Freescale Semiconductor

Getting started with the eGUI (a.k.a. D4D)

By: Technical Information Center

Introduction

The complimentary embedded graphical user interface (eGUI) allows single chip MCU systems to implement a graphical user interface and drive the latest generation of color graphics LCD panels with integrated display RAM and simple serial peripheral interface (SPI) or parallel bus interface. The eGUI supports conventional LCD panels and ColdFire LCD MPUs.

Contents

1. About this document

This document is intended to be a basic guide to get started with the eGUI. If you need a closer view about what this driver is, it is recommended to take a look into the eGUI Introduction document. Before starting, it is needed to download the library from the NXP eGUI website and it is recommended to take a look into the eGUI Reference Manual.

For this document, a demo included in the eGUI drivers will be run and then modified to toggle a LED in the TWR-K70F120M board from the eGUI interface displayed in the TWR-LCD-RGB board.

IMPORTANT: The eGUI is able to be run either in baremetal or over MQX.

2. Running the K70 demo on IAR

This demo is showing the eGUI capability in simulate standard forms and the keyboard and mouse handling. This demo needs theTWR-K70F120M, the TWR-SER and the TWR-LCD-RGB boards; also, it runs over MQX. The hardware and MQX configurations must be as follows:

TWR-SER card settings:

The card has to be setup to provide USB host capability: J10: 1-2; J16: 1-2.

TWR-K70F120M card settings:

Use default setting of card.

TWR-LCD-RGB:

The demo supports revision B and newer revisions of this card.

The MQX Settings:

The only two changes that must be done in the MQX PSP configuration are these:

```
user_config.h
            #define MQX_HAS_TIME_SLICE              1
```

```
small_ram_config.h
            #define MQX_ROM_VECTORS                 0
```

a. Compiling the TWR-K70F120M MQX libraries

Since the demo project is built with IAR, it is needed to compile the BSP, PSP, MFS and USB Host libraries for the TWR-K70F120M for IAR so it is needed to open and compile the projects located in the paths below.

```
C:\Freescale\Freescale_MQX_4_2\mqx\build\iar\bsp_twrk70f120m
C:\Freescale\Freescale_MQX_4_2\mqx\build\iar\psp_twrk70f120m
C:\Freescale\Freescale_MQX_4_2\mfs\build\iar\mfs_twrk70f120m
C:\Freescale\Freescale_MQX_4_2\usb\host\build\iar\usbh_twrk70f120m
```

b. Adding the SD card content

Before running the application in the MCU, it is needed to copy the files from the **Freescale_embedded_GUI_SW_Official_Demos\EGUI_Demo\SD_Card_Content** to the root directory in a micro SD card. This folder contains a text and an image files which will be previewed in the demo and two font files generated with the Freescale eGUI Converter Utility 3.0 that can be configured to be used in the demo.

c. Download the application to the MCU

Once everything is configured, it is time to open the demo in the path **Freescale_embedded_GUI_SW_Official_Demos\EGUI_Demo\TWR_K70\IAR\eGUI_Demo.eww**.

The file **eGUI_Demo\Sources\ScreenData\unicode_test.c** must be deleted from the project and the path **C:\Freescale\Freescale_MQX_4_2\shell\source\include** must be added before to compile by right-clicking over the project and selecting the menu **Options… > C/C++ Compiler > Preprocessor** and then add the path in the section **Additional include directories**.

The micro SD card must be inserted before to run the demo which includes an “Open” button which will pop up a window to explore the micro SD card content and look for files that can be previewed. There is also a “System” button which opens a pop up window to configure stuff like language, colors and fonts.

3. Modifying the eGUI demo

This demo will be modified to create a new window with a button capable to toggle a LED located in the board. This can be done by several ways but for this time a new screen will be created with a big toggle button. The information presented in this chapter is mostly taken from the eGUI Reference Manual.

a. Creating a new screen

The D4D_SCREEN is the basic organizational structure of the eGUI. The whole project structure is divided into individual screens and its objects, therefore screens are used to keep organization of the whole project by the appearance as well as the program structure. It is recommended to create your own source code file for each individual screen. It is possible to go through the screens by system forward and back (activate and escape screen).

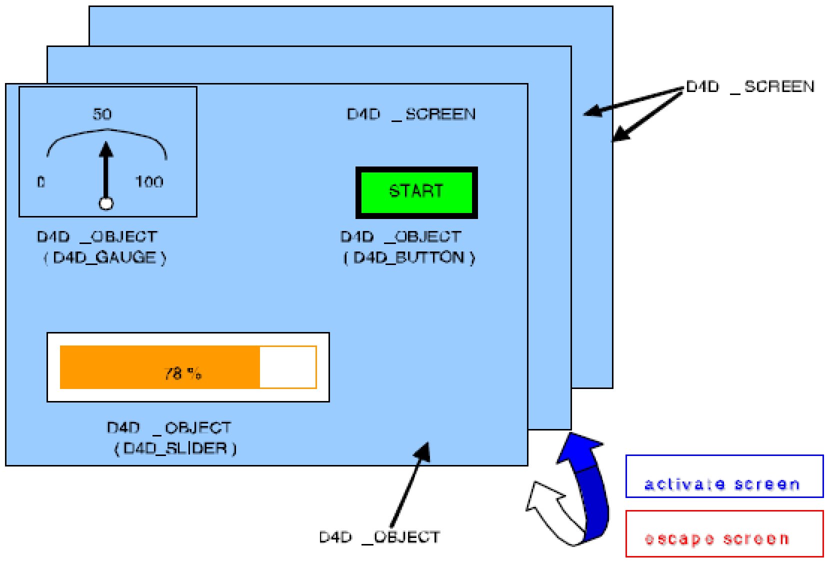

50
0
100
D4D _SCREEN
START
D4D _OBJECT (D4D_GAUGE)
D4D _OBJECT (D4D_BUTTON)
78 %
D4D _OBJECT (D4D_SLIDER)
D4D _SCREEN
D4D _OBJECT
activate screen
escape screen

Figure 1. eGUI structured objects and screens.

The D4D_SCREEN object can be used as a window, have a smaller size, be placed anywhere, and also have a header (title bar) with icon, title, and exit button. Such a screen looks like a window and has many uses (help menu, warning, error, ask, messages, and so on).

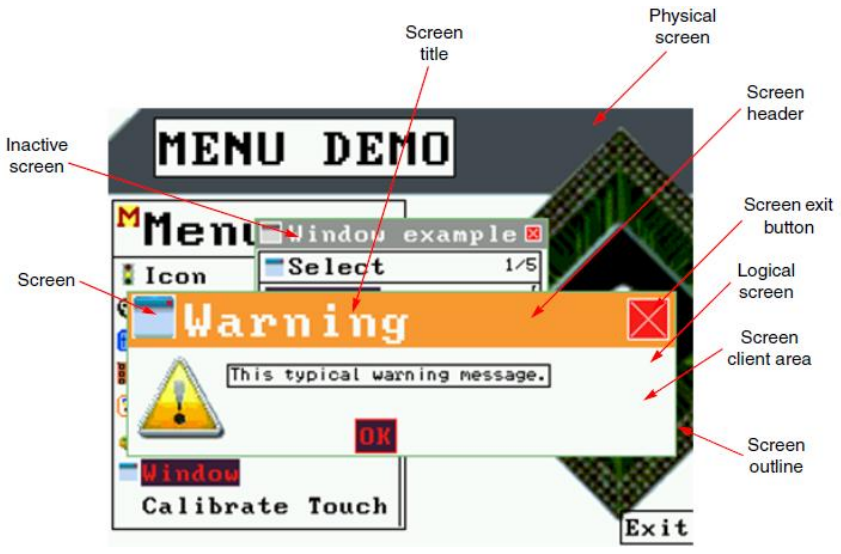


Figure 2. Screen with non-default settings.

i. Adding a new source file for the new screen

Taking the advices from the reference manual, a new source file is created in the path **Freescale_embedded_GUI_SW_Official_Demos\EGUI_Demo\common_sources\d4d_screen_LEDtoggle.c**. This file is added to the **Screens** group by right clicking over **Workspace > eGUI_Demo\Sources\Screens** and then **Add > Add files…** and selecting the new source file.

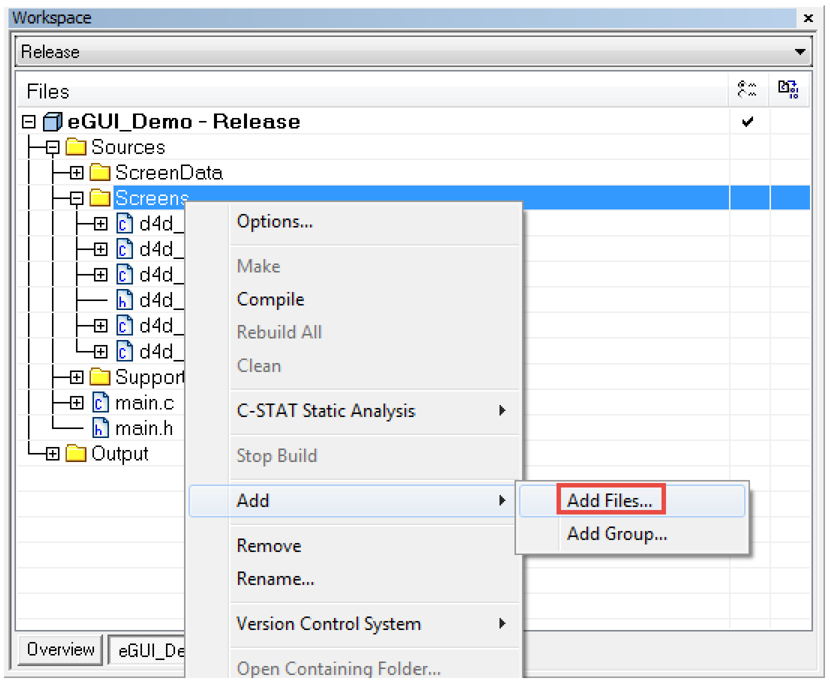


Figure 3. Add the new screen's source file to the project.

ii. Adding the screen header files

Once the screen’s source file has been added, it is needed to define the header files that are mainly used:

```
/*************************************************************************
*       Includes header files
*************************************************************************/
#include "d4d.h"
#include "fonts.h"
#include "strings.h"
```

The **d4d.h** header includes the d4d libraries, i.e. it includes all the needed definitions to use the d4d objects. The **fonts.h** header contains the fonts data generated by the eGUI converter utility. The **strings.h** header includes the strings definitions, this is fundamental to support multi language applications.

iii. Adding some macro definitions

Sometimes it is useful to define macros with information that is usually modified after changing some hardware (like the screen’s size) or while designing interfaces (like the objects’ size). This is a good reason to create macro definitions for this kind of elements and keep the source code organized.

```
/*************************************************************************
*       D4D widgets (graphic objects) declarations
*************************************************************************/

#define SET_SCREEN_SIZE_X       432
#define SET_SCREEN_SIZE_Y       220
```

Since the TWR-LCD-RGB resolution (size) is 480 x 272, the screen size 432 x 220 means that the new screen will look like a non-full-screen window.

iv. Defining screen attributes

As it was said, a screen has several configurations such the existence of a close (exit) button, a title bar, among others. This information is usually defined through macro definitions as it is shown below. In this case, the new window has an outline, a title bar, a filled background, a normal mouse cursor and an exit button.

```
/**************************************************************************
*       D4D widgets (graphic objects) declarations
**************************************************************************/

#define D4D_SCRSET_F              (D4D_SCR_F_OUTLINE | D4D_SCR_F_TITLEBAR |
D4D_SCR_F_BCKG | D4D_SCR_FINT_MOUSE_NORMAL | D4D_SCR_F_EXIT)
```

For more information go to the screen behavior flags section in the eGUI Reference Manual.

v. Declaring the screen

Once the screen's properties have been defined, it is time to declare a new screen. It is done through the **D4D_DECLARE_SCREEN_BEGIN()** and **D4D_DECLARE_SCREEN_END()** sentences. I suggest to expand these sentences by right clicking over them and then select the option **Go to definition of ''**. This will show that in terms of C programming, these macros are declaring a new structure with the screen configurations.

```
/**************************************************************************
*       D4D screen declarations
**************************************************************************/
D4D_DECLARE_SCREEN_BEGIN(screen_LEDtoggle,   ScreenToggleLED_,   20   ,20,
(SET_SCREEN_SIZE_X  +  2),  (SET_SCREEN_SIZE_Y  +  20),  D4D_STR_TOGGLE_LED,
FONT_SYSTEM, NULL, D4D_SCRSET_F, NULL)


D4D_DECLARE_SCREEN_END()
```

It is seen that, among other parameters, it is needed to define a screen's name and a prefix name for some event handlers that must be defined on each screen, this will be implemented later.

Also, the parameter **text** asks for a title bar text. This means that a new string for this screen must be defined. By the moment, it is selected the name **D4D_STR_TOGGLE_LED** for this string, it will be defined in the next step.

vi. Defining a new string

A new string must be defined, as it was defined in the last step, it must be called **D4D_STR_TOGGLE_LED**. Since this demo has support for English, Czech and Chinese, it is needed to add these three strings for the same definition.

The first step to do this is to define a new name for this string and an ID in the **eGUI_Demo\Sources\ScreenData\strings.h** file, in this case the ID 40 was added.

```
#define D4D_STR_TOGGLE_LED                    D4D_DEFSTR("@40")
```

Then, in the **eGUI_Demo\Sources\ScreenData\strings.c** file, a new element must be added in the 40th position (because of the defined ID) of the string's array on each of the three already defined languages. For example, in the case of the English language, the new element must look as follows:

```
D4D_DEFSTR("Toggle LED")                // D4D_STR_TOGGLE_LED
```

A table with the supported languages is declared in this file so if it is needed to add a new language this must be defined here.

vii. Adding the screen interfaces

Finally, it is needed to implement the mandatory event handlers (interfaces) for the screen. These handlers include the OnInit, OnActivate, OnMain, OnDeactivate and OnObjectMessage events. By this moment, these handlers will be empty.

```
/**************************************************************************
*       D4D standard screen functions definitions
*************************************************************************/

// One time called screen function in screen initialization process
static void ScreenToggleLED_OnInit()
{

}

// Screen on Activate function called with each screen activation
static void ScreenToggleLED_OnActivate()
{

}

// Screen "Main" function called periodically in each D4D_poll runs
static void ScreenToggleLED_OnMain()
{

}

// Screen on DeActivate function called with each screen deactivation
static void ScreenToggleLED_OnDeactivate()
{

}

// Screen on message function called with each internal massage for this
screen
static Byte ScreenToggleLED_OnObjectMsg(D4D_MESSAGE* pMsg)
{
  D4D_UNUSED(pMsg);
  return 0;
}
```

b. Opening the new screen

At this point, the screen has been created but it hasn't been opened yet. The event of "open" a screen is called "activate" and the event of "close" it is called "escape".

For the sake of simplicity, the way to activate this new screen will be by replacing the "system" button in the main screen for this new one. This is done by replacing the OnClick event for the "system" button in the "main" screen (i.e. in the main's screen source file

Freescale_embedded_GUI_SW_Official_Demos\EGUI_Demo\common_sources\d4d_screen_main.c).

The first step to activate a screen is to declare as extern the target screen somewhere in the source screen. The parameter of the macro **D4D_EXTERN_SCREEN** makes reference to the name which was given when the screen was declared.

```
// Reference to the target screen
D4D_EXTERN_SCREEN(screen_LEDtoggle);
```

Also, it results convenient to change the button's text from "system" to "Toggle LED". The string **D4D_STR_TOGGLE_LED** defined for the screen's title bar could work. Change the text argument for the **sMain_btnSettings** button declaration from **D4D_STR_SYSTEM** to **D4D_STR_TOGGLE_LED**.

```
D4D_DECLARE_TXT_RBUTTON(sMain_btnSettings,D4D_STR_TOGGLE_LED,
BTN_POS_X(1), BTN_POS_Y, BTN_SIZE_X, BTN_SIZE_Y, MAIN_RADIUS, FONT_SYSTEM,
SFL_BtnSettingsOnClick)
```

The OnClick event is called when the "system" button is pressed, originally it activates the settings screen, so we need to comment out that function and activate the LED toggle screen instead.

```
// On Click Event of "SETTINGS" button
static void SFL_BtnSettingsOnClick(D4D_OBJECT* pThis)
{
  //D4D_ActivateScreen(&screen_settings, D4D_FALSE);

  // Activate the Toggle LED screen.
  D4D_ActivateScreen(&screen_LEDtoggle, D4D_FALSE);
}
```

Since the string is larger, it is recommended to change the BTN_SIZE_X from 100 to 107.

```
#define BTN_SIZE_X              107
```

c. Creating a new button

At this point, a new screen is opened after pressing the “Toggle LED” button. However, it is empty. The natural way to toggle a LED is with a button so a button will be created for this purpose.

The first step to define some macros with the button’s properties like the size, the position, the corner radius and the button flags. The possible button flags are listed in the eGUI Reference Manual.

```
/**************************************************************************
*       D4D widgets (graphic objects) declarations
*************************************************************************/

#define SET_BTN_POS_X           50
#define SET_BTN_POS_Y           50

#define SET_BTN_SIZE_X          (SET_SCREEN_SIZE_X - 2 * SET_BTN_POS_X)
#define SET_BTN_SIZE_Y          (SET_SCREEN_SIZE_Y - 2 * SET_BTN_POS_Y)

#define SET_RADIUS              6

#define SET_BUTTON_FLAGS       (D4D_OBJECT_F_VISIBLE | D4D_OBJECT_F_ENABLED
|       D4D_OBJECT_F_TABSTOP       |       D4D_OBJECT_F_TOUCHENABLE       |
D4D_OBJECT_F_MOUSE_NORMAL         |        D4D_OBJECT_F_FASTTOUCH        |
D4D_OBJECT_F_BEVEL_RAISED | D4D_BTN_F_3D)
```

Then, the button is declared through the macro **D4D_DECLARE_RBUTTON**.

```
/**************************************************************************
*       Local variables declarations
************************************************************************/

// Declare Toggle LED button
D4D_DECLARE_RBUTTON(sLED_btnToggleLED,  D4D_STR_TOGGLE_LED,  SET_BTN_POS_X,
SET_BTN_POS_Y,      SET_BTN_SIZE_X,      SET_BTN_SIZE_Y,       SET_RADIUS,
SET_BUTTON_FLAGS,    NULL,    NULL,    NULL,    FONT_SYSTEM_BIG,    NULL,
BtnToggleLEDOnClick, NULL)
```

The parameter OnClick is a callback which will be executed each time the button is pressed. This callback must be declared and defined by the application.

```
/***************************************************************************
*       Local functions declarations
***************************************************************************/

static void BtnToggleLEDOnClick(D4D_OBJECT* pThis);


/***************************************************************************
*       D4D widgets static events functions definitions
***************************************************************************/

// The Toggle LED button action
static void BtnToggleLEDOnClick(D4D_OBJECT* pThis)
{

}
```

Finally, the button must be added as part of the toggle LED screen. If more widgets want to be added to this screen, these must be added in this section.

```
/***************************************************************************
*       D4D screen declarations
***************************************************************************/

D4D_DECLARE_SCREEN_BEGIN(screen_LEDtoggle,   ScreenToggleLED_,   20   ,20,
(SET_SCREEN_SIZE_X  +  2),  (SET_SCREEN_SIZE_Y  +  20),  D4D_STR_TOGGLE_LED,
FONT_SYSTEM, NULL, D4D_SCRSET_F, NULL)
  D4D_DECLARE_SCREEN_OBJECT(sLED_btnToggleLED)
D4D_DECLARE_SCREEN_END()
```

d. Toggling the LED

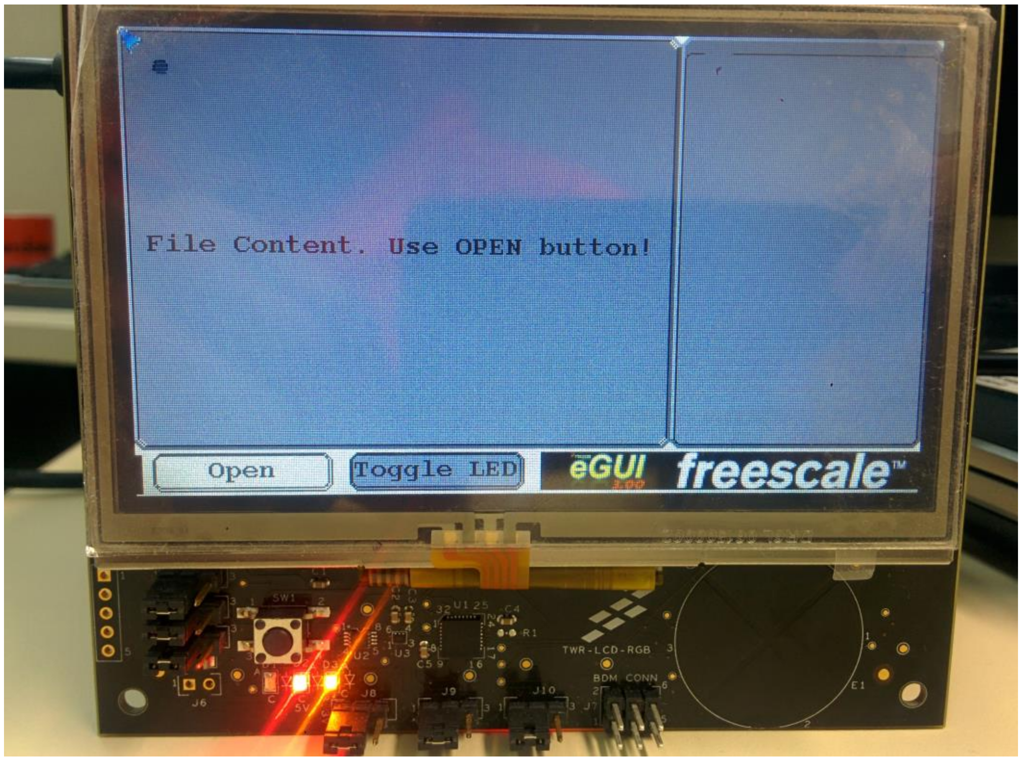
The only step remaining is to toggle the LED into the button's OnClick callback. This is done through the MQX API for the GPIO. The LED must be initialized before to be used, the best moment to do this is in the screen's OnInit handler.

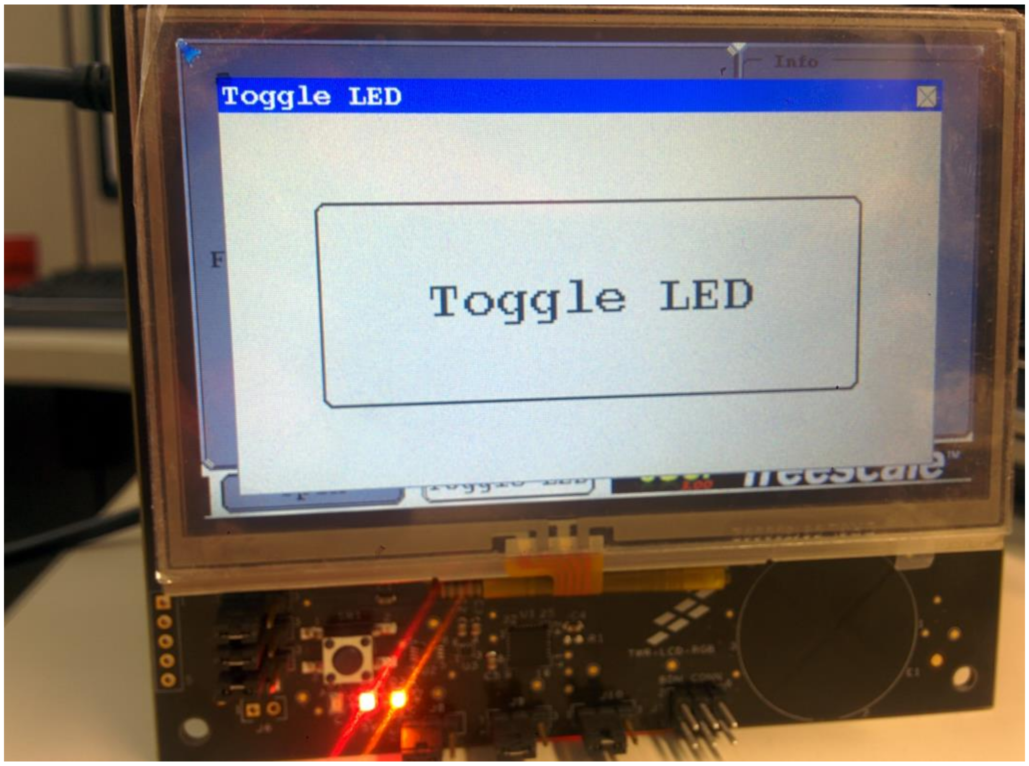
```
/**************************************************************************
*       D4D standard screen functions definitions
**************************************************************************/

// One time called screen function in screen initialization process
static void ScreenToggleLED_OnInit()
{
 /* initialize lwgpio handle (led1) for BSP_LED1 pin
  * (defined in mqx/source/bsp/<bsp_name>/<bsp_name>.h file)
  */
  if       (!lwgpio_init(&ledC,       BSP_LED2,       LWGPIO_DIR_OUTPUT,
LWGPIO_VALUE_NOCHANGE))
  {
    //Error.
  }

  /* swich pin functionality (MUX) to GPIO mode */
  lwgpio_set_functionality(&ledC, BSP_LED1_MUX_GPIO);
}
```

Then, each time the button is pressed, the GPIO's toggle function must be called.

```
/**************************************************************************
*       D4D widgets static events functions definitions
**************************************************************************/
LWGPIO_STRUCT ledC;

// The Toggle LED button action
static void BtnToggleLEDOnClick(D4D_OBJECT* pThis)
{
  lwgpio_toggle_value(&ledC); /* toggle pin value */
}
```

4. Results


File Content. Use OPEN button!
Open
Toggle LED
eGUI
freescale


Toggle LED
Info
Toggle LED

5. Conclusion

This document has demonstrated how easy is to get introduced to the eGUI library. Only two elements (screen and button) were used in this document, however, the eGUI reference manual explains in detail how the other widgets are used by following the same methodology.