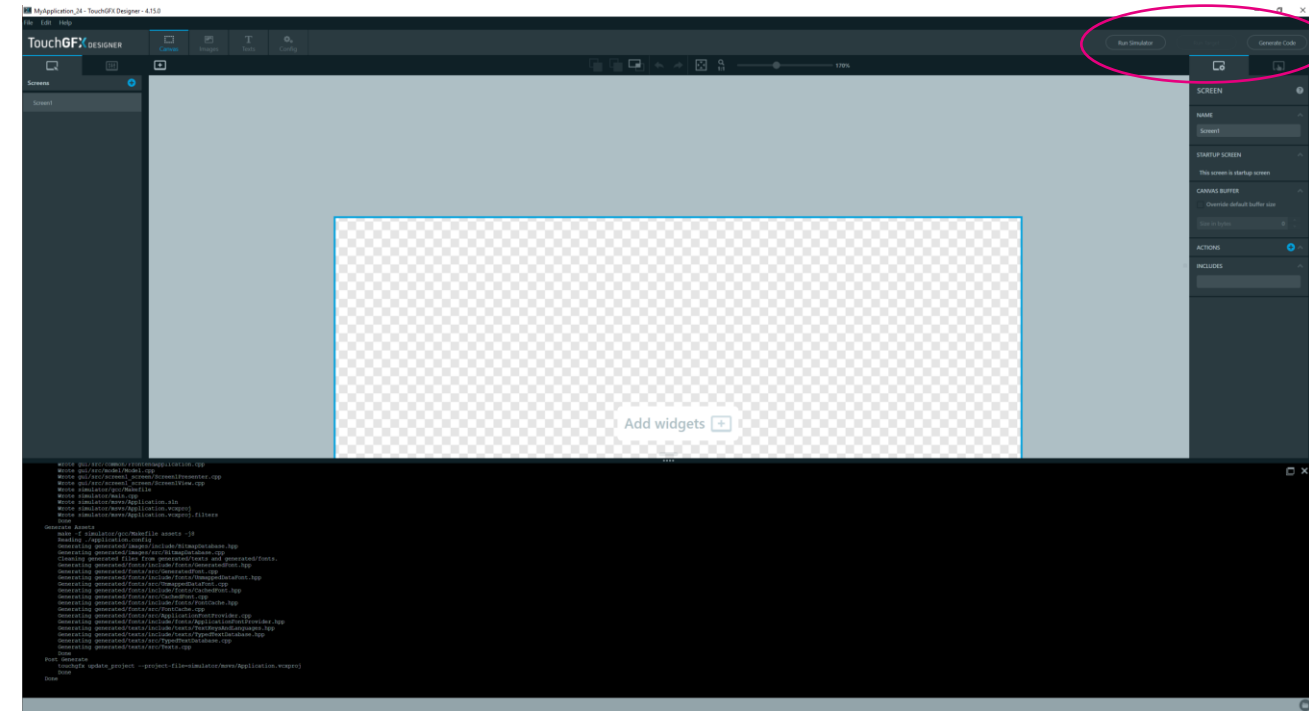
- Navigation bar
  - *Canvas*: drag and drop application building
  - *Images*: management of images used
  - *Text*: management of text and typographies
  - *Config*: configuration of project settings



The main window options

# Main Window

- Project can be run on target or on PC simulator
  - Both commands trigger assets, code generation and compiling
  - Key shortcuts are respectively “F5” and “F6”
- Notification bar and Detailed log are at the bottom of the Main Window
  - Shows status of running command
  - Opens log of last command
  - Can be toggled with keys “Alt” + “L”

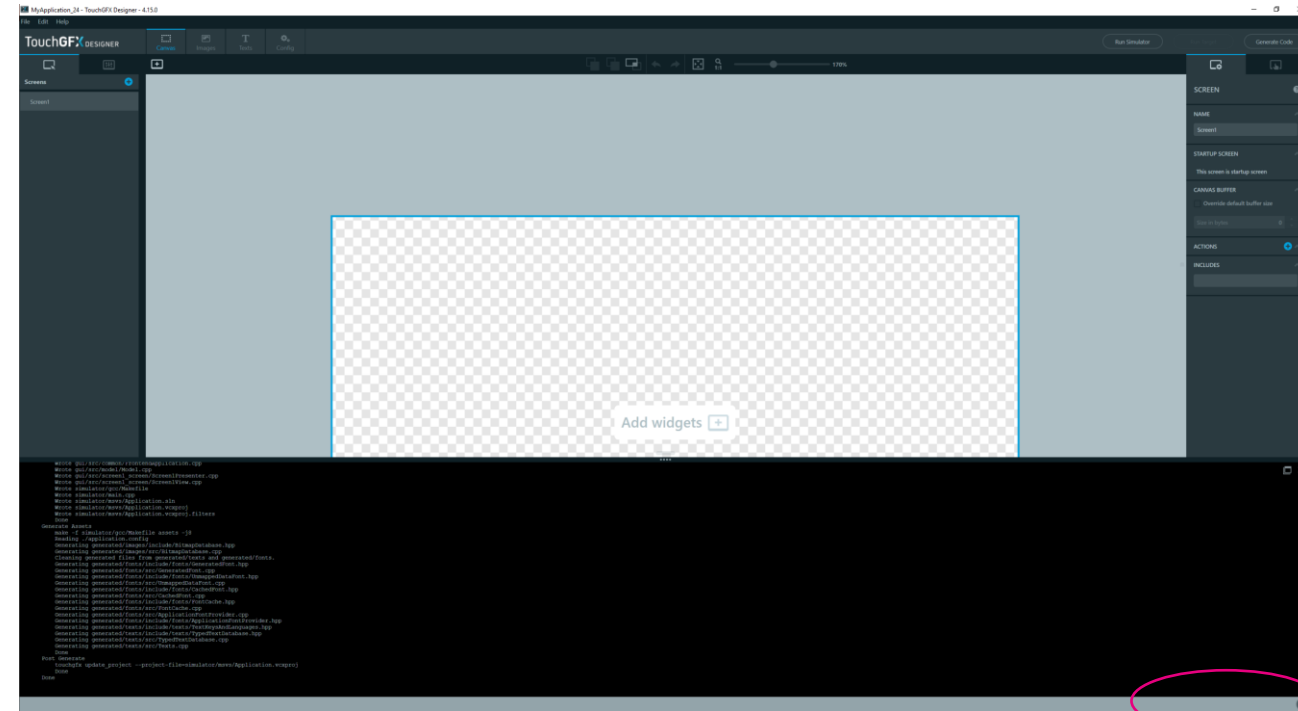


Running and generating code commands, Navigation bar and Browser Code positions



# Main Window

- Project can be run on target or on PC simulator
  - Both commands trigger assets, code generation and compiling
  - Key shortcuts are respectively “F5” and “F6”
- Notification bar and Detailed log are at the bottom of the Main Window
  - Shows status of running command
  - Opens log of last command
  - Can be toggled with keys “Alt” + “L”

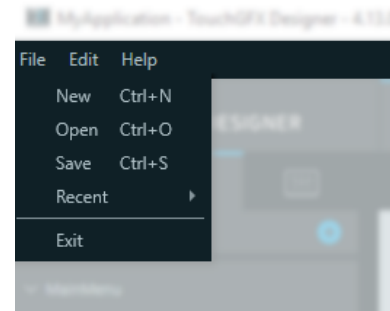


Running and generating code commands, Navigation bar and Browser Code positions

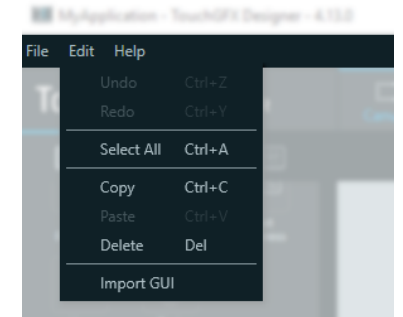
# File Menu

Consists of a File, Edit and Help menu items

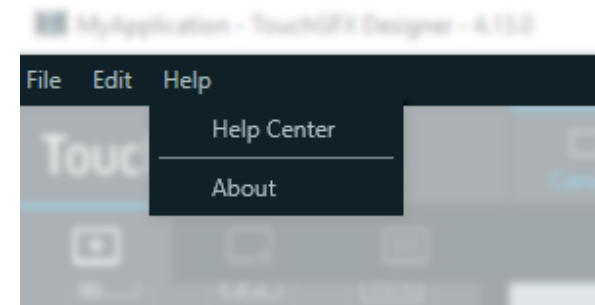
- File Menu used for creating/opening/saving projects
- Edit Menu interacts with UI elements. It can be used to import UI templates or demos after project creation
- Help Menu redirects to official support forum. Also contains Software License Agreement



*File menu item in File Menu*



*Edit menu item in File Menu*

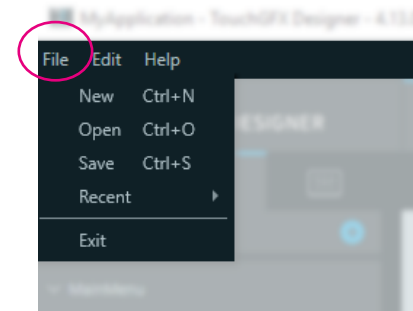


*Help menu item in File Menu*

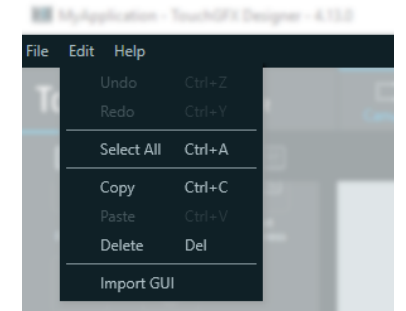
# File Menu

Consists of a File, Edit and Help menu items

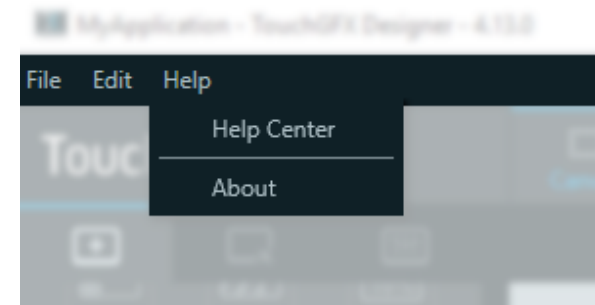
- File Menu used for creating/opening/saving projects
- Edit Menu interacts with UI elements. It can be used to import UI templates or demos after project creation
- Help Menu redirects to official support forum. Also contains Software License Agreement



*File menu item in File Menu*



*Edit menu item in File Menu*

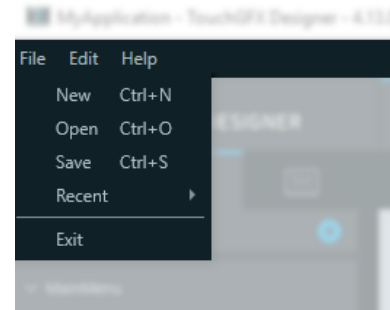


*Help menu item in File Menu*

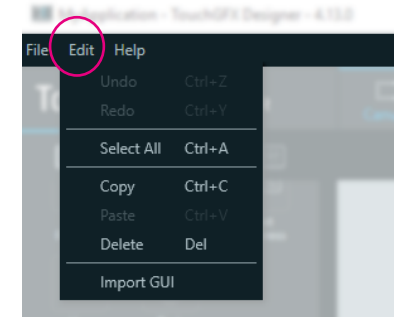
# File Menu

Consists of a File, Edit and Help menu items

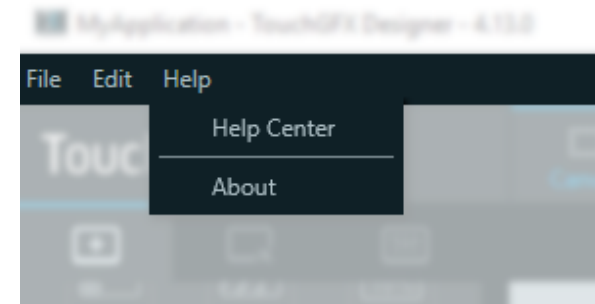
- File Menu used for creating/opening/saving projects
- Edit Menu interacts with UI elements. It can be used to import UI templates or demos after project creation
- Help Menu redirects to official support forum. Also contains Software License Agreement



*File menu item in File Menu*



*Edit menu item in File Menu*

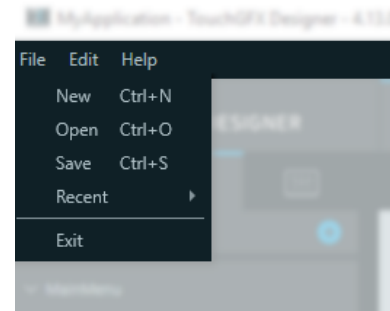


*Help menu item in File Menu*

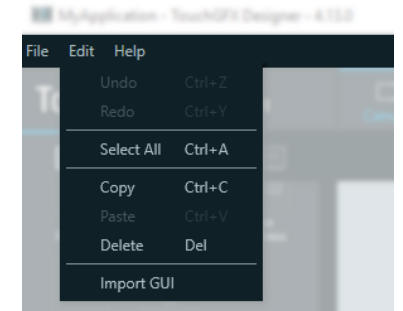
# File Menu

Consists of a File, Edit and Help menu items

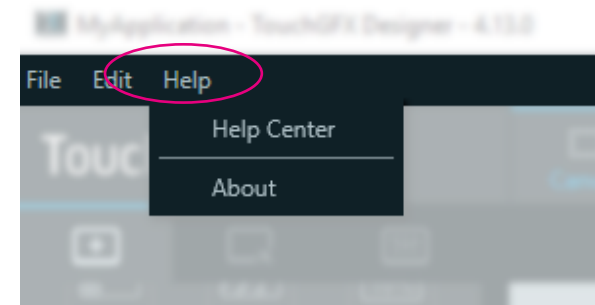
- File Menu used for creating/opening/saving projects
- Edit Menu interacts with UI elements. It can be used to import UI templates or demos after project creation
- Help Menu redirects to official support forum. Also contains Software License Agreement



*File menu item in File Menu*



*Edit menu item in File Menu*



*Help menu item in File Menu*

# Images View

Used to manage images used in a TouchGFX Project

- Images are located under the assets\images folder in the project folder

Contains 3 sections: the tree view on the left, the table view in the middle and the properties view on the right

- *Tree view*: overview of the images and folders
- *Table view*: list of images located in the selected folder
- *Properties view*: properties of an image

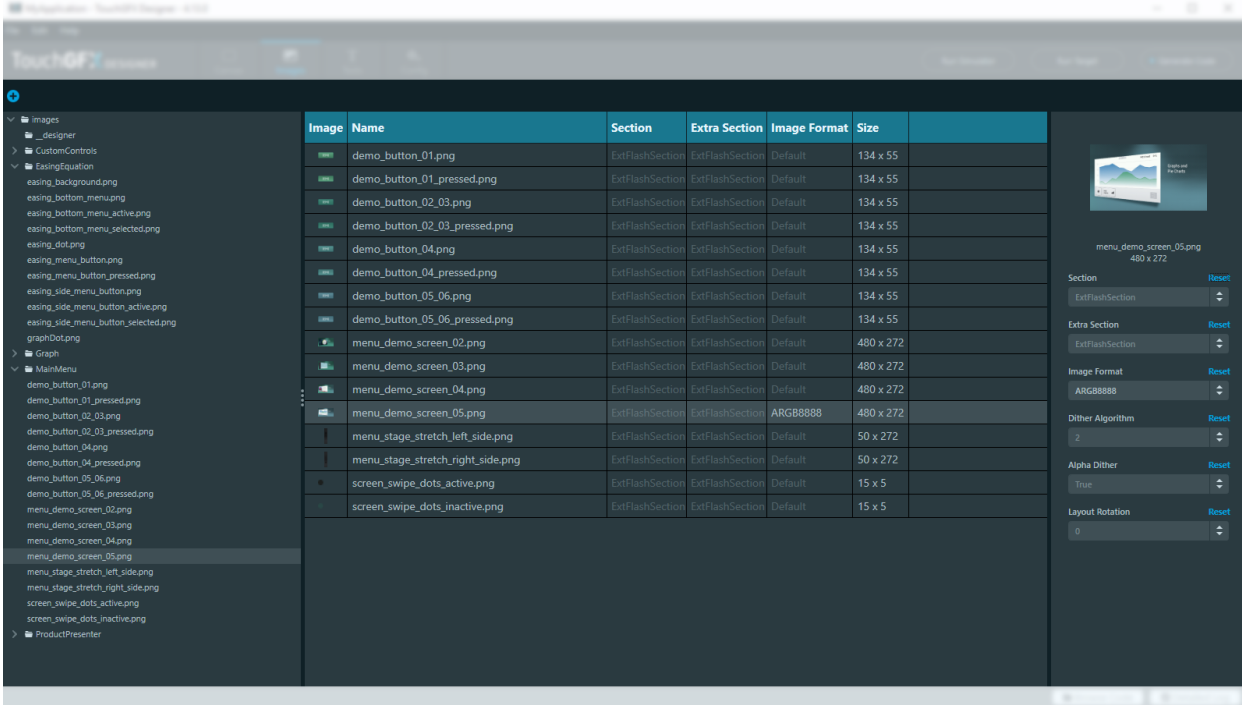


Image View of the UI template "Demo 1"

# Images View

Used to manage images used in a TouchGFX Project

- Images are located under the assets\images folder in the project folder

Contains 3 sections: the tree view on the left, the table view in the middle and the properties view on the right

- *Tree view*: overview of the images and folders
- *Table view*: list of images located in the selected folder
- *Properties view*: properties of an image

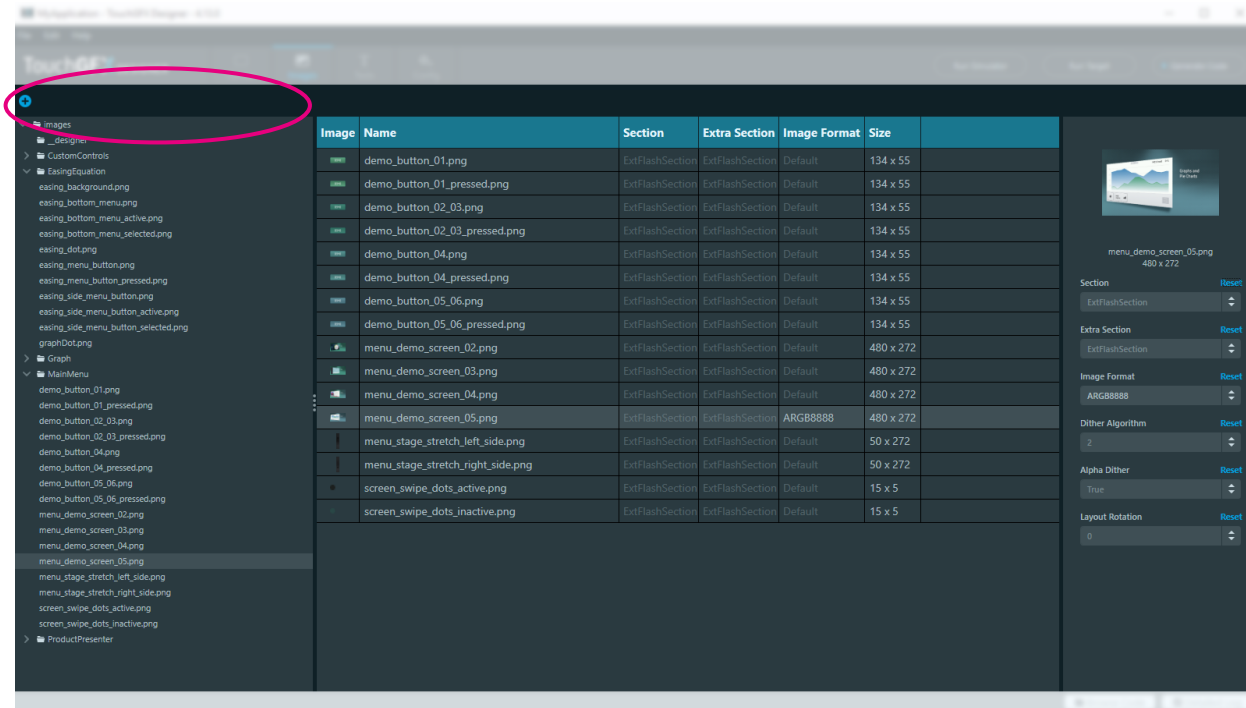


Image View of the UI template "Demo 1"

# Images View

Used to manage images used in a TouchGFX Project

- Images are located under the assets\images folder in the project folder

Contains 3 sections: the tree view on the left, the table view in the middle and the properties view on the right

- *Tree view*: overview of the images and folders
- *Table view*: list of images located in the selected folder
- *Properties view*: properties of an image

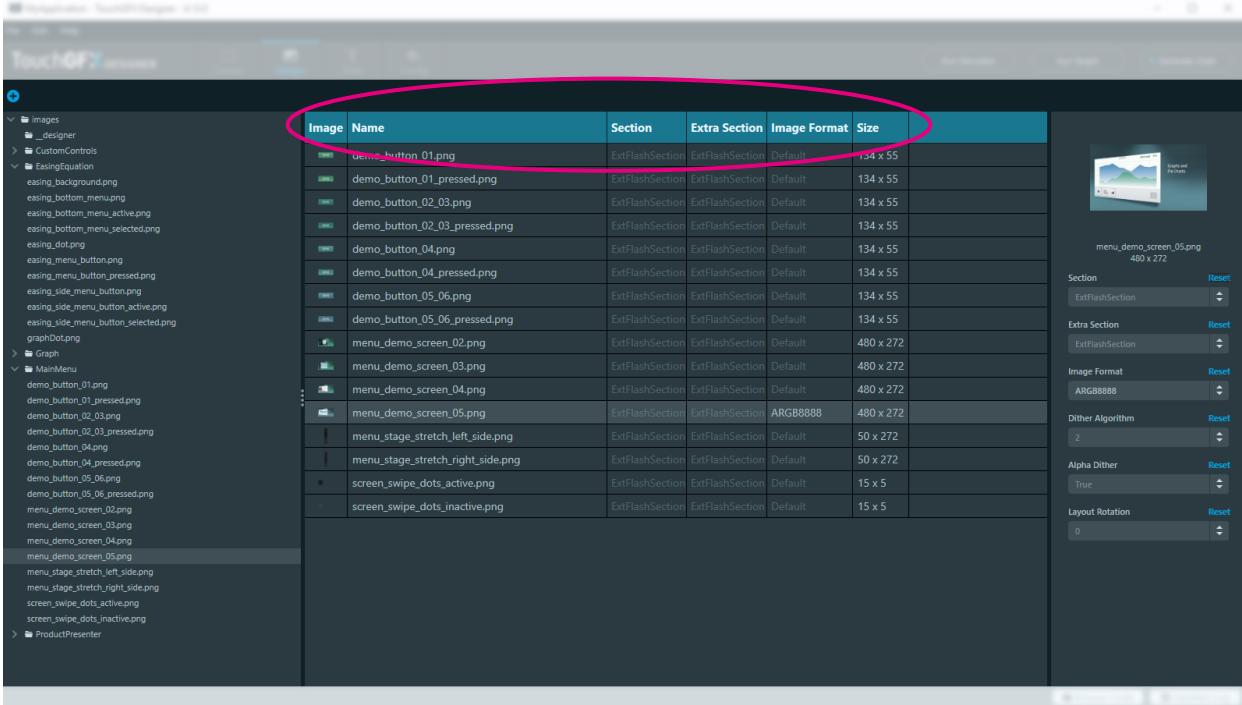


Image View of the UI template "Demo 1"



# Images View

Used to manage images used in a TouchGFX Project

- Images are located under the assets\images folder in the project folder

Contains 3 sections: the tree view on the left, the table view in the middle and the properties view on the right

- *Tree view*: overview of the images and folders
- *Table view*: list of images located in the selected folder
- *Properties view*: properties of an image

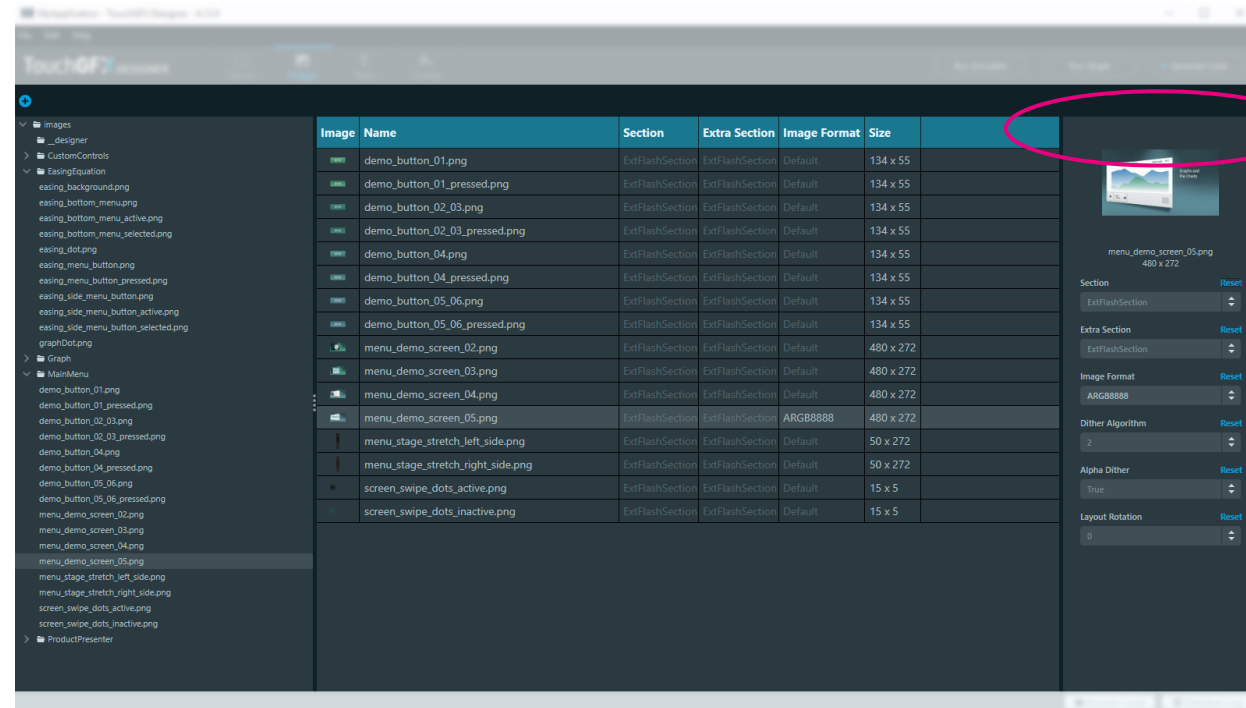
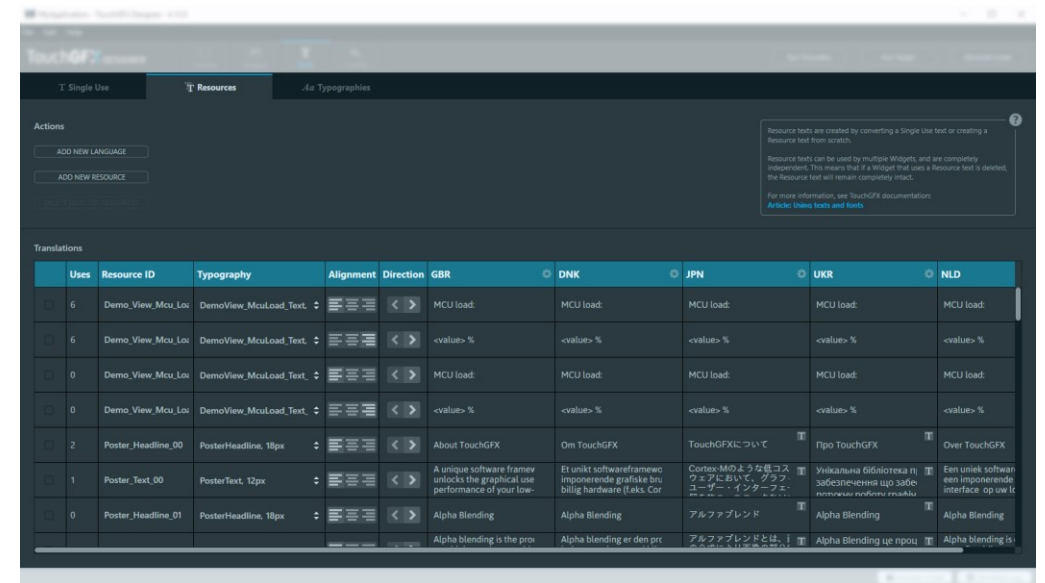


Image View of the UI template "Demo 1"

Used for configuration of texts, translations and typographies

- Single Use and Resources tabs both contain an overview of texts but have different usage



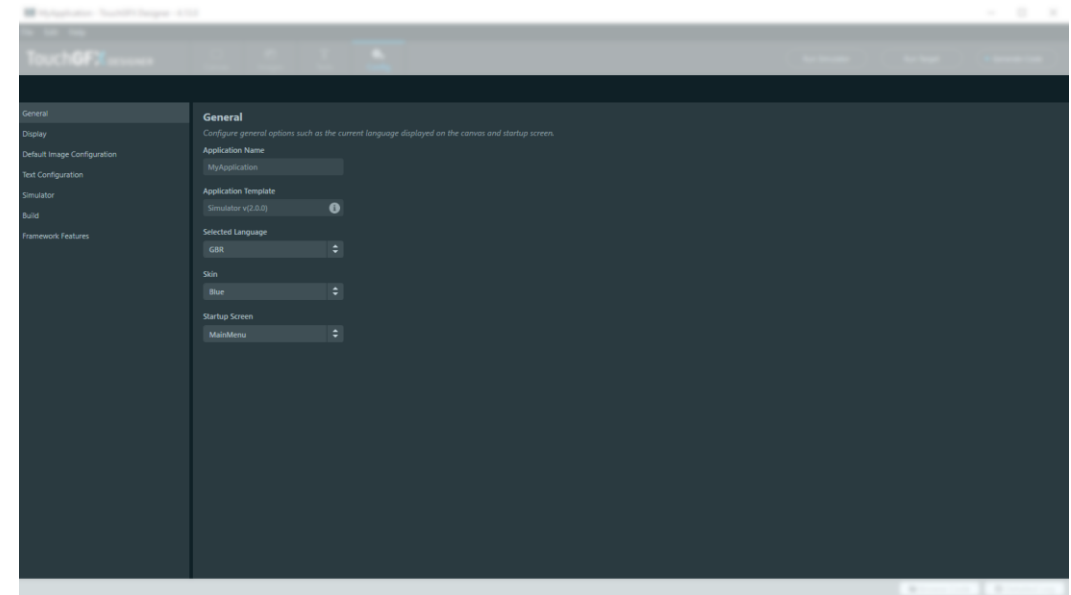
Text View of the UI template "Demo 1"

Working with texts and the Text Views is explained in the "Texts and Fonts" article

# Config View

Various settings affecting the project can be configured in the Config View

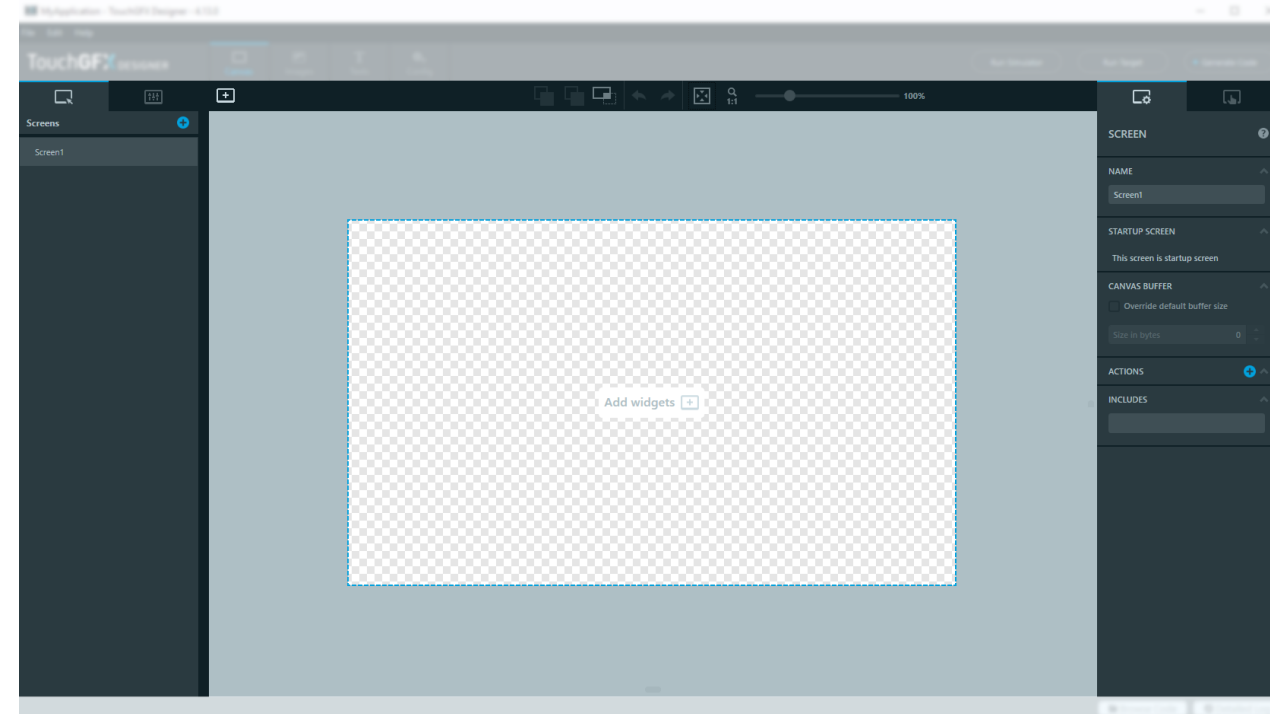
- General settings
  - Application name or startup screen
- Display setting
  - Dimensions, display orientation and color depth
- Default image settings
  - Settings of all images in the project, unless overwritten in the Images View



*Config View*

# Canvas View

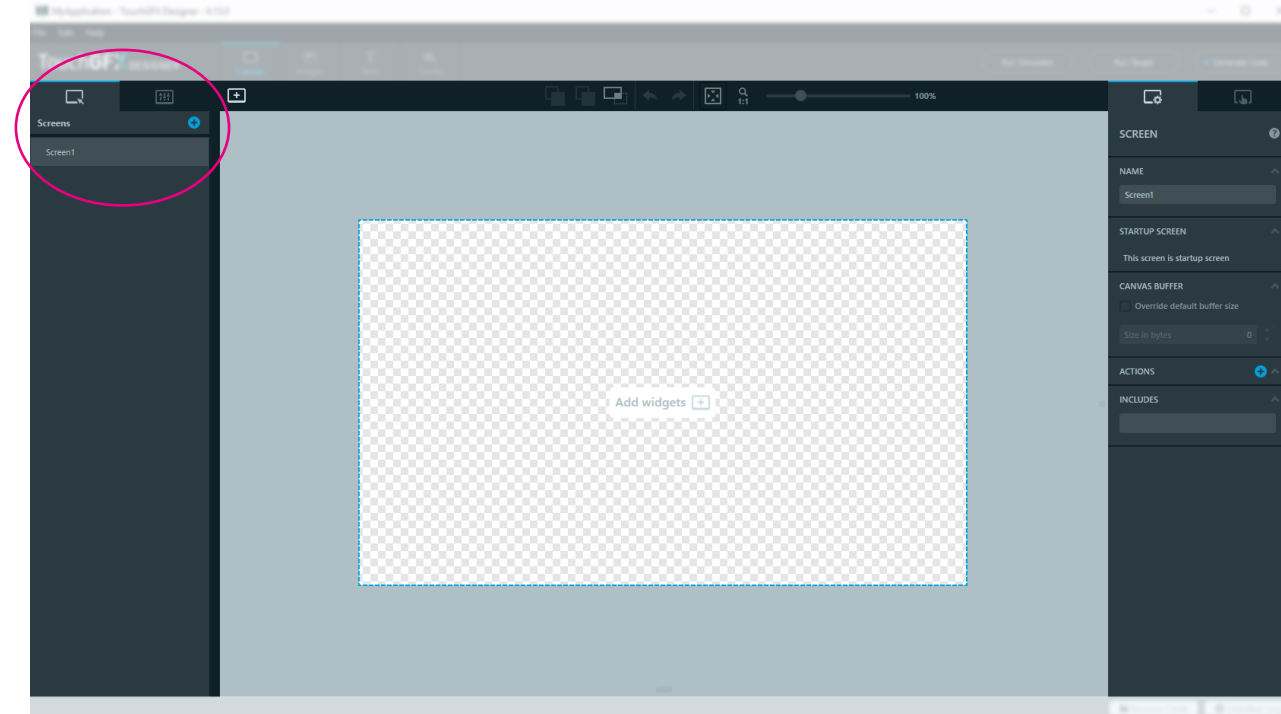
- Landing screen after project creation
  - Visual representation of the selected screen or custom container
- The left bar is where screens and custom containers are configured
  - Additional Screens/Custom containers added by pressing the blue icon
  - Screen and Widget order changed by dragging



*The Canvas View*


# Canvas View

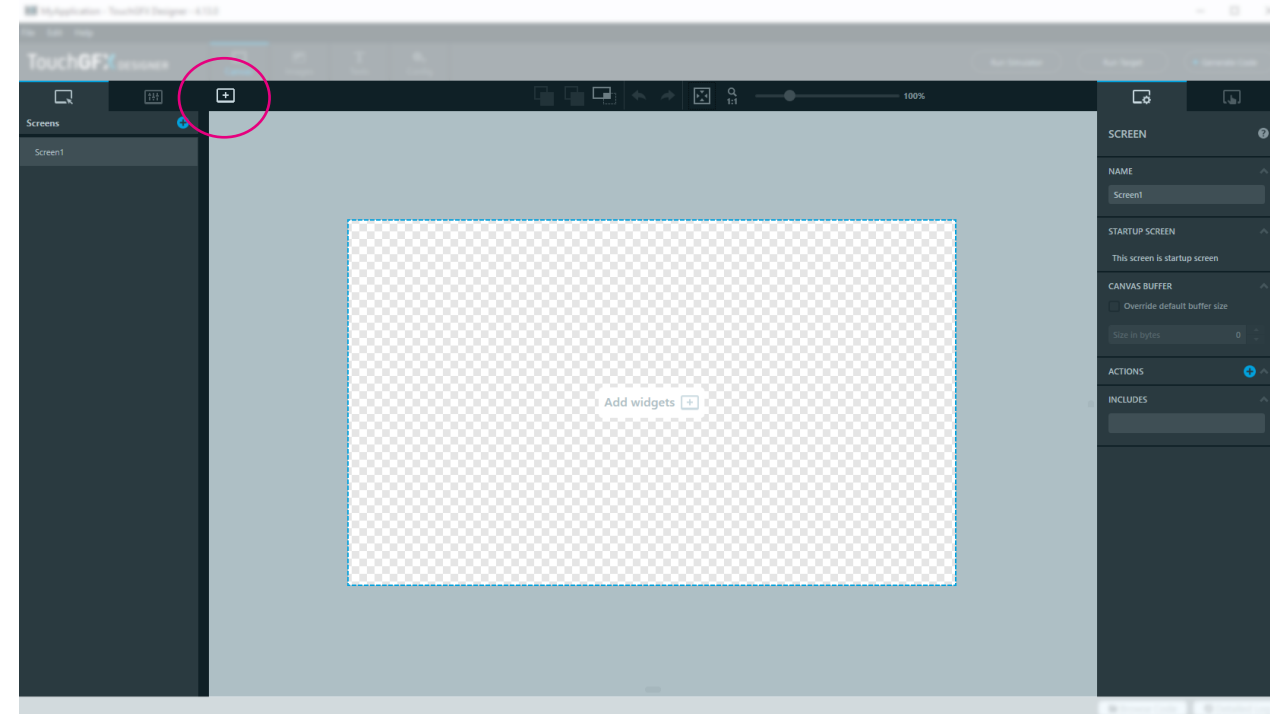
- Landing screen after project creation
  - Visual representation of the selected screen or custom container
- The left bar is where screens and custom containers are configured
  - Additional Screens/Custom containers added by pressing the blue icon
  - Screen and Widget order changed by dragging



*The Canvas View*


# Canvas View

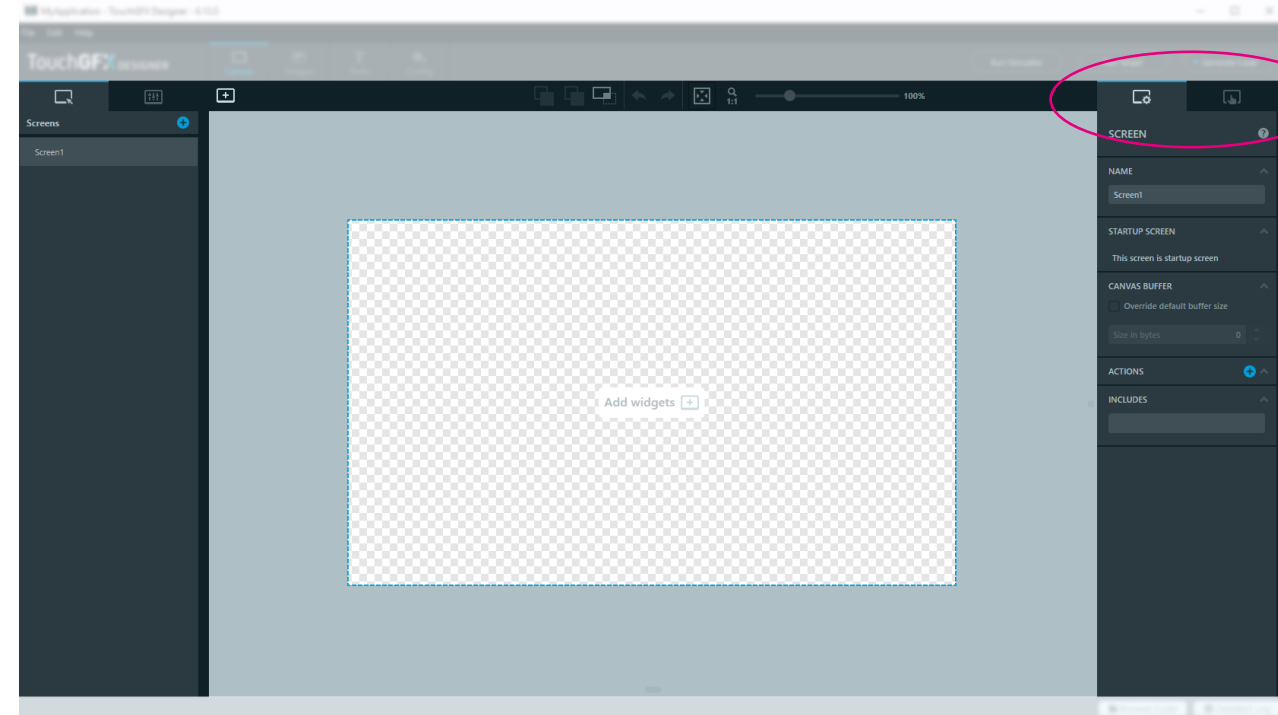
- “Add widget” button  opens the widget menu
  - Contains all available widgets grouped by categories
  - Clicking a widget will add it to the canvas
- The right bar contains two tabs: Properties and Interaction
  - Properties is where widgets and screen properties are set
  - Interaction is where widgets and screen interactions can be created and defined



*The Canvas View*

# Canvas View

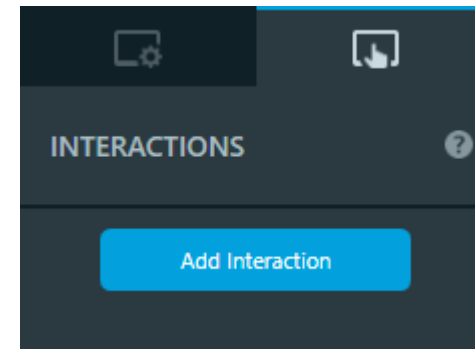
- “Add widget” button  opens the widget menu
  - Contains all available widgets grouped by categories
  - Clicking a widget will add it to the canvas
- The right bar contains two tabs: Properties and Interaction
  - Properties is where widgets and screen properties are set
  - Interaction is where widgets and screen interactions can be created and defined



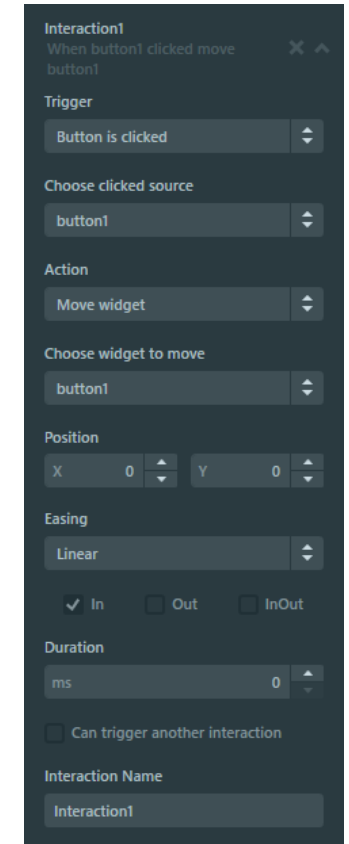
*The Canvas View*

# Interactions

- An Interaction consists of a trigger and an action
  - A Trigger is what will start the interaction
  - An Action is what will happen after a Trigger has been emitted
  - The number of triggers and actions available depends on the widgets added to the project
  - Interactions can be chained



*Interaction tab in TouchGFX Designer*



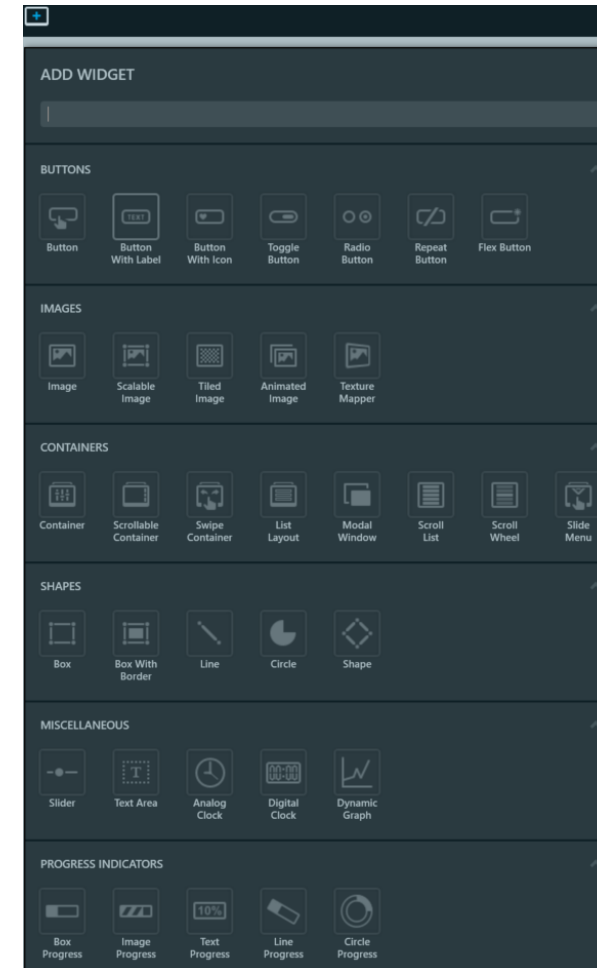
*Moving a widget when a button is pressed*



# UI components

# Widgets

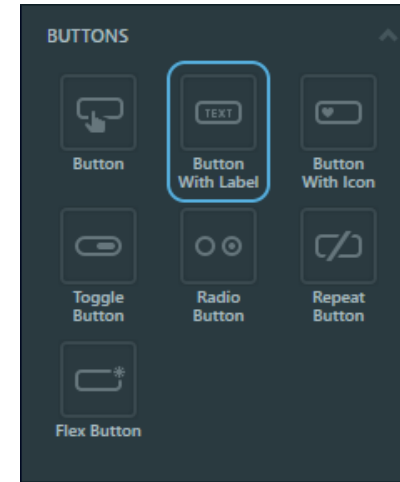
- Wide range of widgets available
- Categories based on properties and usage
  - Custom containers will appear as a new category
- Custom widgets cannot be seen nor configured from Designer



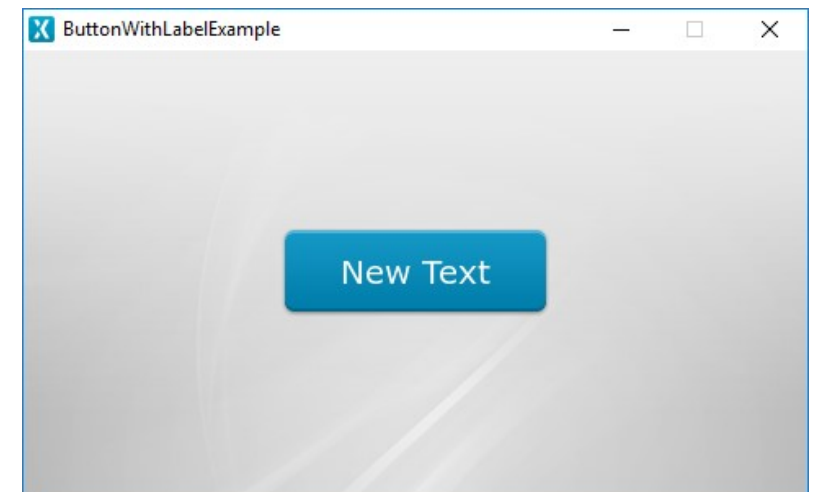
Widget tab

# Button with Label widget

- Widget aware of touch events
  - Sends a callback when released
  - Found in “Buttons” widget group
- “Pressed” and “Released” states
  - Each associated with an image and a text
- Trigger source: “Button is clicked”



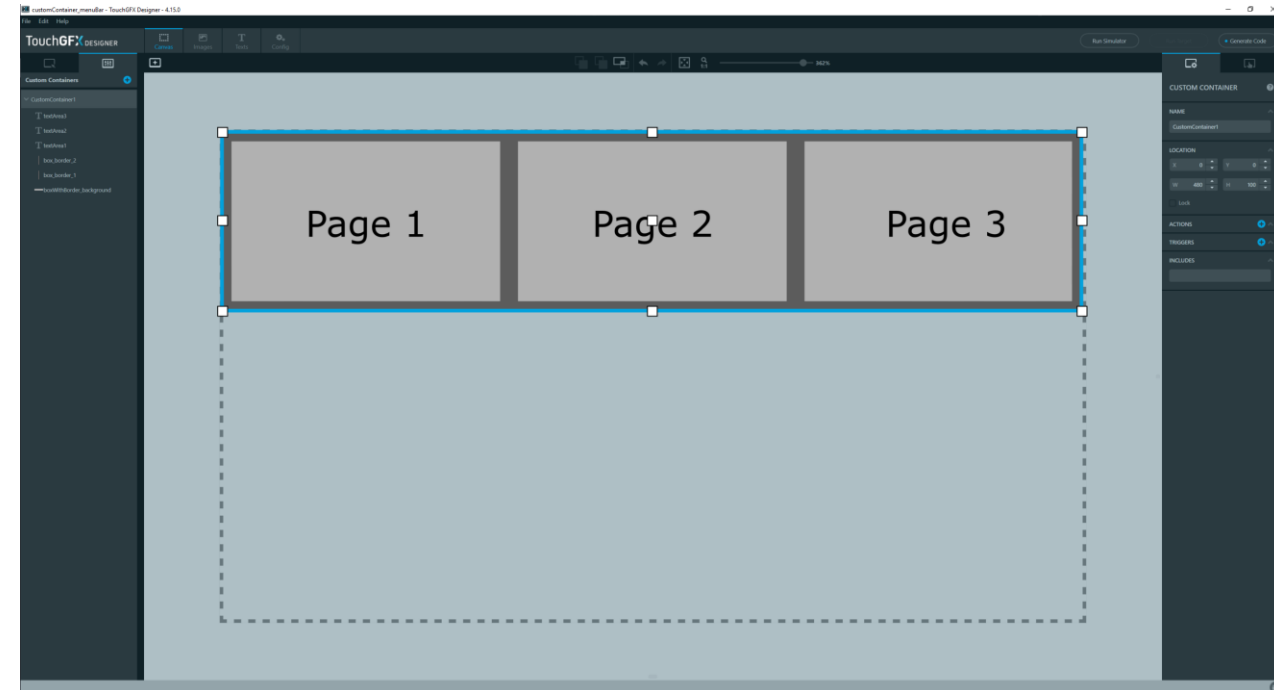
*ButtonWithLabel In TouchGFX Designer*



*ButtonWithLabel widget running in the simulator*

# Custom container

- Wide range of application
  - Up to the user's needs
- Performances dependent on the complexity of the child nodes
- Example: menu bar needed for all the screens of an application



*Custom container widget configuration*

# Working with TouchGFX in User Code

# Working with TouchGFX in User Code

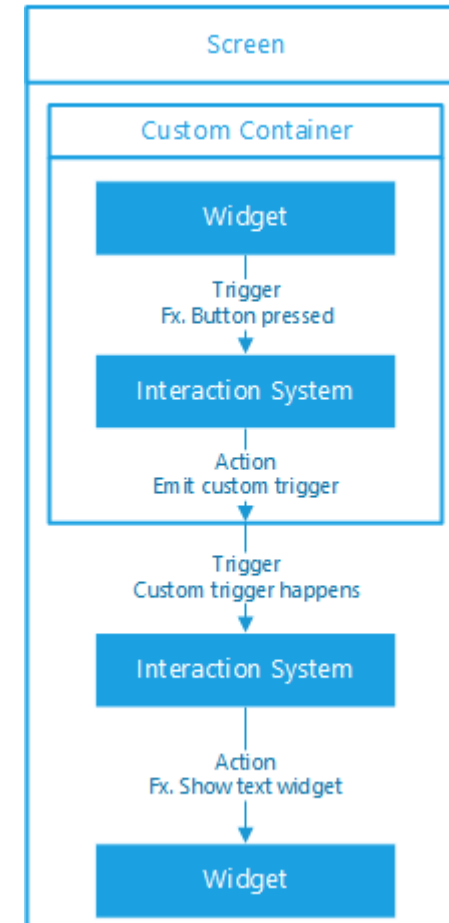
- Actions and Triggers
  - Custom interactions
  - Interface between Designer and User Code
  - Created under screen properties
  - After creation, then added to the interaction system

The screenshot shows the 'TRIGGERS' configuration panel. At the top, there is a header 'TRIGGERS' with a blue plus icon and a close icon. Below the header, the configuration for a trigger named 'trigger1' is shown. It includes a 'bool' type indicator, a 'Name' field with the value 'trigger1', a 'Description' field, and a 'Type' dropdown menu with 'bool' selected and a checkmark.

The screenshot shows the 'ACTIONS' configuration panel. At the top, there is a header 'ACTIONS' with a blue plus icon and a close icon. Below the header, the configuration for an action named 'action1' is shown. It includes a 'Name' field with the value 'action1', a 'Description' field, and a 'Type' dropdown menu with an empty selection and a checkmark.

# Working with TouchGFX in User Code

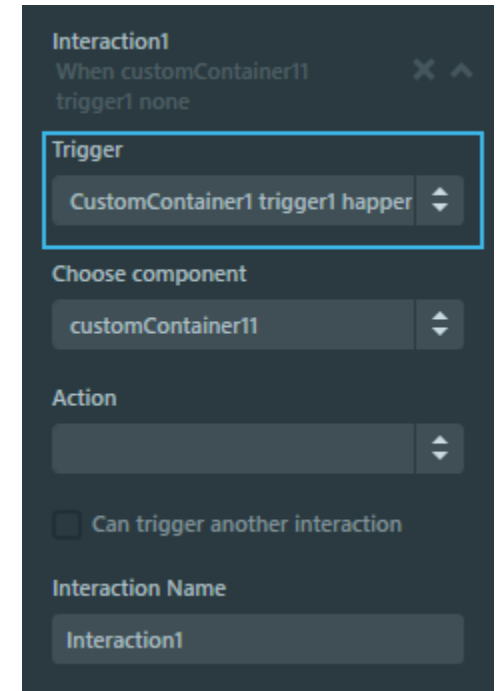
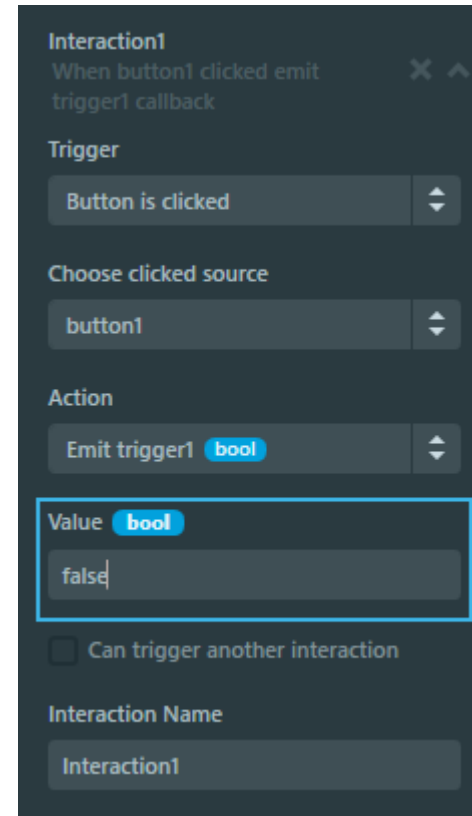
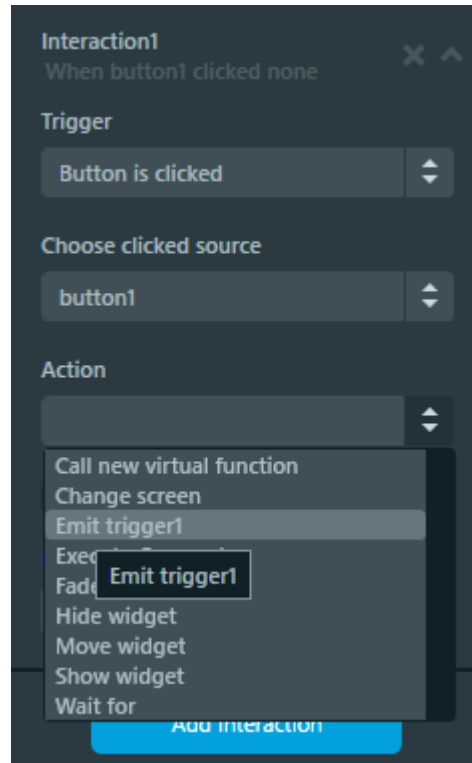
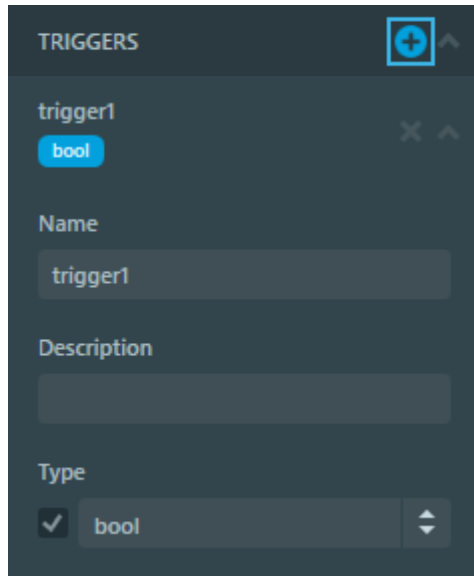
- Triggers
  - Enable a Screen to react on events from a Custom Container
    - Custom Container can emit a custom Trigger as an Action
    - Enable a Screen to react to a custom Trigger from a Custom Container
    - Able to pass data from Custom Container to Screen
  - Generated as C++ callback
  - After creation, the trigger is added to the interaction system
  - Can be emitted from User Code



*Flow for a Custom trigger between a widget in a custom container and a widget in a screen*

# Working with TouchGFX in User Code

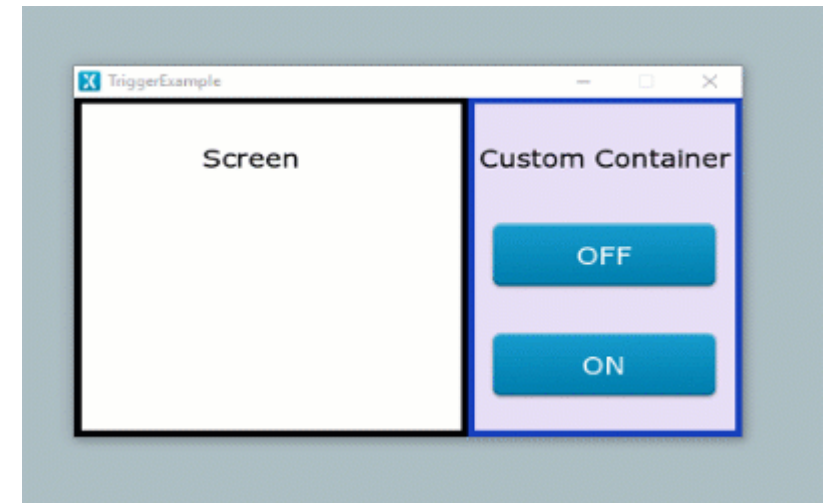
- Creating and using a Trigger





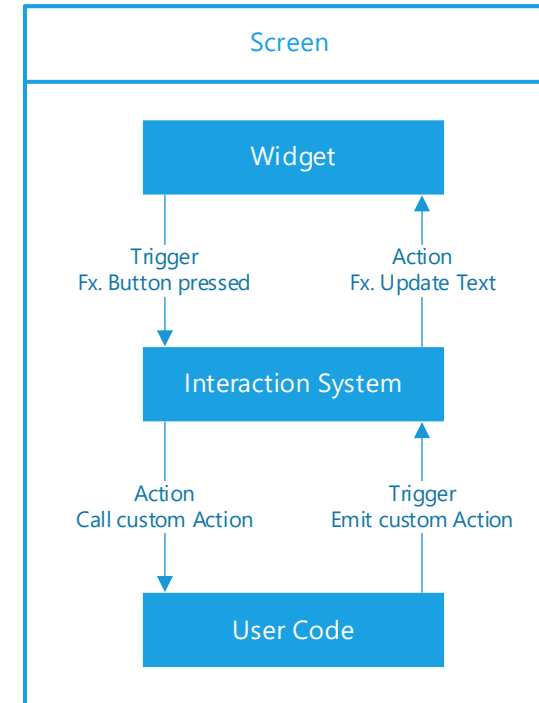
# Working with TouchGFX in User Code

- Triggers – Example
  - Custom Container added to a Screen
  - Pressing ON or OFF emits a ON or OFF Custom Trigger
  - Based on the Custom Trigger, the ON or OFF text will show



# Working with TouchGFX in User Code

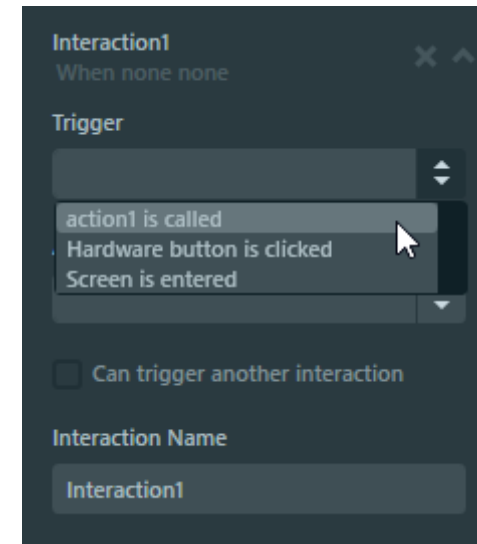
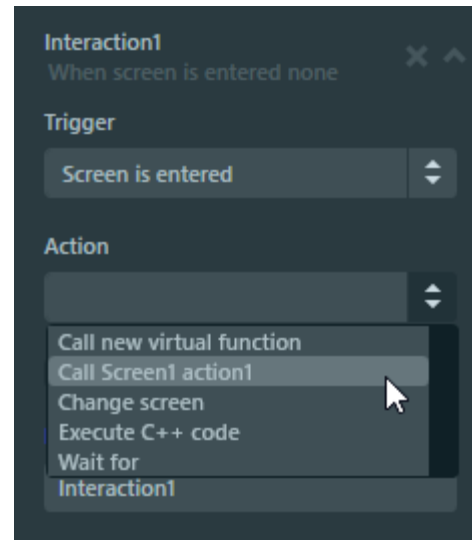
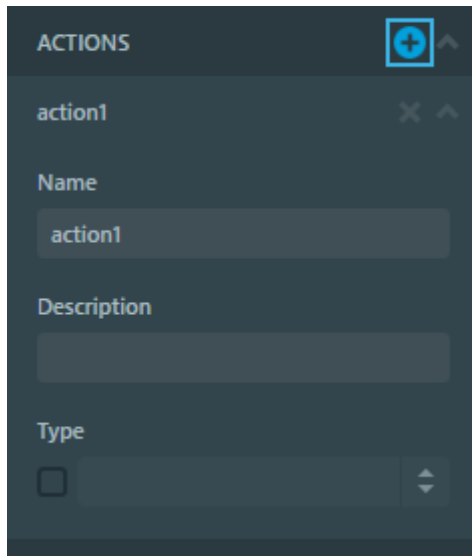
- Actions
  - Interface between Screen/Custom Container and User Code
    - The Interaction System can call Custom Actions in User Code as Actions
    - User code can emit Custom Actions as Triggers via the Interaction System
    - Able to pass data from Screen or Custom Container to User Code
  - Generated as a virtual C++ methods
  - The action is added to the interaction system



*Relations between Widgets, Interactions System and User Code, when using Custom Actions*

# Working with TouchGFX in User Code

- Actions: Designer Setup Example



# Working with TouchGFX in User Code

- Actions: User Code Example

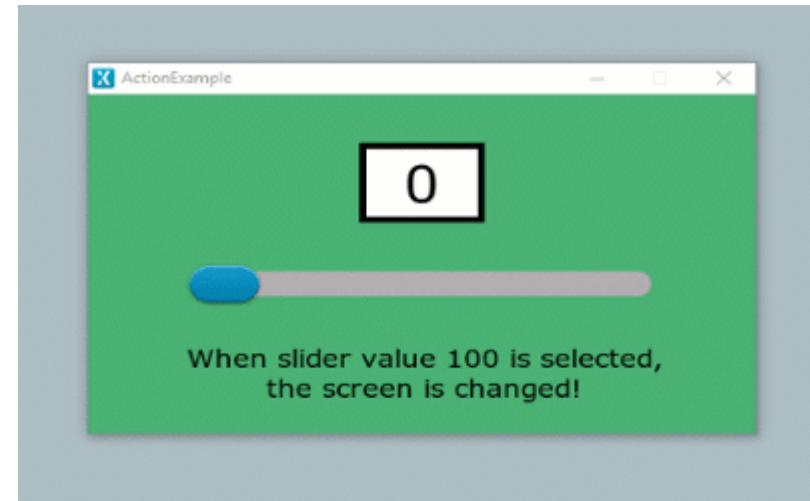
```
Screen1View.cpp  X
-> Screen1View.setupScreen  v  -> void Screen1View::setupScreen()
1  #include <gui/screen1_screen/Screen1View.hpp>
2
3  Screen1View::Screen1View()
4  {
5  }
6
7
8  void Screen1View::setupScreen()
9  {
10     Screen1ViewBase::setupScreen();
11     action1();
12 }
13
```

```
Screen1View.hpp  X  Screen1View.cpp
-> Screen1View  v  -> class Screen1View : public Screen1ViewBase
1  #ifndef SCREEN1VIEW_HPP
2  #define SCREEN1VIEW_HPP
3
4  #include <gui_generated/screen1_screen/Screen1ViewBase.hpp>
5  #include <gui/screen1_screen/Screen1Presenter.hpp>
6
7  class Screen1View : public Screen1ViewBase
8  {
9  public:
10     Screen1View();
11     virtual ~Screen1View() {}
12     virtual void setupScreen();
13     virtual void tearDownScreen();
14
15     virtual void action1();
16 protected:
17 };
18
19 #endif // SCREEN1VIEW_HPP
20
```

```
Screen1View.hpp  Screen1View.cpp  X
-> Screen1View.action1  v  -> virtual void action1()
1  #include <gui/screen1_screen/Screen1View.hpp>
2
3  Screen1View::Screen1View()
4  {
5  }
6
7
8  void Screen1View::setupScreen()
9  {
10     Screen1ViewBase::setupScreen();
11 }
12
13 void Screen1View::tearDownScreen()
14 {
15     Screen1ViewBase::tearDownScreen();
16 }
17
18 void Screen1View::action1()
19 {
20 }
21
22
```

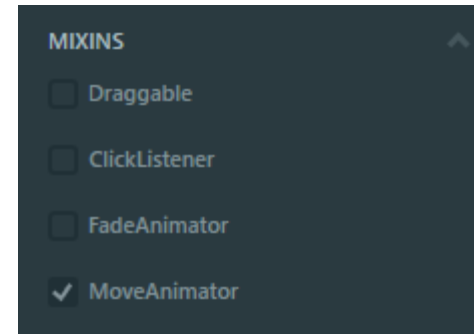
# Working with TouchGFX in User Code

- Actions: Demo
  - When the slider is dragged, the value of the slider is passed with an Action to User Code, which updates the text in the box
  - When the finger (mouse) releases the slider, another action sends the confirmed value to the User Code via an Action
  - If the value is 100 a Custom Action is sent from user code, to changes screen



# Working with TouchGFX in User Code

- Mixins
  - Extend Widgets functionality
  - Mixin functionalities are used in User Code
  - Four different Mixins
    - Move Animator
    - Fade Animator
    - Draggable
    - Click Listener
  - Can be added to a widget in the Designer
  - Container Widgets cannot use Fade Animator

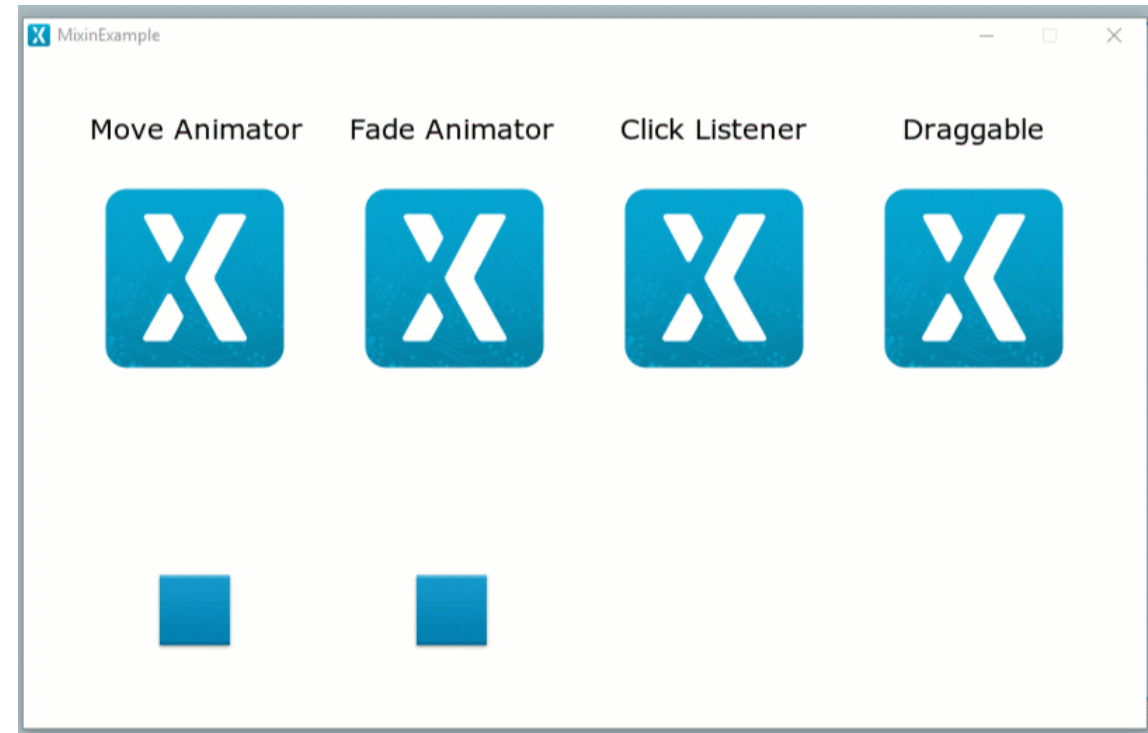


```
touchgfx::MoveAnimator< touchgfx::Box > box;
```

*The Mixins properties in the TouchGFX Designer (top)  
The Mixin Move Animator added to a Widget in User  
Code (bottom)*

# Working with TouchGFX in User Code

- Mixin – Example
  - The four images, has the four Mixins added to them, denoted by the text above them.



- Everything done in the Designer, can also be done in User Code
  - But the Designer can help you with a lot of things
- Avoid going back and forth between User Code and Generated Code
- Remember to utilize the MVP Pattern
- Inspect Generated code
- Reuse code from examples
- [The TocuhGFX API: Button](#)



- Generated Code from Animated Image Example

```
MainViewBase::MainViewBase() :
    buttonCallback(this, &MainViewBase::buttonCallbackHandler),
    animationEndedCallback(this, &MainViewBase::animationEndedCallbackHandler)
{
    __background.setPosition(0, 0, 480, 272);
    __background.setColor(touchgfx::Color::getColorFrom24BitRGB(0, 0, 0));

    boxBackground.setPosition(0, 0, 800, 480);
    boxBackground.setVisible(false);
    boxBackground.setColor(touchgfx::Color::getColorFrom24BitRGB(0, 0, 0));

    imgBackground.setXY(0, 0);
    imgBackground.setBitmap(touchgfx::Bitmap(BITMAP_BG_ID));

    animation.setXY(161, 18);
    animation.setBitmaps(BITMAP_ANI_01_ID, BITMAP_ANI_14_ID);
    animation.setUpdateTicksInterval(2);
    animation.setDoneAction(animationEndedCallback);

    btnToggle.setXY(175, 195);
    btnToggle.setBitmaps(touchgfx::Bitmap(BITMAP_BTN_ID), touchgfx::Bitmap(BITMAP_BTN_PRESSED_ID));
    btnToggle.setLabelText(touchgfx::TypedText(T_TEXTSTART));
    btnToggle.setLabelColor(touchgfx::Color::getColorFrom24BitRGB(213, 115, 0));
    btnToggle.setLabelColorPressed(touchgfx::Color::getColorFrom24BitRGB(213, 115, 0));
    btnToggle.setAction(buttonCallback);

    add(__background);
    add(boxBackground);
    add(imgBackground);
    add(animation);
    add(btnToggle);
}
```

```
void MainViewBase::buttonCallbackHandler(const touchgfx::AbstractButton& src)
{
    if (&src == &btnToggle)
    {
        //buttonClicked
        //When btnToggle clicked execute C++ code
        //Execute C++ code
        if (animation.isAnimatedImageRunning())
        {
            animation.pauseAnimation();
            btnToggle.setLabelText(TypedText(T_TEXTSTART));
        }
        else
        {
            animation.startAnimation(animation.isReverse(), false, true);
            btnToggle.setLabelText(TypedText(T_TEXTSTOP));
        }
    }
}
```

# Where to go next?

- More information can be found in the UI development section:  
[UI Development](#)
- Get started on UI developing with the Hands-on Workshop:  
[UI Development - Getting Started](#)
- Or by following our tutorials:  
[Tutorials](#)

# Thank you

© STMicroelectronics - All rights reserved.

ST logo is a trademark or a registered trademark of STMicroelectronics International NV or its affiliates in the EU and/or other countries.

For additional information about ST trademarks, please refer to [www.st.com/trademarks](http://www.st.com/trademarks).

All other product or service names are the property of their respective owners.



life.augmented

For further support and content,  
**you can visit the TouchGFX online  
documentation site**  
<https://support.touchgfx.com>

