

TouchGFX | Dynamic Graphs with Slide Menu | Demo

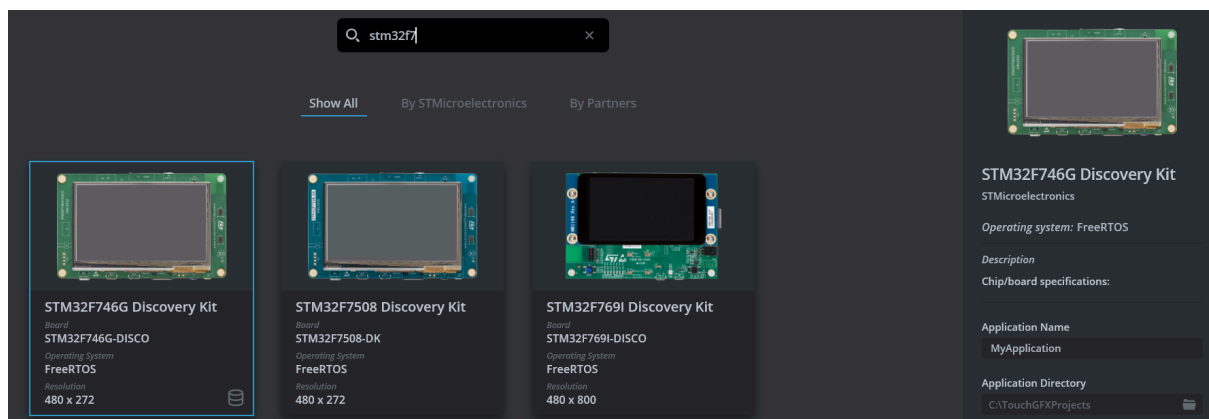
This project showcases how to build two dynamic graphs using TouchGFX Designer and STM32CubeIDE, with toggle buttons to control the visibility of lines, dots, background, and gridlines. A slide menu allows users to expand or collapse the control panel, making it easy to customize the graph display.

Random values between -20 and 100 simulate sensor data for real-time updates.

Project Setup

1. Create a New Application

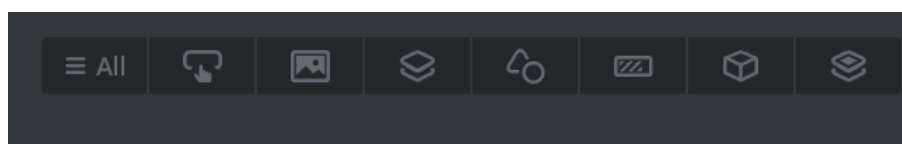
- Launch **TouchGFX Designer**.
- Select your **target board**.
- Click **Create** to start a new project.



Create

2. Add UI Elements

- Add a **scalable image** from your file system or the GFX stock.
- Add **two dynamic graphs** (`dynamicGraphBlue`, `dynamicGraphGreen`).
- Add **six toggle buttons** to control visibility: lines, dots, box and horizontal gridlines.



Images

- Image
- Scalable Image
- Tiled Image
- Animated Image
- Texture Mapper
- SVG Image
- QR Code

Miscellaneous

- Slider
- Text Area
- Analog Clock
- Digital Clock
- Dynamic Graph
- Static Graph
- Gauge
- Video

Buttons

- Button
- Button With Label
- Button With Icon
- Toggle Button
- Radio Button
- Repeat Button
- Flex Button

DynamicGraphSlideMenu

- slideMenu1
 - dynamicGraphBlue
 - dynamicGraphGreen
 - dynamicGraph0
 - box1
 - background

DynamicGraphSlideMenu

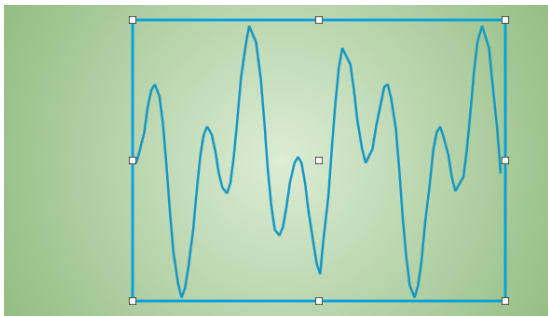
- slideMenu1
 - toggleGrid
 - toggleBox
 - toggleGreenDots
 - toggleGreenLine
 - toggleBlueDots
 - toggleBlueLine

Configure Dynamic Graphs

Graph Properties

For each graph (e.g., `dynamicGraphBlue`):

- **Graph Area Margins & Padding:** Adjust layout spacing.
- **Dynamic Behavior:** Set to `Scroll` to continuously update X-axis values as new data arrives.
- **Number of Data Points:** Defines how many points are visible on the X-axis at a time.
- **Value Range:** Set Y-axis range to `-20` to `100` (constant).
- **Elements:** Add both `Line` and `Dots`.
- **Grid Lines:** Enable horizontal grid lines.
- **Axis Labels:** Add labels for both X and Y axes.



Graph Area Margin

Top 0	Bottom 20
Left 20	Right 10

Graph Area Padding

Top 10	Bottom 10
Left 20	Right 0

Data Points

Dynamic Behavior

☐ ☒ ☐

Number of Data Points

Data Points 31

Value Range

Min -20 Max 100

Elements

☒ Line

☒ Dots

Horizontal Grid Lines

☒ Major Division

☐ Minor Division

X-Axis Labels

☒ Major Labels

☐ Minor Labels

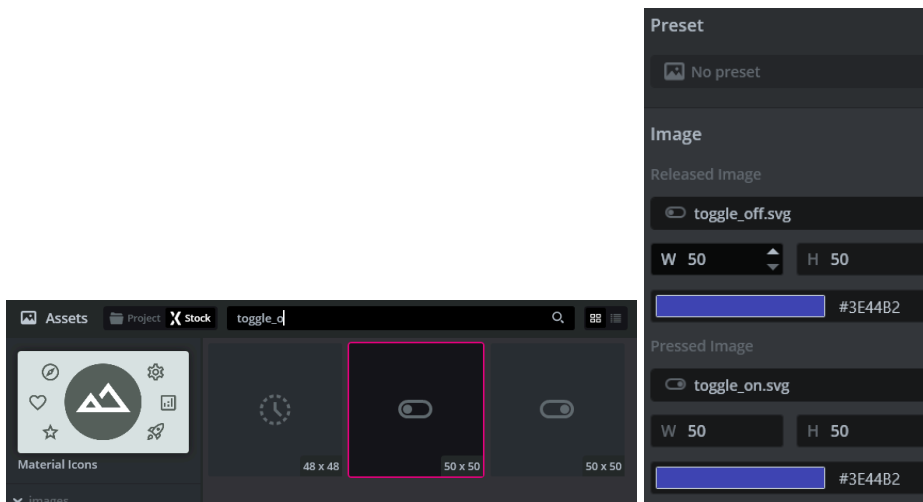
Y-Axis Labels

☒ Major Labels

☐ Minor Labels


Configure Toggle Buttons

- Use stock images: `toggle_off` and `toggle_on`.
- Assign each toggle to control visibility of either line or dots for each graph.



Configure Slide Menu

PropertiesInteractions

 slideMenu1

Location

X0Y0

W75H280

☐ Lock☒ Visible

Expanding Direction

East

Expanding Direction

East

State

Expanded

Collapsed: Visible Pixels26

Expanded: Hidden Pixels0

Expanded Timeout5000

Background

Background Image

slideMenuBG_90_272.png

Background Location

X0Y0

Button

☒ Use Button

Released Image

menu.svg

W25H30

#879AE0

Pressed Image

menu.svg

W25H30

#5270DE

Button Location

X50Y120

Animation

Easing

Cubic

Easing Option

In

Out

InOut

Duration (ms)300

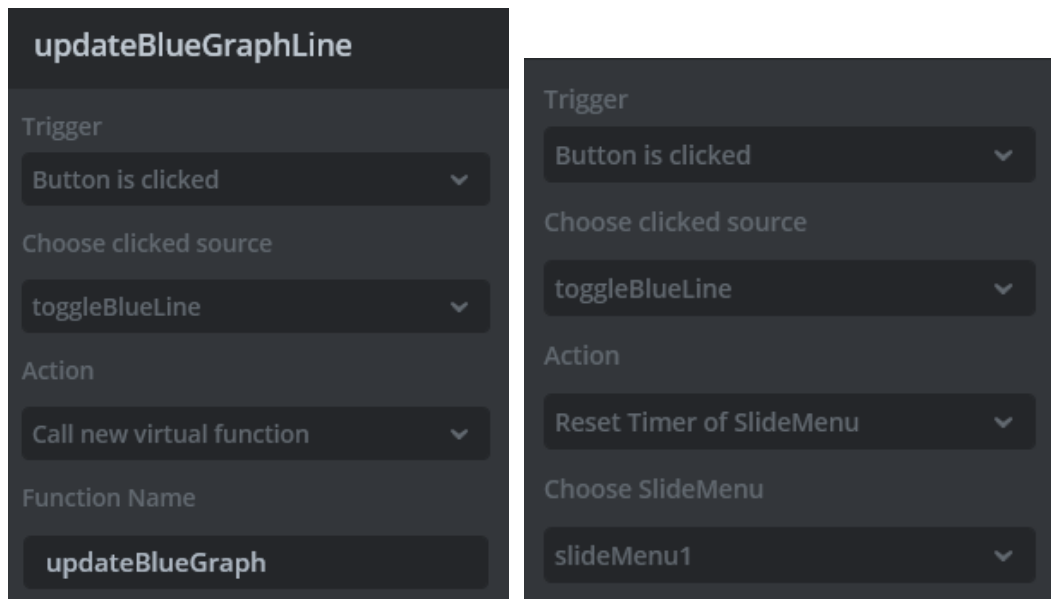
Add Interactions

Here we have two groups of interactions: interactions toggle/graph and toggle/slide menu. In fact, the slide menu is shown for 5 seconds before collapse. This timer is reset everytime user touches a toggle button.

In the generated code we can see the result of the interactions defined in the Designer.

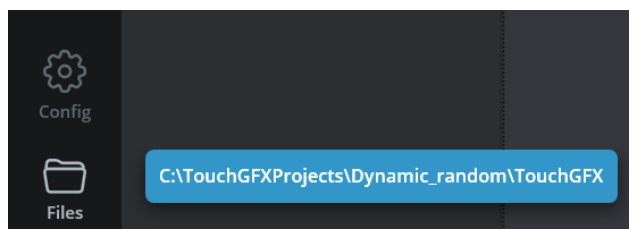
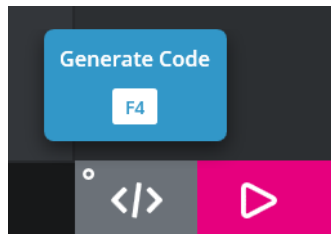
"C:\TouchGFXProjects\Dynamic_random\TouchGFX\generated\gui_generated\src\dynamicgraphs\slidemenu_screen\DynamicGraphSlideMenuViewBase.cpp"

```
void DynamicGraphSlideMenuViewBase::buttonCallbackHandler(const
touchgfx::AbstractButton& src)
{
    if (&src == &toggleBlueLine)
    {
        updateBlueGraph();
        slideMenu1.resetExpandedStateTimer();
    }
    if (&src == &toggleBlueDots)
    {
        updateBlueGraph();
        slideMenu1.resetExpandedStateTimer();
    }
    if (&src == &toggleGreenLine)
    {
        updateGreenGraph();
        slideMenu1.resetExpandedStateTimer();
    }
    if (&src == &toggleGreenDots)
    {
        updateGreenGraph();
    }
    if (&src == &toggleBox)
    {
        customizeScreen();
        slideMenu1.resetExpandedStateTimer();
    }
    if (&src == &toggleGrid)
    {
        customizeScreen();
        slideMenu1.resetExpandedStateTimer();
    }
}
```



Save and Generate Code

- Click **Save** and **Generate Code**.
- Open the project in STM32CubeIDE:
C:\TouchGFXProjects\Dynamic_random\STM32CubeIDE\cproject



"C:\TouchGFXProjects\Dynamic_random\STM32CubeIDE\cproject"

Code

DynamicGraphSlideMenuView::DynamicGraphSlideMenuView()

This constructor initializes the view object for the dynamic graph slide menu. It sets the tick counter and last recorded values for the blue and green graphs to zero. The call to `srand(tickCounter)` seeds the random number generator.

```
DynamicGraphSlideMenuView::DynamicGraphSlideMenuView()  
    : tickCounter(0), lastBlueValue(0), lastGreenValue(0), last0Value(0)  
{  
    srand(tickCounter);  
}
```

setupScreen()

This method prepares the screen when the view is activated. It calls the base class setup routine and configures the initial states of various toggle buttons that control the visibility of graph elements. Specifically, it enables the blue and green lines and dots, while disabling the box and grid overlays. It then updates the blue and green graphs to reflect these settings and applies additional visual customizations through `customizeScreen()`. This ensures that the UI is initialized with a consistent and predictable appearance.

```
void DynamicGraphSlideMenuView::setupScreen()  
{  
    DynamicGraphSlideMenuViewBase::setupScreen();  
    toggleBlueLine.forceState(true);  
    toggleBlueDots.forceState(true);  
    toggleGreenLine.forceState(true);  
    toggleGreenDots.forceState(true);  
    toggleBox.forceState(false);  
    toggleGrid.forceState(false);  
    updateBlueGraph();  
    updateGreenGraph();  
    customizeScreen();  
}
```

tearDownScreen()

This function is called when the view is being deactivated or destroyed. It delegates the

teardown process to the base class, ensuring that any resources or UI elements are properly cleaned up. While it doesn't contain custom logic, it maintains the structure needed for potential future extensions.

```
void DynamicGraphSlideMenuView::tearDownScreen()  
{  
    DynamicGraphSlideMenuViewBase::tearDownScreen();  
}
```

handleTickEvent()

This function is triggered periodically, likely by a timer or system tick. It increments the internal tick counter and, every tenth tick, generates new data points for three graphs—blue, green, and a third unnamed graph (dynamicGraph0). The data points are generated using the generateDeltaValue() method, which simulates fluctuating values within a bounded range. This mechanism drives the dynamic behavior of the graphs, making them appear responsive and animated over time.

```
void DynamicGraphSlideMenuView::handleTickEvent()  
{  
    tickCounter++;  
    if (tickCounter % 10 == 0)  
    {  
        dynamicGraphBlue.addDataPoint(generateDeltaValue(lastBlueValue, -20, 100, 5));  
        dynamicGraphGreen.addDataPoint(generateDeltaValue(lastGreenValue, -20, 100, 5));  
        dynamicGraph0.addDataPoint(generateDeltaValue(last0Value, -20, 100, 5));  
    }  
}
```

generateDeltaValue()

This utility function produces a pseudo-random value based on a previous value, constrained by a specified minimum, maximum, and maximum delta. It calculates a random delta within the range $[-\text{maxDelta}, +\text{maxDelta}]$, applies it to the last value, and clamps the result to stay within the defined bounds. The updated value is stored back into the reference and returned. This function is central to simulating realistic, bounded fluctuations in graph data.

```

int16_t DynamicGraphSlideMenuView::generateDeltaValue(int16_t& lastValue,
int16_t min, int16_t max, int16_t maxDelta)
{
    int16_t delta = (rand() % (2 * maxDelta + 1)) - maxDelta; // Range: [-
maxDelta, +maxDelta]
    int16_t newValue = lastValue + delta;

    // Clamp to min/max
    if (newValue < min) newValue = min;
    if (newValue > max) newValue = max;

    lastValue = newValue;
    return newValue;
}

```

updateLineVisibility()

This method adjusts the visibility of a line graph element based on the state of a toggle button. If the toggle is active, the line is fully opaque; otherwise, it becomes invisible. The `invalidate()` call ensures that the UI is refreshed to reflect the change. This function allows users to control which graph components are visible on the screen.

```

void DynamicGraphSlideMenuView::updateLineVisibility(touchgfx::ToggleButton&
toggle, touchgfx::GraphElementLine& line)
{
    line.setAlpha(toggle.getState() ? 255 : 0);
    line.invalidate();
}

```

updateDotsVisibility()

Similar to `updateLineVisibility()`, this function manages the visibility of dot elements in the graph.

```

void DynamicGraphSlideMenuView::updateDotsVisibility(touchgfx::ToggleButton&
toggle, touchgfx::GraphElementDots& dots)
{
    dots.setAlpha(toggle.getState() ? 255 : 0);
    dots.invalidate();
}

```

updateGraphElements()

This composite function updates both line and dot visibility for a given graph, using their

respective toggle buttons. After adjusting the visibility, it invalidates the graph to ensure the changes are rendered. It encapsulates the logic for synchronizing multiple visual elements tied to a single graph, promoting code reuse and consistency.

```
void DynamicGraphSlideMenuView::updateGraphElements(touchgfx::ToggleButton& lineToggle,
                                                    touchgfx::GraphElementLine& line,
                                                    touchgfx::ToggleButton& dotsToggle,
                                                    touchgfx::GraphElementDots& dots,
                                                    touchgfx::GraphScroll<31>& graph)
{
    updateLineVisibility(lineToggle, line);
    updateDotsVisibility(dotsToggle, dots);
    graph.invalidate();
}
```

updateBlueGraph()

This function updates the blue graph by invoking `updateGraphElements()` with the appropriate toggles and graphical components. It then invalidates the graph to apply the changes. It serves as a dedicated handler for managing the blue graph's visual state.

```
void DynamicGraphSlideMenuView::updateBlueGraph()
{
    updateGraphElements(toggleBlueLine, dynamicGraphBlueLine1,
                        toggleBlueDots, dynamicGraphBlueDots1,
                        dynamicGraphBlue);

    dynamicGraphBlue.invalidate();
}
```

updateGreenGraph()

Analogous to `updateBlueGraph()`, this function handles the update process for the green graph.

```
void DynamicGraphSlideMenuView::updateGreenGraph()
{
    updateGraphElements(toggleGreenLine, dynamicGraphGreenLine1,
                        toggleGreenDots, dynamicGraphGreenDots1,
                        dynamicGraphGreen);

    dynamicGraphGreen.invalidate();
}
```

customizeScreen()

This function applies additional visual customizations to the screen based on the states of the box and grid toggles. It sets the alpha transparency of a UI box and the major Y-axis grid of the third graph (dynamicGraph0) depending on whether their respective toggles are active. The graph is then invalidated to reflect these changes. This function enhances the user's ability to personalize the graph display.

```
void DynamicGraphSlideMenuView::customizeScreen()
{
    if (toggleBox.getState() == false)
    {
        box1.setAlpha(0);
    }
    else
    {
        box1.setAlpha(255);
    }

    if (toggleGrid.getState() == false)
    {
        dynamicGraph0MajorYAxisGrid.setAlpha(0);
    }
    else
    {
        dynamicGraph0MajorYAxisGrid.setAlpha(255);
    }
    dynamicGraph0.invalidate();
}
```

Go to Designer and Run Simulator (or flash the code to your board).

