µEZ GUI Start Here Guide

UEZGUI-1788-70WVM

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-1788-70WVM-BA
- SEGGER J-Link Lite Cortex-M Probe
- FDI J-Link 20-pin to mini-10 pin adapter model JLINK-ARM-AD
- USB Type A to USB Type Mini B Cable (2x)
- Universal AC to 5V USB plug Power Supply Unit

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

- SEGGER J-Link Software
- µEZ Source w/ Project Maker (v2.08 or later)
- One of the following IDEs:
  - Rowley CrossWorks v3.6.5
  - IAR Embedded Workbench v7.60

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

- Start Here Project Files (located in the µEZ source download)

Contents

1. Hardware Verification 3

2. Software Installation 4

A. IDE Installation 4

B. J-Link Installation 4

C. µEZ Installation 4

3. Connecting the J-Link Debugger to the µEZ GUI for programming 5

4. Developing a Simple GUI Application with emWin 5

A. emWin Introduction 5

B. Building and Running the Initial Project 6

C. Adding a Button to the Home Screen 11

D. Creating a New Window 15

E. Adding Callback Functionality 19

F. Interfacing with the Onboard Temperature Sensor 20

G. Creating a Task 22

H. Adding a Back Button 24

5. Restoring the Out-of-Box (OOB) Demo (Optional) 27

6. Next Steps 28

7. Website and Support 28

Important Legal Information

Information in this document is provided solely to enable the use of Future Designs products. FDI assumes no liability whatsoever, including infringement of any patent or copyright. FDI reserves the right to make changes to these specifications at any time, without notice. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of Future Designs, Inc. (FDI) 996 A Cleaner Way, Huntsville, AL 35805.

For more information on FDI or our products please visit www.teamfdi.com.

NOTE: The inclusion of vendor software products in this kit does not imply an endorsement.
© 2016 Future Designs, Inc. All rights reserved.

µEZ® is a registered trademark of Future Designs, Inc. Windows is a registered trademarks of Microsoft. emWin and J-Link are registered trademarks of SEGGER Microcontroller GmbH & Co. KG. IAR EMBEDDED WORKBENCH is a registered trademark of I.A.R. Systems AB.

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-1788-70WVM*

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

1. Connect the USB cable to the mini USB Type B connector (P2) on the µEZ GUI.
2. 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. Test the GUI by selecting "Draw" and drawing lines on the screen with your finger.

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 v7.60 and Rowley CrossWorks 3.6.5 on the LPC1788. This guide uses IAR Embedded Workbench for ARM version 7.60 and Rowley Crossworks for ARM Version 3.6.5. Download your preferred IDE at the appropriate link provided below and install according to the instructions provided on their respective websites.

- IAR Embedded Workbench 7.60 30-day free trial: (select Evaluation License option on install)
  Windows: http://tinyurl.com/glljfsj
  - IAR 30-day evaluation license manual activation instructions: http://tinyurl.com/zyctkha
- Rowley CrossWorks for ARM version 3.6.5 30-day free trial:
  Windows: http://tinyurl.com/z2wgnsk
  Mac: http://tinyurl.com/he4tgtu
  Linux x86: http://tinyurl.com/zzcu5ny
  Linux ARMV7: http://tinyurl.com/hvzpq8e
  - Activating Crossworks 30-day evaluation license instructions: http://tinyurl.com/hlsn9qf

B. J-Link Installation

The µEZ GUI uses a J-Link debugger probe. The SEGGER version 5.12J driver must be installed. Download and install the software from the link below:

- SEGGER J-Link Software 5.12j:
  Windows: http://tinyurl.com/jr6s6xl
  Mac: http://tinyurl.com/zshoj6r
  Linux x86: http://tinyurl.com/jdkzsx9
  Linux ARMV7: http://tinyurl.com/zat25fs

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 (v2.08): 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.

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

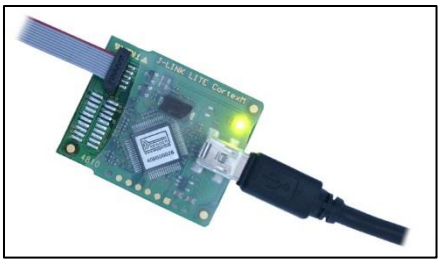


Figure 3: *J-Link Hardware*

1. Connect the J-Link probe to a PC with an included USB cable.
2. Connect the J-Link probe to the Cortex-M 9-pin JTAG connector (J3) on the bottom side of the µEZ GUI. The J-Link 9 pin ribbon cable for the µEZ GUI is supplied in the kit. The 9-pin connector can only be connected one way.
3. If not already done, connect the µEZ GUI to the power supply with the second included USB cable.

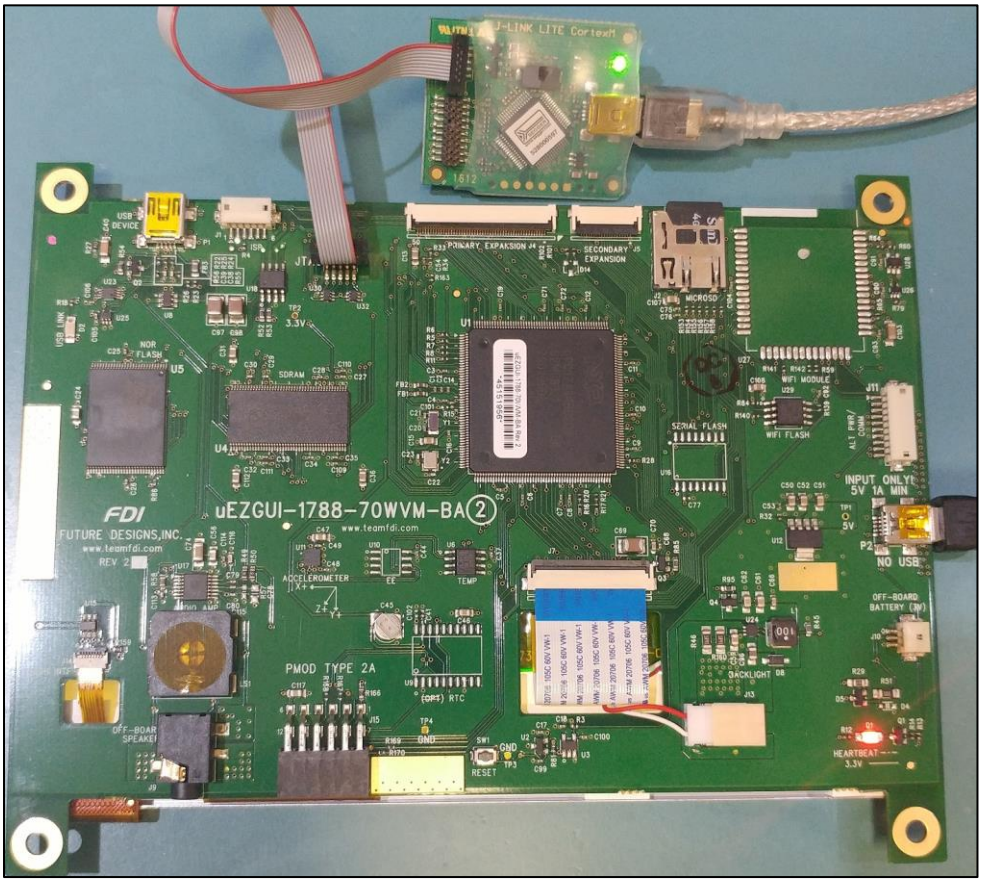


Figure 4: *J-Link Connected to µEZ GUI*

4. 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-1788-70WVM-BA utilizes the LPC1788, 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. Building and Running the Initial Project

Before digging into the code of the project, examine the output by building and running the initial project as outlined in the steps below:

NOTE: The following sections reference the uEZ files from Sourceforge. In the following instructions, the "uEZ_v2.XX_SourceForge" files were unpackaged to the C:\Workspace folder on a Windows PC.

1. Navigate to the extracted folder from the µEZ download. The beginning project for the Start Here Guide is located in the µEZ download in the uEZ_v2.XX_SourceForge/StartHere folder. The finished project is also included in the uEZ_v2.XX_SourceForge/StartHereEnd folder for you to reference as needed.

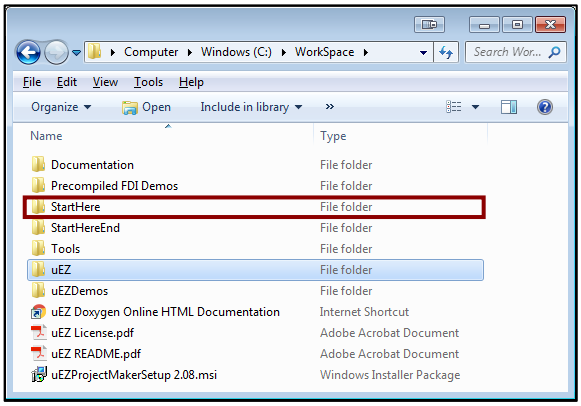


Figure 5: *µEZ files from SourceForge Download (v2.08 shown)*

2. Locate the "uEZ" folder within the µEZ download. This folder includes the µEZ library files needed by the project to function. The uEZ folder must always be in the same directory as the project's main folder, like it is in the download. Otherwise, µEZ projects would not be able to get the library files they need to compile.
3. Navigate through the uEZ folder to uEZ_v2.XX_SourceForge/uEZ/Build/Generic/NXP. Then, locate the folder corresponding to the microcontroller used in the µEZ GUI. NOTE: The processor number is the first set of numbers in the model number of the µEZ GUI. This UEZGUI-1788-70WVM uses an LPC1788 micro-controller and the corresponding folder referenced in this step is labeled accordingly.

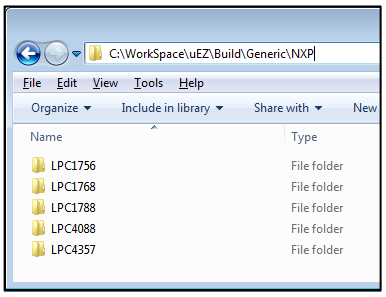


Figure 6: *Navigating to the µEZ Workspace*

4. Navigate to FreeRTOS\IAR\ or FreeRTOS\Crossworks\ and open the IAR (uEZ_NXP_LPC1788_FreeRTOS_IAR.eww) or Crossworks (uEZ_NXP_LPC1788_FreeRTOS_CrossWorks.hzp) workspace/project file. If you have multiple versions of IAR or Crossworks installed, make sure that these projects are opened in the correct installation of the respective IDE.

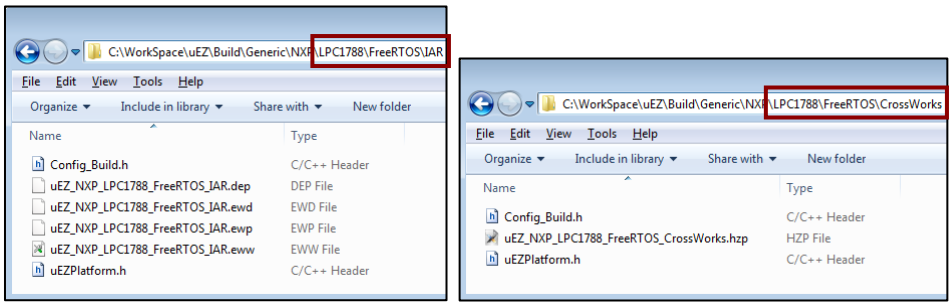


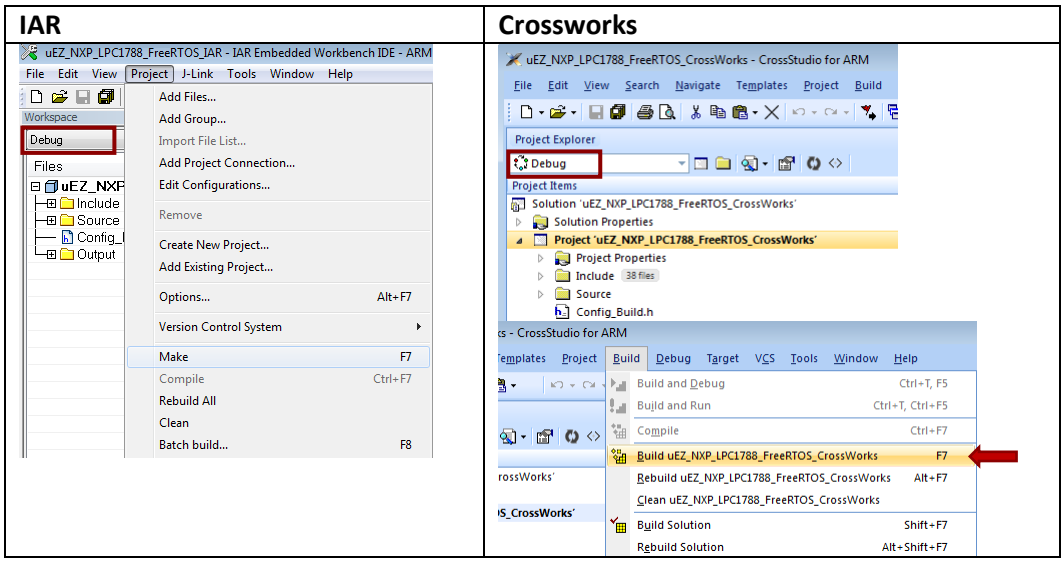
Figure 7: *Opening the µEZ Workspace*

NOTE: Note: Both IAR and Crossworks are shown in this guide, but only one should be used when following this guide. The same version of the IDE must be used both for building the library and final µEZ GUI executable.

The project should now open up in the IDE.

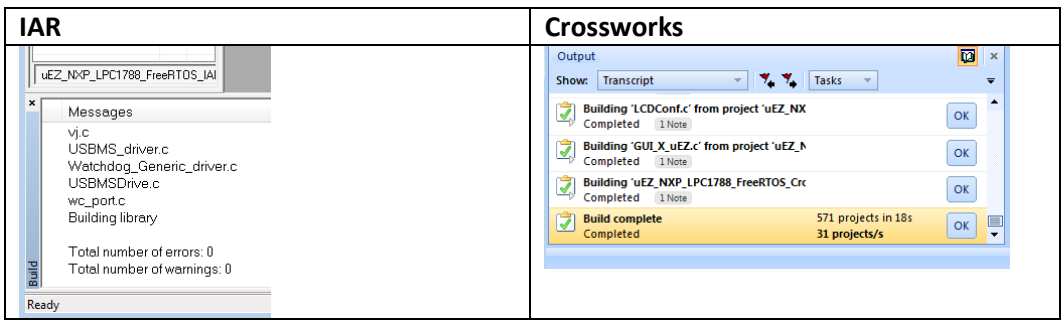
5. After the project opens:
   For IAR: Select the “Debug” Build from the workspace menu if not already selected.
   For Crossworks: Select the “Project Explorer” sub-window if not already selected.
6. Next building the project:
   For IAR: Select “Project > Make” or press F7 to build the µEZ library project.
   For Crossworks: Select “Build->Build…” or press F7 to build the µEZ library project.

Figure 8: *Building the µEZ library project*



7. After the build process is complete, there should be no errors.

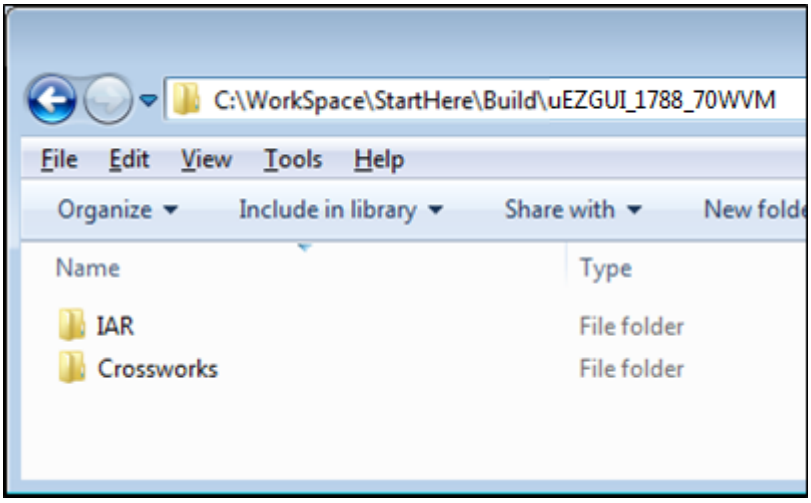
Figure 9: *Build Finished*



8. Close the IDE.

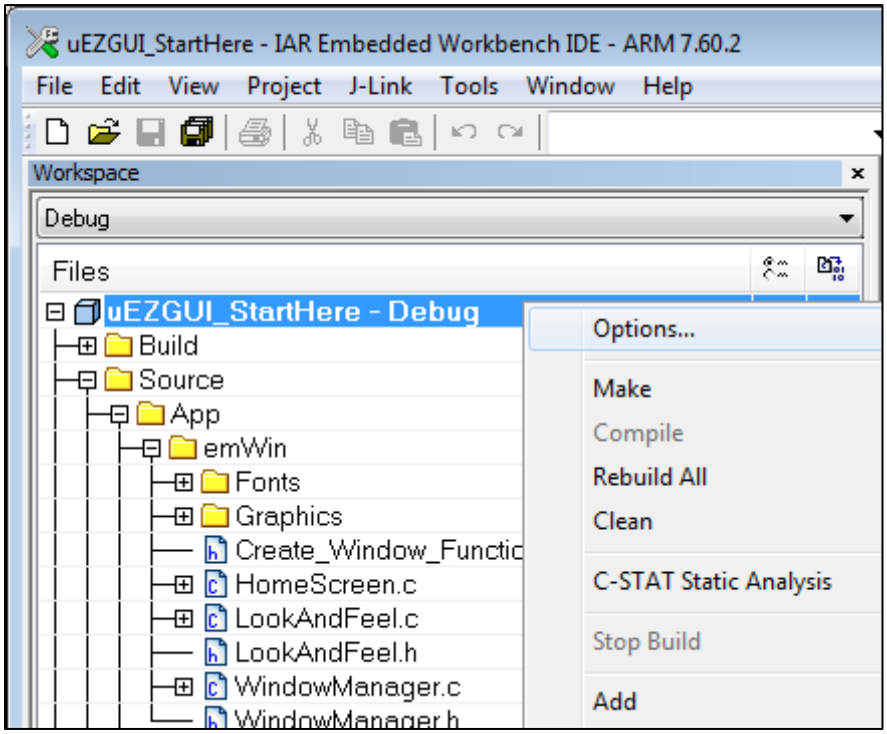
9. Next open up the actual start here project by navigating to the appropriate location:
   For IAR the workspace file is at
   StartHere\Build\uEZGUI_1788_70WVM\IAR\uEZGUI_StartHere.eww.
   For Crossworks the project file is at
   StartHere\Build\uEZGUI_1788_70WVM\Crossworks\uEZGUI_StartHere.hzp.

Figure 10: *Start Here Project Locations*



10. Verify that the Debug build is selected to match the library project.
11. If using IAR, proceed to step 12, otherwise proceed to step 15.
12. Right click the main project name in the workspace and select “Options”

Figure 11: *Debug Settings*



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

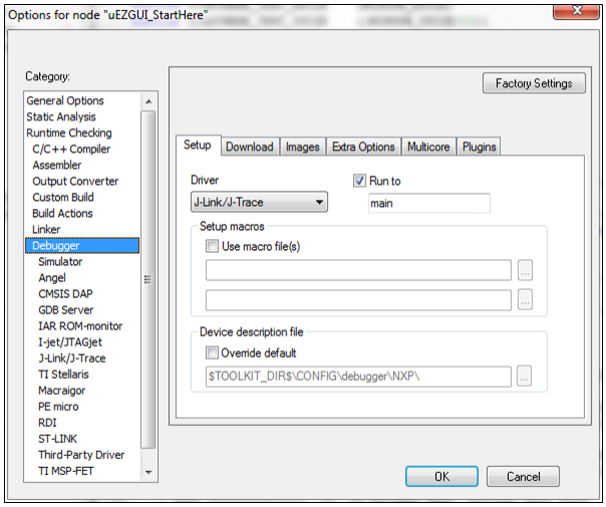


Figure 12: *Debug Settings*

14. Click the [OK] button to exit the dialog. Proceed to step 17
15. If using Crossworks, proceed to step 16, otherwise proceed to step 17.
16. Click "Target > Connect > SEGGER J-Link" from the drop down menu. This activates the J-Link probe.

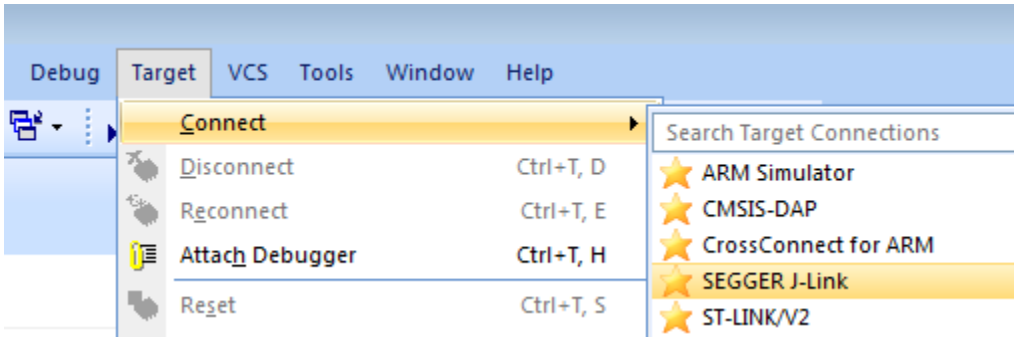


Figure 13: *Debug Settings*

17. Build (if needed), download and run the project following the respective IDE steps.
    For IAR: Press [Ctrl]+[D] or select "Project > Download and Debug" from the menu at the top.
    For Crossworks: Press [F5] or select "Debug > Go" from the menu at the top.
18. If the debugger has halted on a breakpoint, press [F5] to make the project run.

After the project is running on the µEZ GUI, a simple window is shown with three text bars.

Figure 14: *Initial project*

To see where the window is developed, examine the folder structure of the initial project.

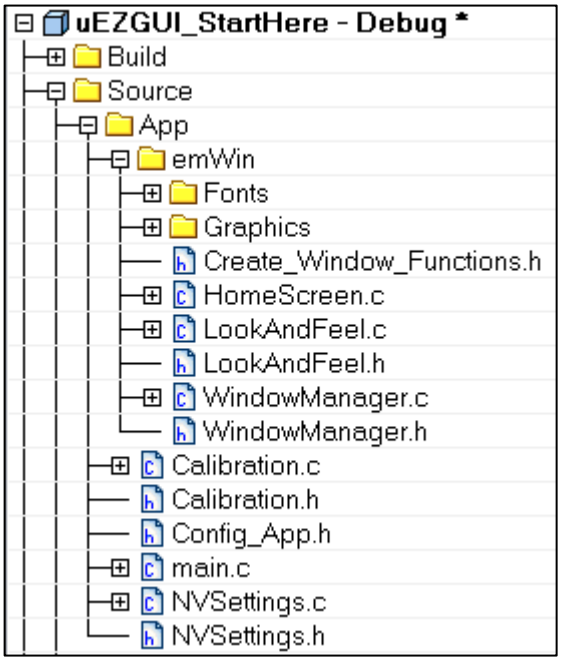


Figure 15: *File Hierarchy*

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.

C. Adding a 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 in the emWin directory.

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 definitions that define the ID, size, and position of each object within the window including the window itself.

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 depressed), 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.

1. The "Start Here" text can be replaced with a button by modifying the existing code. The definitions associated with the "Start Here" text are shown below:

```
/*---------------------------------------------------------------------------*
 * Constants:
 *---------------------------------------------------------------------------*/
#define ID_WINDOW               (GUI_ID_USER + 0x00)
#define ID_TITLE_TEXT           (GUI_ID_USER + 0x01)
#define ID_STARTHERE_TEXT       (GUI_ID_USER + 0x02)
#define ID_FDIINFO_TEXT         (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)

#if(UEZ_DEFAULT_LCD == LCD_RES_WVGA)
#define SPACING                 (10)
#define FONTSIZE                &FONT_LARGE
#else
#define SPACING                 (5)
#define FONTSIZE                &FONT_SMALL
#endif

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

#define STARTHERE_TEXT_XSIZE    (WINDOW_XSIZE)
#define STARTHERE_TEXT_YSIZE    ((WINDOW_YSIZE/10))
#define STARTHERE_TEXT_XPOS     (0)
#define STARTHERE_TEXT_YPOS     ((WINDOW_YSIZE/10)*4 + 10)
```

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. Replace the definitions with the names and values associated with the new button being created:

```
/*---------------------------------------------------------------------------*
 * Constants:
 *---------------------------------------------------------------------------*/
#define ID_WINDOW                (GUI_ID_USER + 0x00)
#define ID_TITLE_TEXT            (GUI_ID_USER + 0x01)
#define ID_MYBUTTON_BUTTON       (GUI_ID_USER + 0x02)
#define ID_FDIINFO_TEXT          (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)

#if(UEZ_DEFAULT_LCD == LCD_RES_WVGA)
#define SPACING                  (10)
#define FONTSIZE                 &FONT_LARGE
#else
#define SPACING                  (5)
#define FONTSIZE                 &FONT_SMALL
#endif

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

#define FDIINFO_TEXT_XSIZE       (WINDOW_XSIZE)
#define FDIINFO_TEXT_YSIZE       ((WINDOW_YSIZE/10))
#define FDIINFO_TEXT_XPOS        (0)
#define FDIINFO_TEXT_YPOS        ((WINDOW_YSIZE/10)*9 + 0)

#define MYBUTTON_BUTTON_XSIZE     (WINDOW_XSIZE/3 + 20)
#define MYBUTTON_BUTTON_YSIZE     ((WINDOW_YSIZE/5))
#define MYBUTTON_BUTTON_XPOS      (WINDOW_XSIZE/3 - 10)
#define MYBUTTON_BUTTON_YPOS      ((WINDOW_YSIZE/10)*4)
```

3. The two arrays under the section header “Local Data” still contain the information from the “Start Here” text. Each widget is represented in one of the sections within each array. Identify where the “Start Here” text is.

```
/*---------------------------------------------------------------------------*
 * 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},
    { TEXT_CreateIndirect, "", ID_STARTHERE_TEXT, STARTHERE_TEXT_XPOS,
        STARTHERE_TEXT_YPOS, STARTHERE_TEXT_XSIZE, STARTHERE_TEXT_YSIZE,
        TEXT_CF_HCENTER| TEXT_CF_VCENTER, 0, 0},
    { TEXT_CreateIndirect, "", ID_FDIINFO_TEXT, FDIINFO_TEXT_XPOS,
        FDIINFO_TEXT_YPOS, FDIINFO_TEXT_XSIZE, FDIINFO_TEXT_YSIZE,
        TEXT_CF_HCENTER| TEXT_CF_VCENTER, 0, 0},
};

/** Generic Mapping of Screen Layout*/
static T_LAFMapping HomeScreenMapping[] = {
    { ID_WINDOW, "", GUI_BLACK, GUI_WHITE, &FONT_SMALL, LAFSetupWindow, 0},
    { ID_TITLE_TEXT, "Welcome To uEZ GUI!", GUI_BLACK, GUI_WHITE, &FONT_LARGE,
        LAFSetupText, 0},
    { ID_STARTHERE_TEXT, "\"START HERE\"", GUI_BLACK, GUI_WHITE, &FONT_LARGE,
        LAFSetupText, 0},
    { ID_FDIINFO_TEXT,
        "Future Designs, Inc. 2016                    www.TeamFDI.com",
        GUI_BLACK, GUI_WHITE, FONTSIZE, LAFSetupText, 0},
    {0},
};
```

4. Replace the “Start Here” text field’s information with the following button information:

```
/*---------------------------------------------------------------------------*
 * 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},
    { BUTTON_CreateIndirect, "", ID_MYBUTTON_BUTTON, MYBUTTON_BUTTON_XPOS,
        MYBUTTON_BUTTON_YPOS, MYBUTTON_BUTTON_XSIZE, MYBUTTON_BUTTON_YSIZE,
        TEXT_CF_HCENTER| TEXT_CF_VCENTER, 0, 0},
    { TEXT_CreateIndirect, "", ID_FDIINFO_TEXT, FDIINFO_TEXT_XPOS,
        FDIINFO_TEXT_YPOS, FDIINFO_TEXT_XSIZE, FDIINFO_TEXT_YSIZE,
        TEXT_CF_HCENTER| TEXT_CF_VCENTER, 0, 0},
};
```

```
/** Generic Mapping of Screen Layout*/
static T_LAFMapping HomeScreenMapping[] = {
    { ID_WINDOW, "", GUI_BLACK, GUI_WHITE, &FONT_SMALL, LAFSetupWindow, 0},
    { ID_TITLE_TEXT, "My Main Screen", GUI_BLACK, GUI_WHITE, &FONT_LARGE,
        LAFSetupText, 0},
    { ID_MYBUTTON_BUTTON, "Temperature", GUI_GRAY, GUI_BLACK, &FONT_LARGE,
        LAFSetupButton, 0},
    { ID_FDIINFO_TEXT,
        "Future Designs, Inc. 2016                    www.TeamFDI.com",
        GUI_BLACK, GUI_WHITE, FONTSIZE, LAFSetupText, 0},
    {0},
};
```

5. Build, download, debug, and run the updated project to display the created button.
   For IAR: Press [Ctrl]+[D].
   For Crossworks: 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.

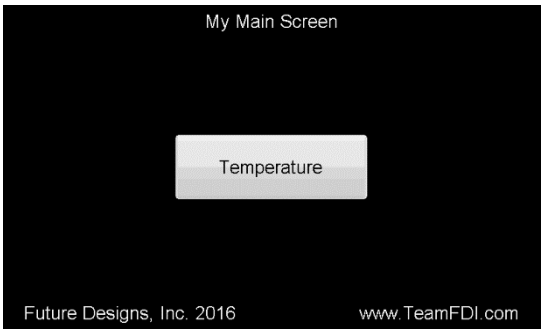


Figure 16: *Button Added*

D. 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
   For IAR: Select “Open Containing Folder…”.
   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.
   For IAR: Select “Add > Add Files…”.
   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).

```
/*---------------------------------------------------------------------------*
 * Local Data:
 *---------------------------------------------------------------------------*/
/** Structure to hold all of the widgets used in this dialog*/
static const GUI_WIDGET_CREATE_INFO _aHomeScreenDialog[] = {
    //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,
```

```
/*---------------------------------------------------------------------------*
 * Local Data:
 *---------------------------------------------------------------------------*/
/** Structure to hold all of the widgets used in this dialog*/
static const GUI_WIDGET_CREATE_INFO _aSecondScreenDialog[] = {
    //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,
```

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

```
/*---------------------------------------------------------------------------*
 * Constants:
 *---------------------------------------------------------------------------*/
#define ID_WINDOW               (GUI_ID_USER + 0x00)
#define ID_TITLE_TEXT           (GUI_ID_USER + 0x01)
#define ID_MYBUTTON_BUTTON      (GUI_ID_USER + 0x02)
#define ID_FDIINFO_TEXT         (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)

#if(UEZ_DEFAULT_LCD == LCD_RES_WVGA)
#define SPACING                 (10)
#define FONTSIZE                &FONT_LARGE
#else
#define SPACING                 (5)
#define FONTSIZE                &FONT_SMALL
#endif

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

#define FDIINFO_TEXT_XSIZE      (WINDOW_XSIZE)
#define FDIINFO_TEXT_YSIZE      ((WINDOW_YSIZE/10))
#define FDIINFO_TEXT_XPOS       (0)
#define FDIINFO_TEXT_YPOS       ((WINDOW_YSIZE/10)*9 + 0)

#define MYBUTTON_BUTTON_XSIZE    (WINDOW_XSIZE/3 + 20)
#define MYBUTTON_BUTTON_YSIZE    ((WINDOW_YSIZE/5))
#define MYBUTTON_BUTTON_XPOS     (WINDOW_XSIZE/3 - 10)
#define MYBUTTON_BUTTON_YPOS     ((WINDOW_YSIZE/10)*4)
```

8. 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 _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},
    { BUTTON_CreateIndirect, "", ID_MYBUTTON_BUTTON, MYBUTTON_BUTTON_XPOS,
        MYBUTTON_BUTTON_YPOS, MYBUTTON_BUTTON_XSIZE, MYBUTTON_BUTTON_YSIZE,
        TEXT_CF_HCENTER| TEXT_CF_VCENTER, 0, 0},
    { TEXT_CreateIndirect, "", ID_FDIINFO_TEXT, FDIINFO_TEXT_XPOS,
        FDIINFO_TEXT_YPOS, FDIINFO_TEXT_XSIZE, FDIINFO_TEXT_YSIZE,
        TEXT_CF_HCENTER| TEXT_CF_VCENTER, 0, 0},
};

/** Generic Mapping of Screen Layout*/
static T_LAFMapping HomeScreenMapping[] = {
    { ID_WINDOW, "", GUI_BLACK, GUI_WHITE, &FONT_SMALL, LAFSetupWindow, 0},
    { ID_TITLE_TEXT, "My Main Screen", GUI_BLACK, GUI_WHITE, &FONT_LARGE,
        LAFSetupText, 0},
    { ID_MYBUTTON_BUTTON, "Temperature", GUI_GRAY, GUI_BLACK, &FONT_LARGE,
        LAFSetupButton, 0},
    { ID_FDIINFO_TEXT,
        "Future Designs, Inc. 2016                    www.TeamFDI.com",
        GUI_BLACK, GUI_WHITE, FONTSIZE, LAFSetupText, 0},
    {0},
};
```

9. 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:

```
void WindowManager_Create_All_Active_Windows(void)
{
    static TBool iHaveRun = EFalse;

    if(!iHaveRun){
        G_SystemWindows[HOME_SCREEN] = HomeScreen_Create();
        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 SECOND_SCREEN                   (1)
```

E. 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 the first widget definition matrix. (the "Temperature" 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_TITLE_TEXT, "My Main Screen", GUI_BLACK, GUI_WHITE,
                            &FONT_LARGE, LAFSetupText, 0},
    { ID_MYBUTTON_BUTTON, "Temperature", GUI_GRAY, GUI_BLACK,
                                &FONT_LARGE, LAFSetupButton,
      (TBool (*)(WM_MESSAGE *, int, int))IHandleSecondScreen},
    { ID_FDIINFO_TEXT,
    "Future Designs, Inc. 2016                          www.TeamFDI.com",
                        GUI_BLACK, GUI_WHITE, FONTSIZE, LAFSetupText, 0},
    {0},
};
```

2. Declare the callback function's prototype between the "Global Data" and "Local Data" sections of the HomeScreen.c file. In addition add the "G_WhichWindow" variable in the "Global Data" section.

```
/*---------------------------------------------------------------------------*
 * Global Data:
 *---------------------------------------------------------------------------*/
static TBool G_Active = EFalse; // Active Flag, tell the dialog when it
receives messages that the screen is in the foreground
TInt32 G_WhichWindow; // Variable to hold the ID of the active window

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


/*---------------------------------------------------------------------------*
 * Local Data:
 *---------------------------------------------------------------------------*/
```

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 the temperature 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 is blank, pressing the button will display a black screen.

F. Interfacing with the Onboard Temperature Sensor

In the following steps of this guide we will demonstrate how to interface with the uEZ GUI hardware using the on-board temperature sensor and how to update the temperature on the screen. We will start by returning to the SecondScreen.c file in the IDE, then creating two new text fields. One text field will be a title for the temperature reading, and the other will show the onboard temperature sensor value. Finally, we will add a back button to return to the home screen.

1. First, 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.

```
#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 fields' position and size.

```
#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/5))
#define TEMP_TEXT_XPOS           (WINDOW_XSIZE/3)
#define TEMP_TEXT_YPOS           ((WINDOW_YSIZE/10)*4)
```

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[] = {
    //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},
};

/** 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},
    {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;
    G_WhichWindow = pMsg->hWin;
```

G. 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 I^2C 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.

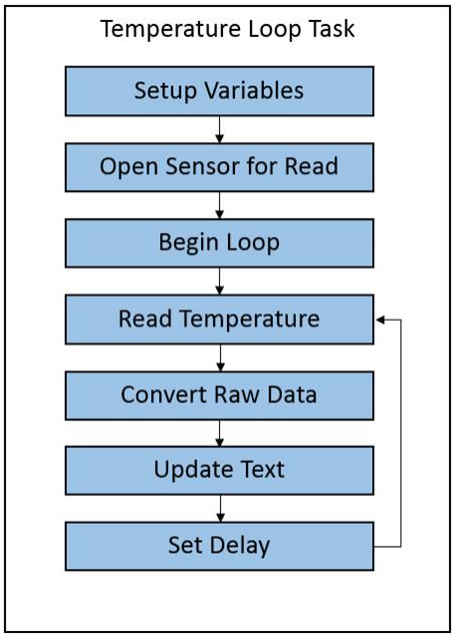


Figure 17: *Temp Loop Task Flow*

To save time, the thread files MyTask.c and MyTask.h are included with the Start Here \Source\App\ directory. In the next steps we will add them into the App directory of the project as done before with the SecondScreen.c. Shown below is an excerpt of the temperature loop task code from MyTask.c

```
#include <uEZ.h>
#include <uEZTemperature.h>
#include <uEZGPIO.h>
#include <uEZProcessor.h>
#include "MyTask.h"
#include <stdio.h>

extern void UpdateTemp(char *myString); // Access to function in
SecondScreen.c

TUInt32 TemperatureLoopTask(T_uezTask aMyTask, void *aParams) {
  char myString[10] = ""; // String to hold temperature
  TInt32 i, f; // Variables for conversion
  T_uezDevice temp; // uEZ device variable
  TInt32 Temperature; // Variable to hold temperature reading

  UEZTemperatureOpen("Temp0", &temp); // Open the on board device

  while(1) {
    UEZTemperatureRead(temp, &Temperature); // Read the temperature

    // Convert to integer and decimal
    i = Temperature >> 16;
    f = ((((TUInt32)Temperature) & 0xFFFF) * 10) >> 16;
    sprintf(myString, "%02d.%01d *C", i, f); // Print to text array

    UpdateTemp(myString); // Update the temperature text on the screen

    UEZTaskDelay(250); // Wait 1/4 seconds before reading again
  }
}
```

All the tasks that will run in a project are called within main.c. In the following tasks we will update main.c to create the new task.

1. Open main.c
2. Add the new #include shown below.

```
#include "MyTask.h"
```

3. Within the function MainTask(), find the existing UEZTaskCreate() function calls.
4. Add a new UEZTaskCreate() function call for the temperature monitoring thread.

```
void MainTask(void) {
  // Run the teperature reading task located in My.Task.c
  UEZTaskCreate(TemperatureLoopTask, "Temperature", 1024,
                (void *)0, UEZ_PRIORITY_NORMAL, 0);
  //Start emWin
  UEZTaskCreate(GUIInterfaceTask, "GUIInterface", (4 * 1024),
                (void *)0, UEZ_PRIORITY_NORMAL, 0);

  while(1) {// Loop forever so that the main task never exits
    UEZTaskDelay(1000);
  }
}
```

5. Use the appropriate steps to Build, download, debug, and run the updated project to verify that there are no errors. The temperature on the second screen should display an accurate reading.

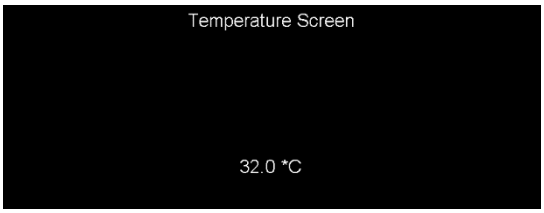


Figure 18: *Second Screen*

A heartbeat task that demonstrates GPIO port functionality using the µEZ GUI's onboard LED is also included in the project's MyTask.c file. It may be implemented using the same instructions provided for the temperature task by calling UEZTaskCreate.

```
TUInt32 HeartbeatTask(T_uezTask aMyTask, void *aParams)
{
    UEZGPIOSetMux(LEDPIN, 0); // Set the Mux
    UEZGPIOOutput(LEDPIN); // Set the pin to output

    while(1) {
        UEZGPIOSet(LEDPIN); // On Board LED High
        UEZTaskDelay(LEDONTIME); // Delay
        UEZGPIOClear(LEDPIN); // On Board LED Low
        UEZTaskDelay(LEDOFFTIME); // Delay
    }
}
```

H. 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)   // ID

#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/3)
#define TEMP_TEXT_YSIZE          ((WINDOW_YSIZE/5))
#define TEMP_TEXT_XPOS           (WINDOW_XSIZE/3)
#define TEMP_TEXT_YPOS           ((WINDOW_YSIZE/10)*4)

#define BACK_BUTTON_XSIZE        (WINDOW_XSIZE/5) // Properties
#define BACK_BUTTON_YSIZE        ((WINDOW_YSIZE/6))
#define BACK_BUTTON_XPOS         (0)
#define BACK_BUTTON_YPOS         ((WINDOW_YSIZE/6)*5)
```

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,
        TEXT_CF_HCENTER| TEXT_CF_VCENTER, 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_GRAY, GUI_BLACK, &FONT_LARGE,
        LAFSetupButton, (TBool (*)(WM_MESSAGE *, int, int)) IHandleBackButton},
    {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 arryas.

```
/*---------------------------------------------------------------------------*
 * 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. Use the appropriate steps to Build, download, debug, and run the updated project to test the new functionality.

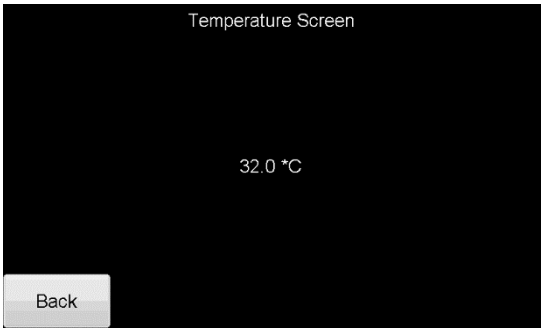


Figure 19: *Final Screen*

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.

5. 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. In the root directory of the SD card, the folder titled "OOB Demo" contains the flash binary and a J-Link script that will flash the original out-of-box to the connected µEZ GUI.

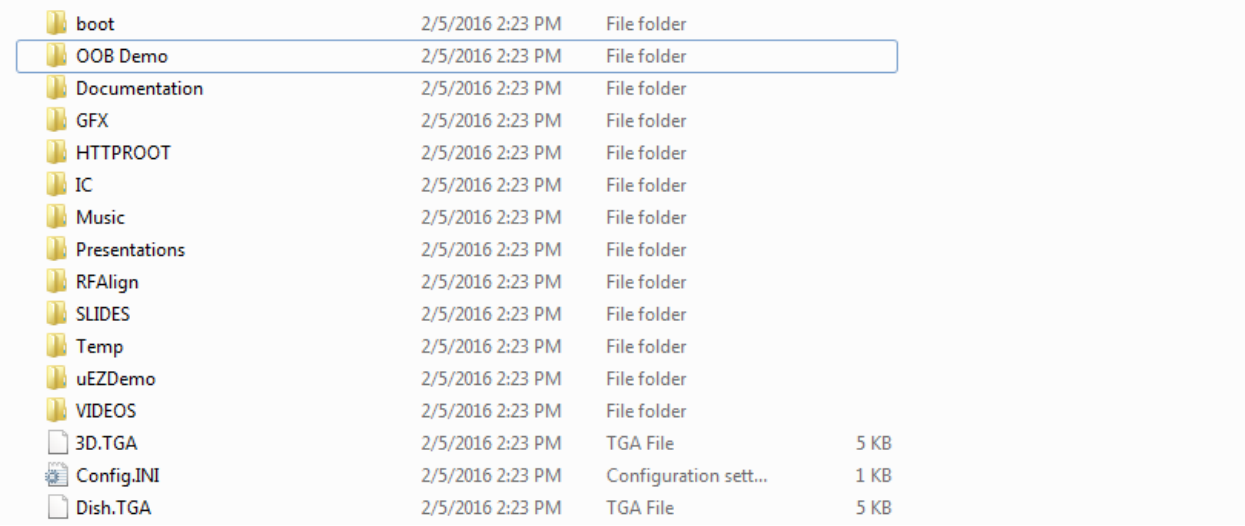


Figure 20: *OOB Demo Directory*

4. Connect the PC to the µEZ GUI via the J-Link debugger probe and double-click the "Install OOB.jlink" script. This reinstalls the OOB demo onto the µEZ GUI.

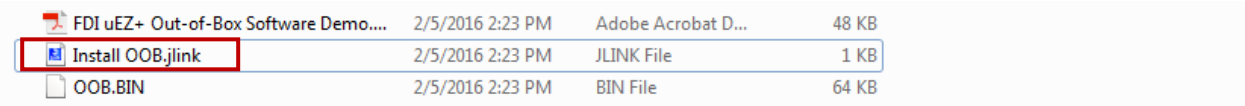


Figure 21: *OOB Demo Files*

6. Next Steps

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

- Create & Compile a Project with µEZ Project Maker: http://tinyurl.com/hopv57u
- 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

7. Website and Support

Documentation:

- µEZ Library Online Documentation ..... http://www.TeamFDI.com/uez/docs/
- emWin Documentation..... https://www.segger.com/emWin.html
- emWin 5.30 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-1788-70WVM..... http://tinyurl.com/hcqhm2b
- µEZ Source w/ Project Maker (v2.08 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

Общество с ограниченной ответственностью «МосЧип» ИНН 7719860671 / КПП 771901001
Адрес: 105318, г.Москва, ул.Щербаковская д.3, офис 1107

Данный компонент на территории Российской Федерации

Вы можете приобрести в компании MosChip.

Для оперативного оформления запроса Вам необходимо перейти по данной ссылке:

http://moschip.ru/get-element

Вы можете разместить у нас заказ для любого Вашего проекта, будь то серийное производство или разработка единичного прибора.

В нашем ассортименте представлены ведущие мировые производители активных и пассивных электронных компонентов.

Нашей специализацией является поставка электронной компонентной базы двойного назначения, продукции таких производителей как XILINX, Intel (ex.ALTERA), Vicor, Microchip, Texas Instruments, Analog Devices, Mini-Circuits, Amphenol, Glenair.

Сотрудничество с глобальными дистрибьюторами электронных компонентов, предоставляет возможность заказывать и получать с международных складов практически любой перечень компонентов в оптимальные для Вас сроки.

На всех этапах разработки и производства наши партнеры могут получить квалифицированную поддержку опытных инженеров.

Система менеджмента качества компании отвечает требованиям в соответствии с ГОСТ Р ИСО 9001, ГОСТ РВ 0015-002 и ЭС РД 009

Офис по работе с юридическими лицами:

105318, г.Москва, ул.Щербаковская д.3, офис 1107, 1118, ДЦ «Щербаковский»

Телефон: +7 495 668-12-70 (многоканальный)

Факс: +7 495 668-12-70 (доб.304)

E-mail: info@moschip.ru

Skype отдела продаж:
moschip.ru
moschip.ru_4
moschip.ru_6
moschip.ru_9