µEZ GUI Start Here Guide

UEZGUI-4357-70WVN

Introduction

At Future Designs, our goal is to make it easy for our customers to get their projects up and running as quickly as possible. In this guide, you will develop a simple graphical user interface (GUI) on the µEZ platform to demonstrate how to use some of the core features of emWin and µEZ. Using this "Hello World" type of walkthrough as a starting point, we hope to shorten the learning curve for GUI development. Aside from this document, there are many additional resources available at www.TeamFDI.com. If you ever need more help, contact us and we will be happy to assist you.

Hardware Used in This Guide: (Included in Kit)

- UEZGUI-4357-70WVN-BA
- SEGGER J-Link Lite Cortex-M Probe
- USB Type A to USB Type Mini B Cable (2x)
- Universal AC to 5V USB plug Power Supply Unit
- Micro SD Card

Software Used in This Guide:
(Installation and usage instructions are provided within the guide)

- SEGGER J-Link Software
- µEZ Source w/ Project Maker (v2.12 or later)
- One of the following IDEs:
  - Rowley CrossWorks v4.8
  - IAR Embedded Workbench v9.30.1

Files Used in This Guide:
(Installation and usage instructions are provided within the guide)

- uEZ Project Maker (Located in the µEZ source download)
- uEZ Project Maker Project (Created using the Project Maker)

Contents

Important Legal Information

1. Hardware Verification

The µEZ GUI kit comes with a pre-installed 4GB or larger SD card that contains files required for the demo application and slideshow to run. It also contains user manuals, schematics, and documentation for the product including this guide. FDI recommends that you visit the documentation tab of the product page of your µEZ GUI to get the latest documentation.

Figure 1: *UEZGUI-4357-70WVN*

Power is supplied via the USB power adapter and cable provided in the kit.

1. Insert the included SD Card into the uEZGUI.
2. Connect the USB cable to the mini USB Type B connector on the µEZ GUI.
3. Connect the other end of the USB cable to the provided universal AC power supply's 5V USB power output.

NOTE: The µEZ GUI must be powered with the included universal AC to 5V USB plug power supply or through the alternate power and com port on the side. Do not try to power the µEZ GUI from a computer's USB port. The standard PC USB port does not provide the necessary current to power the µEZ GUI. The JTAG port and J-Link probe do not power the µEZ GUI.

The following screen will appear once power has been connected to the device:

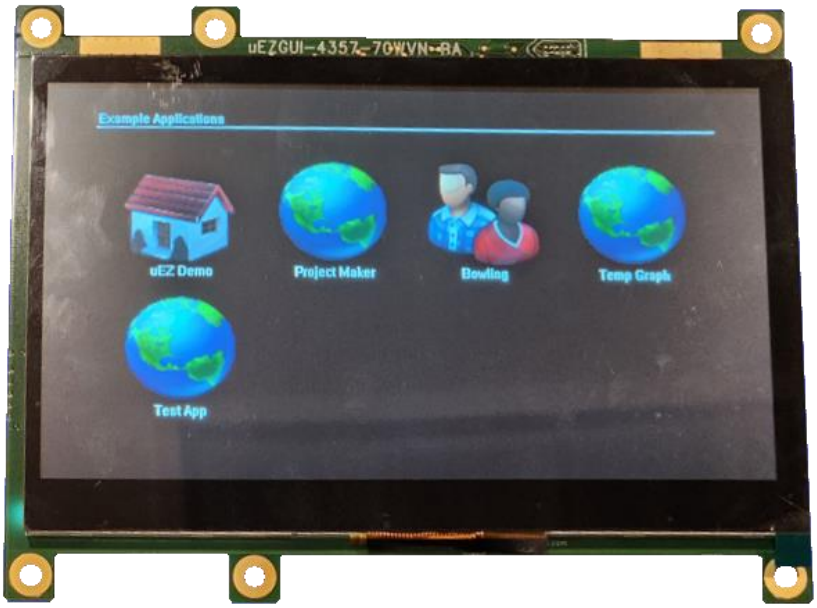


Figure 2: *Out-of-Box (OOB) Demo*

You can now explore the out-of-box demos.

2. Software Installation

A. IDE Installation

All µEZ GUI projects require a development environment to compile and debug the projects. µEZ currently officially supports IAR Embedded Workbench v9.30.1 and Rowley CrossWorks 4.8 on the LPC4357. This guide uses IAR Embedded Workbench for ARM version 9.30.1 and Rowley Crossworks for ARM Version 4.8. Download your preferred IDE at the appropriate link provided below and install according to the instructions provided on their respective websites.

- IAR Embedded Workbench 9.30.1 30-day free trial: (select Evaluation License option on install)
  Windows: https://tinyurl.com/3m93kyfh (.7z)
  - IAR 30-day evaluation license manual activation instructions: http://tinyurl.com/zyctkha
- Rowley CrossWorks for ARM version 4.8 30-day free trial:
  Windows: https://tinyurl.com/3yps2nff
  Mac: https://tinyurl.com/yrrkbrxs
  Linux x64: https://tinyurl.com/5n283eyu
  - Activating Crossworks 30-day evaluation license instructions: http://tinyurl.com/hlsn9qf

B. J-Link Installation

The µEZ GUI uses a mini-J-Link debugger probe. The SEGGER version 6.86b driver (or later) must be installed. Download and install the software from the link below:

- SEGGER J-Link Software 6.86b:
  Windows (x86): https://tinyurl.com/yn7dxxd3
  Mac (x64): https://tinyurl.com/ywpcxp62

Linux ARM (x64): https://tinyurl.com/y3wzrwcm

C. µEZ Installation

µEZ is a middleware API library built on FreeRTOS and emWin. It is created, managed, and regularly updated by FDI. Download the latest µEZ release at the link provided below:

- µEZ (v2.12): https://sourceforge.net/projects/uez/

Extract the µEZ .7z, or .zip file, to the desired location. This folder contains the µEZ source library, helpful tools, documentation, project files referenced in this guide, and the µEZ project maker. The library files will be compiled in subsequent steps.

Note: This walkthrough uses "C:/Users/(user)/Desktop/uEZ-Workspace" as the destination

3. Connecting the J-Link to the µEZ GUI for programming

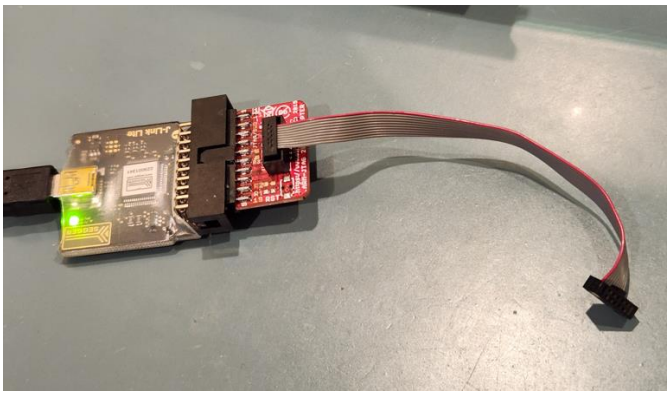


Figure 3: *J-Link Lite Hardware*

1. Connect the J-Link Lite to a PC with an included USB cable.
2. Connect the J-Link Lite to the 20-to-10-pin JTAG adapter and connect to the JTAG connector (J1) on the bottom side of the µEZ GUI. The J-Link 10-pin ribbon cable for the µEZ GUI is supplied in the kit.

> **Note**: Care must be taken when connecting to J1, as J1 is not keyed, thus allowing the user to mistakenly connect to J1 backwards

3. The red part of the ribbon cable indicates Pin 1
4. Locate Pin 1 on J1 on the uEZGUI

Figure 4: *JTAG Pin 1 location*

5. Connect the 10-pin cable to J1 as shown:

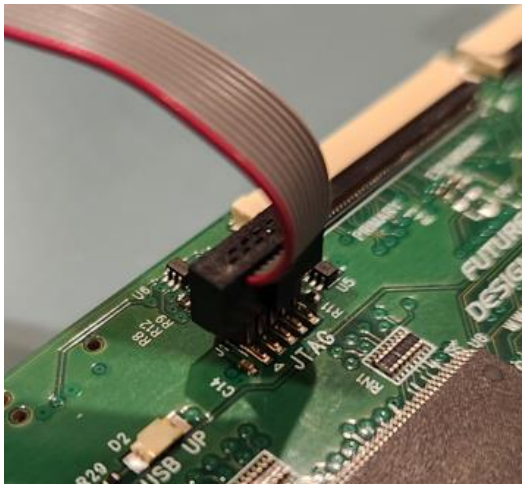


Figure 5: *Jlink connected to JTAG*

6. If not already done, connect the µEZ GUI to the power supply with the second included USB cable.

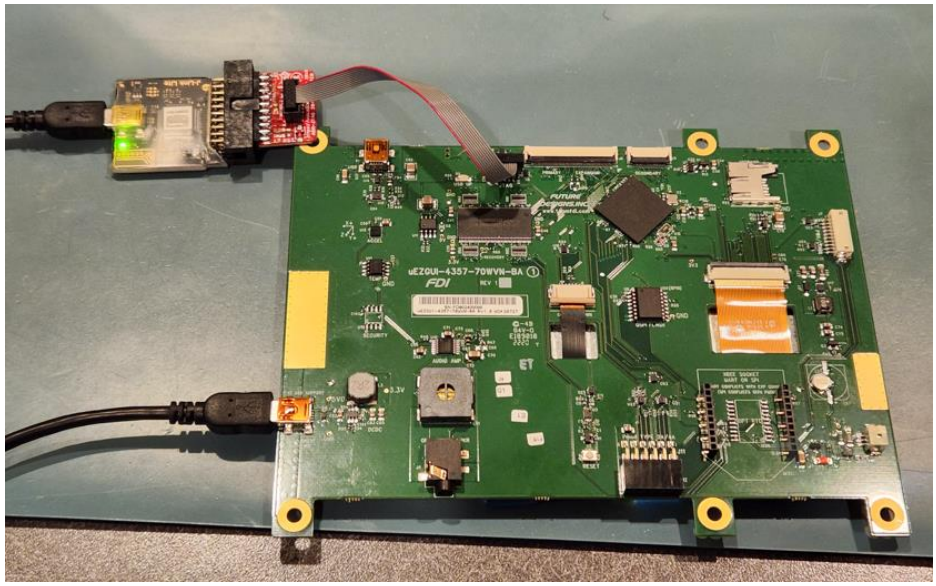


Figure 6: *J-Link and Power Connected to µEZ GUI*

4. Using the uEZ Project Maker to Create a Basic Application

The uEZ Project Maker is standalone software that comes packaged with uEZ downloads. The Project Maker gets you started by creating a basic demo project from which you can build your own fully customizable application for the following uEZGUIs:

- uEZGUI-1788-43WQR
- uEZGUI-1788-56VI
- uEZGUI-1788-70WVT
- uEZGUI-1788-70WVM

- uEZGUI-4088-43WQN
- uEZGUI-4357-50WVN
- uEZGUI-4357-70WVN

This document covers the 4357-70WVN.

A) Running uEZ Project Maker

1. Locate **uEZProjectMaker.exe** in your uEZ-Workspace directory

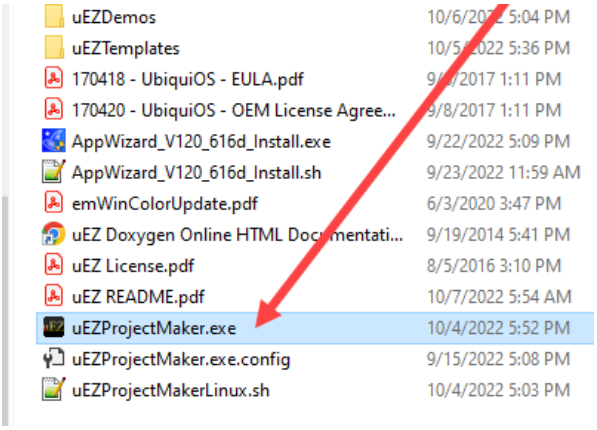


Figure 7: *uEZ Project Maker in our uEZ-Workspace*

2. When you start it, you will see a generic welcome screen.
3. Click <Next>, and select "FreeRTOS".
4. Click <Next> and select "NXP LPC4357" from the list of processors

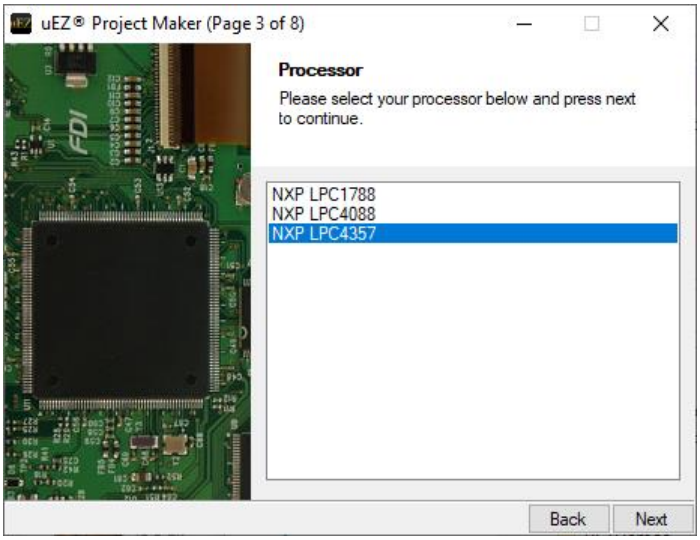


Figure 8: *Selecting the correct processor*

5. Click <Next> and select "uEZGUI-4357-70WVN" from the list of platforms
6. Click <Next> and select the Development Suite you will use from the list of development suites.

NOTE: Do not select "Test Project" for this walkthrough

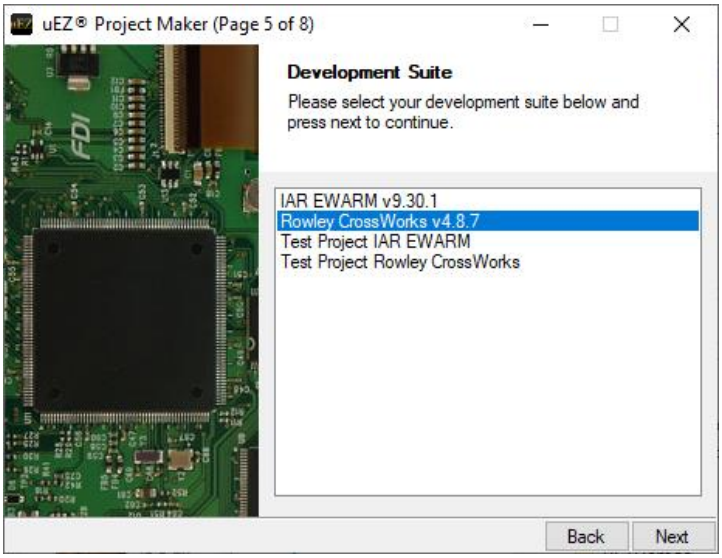


Figure 9: *Selecting a development suite*

7. Click <Next>
8. You will need to specify your project's name and path
    a. Name your project "SampleProject"
    b. Set the Project Path to "**C:\Users\(user)\Desktop\uEZ-Workspace**"
9. Click <Next>
10. You will see a summary window. Verify all settings are correct for your project, then click <Create>
11. After a few seconds pass, a "Finished" window will appear. Click <Exit>
12. Your SampleProject will now show up in your uEZ-Workspace

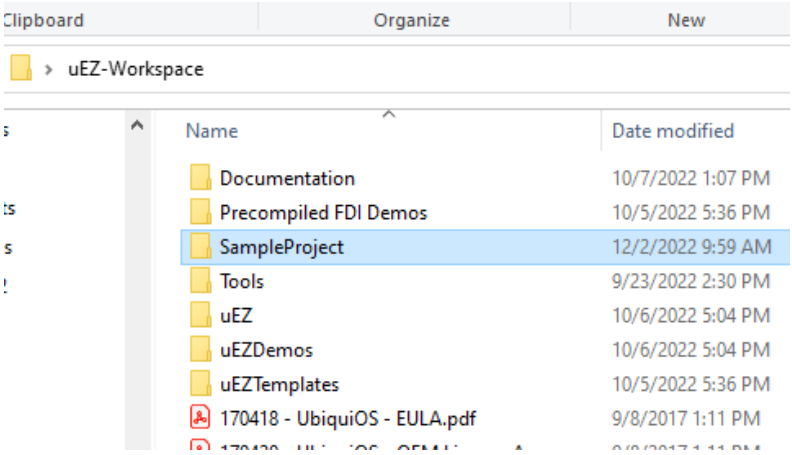


Figure 10: *SampleProject created in the uEZ Workspace*

B) Building and Running SampleProject

NOTE: The following example demonstrates using both Crossworks AND IAR, but you must use the same development suite when building both the uEZ Library and the Sample Project

1. In order to build and run the Sample Project, you will first need to build the uEZ Library
2. Navigate to the project directory for the IDE you chose in the Project Maker.
3. For this walkthrough, it will be either:
   a. "C:\Users**(user)**\Desktop\uEZ-Workspace\uEZ\Build\Generic\NXP\LPC4357\FreeRTOS\CrossWorks"
      or
      "C:\Users**(user)**\Desktop\uEZ-Workspace\uEZ\Build\Generic\NXP\LPC4357\FreeRTOS\IAR"

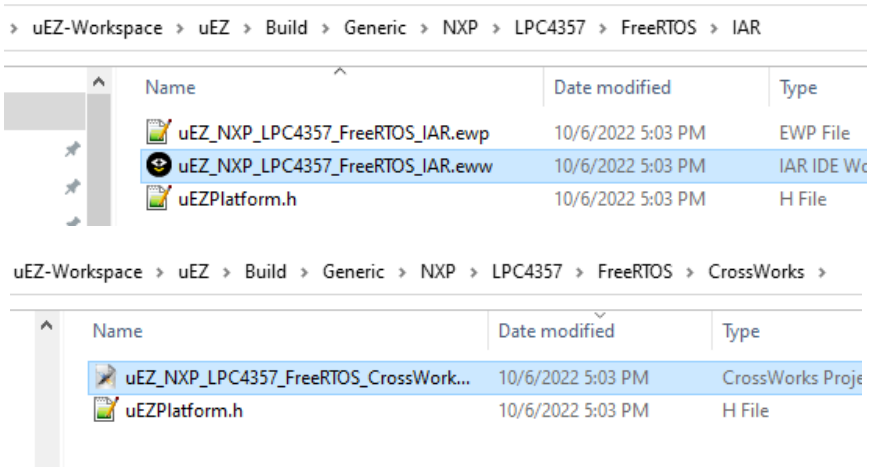
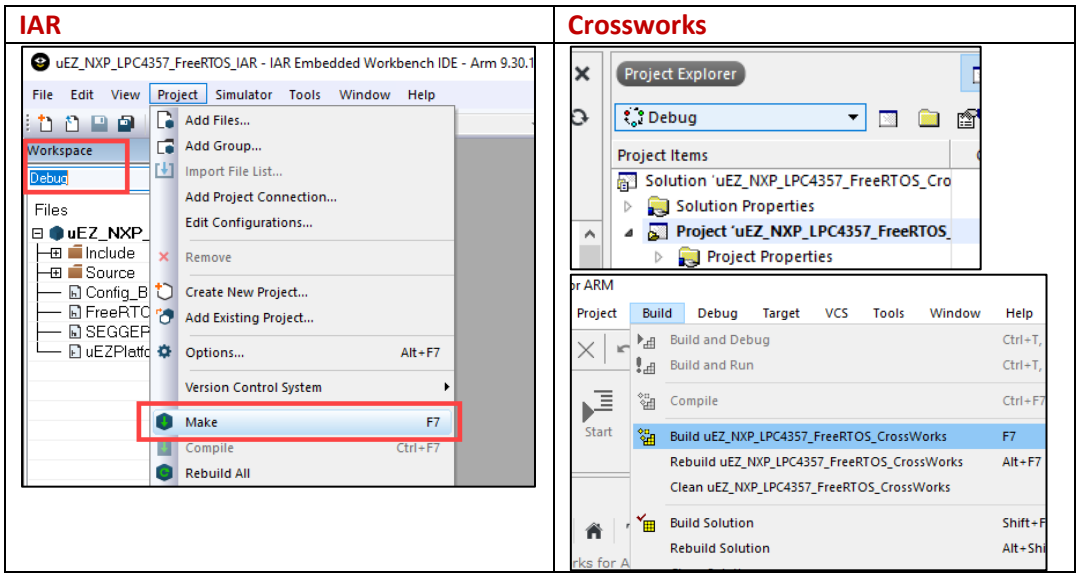


Figure 11: *uEZ Library Project files*

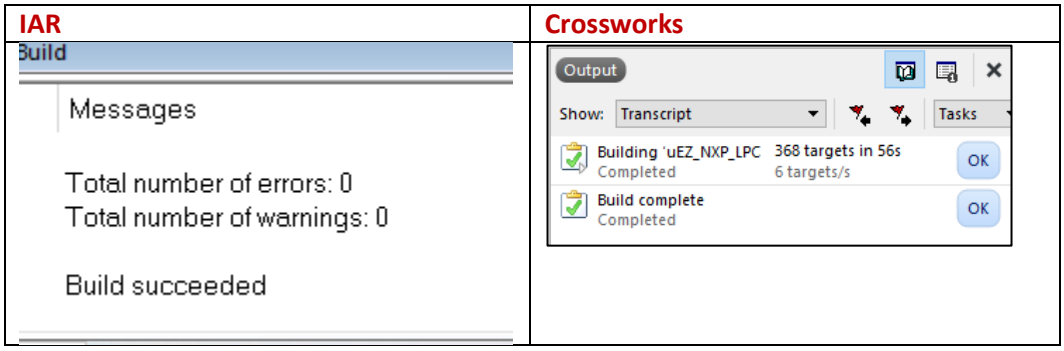
4. Open the relevant Project file for the IDE you are going to use
   "uEZ_NXP_LPC4357_FreeRTOS_CrossWorks.hzp" for Crossworks
   or
   "uEZ_NXP_LPC4357_FreeRTOS_CrossWorks.eww" for IAR
5. After the project opens:
   a. For IAR: Select the "Debug" Build from the workspace menu if not already selected.
   b. For Crossworks: Select the "Project Explorer" sub-window if not already selected.
6. Building the project:
   a. For IAR: Select "Project > Make" or press <F7> to build the µEZ library project.
   b. For Crossworks: Select "Build->Build (project name)" or press <F7> to build the µEZ library project.

Figure 12: *Building the µEZ library project*



7. Once building completes, verify there are 0 errors

Figure 13: *Build Finished*



8. Close the IDE.

9. Navigate to:
   "C:\Users**(user)**\Desktop\uEZ-Workspace\SampleProject\Build\uEZGUI_4357_70WVN\Crossworks"
   or
   "C:\Users**(user)**\Desktop\uEZ-Workspace\SampleProject\Build\uEZGUI_4357_70WVN\IAR"

10. Open the project file:
    "SampleProject.hzp"
    or
    SampleProject.eww"

11. If using IAR…
    a. Right click the main project name in the workspace and select "Options"

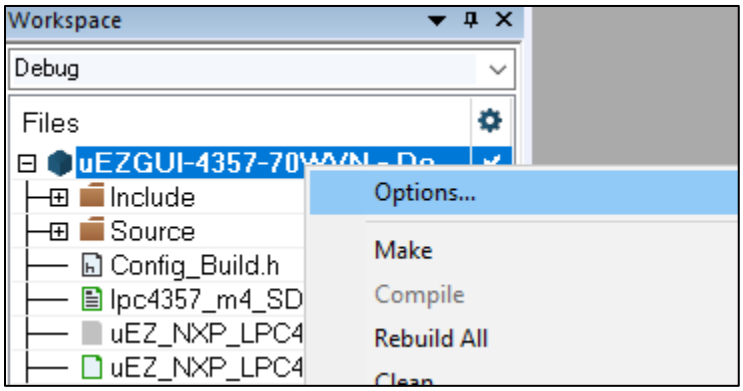


Figure 14: *Debug Settings*

b. Select “Debugger” and verify that the “Driver” is set to “J-Link/J-Trace”.

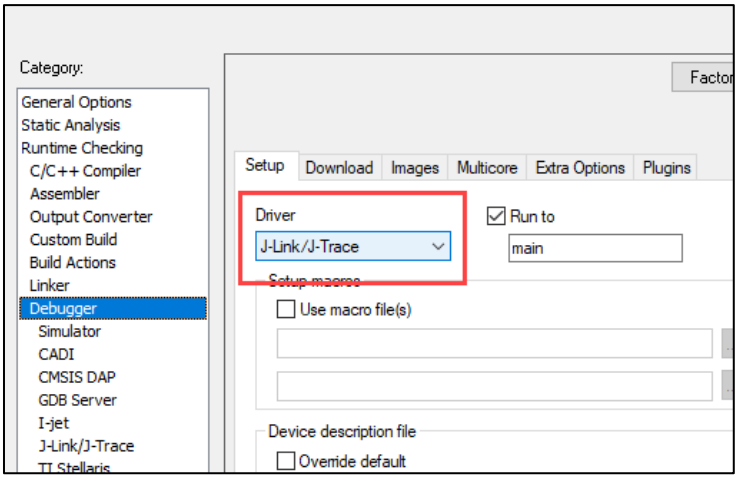


Figure 15: *Debug Settings*

c. Click the <OK> button to exit the dialog.
d. Press <F7>, or select “Project > Make” from the menu at the top
e. It should build with no errors

f. Press <Ctrl> + <D> to download and debug, or click the green circle with a white arrow on the toolbar
g. Proceed to **step 13**

12. If using Crossworks…
    h. Click “Target > Connect > SEGGER J-Link” from the menu at the top. This activates the J-Link.

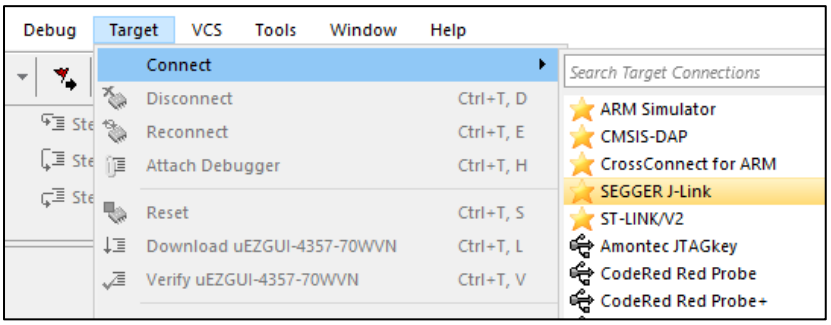


Figure 16: *Debug Settings*

    i. Press <F7>, or select “Build -> Build uEZGUI-4357-70WVN” from the menu at the top
    j. It should build with 0 errors

13. Press <F5> or select “Debug > Go” from the menu at the top.
14. If the debugger has halted on a breakpoint, press <F5> to make the project run.
15. The uEZGUI startup tone will play and you will see the following screen:

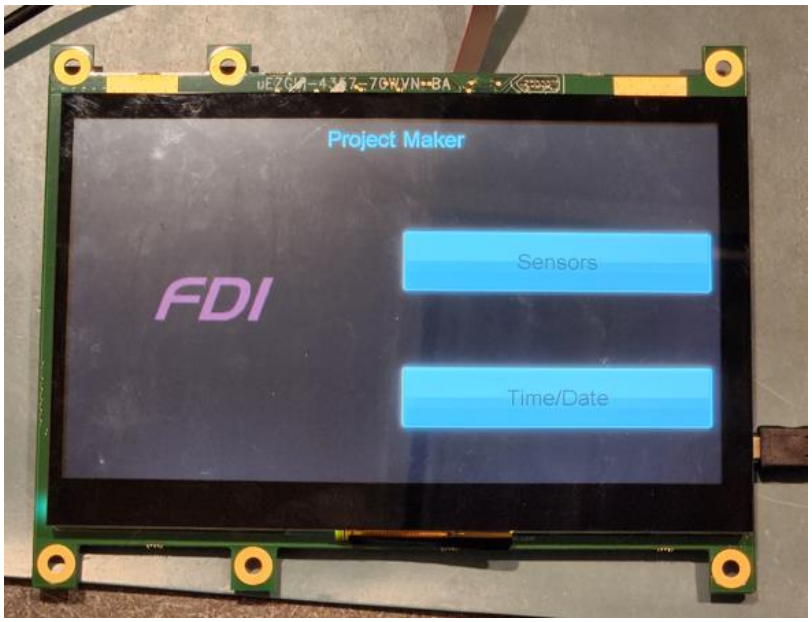


Figure 17: *Default Project Maker Application*

16. Click on the buttons to see a simple example of the accelerometer and temperature sensors, and a demonstration of the uEZGUI RTC (Real-Time Clock).
    a. Note: the RTC will be incorrect when the board is newly programmed or reprogrammed
17. All of the source code used to provide functionality to the project is within the “Source\App” directory. The “Source\App\emWin” directory contains the code that uses emWin to create and manage windows and other graphic designs for the GUI.

5. Developing a Simple GUI Application with emWin

A. emWin Introduction

emWin (segger.com/emwin.html) is a software library from SEGGER which provides an efficient GUI library that is processor and LCD controller independent. emWin enables you to easily add graphics to an application and is included with µEZ. Some of the features of emWin include:

- Basic drawing functions such as drawing lines, squares, circles and polygons.
- More complex functions such as managing windows, button widgets, list-view widgets, edit widgets, etc.
- Displaying images.
- Support for multiple displays.
- Support for multiple layers and transparency settings.
- Control of GUI by mouse and touch screen.
- Rapid development, even without targeted hardware, due to support for simulating the GUI in Microsoft Visual Studio.

NXP® Semiconductors has a license with SEGGER that allows any device using an NXP LPC microcontroller to use the full emWin library without requiring an additional license. Since the UEZGUI-4357-70WVN-BA utilizes the LPC4357, the emWin license applies. See here for more information: http://tinyurl.com/gueckmp, NXP emWin introduction video: http://tinyurl.com/jrata57

This section of the guide will take a simple screen and develop it into a GUI to demonstrate a few of the core features of the µEZ and emWin libraries.

B. Adding a New Button to the Home Screen

To demonstrate the process for adding widgets to a window, this section will walk you through the steps to create a button on the home screen which, once pressed, will open to a new window. To begin, open the HomeScreen.c file under App/emWin.

The HomeScreen.c file contains the information that is used to create a window and any widgets (buttons, etc) that will load with the window. The beginning of the file contains the #includes and a list of #defines that set the ID, size, and position of each object within the window including the window itself.

The next section of code contains prototypes for button callback functions.

In the next section, “Local Data”, there are two array variables which define the window and widgets and associate callback functions to them. The first array, _iHomeScreenDialog[ ], is an emWin array that holds the ID, position, and size for each widget. The second array, HomeScreenMapping[ ] is a µEZ array which holds their text label, color, font, and callback functions.

NOTE: A callback function is a function which is called when an associated event occurs. For buttons, this event is a touch event. When the button is pressed (or released), the callback function is run to perform the desired task.

The remainder of the file contains the emWin event handler function and a window setup function. Any callback function that is used for widgets will also be placed in this section.

NOTE: Most of the code segments in this guide are formatted to be copied and pasted to assist you in easily applying the code to your project.

2. We can add a new button below the FDI Logo by modifying the existing code. The definitions associated with the "Home Screen" text and buttons are shown below:

```
/*---------------------------------------------------------------------------*
 * Constants:
 *---------------------------------------------------------------------------*/
#define ID_WINDOW                (GUI_ID_USER + 0x00)
#define ID_TITLE_TEXT            (GUI_ID_USER + 0x01)
#define ID_LOGO                  (GUI_ID_USER + 0x02)
#define ID_BUTTON_RIGHT_TOP      (GUI_ID_USER + 0x03) // The
top button on the right side
#define ID_BUTTON_RIGHT_MIDDLE   (GUI_ID_USER + 0x04) // The
hidden middle button on the right side
#define ID_BUTTON_RIGHT_BOTTOM   (GUI_ID_USER + 0x05) // The
bottom button on the right side

#define WINDOW_XSIZE             (UEZ_LCD_DISPLAY_WIDTH)
#define WINDOW_YSIZE             (UEZ_LCD_DISPLAY_HEIGHT)
#define WINDOW_XPOS              (0)
#define WINDOW_YPOS              (0)

#if(UEZ_DEFAULT_LCD == LCD_RES_WVGA)
#define SPACING                  (10)
#else
#define SPACING                  (5)
#endif

#define TITLE_TEXT_XSIZE         (WINDOW_XSIZE)
#define TITLE_TEXT_YSIZE         ((WINDOW_YSIZE/10))
#define TITLE_TEXT_XPOS          (0)
#define TITLE_TEXT_YPOS          (0)
```

NOTE: Each widget, including the window itself, has a unique ID defined as shown above. Each additional widget increments by 1 and adding a new ID for an additional widget is as simple as setting it to the next value. GUI_ID_USER is equal to 0x800 by default and needs to be used to avoid conflicts with other areas of the program.

2. Make the following addition to the code:

```
/*---------------------------------------------------------------------------*
 * Constants:
 *---------------------------------------------------------------------------*/
#define ID_WINDOW                (GUI_ID_USER + 0x00)
#define ID_TITLE_TEXT            (GUI_ID_USER + 0x01)
#define ID_LOGO                  (GUI_ID_USER + 0x02)
#define ID_BUTTON_RIGHT_TOP      (GUI_ID_USER + 0x03) // The
top button on the right side
#define ID_BUTTON_RIGHT_MIDDLE   (GUI_ID_USER + 0x04) // The
hidden middle button on the right side
#define ID_BUTTON_RIGHT_BOTTOM   (GUI_ID_USER + 0x05) // The
bottom button on the right side
#define ID_MY_BUTTON             (GUI_ID_USER + 0x06) // Our
custom button

#define WINDOW_XSIZE             (UEZ_LCD_DISPLAY_WIDTH)
#define WINDOW_YSIZE             (UEZ_LCD_DISPLAY_HEIGHT)
#define WINDOW_XPOS              (0)
#define WINDOW_YPOS              (0)

#if(UEZ_DEFAULT_LCD == LCD_RES_WVGA)
#define SPACING                  (10)
#else
#define SPACING                  (5)
#endif

#define TITLE_TEXT_XSIZE         (WINDOW_XSIZE)
#define TITLE_TEXT_YSIZE         ((WINDOW_YSIZE/10))
#define TITLE_TEXT_XPOS          (0)
#define TITLE_TEXT_YPOS          (0)
```

3. The two arrays under the section header “Local Data” contain the information from the base application. Each widget is represented in one of the sections within each array.

```
/*---------------------------------------------------------------------------*
 * Local Data:
 *---------------------------------------------------------------------------
*/
/** Structure to hold all of the widgets used in this dialog*/
static const GUI_WIDGET_CREATE_INFO _iHomeScreenDialog[] = {
    //Function                  Name            ID                  XP
YP                      XS                      YS
    { WINDOW_CreateIndirect     , ""            , ID_WINDOW         ,
WINDOW_XPOS           , WINDOW_YPOS           , WINDOW_XSIZE          ,
WINDOW_YSIZE      , 0, 0, 0},
    { TEXT_CreateIndirect       , ""            , ID_TITLE_TEXT     ,
TITLE_TEXT_XPOS       , TITLE_TEXT_YPOS       , TITLE_TEXT_XSIZE      ,
TITLE_TEXT_YSIZE  , TEXT_CF_HCENTER| TEXT_CF_VCENTER, 0, 0},
    { IMAGE_CreateIndirect      , ""            , ID_LOGO           ,
LOGO_XPOS             , LOGO_YPOS             , LOGO_XSIZE            ,
LOGO_YSIZE        , 0, 0, 0},

    { BUTTON_CreateIndirect     , ""            , ID_BUTTON_RIGHT_TOP   ,
BUTTON_XPOS           , BUTTON_YPOS(1)        , BUTTON_XSIZE          ,
BUTTON_YSIZE      , 0, 0, 0},
    { BUTTON_CreateIndirect     , ""            , ID_BUTTON_RIGHT_MIDDLE,
BUTTON_XPOS           , BUTTON_YPOS(2)        , BUTTON_XSIZE          ,
BUTTON_YSIZE      , 0, 0, 0},
    { BUTTON_CreateIndirect     , ""            , ID_BUTTON_RIGHT_BOTTOM,
BUTTON_XPOS           , BUTTON_YPOS(3)        , BUTTON_XSIZE          ,
BUTTON_YSIZE      , 0, 0, 0},
};

/** Generic Mapping of Screen Layout*/
static T_LAFMapping HomeScreenMapping[] = {
    { ID_WINDOW             , ""                , GUI_BLACK, GUI_WHITE,
&FONT_SMALL, LAFSetupWindow , 0},
    { ID_LOGO               , ""                , GUI_BLACK, GUI_BLACK,
&FONT_SMALL, 0              , 0},
    { ID_TITLE_TEXT         , "Project Maker"   , GUI_BLACK,   GUI_WHITE,
&FONT_LARGE, LAFSetupText   , 0},
    { ID_BUTTON_RIGHT_TOP   , "Sensors"         , GUI_GRAY,  GUI_BLACK,
&FONT_LARGE, LAFSetupButton , (TBool (*)(WM_MESSAGE *, int,
int))IHandleButtonRightTop},
    { ID_BUTTON_RIGHT_MIDDLE, "GUI Builder"     , GUI_GRAY,  GUI_BLACK,
&FONT_LARGE, LAFSetupButton , (TBool (*)(WM_MESSAGE *, int,
int))IHandleButtonRightMiddle},
    { ID_BUTTON_RIGHT_BOTTOM, "Time/Date"       , GUI_GRAY,  GUI_BLACK,
&FONT_LARGE, LAFSetupButton , (TBool (*)(WM_MESSAGE *, int,
int))IHandleButtonRightBottom},
    {0},
};
```

4. Within the array “_iHomeScreenDialog[]”, add the following:

```
static const GUI_WIDGET_CREATE_INFO _iHomeScreenDialog[] = {
    //Function                  Name            ID                  XP
YP                      XS                      YS
    { WINDOW_CreateIndirect     , ""            , ID_WINDOW         ,
WINDOW_XPOS           , WINDOW_YPOS           , WINDOW_XSIZE          ,
WINDOW_YSIZE      , 0, 0, 0},
    { TEXT_CreateIndirect       , ""            , ID_TITLE_TEXT     ,
TITLE_TEXT_XPOS       , TITLE_TEXT_YPOS       , TITLE_TEXT_XSIZE      ,
TITLE_TEXT_YSIZE  , TEXT_CF_HCENTER| TEXT_CF_VCENTER, 0, 0},
    { IMAGE_CreateIndirect      , ""            , ID_LOGO           ,
LOGO_XPOS             , LOGO_YPOS             , LOGO_XSIZE            ,
LOGO_YSIZE        , 0, 0, 0},

    { BUTTON_CreateIndirect     , ""            , ID_BUTTON_RIGHT_TOP   ,
BUTTON_XPOS           , BUTTON_YPOS(1)        , BUTTON_XSIZE          ,
BUTTON_YSIZE      , 0, 0, 0},
    { BUTTON_CreateIndirect     , ""            , ID_BUTTON_RIGHT_MIDDLE,
BUTTON_XPOS           , BUTTON_YPOS(2)        , BUTTON_XSIZE          ,
BUTTON_YSIZE      , 0, 0, 0},
    { BUTTON_CreateIndirect     , ""            , ID_BUTTON_RIGHT_BOTTOM,
BUTTON_XPOS           , BUTTON_YPOS(3)        , BUTTON_XSIZE          ,
BUTTON_YSIZE      , 0, 0, 0},
{ BUTTON_CreateIndirect, "", ID_MY_BUTTON, (BUTTON_XPOS - BUTTON_XSIZE ) -
SPACING, BUTTON_YPOS(3), BUTTON_XSIZE, BUTTON_YSIZE, 0, 0, 0},
};
```

5. Within the array “T_LAFMapping HomeScreenMapping[]”, make the following change:

```
/** Generic Mapping of Screen Layout*/
static T_LAFMapping HomeScreenMapping[] = {
    { ID_WINDOW             , ""                , GUI_BLACK, GUI_WHITE,
&FONT_SMALL, LAFSetupWindow , 0},
    { ID_LOGO               , ""                , GUI_BLACK, GUI_BLACK,
&FONT_SMALL, 0              , 0},
    { ID_TITLE_TEXT         , "Project Maker"   , GUI_BLACK,   GUI_WHITE,
&FONT_LARGE, LAFSetupText   , 0},
    { ID_BUTTON_RIGHT_TOP   , "Sensors"         , GUI_GRAY,  GUI_BLACK,
&FONT_LARGE, LAFSetupButton , (TBool (*)(WM_MESSAGE *, int,
int))IHandleButtonRightTop},
    { ID_BUTTON_RIGHT_MIDDLE, "GUI Builder"     , GUI_GRAY,  GUI_BLACK,
&FONT_LARGE, LAFSetupButton , (TBool (*)(WM_MESSAGE *, int,
int))IHandleButtonRightMiddle},
    { ID_BUTTON_RIGHT_BOTTOM, "Time/Date"       , GUI_GRAY,  GUI_BLACK,
&FONT_LARGE, LAFSetupButton , (TBool (*)(WM_MESSAGE *, int,
int))IHandleButtonRightBottom},
    { ID_MY_BUTTON,           "My Window"       , GUI_GRAY,  GUI_BLACK,
&FONT_LARGE, LAFSetupButton , 0},
    {0},
};
```

6. Build, download, debug, and run the updated project to display the created button.
    a. For IAR: Press <F7> then press <Ctrl> + <D>.
    b. For Crossworks: Press <F7> then press <F5>.

    The button does not do anything yet because it lacks a callback function, but this will be added later once a second window has been created for the button to switch to.

Figure 18: *Button Added*

C. Creating a New Window

In order to have the button open a new window, a new window file similar to the HomeScreen.c file needs to be created. Since each window requires the same general structure, copy the existing HomeScreen.c file, instead of creating a new file from scratch.

1. Right click the HomeScreen.c file
    a. For IAR: Select “Open Containing Folder…”.
    b. For Crossworks: Select “Select in File Explorer”.
2. Copy the HomeScreen.c file located in the emWin directory and rename it SecondScreen.c.
3. Back in the IDE, right click the emWin folder.
    a. For IAR: Select “Add > Add Files…”.
    b. For Crossworks: Select “Add Existing File...”.
4. In the browse files dialog that appears, select SecondScreen.c and click *Open*.
5. Open the new file by double clicking it in the IDE workspace or project explorer.
6. Rename all instances of “HomeScreen” to “SecondScreen” (only in that file).

```
static void IUpdateFields(WM_MESSAGE * pMsg);
/*-----------------------------------------------------------------------
 * Local Data:
 *-----------------------------------------------------------------------
/** Structure to hold all of the widgets used in this dialog*/
static const GUI_WIDGET_CREATE_INFO _iSecondScreenDialog[] = {
    //Function              Name       ID                XP
    { WINDOW_CreateIndirect, "", ID_WINDOW             , WINDOW_XPOS
    { TEXT_CreateIndirect  , "", ID_TITLE_TEXT         , TITLE_TEXT_XPO
```

```
94
95 /** Generic Mapping of Screen Layout*/
96 static T_LAFMapping SecondScreenMapping[] = {
97     //ID, Text Label, label color, font color, font, 'L
98     { ID_WINDOW             , ""                , GUI_B
99     { ID_LOGO               , ""                , GUI_B
```

NOTE: To simplify this process, just press Ctrl+H to open the *Find and Replace* dialog. Insert "HomeScreen" in the *Find what* field and "SecondScreen" in the *Replace with* field. Select *Match case* and make sure *Match whole word* is unselected. Click *Replace All* and the entire document will be updated.

7. If it does not exist, above the "Local Data" section, add a new section – "Global Data" in the HomeScreen.c file.
8. Add the "G_WhichWindow" variable in the "Global Data" section.

```
/*---------------------------------------------------------------------------
-------*
 * Global Data:
 *---------------------------------------------------------------------------
-------*/
TInt32 G_WhichWindow;
```

9. Delete all the widget definitions that were specific to the home screen.

```
28 /*---------------------------------------------------------------------------*
29  * Constants:
30  *---------------------------------------------------------------------------*/
31 #define ID_WINDOW              (GUI_ID_USER + 0x00)
32 #define ID_TITLE_TEXT          (GUI_ID_USER + 0x01)
33 #define ID_LOGO                (GUI_ID_USER + 0x02)
34 #define ID_BUTTON_RIGHT_TOP    (GUI_ID_USER + 0x03) // The top button on the right side
35 #define ID_BUTTON_RIGHT_MIDDLE (GUI_ID_USER + 0x04) // The hidden middle button on the right side
36 #define ID_BUTTON_RIGHT_BOTTOM (GUI_ID_USER + 0x05) // The bottom button on the right side
37 #define ID_MY_BUTTON           (GUI_ID_USER + 0x06) // Our custom button
38
39 #define WINDOW_XSIZE           (UEZ_LCD_DISPLAY_WIDTH)
40 #define WINDOW_YSIZE           (UEZ_LCD_DISPLAY_HEIGHT)
41 #define WINDOW_XPOS            (0)
42 #define WINDOW_YPOS            (0)
43
44 #if(UEZ_DEFAULT_LCD == LCD_RES_WVGA)
45 #define SPACING                (10)
46 #else
47 #define SPACING                (5)
48 #endif
49
50 #define TITLE_TEXT_XSIZE       (WINDOW_XSIZE)
51 #define TITLE_TEXT_YSIZE       ((WINDOW_YSIZE/10))
52 #define TITLE_TEXT_XPOS        (0)
53 #define TITLE_TEXT_YPOS        (0)
54
55 #define LOGO_XSIZE             (160) //Taken from image resolution
56 #define LOGO_YSIZE             (72)
57 #define LOGO_XPOS              (SPACING*10)
58 #define LOGO_YPOS              ((WINDOW_YSIZE/2) - (LOGO_YSIZE/2))
59
60 #define BUTTON_XSIZE           ((WINDOW_XSIZE/2) - (2 * SPACING))
61 #define BUTTON_YSIZE           ((WINDOW_YSIZE / 4) - (4 * SPACING)) //Allow for 4 vertical buttons o
62 #define BUTTON_XPOS            ((WINDOW_XSIZE/2) + SPACING)
63 #define BUTTON_YPOS(n)         ((SPACING + TITLE_TEXT_YSIZE) + (n * (BUTTON_YSIZE + SPACING)))
```

10. Delete all function definitions that were specific to the home screen
    a. IHandleButtonRightTop
    b. IHandleButtonRightMiddle
    c. IHandleButtonRightBottom
    d. ISetLogo

11. Delete all function prototypes that were specific to the home screen
    e. IHandleButtonRightTop
    f. IHandleButtonRightMiddle
    g. IHandleButtonRightBottom
    h. IHandleMyButton
12. In the function definition for "_SecondScreenDialog", delete the call to "ISetLogo(pMsg)" and the WM_GetDialogItem function calls.

```
*       Callback function used by emWin to process events.
* Inputs:
*       WM_MESSAGE *pMsg -- message structure for current dialog.
*-------------------------------------------------------------------------*/
static void _SecondScreenDialog(WM_MESSAGE *pMsg)
{
    int Id, NCode;
    WM_HWIN hItem;

    switch (pMsg->MsgId){
    case WM_INIT_DIALOG:
        Id    = WM_GetId(pMsg->hWinSrc);
        NCode = pMsg->Data.v;
        LAFSetup(pMsg->hWin, ID_WINDOW, SecondScreenMapping);
        ISetLogo(pMsg);

                hItem = WM_GetDialogItem(pMsg->hWin, ID_BUTTON_RIGHT_MIDDLE);
                WM_HideWindow(hItem); // Hide the middle button until we have a GUI

                hItem = WM_GetDialogItem(pMsg->hWin, ID_TITLE_TEXT);
                //TEXT_SetText(hItem, "Demo"); // Example how to change the title te

        break;
    case WM_NOTIFY_PARENT:
        Id = WM_GetId(pMsg->hWinSrc);
```

13. The two arrays that hold the information for the various widgets on a window are shown below. Delete the lines highlighted in red that are specific to the first window. The only remaining values within each of these array variables define the window's properties and title text that will be used by emWin to setup the window.

```
/*---------------------------------------------------------------------------*
 * Local Data:
 *---------------------------------------------------------------------------*/
/** Structure to hold all of the widgets used in this dialog*/
static const GUI_WIDGET_CREATE_INFO _iSecondScreenDialog[] = {
    //Function               Name       ID                  XP                YP
    { WINDOW_CreateIndirect, "", ID_WINDOW             , WINDOW_XPOS    , WINDOW_YPO
    { TEXT_CreateIndirect  , "", ID_TITLE_TEXT         , TITLE_TEXT_XPOS, TITLE_TEXT
    { IMAGE_CreateIndirect , "", ID_LOGO               , LOGO_XPOS      , LOGO_YPOS

    { BUTTON_CreateIndirect, "", ID_BUTTON_RIGHT_TOP   , BUTTON_XPOS    , BUTTON_YPO
    { BUTTON_CreateIndirect, "", ID_BUTTON_RIGHT_MIDDLE, BUTTON_XPOS    , BUTTON_YPO
    { BUTTON_CreateIndirect, "", ID_BUTTON_RIGHT_BOTTOM, BUTTON_XPOS    , BUTTON_YPO
    { BUTTON_CreateIndirect, "", ID_MY_BUTTON, (BUTTON_XPOS - BUTTON_XSIZE ) - SPACI
};

/** Generic Mapping of Screen Layout*/
static T_LAFMapping SecondScreenMapping[] = {
    //ID, Text Label, label color, font color, font, 'Look-And-Feel' function (0 if
    { ID_WINDOW             , ""               , GUI_BLACK, GUI_WHITE, &FONT_SMALL,
    { ID_LOGO               , ""               , GUI_BLACK, GUI_BLACK, &FONT_SMALL,
    { ID_TITLE_TEXT         , "Project Maker"  , GUI_BLACK,   GUI_WHITE, &FONT_LARG
    { ID_BUTTON_RIGHT_TOP   , "Sensors"        , GUI_GRAY,  GUI_BLACK, &FONT_LARGE,
    { ID_BUTTON_RIGHT_MIDDLE, "GUI Builder"    , GUI_GRAY,  GUI_BLACK, &FONT_LARGE,
    { ID_BUTTON_RIGHT_BOTTOM, "Time/Date"      , GUI_GRAY,  GUI_BLACK, &FONT_LARGE,
    { ID_MY_BUTTON,           "My Window"      , GUI_GRAY,  GUI_BLACK, &FONT_LARGE,
    {0},
};
```

14. This file now contains the minimum required information to setup a blank window. The final step to creating the window is to update the WindowManager.c file. Open the file and update the WindowManager_Create_All_Active _Windows function as shown below:

```
/*---------------------------------------------------------------
---------------*
 * Routine: WindowManager_Create_All_Active_Windows
 *---------------------------------------------------------------
---------------*/
/** Create all the windows that the system will use.
 *
 */
 /*---------------------------------------------------------------
----------------*/
void WindowManager_Create_All_Active_Windows(void)
{
    static TBool iHaveRun = EFalse;

    if(!iHaveRun){
        TimeDate_Initialize();
        Sensor_Initialize();
       G_SystemWindows[HOME_SCREEN] = HomeScreen_Create();
       G_SystemWindows[TIMEDATE_SCREEN] = TimeDate_Create();
       G_SystemWindows[SENSOR_SCREEN] = Sensor_Create();
        //TODO add new window creation function here.
        G_SystemWindows[SECOND_SCREEN] =
SecondScreen_Create();
    }
}
```

You also need to add a #define identifier for SECOND_SCREEN to the top of the WindowManager.h file as shown below if not already present:

```
//each window has a unique identifier
#define HOME_SCREEN                     (0)
#define TIMEDATE_SCREEN                 (1)
#define SENSOR_SCREEN                   (2)
#define SECOND_SCREEN                   (3)
```

D. Adding Callback Functionality

Now that the second screen has been created, return to HomeScreen.c and add a callback function to the previously created button. Once this callback function is created, the button will switch to the newly created second window when it is pressed.

1. Add the name of the callback function into HomeScreenMapping matrix. (the "My Window" button) You can choose any name to give the callback function. In this example we used **(TBool (*)(WM_MESSAGE *, TInt32, TInt32)) IHandleSecondScreen** to make the function name easily understandable.

```
/** Generic Mapping of Screen Layout*/
static T_LAFMapping HomeScreenMapping[] = {
    { ID_WINDOW             , ""                , GUI_BLACK, GUI_WHITE,
&FONT_SMALL, LAFSetupWindow , 0},
    { ID_LOGO               , ""                , GUI_BLACK, GUI_BLACK,
&FONT_SMALL, 0              , 0},
    { ID_TITLE_TEXT         , "Project Maker"   , GUI_BLACK,
GUI_WHITE, &FONT_LARGE, LAFSetupText    , 0},
    { ID_BUTTON_RIGHT_TOP   , "Sensors"         , GUI_GRAY,  GUI_BLACK,
&FONT_LARGE, LAFSetupButton , (TBool (*)(WM_MESSAGE *, int,
int))IHandleButtonRightTop},
    { ID_BUTTON_RIGHT_MIDDLE, "GUI Builder"     , GUI_GRAY,  GUI_BLACK,
&FONT_LARGE, LAFSetupButton , (TBool (*)(WM_MESSAGE *, int,
int))IHandleButtonRightMiddle},
    { ID_BUTTON_RIGHT_BOTTOM, "Time/Date"       , GUI_GRAY,  GUI_BLACK,
&FONT_LARGE, LAFSetupButton , (TBool (*)(WM_MESSAGE *, int,
int))IHandleButtonRightBottom},
    { ID_MY_BUTTON,           "My Window"       , GUI_GRAY,  GUI_BLACK,
&FONT_LARGE, LAFSetupButton , (TBool (*)(WM_MESSAGE *, int,
int))IHandleSecondScreen},
    {0},
};
```

2. Declare the prototype for the IHandleSecondSCreen callback function in the "Prototypes" sections of the HomeScreen.c file.

```
/*-------------------------------------------------------------------
--------*
 * Prototypes:
 *-------------------------------------------------------------------
--------*/
static TBool IHandleButtonRightTop(WM_MESSAGE * pMsg, int aNCode, int
aID);
static TBool IHandleButtonRightMiddle(WM_MESSAGE * pMsg, int aNCode,
int aID);
static TBool IHandleButtonRightBottom(WM_MESSAGE * pMsg, int aNCode,
int aID);
static TBool IHandleSecondScreen(WM_MESSAGE * pMsg, int aNCode, int
aID);
```

3. Create the callback function below the “Local Data” section. This function takes a message from the event handler and switches windows when the button is pressed.

```
/*---------------------------------------------------------------
------------*
 * Routine: IHandleSecondScreen
 *---------------------------------------------------------------
------------*
 * Description:
 *      Change to the second screen when our new button is
pressed
 *
 *---------------------------------------------------------------
------------*/
static TBool IHandleSecondScreen(WM_MESSAGE * pMsg, int aNCode,
int aID)
{
    if (aNCode == WM_NOTIFICATION_RELEASED)
    {
        WindowManager_Show_Window(SECOND_SCREEN);
    }
    return EFalse;
}
```

4. Use the appropriate steps to Build, download, debug, and run the updated project to test the functionality. Since the second window contains no widgets, all we will see is a blank screen.

Figure 19: *Blank window after clicking our button*

E. Setup for Interfacing with the Onboard Temperature Sensor

In the following steps of this guide, we will demonstrate how to interface with the uEZ GUI hardware by recreating the temperature sensor demo that is built into the Project Maker, and how to update the temperature on the screen by creating a task (More details below in *Section F – Creating a Task*). For now we will focus on setting up our SecondScreen.

Start by returning to the SecondScreen.c file in the IDE, then creating two new text widgets. One text widget will be a title for the temperature screen, and the other will show the onboard temperature sensor value.

1. Open SecondScreen.c to add the new text fields and give the temperature text an update function.
2. Define a new ID for the Temperature Text and Title Text.

```
#define ID_WINDOW                   (GUI_ID_USER + 0x00)
#define ID_TITLE_TEXT               (GUI_ID_USER + 0x01)
#define ID_TEMP_TEXT                (GUI_ID_USER + 0x02)
```

3. Define the temperature widgets' positions and sizes.

```
#define TITLE_TEXT_XSIZE            (WINDOW_XSIZE)
#define TITLE_TEXT_YSIZE            ((WINDOW_YSIZE/10))
#define TITLE_TEXT_XPOS             (0)
#define TITLE_TEXT_YPOS             (0)

#define TEMP_TEXT_XSIZE             (WINDOW_XSIZE/3)
#define TEMP_TEXT_YSIZE             (WINDOW_YSIZE/12)
#define TEMP_TEXT_XPOS              ((WINDOW_XSIZE/2) -
(TEMP_TEXT_XSIZE/2))
#define TEMP_TEXT_YPOS              (WINDOW_YSIZE/2) -
(TEMP_TEXT_YSIZE)
```

NOTE: We use WINDOW_XSIZE and WINDOW_YSIZE to size and place all GUI elements relative to the screen, so it is easy to change to a different LCD screen resolution without need to re-write the element placement code.

NOTE: The µEZ GUI screen coordinates are defined such that the coordinates (0,0) are located at the upper left corner of the screen.

4. Add the temperature widget to each of the "Local Data" arrays. This text field will not need a callback function. Instead, the temperature update function will be called periodically by its own task which will be created later. Also, rename the title text to display "Temperature Screen".

```
/*---------------------------------------------------------------------------*
 * Local Data:
 *---------------------------------------------------------------------------
*/
/** Structure to hold all of the widgets used in this dialog*/
static const GUI_WIDGET_CREATE_INFO _iSecondScreenDialog[] = {
    { WINDOW_CreateIndirect, "", ID_WINDOW, WINDOW_XPOS, WINDOW_YPOS,
WINDOW_XSIZE, WINDOW_YSIZE, 0, 0, 0},
    { TEXT_CreateIndirect, "", ID_TITLE_TEXT, TITLE_TEXT_XPOS,
TITLE_TEXT_YPOS, TITLE_TEXT_XSIZE, TITLE_TEXT_YSIZE, TEXT_CF_HCENTER|
TEXT_CF_VCENTER, 0, 0},
    { TEXT_CreateIndirect, "", ID_TEMP_TEXT, TEMP_TEXT_XPOS, TEMP_TEXT_YPOS,
TEMP_TEXT_XSIZE, TEMP_TEXT_YSIZE, 0, 0, 0},
};

/** Generic Mapping of Screen Layout*/
static T_LAFMapping SecondScreenMapping[] = {
    { ID_WINDOW             , ""                , GUI_BLACK, GUI_WHITE,
&FONT_LARGE, LAFSetupWindow , 0},
    { ID_TITLE_TEXT, "Temperature Screen", GUI_BLACK, GUI_WHITE, &FONT_LARGE,
LAFSetupText, 0},
    { ID_TEMP_TEXT, "Temp", GUI_BLACK, GUI_WHITE, &FONT_LARGE, LAFSetupText,
0},
    {0},
};
```

5. Add the function for updating the temperature. This function is a wrapper that calls the emWin function to change the text value of the text widget. It updates the text to whatever character array is passed to it through the “myString” argument.

```
/*---------------------------------------------------------------------------*
 * Routine:    UpdateTemp
 *---------------------------------------------------------------------------*
 * Description:
 *      Update the temperature text.
 *---------------------------------------------------------------------------*/
void UpdateTemp(char *myString)
{
    TEXT_SetText(WM_GetDialogItem(G_WhichWindow, ID_TEMP_TEXT), (const
char*)myString);
}
```

6. Add the line shown below to the beginning of the _SecondScreenDialog, to update the ID of the active window.

```
static void _SecondScreenDialog(WM_MESSAGE *pMsg)
{
    int Id, NCode;
    WM_HWIN hItem;
    G_WhichWindow = pMsg->hWin;
```

F. Creating a Task

The next set of steps will describe the process for setting up a new task to monitor the data provided by the onboard I²C digital temperature sensor. Creating new tasks allows multiple continuous processes to operate “simultaneously” by taking advantage of FreeRTOS’ ability to simulate threading without the need for complex hardware. In effect, tasks are equivalent to threads. It is important that a small delay is introduced

into the task's loop to avoid bogging down the processor. Doing this allows the other tasks to take priority during the delayed time and doesn't use unnecessary processing. The following flow diagram describes the task as it will be designed.

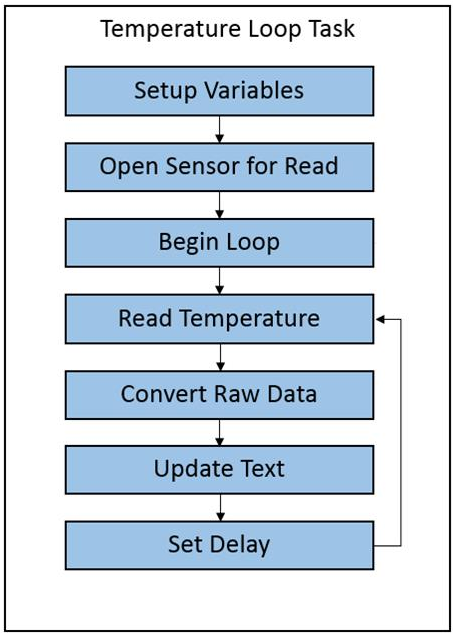


Figure 120: *Temp Loop Task Flow*

In the Project Maker sensors demo, the temperature sensor is read periodically using a timer.

We will create a task for reading the temperature sensor using delays.

1. If using IAR…
    a. Navigate to your App directory in the file explorer ("C:\Users\(**user**)\Desktop\uEZ-Workspace\SampleProjectIAR\Source\App")
    b. Right Click -> new -> text file
    c. Name your text file "CustomTasks.c"
    d. Right Click -> new -> text file
    e. Name your text file "CustomTasks.h"
    f. Back in IAR…
        i. Right click on App in the project explorer
        ii. Click Add -> Add Files…
        iii. Select CustomTasks.c and CustomTasks.h
2. In Crossworks…
    a. Right Click on the App folder in the project explorer
    b. select "Add new file"
    c. Select "C File" and enter the name "CustomTasks.c".
    d. Select <OK>
    e. Right click the "App" directory again and select "add new file"
    f. Select "Header File" and name it "CustomTasks.h"

3. We will declare our prototypes for our new task in CustomTasks.h, and define them in CustomTasks.c
4. We first need a prototype for our temperature reading function.
5. We also need a custom type for storing the read temperature. We will use a struct rather than a variable so that we can expand on this later if we wish, such as by adding Fahrenheit readings.
6. CustomTasks.h will look as follows:

```
/*-------------------------------------------------------------------------
-----*
 * File: CustomTasks.h
 *-------------------------------------------------------------------------
-----*/

/*-------------------------------------------------------------------------
-----*
 * uEZ(R) - Copyright (C) 2007-2016 Future Designs, Inc.
 *-------------------------------------------------------------------------
-----*/

/*-------------------------------------------------------------------------
-----*
 * Function Prototypes:
 *-------------------------------------------------------------------------
-----*/
TUInt32 TemperatureReadLoopTask(T_uezTask aMyTask, void *aParams);

/*-------------------------------------------------------------------------
-----*
 * Types:
 *-------------------------------------------------------------------------
-----*/
typedef struct {
        char iBoardTemp_celsius[32];
}T_TempReadings;

/*-------------------------------------------------------------------------
-----*
 * End of File:    CustomTasks.h
 *-------------------------------------------------------------------------
-----*/
```

7. Our Header file is now complete for creating the temp-reading task.
8. In our CustomTasks.c file, we need to define our TemperatureReadLoopTask
9. In CustomTasks.c paste the following code:

```
/*-------------------------------------------------------------------
------*
 * File: CustomTasks.c
 *-------------------------------------------------------------------
------*/

/*-------------------------------------------------------------------
------*
 * uEZ(R) - Copyright (C) 2007-2016 Future Designs, Inc.
 *-------------------------------------------------------------------
------*/

/*-------------------------------------------------------------------
------*
 * Includes:
 *-------------------------------------------------------------------
------*/
#include <uEZ.h>
#include <uEZTemperature.h>
#include <uEZGPIO.h>
#include <uEZProcessor.h>
#include "CustomTasks.h"
#include <stdio.h>

/*-------------------------------------------------------------------
------*
 * Globals:
 *-------------------------------------------------------------------
------*/
static T_TempReadings G_SensorSettings = {"00.0 C"}; // Initialize
the celsius reading
```

```
/*-------------------------------------------------------------------
------*
 * Routine:    TemperatureReadLoopTask
 *-------------------------------------------------------------------
------*
 * Description:
 *      Task to take temperature reading and update screen
 *-------------------------------------------------------------------
------*/
TUInt32 TemperatureReadLoopTask(T_uezTask aMyTask, void *aParams)
{
    TInt32 reading;
    TInt32 whole, fract;
    T_uezDevice temp;
    TInt32 openSensorResult = UEZTemperatureOpen("Temp0", &temp);
    if(openSensorResult == UEZ_ERROR_NONE)
    {
        while (1)
        {
            T_uezError tempReadErr = UEZTemperatureRead(temp,
&reading);
            if (tempReadErr == UEZ_ERROR_NONE)
            {
                whole = reading >> 16;
                // Convert to 1 digit decimal
                fract = ((((TUInt32)reading) & 0xFFFF) * 10) >> 16;
                // Put the read temperature into the temp-reading
member of the struct
                sprintf(G_SensorSettings.iBoardTemp_celsius, "Temp:
%02d.%01d C", whole, fract);
            }
            UEZTaskDelay(250);
        }
    }
    else
    {
        printf("Error %d. Failed to open temp sensor.\r\n",
openSensorResult);
    }
    return openSensorResult;
}

/*-------------------------------------------------------------------
------*
 * End of File:    CustomTasks.c
 *-------------------------------------------------------------------
------*/
```

10. We currently do not have a way to report the temperature reading back to SecondScreen. This can be resolved by adding an **extern** function prototype in the file or function where we want the change to occur, and then defining the prototype in the file where we want to see the change take place.

11. Add the following section to “CustomTasks.c”:

```
/*---------------------------------------------------------------
----------*
 * Externs:
 *---------------------------------------------------------------
----------*/
extern void UpdateTempText(char *myString); // Access to the
update function in SecondScreen.c
```

12. Now, in “SecondScreen.c”, add the following function definition:

```
/*---------------------------------------------------------------
----------*
 * Routine:    UpdateTemp
 *---------------------------------------------------------------
----------*
 * Description:
 *      Update the temperature text.
 *---------------------------------------------------------------
----------*/
void UpdateTempText(char *myString)
{
    TEXT_SetText(WM_GetDialogItem(G_WhichWindow, ID_TEMP_TEXT),
(const char*)myString);
}
```

13. “TEXT_SetText” is an emWin function for setting the text of a text widget within a specified window.
14. In “SecondScreen.c”, add the following #include at the top:

```
#include "CustomTasks.h"
```

This will expose SecondScreen to CustomTasks so that UpdateTempText will work.

15. Now back in “CustomTasks.c”, add the following line to the “TemperatureReadLoopTask” function:

```
/*-------------------------------------------------------------------
-------*
 * Routine:    TemperatureReadLoopTask
 *-------------------------------------------------------------------
-------*
 * Description:
 *      Task to take temperature reading and update screen
 *-------------------------------------------------------------------
-------*/
TUInt32 TemperatureReadLoopTask(T_uezTask aMyTask, void *aParams)
{
    TInt32 reading;
    TInt32 whole, fract;
    T_uezDevice temp;
    TInt32 openSensorResult = UEZTemperatureOpen("Temp0", &temp);
    if(openSensorResult == UEZ_ERROR_NONE)
    {
        while (1)
        {
            T_uezError tempReadErr = UEZTemperatureRead(temp,
&reading);
            if (tempReadErr == UEZ_ERROR_NONE)
            {
                whole = reading >> 16;
                // Convert to 1 digit decimal
                fract = ((((TUInt32)reading) & 0xFFFF) * 10) >> 16;
                // Put the read temperature into the temp-reading
member of the struct
                sprintf(G_SensorSettings.iBoardTemp_celsius, "Temp:
%02d.%01d C", whole, fract);
                // Update the temperature text
                UpdateTempText(G_SensorSettings.iBoardTemp_celsius);
            }
            UEZTaskDelay(250);
                    }
    }
    else
    {
        printf("Error %d. Failed to open temp sensor.\r\n",
openSensorResult);
    }
    return openSensorResult;
}

/*-------------------------------------------------------------------
-------*
 * End of File:    CustomTasks.c
 *-------------------------------------------------------------------
-------*/
```

16. Lastly, we need to kick off this temperature-reading task when the application starts.
17. In “main.c”, add the following include at the top:

```
#include "CustomTasks.h"
```

18. Now add this line below the HeartBeat task’s initialization:

```
// Start up the heart beat of the LED first thing.
    UEZTaskCreate(HeartbeatTask, "Heart", 64, (void *)0,
UEZ_PRIORITY_NORMAL, 0);

// Temp-reading task
UEZTaskCreate(TemperatureReadLoopTask, "TempTask", 1024, (void
*)0, UEZ_PRIORITY_NORMAL, 0);
```

19. Build and run the program.
20. When you press the “My Window” button, the SecondScreen window will now display the title and the temperature. The temperature will update periodically.

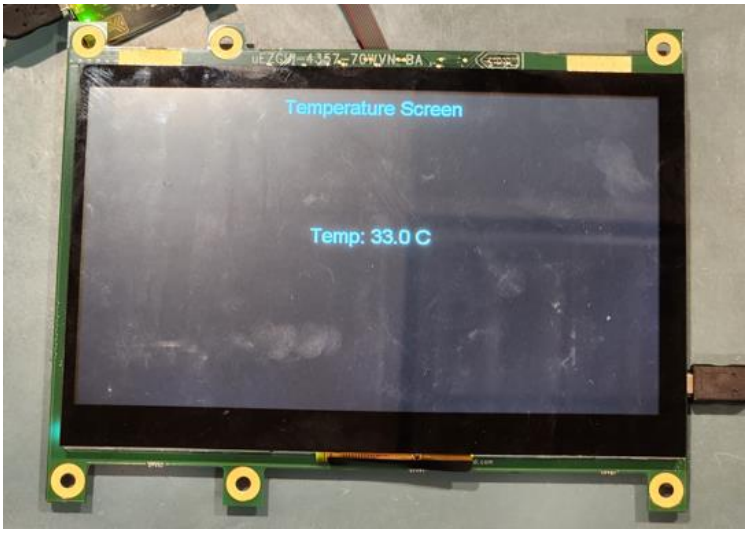


Figure 21: *Second Screen with temp*

G. Adding a Back Button

In the following steps, we will add a back button to the second screen by defining the ID, size, and position inside SecondScreen.c. Then, update the widget matrices and add a callback function that will switch back to the home screen.

1. Open SecondScreen.c.
2. Define the new button widget’s ID and properties.

```
#define ID_WINDOW                   (GUI_ID_USER + 0x00)
#define ID_TITLE_TEXT               (GUI_ID_USER + 0X01)
#define ID_TEMP_TEXT                (GUI_ID_USER + 0X02)
#define ID_BACK_BUTTON              (GUI_ID_USER + 0X03)

#define WINDOW_XSIZE                (UEZ_LCD_DISPLAY_WIDTH)
#define WINDOW_YSIZE                (UEZ_LCD_DISPLAY_HEIGHT)
#define WINDOW_XPOS                 (0)
#define WINDOW_YPOS                 (0)

#define TITLE_TEXT_XSIZE            (WINDOW_XSIZE)
#define TITLE_TEXT_YSIZE            ((WINDOW_YSIZE / 10))
#define TITLE_TEXT_XPOS             (0)
#define TITLE_TEXT_YPOS             (0)

#define TEMP_TEXT_XSIZE             (WINDOW_XSIZE / 2)
#define TEMP_TEXT_YSIZE             (WINDOW_YSIZE / 12)
#define TEMP_TEXT_XPOS              ((WINDOW_XSIZE / 2) -
(TEMP_TEXT_XSIZE / 2))
#define TEMP_TEXT_YPOS              (WINDOW_YSIZE / 2) -
(TEMP_TEXT_YSIZE)

#define BACK_BUTTON_XSIZE           (WINDOW_XSIZE / 4)
#define BACK_BUTTON_YSIZE           ((WINDOW_YSIZE / 10))
#define BACK_BUTTON_XPOS            ((WINDOW_XSIZE / 2) -
(BACK_BUTTON_XSIZE / 2))
#define BACK_BUTTON_YPOS            ((WINDOW_YSIZE / 2) +
(TEMP_TEXT_YSIZE / 2))
```

3. Add a line in both of the local data arrays for the new back button.

```
/*---------------------------------------------------------------------------*
 * Local Data:
 *---------------------------------------------------------------------------*/
/** Structure to hold all of the widgets used in this dialog*/
static const GUI_WIDGET_CREATE_INFO _iSecondScreenDialog[] = {
    //Function, Name, ID, XP, YP, XS, YS
    { WINDOW_CreateIndirect, "", ID_WINDOW, WINDOW_XPOS, WINDOW_YPOS,
WINDOW_XSIZE, WINDOW_YSIZE, 0, 0, 0},
    { TEXT_CreateIndirect, "", ID_TITLE_TEXT, TITLE_TEXT_XPOS, TITLE_TEXT_YPOS,
TITLE_TEXT_XSIZE, TITLE_TEXT_YSIZE, TEXT_CF_HCENTER | TEXT_CF_VCENTER, 0, 0},
    { TEXT_CreateIndirect, "", ID_TEMP_TEXT, TEMP_TEXT_XPOS, TEMP_TEXT_YPOS,
TEMP_TEXT_XSIZE, TEMP_TEXT_YSIZE, TEXT_CF_HCENTER | TEXT_CF_VCENTER, 0, 0},
    { BUTTON_CreateIndirect, "", ID_BACK_BUTTON, BACK_BUTTON_XPOS,
BACK_BUTTON_YPOS, BACK_BUTTON_XSIZE, BACK_BUTTON_YSIZE, 0, 0, 0},
};

/** Generic Mapping of Screen Layout*/
static T_LAFMapping SecondScreenMapping[] = {
    { ID_WINDOW, "", GUI_BLACK, GUI_WHITE, &FONT_SMALL, LAFSetupWindow , 0},
    { ID_TITLE_TEXT, "Temperature Screen", GUI_BLACK, GUI_WHITE, &FONT_LARGE,
LAFSetupText , 0},
    { ID_TEMP_TEXT, "Temp", GUI_BLACK, GUI_WHITE, &FONT_LARGE, LAFSetupText ,
0},
    { ID_BACK_BUTTON, "Back", GUI_RED, GUI_WHITE, &FONT_LARGE, LAFSetupButton ,
0},
    {0},
};
```

4. Create the callback function for switching back to the home screen by adding the associated handling function to the file below the local data arrays.

```
/*---------------------------------------------------------------------------*
 * Routine:    IHandleBackButton
 *---------------------------------------------------------------------------*
 * Description:
 *      Change to the home screen when the back button is pressed.
 *---------------------------------------------------------------------------*/
static TBool IHandleBackButton(WM_MESSAGE * pMsg, int aNCode, int aID)
{
    if (aNCode == WM_NOTIFICATION_RELEASED)
    {
        WindowManager_Show_Window(HOME_SCREEN);
    }
    return EFalse;
}
```

5. Add the function prototype after the “Global Data” section at the top of the file.

```
/*---------------------------------------------------------------------------*
 * Function Prototypes:
 *---------------------------------------------------------------------------*/
static TBool IHandleBackButton(WM_MESSAGE * pMsg, int aNCode, int aID);
```

6. Now add this function to the SecondScreenMapping array's Back Button entry

```
/** Generic Mapping of Screen Layout*/
static T_LAFMapping SecondScreenMapping[] = {
    { ID_WINDOW, "", GUI_BLACK, GUI_WHITE, &FONT_SMALL, LAFSetupWindow , 0},
    { ID_TITLE_TEXT, "Temperature Screen", GUI_BLACK, GUI_WHITE, &FONT_LARGE,
LAFSetupText , 0},
    { ID_TEMP_TEXT, "Temp", GUI_BLACK, GUI_WHITE, &FONT_LARGE, LAFSetupText ,
0},
    { ID_BACK_BUTTON, "Back", GUI_RED, GUI_WHITE, &FONT_LARGE, LAFSetupButton
, (TBool (*)(WM_MESSAGE *, TInt32, TInt32)) IHandleBackButton},
    {0},
};
```

7. Use the appropriate steps to Build, download, debug, and run the updated project to test the new functionality.

Figure 22: *Final Screen*

Congratulations! This concludes the walkthrough for building a simple GUI! Professional looking designs with complex functionality can be designed using advanced features provided in the emWin and µEZ libraries. Documentation and support for learning these features is available through Future Designs, Inc. for emWin and µEZ libraries.

The following optional section provides instructions for how to restore the factory demo to the µEZ GUI if desired.

6. Restoring the Out-of-Box (OOB) Demo (Optional)

1. Remove the SD card from the µEZ GUI.
2. Insert the µEZ GUI's SD card into the PC (depending on your PC, an SD card reader or adapter may be required).
3. Rename the file "INSTALL.FIN" to "INSTALL.INI", and click OK when prompted with a warning about changing the file extension.
4. In the root directory of the SD card, the folder titled "boot" contains the flash binary and a J-Link script that will flash the original out-of-box to the connected µEZ GUI.

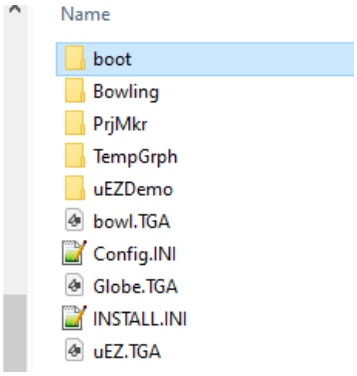


Figure 23: *Boot directory*

5. Connect the PC to the µEZ GUI via the J-Link debugger probe and double-click the "Install OOB.jlink" script.

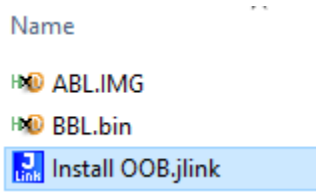


Figure 24: *OOB bootloader and demo*

Note: This will reinstall the base bootloader which was overwritten by the project maker example we worked through above.

6. Re-insert the SD card into the uEZGUI
7. The OOB demo will now be restored.

7. Guides and User's Manuals

Additional examples can be found on the www.TeamFDI.com website including but not limited to the following application notes:

- RS-232 Serial Communications: http://tinyurl.com/j24l5ss
- TTL UART Serial Communications: http://tinyurl.com/zrc7gp2
- Video Conversion Guide: ..... http://tinyurl.com/jxthcvv

FDI also provides documentation for using the demo bootloader for playing videos and slideshows as well as running other applications from the menu.

- Slideshow Creation Guide: http://tinyurl.com/zlu9ood
- Bootloader User's Manual: http://tinyurl.com/z6lw2fd
- JTAG Programming µEZ GUI: http://tinyurl.com/h9dnu96

8. Website and Support

Documentation:

- µEZ Library Online Documentation ..... http://www.TeamFDI.com/uez/docs/
- emWin Documentation ..... https://www.segger.com/emWin.html
- emWin 5.48 user's manual http://tinyurl.com/go4lj4j

Support & Downloads:

- FDI Support Home Page ..... http://www.TeamFDI.com/support
- FDI Forums ..... http://www.teamfdi.com/forum
- emWin FAQs ..... https://www.segger.com/emWin-faqs.html
- µEZGUI-4357-70WVN https://tinyurl.com/bdexh7mv
- µEZ Source w/ Project Maker (v2.12 or later) ..... http://sourceforge.net/projects/uez/files/
- Start Here Guide ..... http://www.teamfdi.com/StartHere

Contact Information:

Future Designs, Inc.

996 A Cleaner Way SW
Huntsville, AL 35805
Phone: (256) 883-1240
Fax: (256) 883-1241
Email: Sales@TeamFDI.com

http://www.TeamFDI.com