MUSTURN

LCD Touch Display Total Solution

TouchGFX Designer User Guide

경기도 부천시 오정구 석천로 397 부천테크노파크 쌍용3차 303동 601호
TEL : 032-323-5953 / FAX : 032-323-5943 / Web Site : www.musturn.com

Table of contents

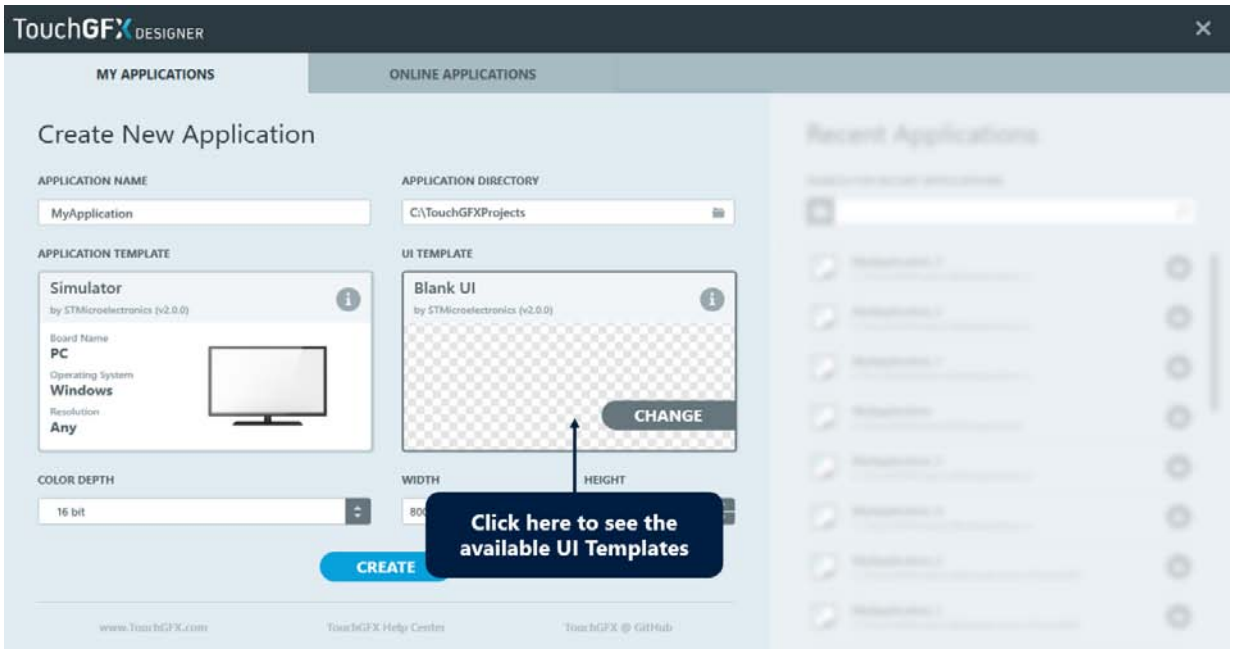
튜토리얼 1 : 예제 실행하기

이 튜토리얼을 따라 TouchGFX의 기본사항을 배울 수 있습니다. 제공된 예제를 TouchGFX 시뮬레이터 및 STM32 평가 키트에서 실행하는 방법에 대해 알아봅니다.

1. TouchGFX 시뮬레이터를 사용하여 예제 실행하기

TouchGFX는 TouchGFX Designer를 통해 사용할 수 있는 UI 예제들을 다양하게 제공하고 있으며, 이러한 예제는 모두 하나의 TouchGFX 주제 또는 위젯에 초점을 맞추고 있으므로 특정 TouchGFX 주제에 대해 자세히 알아보는 데 도움이 될 수 있습니다.

가. UI Template 선택하기

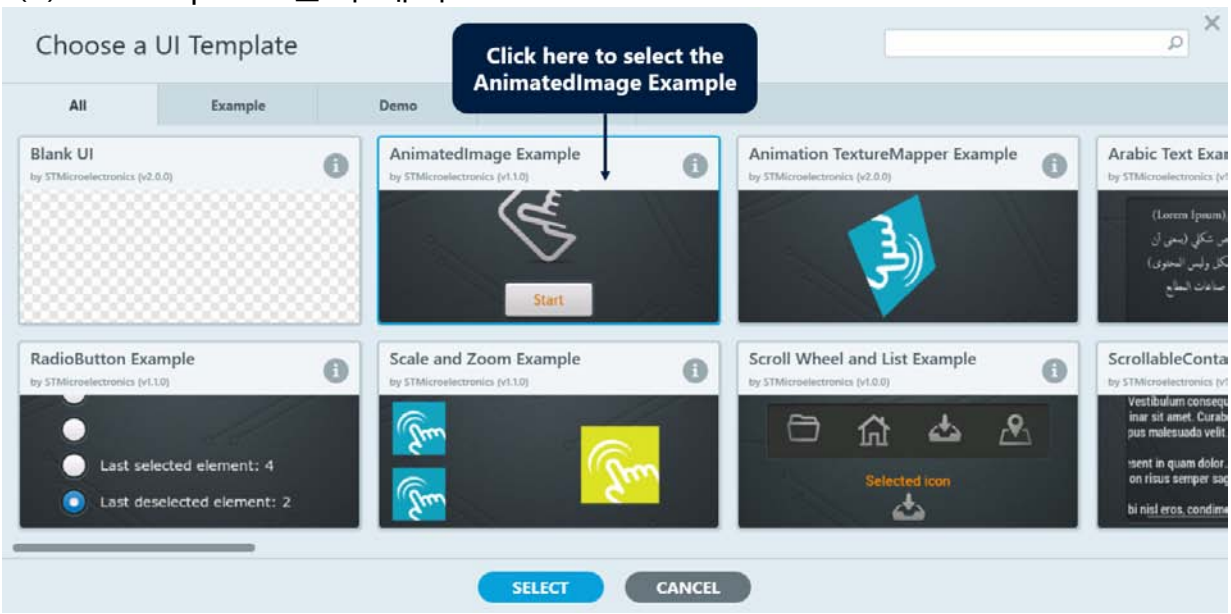


예제를 자체 프로젝트의 시작점으로 사용하거나 참조 예제로 사용할 수 있습니다. 예제는 TouchGFX 시뮬레이터를 사용하는 PC, STM32 평가키트 또는 사용자 지정 STM32 기반 하드웨어에서 실행할 수 있습니다.

■ 예제 프로젝트를 만들려면 CTRL + N 또는 TouchGFX Designer의 상단표시줄 메뉴에서 "File → New "를 선택합니다.

■ UI Template 섹션에서 "Change" 버튼을 클릭하여 사용 가능한 예제를 선택합니다.

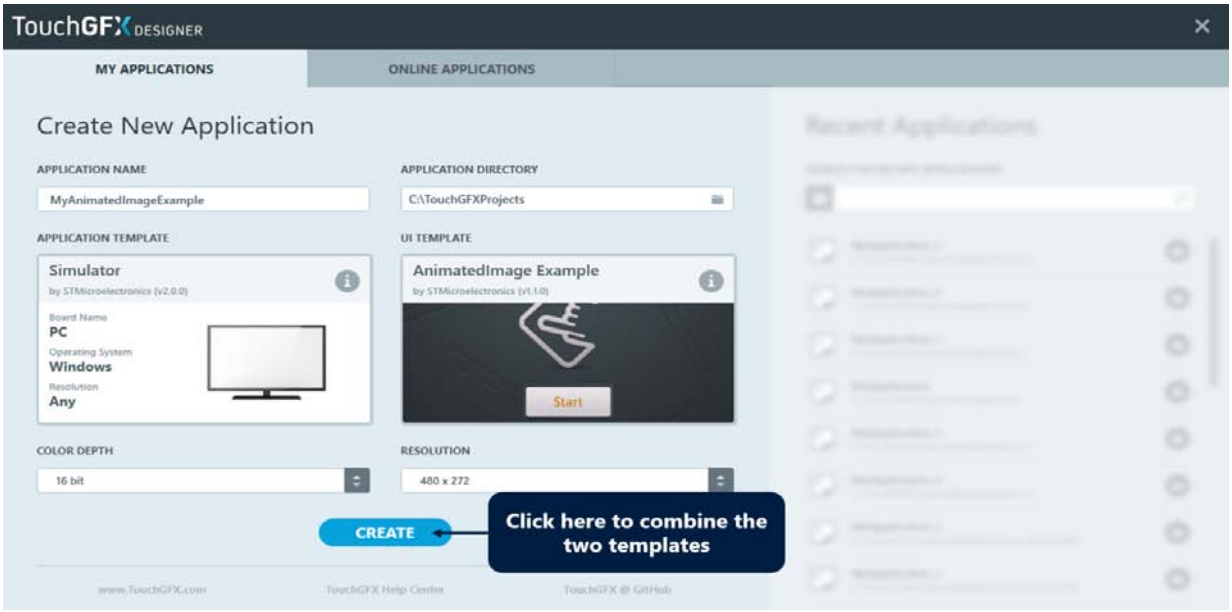
(1) UI Template 선택 예시



■ "AnimatedImage Example" 예제를 선택합니다.

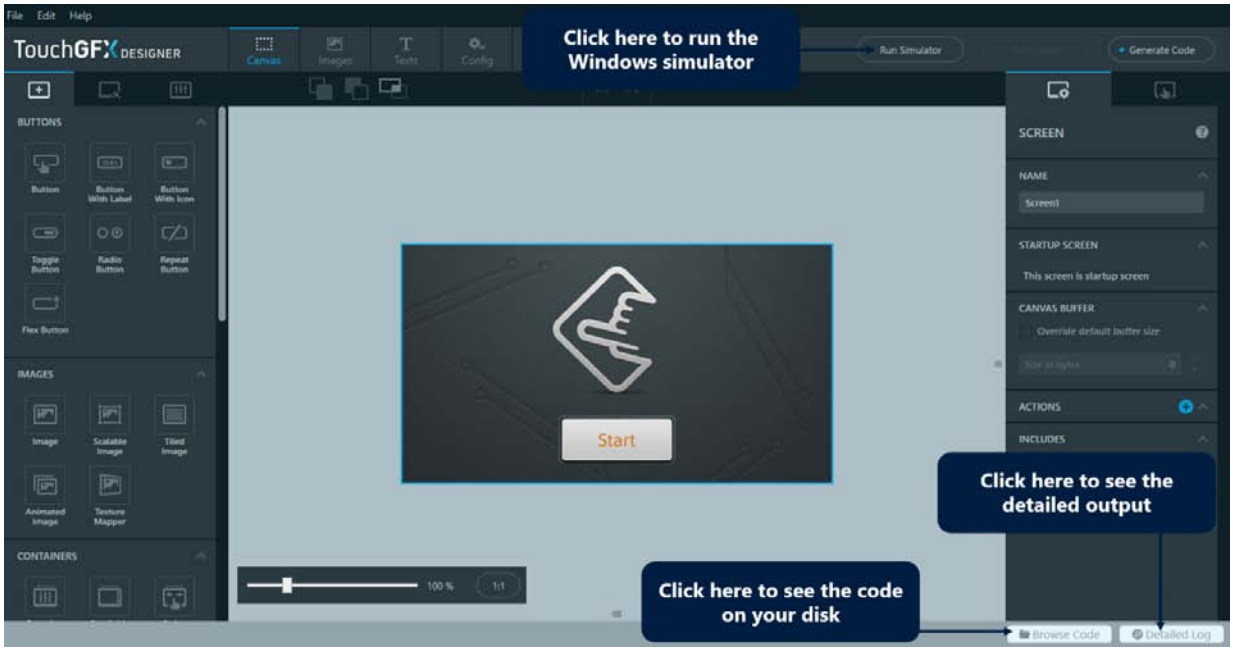
■ 하단의 "Select" 버튼을 클릭합니다.
UI Template 선택단계에서 "Select"을 클릭하면 TouchGFX Designer가 프로젝트를 생성 할 준비가 됩니다. 이 프로젝트 이름"MyAnimated ImageExample"로 지정했습니다.

나. 프로젝트 생성하기



"Create"를 클릭하여 프로젝트를 만듭니다. TouchGFX Designer는 이제 선택한 UI 템플릿을 "Simulator" 응용 프로그램 템플릿과 결합하여 완전한 프로젝트를 생성합니다. 이 프로세스는 다운로드 속도에 따라 생성시간의 차이가 있습니다.

다. TouchGFX 시뮬레이터 실행하기



이제 TouchGFX Designer가 결합된 프로젝트를 표시됩니다. Windows 시뮬레이터를 실행하려면 F5 또는 오른쪽 상단 "Run Simulator" 버튼을 클릭합니다.

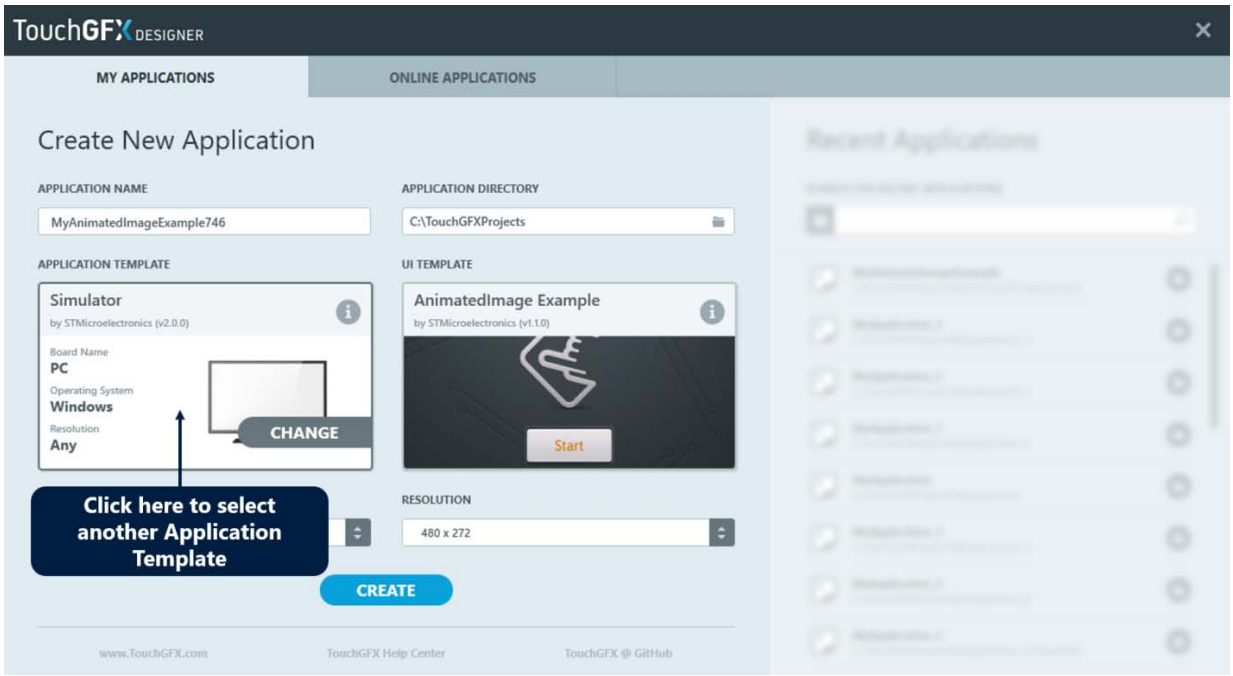
(1) TouchGFX 시뮬레이터 실행 결과

TouchGFX 시뮬레이터는 이제 일반 Windows 응용 프로그램으로 표시됩니다. 제목 표시줄에 응용 프로그램 이름이 표시됩니다. 예제와 상호 작용하려면 "Start" 버튼을 클릭하십시오.

2. STM32 평가 키트에서 예제 실행하기

이 단계에서는 평가 키트인 STM32F746-Disco 보드용 프로젝트를 시작하는 방법과 해당 보드에서 TouchGFX 예제 중 하나를 실행하는 방법을 배웁니다. STM32 평가 키트가 없는 경우 이 단계를 건너뛸 수 있습니다. 다른 SMT32 평가 키트가 있는 경우 지원되는 보드 목록에서 해당 보드를 찾을 수 있는지 확인 하시길 바랍니다.

가. UI Template 선택하기

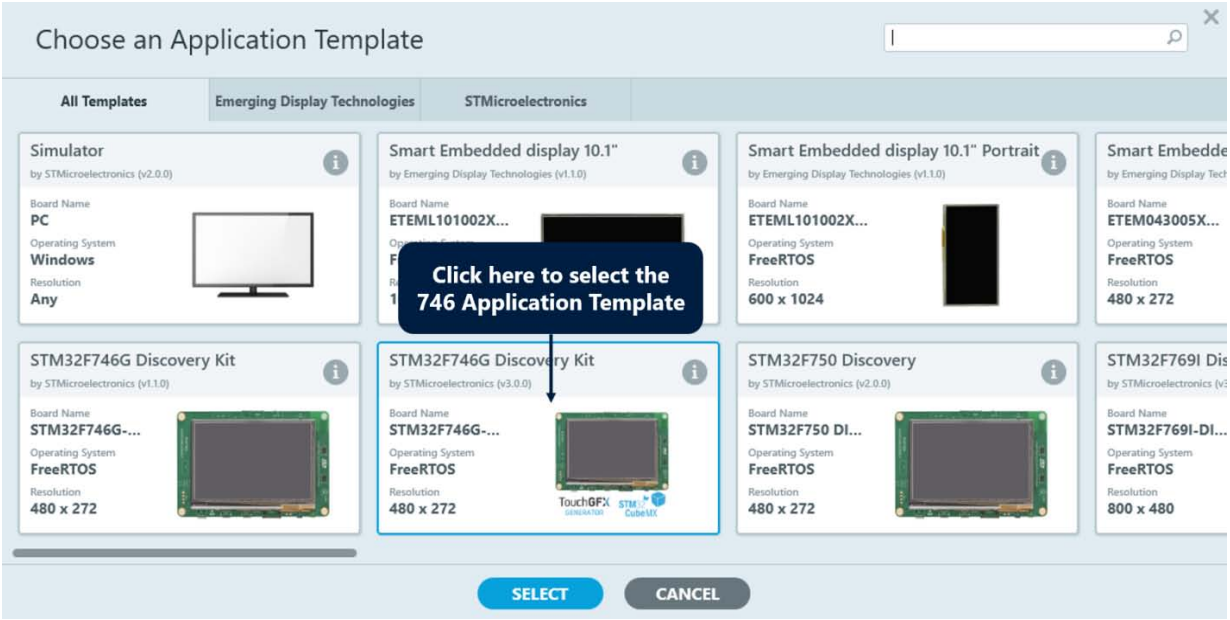


TouchGFX Designer는 다양한 STM32 평가 키트와 일치하는 애플리케이션 템플릿 목록과 함께 제공됩니다.

■ 이러한 템플릿을 기반으로새로운 프로젝트를 생성하려면 CTRL + N 또는 TouchGFX Designer의 상단표시줄 메뉴에서 “File → New “를 선택합니다.

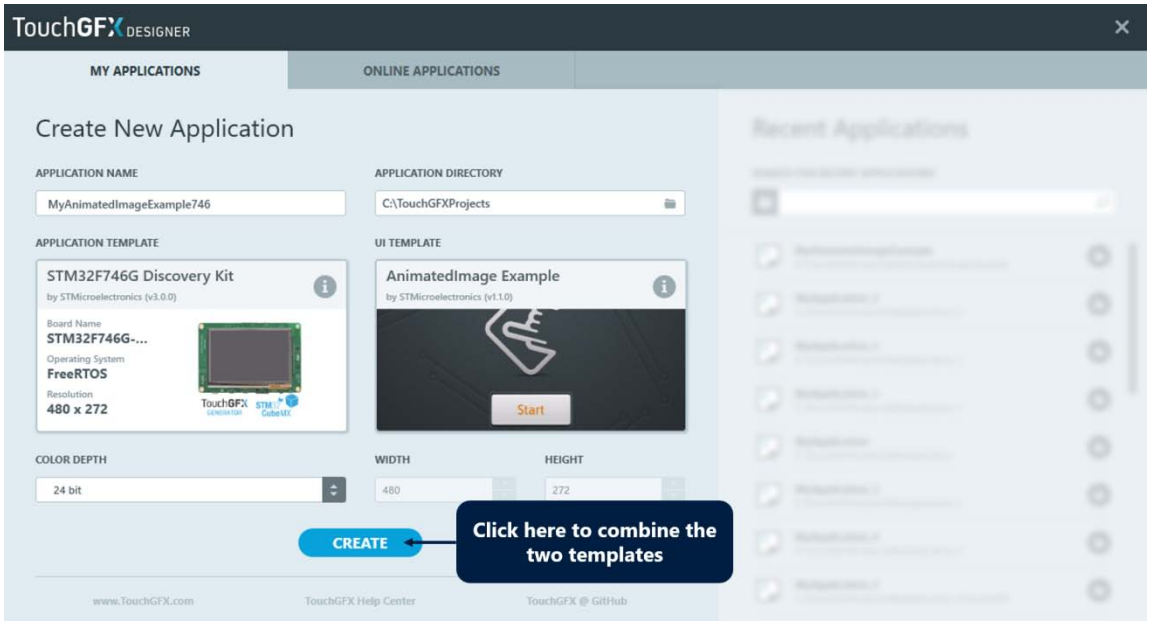
■ UI Template으로 "AnimatedImage Example"을 선택합니다(아직 선택하지 않은 경우) 다른 템플릿을 선택하려면"Application Template” 섹션을 클릭하십시오. 기본 애플리케이션 템플릿 “Simulator"를 사용하면 Windows에서만 실행할 수 있습니다.

나. 평가키트 선택하기



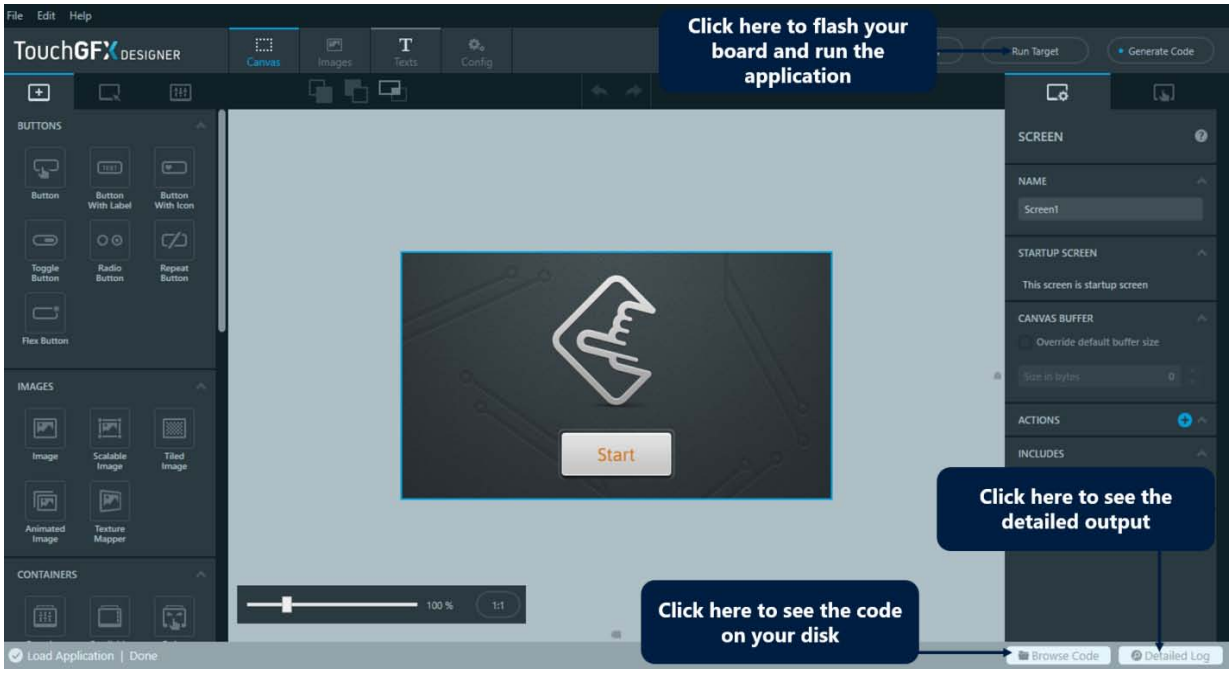
이 단계에서는 STM32F746-Disco 보드를 사용하므로 "STM32F746G Discovery Kit"를 클릭하고 "Select"를 클릭합니다.

다. 프로젝트 생성하기



응용 프로그램 이름을 변경할 수 있습니다. "MyAnimatedImageExample746"으로 변경했습니다. 계속하려면 "Create" 버튼을 클릭하세요.

라. 프로젝트 플래시 하기



프로젝트의 모양은 이전 단계에서 본 것과 유사합니다. 유일한 차이점은 이제 "Run Simulator" 버튼 옆에 "Run Target" 버튼도 있다는 것입니다.

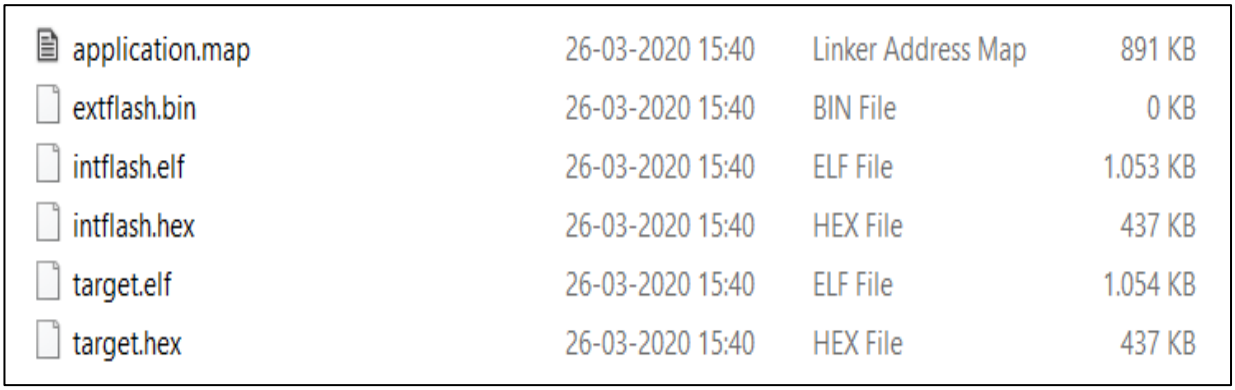
■ "Run Target"버튼(또는 F6)을 누르면 TouchGFX Designer가 GNU ARM C 컴파일러를 사용하여 프로젝트를 컴파일하고 대상에 애플리케이션을 플래시합니다.

■ TouchGFX Designer 하단의 상태 표시 줄에 진행 상황이 출력됩니다. 빌드 및 플래싱 단계에 대한 자세한 내용을 보려면 "Detailed Log" (또는 ALT + L) 버튼을 누르면 확인 할 수 있습니다.

■ TouchGFX Designer는 플래싱이 완료가 되면 상태 표시줄에 "FlashingDone"을 기록합니다. 이때 보드에서는 실행 중인 애플리케이션이 표시되어야 합니다.

안내	대상을 플래시 하려면 Cube Programmer/ST-Link Utility가 설치되어 있어야 합니다. ■ STM32 Cube Programmer ■ STM32 ST-LINK Utility

(1) 응용 프로그램 찾아보기



오른쪽 하단의 "Browse Code" 버튼을 클릭하면 TouchGFX Designer가 새 프로젝트가 있는 디렉토리를 보여주는 파일 브라우저를 엽니다. Build \ bin을 탐색하면 그림과 다음 파일을 볼 수 있습니다.

target.hex파일은 보드의 STM32 응용 프로그램입니다. 현 보드에 프로그래밍된 TouchGFX Designer 파일입니다.

Cube Programmer 또는 ST-Link Utility를 사용하여 수동으로 보드를 플래시할 수도 있습니다. 자세한 내용은 컴파일 및 플래싱 페이지를 참조하십시오

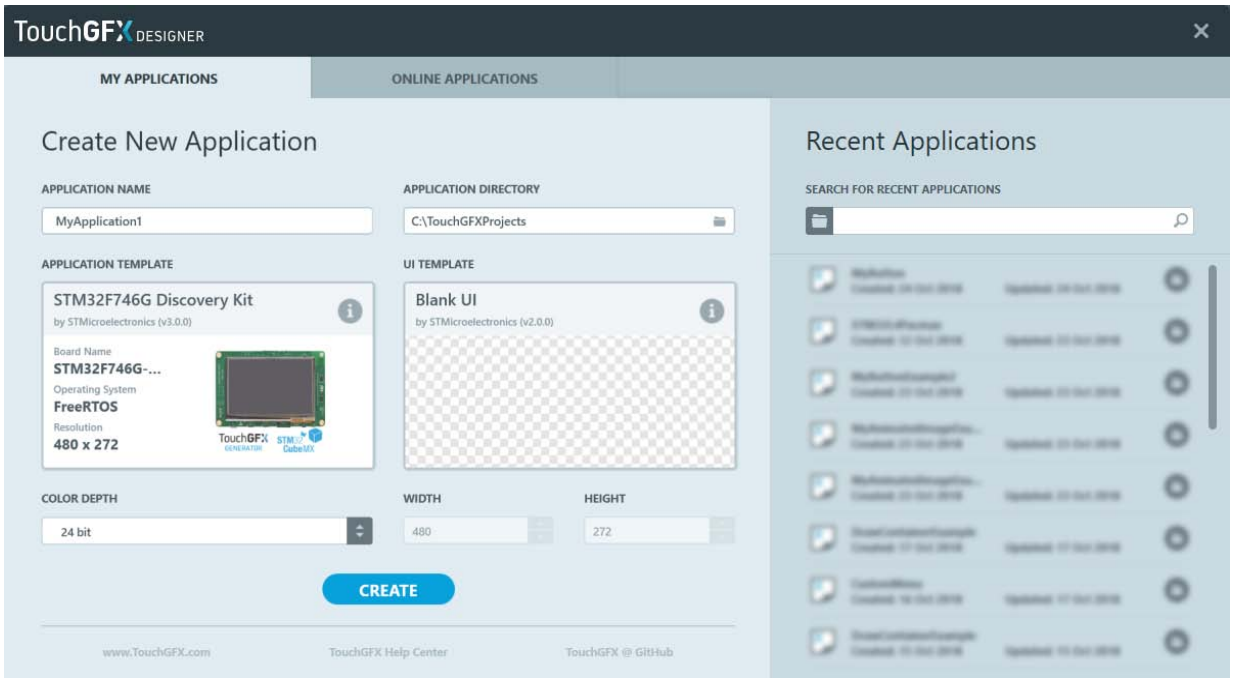
튜토리얼 2 : 나만의 응용 프로그램 만들기

이 튜토리얼을 따라 TouchGFX의 기본 사항에 대해 자세히 알 수 있습니다. 프로그램에 이미지를 추가하고 버튼을 사용하는 방법을 배우게 됩니다. 또한 텍스트와 계산된 숫자를 사용하는 방법도 볼 것입니다. 마지막 단계에서는 TouchGFX Designer로 만든 UI의 형태을 개선시키는 코드를 작성해 볼 것입니다.

1. 배경 이미지 설정하기

이 단계에서는 PNG 이미지를 배경으로 설정하는 방법에 대해 알아 볼 것 입니다.
그 전에 새로운 프로젝트를 생성할 것입니다.

가. 새로운 프로젝트 생성하기



TouchGFX Designer에서 새로운 프로젝트를 생성합니다. 프로젝트의 이름을 "MyApplication1"으로 지정합니다. 이 프로젝트는 "STM32F746G Discovery Kit" 애플리케이션 템플릿과 "Blank UI" UI 템플릿을 기반으로 합니다.

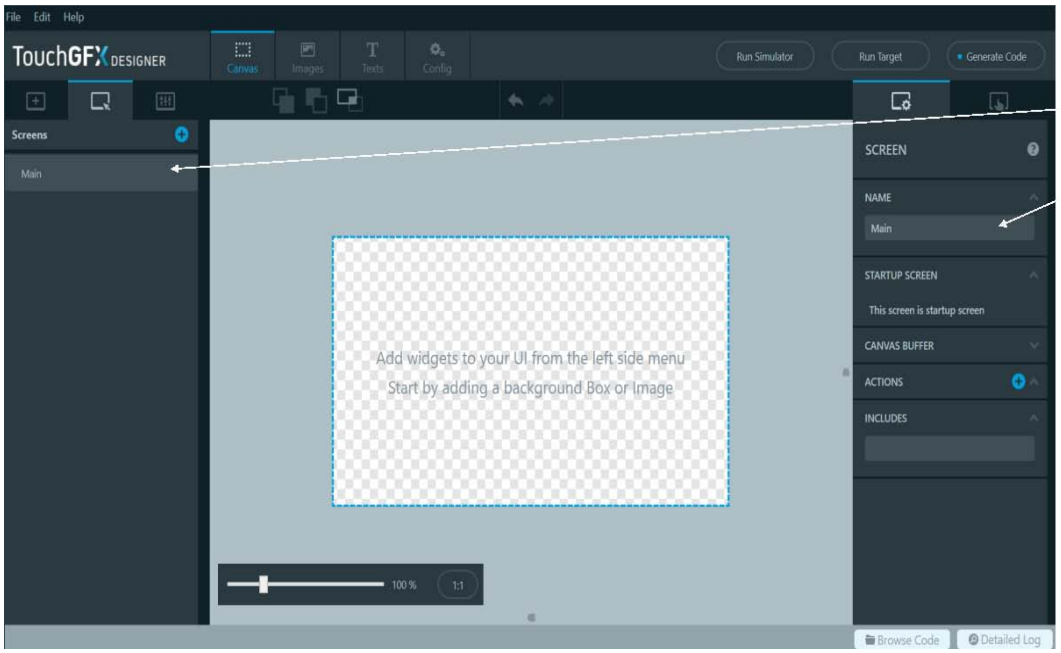
다른 STM32 평가 키트가 있는 경우 애플리케이션 템플릿을 변경할 때 TouchGFX Designer에 표시된 목록을 확인하여 지원되는지 확인하십시오. 지원되는 보드가 없는경우 "Simulator" 응용 프로그램 템플릿을 선택하고 컴퓨터에서 응용프로그램을 실행할 수 있습니다.

본 튜토리얼은 480x272 해상도의 디스플레이에서 실행됩니다. 다른 해상도의 프로그램 템플릿을 선택하면 그래픽이 화면에 맞지 않을 수도 있지만 튜토리얼을 완료할 수 있습니다.

■ 예제 프로젝트를 만들려면 CTRL + N 또는 TouchGFX Designer의 상단 표시줄 메뉴에서 "File → New "를 선택합니다.

■ UI Template 섹션에서 "Change" 버튼을 클릭하여 사용 가능한 예제를 선택합니다.

나. 화면 이름 변경하기

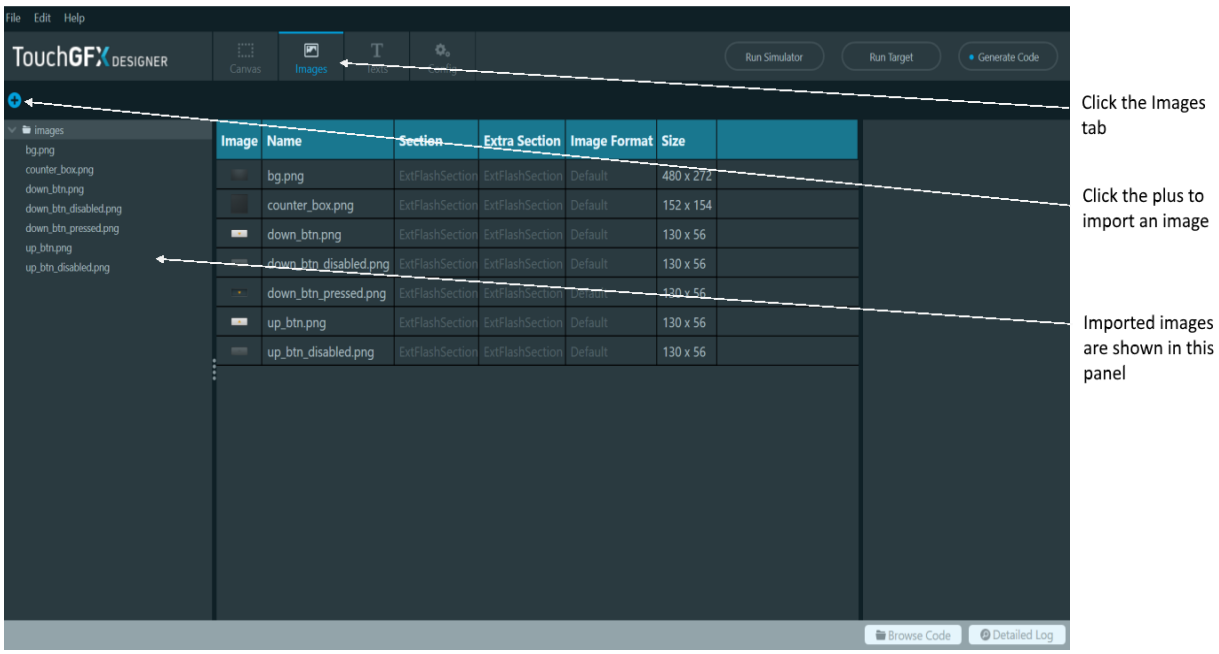


새로 생성된 빈 프로젝트가 있으므로 수정을 시작하겠습니다. TouchGFX 응용프로그램은 여러 화면으로 구성됩니다. 화면에는 사용자 인터페이스를 구성하는 여러 위젯이 포함되어 있습니다. 화면은 한 번에 하나의 화면만 사용자에게 표시됩니다.

먼저 초기 화면의 이름을 위와 같이 “Main"으로 변경합니다.

다. 배경 삽입하기

(1) 배경 이미지 가져오기



일반적으로 하나 이상의 위젯으로 화면의 전체 배경을 덮는 것이 좋습니다. 예를 들어 박스 칸 또는 이미지가 될 수 있습니다. 이 응용 프로그램에서는 이미지를 사용합니다.

TouchGFX Designer에서 이미지를 사용하려면 먼저 파일을 가져와야 합니다. TouchGFX는 BMP 및 PNG 이미지를 지원합니다(TouchGFX Designer는 PNG 이미지 지원함). PNG 파일은 BMP 파일보다 작고 투명 픽셀을 지원하므로 선호됩니다.

이 튜토리얼에서 사용할 이미지는 이 링크에서 다운로드를 할 수 있습니다. 디스크의 디렉토리에 파일의 압축을 풉니다.

"background.png"라는 파일을 배경으로 사용하려고 합니다. 해당 파일을 가져오려면

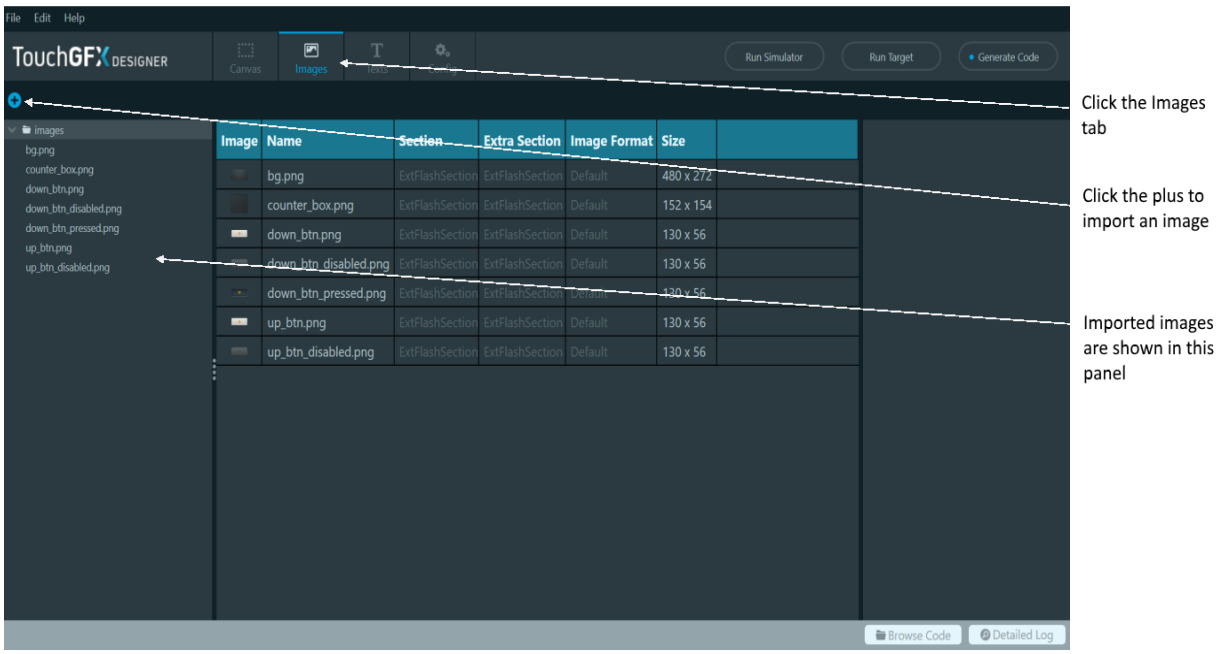
■ 이미지 탭을 선택하고 파란색 더하기 아이콘을 클릭합니다.

■ 압축을 푼 폴더로 이동하여 "background.png" 파일을 선택합니다. 열기를 눌러 가져옵니다.

■ 파일 탐색기에서 이미지 탭으로 이미지를 "Drag and Drop"하거나 캔버스에서 직접 프로젝트로 가져올 수도 있습니다.

■ 가져온 이미지는 프로젝트로 변환 및 컴파일되어 플래시 공간을 차지한다는 점을 유의하시길 바랍니다. 따라서 필요한 이미지만 가져옵니다.

(2) 이미지 위젯 삽입하기



이제 응용프로그램에서 이미지를 사용할 준비가 되었습니다. 그러기 위해서는 Image 위젯이 필요합니다.

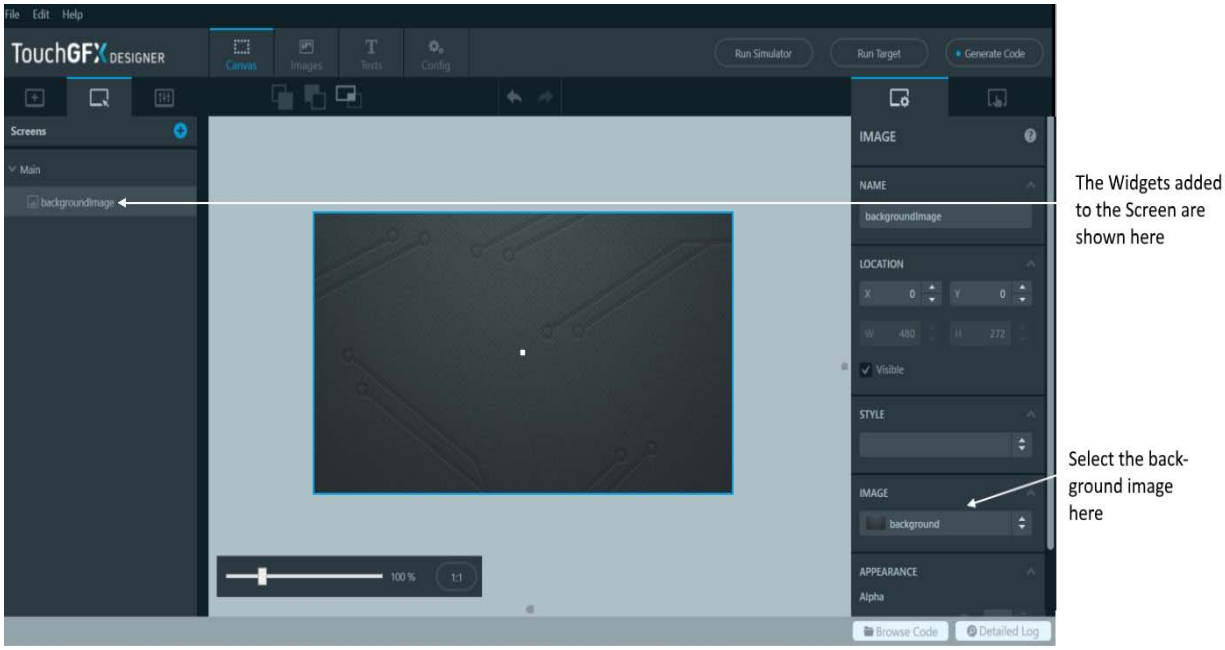
■ 캔버스 탭에서 위젯 탭을 선택합니다.

■ 위젯 목록에서 이미지 위젯을 찾습니다.

■ 이미지 위젯을 클릭하면 화면에 이미지 위젯이 추가됩니다.

위젯의 이름을 위젯의 의미와 연관시켜 변경하는 것은 좋은 원칙입니다. 현재 이 경우 "backgroundImage"와 같은 것입니다.

(3) 가져온 이미지 파일을 배경으로 선택하기



위젯을 삽입한 후 일반적으로 Position 또는 Color 와 같은 위젯의 일부 속성을 구성해야 합니다. 위젯의 속성은 TouchGFX Designer의 오른쪽에 표시됩니다. 이 경우에는 좌표의 위치가 (0,0)의 위치에 만족하지만 이전에 가져온 "background.png" 파일을 선택하도록 Image 속성을 변경하려고 합니다. 이미지 드롭다운 목록에서 "background.png"를 선택합니다.

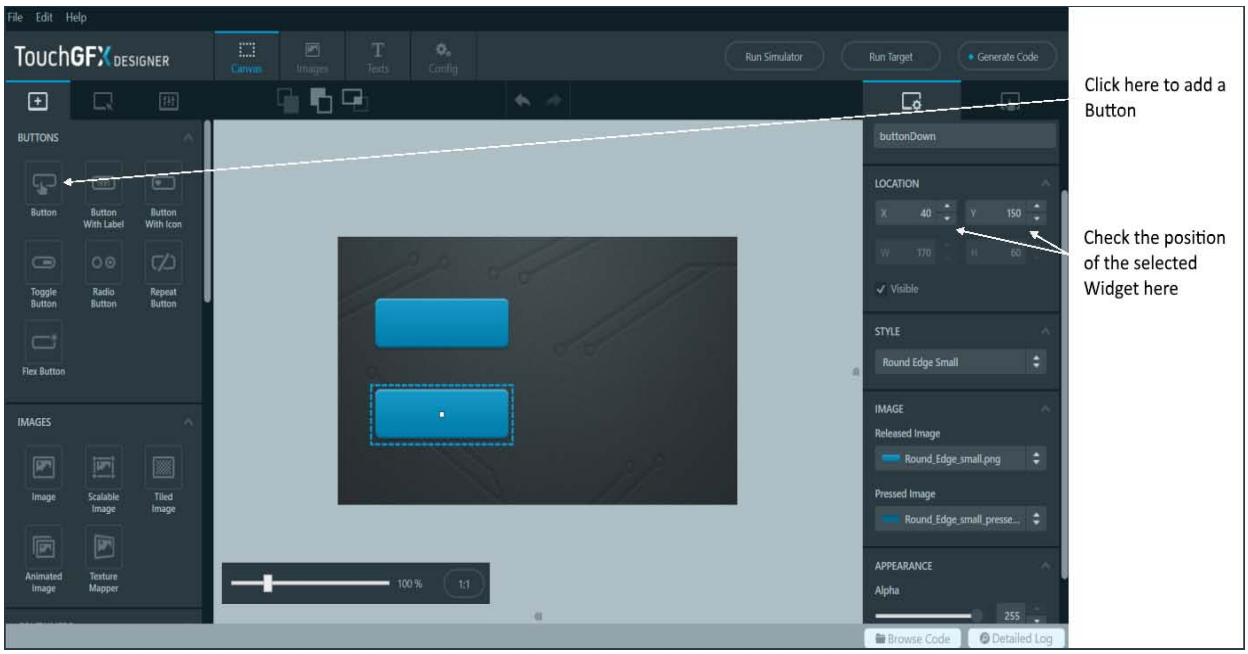
이제 전체 사용자 인터페이스를 포괄하는 배경 이미지만으로 구성된 하나의 화면이 있는 간단한 응용 프로그램을 만들었습니다.

다음으로 이동하기 전에 "Run Simulator" 버튼을 눌러 프로젝트가 컴파일이 되고 실행이 되는지 확인하십시오. 아직 활성 위젯을 추가하지 않았으므로 응용프로그램과 상호 작용할 수 없습니다

2. 버튼 추가하기

이 단계에서 응용 프로그램에 두 개의 버튼을 추가하고 다른 PNG 이미지파일을 사용하여 버튼을 커스텀하여 사용해 봅니다.

가. 버튼 아이콘 추가하기



■ 위젯 탭에서 버튼 위젯을 클릭하여 화면에 버튼을 추가합니다.

■ 새 위젯을 마우스로 드래그하여 이동합니다.

■ 버튼을 X - 40, Y - 60에 배치합니다.

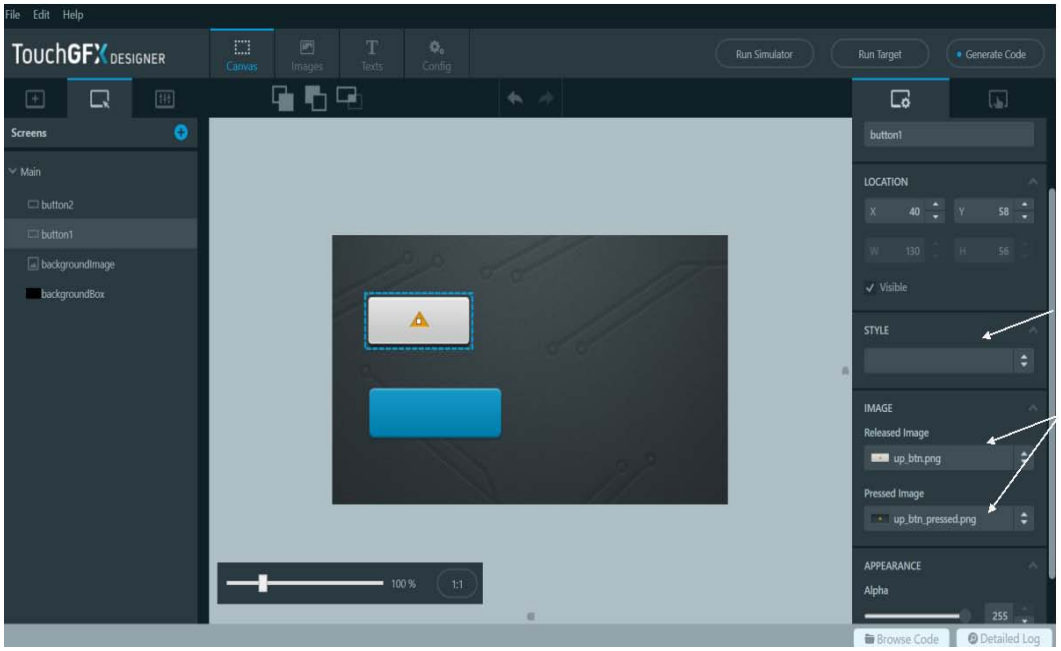
■ 새 위젯의 이름을 "buttonUp"으로 지정합니다.

■ X - 40, Y – 150 위치에 다른 버튼을 추가합니다. 이 위젯의 이름을 "buttonDown"으로 지정합니다.

X 및 Y 속성의 위치 조정버튼을 사용하여 위젯의 위치를 미세 조정할 수 있습니다. 버튼 위젯을 선택하고(Canvas에서 클릭하여) 키보드 방향키를 사용하여 위치를 조정할 수도 있습니다.

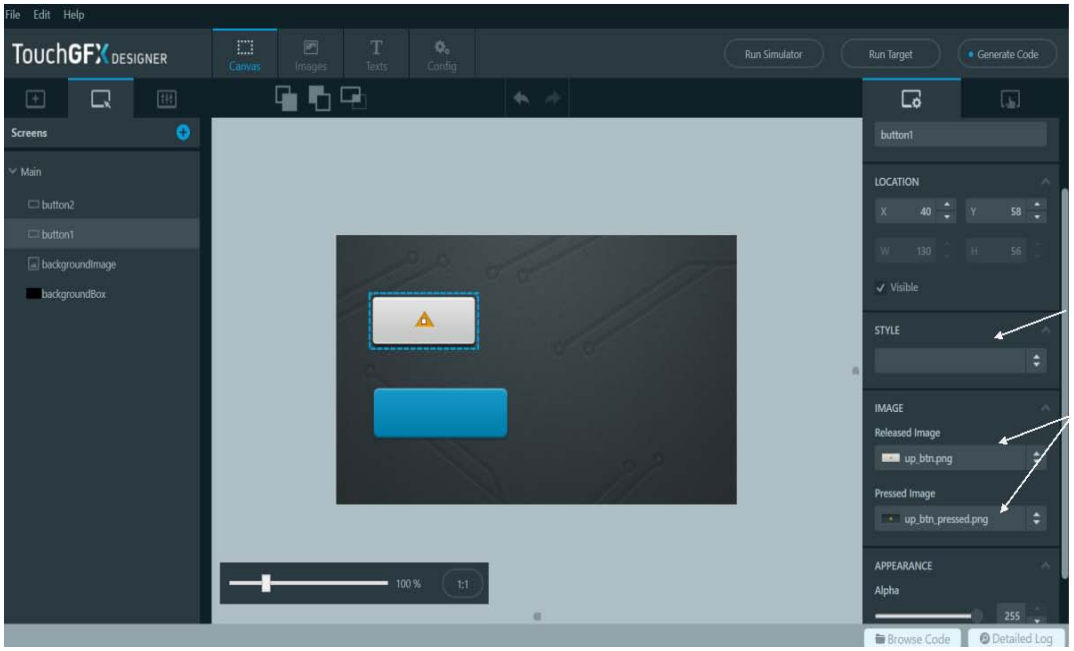
나. 버튼 아이콘 모양 변경하기

(1) 변경할 버튼의 비트맵 설정하기



이 단계에서는 버튼의 모양을 변경합니다. 버튼은 두 개의 이미지로 구성되며 버튼을 눌렀을 때 하나의 이미지가 표시되고 버튼을 누르지 않았을 때 다른 이미지가 표시됩니다. 대부분의 위젯은 기본적으로 특정 형태을 설명하는 위젯의 특정 속성에 대한 정의된 스타일 집합과 함께 제공됩니다. 이러한 스타일은 빠른 프로토타입에 적합하지만 대부분 실제 응용프로그램을 만들 때 대체합니다.

(2) 변경할 버튼이미지 가져오기



이전 단계에서와 같이 이미지 탭으로 이동하고 "plus" 아이콘을 클릭하여 일부 이미지를 가져옵니다.

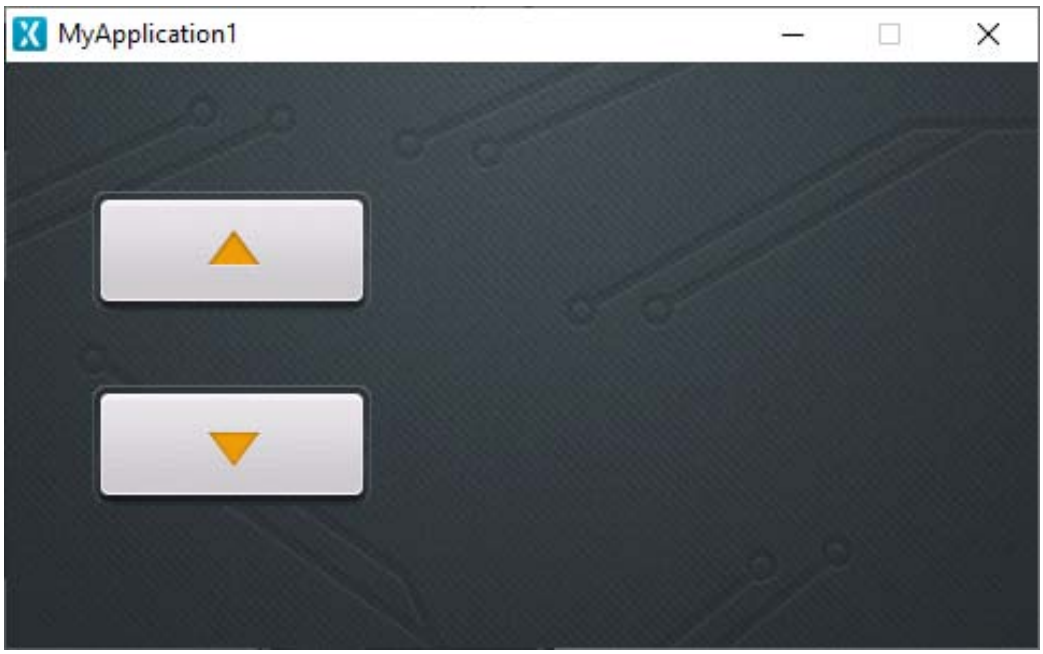
"button_down_pressed.png", "button_down_released.png", "button_up_pressed.png" 및 "button_up_released.png" 이미지를 가져옵니다.

이제 "buttonUp" 버튼을 선택합니다. 해당 버튼의 경우 Released 이미지 속성에 대해 "button_up_released.png"를 선택합니다.
Pressed Image에 대해"button_up_pressed.png"를 선택합니다.

"buttonDown"의 경우 Released 이미지에 대해 "button_down_released.png"를 선택하고 Pressed 이미지에 대해 "button_down_pressed.png"를 선택합니다.

완료된 사항은 TouchGFX Designer에서 형태가 바뀐 버튼을 바로 볼 수 있습니다.

다. 시뮬레이터 실행 결과



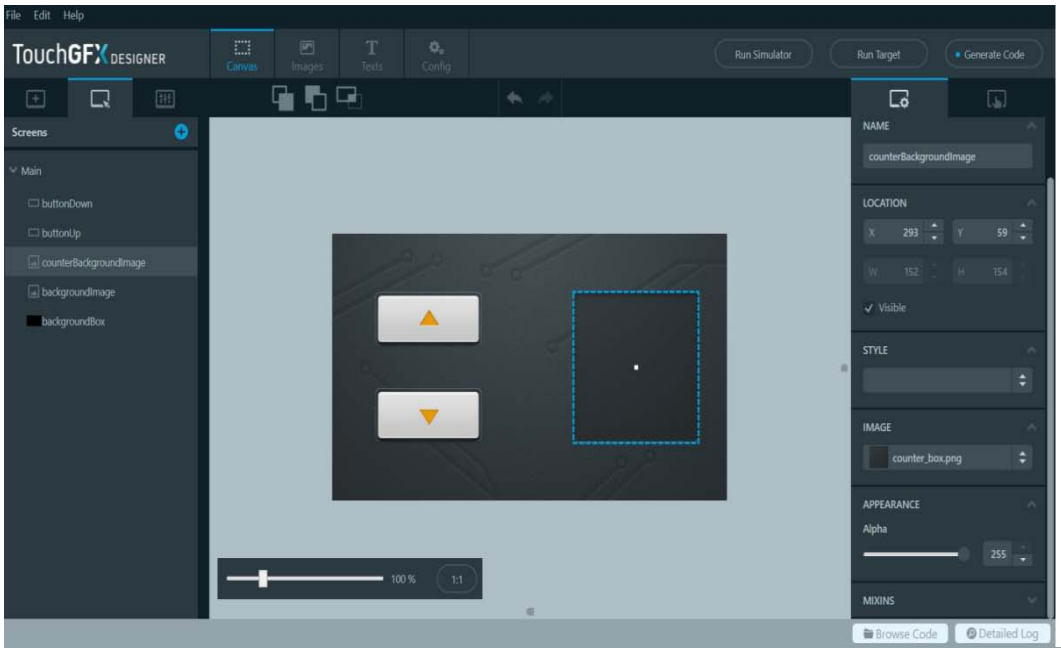
이제 버튼 설정이 완료되었다면 "Run Simulator"을 클릭하여 응용프로그램을 실행합 니다. 두 버튼을 모두 클릭하여 버튼이 올바르게 작동하는지 확인합니다.

안내	TouchGFX의 대부분의 위젯은 크기가 정의된 이미지를 사용하므로 사용자가 직접 크기를 조정할 수 없습니다. 이는 기능상의 이유로 수행됩니다(일반 UI 구성 요소 기능 참조). 현재 튜토리얼와 같이 위젯의 크기를 변경하려는 경우 사용자가 원하는 크기의 새로운 버튼 이미지 세트를 생성하여 기존 Released와 Pressed 이미지를 대신 하여 사용해야 합니다.

3. 텍스트 추가하기

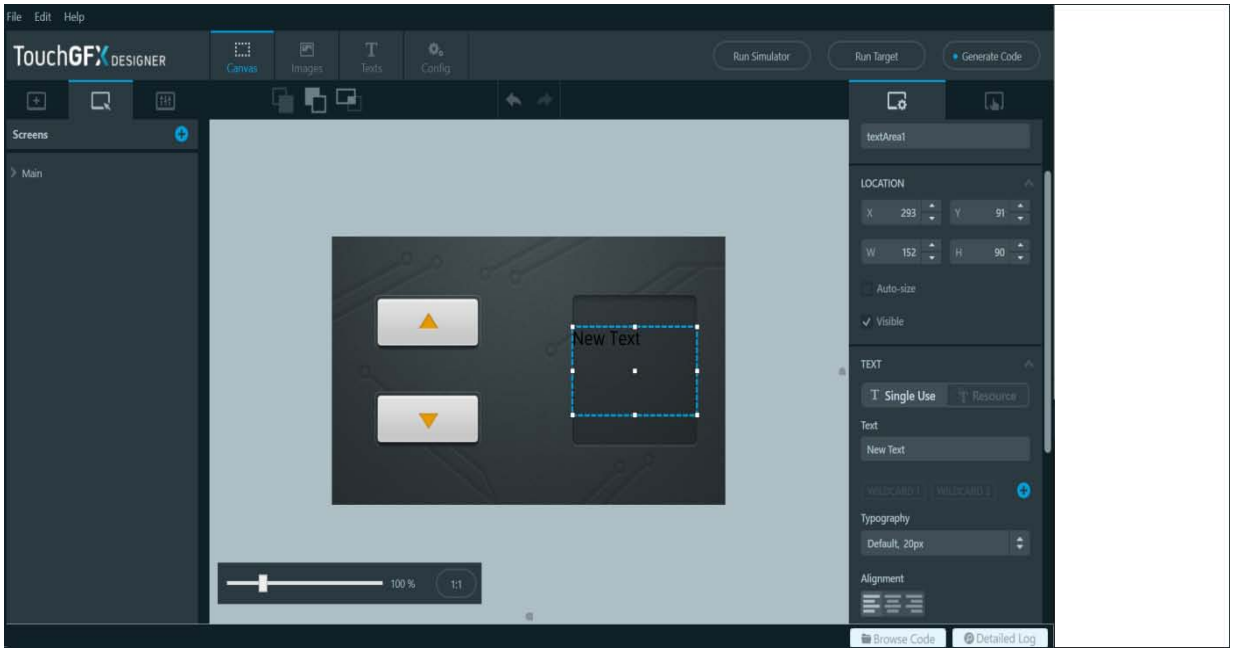
이 단계에서는 응용 프로그램에 대형 TextArea 위젯을 추가합니다. 모든 텍스트는 TextArea 위젯을 사용하여 표시됩니다. 하지만 응용 프로그램에 TextArea 위젯을 추가하기 전 텍스트에 더 나은 배경을 제공하기 위해 다른 이미지를 추가합니다.

가. 텍스트 배경 추가하기



- 다른 이미지 파일 "counter_box.png"를 가져옵니다.
- 새 이미지 위젯을 추가합니다.
- 새 이미지 위젯의 이름을 "textBackground"로 지정합니다.
- 이미지 위젯을 x=250, y=59에 위치시킵니다.
- 위젯의 이미지 속성을 가져온 이미지인 "counter_box"으로 설정합니다.

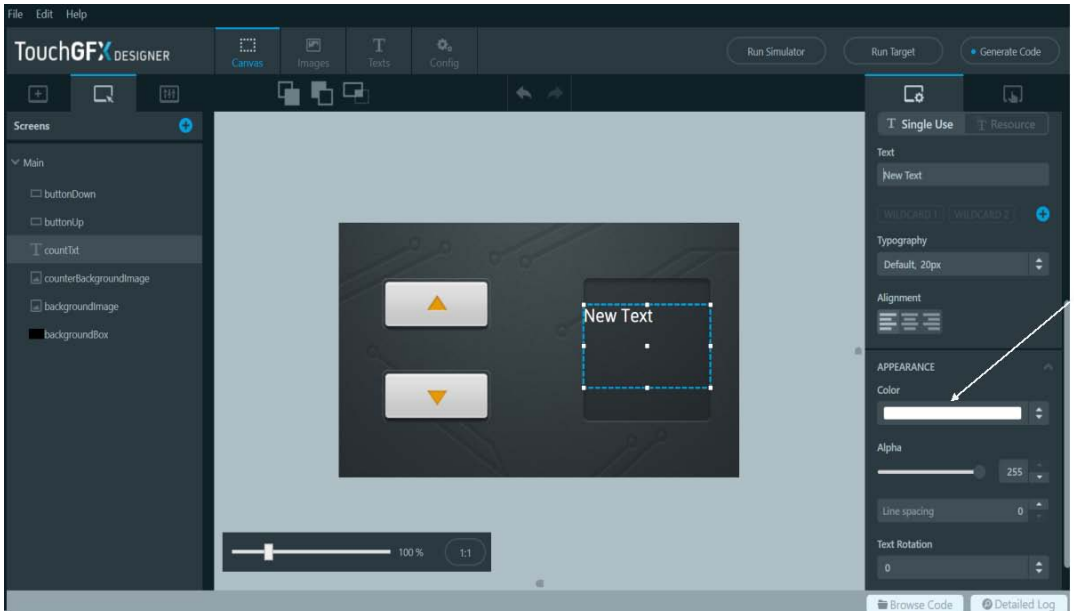
나. 텍스트 내용 추가하기



이제 TextArea 위젯을 추가할 준비가 되었습니다. 위젯 탭에서 TextArea 아이콘을 클릭합니다. TextArea 위젯의 이름을 "textCounter"로 바꾸고 위젯을 X - 250, Y – 90 위치로 이동합니다. 위젯을 큰 텍스트로 표시하려면 TextArea 위젯 설정에서 "Auto-Size" 설정을 클릭하여 취소하고 텍스트 크기를 W - 152, H - 90으로 설정합니다.

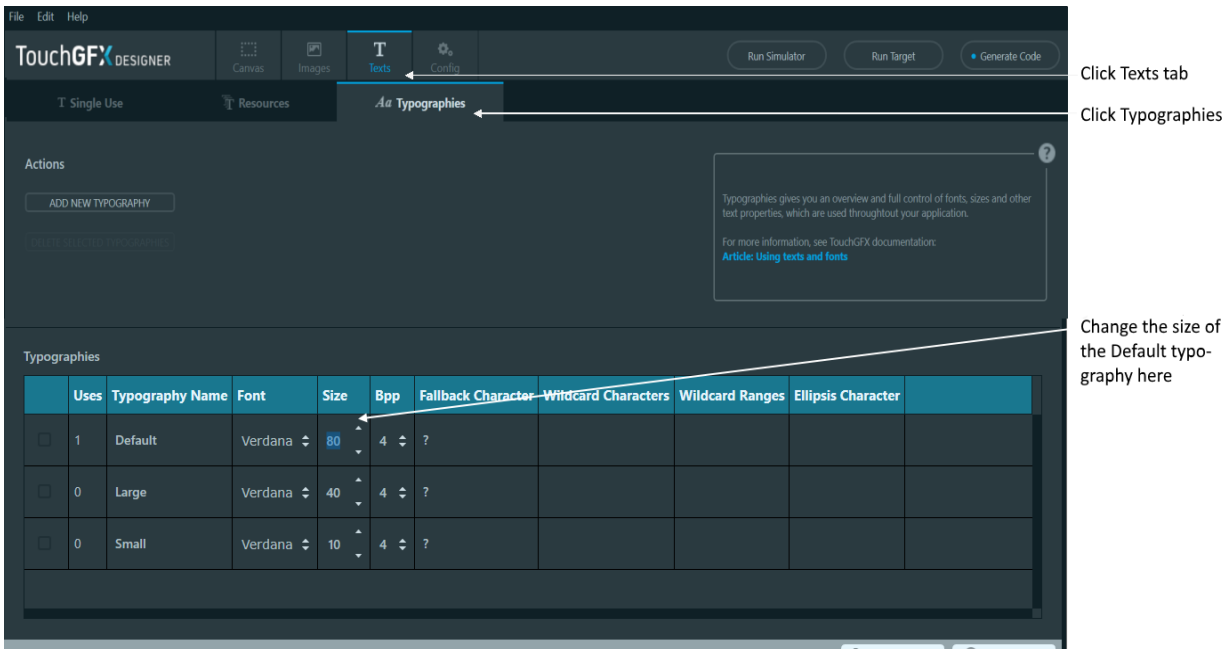
다. 텍스트 변경하기

(1) 텍스트 색상 변경하기



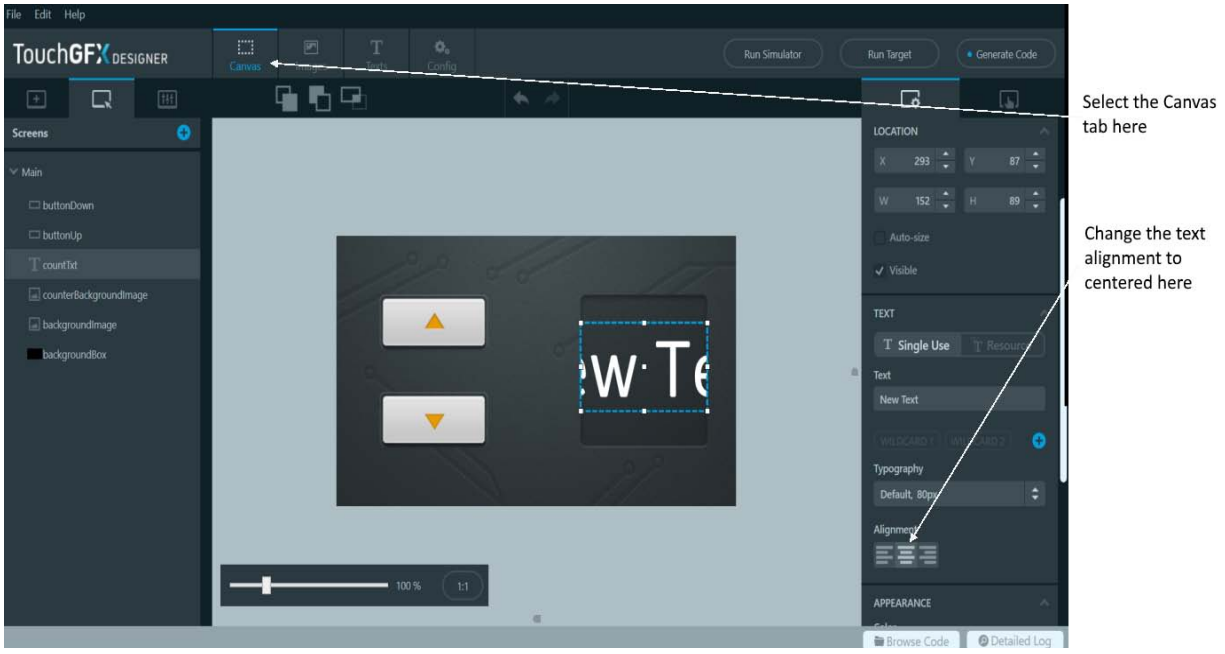
TextArea 위젯의 기본 색상은 검정으로 현재 배경에서 다소 어둡습니다. 보이기 위해 "textCounter" 의 Color 속성을 선택하고 색상을 흰색으로 변경합니다.

(2) 텍스트 크기 변경하기



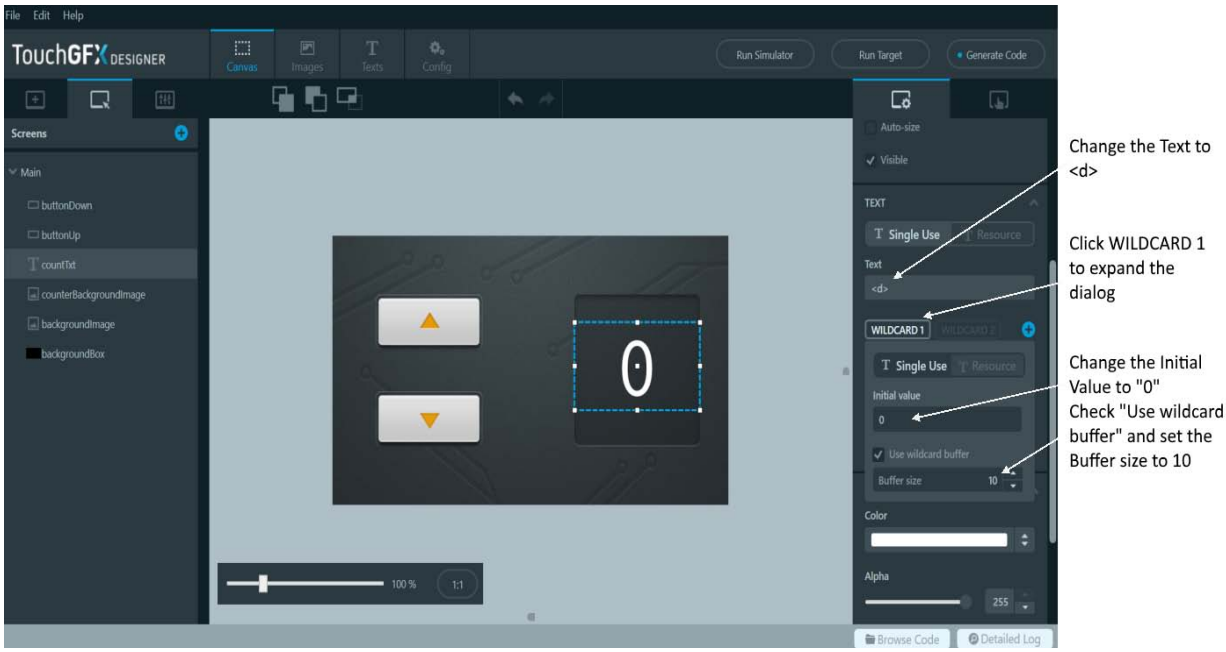
TouchGFX Designer 상단의 텍스트 탭을 선택하고 해당 Typographic를 클릭한 다음 "Size"의 "Default"값을 80으로 변경합니다.

(3) 텍스트 정렬 변경하기



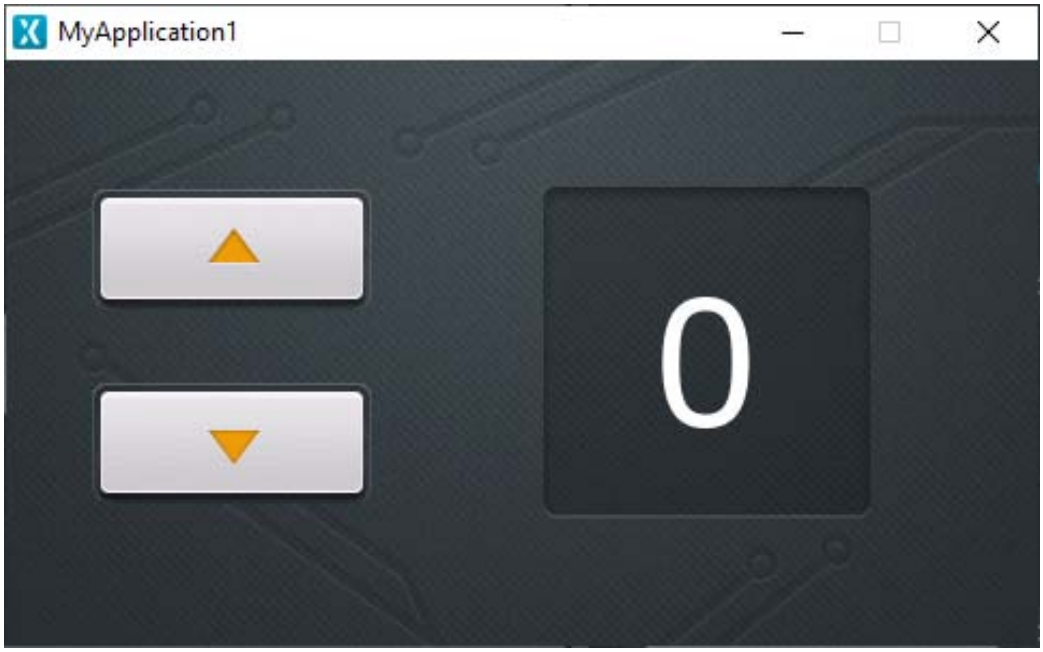
화면으로 돌아가서(상단의 "Canvas" 탭을 클릭하여) 이제 텍스트가 훨씬 더 커진 것을 볼 수 있습니다. 완전한 텍스트 "New Text"를 읽을 수 없습니다. 텍스트를 가운데에 맞추려면 Alignment 속성 아래의 "centered here"아이콘을 클릭하십시오 .

라. 와일드카드 텍스트 사용하기



버튼으로 숫자를 변경하여 화면에 표시되는 TextArea를 구성하려고 합니다. 그렇게 하기 위해서는 와일드카드를 포함한 텍스트를 구성해야 합니다. 와일드카드의 텍스트 표기법은 (" < d > ")입니다. 이 단계에서는 숫자를 표시하는 것이므로 텍스트의 내용을 "< [d] >"로 변경합니다. 와일드 카드는 고정 텍스트와 같이 결합하여 사용 할 수 있습니다. (예: "Temperature : < temp > °C").

마. 시뮬레이터 실행 결과

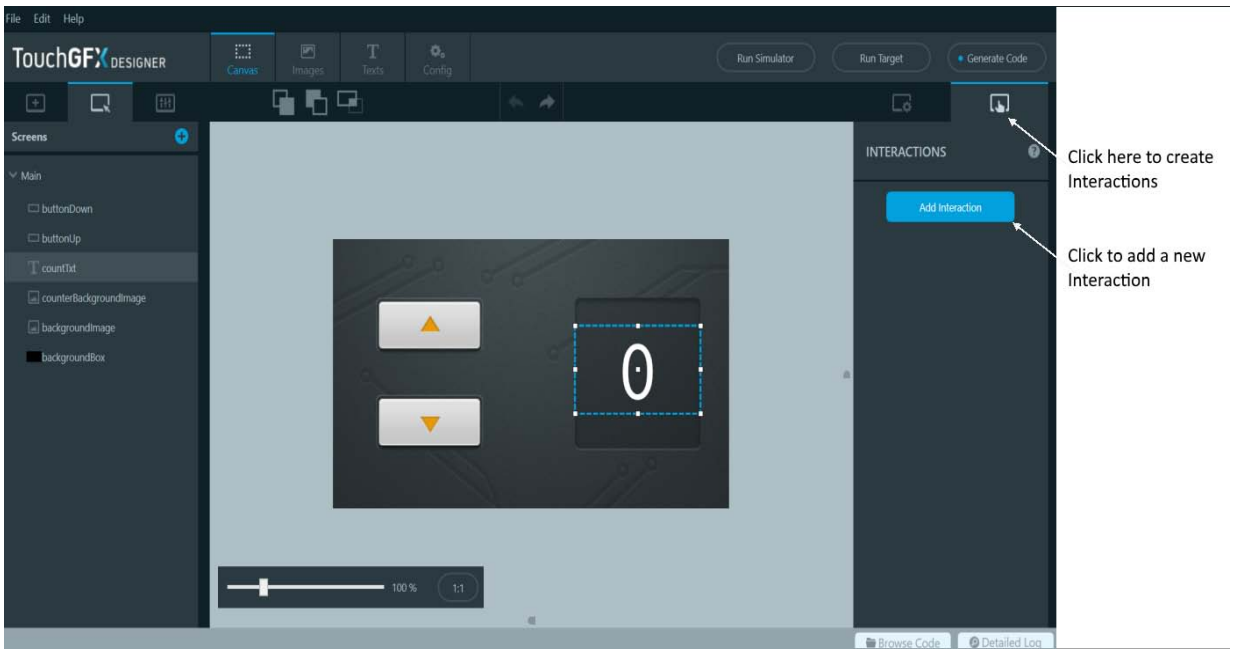


안내	텍스트 및 글꼴 사용에 대한 자세한 내용은 텍스트 및 글꼴 페이지를 참조하십시오.

4. 코드 추가하기

TouchGFX Designer를 사용하면 상호 작용을 통해 동작을 버튼에 쉽게 연결할 수 있습니다. 상호 작용은 트리거(예: 버튼 누름)를 동작(예: 코드 실행 또는 요소 이동)에 연결합니다.

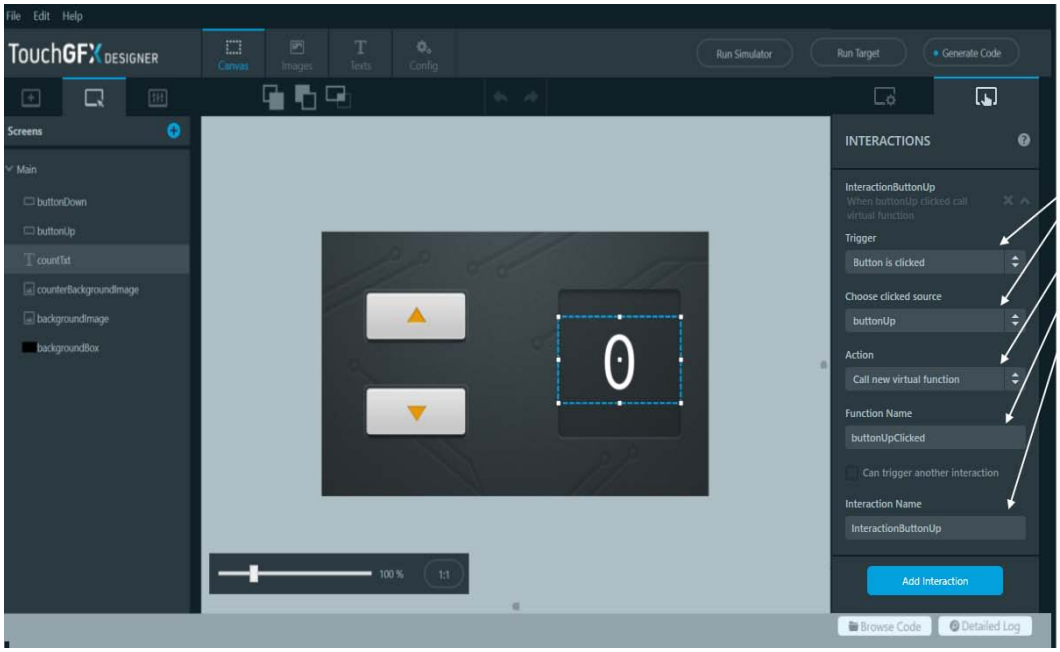
가. 상호 작용 추가하기



오른쪽 상단 모서리에 있는 상호 작용 탭을 선택하고 "Add Interaction" 버튼을 클릭하여 새 상호 작용을 생성합니다

각 버튼에 대해 하나씩 총 두 개의 상호 작용을 생성합니다. 현재 화면에서 C++ 메서드를 호출하도록 두 상호 작용을 모두 설정합니다.

나. 트리거 구성하기



■ 트리거 속성을 "Button is clicked"로 변경합니다.

■ "Choose cilcked source"의 속성을 "buttonUp"으로 지정합니다.

■ "Action"의 속성을 "Call new virtual function"로 변경합니다 .

■ "Function Name"의 이름을 "buttonUpClicked"으로 지정합니다.

ButtonDown버튼도 Clicked Source를 이용하여 유사한 상호작용을 생성합니다.

■ 트리거 속성을 "Button is clicked"로 변경합니다.

■ "Choose cilcked source"의 속성을 "buttonDown"으로 지정합니다.

■ "Action"의 속성을 "Call new virtual function"로 변경합니다 .

■ "Function Name"의 이름을 "buttonDownClicked"으로 지정합니다.

"Generate Code" 버튼 또는 "Run Simulator" 버튼을 클릭하면 TouchGFX Designer는 생성된 코드를 해당 단계에서 생성한 상호 작용에 대한 정보로 업데이트를 합니다. 이것은 "ViewBase" 클래스에 두 개의 새 가상 함수가 생성됨을 의미합니다.

(1) 코드 파일 찾아보기

IDE(통합개발환경)	프로젝트 파일 경로
Visual Stuido	simulator/msvs/Application.sln
CubeIDE	ProjectName.ioc
IAR Embedded Workbench 8	target/IAR8.x/application.eww
KEIL uVision V5	target/Keil/application.uvprojx

코드 파일 찾아보기를 알아보고 자신의 코드를 실행하는 방법을 봅시다. 하단 바에서 "Browse Code" 버튼을 클릭하십시오. 이렇게 하면 응용 프로그램 폴더에 파일 탐색기가 배치됩니다. 하단의 폴더로 이동합니다.

"MyApplication1/generated/gui_generated/include/gui_generated/main_screen/"

MainViewBase.hpp 파일을 엽니다. IDE에 따라 프로젝트 파일 경로는 위와 같습니다.

안내 모든 프로젝트 파일이 기본적으로 존재하는 것은 아닙니다. 도구 체인을 변경하려면 CubeMX 도구를 사용해야 합니다. TouchGFX와 연계한 IDE 사용 페이지 에서 자세한 내용을 확인하시길 바랍니다.

다. 코드 구현하기

(1) 메서드의 구조 확인하기

MainViewBase.hpp

```
//********************************************************************************/
/********** THIS FILE IS GENERATED BY TOUCHGFX DESIGNER, DO NOT MODIFY ***********/
/********************************************************************************/
#ifndef MAINVIEWBASE_HPP
#define MAINVIEWBASE_HPP

#include <gui/common/FrontendApplication.hpp>
#include <mvp/View.hpp>
#include <gui/main_screen/MainPresenter.hpp>
#include <touchgfx/widgets/Image.hpp>
#include <touchgfx/widgets/Button.hpp>
#include <touchgfx/widgets/TextAreaWithWildcard.hpp>

class MainViewBase : public touchgfx::View<MainPresenter>
{
public:
  MainViewBase();
  virtual ~MainViewBase() {}
  virtual void setupScreen();

  /*
    * Custom Action Handlers
    */
  virtual void buttonUpClicked()
  {
    // Override and implement this function in MainView
  }

  virtual void buttonDownClicked()
  {
    // Override and implement this function in MainView
  }
...
```

새 가상 메서드는 MainViewBase 클래스의 Public 구조에서 확인 할 수 있습니다. 현재 생성된 메서드의 내용은 비어 있습니다. 설계자는 이러한 메서드를 MainView인 하위 클래스에서 구현합니다.

(2) 가상 메서드 구현하기

(가) 하위 클래스에 함수 선언하기

MainView.hpp

```
#ifndef MAIN_VIEW_HPP
#define MAIN_VIEW_HPP

#include <gui_generated/main_screen/MainViewBase.hpp>
#include <gui/main_screen/MainPresenter.hpp>

class MainView : public MainViewBase
{
public:
  MainView();
  virtual ~MainView() {}
  virtual void setupScreen();
  virtual void tearDownScreen();
  virtual void buttonUpClicked();
  virtual void buttonDownClicked();
}
```

" MyApplication1/gui/include/gui/main_screen/MainView.hpp" 에서 파일을 엽니다. 이 파일을 열고 "MainView" 클래스에 두 개의 함수 선언합니다.

(나) 가상 메서드 구현하기

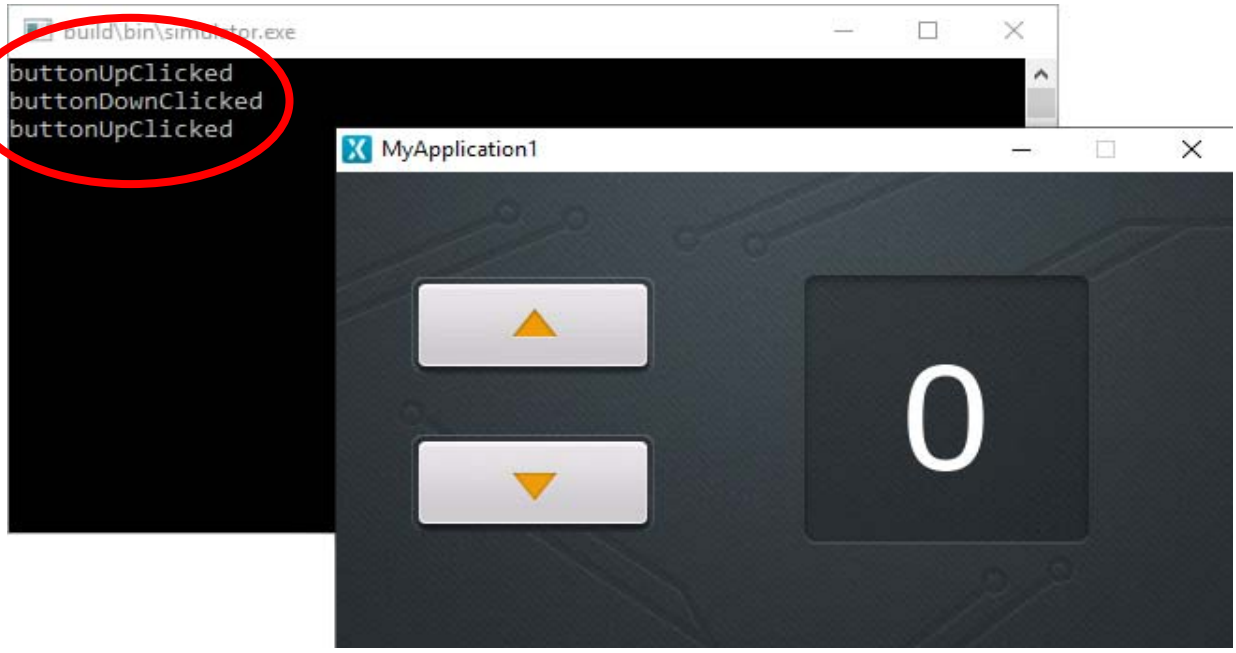
MainView.cpp

```
#include <gui/main_screen/MainView.hpp>

MainView::MainView()
{

}

void MainView::setupScreen()
{
  MainViewBase::setupScreen();
}

void MainView::tearDownScreen()
{
  MainViewBase::tearDownScreen();
}

void MainView::buttonUpClicked()
{
  touchgfx_printf("buttonUpClicked\n")
}

void MainView::buttonDownClicked()
{
  touchgfx_printf("buttonDownClicked\n")
}
```

다음 동작은 .cpp 파일에 구현을 통해 두 개의 메서드을 추가하는 것입니다.
"MyApplication1/gui/src/main_screen/MainView.cpp"에서 파일을 엽니다.

위의 구현에서 touchgfx_printf에 대한 호출을 추가했습니다. 이 기능은 시뮬레이터를 실행할 때 텍스트의 내용을 출력하는 데 사용합니다. 이 단계에서는 버튼을 클릭하였을 때 해당 touchgfx_printf의 텍스트 내용이 출력될 것입니다. 이때 응용 프로그램에서는 텍스트의 내용이 출력되지 않습니다(시뮬레이터 실행결과 참고).

(다) 시뮬레이터 실행 결과



TouchGFX Designer에서 "Run Simulator"를 다시 클릭하여 코드를 실행합니다.
버튼을 여러 번 클릭하여 상호 작용 및 메서드가 작동하는지 확인합니다.

(3) 카운터 값 업데이트하기

(가) 새 정수 변수 추가하기

MainView.hpp

```
#ifndef MAIN_VIEW_HPP
#define MAIN_VIEW_HPP

#include <gui_generated/main_screen/MainViewBase.hpp>
#include <gui/main_screen/MainPresenter.hpp>

class MainView : public MainViewBase
{
public:
  MainView();
  virtual ~MainView() {}
  virtual void setupScreen();
  virtual void tearDownScreen();
  virtual void buttonUpClicked();
  virtual void buttonDownClicked();
protected:
  int counter;
}
```

마지막 동작은 사용자가 버튼을 누를 때 카운터 값을 업데이트하는 새 메서드 C++ 코드를 작성하는 것입니다. 그렇게 하려면 먼저 MainView.hpp 클래스의 protected 구조에 "int"를 사용하여 정수 변수 "counter"를 추가합니다.

(나) 가상 메서드 구현하기

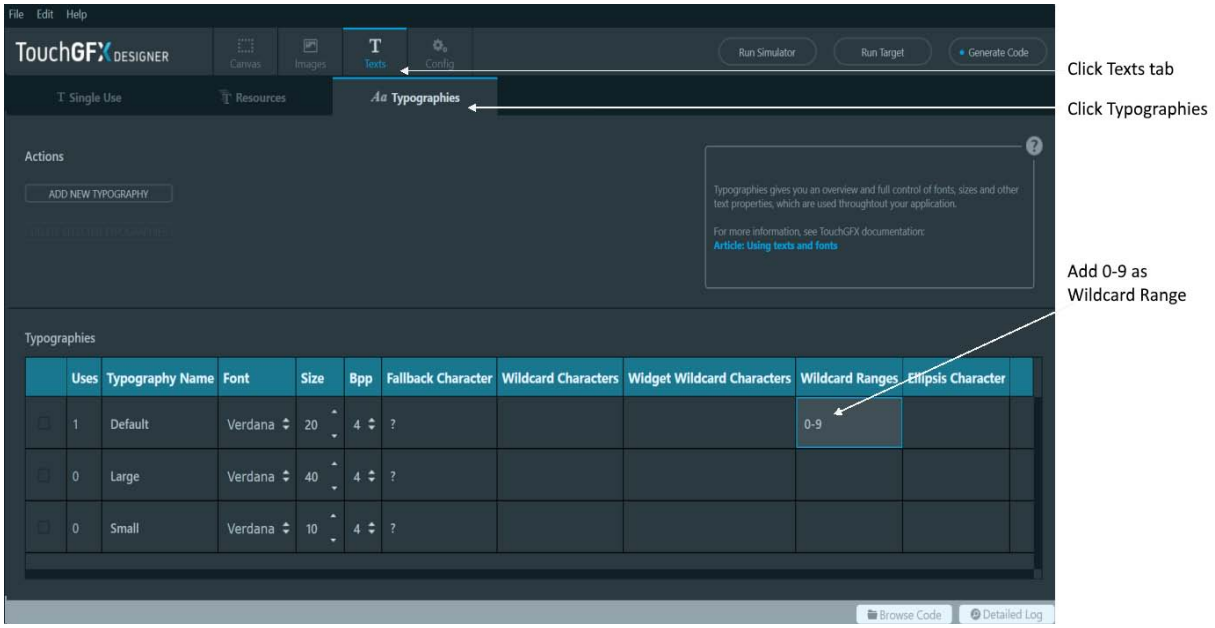
MainView.cpp

```
void MainView::buttonUpClicked()
{
  touchgfx_printf("buttonUpClicked\n")

  counter++;
  Unicode::snprintf(textCounterBuffer, TEXTCOUNTER_SIZE, "%d", counter);
  // Invalidate text area, which will result in it being redrawn in next tick.
  textCounter.invalidate();
}

void MainView::buttonDownClicked()
{
  touchgfx_printf("buttonDownClicked\n")

  counter--;
  Unicode::snprintf(textCounterBuffer, TEXTCOUNTER_SIZE, "%d", counter);
  // Invalidate text area, which will result in it being redrawn in next tick.
  textCounter.invalidate();
}
```

"MainView.cpp"에서 buttonUpClicked 은 카운터 값을 증가합니다. 그런 다음 새 값이 문자열로 변환되고 이전 단계에서 텍스트에 대해 구성한 10자 버퍼에 복사됩니다.

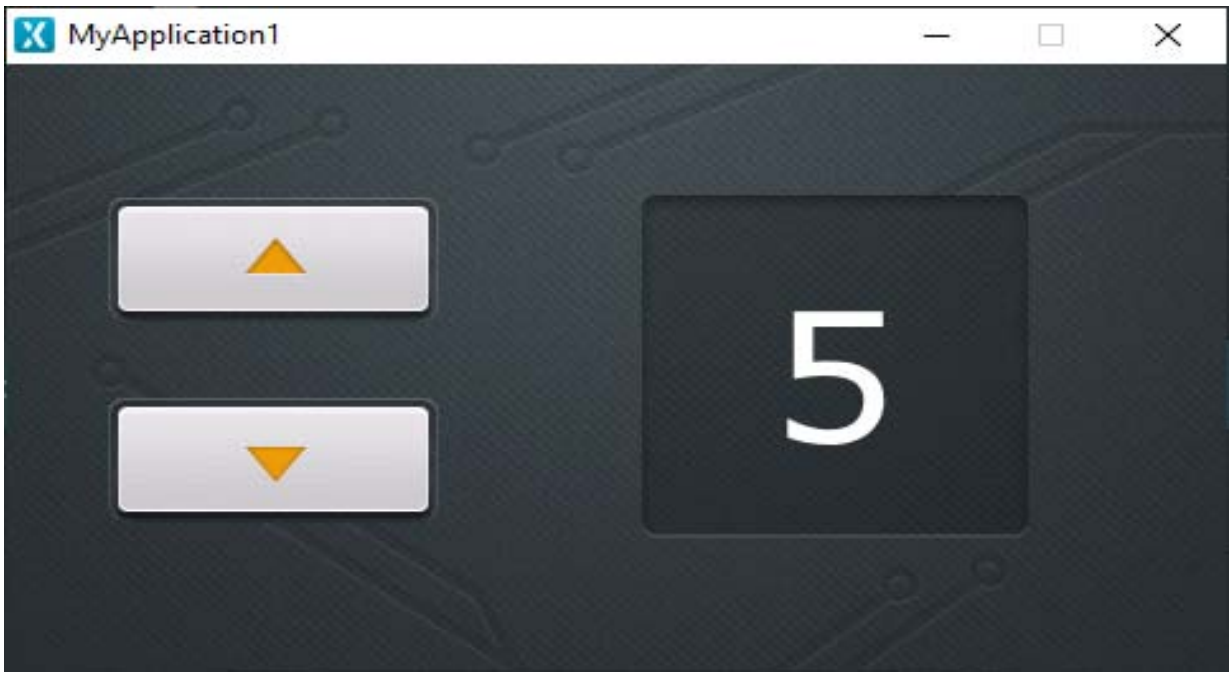
업데이트 후 "textCounter" 위젯에서 "invalidate()"를 호출합니다. 이렇게 하면 카운터 값이 업데이트된 후 TextArea가 다시 표현됩니다.

(다) 와일드카드 Typographies 설정하기



TouchGFX에는 사용된 글꼴에서 필요한 문자만 포함되었으므로 Typographies의 목록 중에 "Default" Typography가 0-9 문자를 포함하도록 TouchGFX Designer에서 설정해야 합니다. 설정을 하려면 "Text" 탭을 클릭한 다음 "Typographies"탭을 클릭합니다. 목록 중에 "Default"Typography의 "Wild card Range" 열에 "0-9" 범위를 추가합니다.

(라) 시뮬레이터 실행 결과



안내	프로그램은 현재와 같이 음수를 올바르게 처리하지 않습니다. buttonDownClicked()카운터가 0 아래로 떨어지지 않도록 함수에 가드를 추가 하거나 사용된 타이포그래피에 "-" 문자를 추가하여 이 문제를 해결할 수 있습니다 . 기본 타이포그래피의 와일드카드 문자 셀에 빼기("-")를 추가하면 됩니다.

튜토리얼 3 : 여러 화면이 있는 응용 프로그램

응용 프로그램에서 여러 화면을 만들고 화면 간에 데이터를 공유하는 방법을 배웁니다. 하나의 화면에는 사용하여 시와 분을 설정하고 실행 중인 시계가 있는 다른 화면에는 시간을 전달하는 시계를 시뮬레이션하는 응용 프로그램을 만들 것입니다.

또한 TouchGFX Designer를 사용하여 화면 변경과 같은 상호 작용을 기반으로 애니메이션을 만드는 방법을 배웁니다.

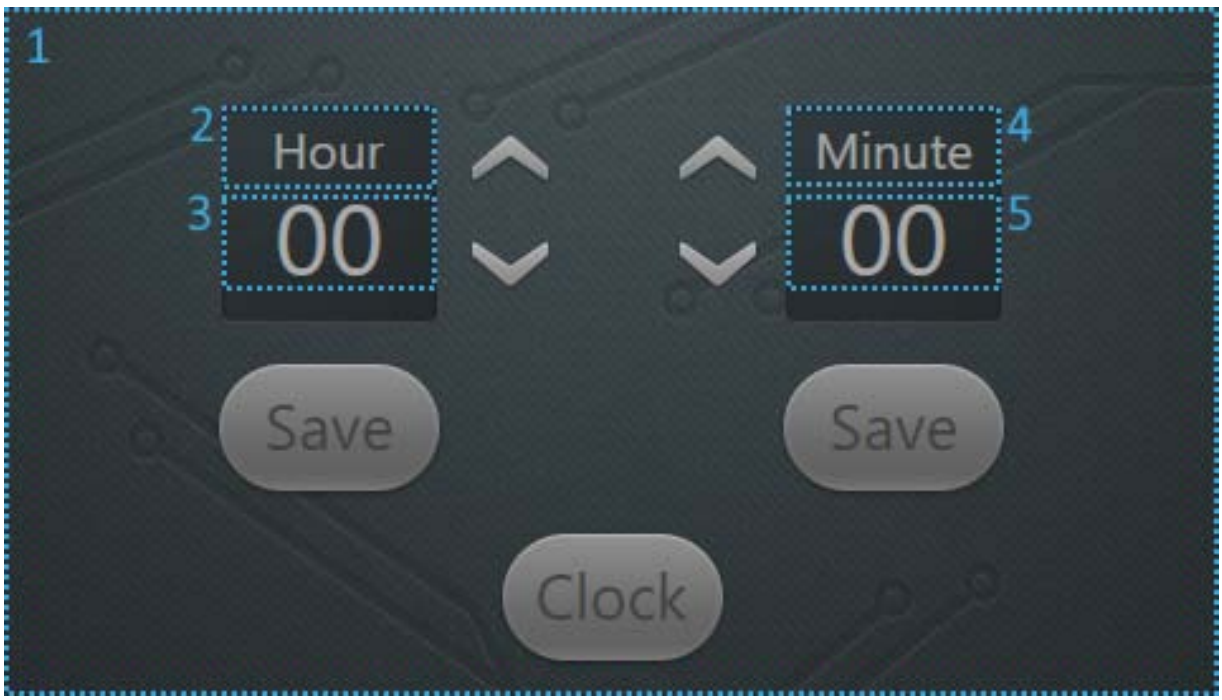
이 튜토리얼에서 사용할 이미지는 이 링크에서 다운로드 할 수 있습니다. 디스크의 디렉토리에 파일의 압축을 풉니다.

1. 두 개의 화면 설정하기

튜토리얼 2에서 했던 것처럼 새 프로젝트를 만듭니다. 그래픽은 480x272의 해상도를 위해 설계되었습니다. "STM32F746G Discovery Kit"용 애플리케이션 템플릿을 기반으로 하는 프로젝트의 경우 프로젝트에 이 해상도가 적용됩니다. 다른 해상도의 데모 보드가 있는 경우 그래픽을 수정하여 이에 맞게 수정할 수 있어야 합니다. 데모 보드가 없으면 “Simulator" 애플리케이션 템플릿을 기반으로 해상도를 480x272으로 입력해야 합니다.

가. Screen1 설정하기

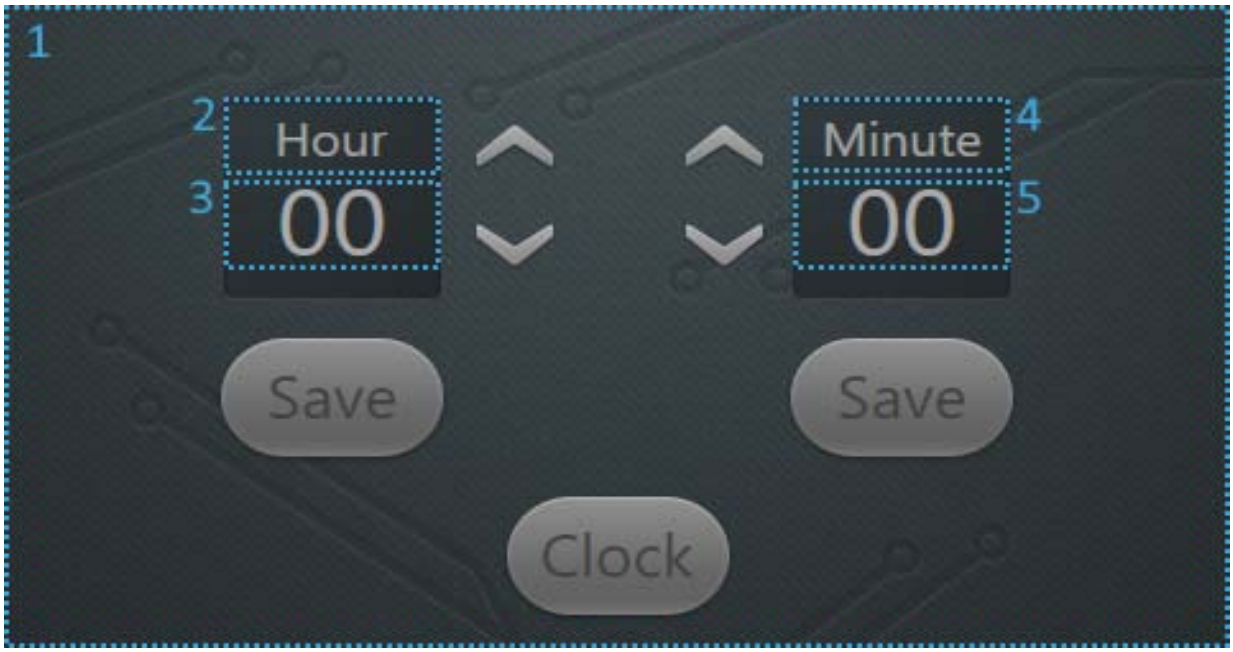
(1) 배경과 텍스트영역 속성 설정하기



응용프로그램의 배경과 텍스트 영역을 삽입합니다. 그림과 표에 같이 위젯을 추가하고 위젯의 속성을 변경합니다.

위 치	위 젯	속 성
1	Image	이름 : Background 위치 : X - 0, Y - 0 파일명:background_Screen1.png
2	TextArea	이름 : textAreaHourCaption 위치: X - 86, Y – 46, W - 85, H - 24 텍스트 - 텍스트 내용 : "Hour" - 텍스트 크기 : 20px - 텍스트 정렬 : 가운데 - 색상: "#FFABABAB"
3	TextArea	이름 : textAreaHour 위치: X - 87, Y – 70, W - 83, H - 50 텍스트 - 텍스트 내용 : < Default > - 텍스트 크기 : 40px - 텍스트 정렬 : 가운데 - 색상: "#FFABABAB" 와일드카드 - 초기값 : 00 - 버퍼 크기 : 3
4	TextArea	이름 : textAreaMinuteCaption 위치: X - 309, Y – 46, W - 85, H - 24 텍스트 - 텍스트 내용 : "Minute" - 텍스트 크기 : 20px - 텍스트 정렬 : 가운데 - 색상: "#FFABABAB"
5	TextArea	이름 : textAreaMinute 위치: X – 311, Y – 70, W - 83, H - 50 텍스트 - 텍스트 내용 : < Default > - 텍스트 크기 : 40px - 텍스트 정렬 : 가운데 - 색상: "#FFABABAB" 와일드카드 - 초기값 : 00 - 버퍼 크기 : 3

(2) 버튼 속성 설정하기



응용프로그램의 버튼을 삽입합니다. 그림과 표에 같이 위젯을 추가하고 위젯의 속성을 변경합니다.

위 치	위 젯	속 성
1	Button	이름 : buttonHourUp 위치 : X - 184, Y - 51 파일명 - Released Image : Up_arrow.png - Pressed Image : Up_arrow_pressed.png
2	Button	이름 : buttonHourDown 위치 : X - 184, Y - 93 파일명 - Released Image : Down_arrow.png - Pressed Image : Down_arrow_pressed.png
3	Button	이름 : buttonMinuteUp 위치 : X - 266, Y - 51 파일명 - Released Image : Up_arrow.png - Pressed Image : Up_arrow_pressed.png
4	Button	이름 : buttonMinuteDown 위치 : X - 266, Y - 93 파일명 - Released Image : Down_arrow.png - Pressed Image : Down_arrow_pressed.png

위 치	위 젯	속 성
5	Button With Label	이름 : buttonSaveHour 위치: X - 80, Y – 137 텍스트 - 텍스트 내용 : "Save" - 텍스트 크기 : 25px - 텍스트 정렬 : 가운데 색상 - Released Color : #FF424242 - Pressed Color : #FFA6A6A6 파일명 - Released Image : btn_round.png - Pressed Image: btn_round_pressed.png
6	Button With Label	이름 : buttonSaveMinute 위치: X - 303, Y – 137 텍스트 - 텍스트 내용 : "Save" - 텍스트 크기 : 25px - 텍스트 정렬 : 가운데 색상 - Released Color : #FF424242 - Pressed Color : #FFA6A6A6 파일명 - Released Image : btn_round.png - Pressed Image: btn_round_pressed.png
7	Button With Label	이름 : buttonClock 위치: X - 192, Y – 204 텍스트 - 텍스트 내용 : "Clock" - 텍스트 크기 : 25px - 텍스트 정렬 : 가운데 색상 - Released Color : #FF424242 - Pressed Color : #FFA6A6A6 파일명 - Released Image : btn_round.png - Pressed Image: btn_round_pressed.png

그래픽 요소를 설정한 상태에서 다음 단계는 버튼에 대한 트리거를 추가하여 선택한 값을 조정하고 저장합니다.

(3) 트리거 추가 및 속성 설정하기

Interaction	속 성
시간 증가 버튼을 눌렀을시	트리거 : 버튼 클릭 클릭한 소스 : buttonHourUp 결과 : 새로운 가상 함수 호출 함수 이름 : buttonHourUpClicked
시간 감소 버튼을 눌렀을시	트리거: 버튼 클릭 클릭한 소스 : buttonHourDown 결과 : 새로운 가상 함수 호출 함수 이름 : buttonHourDownClicked
분 증가 버튼을 눌렀을시	트리거 : 버튼 클릭 클릭한 소스 : buttonMinuteUp 결과 : 새로운 가상 함수 호출 함수 이름 : buttonMinuteUpClicked
분 감소 버튼을 클릭할시	트리거 : 버튼 클릭 클릭한 소스 : buttonMinuteDown 결과 : 새로운 가상 함수 호출 함수 이름 : buttonMinuteDownClicked
시간 저장 버튼을 클릭할시	트리거 : 버튼 클릭 클릭한 소스 : buttonSaveHour 결과 : 새로운 가상 함수 호출 함수 이름 : buttonSaveHourClicked
분 저장 버튼을 클릭할시	트리거 : 버튼 클릭 클릭한 소스 : buttonSaveMinute 결과 : 새로운 가상 함수 호출 함수 이름 : buttonSaveMinuteClicked

(4) 코드 추가하기

"Generate Code" 또는 "Run Simulator"을 누르면 지정된 가상 함수가 TouchGFX Designer에 의해 생성됩니다.

시간 및 분 값을 추적하기 위해 두 개의 카운터도 추가되며, 화살표 버튼를 누르면 해당 값이 시계 값에 맞게 변경됩니다.

하위클래스에서 버튼 기능에 대한 로직은 다음과 같은 방식으로 추가되어야 합니다.

Screen1View.hpp

```
...
public:
...
     virtual void buttonHourUpClicked();
     virtual void buttonHourDownClicked();
     virtual void buttonMinuteUpClicked();
     virtual void buttonMinuteDownClicked();

protected:
     int16_t hour;
     int16_t minute;
...
```

4개의 버튼에 대한 기능을 구현합니다. 시간 및 분 값을 추적하기 위해 2개의 카운터도 추가됩니다. 헤더파일에 위와 같이 함수 및 정수 변수를 추가합니다.

Screen1View.cpp

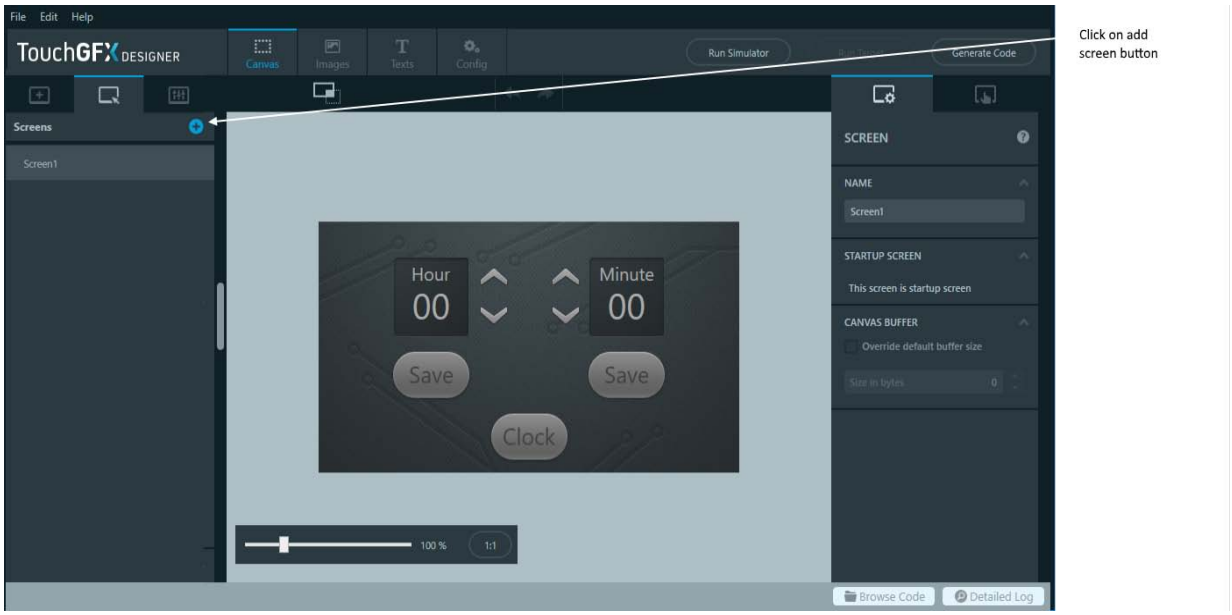
```
...
void Screen1View::buttonHourUpClicked()
{
    hour = (hour + 1) % 24; // Keep new value in range 0..23
    Unicode::snprintf(textAreaHourBuffer, TEXTAREAHOUR_SIZE, "%02d", hour);
    textAreaHour.invalidate();
}

void Screen1View::buttonHourDownClicked()
{
    hour = (hour + 24 - 1) % 24; // Keep new value in range 0..23
    Unicode::snprintf(textAreaHourBuffer, TEXTAREAHOUR_SIZE, "%02d", hour);
    textAreaHour.invalidate();
}

void Screen1View::buttonMinuteUpClicked()
{
    minute = (minute + 1) % 60; // Keep new value in range 0..59
    Unicode::snprintf(textAreaMinuteBuffer, TEXTAREAMINUTE_SIZE, "%02d", minute);
    textAreaMinute.invalidate();
}

void Screen1View::buttonMinuteDownClicked()
{
    minute = (minute + 60 - 1) % 60; // Keep new value in range 0..59
    Unicode::snprintf(textAreaMinuteBuffer, TEXTAREAMINUTE_SIZE, "%02d", minute);
    textAreaMinute.invalidate();
}
```

.cpp 파일에 4가지 기능에 대한 메서드는 위와 같은 방식으로 추가합니다.

나. Screen2 설정하기

Screen2는 Screen1에 저장된 값부터 실행 중인 시계가 있는 곳입니다.
시계 외에 Screen2는 시계 주위에 움직이는 원으로 구성되어 시계가 실행 중임을 나타냅니다. 추가로 Screen2에서 Screen1으로 돌아가기 위해 화면을 Screen1로 변경하는 버튼이 구현됩니다.

(1) 새로운 화면 추가하기



Screen2에 요소를 추가하기 전에 새로운 화면을 생성해야 합니다. 이것은 위 그림과 같이 TouchGFX Designer의 화면 추가 버튼으로 수행됩니다.

(2) 위젯 추가 및 속성 설정하기

시계와 원이 보기로 이동하여 해당 위치에 애니메이션 효과가 적용되며 시계는 왼쪽에서 원은 오른쪽에서 이동합니다. 따라서 두 위젯은 처음에 TouchGFX Designer에서 Canvas 외부에 배치됩니다. 위젯의 배치는 그림과 표와 설정합니다.

위 치	위 젯	속 성
1	Image	이름 : Background 위치 : X - 0, Y - 0 파일명:background_Screen2.png
2	Button	이름 : buttonSettings 위치 : X - 422, Y - 10 파일명 - Released Image : configuration.png - Pressed Image : configuration.png
3	TextArea	이름 : textClock 위치: X - 156, Y – 109, W - 156, H - 54 텍스트 - 텍스트 내용 : <Hour> : <Minute> - 텍스트 크기 : 40px - 텍스트 정렬 : 가운데 - 색상 : "#FFABABAB" 와일드카드1 - 초기값 : 00 - 버퍼 크기 : 3 와일드카드2 - 초기값 : 00 - 버퍼 크기 : 3
4	Circle	이름 : Circle 위치: X - 479, Y – 36, W - 200, H - 200 디자인 - 원의 중심 : X - 100, Y – 100 - 원의 방향 : Start – 0, End – 180 - 원의 반지름 : 72 - 선 너비 : 6 - Cap Style : 삼각형

(3) 트리거 추가 및 속성 설정하기

두 화면 사이를 전환하는 기능을 추가해야 합니다. Screen1의 "Clock" 버튼과 Screen2의 "Configuration"버튼에 상호작용을 추가합니다. 또한 Screen2에 들어갈 때 Screen2 바깥에 위치한 두 개의 요소가 제자리로 이동합니다.

위 치	위 젯	속 성
Screen1	"Screen2"로 변경	트리거 : 버튼 클릭 클릭한 소스 : buttonClock 결과 : 화면 변경 변경할 화면 : Screen2 전환 방법 : 덮어쓰기 전환 방향 : 북쪽
Screen2	"Screen1"로 변경	트리거 : 버튼 클릭 클릭한 소스 : 버튼설정 결과 : 화면 변경 변경할 화면 : Screen1 전환 방법 : 슬라이드 전환 방향 : 남쪽
Screen2	원을 제자리로 이동	트리거: 화면이 입력됨 결과 : 위젯 이동 이동할 위젯 : Circle 위치 : X - 140, Y - 36 변화율 : easeOutCubic 지속 시간 : 750ms
Screen2	텍스트를 제자리로 이동	트리거 : 화면이 입력됨 결과 : 위젯 이동 이동할 위젯 : textClock 위치 : X - 162, Y - 109 변화율 : easeOutCubic 지속 시간 : 750ms

(4) 코드 추가하기

Screen2View.hpp

```
public:
...
    virtual void handleTickEvent();

protected:
    int16_t hour;
    int16_t minute;
    int16_t tickCount;
    int16_t addStart;
    int16_t addEnd;
...
```

시계를 업데이트하고 런타임에 원에 애니메이션을 적용하려면 가상 기능 "handle TickEvent"이 사용됩니다. "handleTickEvent"는 TouchGFX 프레임워크에 의해 주기적으로 호출되어 활성 화면의 요소를 동적으로 업데이트할 수 있습니다. 이 경우에는 시계와 원이 됩니다.

Screen1과 유사하게 시간 및 분 카운터는 시계를 추적하는 데 사용됩니다. "handleTickEvent"는 시계가 업데이트되어야 하는 것보다 더 자주 호출 되기 때문에 시계 업데이트 사이의 틱 수를 결정하기 위해 tickCounter가 추가됩니다. 원에서 호의 각도를 업데이트하기 위해 addStart 및 addEnd 기능이 사용됩니다. 아래와 같이 Screen2View.hpp에 도시 된 바와 같이 함수 및 변수가 추가됩니다.

"int16_t"의 구현으로 Screen2View.cpp시계와 원을 업데이트하는 코드는 아래와 같습니다.

Screen2View.cpp

```
...
void Screen2View::handleTickEvent()
{
    if (tickCount == 60)
    {
        minute++;
        hour = (hour + (minute / 60)) % 24;
        minute %= 60;

        Unicode::snprintf(textClockBuffer1, TEXTCLOCKBUFFER1_SIZE, "%02d", hour);
        Unicode::snprintf(textClockBuffer2, TEXTCLOCKBUFFER2_SIZE, "%02d", minute);

        textClock.invalidate();

        tickCount = 0;
    }

    if (!textClock.isMoveAnimationRunning())
    {
        tickCount++;
        if (circle.getArcStart() + 340 == circle.getArcEnd())
        {
            addStart = 2;
            addEnd = 1;
        }
        else if (circle.getArcStart() + 20 == circle.getArcEnd())
        {
            addStart = 1;
            addEnd = 2;
        }
        circle.invalidate();
        circle.setArc(circle.getArcStart() + addStart, circle.getArcEnd() + addEnd);
        circle.invalidate();
    }
}
```

튜토리얼 2에서 배운 것처럼 와일드카드에 사용된 문자를 추가해야 합니다. 이 경우 TextArea에 사용되는 Typographies에 대해 "wildcard range" 열에 "0-9"를 추가해야 합니다 .

안내	이 단계에서는 Circle 위젯이 사용되었습니다. Circle 페이지에서 Circle 위젯에 대해 자세히 알아보세요 .

2. 데이터 저장하기

이 단계에서는 화면 전환 시 데이터를 저장하는 방법과 저장된 데이터를 검색하는 방법을 보여줍니다.

TouchGFX 응용 프로그램은 "Model-View-Presenter" 디자인 패턴을 따르므로 보기에서 (예: 화면) 조작된 데이터를 유지하려면 프레젠터를 통해 모델로 보내야 합니다. 해당 디자인 패턴에 대한 자세한 내용은 Model-View-Presenter 디자인 패턴 페이지에서 확인할 수 있습니다.

가. Screen1에 데이터 저장하기

(1) 모델에 시 / 분 데이터 추가하기

Model.hpp

```
...
public:
    void saveHour(int16_t saveHour)
    {
        hour = saveHour;
    }

    void saveMinute(int16_t saveMinute)
    {
        minute = saveMinute;
    }

    int16_t getHour()
    {
        return hour;
    }

    int16_t getMinute()
    {
        return minute;
    }

protected:
    int16_t hour;
    int16_t minute;
...
```

안내	int16_t 형식을 사용하려면 model.hpp 파일에도 #include <touchgfx/hal/types.hpp>가 포함되어야 합니다.

모델은 응용 프로그램에 대한 데이터를 보관하는 역할을 합니다. 버튼 상태, 현재 보이는 위젯 등과 같은 임시 데이터는 모델에 있어서는 안 됩니다.

모델을 통해 데이터를 저장하고 검색하려면 보호된 시간 및 분 값을 모델에 추가하고 이러한 값에 액세스할 수 있는 Public 구조의 메서드를 추가합니다.

Model.cpp에서 시와 분의 값을 초기화해야 합니다.

Model.cpp

```
...
Model::Model() : modelListener(0), hour(0), minute(0)
{
}
...
```

이 코드를 사용하면 모델에 시간과 분이 포함됩니다. 이 모델은 모든 사용자가 사용할 수 있으므로 사용자 간에 정보를 공유하는 데 권장되는 방법입니다.

이 모델은 UI가 하드웨어 주변기기 및 기타 소프트웨어 모듈과 같은 나머지 시스템에 연결할 수 있는 곳이기도 합니다.

(2) View에서 모델 엑세스 하기

Screen1Presenter.hpp

```
...
public:
    void saveHour(int16_t hour)
    {
        model->saveHour(hour);
    }

    void saveMinute(int16_t minute)
    {
        model->saveMinute(minute);
    }

    int16_t getHour()
    {
        return model->getHour();
    }

    int16_t getMinute()
    {
        return model->getMinute();
    }
...
```

이제 View에서 모델의 데이터를 액세스하려면 Screen1View의 모델에서 데이터를 로드하고 저장할 수 있는 함수을 다음과 같이 구현해야 합니다.

Screen2도 모델의 데이터에 액세스할 수 있어야 하므로 Screen2Presenter.hpp에 동일한 함수를 구현하십시오.

(3) 모델 데이터 업데이트 하기

(가) Screen1 데이터 업데이트 하기

Screen1View.cpp

```
...
void Screen1View::setupScreen()
{
    Screen1ViewBase::setupScreen();

    hour = presenter->getHour();
    minute = presenter->getMinute();

    Unicode::snprintf(textAreaHourBuffer, TEXTAREAHOUR_SIZE, "%02d", hour);
    Unicode::snprintf(textAreaMinuteBuffer, TEXTAREAMINUTE_SIZE, "%02d", minute);
}
...
```

이제 모델의 값으로 Screen1View에서 시와 분을 초기화하고 텍스트 영역에 대한 버퍼를 초기화할 수 있습니다.

시간 및 분 값을 저장하기 위해 상호 작용에 따라 생성된 가상 함수는 Screen1View.hpp에서 구현되며, 사용자를 통해 모델에 값을 저장합니다. 완료시 Screen1은 이제 모델에서 시간과 분에 대한 초기 값을 가져옵니다.

Screen1View.hpp

```
...
public:
    virtual void buttonSaveHourClicked()
    {
        presenter->saveHour(hour);
    }

    virtual void buttonSaveMinuteClicked()
    {
        presenter->saveMinute(minute);
    }
...
```

(나) Screen2 데이터 업데이트 하기

Screen2는 또한 해당 시와 분 값을 초기화 후에 모델과 동기화를 해야 합니다.

Screen1과 마찬가지로 텍스트 시계에 표시되는 초기 값은 모델의 데이터와 일치 해야 합니다.

Screen2View.cpp

```
...
void Screen2View::setupScreen()
{
    Screen2ViewBase::setupScreen();

    hour = presenter->getHour();
    minute = presenter->getMinute();

    Unicode::snprintf(textClockBuffer1, TEXTCLOCKBUFFER1_SIZE, "%02d", hour);
    Unicode::snprintf(textClockBuffer2, TEXTCLOCKBUFFER2_SIZE, "%02d", minute);
}
...
```

모델에서 시간과 분을 가져옵니다. 업데이트된 값은 화면을 종료할 때 모델로 다시 전송되어야 합니다. (화면1의 구성 화면으로 이동)

그러면 업데이트된 시간 및 분 값이 구성 화면으로 이동하기 직전에 모델에 전송됩니다.

튜토리얼 4 : 사용자 정의로 스크롤 휠 만들기

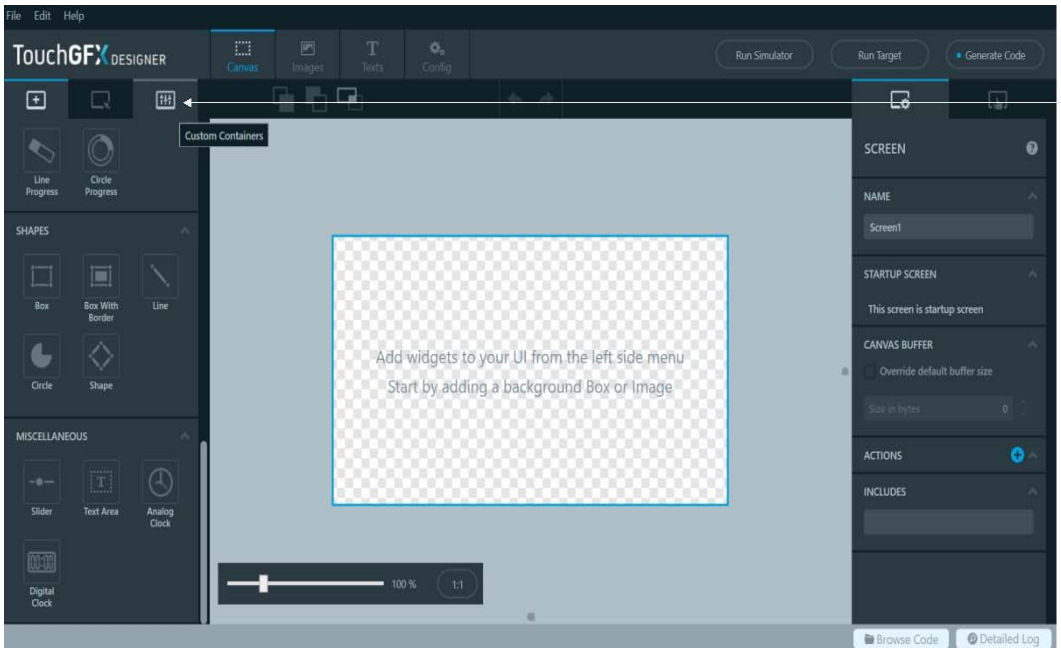
이 튜토리얼에서는 두 가지 위젯(사용자 정의 컨테이너 및 스크롤 휠)을 작성하고 구성하는 방법에 대해 알아봅니다. 사용자 정의 컨테이너는 다른 여러 위젯을 결합하여 사용자 정의 위젯을 작성하고 사용자 정의 컨테이너의 위젯에 대한 특정 동작을 추가할 수 있는 위젯입니다. 스크롤 휠은 여러 선택 가능한 항목으로 구성된 스크롤 가능 메뉴를 만드는 데 사용되는 위젯입니다. 또한 이 튜토리얼에서는 위젯의 동작을 변경하기 위해 사용자 코드를 작성하는 방법에 대해 설명합니다.

사용자 정의 컨테이너 및 스크롤 휠에 대한 자세한 내용은 Custom Container 및 Scroll Wheel 의 두 페이지에서 찾을 수 있습니다 .

튜토리얼의 그래픽은 이 링크 에서 다운로드할 수 있습니다. 자산 아래의 이미지 폴더에서 파일의 압축을 풉니다. 이 튜토리얼에서 사용되는 프로젝트의 경우 “MyApplication2₩assets₩images” 입니다.

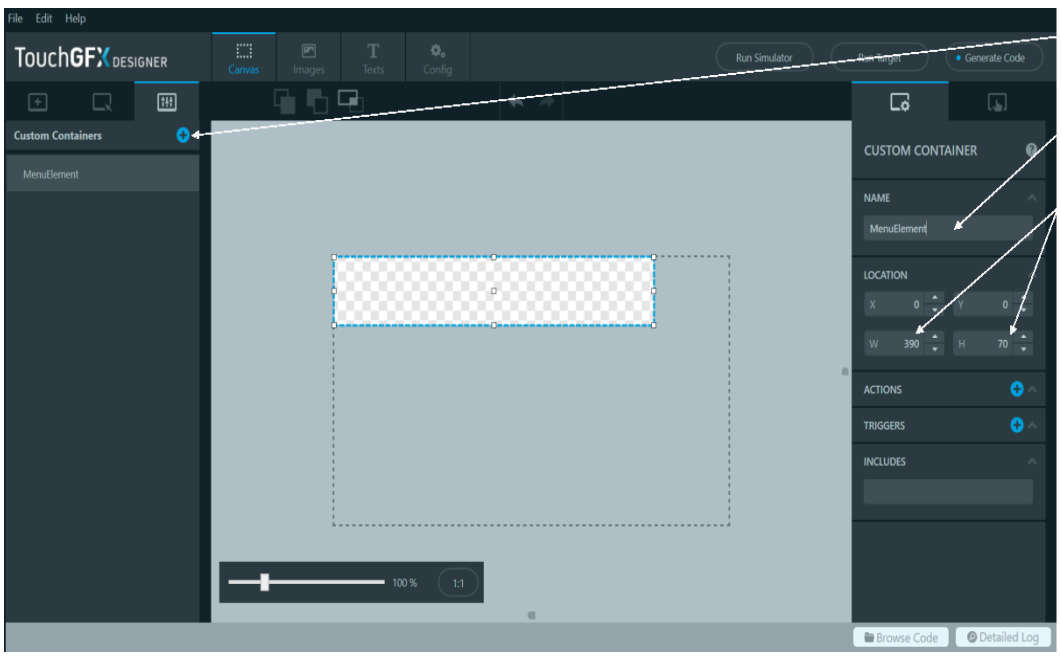
1. 사용자 정의 컨테이너 생성하기

가. 사용자 정의 컨테이너 메뉴 선택하기



TouchGFX Designer로 새 프로젝트를 생성하여 시작합니다. 새 프로젝트가 준비되면 TouchGFX Designer의 화면 탭에서 "Custom Container"로 변경합니다.

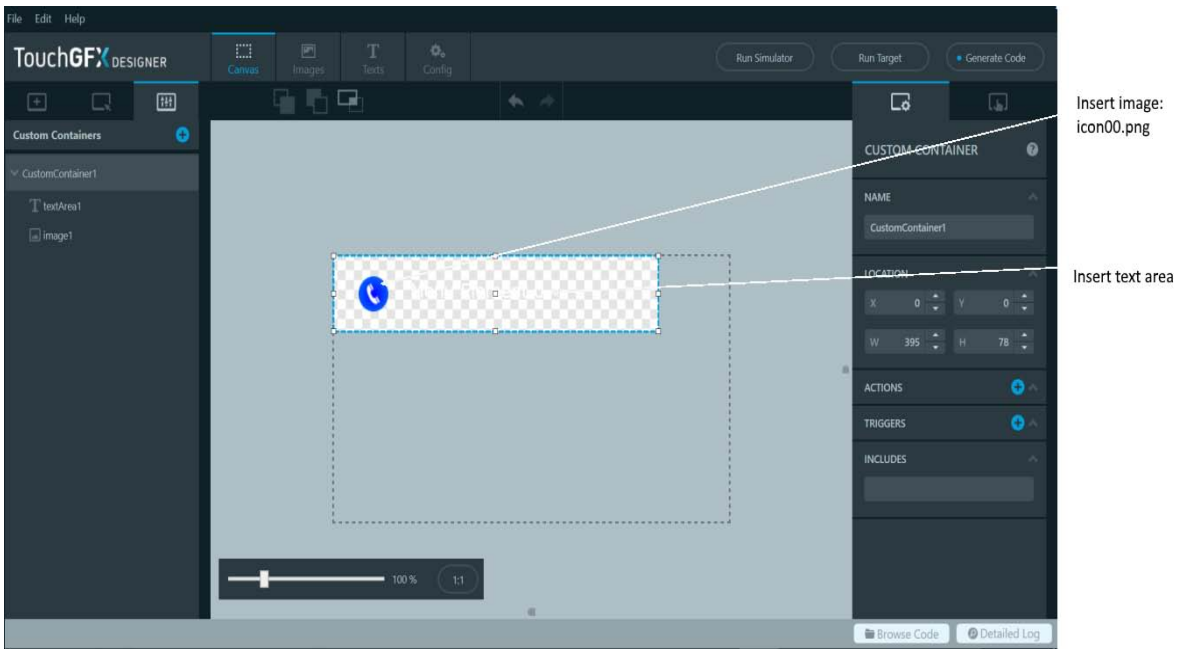
나. 사용자 정의 컨테이너 생성 및 속성 설정하기



사용자 정의 컨테이너 생성을 위한 탭은 Screens 탭과 유사하며, 사용자 정의 컨테이너를 새로 생성하는 방법은 새 화면을 생성하는 방법과 동일합니다. Custom Container가 생성된 후 Custom Container를 제외한 위젯을 추가하고 Custom Container의 크기와 이름을 편집할 수 있습니다.

사용자 정의 컨테이너 탭에서 파란색 더하기 아이콘을 사용하여 새로운 사용자 정의 컨테이너를 만들고 이름을 "MenuElement"로 바꾸고 너비(W)를 390으로, 높이(H)를 70으로 변경합니다

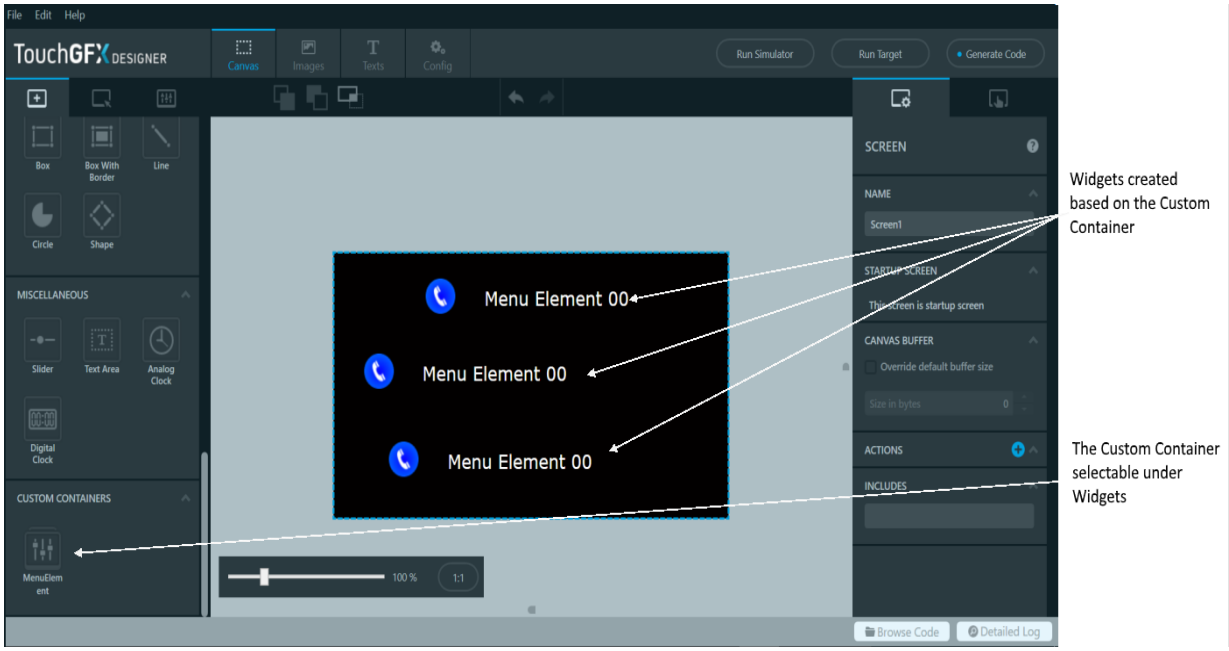
다. 사용자 정의 컨테이너에 위젯 추가하기



사용자 정의 컨테이너가 생성되고 속성이 설정되면 위젯을 사용자 정의 컨테이너에 추가할 수 있습니다. 사용자 정의 컨테이너는 이미지와 와일드카드가 있는 텍스트 영역으로 구성됩니다. 두 위젯은 아래와 같은 방식으로 삽입됩니다.

위 젯	속 성
Image	이름 : icon 위치 : X - 34, Y - 17 파일명 : icon00.png
TextArea	이름 : text 위치 : X - 93, Y – 23 텍스트 - 텍스트 내용 : Menu Element<value> - 텍스트 크기 : 20px - 텍스트 정렬 : 왼쪽 정렬 - 색상: "#FFFFFFFF" 와일드카드 - 초기값 : 00 - 버퍼 크기 : 3

라. 화면에 사용자 정의 컨테이너 추가하기



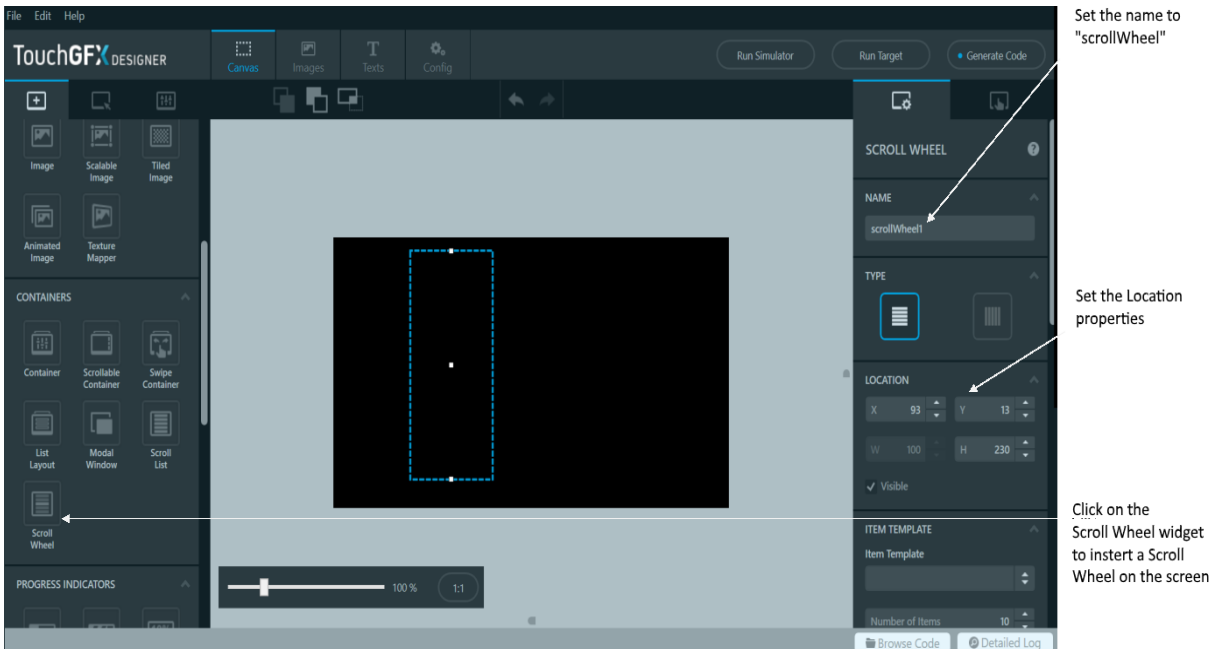
화면 탭으로 돌아가서 위젯 메뉴의 사용자 정의 컨테이너에서 "MenuElement"를 선택할 수 있습니다. 검정 배경을 놓고 생성한 사용자 정의 컨테이너를 위젯으로 화면에 추가합니다.

2. 스크롤 휠 만들기

2단계에서는 1단계에서 만든 사용자 정의 컨테이너인 "MenuElement"를 사용하여 스크롤 휠을 만듭니다. 1단계에서 설명한 것처럼 스크롤 휠은 선택 가능한 여러 항목이 포함된 스크롤 가능한 메뉴를 만드는 데 사용됩니다. 휠의 항목은 스크롤할 때 동적으로 업데이트되고 항목을 선택하면 포커스로 이동됩니다.

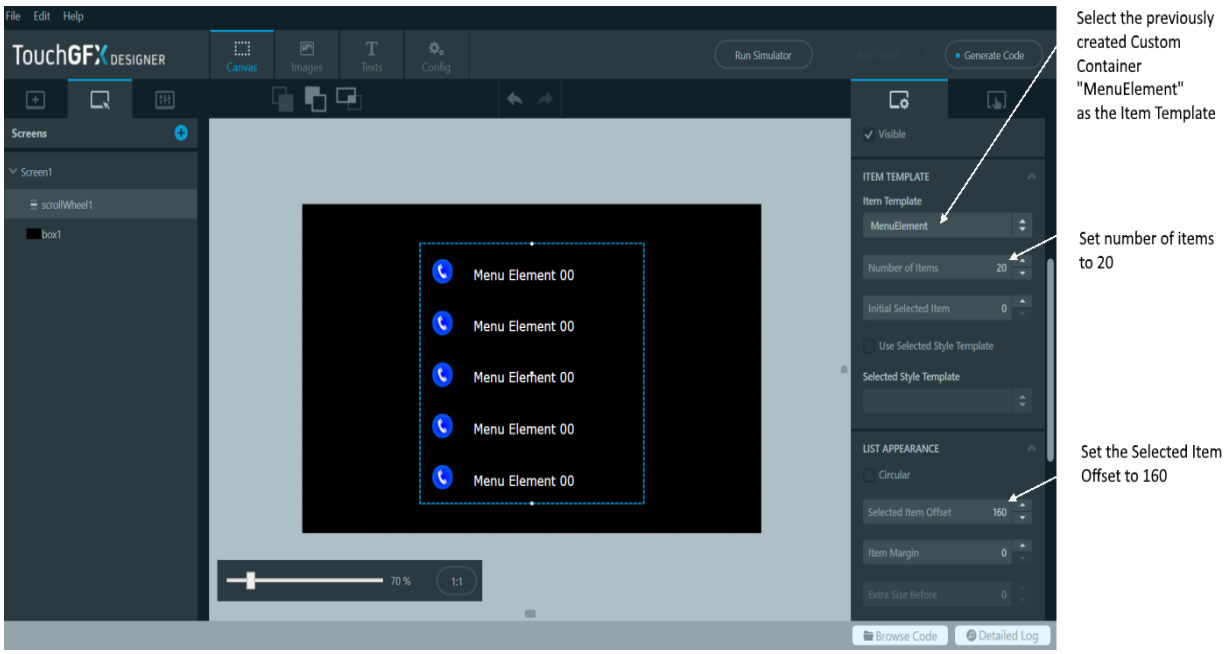
스크롤 휠에 항목을 추가하는 동작은 "Item Template"로 사용할 사용자 정의 컨테이너를 선택하여 수행됩니다. "Item Template"의 개념은 사용자 정의 컨테이너의 위젯을 스크롤 휠의 항목에 대한 기초로 사용하고 사용자 코드를 사용하여 런타임에 항목의 위젯을 업데이트합니다.

가. 스크롤 휠 생성 / 이름 및 위치 속성 설정하기



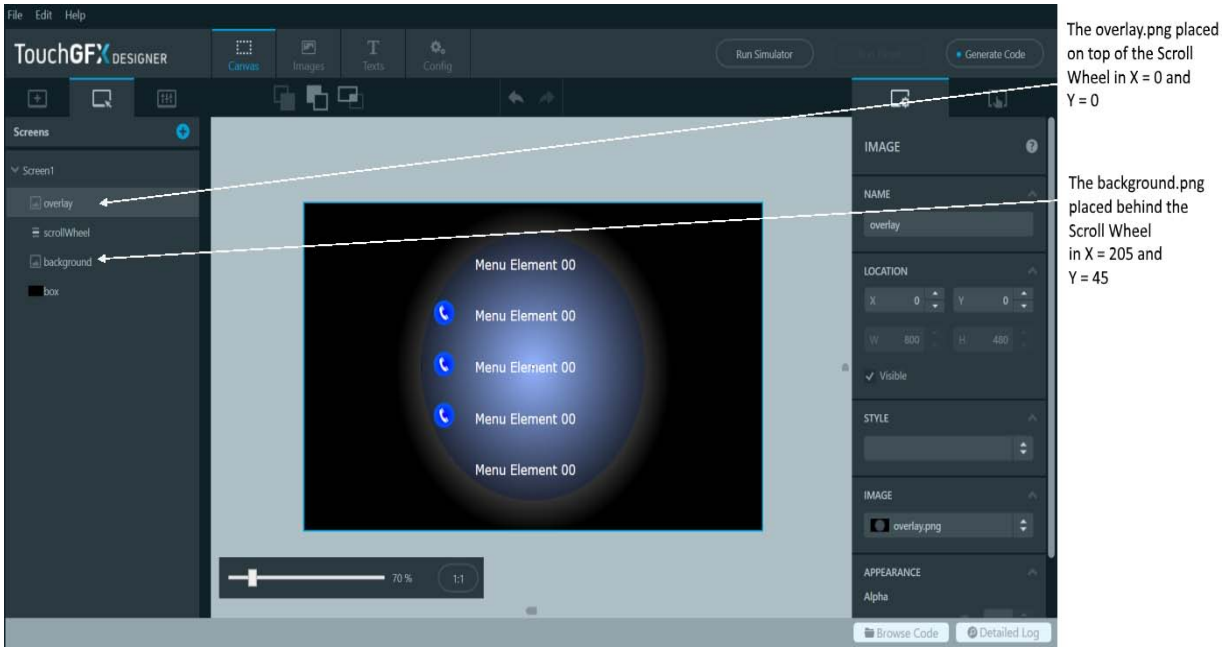
스크롤 휠을 생성하기 전에 화면에서 사용자 정의 컨테이너를 제거하고 검정 배경만 남겨둡니다. 컨테이너 섹션의 위젯 탭에 있는 스크롤 휠을 선택합니다. 스크롤 휠을 생성하고 위치 속성을 X = 208, Y = 45, H = 390으로 설정하고 이름을 "scrollWheel"로 변경합니다.

나. 스크롤 휠에 항목 추가하기



1단계에서 생성된 "MenuElement"를 “Item Template"으로 선택합니다. 스크롤 휠 속성 "Item Template" 아래의 드롭다운 목록에서 수행됩니다. 스크롤 휠의 항목 수는 "Item Template"에서도 설정됩니다. 이것을 20개 항목으로 설정합니다. 스크롤 휠은 선택한 항목에 초점을 맞추는 방식으로 작동하므로 “List Apperance" 속성에서 “Selected Item Offset"을 설정하여 선택한 항목의 위치를 설정합니다. 선택한 항목이 스크롤 휠의 중간에 있기를 원하므로 "Selected Item Offset"을 160으로 설정합니다

다. 스크롤 휠에 그래픽 추가하기



"scrollWheel”의 영역을 강조하기 위해 다운로드한 .zip 파일의 background.png 및 overlay.png 그림 두 개가 사용되며 애플리케이션에 이미지 위젯으로 추가됩니다. 두 이미지는 "scrollWheel"의 영역을 강조 표시하는 배경과 가장자리로 항목을 이동할 때 "scrollWheel”에 숨기는 오버레이입니다.

background.png 이미지는 X = 205, Y = 45에 배치되고 "scrollWheel” 뒤에 배치되어 "scrollWheel"의 항목이 배경 상단에 그려집니다. overlay.png 는 "scrollWheel" 상단의 X = 0 및 Y = 0에 배치되어 항목이 overlay.png 하단에 그려지고 overlay.png가 투명하지 않은 항목을 숨깁니다.

"scrollWheel"에 대한 정적 프로퍼티만을 조정했기 때문에 여기에 논리가 추가되지 않았습니다. 따라서 애플리케이션을 실행하면 모두 동일한 20개 항목으로 구성된 스크롤 가능한 메뉴가 나타납니다. 다음 단계에서는 런타임에 휠의 항목을 업데이트하는 사용자 코드를 사용하여 "scrollWheel"에 논리를 추가합니다.

라. 스크롤 휠에 사용자 코드 추가하기

TouchGFX 디자이너에서 생성 및 구성한 스크롤 휠인 "scrollWheel"을 사용하여 이 단계에서는 사용자 코드를 통해 "scrollWheel"의 항목을 업데이트하는 로직을 생성하여 항목의 위치에 따라 다른 그래픽을 표시합니다. 따라서 이 단계에서는 설계자 생성 코드를 사용자 코드와 통합하는 동작을 수행합니다. 설계자 코드와 사용자 코드를 통합하는 방법에 대한 자세한 설명은 코드 구조 페이지 에서 확인할 수 있습니다 .

(1) “MenuElement”의 이미지 및 텍스트 변경하기

MenuElement.hpp

```
#ifndef MENUELEMENT_HPP
#define MENUELEMENT_HPP

#include <gui_generated/containers/MenuElementBase.hpp>
#include <BitmapDatabase.hpp>

class MenuElement : public MenuElementBase
{
public:
    MenuElement();
    virtual ~MenuElement() {}

    virtual void initialize();

    void setNumber(int no)
    {
        Unicode::itoa(no, textBuffer, TEXT_SIZE, 10);
        switch (no % 7)
        {
        case 0:
        icon.setBitmap(Bitmap(BITMAP_ICON00_ID));
            break;
        case 1:
        icon.setBitmap(Bitmap(BITMAP_ICON01_ID));
            break;
        case 2:
        icon.setBitmap(Bitmap(BITMAP_ICON02_ID));
            break;
        case 3:
        icon.setBitmap(Bitmap(BITMAP_ICON03_ID));
            break;
        case 4:
        icon.setBitmap(Bitmap(BITMAP_ICON04_ID));
            break;
        case 5:
        icon.setBitmap(Bitmap(BITMAP_ICON05_ID));
            break;
        case 6:
        icon.setBitmap(Bitmap(BITMAP_ICON06_ID));
            break;
        }
    }
protected:
};

#endif // MENUELEMENT_HPP
```

스크롤 휠의 항목은 1단계에서 만든 "MenuElement" 를 기반으로 하므로 아이콘 변경 및 와일드카드 업데이트를 위한 사용자 코드를 "MenuElement"에 추가해야 합니다. 사용자 정의 컨테이너가 TouchGFX Designer에서 생성되면 사용자 코드가 통합되어야 하는 사용자 정의 컨테이너와 동일한 이름을 가진 .hpp 및 .cpp 파일이 생성됩니다. 예제 애플리케이션에서 "MenuElement"에 대해 생성된 파일의 위치는 다음과 같습니다.

"MyApplication2₩gui₩include₩gui₩containers₩MenuElement.hpp"
"MyApplication2₩gui₩src₩containers₩MenuElement.cpp"

"scrollWheel"의 항목이 텍스트와 아이콘을 변경할 수 있게 활성화하려면 함수 "setNumber(int no)"를 "MenuElement"에 추가해야 합니다. 이 함수는 변수 "no"를 사용하여 이미지 위젯에 표시할 아이콘을 결정하고 텍스트 영역 위젯에서 와일드카드를 "no" 값으로 변경합니다.

setNumber(int no)의 선언 및 구현은 위와 같이 MenuElement.hpp에서 수행됩니다.

(2) 스크롤 휠의 항목 업데이트 하기

Screen1View.hpp

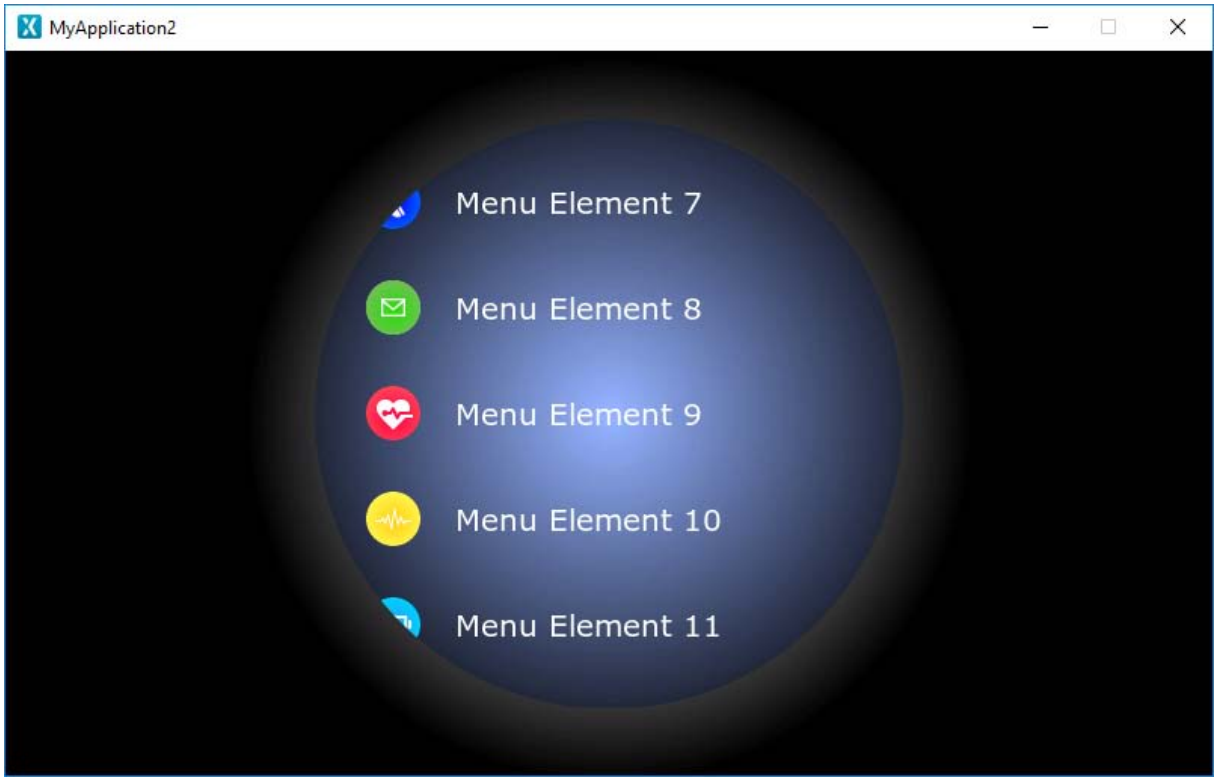
```
#ifndef SCREEN1VIEW_HPP
#define SCREEN1VIEW_HPP

#include <gui_generated/screen1_screen/Screen1ViewBase.hpp>
#include <gui/screen1_screen/Screen1Presenter.hpp>

class Screen1View : public Screen1ViewBase
{
public:
    Screen1View();
    virtual ~Screen1View() {}
    virtual void setupScreen();
    virtual void tearDownScreen();
    virtual void scrollWheelUpdateItem(MenuElement& item, int16_t itemIndex)
    {
        item.setNumber(itemIndex);
    }
protected:
};

#endif // SCREEN1VIEW_HPP
```

스크롤 휠을 만들 때 TouchGFX 디자이너는 스크롤 휠에 새 항목이 표시될 때마다 호출되는 가상 함수을 생성합니다. 사용자 코드에서 이 함수를 재정의하면 코드가 스크롤 휠의 항목과 상호 작용할 수 있습니다.

함수의 이름은 UpdatedItem이 추가된 스크롤 휠의 이름입니다. 현 튜토리얼의 경우 이 함수를 "scrollWheelUpdateItem(MenuElement& item, int16_t itemIndex)" 라고 합니다.

매개 변수 "itemIndex"는 현재 업데이트 중인 항목과 현재 스크롤 휠에 표시되는 MenuElement의 인스턴스에 대한 참조임을 알리는 인덱스 값입니다. 매개변수를 포함하여 업데이트 중임을 알리는 색인 setNumber()가 호출되어 매개변수 값에 따라 업데이트되는 항목의 색상이 변경됩니다. 인덱스. Scroll Wheel 항목을 업데이트 하는데 사용되는 코드는 위과 같습니다.

(3) 시뮬레이터 실행 결과



이제 응용 프로그램에 대한 시뮬레이터를 실행하면 항목의 텍스트에 색인 값이 포함되고 아이콘이 표시되는 항목에 따라 변경됨을 보여줍니다. 해당 이미지는 구현된 코드로 실행되는 시뮬레이터의 예를 보여줍니다.

마. 스크롤 휠에 사용자 지정 동작 추가하기

항목을 스크롤할 때 스크롤 휠을 원형 패턴으로 이동하여 다이얼과 유사한 패턴으로 이동하여 스크롤 휠에 대한 사용자 지정 동작을 추가합니다.

(1) “MenuElement”에 사용자 지정 동작 추가하기

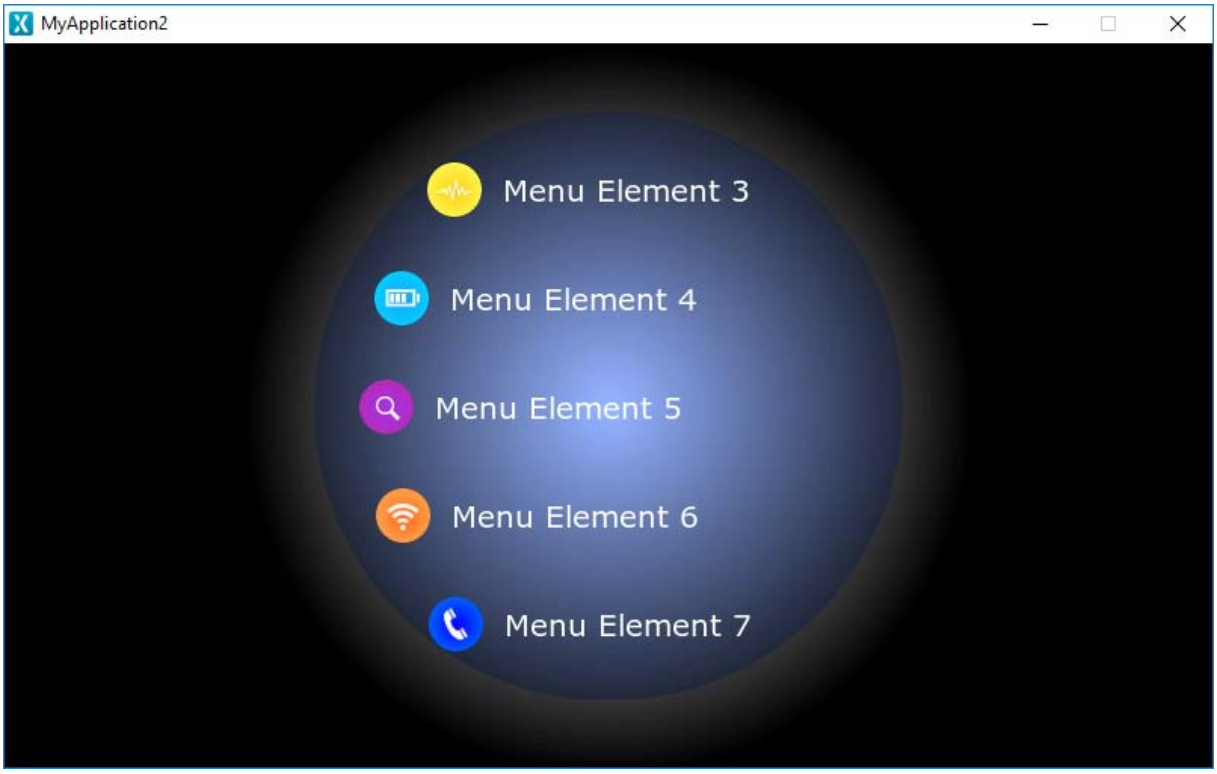
MenuElement.hpp

```
#ifndef MENUELEMENT_HPP
#define MENUELEMENT_HPP

#include <gui_generated/containers/MenuElementBase.hpp>
#include <BitmapDatabase.hpp>
#include <math.h>

class MenuElement : public MenuElementBase
{
public:
    MenuElement();
    virtual ~MenuElement() {}

    virtual void initialize();

    //Adjusts the position of the text and the icon, based in the calculated offset(x)
    void offset(int16_t x)
    {
        icon.moveTo(30 + x, icon.getY());
        text.moveTo(80 + x, text.getY());
    }

    //The new declaration and implementation of the setY() function
    virtual void setY(int16_t y)
    {
        //set Y as normal
        MenuElementBase::setY(y);

        const int circleRadius = 250;

        //center around middle of background
        y = y + getHeight() / 2 - 390 /2;

        //calculate x
        float x_f = circleRadius - sqrtf((float)((circleRadius * circleRadius) - (y * y)));
        int16_t x = (int16_t)(x_f + 0.5f);

        offset(x);
    }

    ...
```

스크롤 휠을 다이얼 패턴으로 움직이도록 하려면 스크롤 휠에 표시되는 각 항목에 대해 이미지 및 텍스트 위젯의 수평 위치를 이동하여 수행합니다. 이를 위해 "MenuElement" 에 대한 함수 "setY()"가 재정의됩니다 "setY()"함수는 사용자 정의 컨테이너가 수직 방향으로 이동할 때마다 호출되며 사용자 정의 컨테이너를 새 위치에 다시 그리는 데 사용됩니다. " setY()"를 재정의 하면 스크롤 휠이 이동할 때마다 이미지 및 텍스트 위젯을 재정렬할 수 있습니다.

위 이미지는 MenuElement.hpp에서 새로운 setY() 함수를 구현하고 두 위젯을 이동하는 방법을 설명합니다. 이때 math.h가 포함되어야 합니다.

(2) 시뮬레이터 실행 결과



함수 "setY()" 이 구현된 상태에서 시뮬레이터를 실행하면 스크롤 휠이 이제 오버레이의 곡선에 맞춰 다이얼 패턴으로 움직이는 것을 볼 수 있습니다.

안내	튜토리얼에서 사용된 개념에 대해 자세히 알아보려면 페이지에서 참고하시길 바랍니다. - Scroll Wheel 페이지 에서는 TouchGFX Designer에서 스크롤 휠을 생성 및 구성하는 방법과 사용자 코드에서 로직을 생성하는 방법을 설명합니다. - 사용자 정의 컨테이너 페이지 에서는 사용자 정의 컨테이너의 개념과 사용법에 대해 설명합니다.

튜토리얼 5 : 사용자 지정 트리거 및 동작 만들기

TouchGFX Designer를 사용하면 사용자 정의 트리거 및 동작으로 사용자 고유의 상호 작용 구성요소를 정의할 수 있습니다. 응용 프로그램의 각 화면은 코드뿐만 아니라 TouchGFX Designer 내에서 호출할 수 있는 동작 (C++에서는 빈 구현)을 포함할 수 있으며, 사용자 지정 컨테이너는 응용프로그램이 반응할 수 있는 트리거의 모음 (C++의 콜백과 동일)을 가질 수 도 있다.

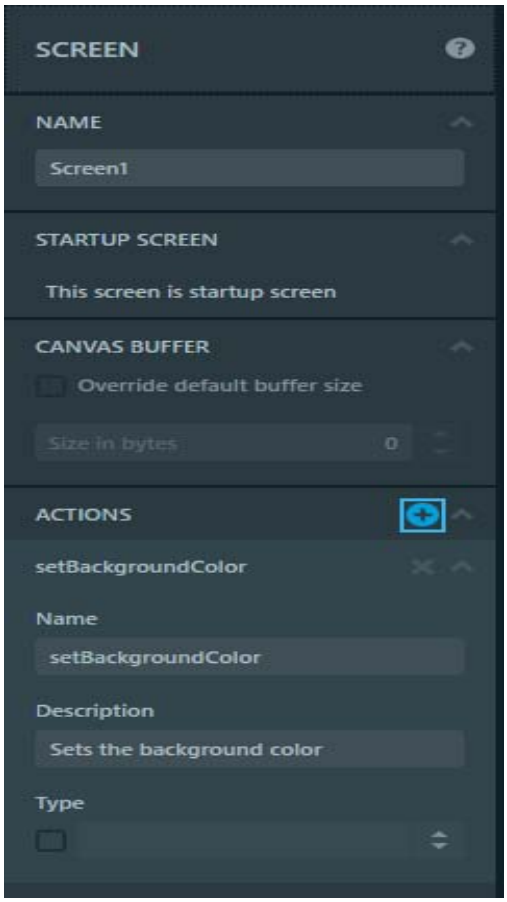
이 튜토리얼에서는 이 기능을 통해 보다 깨끗하고 역동적인 TouchGFX 응용 프로그램을 만들 수 있는 가능성을 알아봅니다.

1. 화면에 사용자 지정 동작 추가하기

가. 배경 상자와 버튼이 있는 새 응용 프로그램 만들기

크기가 480x272인 새 빈 응용 프로그램을 만들고 배경용 상자(이름을 “background” 으로 지정)와 버튼(이름을 “button"으로 지정)을 삽입합니다.

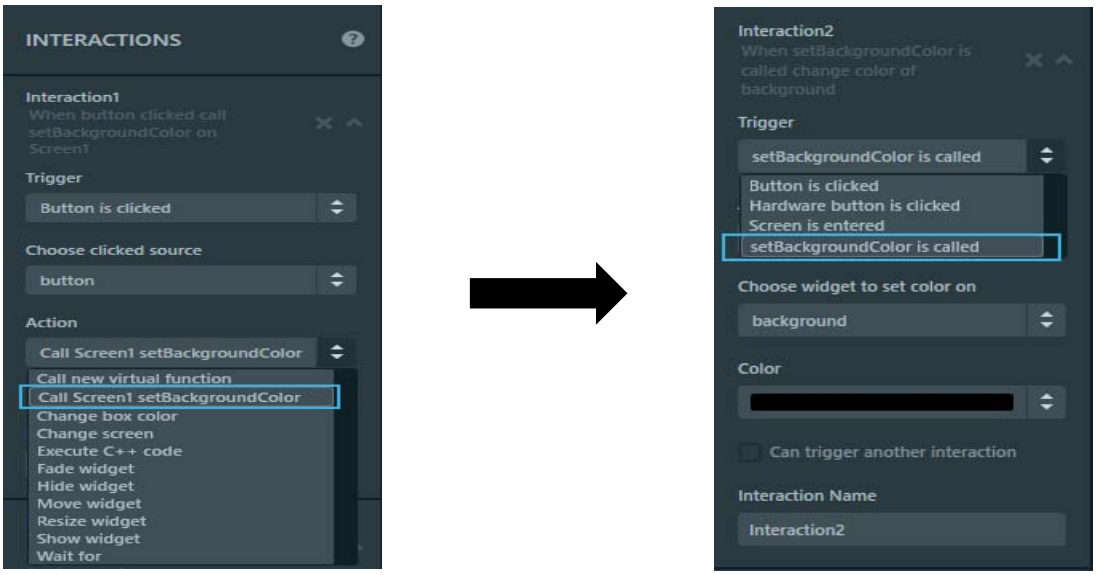
나. 응용 프로그램에 사용자 지정 동작 추가하기



다음으로 화면에 사용자 지정 동작을 추가하겠습니다. 화면의 Properties 탭에서 Screen을 선택하고 "ACTIONS" 그룹에서 + 버튼을 누르면 됩니다. 동작의 이름을 "setBackgroundColor"로 지정하고 "Description"칸에 ""Sets the background color" 으로 지정 합니다.

Screen1ViewBase.hpp에 setBackgroundColor()라는 가상 메서드를 생성되고 이 가상 메서드는 Screen1ViewBase.cpp에 빈 구현으로 구성되어 있습니다.

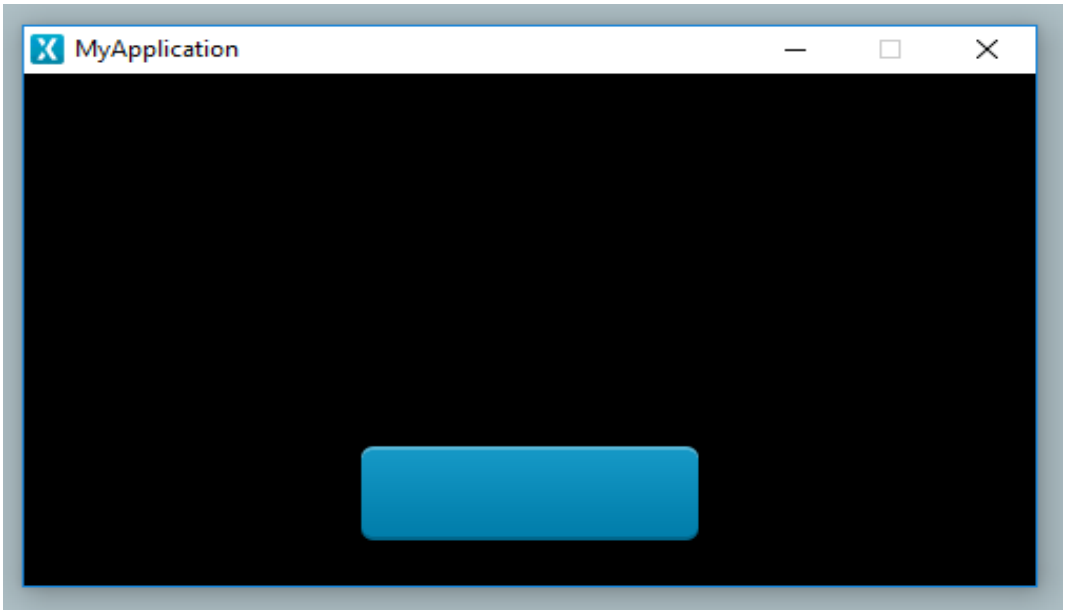
다. 사용자 지정 동작 설정하기



Screen1View.cpp파일의 사용자 코드에서 재정의 하거나 TouchGFX Designer를 통해 상호 작용을 만들어 메서드에 기능을 추가할 수 있습니다. 화면의 "ineraction" 탭으로 이동하고 버튼을 클릭할 때 새 메서드를 호출하는 상호 작용을 추가하여 사용해 봅니다.

이제 setBackgroundColor가 호출 될 때 실제로 어떤 일이 발생하는지 지정합니다. 이 동작은 새로운 사용자 지정 동작을 다른 상호 작용의 트리거로 사용하여 수행됩니다. "setBackgroundColor is call"의 트리거가 발생할 때 "Change box color" 동작을 사용하여 배경 상자 색상을 검정으로 설정하는 것으로 시작하겠습니다 .

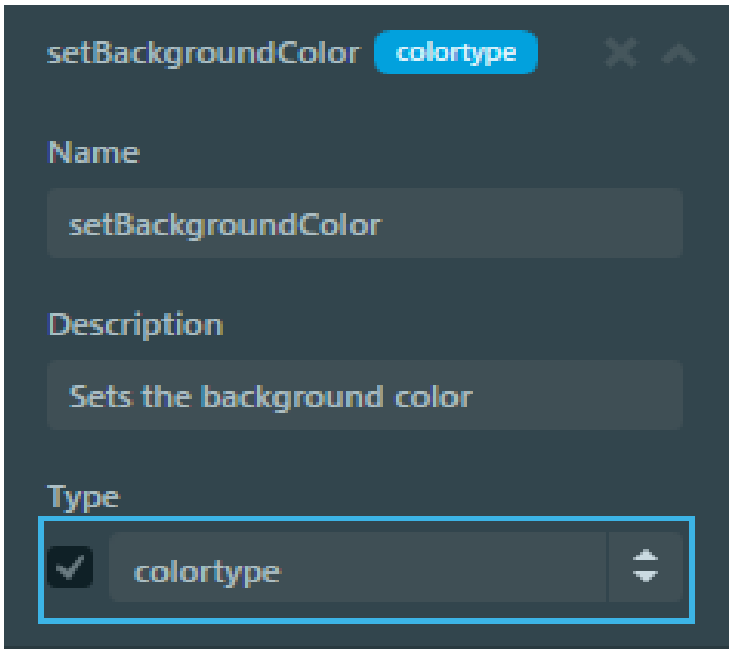
라. 시뮬레이터 실행 결과



시뮬레이터 실행 후 화면의 버튼을 누르면 배경색이 흰색에서 검정으로 바뀝니다.

2. 사용자 지정 동작에 값 전달하기

가. 사용자 지정 동작에 매개변수 설정하기

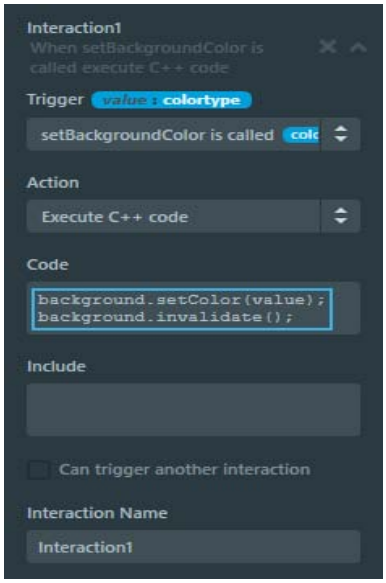


화면에서 상호 작용 탭으로 이동하면 새로운 상호 작용을 설정하게 되므로 각각의 상호 작용에 대해 x 버튼을 눌러 현재 상호 작용 두 개를 삭제합니다.

화면의 속성 탭으로 이동하여 setBackgroundColor라는 사용자 지정 동작으로 이동한 다음 동작으로 전달할 매개 변수 유형인 유형 및 입력 "colortype" 확인란을 선택 취소합니다(colortype은 색상을 설명하기 위한 기본 제공 TouchGFX 유형). 매개 변수의 이름을 지정할 수 없으며 이름이 "Value"으로 지정됩니다.

나. 매개변수 값을 사용한 트리거 설정하기

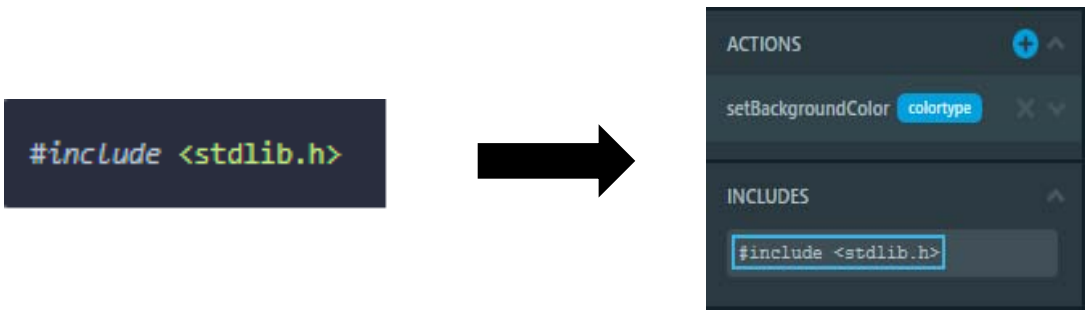
```
background.setColor(value);
background.invalidate();
```



다음으로 새로 추가된 매개 변수 값을 사용하는 상호 작용을 설정하겠습니다. "setBackgroundColor is call" 트리거와 "Execute C++ code"를 사용하여 이를 수행합니다. 우리는 우리의 새로운 매개 변수를 사용하여 배경 박스의 색상을 설정하기를 원하므로 실행할 코드는 다음과 같아야 합니다.

다. 버튼을 사용한 트리거 설정하기

(1) rand 함수 이용하기

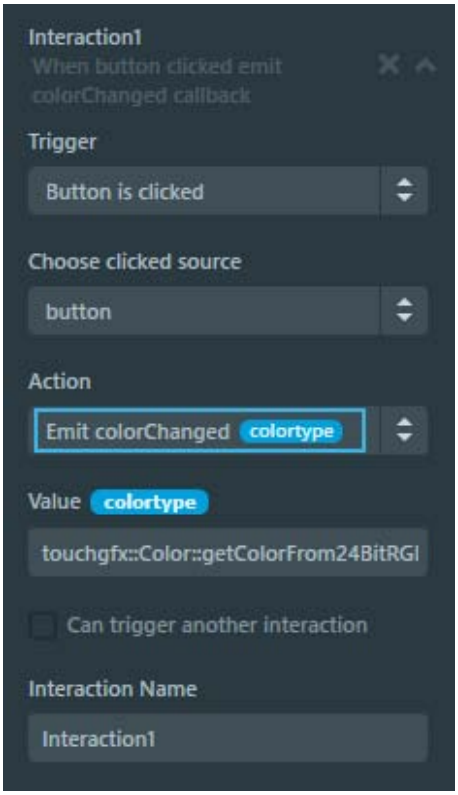


다음으로 버튼을 클릭했을 때 실제로 setBackgroundColor를 호출하는 상호 작용을 설정하겠습니다. "Button is clicked" 트리거및 "Call Screen1 setBackgroundColor" 동작을 사용하여 다른 상호 작용을 추가하고 값 속성이 예상되는 유형도 표시합니다.

stdlib.h의 함수 rand()를 사용하여 0에서 255 사이의 세 개의 무작위 숫자를 얻은 다음 이 값을 사용하여 색상을 지정함으로써 임의 색상을 "setBackgroundColor"로 전달합니다. rand()에 접근하기 위해서는 "stdlib.h"를 우리 애플리케이션에 포함시켜야 합니다. TouchGFX Designer 내에서 스크린과 맞춤형 컨테이너를 사용할 수 있습니다. 화면의 속성 탭으로 이동하여 "INCLUDES" 그룹에서 다음을 입력합니다.

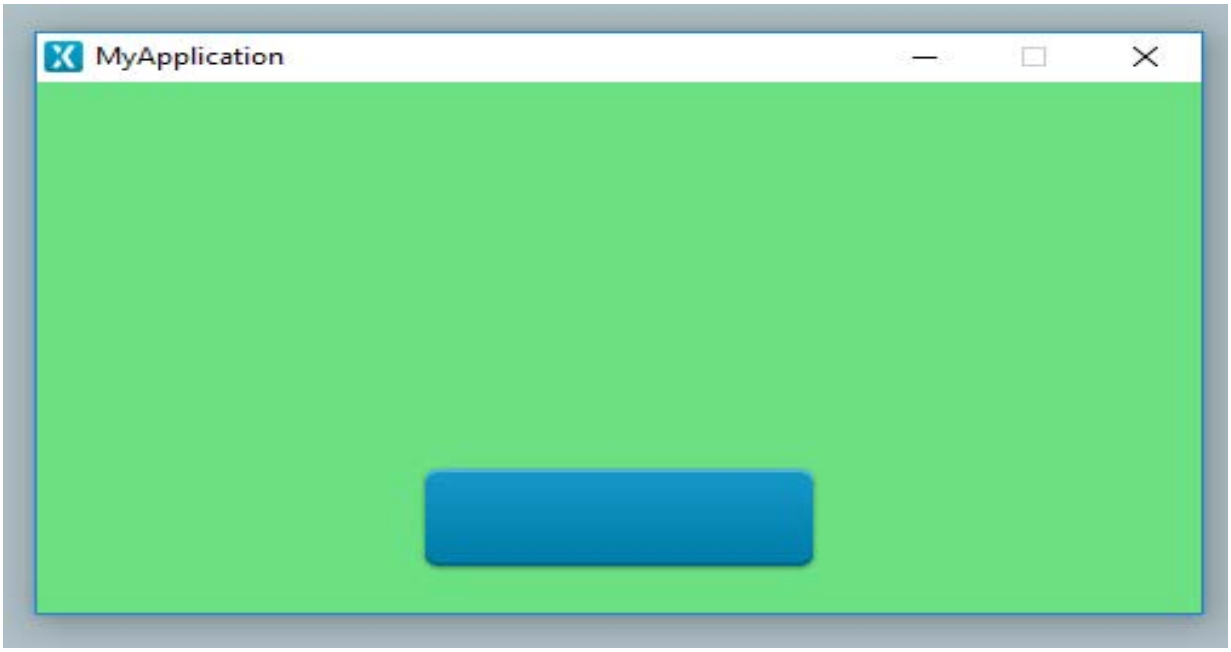
(2) 트리거값 속성 설정하기

```
touchgfx::Color::getColorFrom24BitRGB(rand()%256, rand()%256, rand()%256)
```



rand()을 사용하여 0에서 255 사이의 3개의 난수를 얻어 이를 사용해 색상값을 지정하고 임의의 색상값을 매개변수에 전달합니다.

라. 시뮬레이터 실행 결과



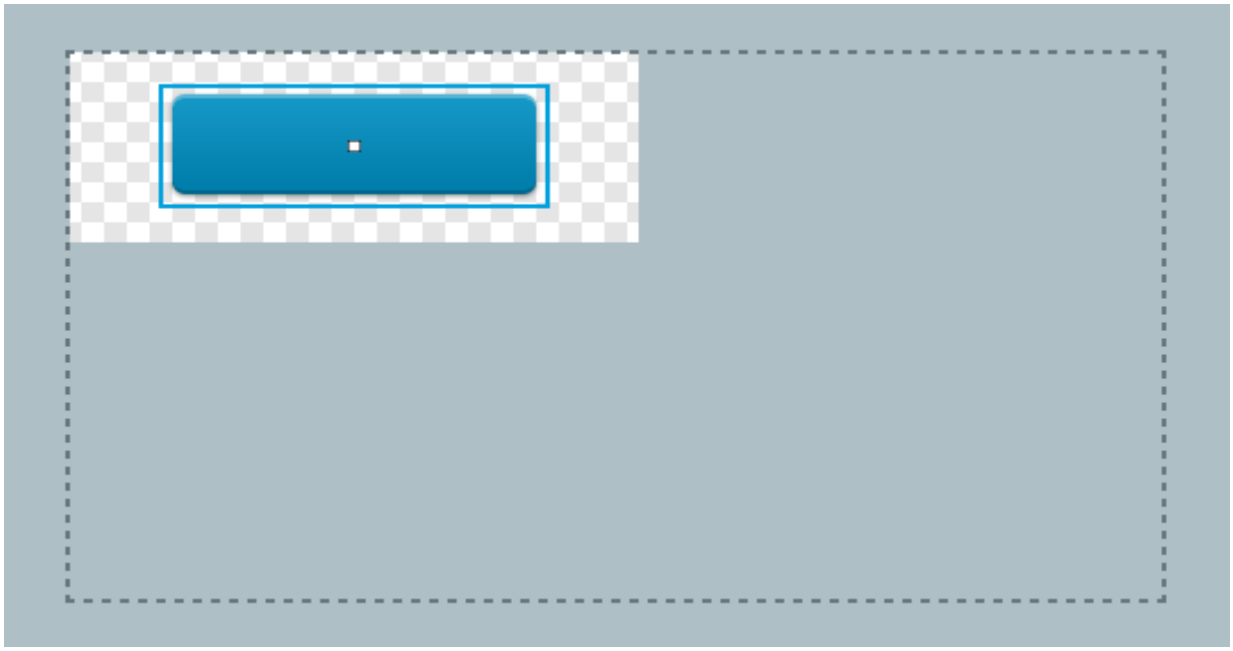
버튼을 클릭할 때마다 배경이 임의의 색으로 변경됩니다.

3. 사용자 정의 컨테이너에서 트리거 사용하기

사용자 정의 컨테이너는 사용자 정의 트리거 모음을 정의할 수도 있으므로 이 섹션에서는 다음을 수행하여 애플리케이션을 확장합니다.

- ColorEmitter라는 새 사용자 정의 컨테이너를 만듭니다.
- ColorEmitter에 "colorChanged"라는 사용자 지정 트리거를 추가합니다.
- colorChanged 트리거를 사용하여 버튼을 눌렀을 때 응용 프로그램에 임의의 색상 신호를 보냅니다.
- colorChanged 트리거를 수신하도록 화면에서 상호 작용 설정합니다.
- ColorEmitter가 보내는 색상을 사용하여 배경의 색상을 설정합니다.

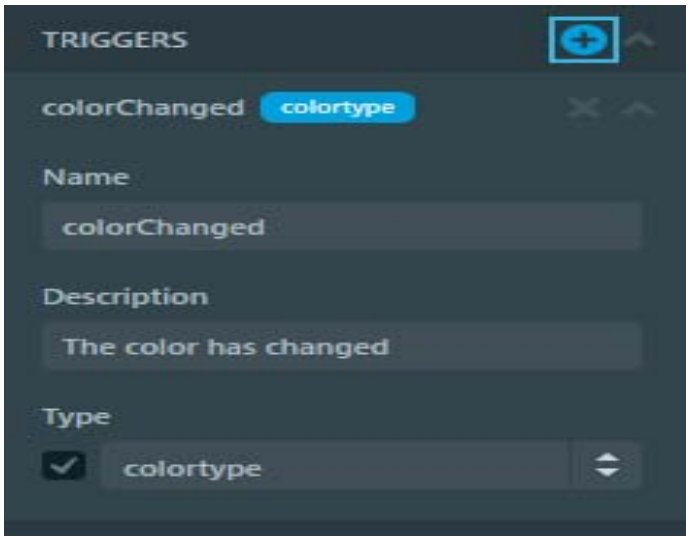
가. 사용자 정의 컨테이너 생성하기



사용자 정의 트리거를 사용하여 응용 프로그램의 일부 이벤트를 표시해 보겠습니다. 버튼 상호 작용이 임의 색상을 setBackgroundColor로 전달하는 대신 사용자 정의 컨테이너가 임의 색상을 화면으로 보내도록 한 다음에 사용자 정의 컨테이너가 통신하는 값이면 무엇이든 화면이 사용하도록 하겠습니다. 이는 응용 프로그램에서 서로 통신하는 서로 다른 UI 구성 요소의 간단한 예가 되어 더 작고 재사용 가능한 구성 요소를 만들 수 있습니다.

먼저 "ColorEmitter"라는 새 사용자 정의 컨테이너를 생성하고 "Button"이라고 부르는 단추를 생성합니다. 위 이미지와 유사한 것이 있어야 합니다

나. 사용자 정의 컨테이너에 사용자 지정 트리거 추가하기



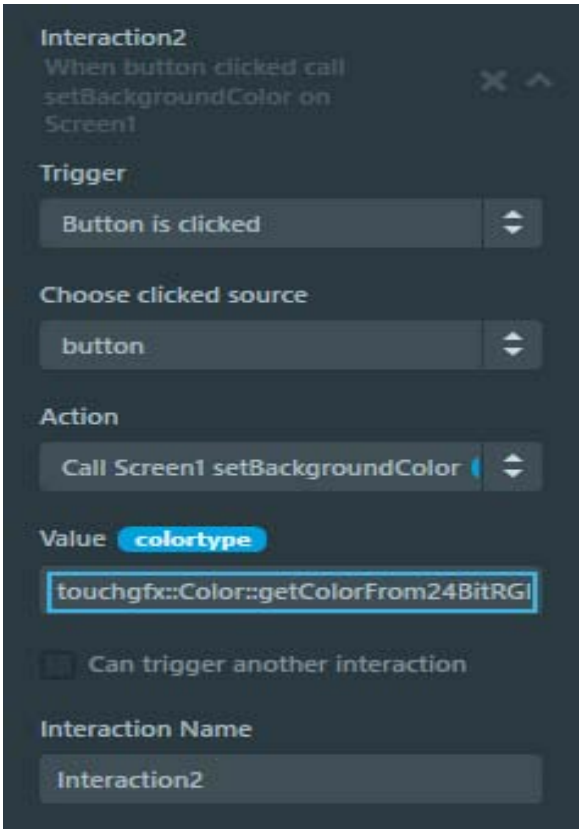
버튼을 클릭할 때마다 ColorEmitter가 화면에 임의의 색을 발산하도록 만들어 봅시다.

"ColorEmitter"가 화면에 색상을 발산하도록 하려면 먼저 사용자 지정 트리거를 생성해야 합니다. 사용자 정의 컨테이너의 속성 탭으로 이동하여 "Trigger" 그룹에 +버튼을 클릭합니다. 이 때 트리거 Name을 "colorChanged"로 지정하고 Description을 "The color has changed"으로 지정하고 Type을 "colortype"으로 지정합니다.

다. 트리거 속성 설정하기

(1) 트리거값 속성 설정하기

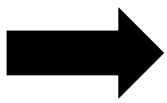
```
touchgfx::Color::getColorFrom24BitRGB(rand()%256, rand()%256, rand()%256)
```

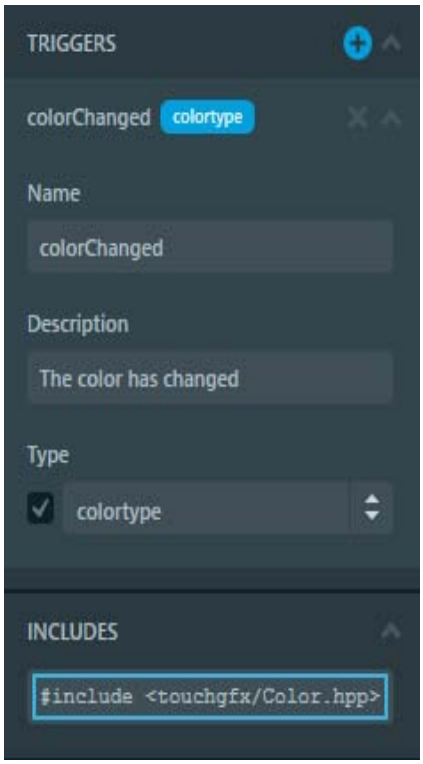


그런 다음 사용자 정의 컨테이너의 Interaction탭으로 이동하여 새로운 상호 작용을 만듭니다. 트리거 "Button is clicked" 및 "Emit colorChanged" 동작을 사용합니다.
이제 랜덤 색상을 전달하려고 하므로 값 속성에 대해 이전과 동일한 코드를 사용합니다.

(2) 헤더파일 추가하기

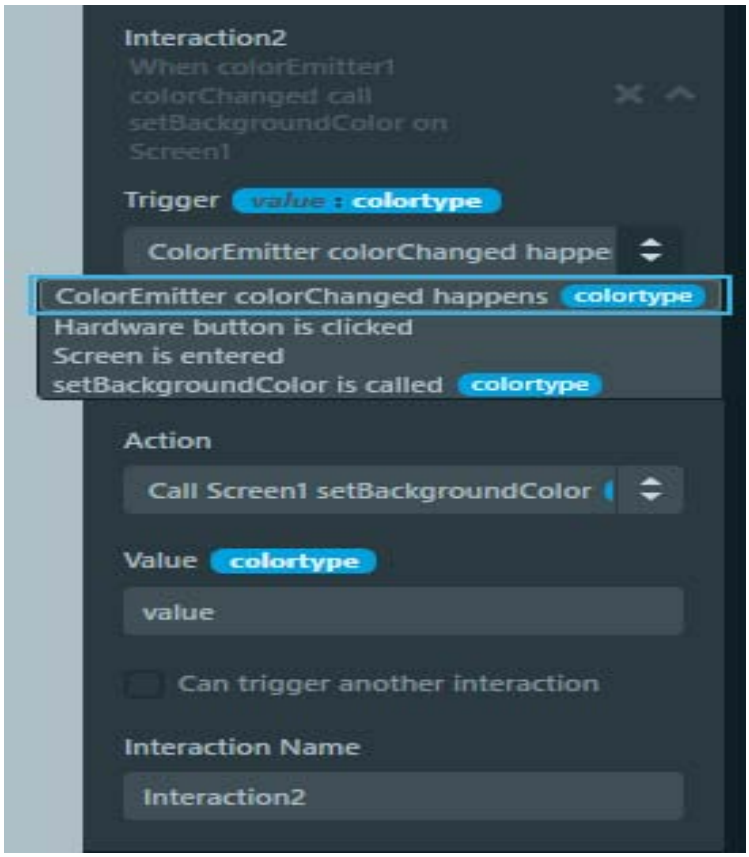
```
#include <touchgfx/Color.hpp>
```





그러나 "touchgfx : : Color"네임스페이스가 사용자 정의 컨테이너에 자동으로 포함되지 않기 때문에 작동 하지 않습니다. 따라서 이전과 마찬가지로 사용자 정의 컨테이너에 대한 헤더파일을 제공할 것입니다.

사용자 정의 컨테이너의 속성 탭으로 이동하여 "INCLUDES" 그룹에 다음을 입력합니다.

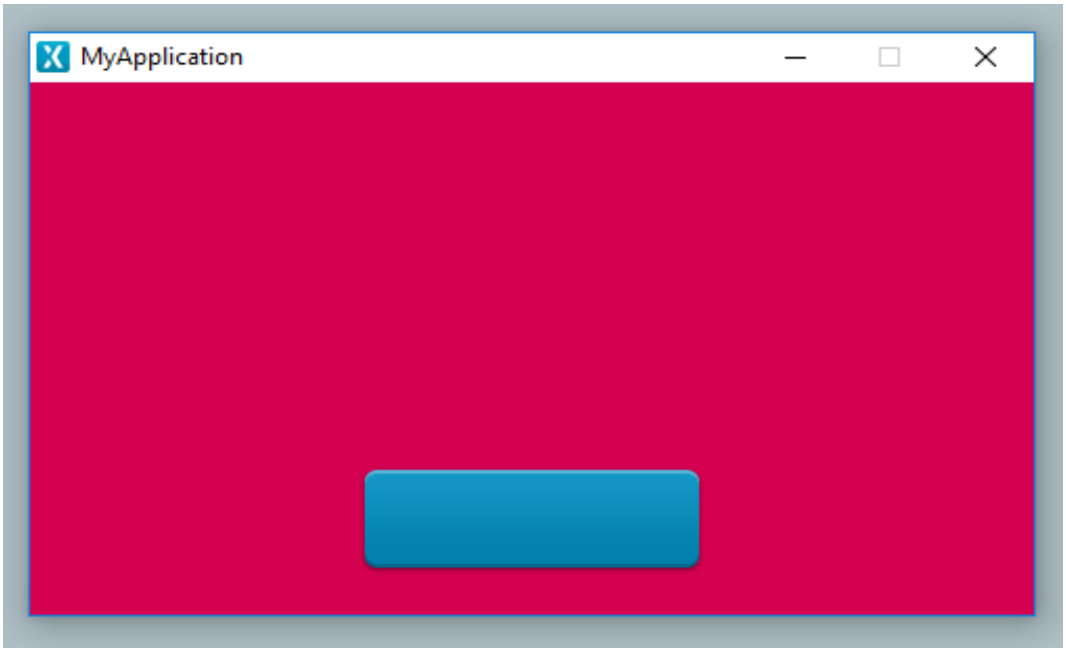


이전 버튼을 우리가 만든 새로운 "ColorEmitter" 사용자 정의 컨테이너로 교체하려고 합니다. Screen1을 선택하고 이때 버튼을 삭제합니다. 삭제시 이 버튼을 사용하던 상호작용에 도메인 오류가 발생하므로 ColorEmitter에 대해 새 상호작용을 생성합니다. 그러므로 해당 상호작용도 삭제하십시오.

이제 Screen1에 ColorEmitter의 인스턴스를 삽입한 다음 Screen1에 새 상호 작용을 만듭니다. 트리거의 경우 "ColorEmitter color Changed happens" 이라는 옵션이 표시되어야 합니다. 해당 항목을 선택하고 Action에 "Call Screen1 setBackgroundColor"를 사용합니다.

이제 colorChanged extract의 값을 입력해야 합니다. 상호작용은 사용자 정의 컨테이너에서 전달된 값을 사용합니다. 따라서 value 속성에 "value"를 입력하여 항상 "value"로 지정 하도록 합니다.

라. 시뮬레이터 실행 결과



이제 시뮬레이터를 실행하고 버튼을 다시 한 번 눌러봅니다. 동일한 동작이 나타나야 하며, 배경은 임의의 색상으로 바뀌어야 합니다.

이제는 단순히 화면에 모든 기능을 구현하는 대신 스크린과 일부 더 작고 재사용 가능한 구성 요소들, 즉 간단한 ColorEmitter 간의 통신을 성공적으로 만들었습니다.