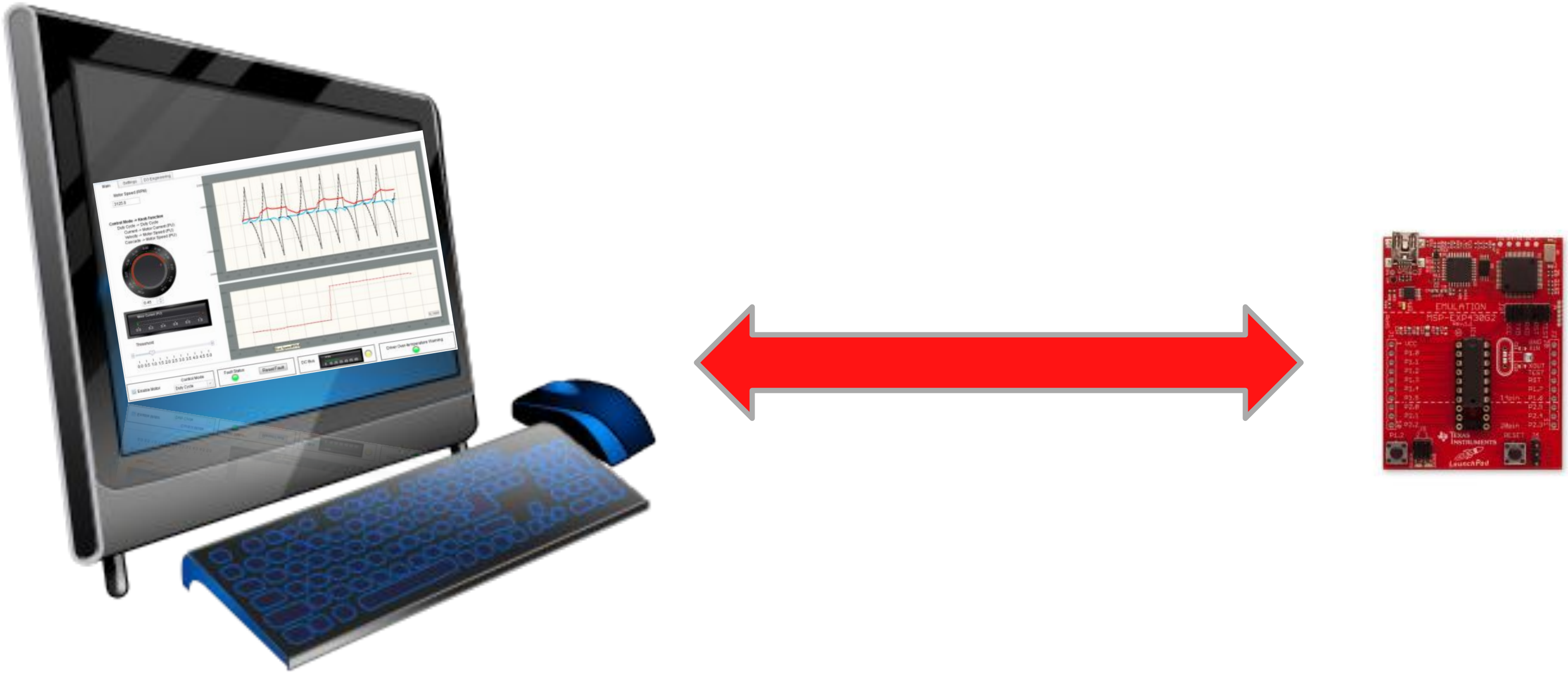
GUI Composer

Texas Instruments

Embedded Development Tools

GUI Composer: See & Control

- Create GUI applications that provide:
  - Visibility into what is happening in the target application
  - The ability to control target variables

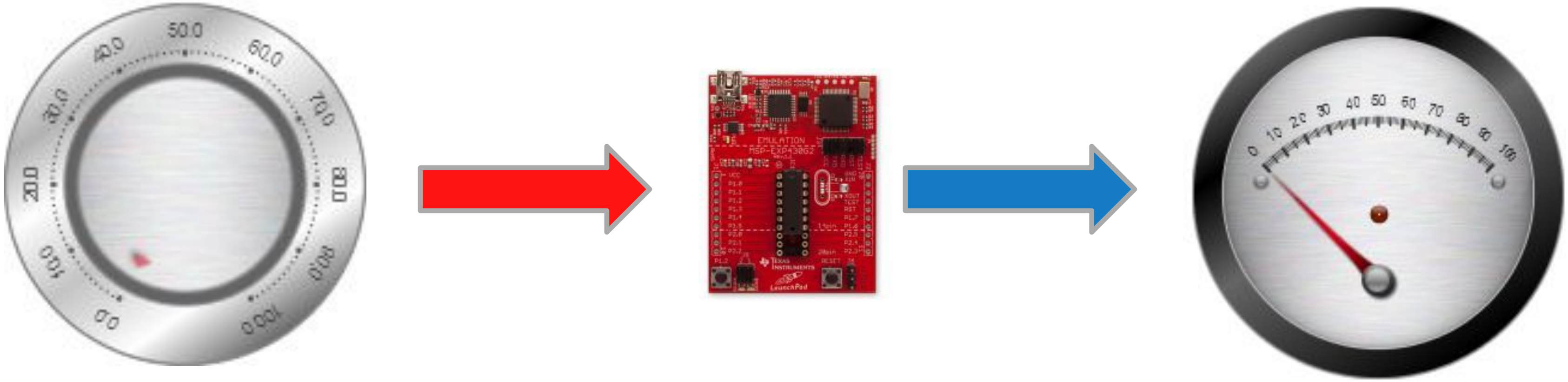


When to use GUI Composer

- While debugging
  - Create simple displays that allow you to quickly see target status in a meaningful way
- Standalone applications
  - Create GUI applications that allow you to see and control your application outside of the CCS environment
  - Great for demonstrations

GUIs are Comprised of Widgets

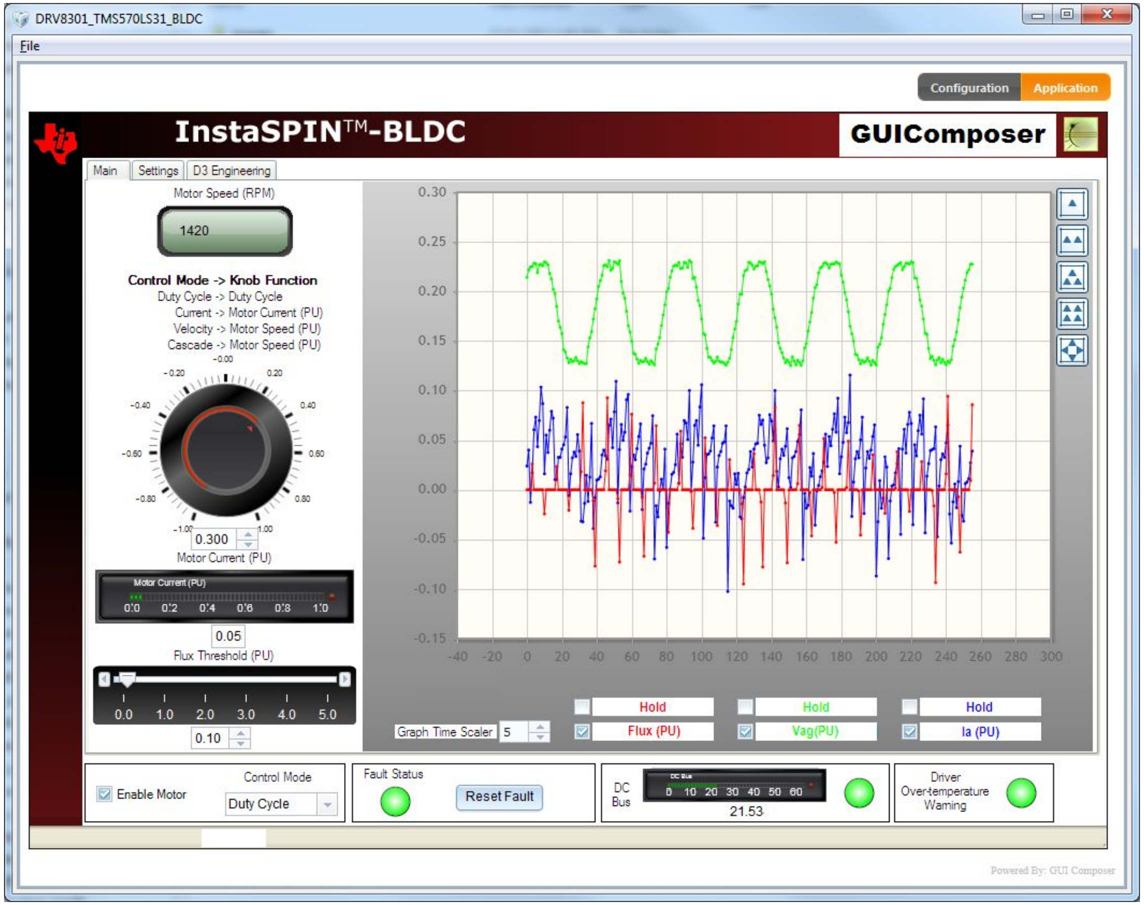
- GUI Composer Applications are made up of HTML5 widgets
- Control widgets (dials, edit boxes…)
  - Let you adjust the value of target variables
- Display widgets (meters, graphs, lights…)
  - Show the value of target variables



Example Application: InstaSpin

- GUI application for demonstrating motor control development kit

- Widgets
  - Knobs
  - Graphs
  - Meters
  - Status lights
  - Sliders
  - Edit boxes



Motor Speed (RPM) 1420

0.300 Motor Current (PU)

0.05 Flux Threshold (PU)

0.10

Graph Time Scaler 5

☑ Enable Motor

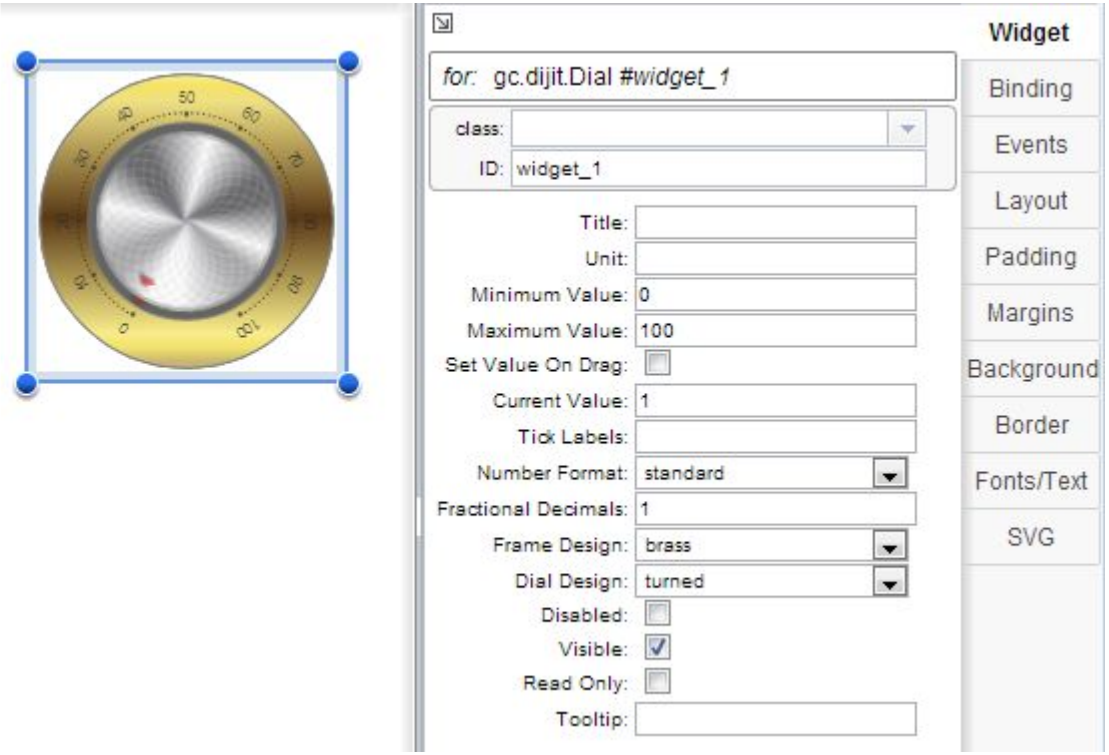
Control Mode: Duty Cycle

DC Bus 21.53

Customize how Widgets Look

- Most widgets can be customized to get the exact appearance that you want

- Adjust:
  - Design
  - Color
  - Labels
  - Number format
  - …



for: gc.dijit.Dial #*widget_1*

class:

ID: widget_1

Title:

Unit:

Minimum Value: 0

Maximum Value: 100

Set Value On Drag: □

Current Value: 1

Tick Labels:

Number Format: standard

Fractional Decimals: 1

Frame Design: brass

Dial Design: turned

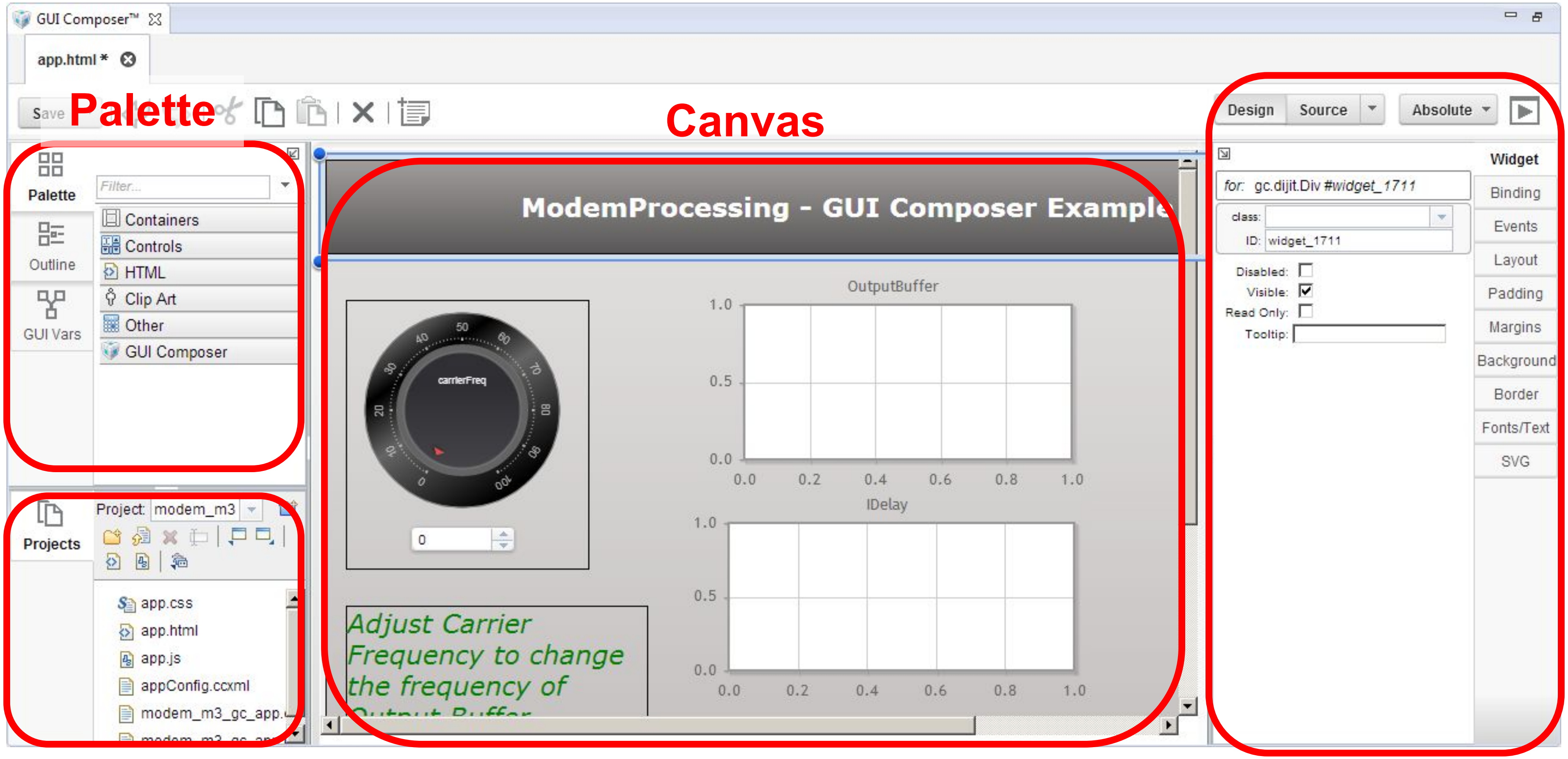
Disabled: □

Visible: ☑

Read Only: □

Tooltip:

GUI Composer Builder



Project: modem_m3

for: gc.dijit.Div #widget_1711

class:

ID: widget_1711

Disabled: □

Visible: ☑

Read Only: □

Tooltip:

Adjust Carrier Frequency to change the frequency of Output Buffer

Palette

Canvas

Projects

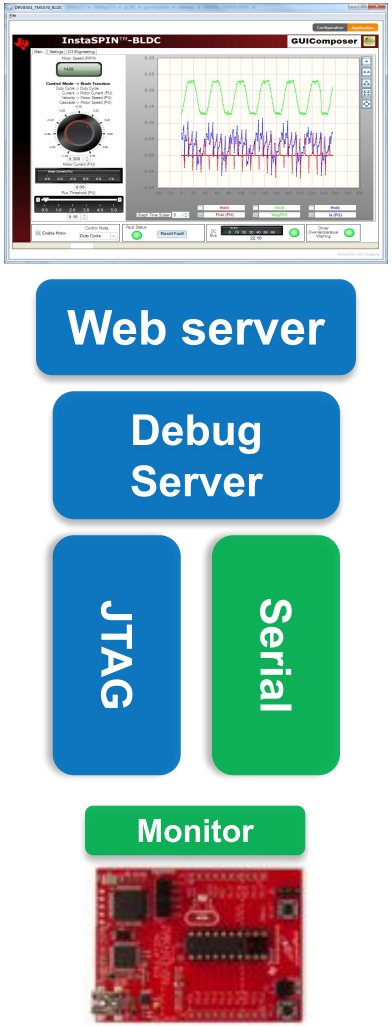
Properties

Application Models

- Program Model
  - CCS Debugger is used to translate symbols to addresses
  - Writes to the target are done via JTAG or via a command to a target monitor
  - Data is requested by the GUI application
  - JTAG or Serial connections are used
- Streaming Model
  - Data is streamed from the target application up to the host
  - Serial or Ethernet connections are used

Program Model

- Symbol translation
  - Symbols are translated to addresses by the CCS debugger
- Data display
  - GUI application requests data and it is read over JTAG or via request to a monitor service running on the device
- Data writes
  - Write is performed via JTAG or can be passed to a monitor service running on the device



Web server

Debug Server

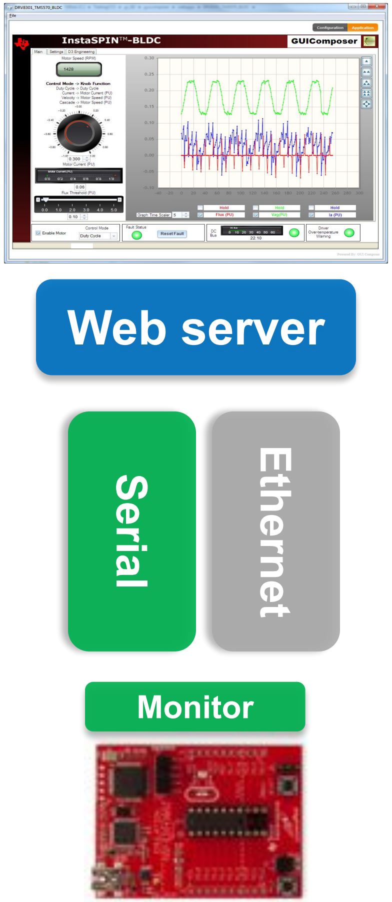
JTAG

Serial

Monitor

Streaming Model

- Data display
  - Target application pushes data up to the GUI for display
- Data writes
  - Data is passed to a monitor service running on the device



Adjusting Data Format

- It is possible to adjust the value of data that is displayed
  - Data may be stored in 1 unit of measure on the device but you wish to display it in the GUI in another
- Pre-processing Function
  - Takes the value from the target and adjusts it for display
- Post-processing Function
  - Takes the value entered in the widget and adjusts it prior to writing it to the device

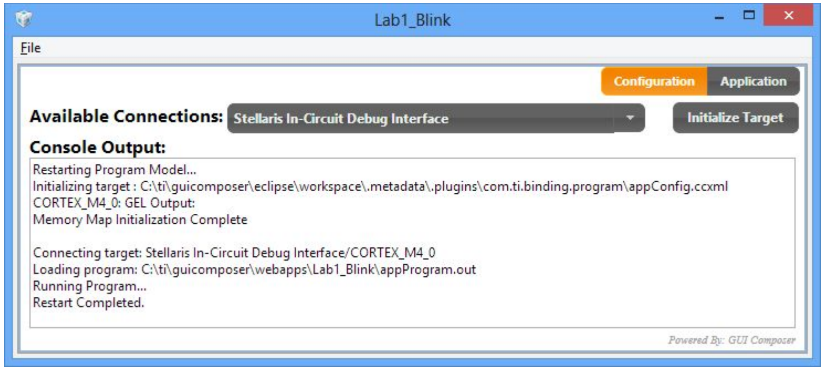
Types of Applications

- CCS Plug-in
  - Feature within CCS
  - Available from View menu within CCS

- Standalone application
  - Can be run without CCS
  - Requires the GUI Composer Runtime to be installed
    - http://processors.wiki.ti.com/index.php/GUI_Composer
  - Used for larger applications

Standalone Application

- Configuration View
  - Appears when the standalone app starts up and shows the progress of the connection to the device being initialized, program loaded/flashed and then run
  - Can be accessed later if you need to re-initialize the target

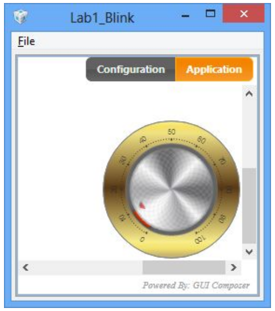


Available Connections: Stellaris In-Circuit Debug Interface

Console Output:

```
Restarting Program Model...
Initializing target : C:\ti\guicomposer\eclipse\workspace\.metadata\.plugins\com.ti.binding.program\appConfig.ccxml
CORTEX_M4_0: GEL Output:
Memory Map Initialization Complete

Connecting target: Stellaris In-Circuit Debug Interface/CORTEX_M4_0
Loading program: C:\ti\guicomposer\webapps\Lab1_Blink\appProgram.out
Running Program...
Restart Completed.
```

- Application View
  - Switches to this view when initialization is complete
  - Shows all your widgets



GUI Composer Runtime

- GUI Composer applications need access to services (target read/write…)
  - These services can be provided by CCS, or if CCS is not present then they are provided by the GUI Composer Runtime
- Subset of CCS Functionality
- Can be bundled with GUI Composer applications

Lab Requirements

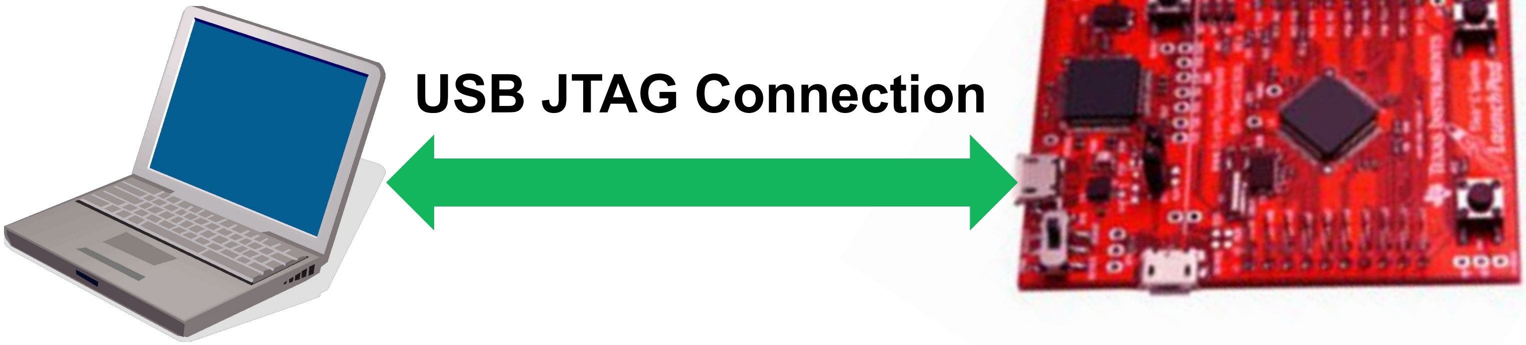
- Software:
  - Code Composer Studio v6.1.1.00022
  - GUI Composer (add-on in CCS)
  - TivaWare for C series 2.1.1.71
  - TI-RTOS for TivaC the 2.14.00.10 (only required for Lab2)
  - GUI Composer Runtime v6.1.1
- Hardware:
  - Tiva C TM4C123GXL Launchpad
  - 1 micro USB cable (included with LaunchPad kit)

  Note: Labs should also run on TM4C1294XL Launchpad as long as appropriate example projects for that device are used
- See Installation Instructions in next slide

Installation Instructions

- Download and install Code Composer Studio v6.1.1.00022 from http://processors.wiki.ti.com/index.php/Download_CCS
- Start CCS and click on App Center in the Getting Started view or open App Center from menu View->CCS App Center
- In App Center:
  - Under Code Composer Studio Add-ons, select the following and install
    - GUI Composer
    - TI-RTOS for TivaC (only required for Lab2)
  - Under Code Composer Studio Standalone Software, select TivaWare, click on Download, then download and install to c:\ti
- Download GUI Composer Runtime v6.1.1 from http://processors.wiki.ti.com/index.php/Category:GUI_Composer#GUI_Composer_Downloads and install to c:\ti

Tiva C LaunchPad: Hardware Setup



Agenda ...

LAB 1: JTAG TRANSPORT

LAB1A: CREATE AND USE DIAL WIDGET
LAB1B: CREATE AND USE MORE WIDGETS

LAB conventions

- Lab steps are numbered for easier reference
  1. …
  2. …

- Explanations, notes, warnings are written in blue
  - Warnings are shown with ⚠
  - Information is marked with ⓘ
  - Tips and answers are marked with 💡
  - Questions are marked with ❓

JTAG Transport: Exercise Summary

- **Key Objectives**
  - Create a GUI that will create different widgets for controlling and visualizing target variables
  - Debug the basic blinky program and then a modified version of the blinky program
  - Run GUI composer to view and control the application
- **Tools and Concepts Covered**
  - JTAG transport connection
  - GUI Builder tool
  - Variable binding
  - Target variable modification
  - Target variable display
  - Pre/post processing functions

LAB1A: CREATE AND USE DIAL WIDGET

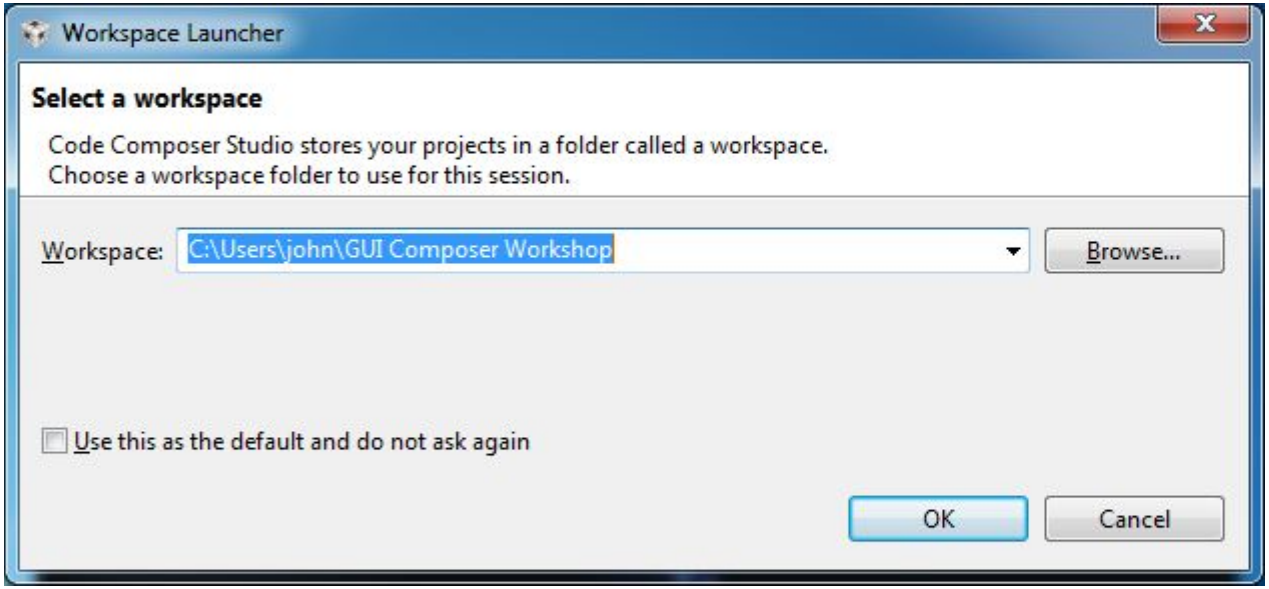
15 MINUTES

Open your lab materials and complete LAB 1A

Launch CCS

1. Double click on the Code Composer Studio desktop icon

2. Specify “GUI Composer Workshop” as the workspace



Select a workspace

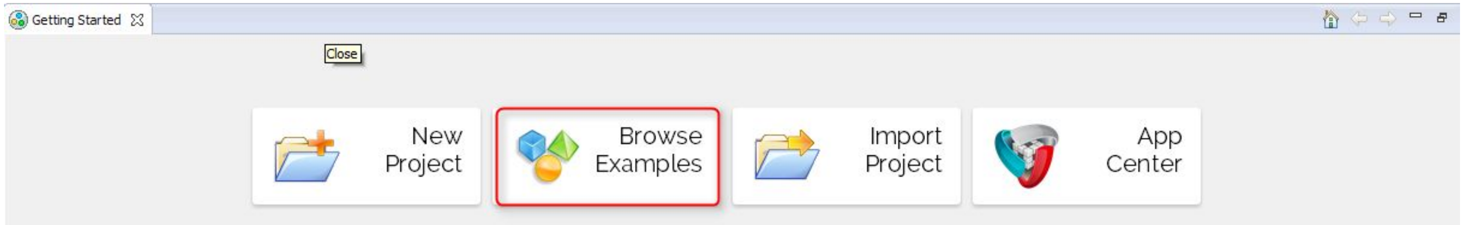
Code Composer Studio stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: C:\Users\john\GUI Composer Workshop

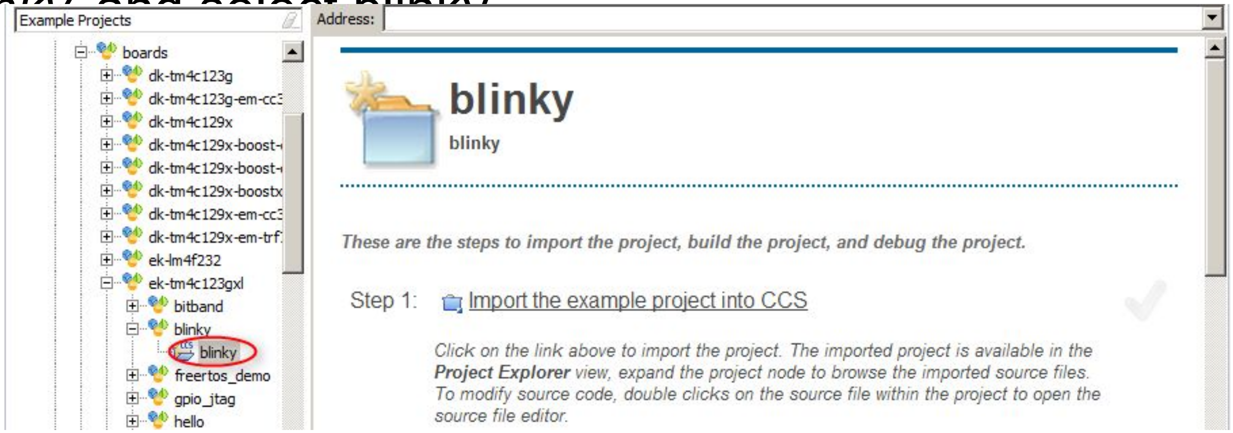
□ Use this as the default and do not ask again

Import ‘blinky’ Project

1. In the “Getting Started” page, click on *Browse Examples*



2. Expand *TivaWare_C_Series-2.1.0.12573->examples->boards->ek-tm4c123gxl ->blinky, and select blinky*



blinky

blinky

These are the steps to import the project, build the project, and debug the project.

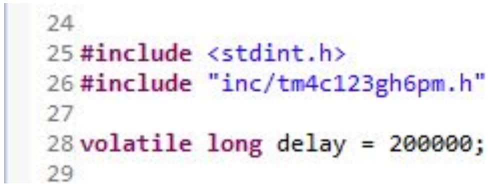
Step 1: Import the example project into CCS

*Click on the link above to import the project. The imported project is available in the **Project Explorer** view, expand the project node to browse the imported source files. To modify source code, double clicks on the source file within the project to open the source file editor.*

3. In the right-pane, click on Step1 to **Import the example project into CCS**

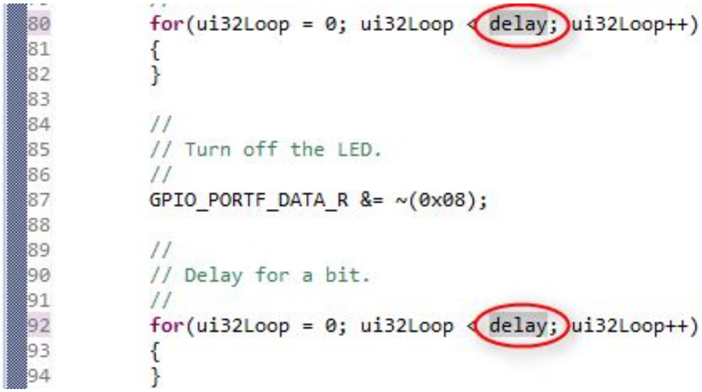
Modify the Code

1. Expand the project in the *Project Explorer* view to view its contents
2. Double-click on **blinky.c** to open it
3. Add a global variable called *delay* after the include statements as shown here:



```

#include <stdint.h>
#include "inc/tm4c123gh6pm.h"

volatile long delay = 200000;

```

4. Change 200000 in both ‘for’ loops to use the variable *delay* instead

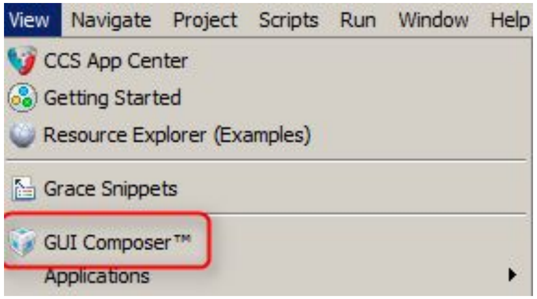


```
        for(ui32Loop = 0; ui32Loop < delay; ui32Loop++)
        {
        }

        //
        // Turn off the LED.
        //
        GPIO_PORTF_DATA_R &= ~(0x08);

        //
        // Delay for a bit.
        //
        for(ui32Loop = 0; ui32Loop < delay; ui32Loop++)
        {
        }
```

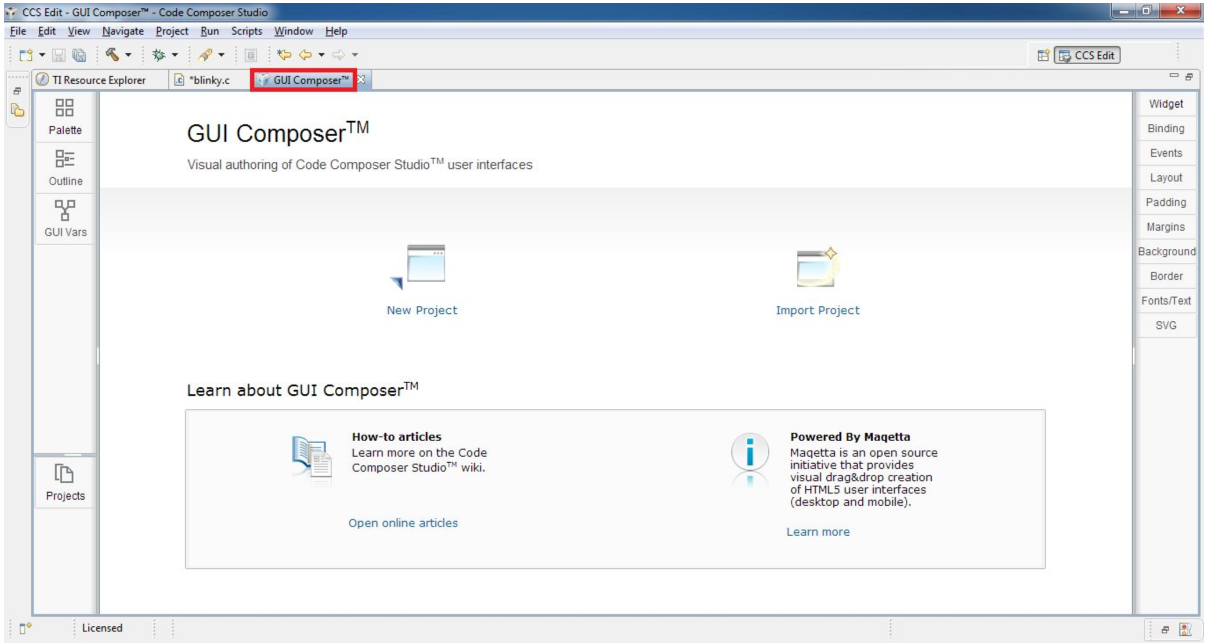
5. Click the save button 💾 to save the file blinky.c

Open GUI Composer

1. Go to menu *View -> GUI Composer*



2. Double-click on the GUI Composer tab to maximize it



GUI Composer^{TM}

Visual authoring of Code Composer Studio^{TM} user interfaces

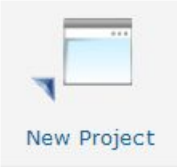
Learn about GUI Composer^{TM}

How-to articles
Learn more on the Code Composer Studio™ wiki.

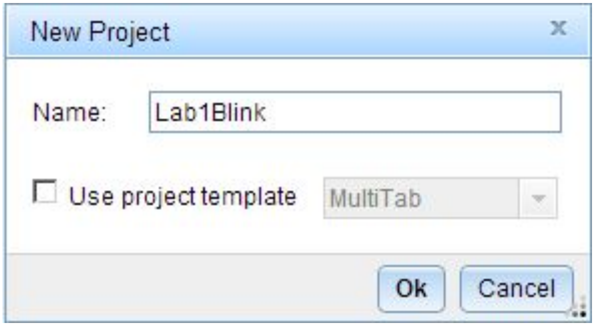
Powered By Maqetta
Maqetta is an open source initiative that provides visual drag&drop creation of HTML5 user interfaces (desktop and mobile).

Create a New GUI Composer Project

1. Click on the *New Project* button



2. Name the project **Lab1Blink**


New Project

Name: Lab1Blink

☐ Use project template MultiTab

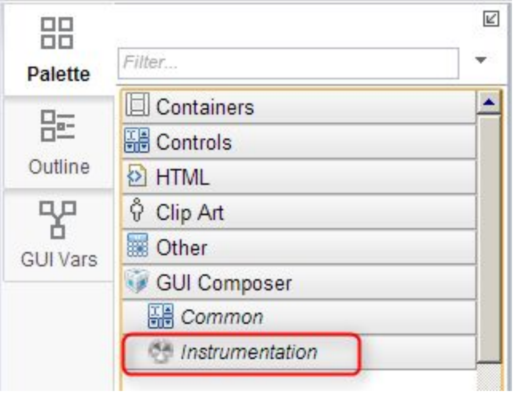
3. Click Ok

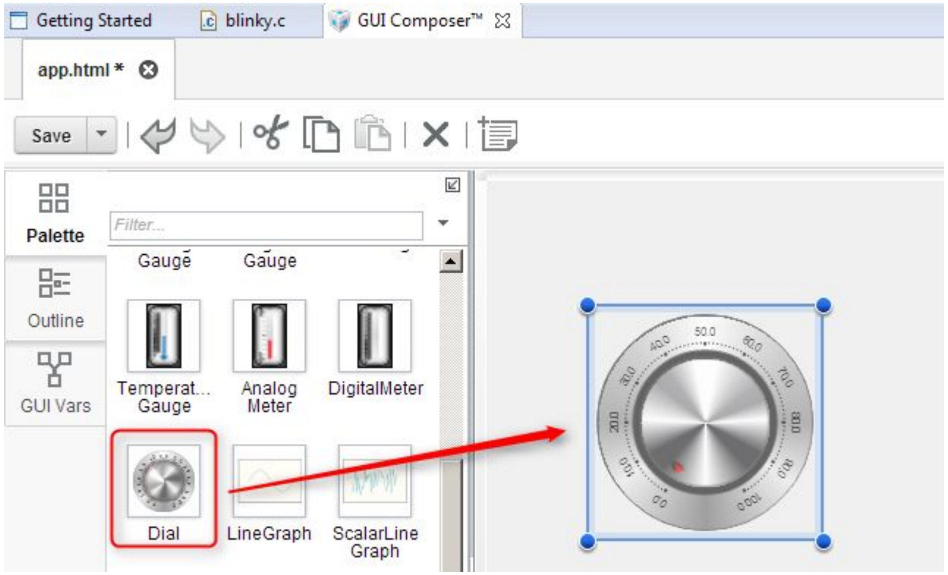
This opens a file called app.html in the GUI Composer Editor which is your HTML5 source file. On the left-hand side is the Palette and Projects list. The middle area is the canvas.

Add a Dial Widget

The Dial widget may be used to provide a user input to control a numeric variable

1. Go to the *Palette* on the left. It should be open by default
2. Expand *GUI Composer->Instrumentation*
3. Click on *Dial*, hold the left mouse button down and drag it onto the upper left part of the canvas in the middle of the screen and release the button





Change the Appearance of the Dial

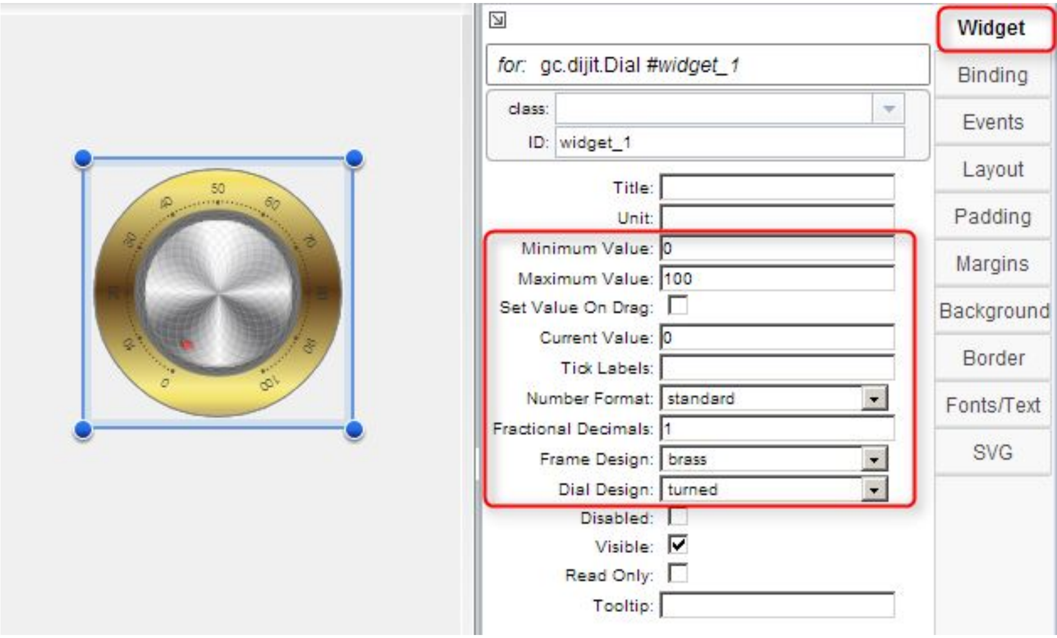
1. Select the Dial on the canvas

ⓘ There should be a blue square around it

2. Click on *Widget* on the right

ⓘ This will display some properties of the selected Widget

3. Set the following
   - Minimum value = 0
   - Maximum value = 100
   - Number Format = standard
   - Frame Design = brass
   - Dial Design = turned



for: gc.dijit.Dial *#widget_1*

class:

ID: widget_1

Title:

Unit:

Minimum Value: 0

Maximum Value: 100

Set Value On Drag: □

Current Value: 0

Tick Labels:

Number Format: standard

Fractional Decimals: 1

Frame Design: brass

Dial Design: turned

Disabled: □

Visible: ☑

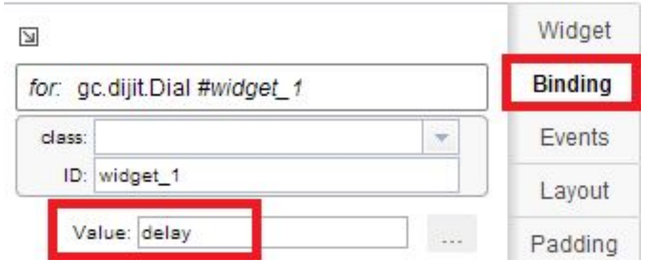
Read Only: □

Tooltip:

Bind a Variable to the Dial

1. Make sure the Dial is selected
2. Click on *Binding* on the right
3. In *Value:* field, add *delay*

ⓘ This binds the variable “delay” to the Dial widget, so the value of “delay” may be controlled by the widget



for: gc.dijit.Dial #widget_1

class:

ID: widget_1

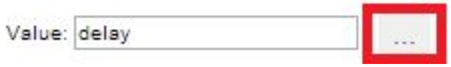
Value: delay

Define Pre/Post Processing Functions

Pre/Post Processing functions can be used to transform data or control format of displayed data. Preprocessing function is called when data is sent from target to widget and Post processing function is called in reverse direction.

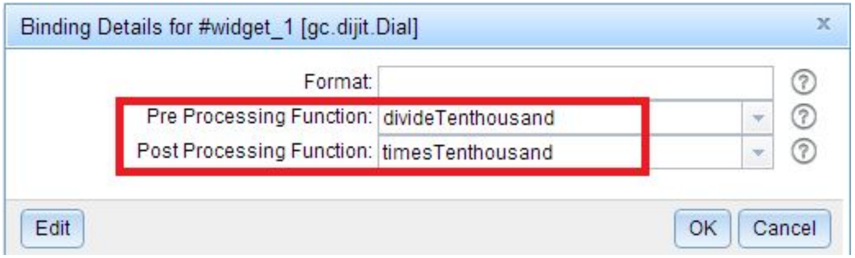
Here we demonstrate how to use these functions to adjust the data value

1. Click on … button beside *Value*



Value: delay

2. Add a pre-processing function called **divideTenthousand**
3. Add a post-processing function called **timesTenthousand**
4. Click OK



Binding Details for #widget_1 [gc.dijit.Dial]

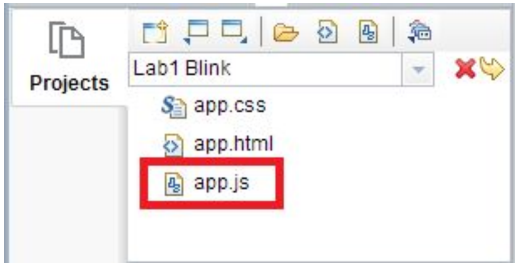
Format:

Pre Processing Function: divideTenthousand

Post Processing Function: timesTenthousand

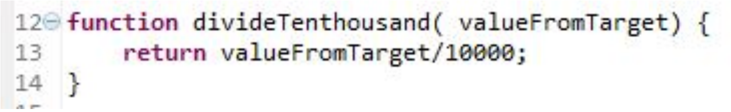
Define Pre/Post Processing Functions

5. Double-click on *app.js* in the Projects area at the bottom left



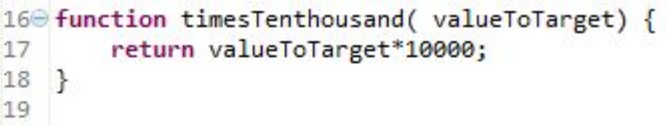
ⓘ This opens app.js in the CCS Editor with stub functions for divideTenthousand and timesTenthousand

6. In the **divideTenthousand** function change *return valueFromTarget;* to *return valueFromTarget/10000;*



```
function divideTenthousand( valueFromTarget) {
    return valueFromTarget/10000;
}
```

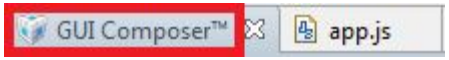
7. In the **timesTenthousand** function change *return value**From**Target;* to *return value**To**Target*10000;*



```
function timesTenthousand( valueToTarget) {
    return valueToTarget*10000;
}
```

8. Press the save button 💾 on the main toolbar to save *app.js* file

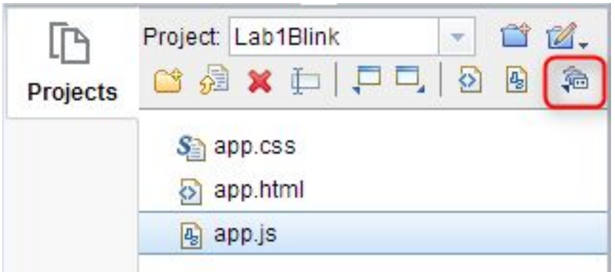
9. Click on GUI Composer tab in the editor



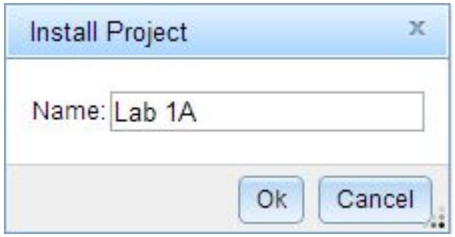
10. Click on the *Save* button Save ▾ to save app.html file

Generate App as a CCS Plug-in

1. Click on the *Install Project* button in the Projects area

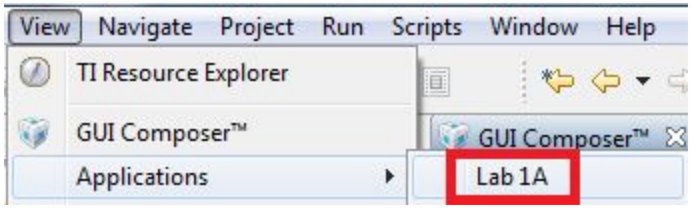


2. Name the App **Lab 1A** and click *OK*


Install Project

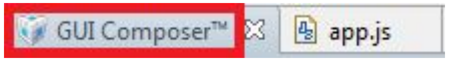
Name: Lab 1A

There will now be a menu item for the App in the CCS menu *View->Applications*

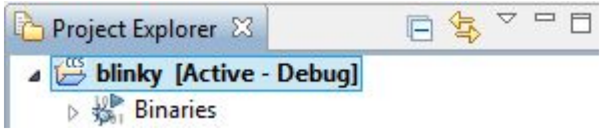


Load Target Application

1. If the *Project Explorer* is not visible double-click on the GUI Composer tab to restore it to its normal size



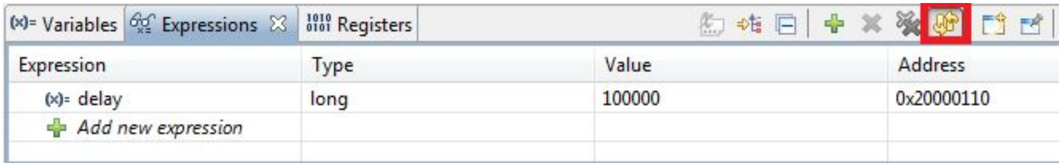
2. Select **blinky** in the *Project Explorer* view



3. Click the bug button

This will build the project, launch the debugger, flash the program onto the device and run to main()

4. Go to the *Expressions view* and add **delay**
5. Click the *Continuous Refresh* button

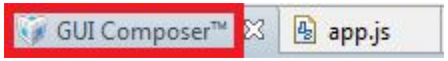


Expression	Type	Value	Address
delay	long	100000	0x20000110
Add new expression			

This will allow CCS to periodically read and display the value of **delay** as the program runs

Preview the App

1. Click on GUI Composer tab in the editor



2. Click on the *Preview* button at the top right ▶

If the program is already loaded on the device, as in this case, then preview mode will allow you to use the widgets

If there are errors or symbols are not loaded then a red X will appear next to the widgets

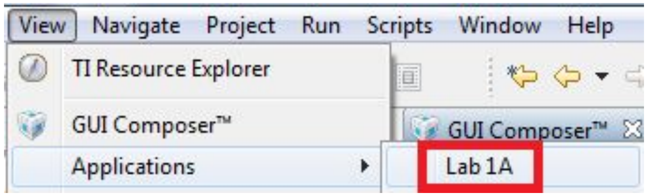
Test the App using Preview Mode

The App can be used either directly from Preview Mode or can be started up as a CCS Plug-in. This slides uses Preview Mode and next slide uses the Plug-in

1. Click the Run button ▶ in the *Debug* view to run the target application
2. Observe the blink rate of the LED on the Launchpad
3. Click on the dial and rotate the dial to adjust the value
4. Observe that the blink rate as well as the value of delay in the *Expressions* view changes as the dial is rotated
5. Click on *Exit Preview Mode* button

Exit Preview Mode

Test the App using Plug-in

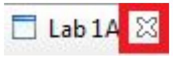


1. Go to menu *View -> Applications -> Lab 1A* to open the GUI composer app

   ⓘ This will open a new view named Lab1A with the GUI app
2. Arrange the view so that you can see the dial
3. Observe the blink rate of the LED on the Launchpad
4. Click on the dial and rotate the dial to adjust the value
5. Observe that the blink rate as well as the value of delay in the *Expressions* view changes as the dial is rotated

Clean Up

1. Click the X on the Lab 1A tab to close the GUI composer app



2. Click the Terminate button ■ on the *Debug View* to close the debug session

ⓘ CCS will shutdown the debugger and return to the CCS Edit perspective

LAB1B: CREATE AND USE MORE WIDGETS

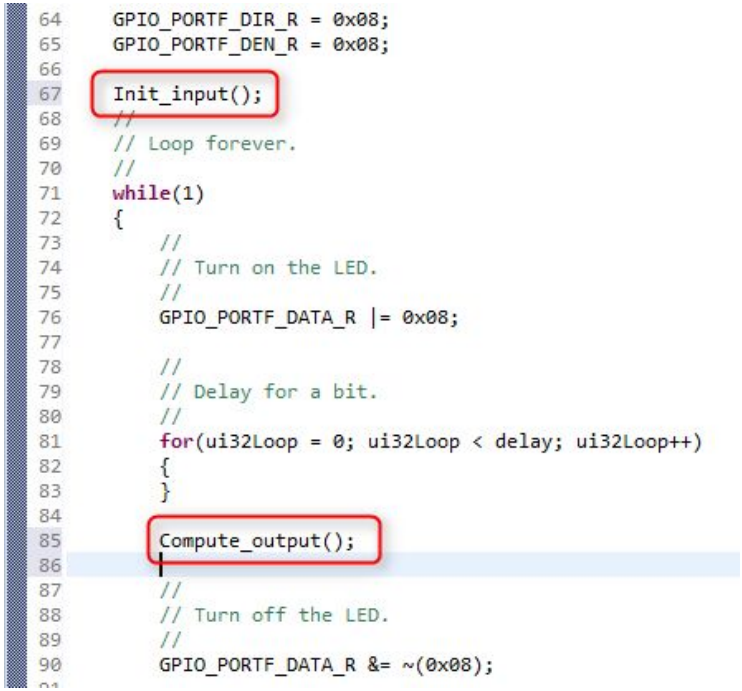
30 MINUTES

Open your lab materials and complete LAB 1B

Modify the Code

This step modifies the program code to add a computation loop

1. In *Project Explorer* view, double-click on **blinky.c** to open it if it is not already open
2. Add a call to Init_input() before the while(1) loop
3. Add a call to Compute_output() after the first for loop



```
GPIO_PORTF_DIR_R = 0x08;
GPIO_PORTF_DEN_R = 0x08;

Init_input();
//
// Loop forever.
//
while(1)
{
    //
    // Turn on the LED.
    //
    GPIO_PORTF_DATA_R |= 0x08;

    //
    // Delay for a bit.
    //
    for(ui32Loop = 0; ui32Loop < delay; ui32Loop++)
    {
    }

    Compute_output();

    //
    // Turn off the LED.
    //
    GPIO_PORTF_DATA_R &= ~(0x08);
```

Modify the Code

4. Add the following code at the top of the file after the declaration of variable delay

5. Click the Save 💾 button to save the file **blinky.c**

6. Do a quick review of the code to see what computations are being done

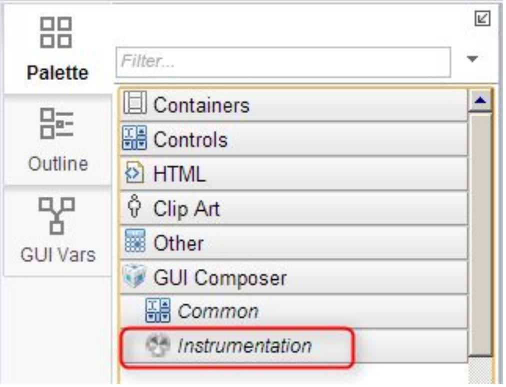
```
#define ARRAY_SIZE 20
volatile int input[ARRAY_SIZE];
volatile unsigned int first_output[ARRAY_SIZE];
volatile unsigned int second_output[ARRAY_SIZE];
volatile int Vin = 20;
volatile int Pin = 30;
volatile unsigned int Vout, Pout, Value1, Value2;

void Init_input();
void Compute_output();

void Init_input()
{
       int i = 0;
       for( i = 0; i < ARRAY_SIZE; ++i) {
              input[i] = i+ (i*10);
       };
}

void Compute_output()
{
    int i = 0;
    for (i = 0; i < ARRAY_SIZE; i++)
    {
    Vout = input[i] * Vin;
    first_output[i] = Vout * 0.025;
    Value1 = first_output[i];

    Pout = ARRAY_SIZE * Pin;
    second_output[i] = Pout * 0.075;
    Value2 = second_output[i];
    }

}
```

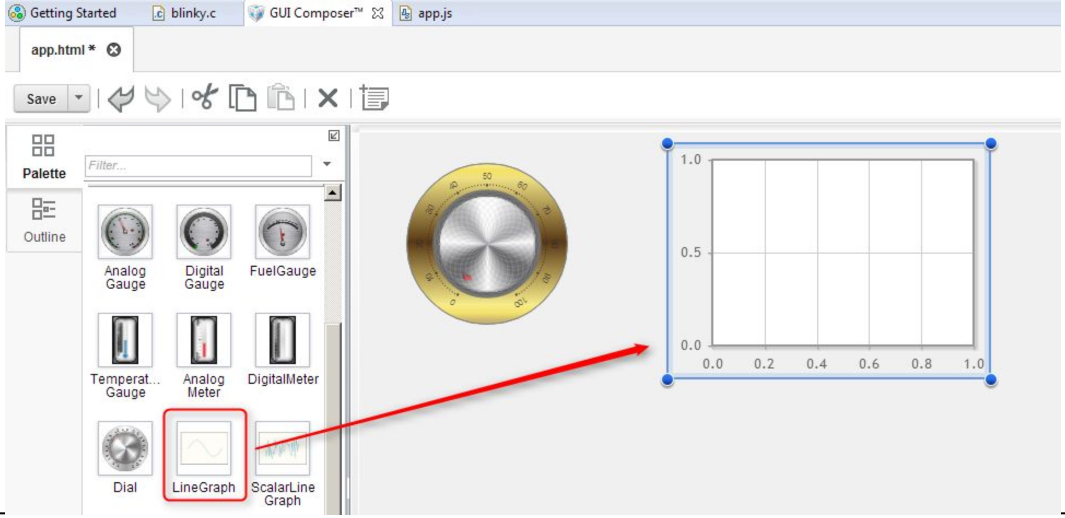
Add a Line Graph Widget

The LineGraph widget is a graph that can show up to 8 lines where each line represents the values of a variable of Array type

1. Go to the *GUI Composer* view and double-click on the tab to maximize it
2. Expand *GUI Composer->Instrumentation* if it is not already open
3. Click on *LineGraph*, hold the left mouse button down, drag it onto the canvas to the right of the Dial, and release the button

Adjust the size to make the widget smaller if desired



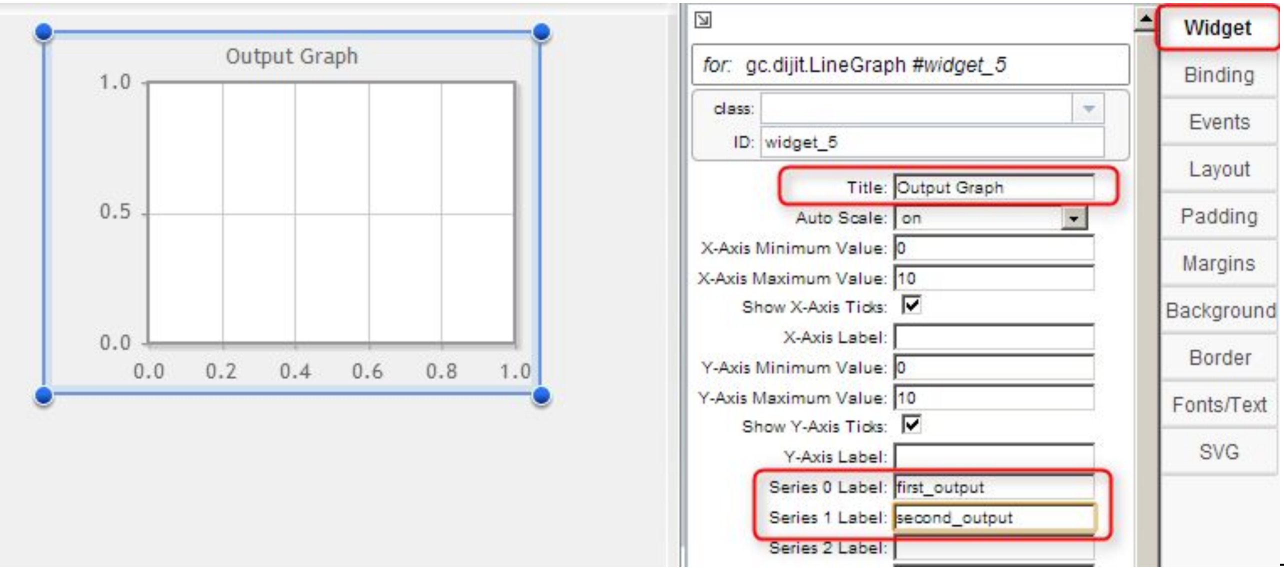


Set the Properties of the Line Graph

1. Select the LineGraph on the canvas

ⓘ There should be a blue square around it

2. Click on *Widget* on the right
3. Set the following
   - Title: Output Graph
   - Series 0 Label: first_output
   - Series 1 Label: second_output



for: gc.dijit.LineGraph #widget_5

class:

ID: widget_5

Title: Output Graph

Auto Scale: on

X-Axis Minimum Value: 0

X-Axis Maximum Value: 10

Show X-Axis Ticks: ☑

X-Axis Label:

Y-Axis Minimum Value: 0

Y-Axis Maximum Value: 10

Show Y-Axis Ticks: ☑

Y-Axis Label:

Series 0 Label: first_output

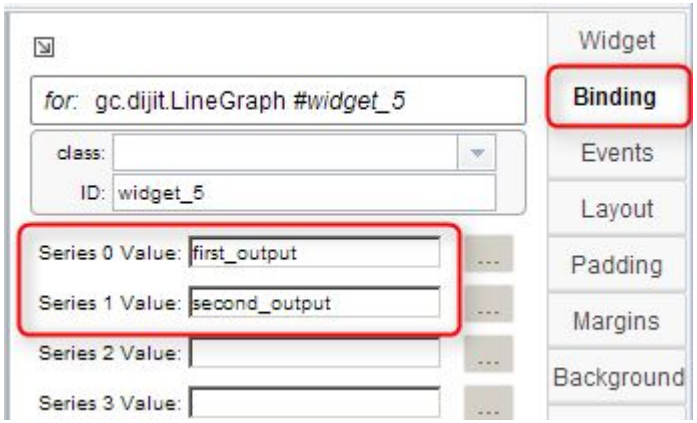
Series 1 Label: second_output

Series 2 Label:

Bind Variables to the Line Graph

1. Make sure the Line Graph is selected
2. Click on *Binding* on the right
3. Set the following
   - Series 0 Value: first_output
   - Series 1 Value: second_output

ⓘ This binds the Series 0 value of the graph to variable “first_output” and Series 1 value of the graph to variable “second_output”



for: gc.dijit.LineGraph *#widget_5*

class:

ID: widget_5

Series 0 Value: first_output

Series 1 Value: second_output

Series 2 Value:

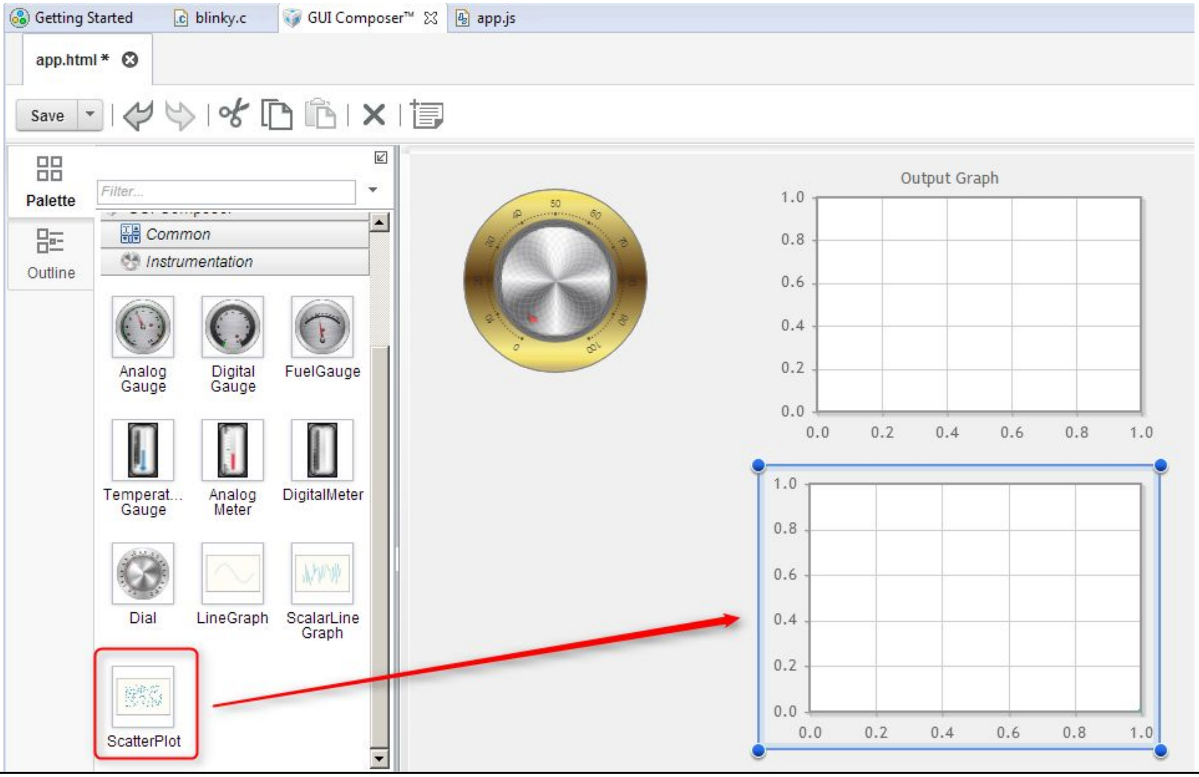
Series 3 Value:

Add a Scatter Plot Widget

The ScatterPlot widget is a graph that displays X-Y array data

1. In the GUI Composer Palette, under*GUI Composer->Instrumentation,* click on *ScatterPlot*, hold the left mouse button down, drag it onto the canvas below the Line Graph, and release the button

Adjust the size to make the widget smaller if desired



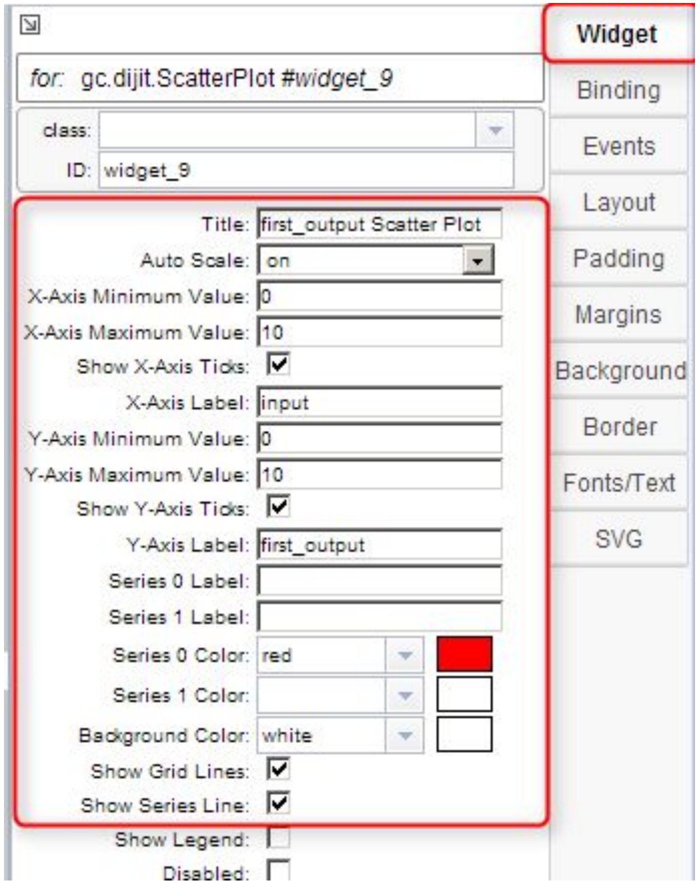
Set the Properties of the Scatter Plot

1. Select the Scatter Plot on the canvas

   ⓘ There should be a blue square around it

2. Click on *Widget* on the right
3. Set the following
   - Title: first_output Scatter Plot
   - X-Axis Label: input
   - Y-Axis Label: first_output
   - Series 0 Color: red
   - Show Series Line: enable

     ⓘ Without this enabled the graph will have disconnected dots. Give both versions a try if you wish



for: gc.dijit.ScatterPlot #*widget_9*

class:

ID: widget_9

Title: first_output Scatter Plot

Auto Scale: on

X-Axis Minimum Value: 0

X-Axis Maximum Value: 10

Show X-Axis Ticks: ☑

X-Axis Label: input

Y-Axis Minimum Value: 0

Y-Axis Maximum Value: 10

Show Y-Axis Ticks: ☑

Y-Axis Label: first_output

Series 0 Label:

Series 1 Label:

Series 0 Color: red 🟥

Series 1 Color:

Background Color: white

Show Grid Lines: ☑

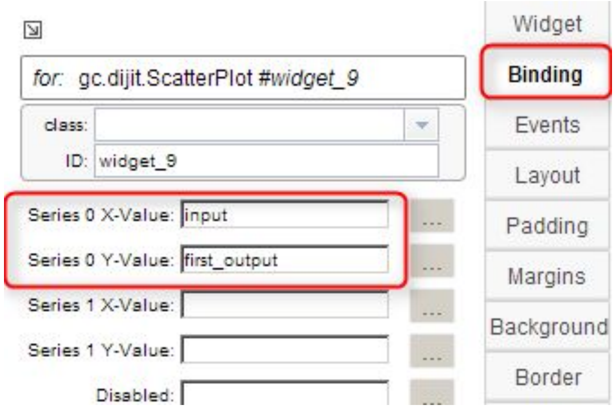
Show Series Line: ☑

Show Legend: ☐

Disabled: ☐

Bind Variables to the Scatter Plot

1. Make sure the ScatterPlot is selected on the canvas
2. Click on *Binding* on the right
3. Set the following
   - Series 0 X-Value: input
   - Series 0 Y-Value: first_output



for: gc.dijit.ScatterPlot #widget_9

class:

ID: widget_9

Series 0 X-Value: input

Series 0 Y-Value: first_output

Series 1 X-Value:

Series 1 Y-Value:

Disabled:

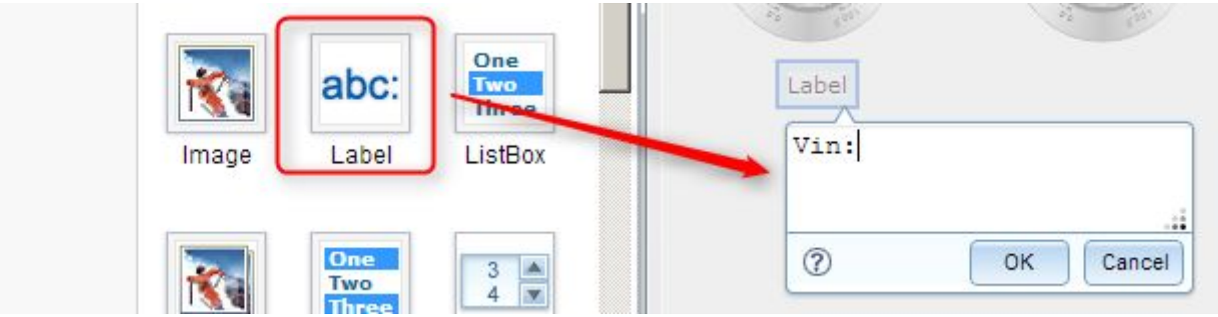
This binds the Series 0 X-value of the graph to variable “input” and Series 0 Y-value of the graph to variable “first_output”, so the plot will show values of input against first_output. If you wish to plot another series of values you can use the Series 1 X-Value and Y-Value

4. Click the Save button Save ▾ to save app.html

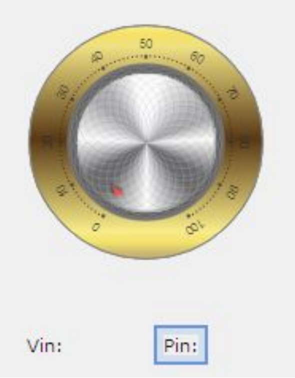
Add Label Widgets

The Label widget is used to add text to the GUI application

1. In the GUI Composer Palette, expand *GUI Composer->Common*
2. Click on *Label*, hold the left mouse button down, drag it onto the left side of the canvas, and release the button
3. Enter **Vin:** for the Label and click OK



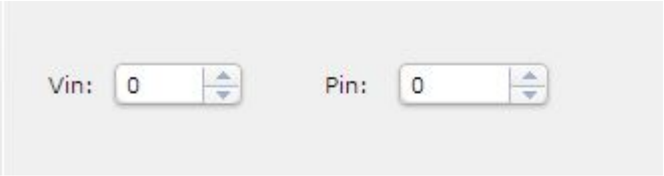
4. Do the above step again to drag another Label into the canvas
5. Enter **Pin:** for the Label and click OK
6. Position the labels below the dial as shown



Add Number Spinner Widgets

The NumberSpinner widget provides arrows to increment/decrement values and is useful to provide fine value control in your application. It can also accept values that user enters with a keyboard

1. In the GUI Composer Palette, under *GUI Composer->Common,* click on *NumberSpinner*, hold the left mouse button down and drag it onto the canvas and release the button
2. Do the above step again to drag another NumberSpinner into the canvas
3. Adjust the size and position of the Labels and NumberSpinners so they are positioned next to the two labels as shown here

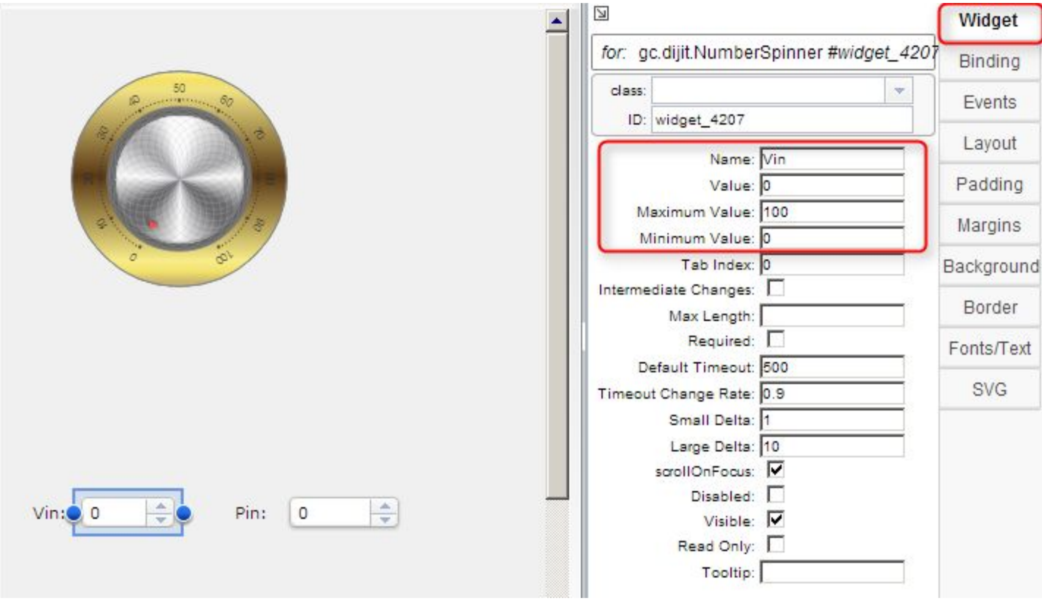


Set Properties and Bindings for Number Spinners

1. Select the *NumberSpinner* next to the Vin Label

   There should be a blue square around it

2. Click on *Widget* on the right
3. Set the following
   - Name: Vin
   - Value: 0
   - Maximum Value: 100
   - Minimum Value: 0



for: gc.dijit.NumberSpinner #widget_4207

class:

ID: widget_4207

Name: Vin

Value: 0

Maximum Value: 100

Minimum Value: 0

Tab Index: 0

Intermediate Changes: □

Max Length:

Required: □

Default Timeout: 500

Timeout Change Rate: 0.9

Small Delta: 1

Large Delta: 10

scrollOnFocus: ☑

Disabled: □

Visible: ☑

Read Only: □

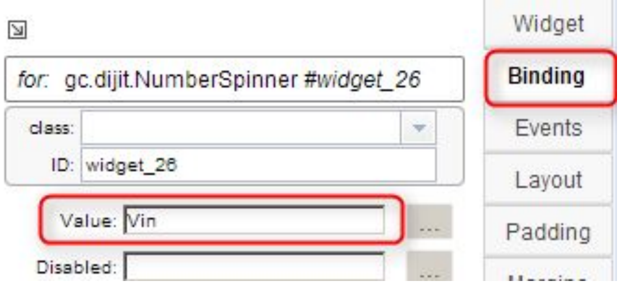
Tooltip:

Vin: 0

Pin: 0

4. Click on *Binding* on the right
5. Set the following
   - Value: Vin

   This binds the variable “Vin” to the NumberSpinner widget, so the value of “Vin” may be controlled by the widget



for: gc.dijit.NumberSpinner #widget_26

class:

ID: widget_26

Value: Vin

Disabled:

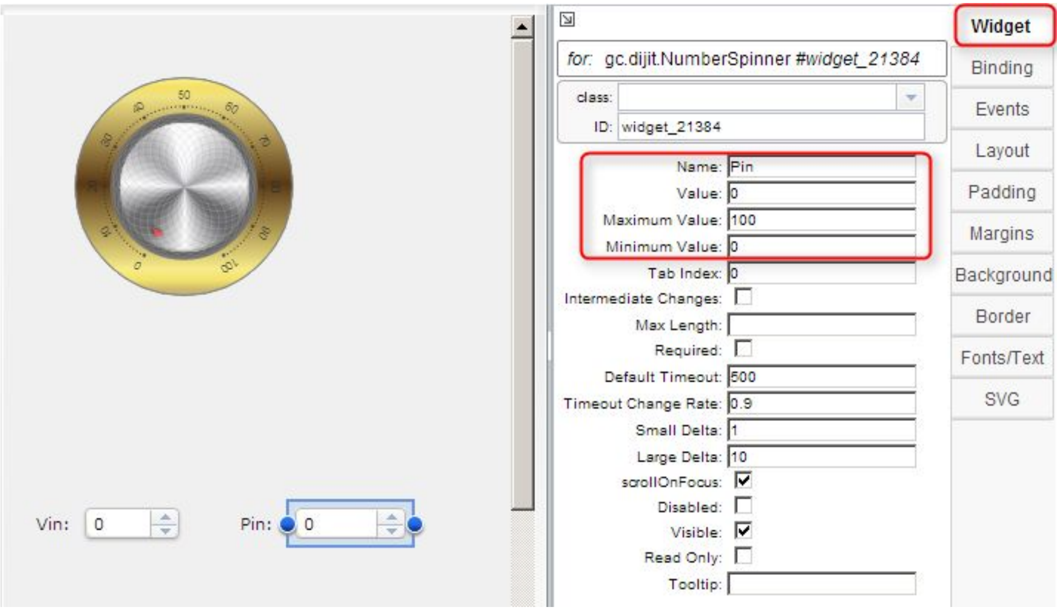
Set Properties and Bindings for Number Spinners

1. Select the *NumberSpinner* next to the Pin Label

   There should be a blue square around it

2. Click on *Widget* on the right
3. Set the following
   - Name: Pin
   - Value: 0
   - Maximum Value: 100
   - Minimum Value: 0
4. Click on *Binding* on the right
5. Set the following
   - Value: Pin

   This binds the variable “Pin” to the NumberSpinner widget, so the value of “Pin” may be controlled by the widget



for: gc.dijit.NumberSpinner #widget_21384

class:

ID: widget_21384

Name: Pin

Value: 0

Maximum Value: 100

Minimum Value: 0

Tab Index: 0

Intermediate Changes: □

Max Length:

Required: □

Default Timeout: 500

Timeout Change Rate: 0.9

Small Delta: 1

Large Delta: 10

scrollOnFocus: ☑

Disabled: □

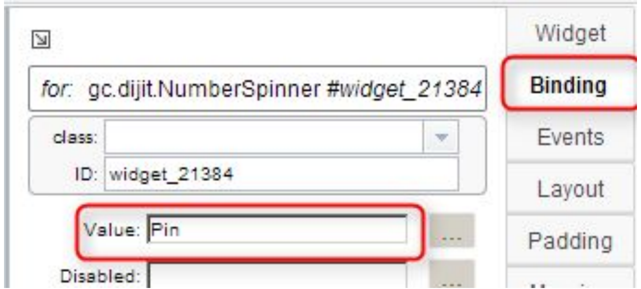
Visible: ☑

Read Only: □

Tooltip:

Vin: 0

Pin: 0



for: gc.dijit.NumberSpinner #widget_21384

class:

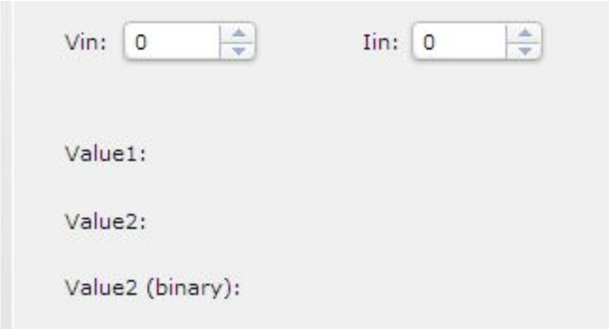
ID: widget_21384

Value: Pin

Disabled:

Add more Label Widgets

1. In the GUI Composer Palette, under *GUI Composer->Common,* click on *Label*, hold the left mouse button down and drag it onto the canvas and release the button
2. Enter **Value1:** for the Label and click OK
3. Drag two more *Labels* into the canvas
4. Enter **Value2:** for the second Label and click OK
5. Enter **Value2 (binary):** for the third Label and click OK
6. Position the Labels below the *NumberSpinners* as shown here or as you prefer on the canvas



Vin: 0

Iin: 0

Value1:

Value2:

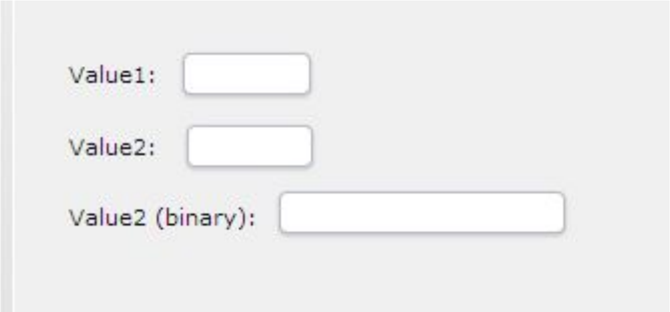
Value2 (binary):

7. OPTIONAL: Select one of the Labels, click on Fonts/Text on the right and modify size, color etc. to see its effect on the Label

Add TextBox Widgets

The TextBox widget can be used to display a string or a number and can allow user to enter strings or values

1. In the GUI Composer Palette, click on *TextBox*, hold the left mouse button down and drag it onto the canvas and release the button
2. Do the above step two more times to drag two more *TextBoxes* into the canvas
3. Adjust the size of the TextBoxes and position them besides each of the Labels from the previous slide

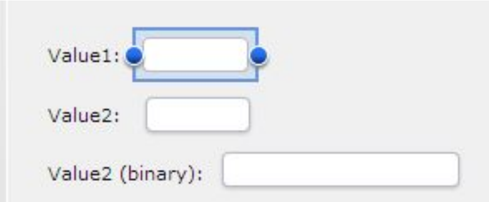


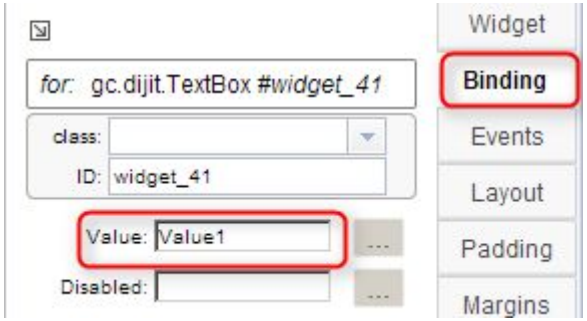
4. Click the Save button to save app.html

Set Bindings for TextBoxes

1. Select the first TextBox next to Label Value1
2. Click on *Binding* on the right
3. Set the following
   - Value: Value1

   ⓘ This binds the variable “Value1” to the TextBox widget, so “Value1” can be displayed in the widget





for: gc.dijit.TextBox #widget_41

class:

ID: widget_41

Value: Value1

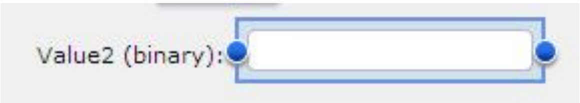
Disabled:

4. Select the second TextBox next to Label Value2
5. Click on *Binding* on the right
6. Set the following
   - Value: Value2

   ⓘ This binds the variable “Value2” to the TextBox widget, so “Value2” can be displayed in the widget

Set Bindings for TextBoxes

7. Select the third TextBox next to Label Value2 (binary)



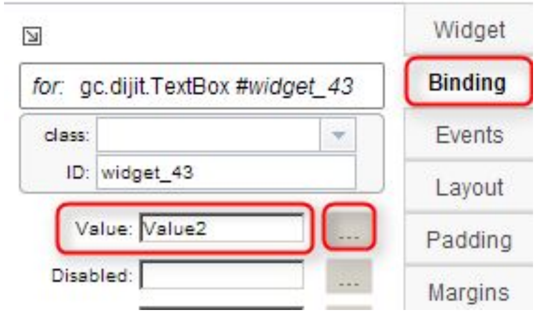
Value2 (binary):

8. Click on *Binding* on the right

9. Set the following

   – Value: Value2

     This binds the variable “Value2” to the TextBox widget, so “Value2” can be displayed in the widget



for: gc.dijit.TextBox *#widget_43*

class:

ID: widget_43

Value: Value2

Disabled:

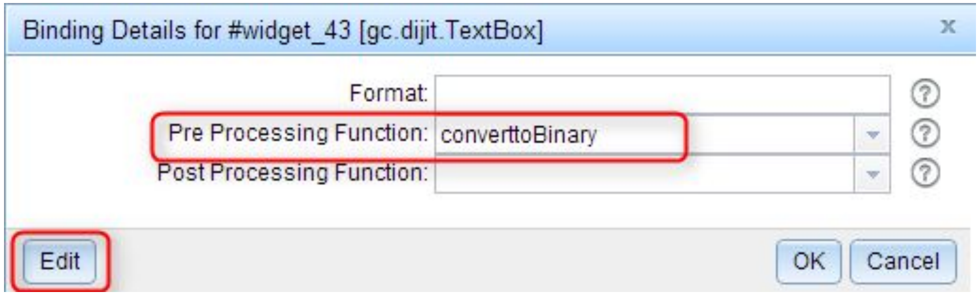
10. Click on … button beside Value2

11. Add a pre-processing function called **converttoBinary**

12. Click on Edit button

    This opens the file app.js in the CCS Editor with a stub function

    If you are prompted asking if you want to replace the contents of app.js with these changes, click Yes



Binding Details for #widget_43 [gc.dijit.TextBox]

Format:

Pre Processing Function: converttoBinary

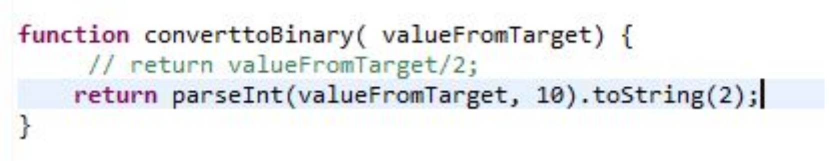
Post Processing Function:

Set Bindings for TextBoxes

13. In the **converttoBinary** function within app.js, change the code to:
return parseInt(valueFromTarget, 10).toString(2);

This is a Javscript function to change an integer to binary

Since Preprocessing function is called when data is sent from target to widget, this enables the TextBox widget to display the value of variable in binary format



```
function converttoBinary( valueFromTarget) {
     // return valueFromTarget/2;
    return parseInt(valueFromTarget, 10).toString(2);
}
```

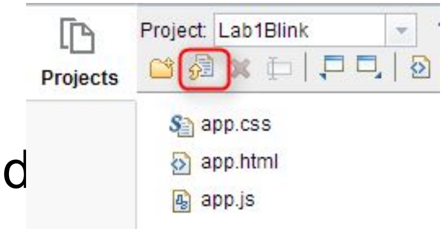
14. Press the save button 💾 on the main toolbar to save the app.js file
15. Go back to the *GUI Composer* view and click OK on the Binding Details window

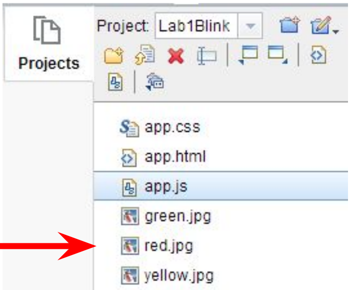
Add MultiImage Widget

The MultiImage widget can be used to display visual status information, such as a selected image, based on the value of a variable it is bound to. By default, first image will be displayed when value of variable=0, second image when value of variable=1 and so on

Here the MultiImage widget is used to display a LED image of particular color based on the value of variable “Value1”
The steps below upload the images into the GUI Composer project

1. In the GUI Composer Projects view, click on Add Files
2. Click on Select Files
3. Browse to c:\guicomposer_workshop, select red.jpg and click Open
4. Click on Upload
5. Do steps 2 through 4 for yellow.jpg and green.jpg
6. Close the *Add File* pop-up window
7. The files should now be in the Projects view



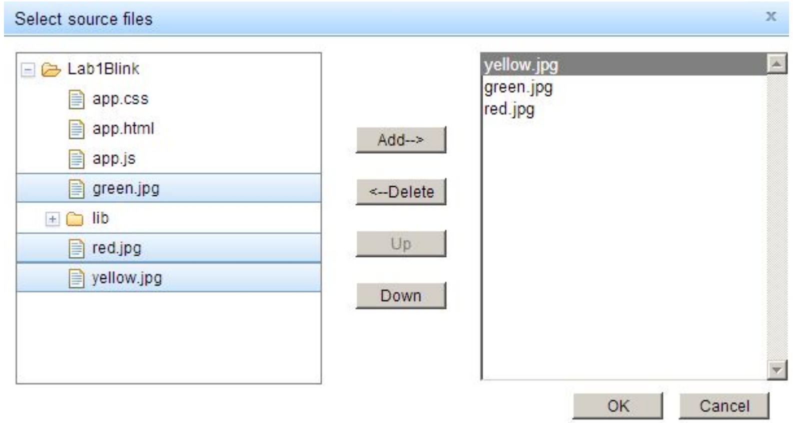


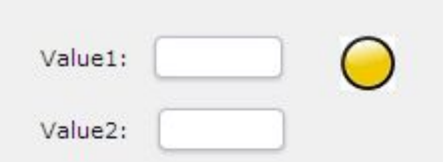
Add MultiImage Widget

The steps below set which image is to be displayed when value of variable is 0, 1, 2, etc.

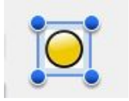
1. In the GUI Composer Palette, click on *MultiImage*, hold the left mouse button down and drag it onto the canvas and release the button
2. In the pop-window window, select files green.jpg, red.jpg and yellow.jpg one at a time and click on Add
3. Select yellow.jpg and click on Up to move it to the first in the list, followed by green.jpg and then red.jpg

   This sets it so yellow.jpg is displayed when value of variable is 0, green.jpg is displayed when value of variable is 1 and red.jpg is displayed when value of variable is 2
4. Click OK
5. Position the widget next to the TextBox beside Value1

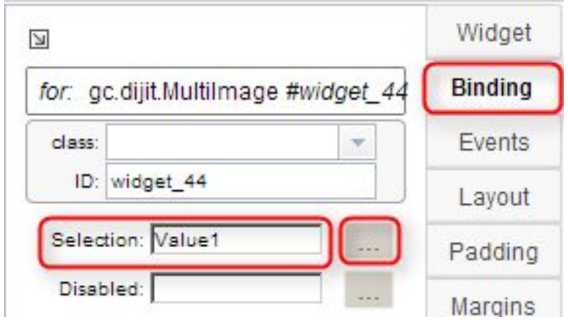




Set Binding for MultiImage



1. Select the *MultiImage* widget
2. Click on *Binding* on the right
3. Set the following
   - Selection: Value1
     This binds the variable “Value1” to the MultiImage widget



for: gc.dijit.MultiImage #widget_44

class:

ID: widget_44

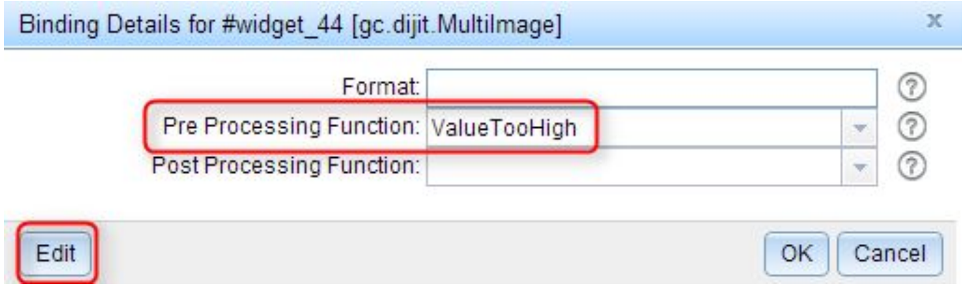
Selection: Value1 ...

Disabled: ...

4. Click on … button beside Value1
5. Add a pre-processing function called **ValueTooHigh**
6. Click on Edit button

   This opens the file app.js in the CCS Editor with a stub function

   If you are prompted asking if you want to replace the contents of app.js with these changes, click Yes



Binding Details for #widget_44 [gc.dijit.MultiImage]

Format:

Pre Processing Function: ValueTooHigh

Post Processing Function:

Set Binding for MultiImage

7. In the **ValueTooHigh** function enter the code as shown below:

```
function ValueTooHigh( valueFromTarget) {
if (valueFromTarget <50 )
    return 0;
else
    if (valueFromTarget > 300 )
    return 2;
else
    return 1;
}
```

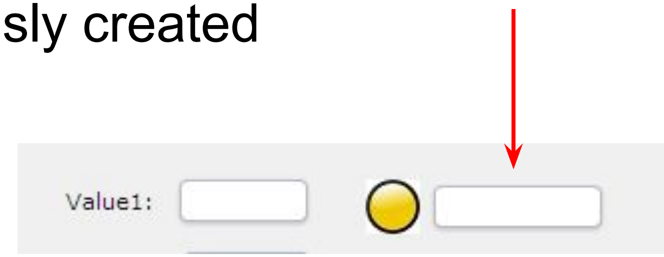
This function processes the variable “Value1” and returns appropriate index (0,1 or 2) that in turn determines the image to be displayed. Return value of 0 displays first image (yellow.jpg), 1 displays second image (green.jpg), 2 displays third image (red.jpg)

8. Press the save button 💾 on the main toolbar to save the app.js file
9. Go back to the *GUI Composer* view and click OK on the Binding Details window

Add TextBox Widget

This TextBox widget prints out if Value1 is “Too High”, “Too Low” or “OK” based on some checks. We will not be binding this widget to a variable but will instead use GUI Vars (see next slides) to determine the value to be written to the TextBox

1. In the GUI Composer Palette, click on *TextBox*, hold the left mouse button down and drag it onto the canvas and release the button
2. Resize and position the TextBox to the right of MultiImage widget previously created



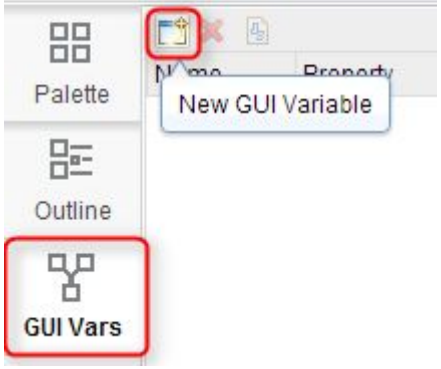
Add GUI Vars

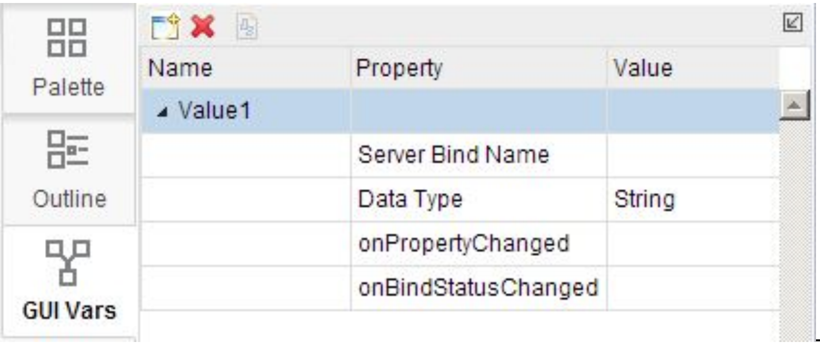
GUI Vars allow you to bind to a target variable without a widget. It lets you perform an action when the value of the variable changes

Here GUI Vars is used to bind to variable “Value1” and write out text to the TextBox based on the value of the variable

1. In *GUI Composer* view, click on *GUI Vars* on the left
2. Click on New GUI Variable icon
3. Give it the name Value1

   It can be any name but keeping it the same as variable name for simplicity
4. Click OK
5. Expand the newly created item to edit its properties





Name	Property	Value
Value1		
	Server Bind Name	
	Data Type	String
	onPropertyChanged	
	onBindStatusChanged	

Add GUI Vars

5. For *Server Bind Name*, click on Value column and enter **Value1** (this is the target variable you want to listen to)
6. For *Data Type*, click on Value column and select **Long** (closest match to your variable type)
7. For *onPropertyChanged*, click on Value column
8. Click on Edit button

ⓘ This opens the file app.js in the CCS Editor with a stub function

ⓘ If you are prompted asking if you want to replace the contents of app.js with these changes, click Yes

Add GUI Vars

9. In the **onValue1PropertyChanged** function in app.js, add the following code:

```
function onValue1PropertyChanged( propertyName, newValue, oldValue) {

    var t = $TI.GUIVars;
    var var0 = t.getValue('Value1');

    if (var0 < 50)
    {dijit.byId('widget_46').set('value', "TOO LOW");}
    else
    if (var0 > 300)
    {dijit.byId('widget_46').set('value', "TOO HIGH");}
    else
    {dijit.byId('widget_46').set('value', "OK");}
}
```

ⓘ This function gets the value of variable “Value1”, checks if it is less than or greater than certain values, and based on the result writes a particular text to the TextBox widget previously created (widget_46)

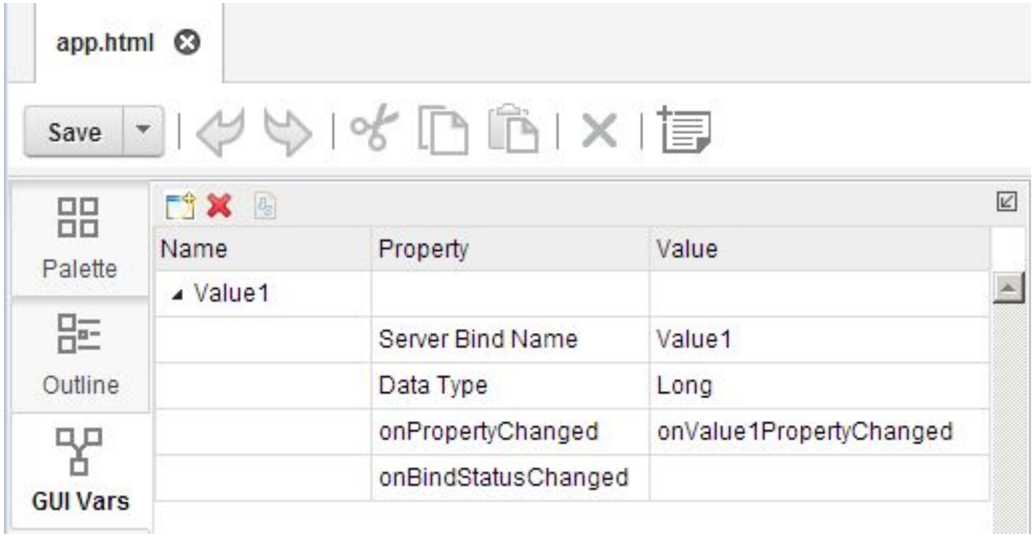
10. Adjust the widget ID in the code so it matches the widget ID for the last TextBox created

ⓘ To find the widget ID, select the TextBox created in slide 60, click Binding on the right and check the widget ID

11. Press the save button 💾 on the main toolbar to save the app.js file

Add GUI Vars

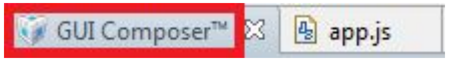
12. Go back to *GUI Composer* view and click on Save button Save ▾ to save app.html



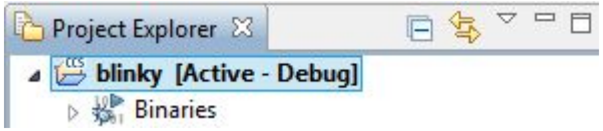
Name	Property	Value
◢ Value1		
	Server Bind Name	Value1
	Data Type	Long
	onPropertyChanged	onValue1PropertyChanged
	onBindStatusChanged	

Load Target Application

1. If the *Project Explorer* is not visible double-click on the GUI Composer tab to restore it to its normal size



2. Select **blinky** in the *Project Explorer* view

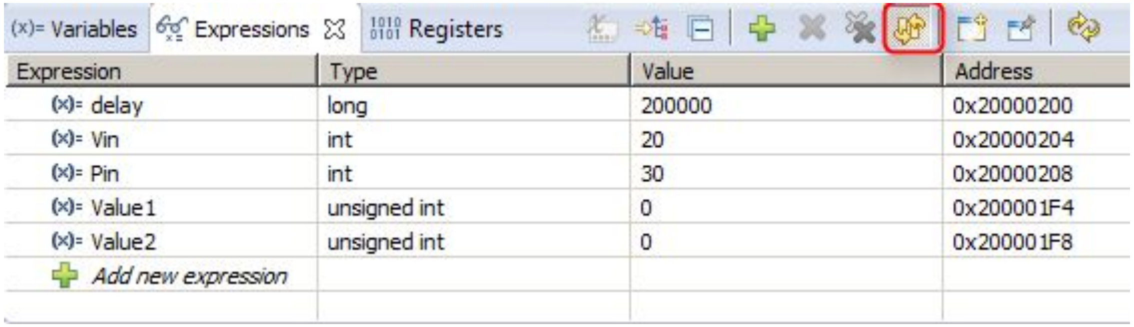


3. Click the bug button

This will build the project, launch the debugger, flash the program onto the device and run to main()

4. Click on the *Expressions view* and add Vin, Pin, Value1and Value2
5. Ensure that *Continuous Refresh* button is still enabled

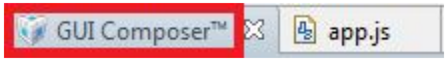
This tells CCS to periodically read and display the value of the variables as the program runs



Expression	Type	Value	Address
(x)= delay	long	200000	0x20000200
(x)= Vin	int	20	0x20000204
(x)= Pin	int	30	0x20000208
(x)= Value1	unsigned int	0	0x200001F4
(x)= Value2	unsigned int	0	0x200001F8
Add new expression			

Preview the App

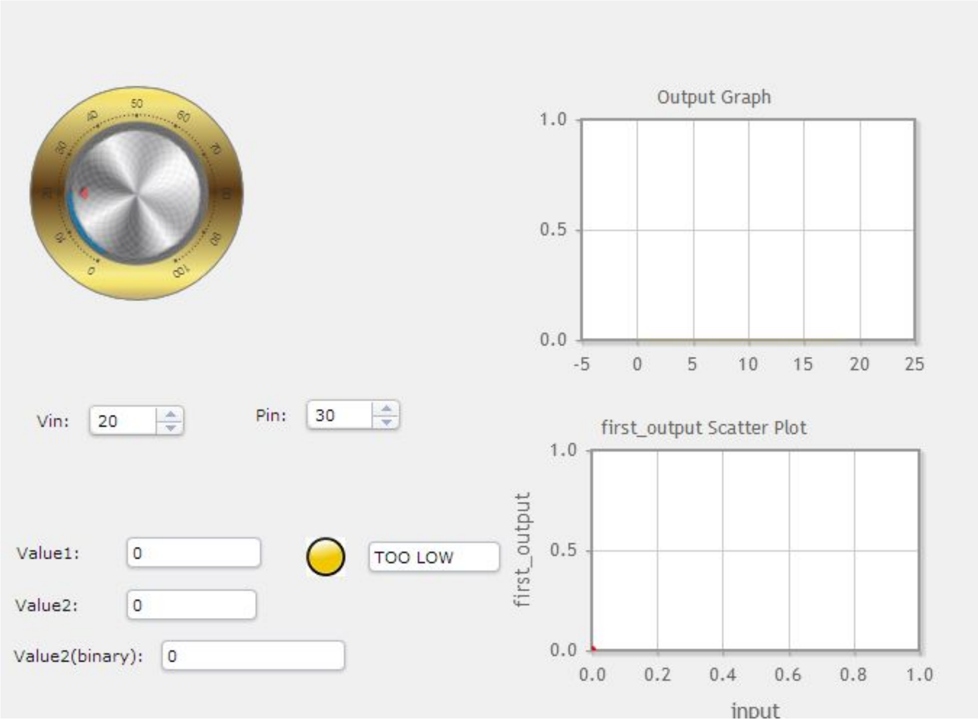
1. Click on GUI Composer tab in the editor



2. Click on the *Preview* button ▶ at the top right

ⓘ If the program is already loaded on the device then preview mode will allow you to use the widgets

⚠ If there are errors or symbols are not loaded a red X will appear



Vin: 20

Pin: 30

Value1: 0

TOO LOW

Value2: 0

Value2(binary): 0

Test the App

1. Click the Run button ▶ to run the target application
2. Go back to *GUI Composer* view and observe the following:
   - LED on Launchpad is blinking
   - In *Expressions* view, value of **delay** is 200000
   - Line Graph and Scatter Plot are updated
   - In *Expressions* view, **Vin** and **Pin** show their initial values: 20 and 30
   - In the GUI, **Value1** and **Value2** are displayed, with **Value2** also displayed in binary, and their values match those in the *Expressions* view
   - In the GUI, **Green LED light** is displayed based on **Value1** being within the desired range (>50 and <300) and Text Box next to it says **OK**

Test the App

3. Adjust the value of **Vin** by using the Number Spinner or typing a value in the box
4. Observe the following:
   - **Vin** is updated in *Expressions* view and GUI
   - **Value1** is updated accordingly
   - Line Graph and Scatter Plot are updated accordingly
   - If **Vin** is set to a value (> 57) that makes **Value1** greater than 300, then **Red LED light** is displayed and Text Box next to it says **TOO HIGH**
   - If **Vin** is set to a value (< 10) that makes **Value1** less than 50, then **Yellow LED light** is displayed and Text Box next to it says **TOO LOW**
5. Adjust the value of **Pin** by using the Number Spinner or typing a value in the box
6. Observe the following:
   - **Pin** is updated in *Expressions* view and GUI
   - **Value2** and **Value2 (binary)** are updated accordingly
   - Line Graph is updated accordingly

Clean Up

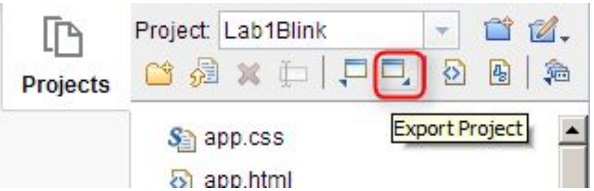
1. Click on the *Exit Preview Mode* button

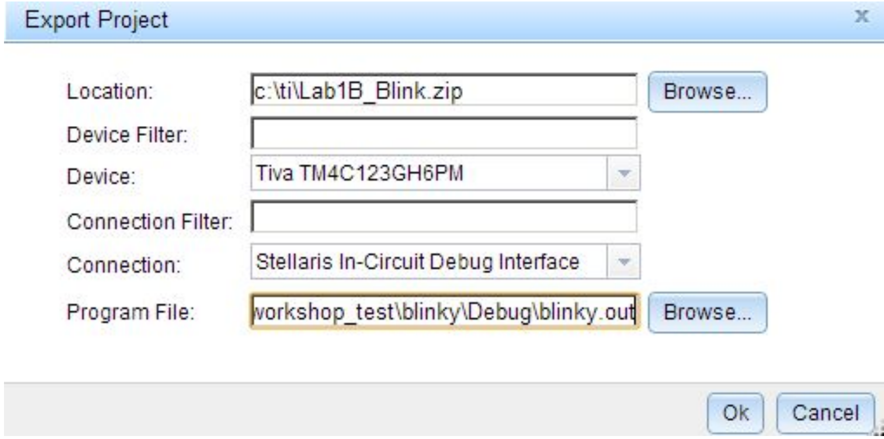
2. Click the Terminate button ■ on the *Debug View* to close the debug session

ⓘ CCS will shutdown the debugger and return to the Edit perspective

Exporting the GUI Application

1. Click on the *GUI Composer* view
2. In the *Projects* area click on the *Export Project* button
3. Specify the following:
   - Location: *C:\ti\Lab1B_Blink.zip*
     (location for saving exported project)
   - Device: *Tiva TM4C123GH6PM*
   - Connection: *Stellaris In-Circuit Debug Interface*
   - Program File: *C:\Users\<username>\GUI Composer Workshop\blinky\Debug\blinky.out*
     (browse to location of program file)
4. Click Ok





Export Project

Location: c:\ti\Lab1B_Blink.zip

Device Filter:

Device: Tiva TM4C123GH6PM

Connection Filter:

Connection: Stellaris In-Circuit Debug Interface

Program File: workshop_test\blinky\Debug\blinky.out

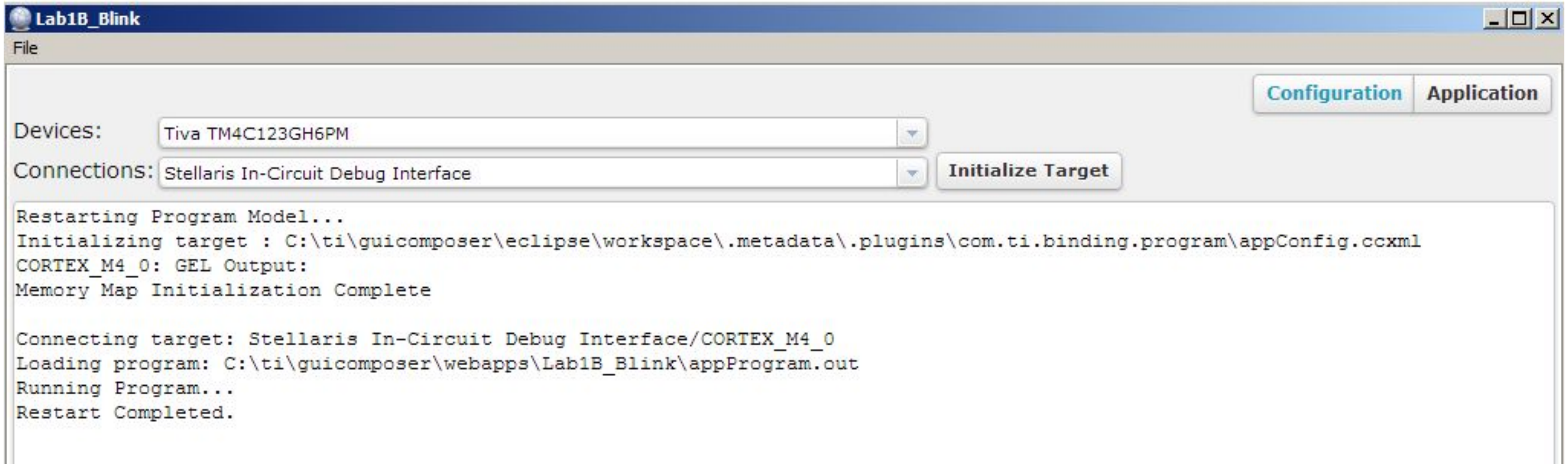
Add App to GUI Composer Runtime

1. Open a file explorer window
2. Go to c:\ti
3. Right click on Lab1B_Blink.zip
4. Select Extract All
5. Extract the files to c:\ti\guicomposer\webapps

Run the Standalone Application

1. Close CCS
2. Double click on *Launcher.exe* located in c:\ti\guicomposer\webapps\Lab1B_Blink

The splash screen will appear then it will go through a startup sequence while it connects to the device and flashes the program



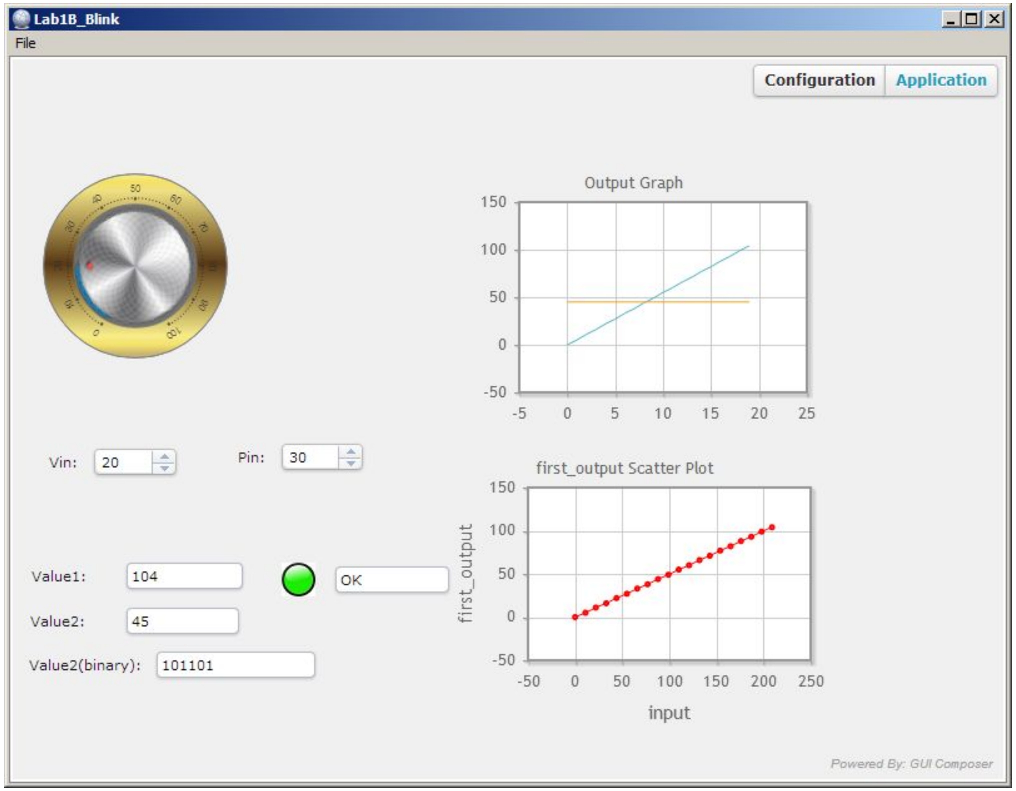
Devices: Tiva TM4C123GH6PM

Connections: Stellaris In-Circuit Debug Interface

```
Restarting Program Model...
Initializing target : C:\ti\guicomposer\eclipse\workspace\.metadata\.plugins\com.ti.binding.program\appConfig.ccxml
CORTEX_M4_0: GEL Output:
Memory Map Initialization Complete

Connecting target: Stellaris In-Circuit Debug Interface/CORTEX_M4_0
Loading program: C:\ti\guicomposer\webapps\Lab1B_Blink\appProgram.out
Running Program...
Restart Completed.
```

Run the Standalone Application

When configuration is complete, the GUI app will come up



Lab1B_Blink

File

Configuration Application

Vin: 20

Pin: 30

Value1: 104

Value2: 45

Value2(binary): 101101

OK

3. Try adjusting the dials and other parameters (Vin, Pin etc) as we did before and observe the graphs and values change accordingly
4. Close the application window when done

LAB 2: UART TRANSPORT

LAB conventions

- Lab steps are numbered for easier reference
  1. …
  2. …

- Explanations, notes, warnings are written in blue
  - Warnings are shown with ⚠
  - Information is marked with ℹ
  - Tips and answers are marked with 💡
  - Questions are marked with ❓

UART Transport: Exercise Summary

- **Key Objectives**
  - Use a TI-RTOS application with UARTMon module enabled
  - Create a simple GUI that binds a widget to a target variable
  - Use UART communication to view and control the application through GUI composer
- **Tools and Concepts Covered**
  - UART transport using a TI-RTOS application
  - GUI Builder tool
  - Variable binding
  - Target variable modification
  - Target variable display

- **NOTE:** This lab uses a TI-RTOS program. For using UART communication with a non TI-RTOS program, please refer to this wiki page: **http://processors.wiki.ti.com/index.php/ProgramModelUart_GuiComposer**

LAB 2: UART TRANSPORT EXAMPLE

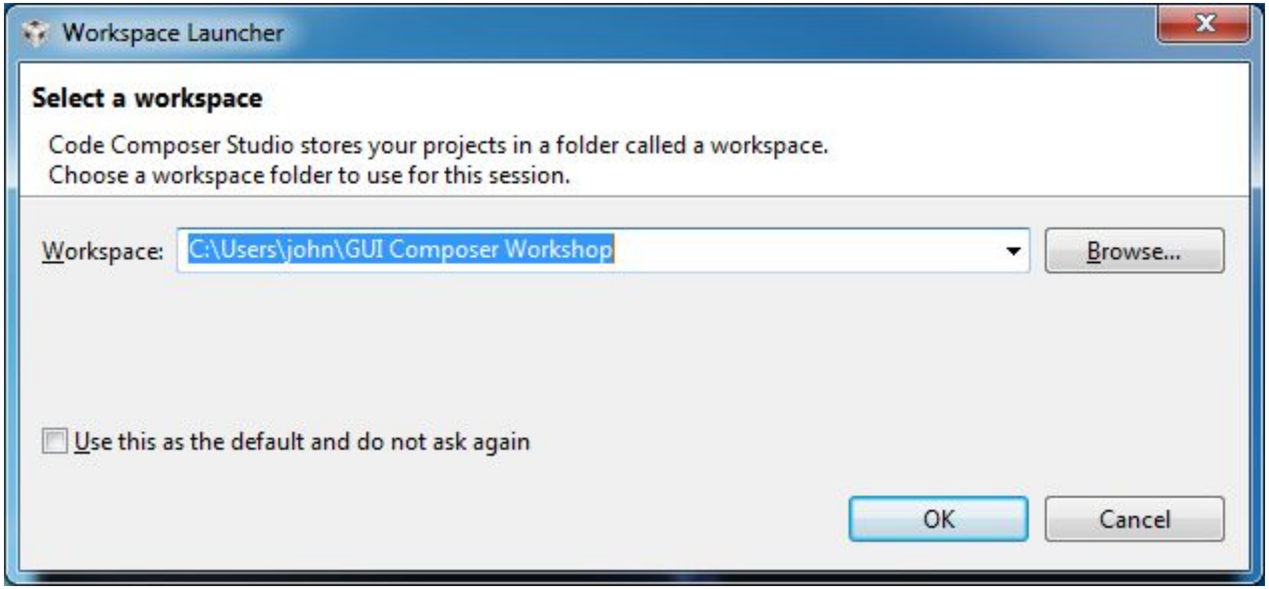
20 MINUTES

Open your lab materials and complete LAB 2

Launch CCS

1. Double click on the Code Composer Studio desktop icon

2. Specify “GUI Composer Workshop” as the workspace



Select a workspace

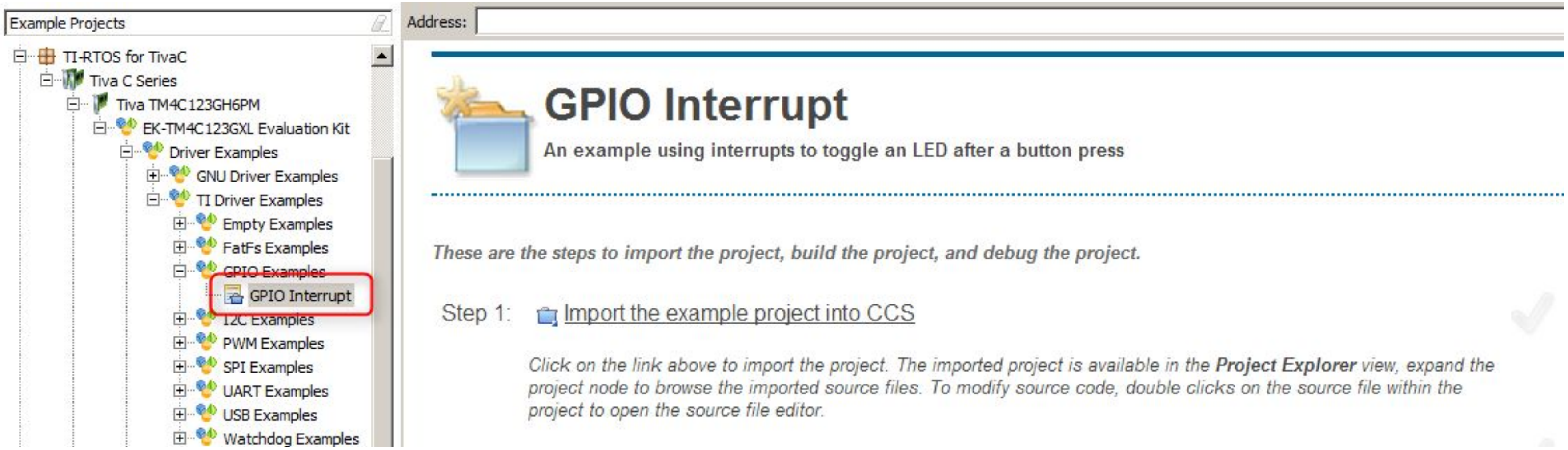
Code Composer Studio stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: C:\Users\john\GUI Composer Workshop

□ Use this as the default and do not ask again

Import and Build ‘GPIO Interrupt’ Project

1. Go to menu *View->Resource Explorer (Examples)*
2. Expand *TI-RTOS for TivaC->Tiva C Series->Tiva TM4C123GH6PM-> EK-TM4C123GXL Evaluation Kit-> Driver Examples->>TI Driver Examples->GPIO Examples,* and select GPIO Interrupt



GPIO Interrupt

An example using interrupts to toggle an LED after a button press

These are the steps to import the project, build the project, and debug the project.

Step 1: Import the example project into CCS

*Click on the link above to import the project. The imported project is available in the **Project Explorer** view, expand the project node to browse the imported source files. To modify source code, double clicks on the source file within the project to open the source file editor.*

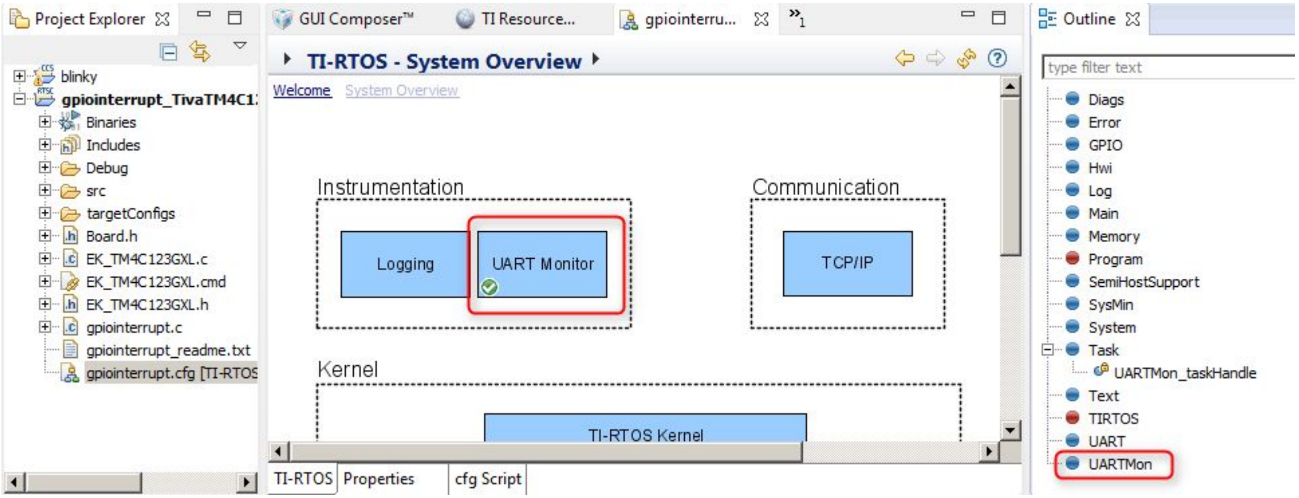
Import and Build ‘GPIO Interrupt’ Project

3. In the right-pane, click on Step1 to **Import the example project into CCS**
4. Click on Step2 to **Build the imported project**
5. Click on Step 3 for **Debugger Configuration** and select **Stellaris In-Circuit Debug Interface** as the Connection

Review the RTOS configuration

1. In *Project Explorer* view, expand the project **gpiointerrupt_TivaTM4C123GH6PM**
2. Double-click on **gpiointerrupt.cfg** to open it
3. Click on *System Overview*

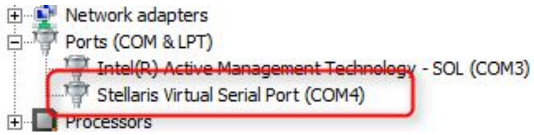
   Notice that UART Monitor is enabled in the Property view as well as Outline view



This module enables the host to communicate with target device using UART. It consists of a running task that uses the TI-RTOS UART driver to respond to requests to read/write memory at specified addresses on the target.

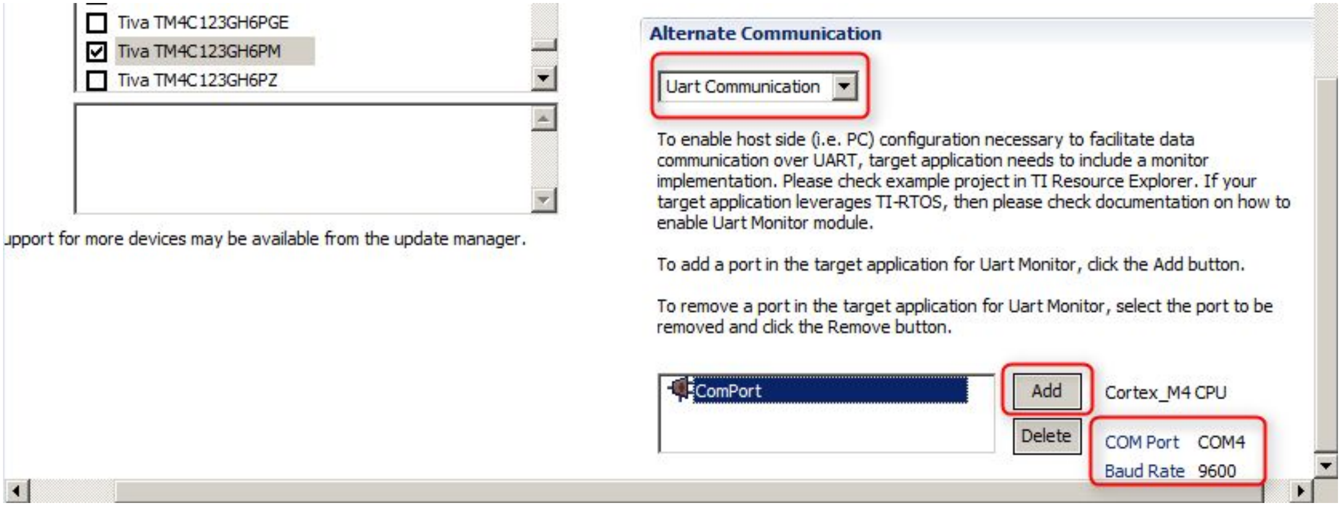
Add UART Communication to target config

1. Find the COM Port number for your device using Device Manager



Network adapters
Ports (COM & LPT)
Intel(R) Active Management Technology - SOL (COM3)
Stellaris Virtual Serial Port (COM4)
Processors

2. In *Project Explorer* view, expand **targetConfigs** folder and double-click on **Tiva TM4C123GH6PM.ccxml** to open it
3. Under Alternate Communication, select UART Communication and click the Add button to add a port for the target to listen to
4. Click on ComPort and adjust the COM Port number there to match the one your target is using. Leave the Baud Rate setting as 9600



☐ Tiva TM4C123GH6PGE
☑ Tiva TM4C123GH6PM
☐ Tiva TM4C123GH6PZ

upport for more devices may be available from the update manager.

Alternate Communication

Uart Communication

To enable host side (i.e. PC) configuration necessary to facilitate data communication over UART, target application needs to include a monitor implementation. Please check example project in TI Resource Explorer. If your target application leverages TI-RTOS, then please check documentation on how to enable Uart Monitor module.

To add a port in the target application for Uart Monitor, click the Add button.

To remove a port in the target application for Uart Monitor, select the port to be removed and click the Remove button.

ComPort

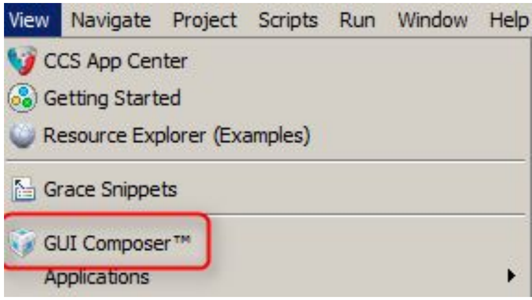
Cortex_M4 CPU

COM Port COM4
Baud Rate 9600

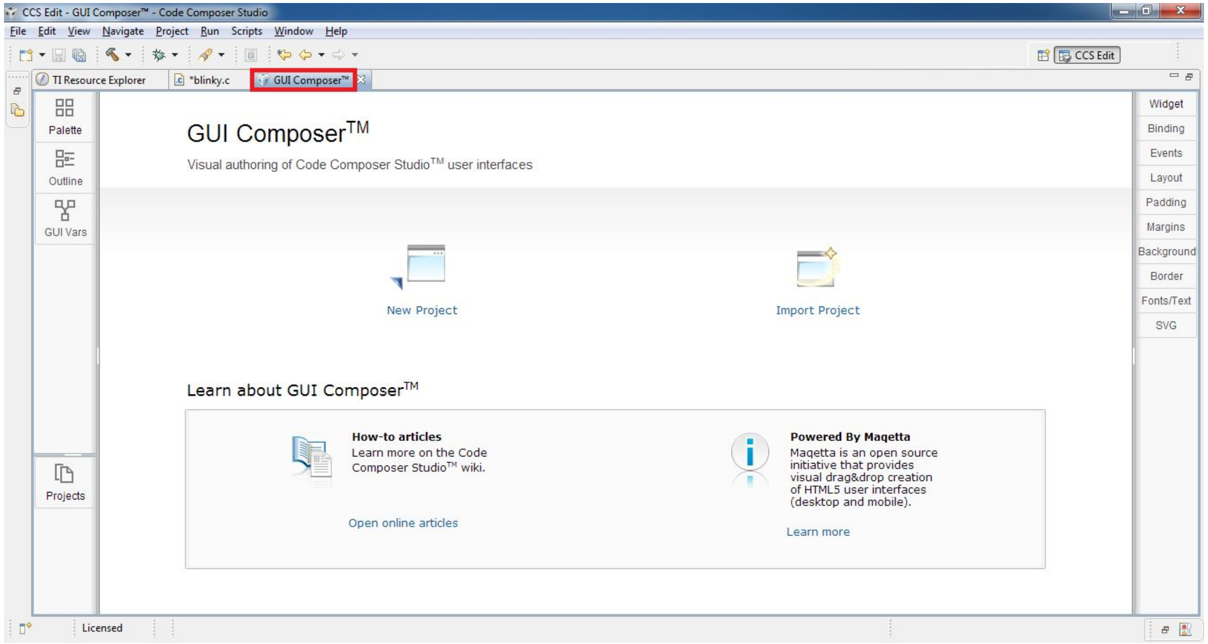
5. Click on Save to save the ccxml file

Open GUI Composer

1. Go to *GUI Composer* view if it is already open, else open the view from menu *View -> GUI Composer*



2. Double-click on the GUI Composer tab to maximize it



GUI Composer^{TM}

Visual authoring of Code Composer Studio^{TM} user interfaces

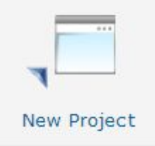
Learn about GUI Composer^{TM}

How-to articles
Learn more on the Code Composer Studio^{TM} wiki.

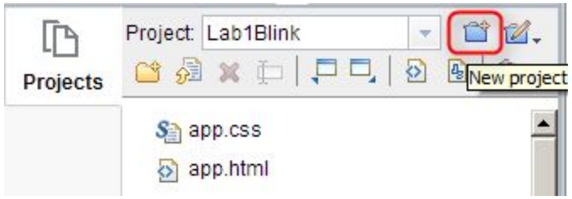
Powered By Maqetta
Maqetta is an open source initiative that provides visual drag&drop creation of HTML5 user interfaces (desktop and mobile).

Create a New GUI Composer Project

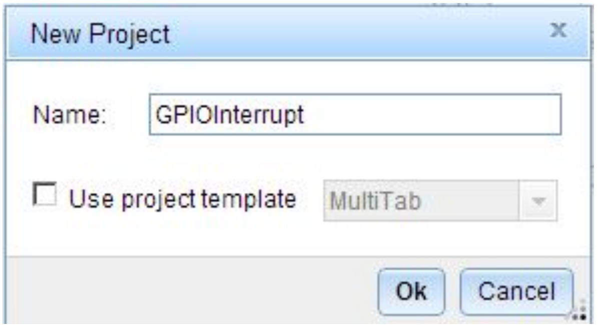
1. If GUI Composer is being opened for the first time, click on the *New Project* button



2. If GUI Composer has been previously opened , click on New Project button in the Projects view


Project: Lab1Blink

3. Name the project **GPIOInterrupt**


New Project

Name: GPIOInterrupt

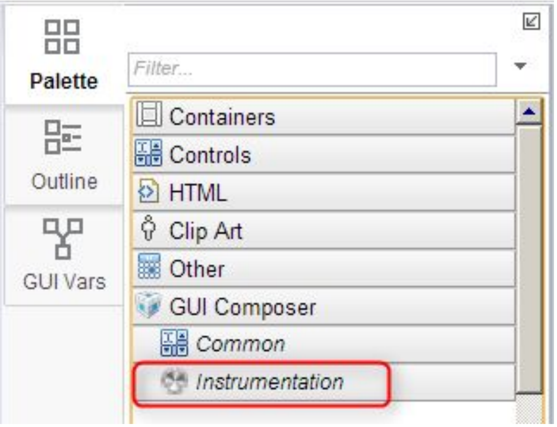
☐ Use project template MultiTab

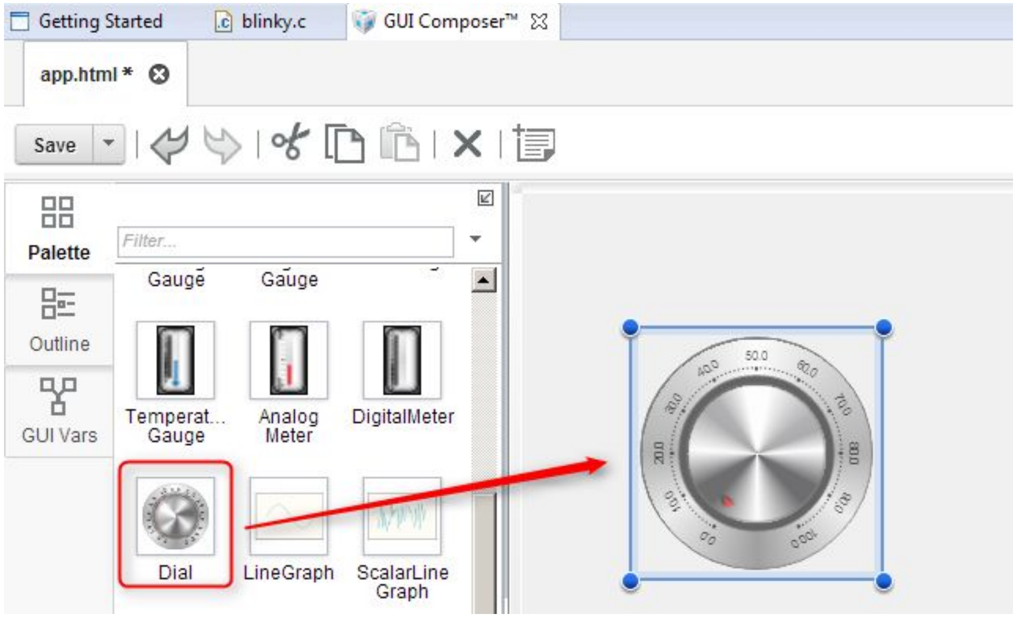
4. Click Ok

Add a Dial Widget

The Dial widget may be used to provide a user input to control a numeric variable, as well as to view the variable as it value changes

1. Go to the GUI Composer *Palette* and expand *GUI Composer->Instrumentation*
2. Click on *Dial*, hold the left mouse button down and drag it onto the canvas in the middle of the screen and release the button





Change the Appearance of the Dial

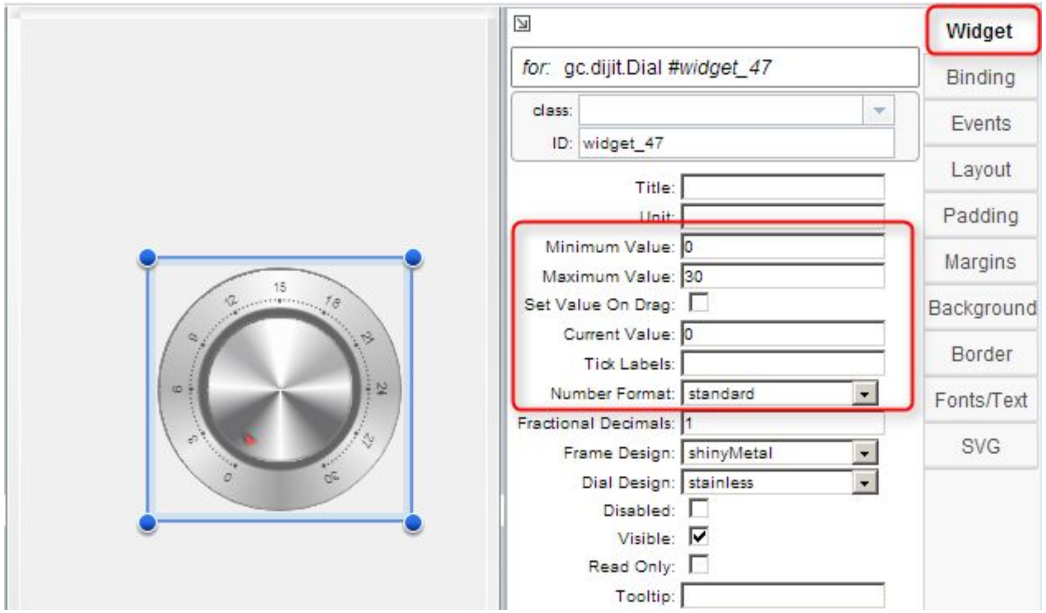
1. Select the Dial on the canvas

There should be a blue square around it

2. Click on *Widget* on the right

This will display some properties of the selected Widget

3. Set the following
   - Minimum value = 0
   - Maximum value = 30
   - Number Format = standard



for: gc.dijit.Dial #*widget_47*

class:

ID: widget_47

Title:

Unit:

Minimum Value: 0

Maximum Value: 30

Set Value On Drag: □

Current Value: 0

Tick Labels:

Number Format: standard

Fractional Decimals: 1

Frame Design: shinyMetal

Dial Design: stainless

Disabled: □

Visible: ☑

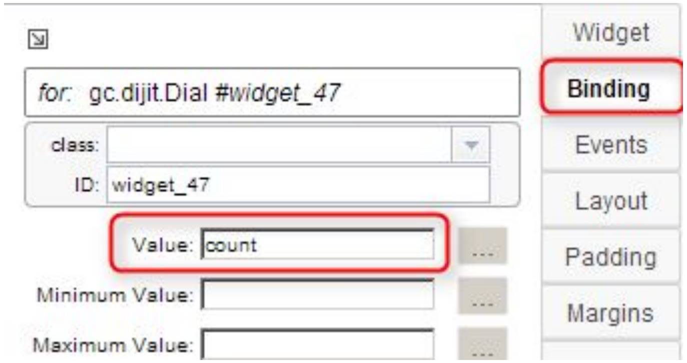
Read Only: □

Tooltip:

Bind a Variable to the Dial

1. Make sure the Dial is selected
2. Click on *Binding* on the right
3. In *Value:* field, add *count*

This binds the variable “count” to the Dial widget



for: gc.dijit.Dial #widget_47

class:

ID: widget_47

Value: count

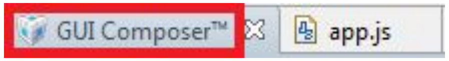
Minimum Value:

Maximum Value:

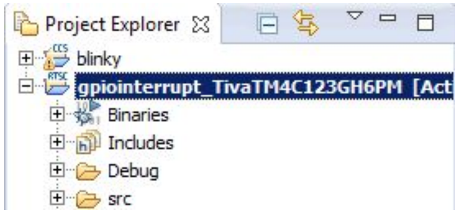
4. Click on the *Save* button at the top of the GUI Composer view

Load Target Application

1. If the *Project Explorer* view is not visible double-click on the GUI Composer tab to restore it to its normal size



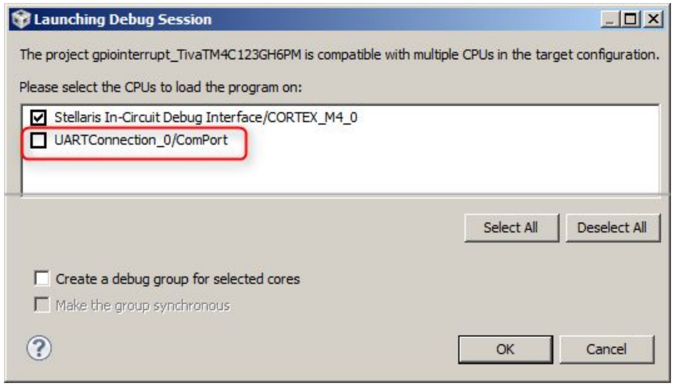
2. Select **gpiointerrupt_TivaTM4C123GH6PM** in the *Project Explorer* view



3. Click the bug button to debug the project

This will build the project (if required), launch the debugger, flash the program onto the device and run to main()

If this is the first time launching the debugger for this target configuration, this pop-up will appear



The project gpiointerrupt_TivaTM4C123GH6PM is compatible with multiple CPUs in the target configuration.

Please select the CPUs to load the program on:

☑ Stellaris In-Circuit Debug Interface/CORTEX_M4_0

□ UARTConnection_0/ComPort

□ Create a debug group for selected cores

□ Make the group synchronous

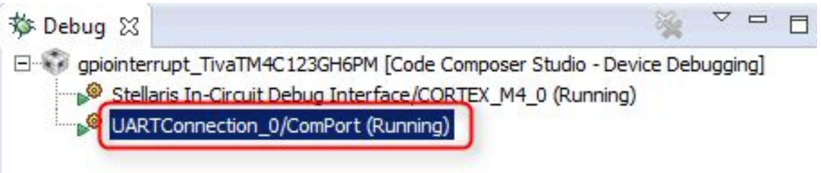
4. De-select UARTConnection_0/ComPort and click OK

This will load the program through Stellaris ICDI JTAG

Load Symbols for UART Connection

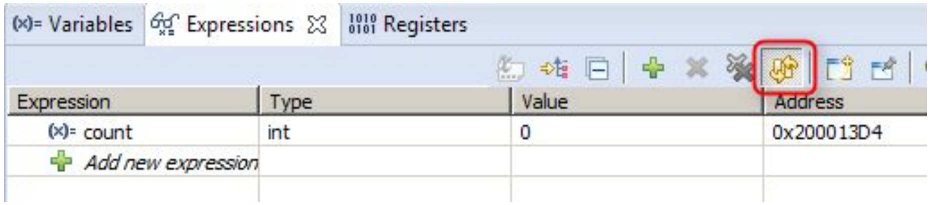
Once the program is running, load symbols for the UART connection

1. In the *Debug* view, click the Run button to run the code
2. In the *Debug* view, click on *UARTConnection_0/ComPort (Running)*



gpiointerrupt_TivaTM4C123GH6PM [Code Composer Studio - Device Debugging]
Stellaris In-Circuit Debug Interface/CORTEX_M4_0 (Running)
UARTConnection_0/ComPort (Running)

3. Go to menu *Run->Load->Load Symbols*, select the gpiointerrupt_TivaTM4C123GH6PM.out file and click OK
4. Go to the *Expressions* view and add **count.** Delete other variables that may be in the *Expressions* view



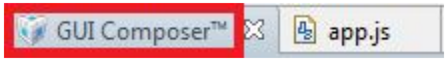
Expression	Type	Value	Address
count	int	0	0x200013D4
Add new expression			

5. Click the *Continuous Refresh* button

This will allow CCS to periodically read and display the value of **count** as the program runs

Preview the App

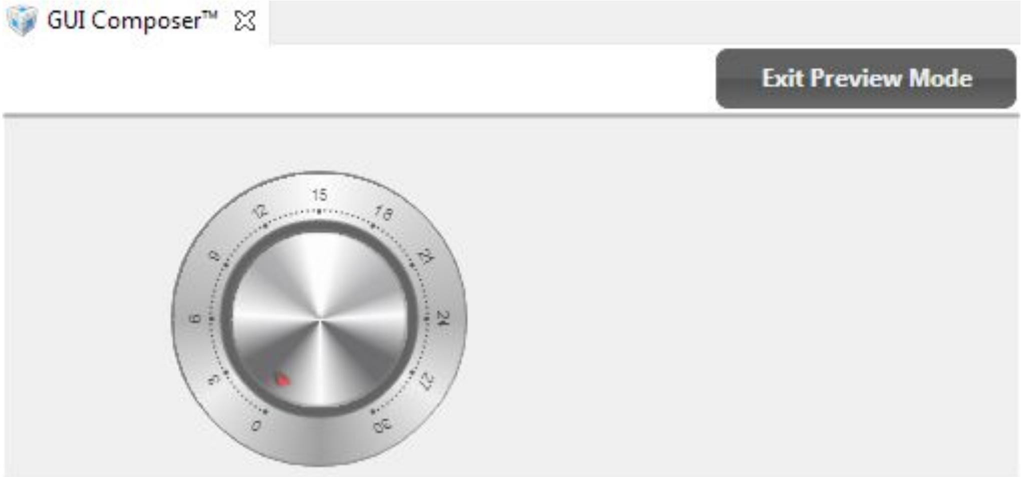
1. Click on GUI Composer tab in the editor



2. Click on the *Preview* button ▶ at the top right

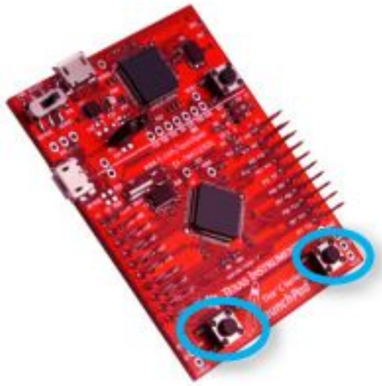
ⓘ Since the symbols are already loaded for the UART Connection, the preview mode will allow you to use the widgets. In this case, the dial widget should be visible

⚠ Note: If there are errors or symbols are not loaded a red X will appear



Test the App

1. Press the SW1 or SW2 buttons on the Launchpad



2. With each button press, observe the following:
   - LEDs on Launchpad toggle (different LED for each button)
   - In *Expressions* view, value of **count** increases
   - In the GUI, the value of count is reflected in the Dial
3. Modify or reset the value of count using the Dial and observe the value change in the *Expressions* view as well

Clean Up

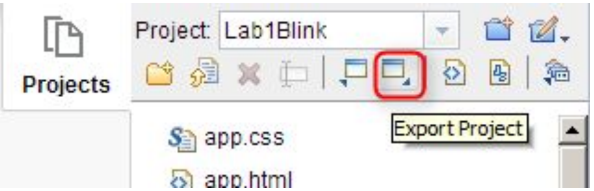
1. Click on the *Exit Preview Mode* button

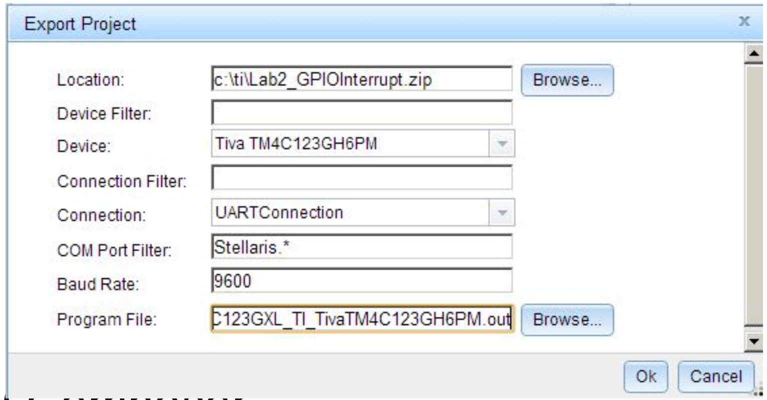
2. Click the Terminate button 🟥 on the *Debug View* to close the debug session

ⓘ CCS will shutdown the debugger and return to the CCS Edit perspective

Exporting the GUI Application

1. Click on the *GUI Composer* view
2. In the *Projects* area click on the *Export Project* button
3. Specify the following:
   - Location: *C:\ti\Lab2_GPIOInterrupt.zip* (location for saving exported project)
   - Device: *Tiva TM4C123GH6PM*
   - Connection: *UART Connection*
   - COM Port Filter: *Stellaris.**
   - Baud rate: *9600*
   - Program File: *C:\Users\<username>\GUI Composer Workshop\gpiointerrupt_EK_TM4C123GXL_TI_TivaTM4C123GH6PM\Debug\gpiointerrupt_EK_TM4C123GXL_TI_TivaTM4C123GH6PM.out* (browse to location of program file)
4. Click Ok




Export Project

Location: c:\ti\Lab2_GPIOInterrupt.zip Browse...

Device Filter:

Device: Tiva TM4C123GH6PM

Connection Filter:

Connection: UARTConnection

COM Port Filter: Stellaris.*

Baud Rate: 9600

Program File: C123GXL_TI_TivaTM4C123GH6PM.out Browse...

Ok Cancel

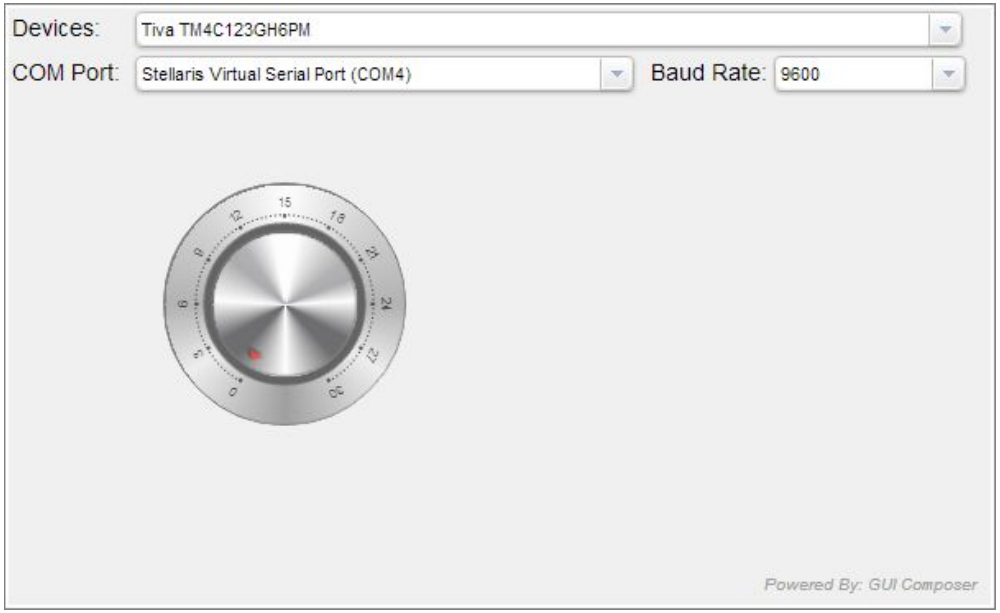
Add App to GUI Composer Runtime

1. Open a file explorer window
2. Go to c:\ti
3. Right click on Lab2_GPIOInterrupt.zip
4. Select Extract All
5. Extract the files to c:\ti\guicomposer\webapps

Run the Standalone Application

1. Close CCS
2. Power cycle the Launchpad
3. Double click on *Launcher.exe* located in c:\ti\guicomposer\webapps\Lab2_GPIOInterrupt

The splash screen will appear then it will establish UART connection with the device, and the GUI app will come up



Devices: Tiva TM4C123GH6PM

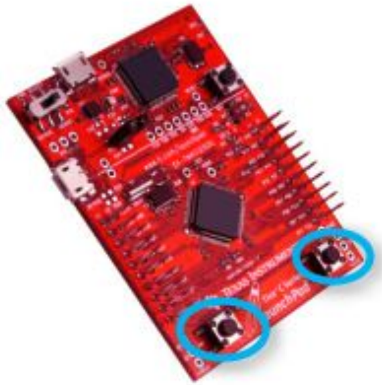
COM Port: Stellaris Virtual Serial Port (COM4)

Baud Rate: 9600

Powered By: GUI Composer

Test the Standalone Application

1. Press the SW1 or SW2 buttons on the Launchpad



2. With each button press, observe the value of count reflected in the Dial in the GUI app

   ⓘ The data is being sent from target to GUI app using UART

3. Close the application window when done

Exercise Summary

- After completing the labs you should be familiar with:
  - Using GUI Composer to create widgets for controlling and visualizing target variables
  - Running GUI Composer app from within CCS or standalone
  - Using JTAG and UART transport for viewing and controlling the application through GUI composer

- Additional References:
  - GUI Composer wiki: http://processors.wiki.ti.com/index.php/Category:GUI_Composer