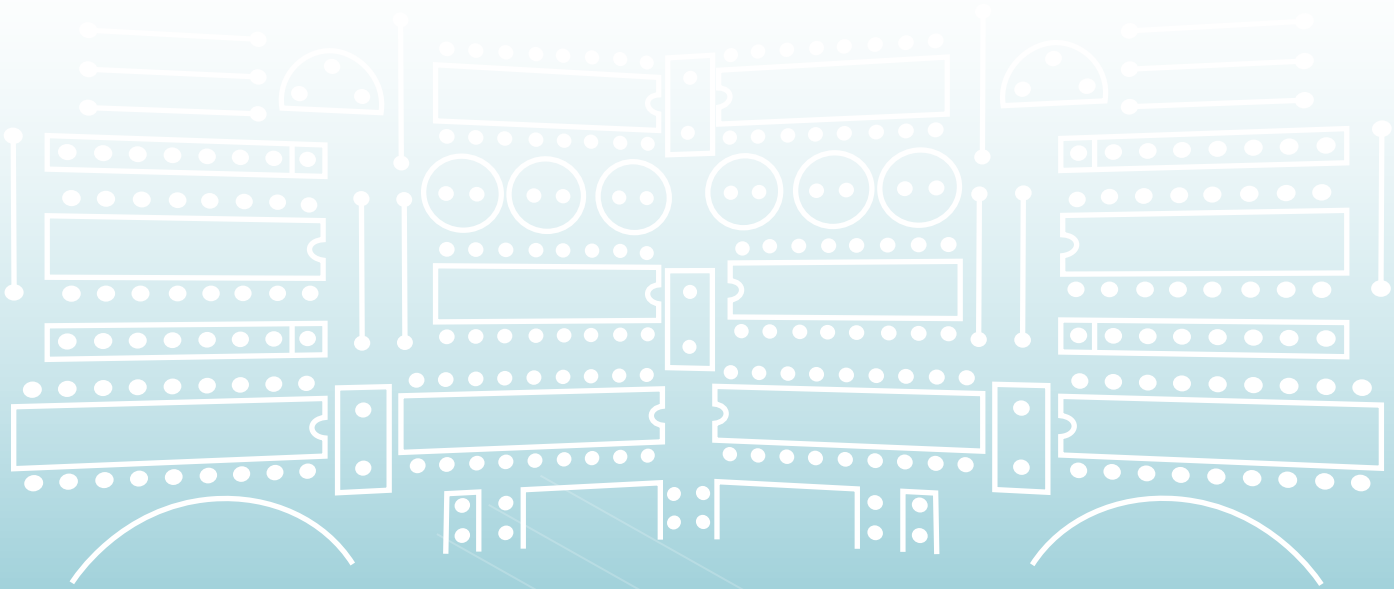


# PROTEUS DESIGN SUITE

## IoT Builder Help



# COPYRIGHT NOTICE

## © Labcenter Electronics Ltd 1990-2019. All Rights Reserved.

The Proteus software programs (Proteus Capture, PROSPICE Simulation, Schematic Capture and PCB Layout) and their associated library files, data files and documentation are copyright © Labcenter Electronics Ltd. All rights reserved. You have bought a licence to use the software on one machine at any one time; you do not own the software. Unauthorized copying, lending, or re-distribution of the software or documentation in any manner constitutes breach of copyright. Software piracy is theft.

PROSPICE incorporates source code from Berkeley SPICE3F5 which is copyright © Regents of Berkeley University. Manufacturer's SPICE models included with the software are copyright of their respective originators.

The Qt GUI Toolkit is copyright © 2012 Digia Plc and/or its subsidiary(-ies) and licensed under the LGPL version 2.1. Some icons are copyright © 2010 The Eclipse Foundation licensed under the Eclipse Public Licence version 1.0. Some executables are from binutils and are copyright © 2010 The GNU Project, licensed under the GPL 2.

## WARNING

You may make a single copy of the software for backup purposes. However, you are warned that the software contains an encrypted serialization system. Any given copy of the software is therefore traceable to the master disk or download supplied with your licence.

Proteus also contains special code that will prevent more than one copy using a particular licence key on a network at any given time. Therefore, you must purchase a licence key for each copy that you want to run simultaneously.

## DISCLAIMER

No warranties of any kind are made with respect to the contents of this software package, nor its fitness for any particular purpose. Neither Labcenter Electronics Ltd nor any of its employees or sub-contractors shall be liable for errors in the software, component libraries, simulator models or documentation, or for any direct, indirect or consequential damages or financial losses arising from the use of the package.

Users are particularly reminded that successful simulation of a design with the PROSPICE simulator does not prove conclusively that it will work when manufactured. It is always best to make a one off prototype before having large numbers of boards produced.

Manufacturers' SPICE models included with PROSPICE are supplied on an 'as-is' basis and neither Labcenter nor their originators make any warranty whatsoever as to their accuracy or functionality



# TABLE OF CONTENTS

<b>COPYRIGHT NOTICE</b> .....	1
<b>TABLE OF CONTENTS</b> .....	1
<b>IoT BUILDER TUTORIAL</b> .....	1
What is IOTBuilder?	1
IOTBuilder Targets	2
Guided Tour	4
The Project Tree	5
The Editing Window	7
The Graphics Panel	8
The Colour Palette	9
Zoom and Navigation	10
Grids and Snap	11
<b>TUTORIAL 1: Blink an LED</b> .....	1
Introduction	1
Project Setup	2
Design the Front Panel	3
Writing the Program	9
Simulate and Test	13
The App	15
Programming	18
Controlling the Hardware	21
<b>TUTORIAL 2: LOGGING THERMOMETER</b> .....	1
Introduction	1
Front Panel Design	1
Writing the Firmware (Flowchart)	10
Simulation	24
Debug	27

Deployment	33
<b>SOURCE CODE PROJECTS .....</b>	<b>1</b>
Introduction	1
New Project Wizard	1
Design Phase	2
<b>PROGRAMMING THE HARDWARE .....</b>	<b>1</b>
Overview	1
Raspberry Pi 3	1
Arduino Yun via SSH	1
Arduino Yun via USB	3
Computer / Network / Lab Setup	4
Setting The Arduino Yun On To Your Network	6
<b>THE MOBILE APP .....</b>	<b>1</b>
Overview	1
Download and Installation	1
Discovery	1
<b>ADVANCED PANEL DESIGN .....</b>	<b>1</b>
How a Virtual Front Panel Works	1
Using Inkscape to Edit the Panel	1
Control Editing	3
<b>IoT Controls .....</b>	<b>1</b>
BUTTONS	1
To add a button	1
Design Time Configuration	2
Programming Methods	6
SWITCHES	7
To add a switch	7
Design Time Configuration	8
Programming Methods	11
DISPLAYS	12
To add a display control	13
Programming Methods	18

## Visual Designer

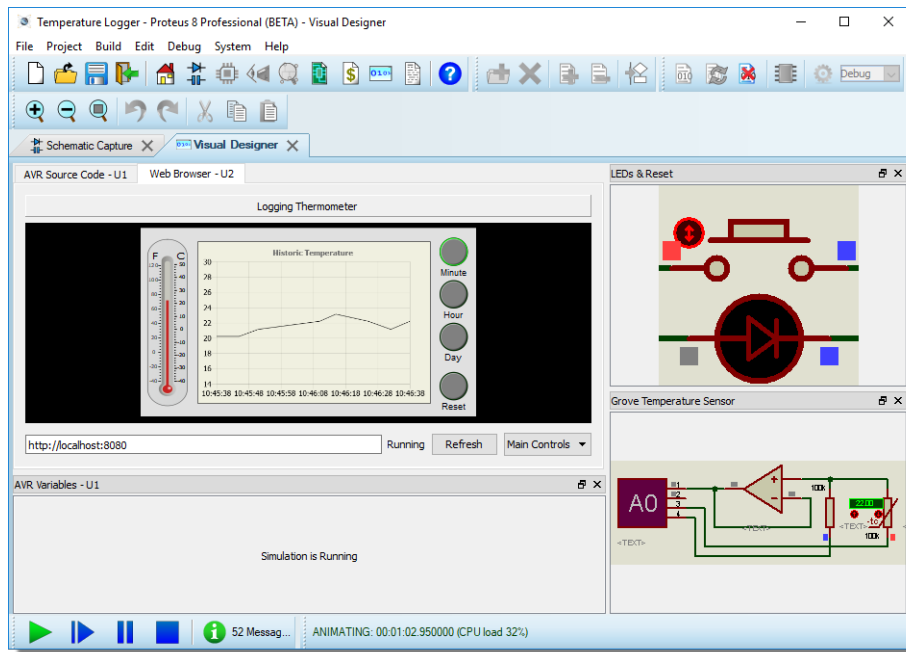
INDICATORS	19
To add an indicator	19
Programming Methods	21
DIALS AND SLIDERS	21
To add a dial/slider	21
Design Time Configuration	22
Programming Methods	24
ALERTS	25
TEXT BOXES AND TERMINALS	27
Text Input Control	28
Teletype Terminal Control	28
Text Log Control	29
<b>Advanced Controls .....</b>	<b>Error! Bo</b>
CLOCKS AND TIMERS	1
Understanding Time	2
Clock Properties	4
Timer Properties	5
LINE CHART	7
Design Time Properties	7
Programming Methods	8
BAR CHART	9
Design Time Configuration	9
Programming Methods	10
HISTOGRAM	12
To add a histogram control:	12
WIND ROSE CONTROL	14
Programming Methods	17



# IoT BUILDER TUTORIAL

## What is IOTBuilder?

IOTBuilder is a product that works with Proteus Visual Designer for Arduino and Visual Designer for Raspberry Pi to allow the development of remote user interfaces to embedded design products based on the Arduino Yun and Raspberry Pi 3 hardware.



The basic workflow is:

1. Create a Visual Designer (flowchart) project or a Proteus VSM source code project with an IOT enabled target such as Arduino Yun or Raspberry Pi 3.
2. Add the electronic shields, sensors and breakout boards to the schematic via the Visual Designer peripheral gallery.
3. Add IOT Controls (dials, buttons, displays, etc.) to the Virtual Front Panel and then lay out the user interface. All of these controls hook into the main program using simple Visual Designer flowchart methods making it quick and easy to program.
4. Write the program. All of the shields and controls discussed in the steps above present as simple flowchart methods (Visual Designer) or as drag and drop functions (Proteus VSM) making it simple and fun to develop.
5. Simulate and debug. Press play and interact with your front panel and your virtual hardware on the schematic. Set breakpoints and single step debug.



6. Deploy. Once everything is proven correct in Proteus use the programmer to deploy to the hardware. Then use your phone, tablet or browser to control your hardware appliance.

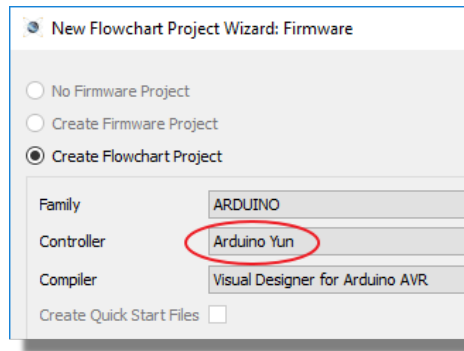
## **IOTBuilder Targets**

---

While other target platforms are in development IoT Builder is currently targeted at Arduino and Raspberry Pi. Arduino is supported via both the Arduino Yun board and also through the Arduino Yun shield or ESP8266 breakout board.

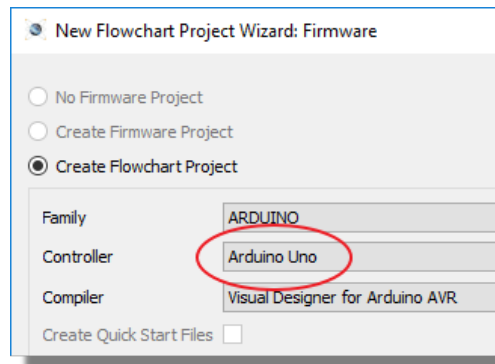
### Arduino Yun

From the homepage select new flowchart project and specify Arduino Yun as the target.

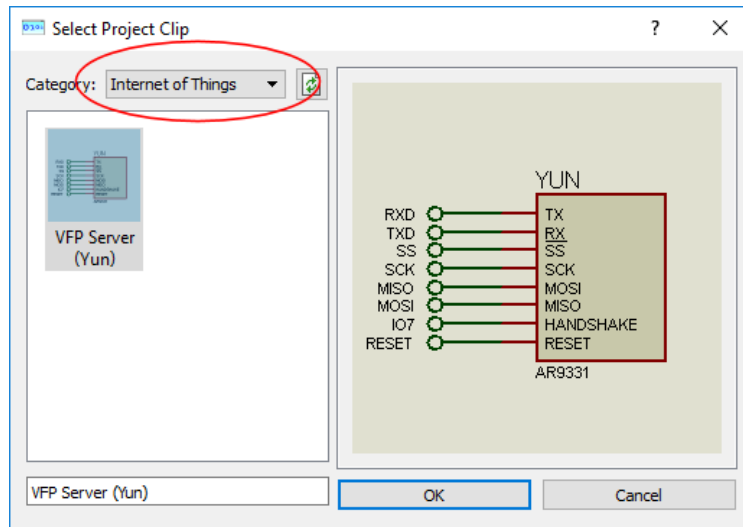


### Arduino Yun Shield

From the homepage select new flowchart project and specify either **Arduino Uno** or **Arduino Mega** as the target



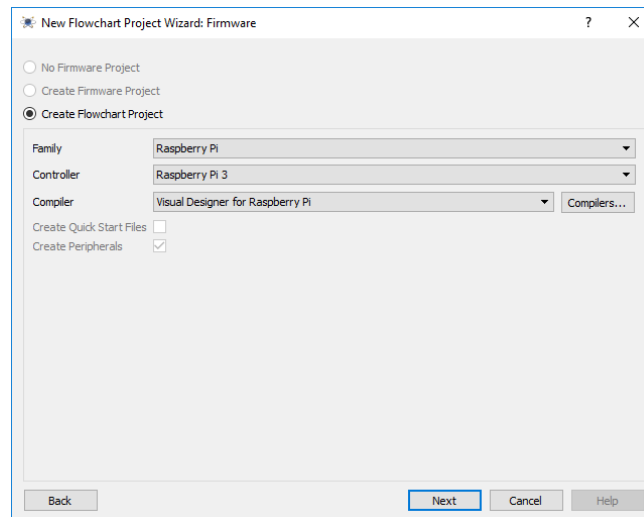
In the flowchart module select Add Peripheral from the Project menu, navigate to Internet of Things Peripherals and add the Yun shield.



- Using an Arduino Yun baseboard is simpler but the Yun shield or ESP8266 tends to have better wifi and allows you to couple to a Mega which has more program ROM.

### Raspberry Pi 3

From the homepage select new flowchart project and specify Raspberry Pi 3 as the target.



- Raspberry Pi Zero should work (but has not been tested) as it shares the same header as the Raspberry Pi 3. Raspberry Pi 2 will definitely not work with Proteus.

The procedure above is slightly different if you are planning to create a source code project rather than a flowchart project as you will need to start with 'New Project' instead of 'New Flowchart Project'.

### About this Documentation

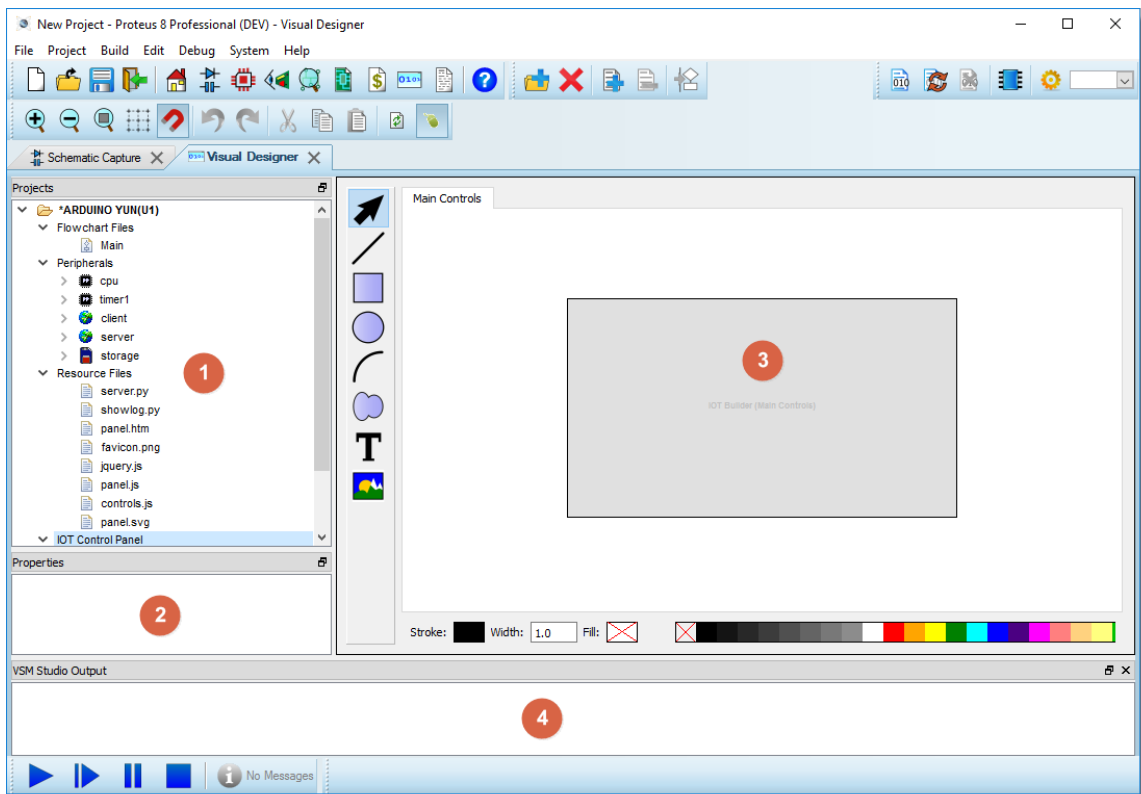
This manual is intended to cover the use of the IOT Builder module in Proteus. IOT Builder is an add-on product to Visual Designer and so reference will be made throughout to Visual Designer programming methods. These should be easily understood in context but is explained in the Visual Designer Documentation.

It is also entirely possible to use IoT Builder as an add-on to Proteus VSM for Arduino. This is discussed in More Details again in the Visual Designer help files or documentation.

## Guided Tour

---

This topic covers in brief the various control sections of the front panel editor in IOT Builder. The front panel editor is accessible from the project tree when one or more IOT Controls are added to the current project.



1. Project Tree / File structure
2. Control Properties
3. Editing Window
4. Output Window

## The Project Tree

The project tree sits on the left hand side of the editor. In the context of IOT Builder it serves four main purposes:

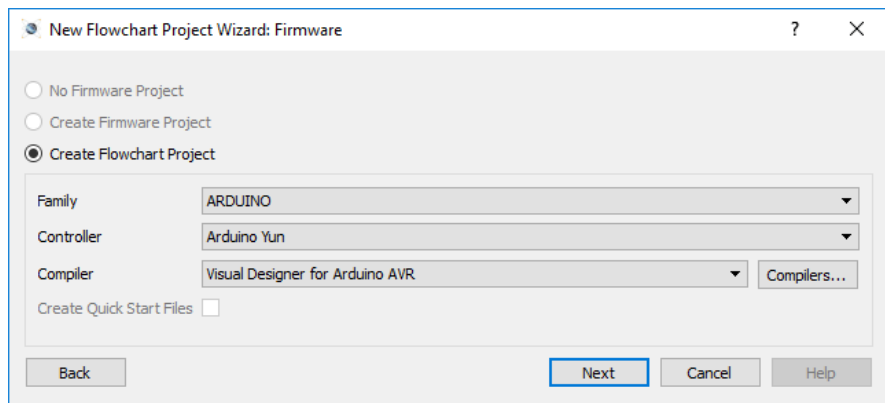
- 1) Switch between panel design and flowchart program.
- 2) Context menu command set for adding IOT controls to the front panel.
- 3) Drag and drop placement of controls on the front panel.
- 4) Configuration of controls from the property panel at the bottom.

The following short example illustrates these points.

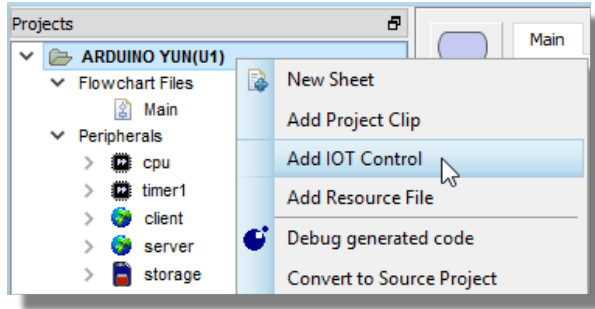
### Adding a button

The following procedure uses an Arduino Yun target but the process is the same regardless of target so you could also follow these steps with targets such as Raspberry Pi.

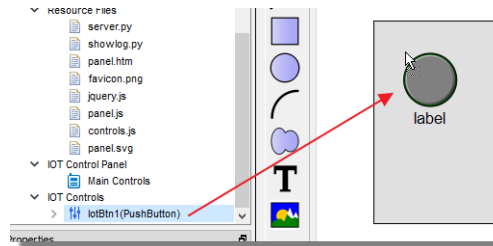
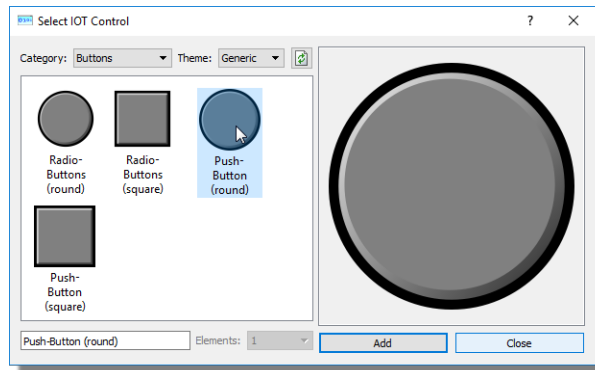
Start a new flowchart project and select Arduino Yun as the target device.



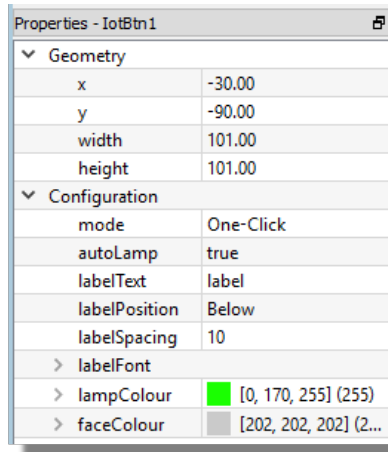
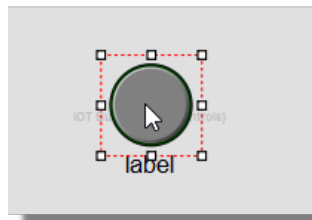
Right click on the Yun in the Project Tree and select 'Add IOT control' from the context menu.



Select a button from the resulting dialogue and then drag and drop it onto the front panel editor.



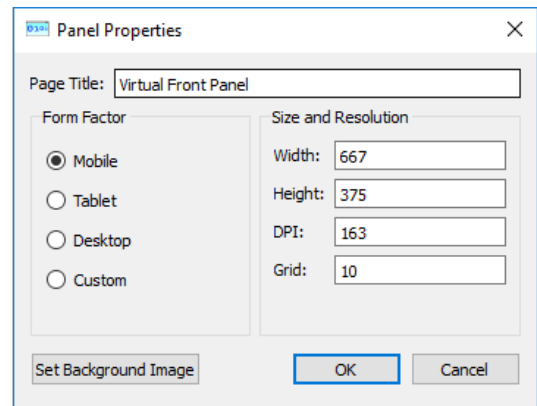
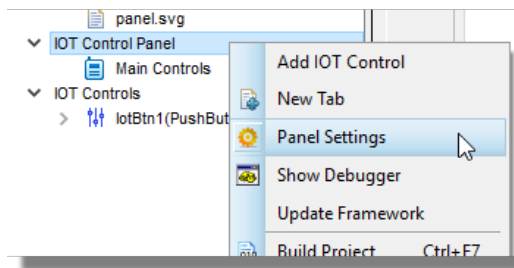
Left click to select the button and then edit it's properties from the property page at the bottom of the project tree.



## The Editing Window

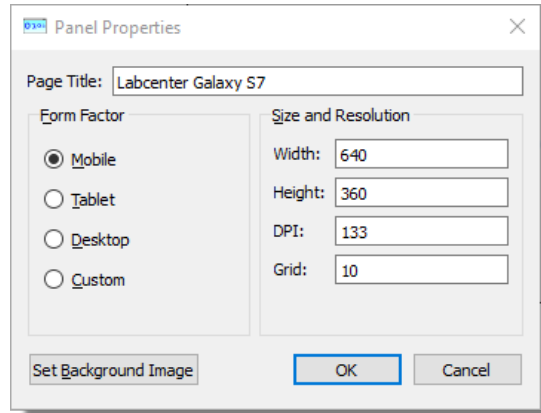
The drawing area for the front panel is determined by the target device (desktop browser, iPad, iPhone, Galaxy S6, etc.). To set or change the target device:

1) Right click on IOT Panel in the Project Tree and select Panel Settings from the resulting context menu.



**i** The IOT Panel becomes available in the project tree as soon as you add an IOT control to your project. You can do this via the add IOT Control command on the Project Menu or through the right click context menu.

2) Select the form factor on the left of the dialogue form and then the size and resolution for the device on the right hand side. The values to be input here are the CSS width and height (CSS Device Independent Pixels).



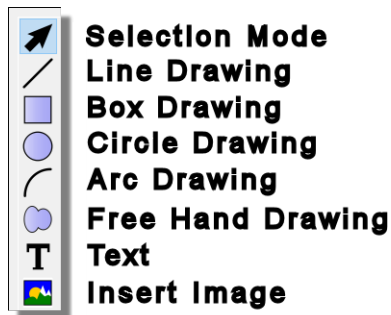
④ You can find a fairly exhaustive list on this website: <https://www.mydevice.io/#compare-devices>

3) Close the dialogue form to resize your panel to the target device. Note that if you do this once IOT controls have been placed and you go from a large device to a small device there may not be enough space for all of the controls you have placed. It's best to choose your target device at the beginning.

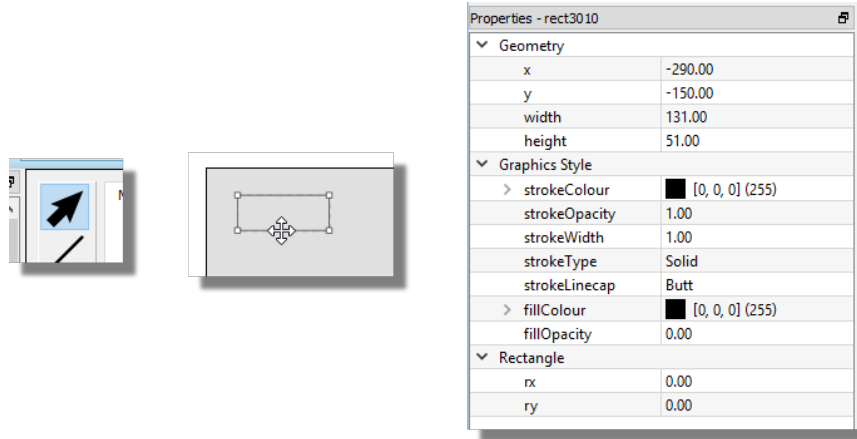
## The Graphics Panel

---

The graphics panel on the left hand side contains a selection mode and then a series of graphics and text primitives as described below:



After placing graphics, you can position or edit using the property page at the bottom of the [Project Tree](#). First enter selection mode, left click on the graphic and then edit in the property panel.

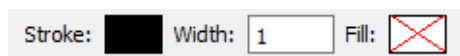


The most common use for this toolbar is to create some graphics and text for grouping other controls together on your panel.

Note that this text mode is for entering static text on the panel. If you want to input text commands on your device to send to your hardware then you need to add an IOTControl such as text input or terminal.

## The Colour Palette

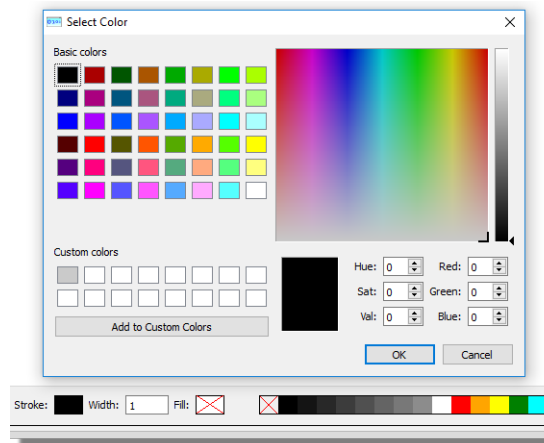
The colour palette sits at the bottom of the front panel editor and dictates the default stroke and fill colour of the next drawn graphic.



### Stroke

The stroke is the 'pen' used to draw the graphic lines. It has a colour and a thickness. You can change the colour of the stroke by double clicking on the existing colour and you can change the thickness of the pen via the edit box at the side.





## Fill

The fill specifies whether the object is wireframe (not coloured in) or in which colour it should be filled. You can change the fill by clicking on the desired colour from the palette to the right of the existing fill colour.

Graphics Style	
> strokeColour	█ [0, 0, 0] (255)
strokeOpacity	1.00
strokeWidth	1.00
strokeType	Solid
strokeLinecap	Butt
> fillColour	█ [0, 0, 0] (255)
fillOpacity	1.00

Note that changes made here will affect the default for future graphic objects. If you want to change the stroke/fill of an existing graphic you must first select it and then adjust via the property page at the bottom of the project tree.

## Zoom and Navigation

---

You can zoom in or out of the panel via the icons at the top of the application.

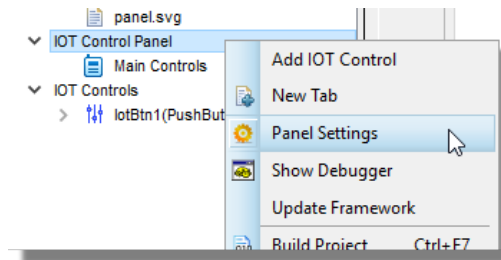


Alternatively, you can use the F6 key to zoom in, the F7 key to zoom back out and the F8 key to return the zoom level to the default view.

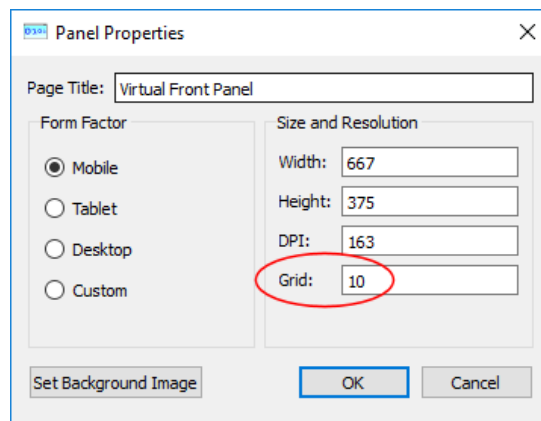
## Grids and Snap

You can configure and turn on grids to aid placement and also snap the mouse cursor to the current grid.

You can set up your grid by right clicking on the IOT Control Panel in Project Tree and selecting Panel Settings from the context menu.



The setting at the bottom right allows you to specify a grid spacing and defaults to 10.



If you set this higher you will have less grid squares on your panel which means less snap points. By contrast, if you set this lower you will have more freedom to position on the panel but it might be harder to align objects precisely. We'd suggest that you leave this at default value unless you know what you are doing.

Once configured you can turn grid and snap on or off via the icons at the top of the Panel Editor.





# TUTORIAL 1: Blink an LED

## Introduction

This tutorial is a short introduction to designing, developing and deploying a project with IOTBuilder. We've made the appliance as simple as possible to concentrate on the workflow from starting Proteus to controlling your Arduino Yun from your mobile phone.

This tutorial also assumes that you are familiar with flowchart programming with Visual Designer. If not, we recommend that you work through the tutorials in the Visual Designer reference manual linked below.

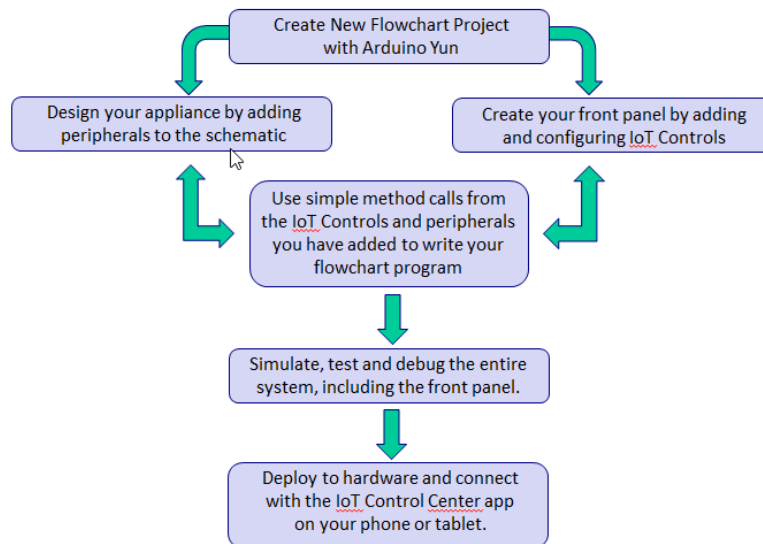
### Terminology

We refer to the **appliance** as the hardware being designed. This is 'virtual hardware' on the schematic during simulation and then physical hardware when deployed. In this tutorial, it is therefore the Arduino Yun plus any sensors, shields or breakout boards wired to it.

We refer to the **controller** as the device sending commands to the appliance. In the context of this tutorial this will be the front panel during the simulation and debug phase and the mobile phone or tablet when deployed.

### Design Process

The workflow from project start to working product involves several distinct steps as shown below but the order is fairly flexible. We'll walk through the process in the following topics.

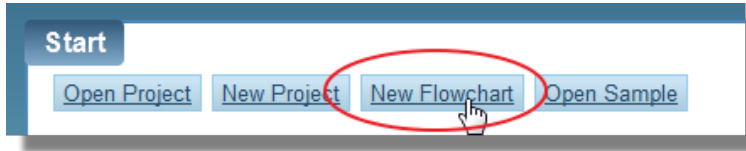


*Creating an IoT product with Arduino Yun and IoT Builder.*

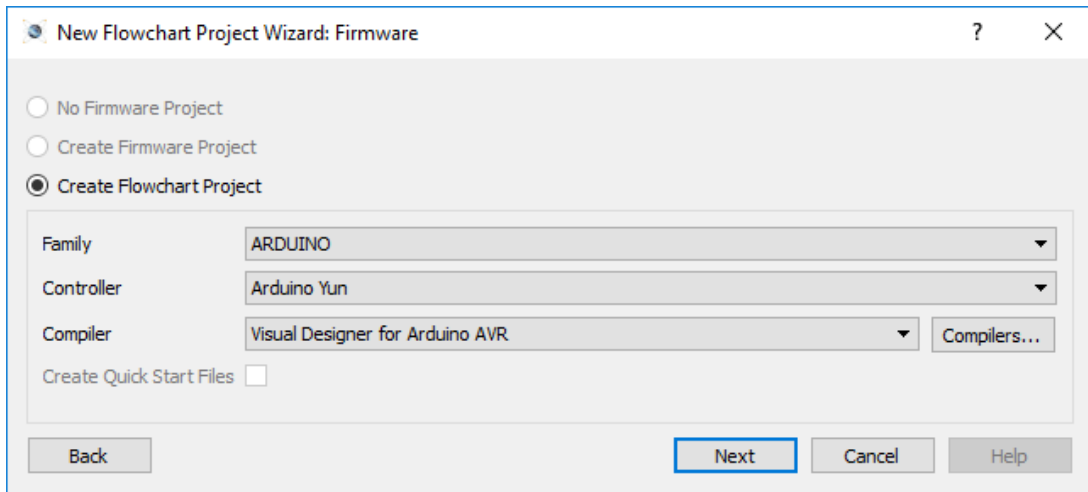
## Project Setup

---

Launch the New Flowchart Wizard from the home page in Proteus, specify your name for the project and the folder which you wish to save into.

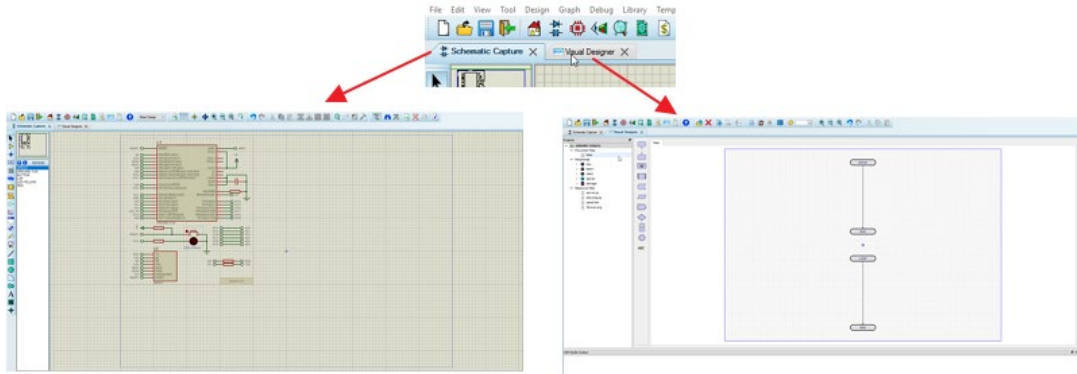


On the next screen we need to create our flowchart project. This creates the firmware which we will be using for the Arduino Yun controller in our connected appliance. This should look like the following:



We'll get a configuration summary page and can then finish to setup the project.

You should now have two tabs open, one for a schematic in which an Arduino Yun has been placed and the second in Visual Designer in which the normal skeleton program constructs for Arduino have been placed.

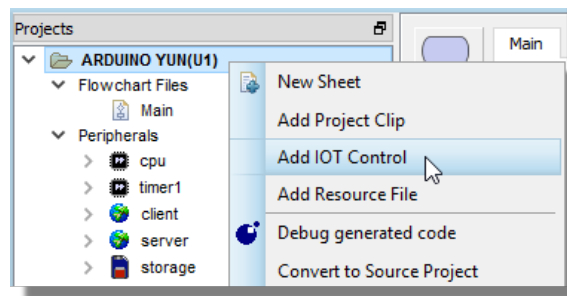


*Correct Setup Complete.*

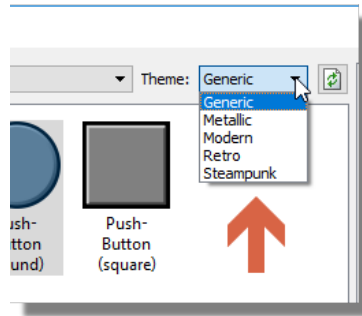
## Design the Front Panel

The front panel is the control interface to our appliance. It needs to contain the elements we need to send commands to the appliance and it contains the elements we need to display data sent from the appliance.

Since this is our 'hello world' project, we are going to turn an LED on and off from a button control on our front panel. We add internet of things controls either from the Project Menu in Visual Designer or by right clicking on the Arduino Yun text on the project tree.

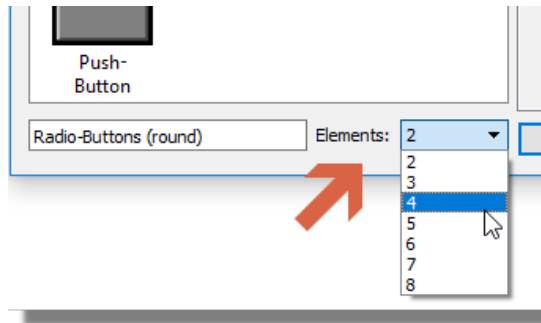


From the resulting dialogue form we need to choose a button and also a number of elements. In our case we need only one button but you could choose to have several elements in a radio button type situation. Select which theme you would like to use:



*In our case we will be using the Generic single Round button*

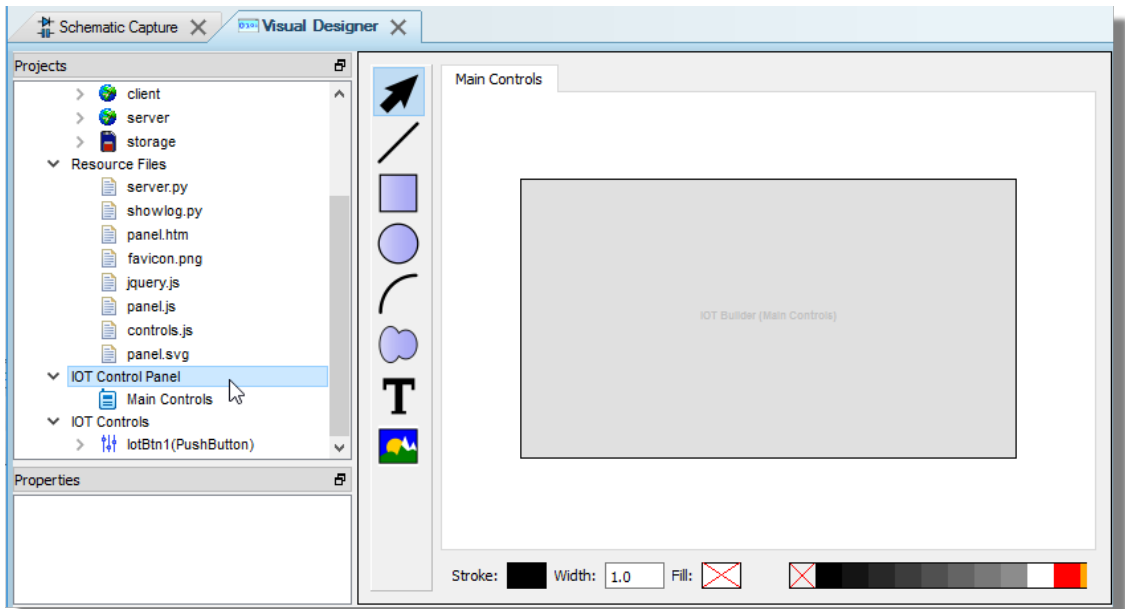
There are two options depending on the number of buttons required on your controller. A Single button (Push-Button Round/Square) or a set of button elements or a set number of buttons (Radio-Buttons Round/Square). If you select Radio-Buttons option, the elements dropdown box will be enabled for you to select the number of elements (buttons) required:



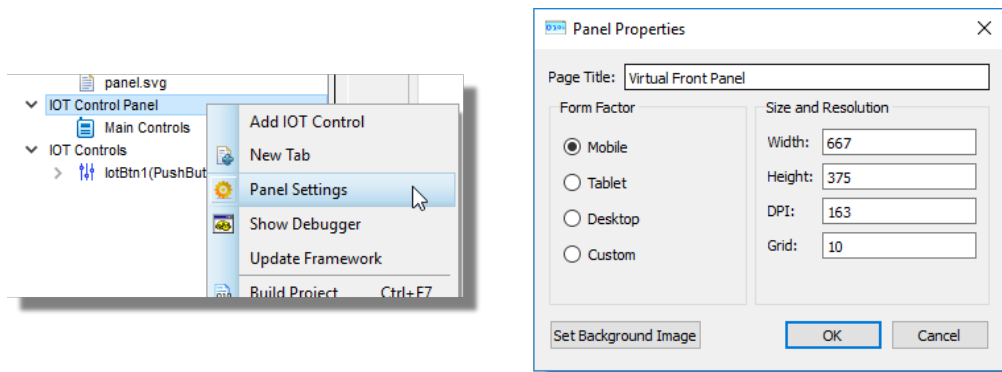
Our themes use what we call Auto Lamp which is a property of the push button control. When this is set to True, it means that the light on the button will automatically come on when it's clicked, no matter whether there is code associated to it or not. This is graphical / aesthetics only, and nothing to do with the programming.

**Note the theme selector towards the top right of the dialogue. This allows you to style your front panel in different themes such as retro, steampunk, modern, classic, etc.**

After we add the button, notice that several changes have occurred in the Project Tree. We now have an IoT Control (our button) at the bottom and also an IoT Panel and a couple more resource files. Double click on the Panel in the project tree to open in the editing window:



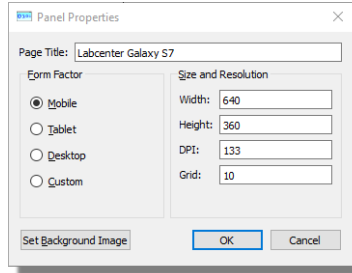
The Editing Window is now showing what will be the user interface for our appliance. By default, it is set to a mobile phone size but you can change this by right clicking on the front panel in the project tree and choosing the panel settings option from the context menu.



**i** You'll find a list of phone and tablet values here: <https://www.mydevice.io/#compare-devices>. The width and height are the CSS Width and CSS Height in this table and the DPI is the physical PPI divided by the pixel ratio. An iPhone 7 for example is therefore 375x667 and the DPI is  $326/2 = 163$ .

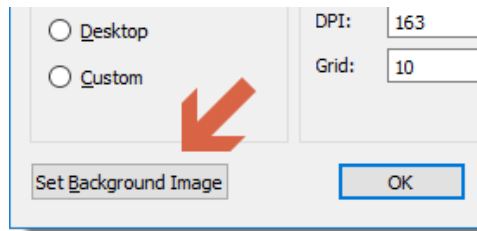
Here we've changed to a Galaxy S7, the phone that we will be using as our controller:





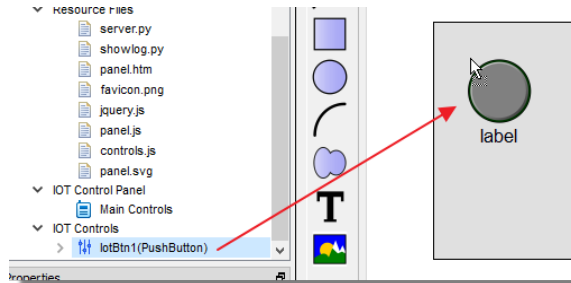
While you can change from a desktop browser to a tablet or a phone at any time it is really best to make a decision early on because any placed controls will not move if you resize the panel. This means if you resize the panel downwards (e.g. tablet to mobile) it is awkward to re-position all of the controls on the panel.

You can also change the background image. The default is a light grey box for simplification, however you may want to upgrade this to something which matches your theme. Simply download an image as a JPG or PNG file in to the Background folder and insert it using the Set Background option:

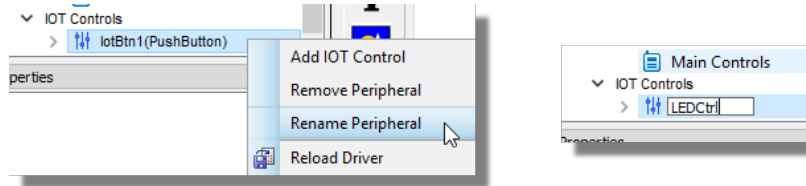


You can also change the background image. The default is a light grey box for simplification, however we have supplied several background images that match our themes. These are stored in a background folder (*C:\ProgramData\Labcenter Electronics\Proteus 8 Professional\VSM Studio\controls\Backgrounds\.*). Alternatively you may want to upgrade this to something which matches your theme. Simply download an image as a JPG, SVG or PNG file in to the Background folder and insert it using the Set Background option.

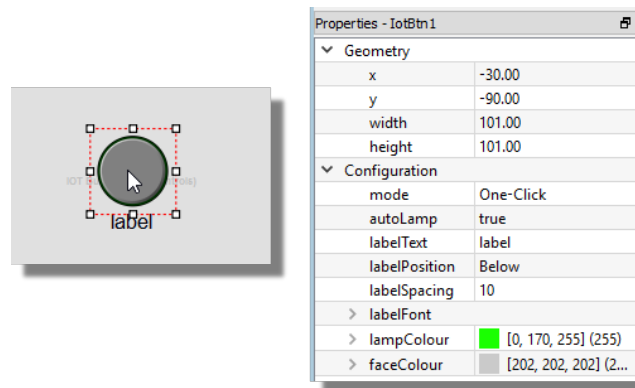
To place our button on the panel simply drag and drop from the button entry on the project tree to the panel.



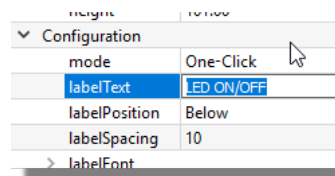
To rename the button right click on the item in the project tree and select rename peripheral from the resulting context menu.



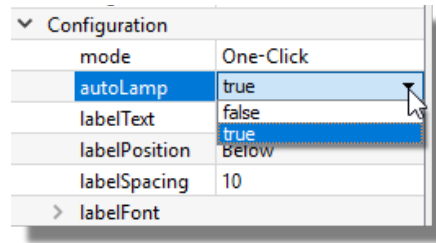
We can edit almost anything we need to with the button via the property panel underneath the project tree. This shows all of the configuration properties available for the currently selected object on the panel. If the button is not selected (does not have a dashed box around it) click left on it once and then view the device properties.



The property section for a button is fairly self-explanatory, consisting of a geometry section for accurate positioning and a configuration section. Here we can specify the type of button (single click is fine) and change the label to something sensible.

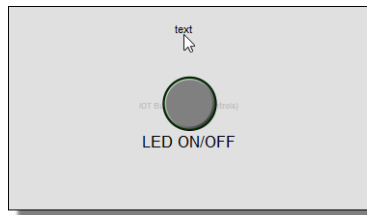


## AutoLamp Property

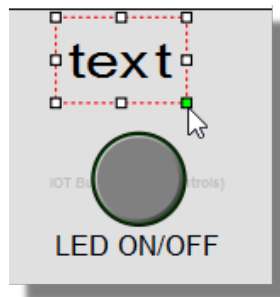


You can set an autolamp property on the IoT button. If so, it becomes a toggling button by default. However, if you turn autolamp off then you will need to use `setLamp()` on and off in the flowchart - the benefit of this being that it is hardware appliance driven rather than implicit in the GUI and therefore guaranteed to be correct. Alternatively if your program does not rely on an LED light and the on off will be for graphical purposes only, set the AutoLamp on the button to be on.

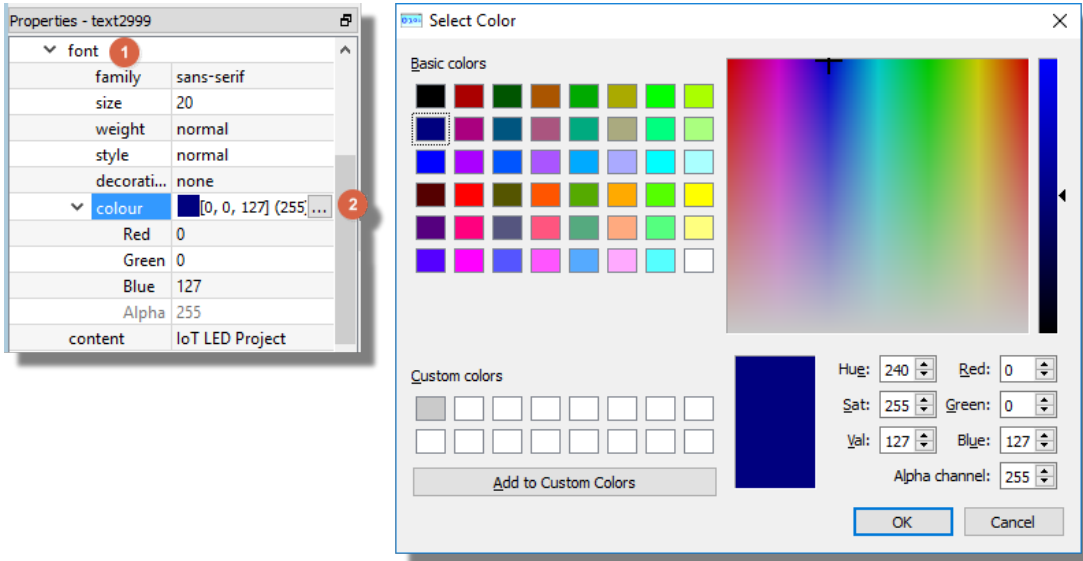
You can add title or graphics to the front panel directly from the primitives on the left hand pane. For example, to add a title we would first select text mode and then place the text indicator in approximately the right position on the panel.



After placement, we stretch the text box to the size we want.



Finally, we can change the font, colour and text content in the property page. In the left hand pane, drop the Font option down and click on the colour.



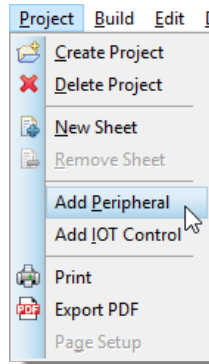
If you have Radio buttons and wish to resize them, Right click on the button > Transfer > Restore Aspect Ratio. More information is explained in the [Guided Tour](#) section.

**i** If you require more than one front panel, you can add multiple tabs by right clicking on the 'Main Controls' in the project tree and select New Tab:

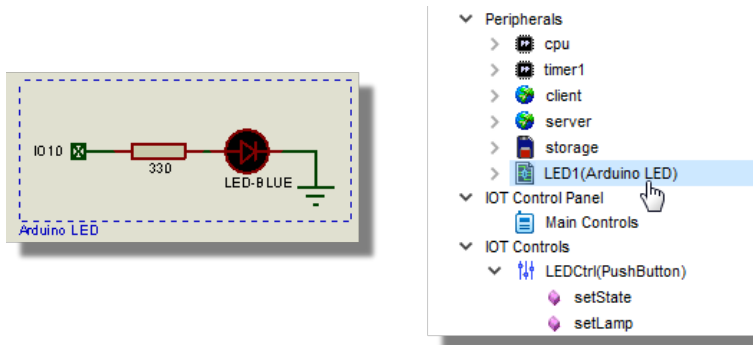


## Writing the Program

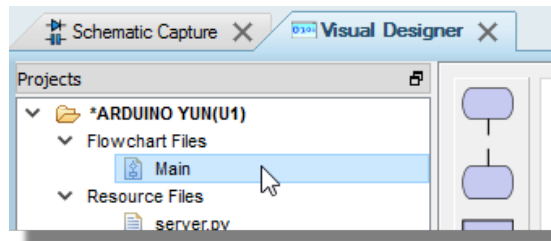
Now that we have a front panel and a placed control we can look at how this is going to interact with our appliance. Since the button is going to turn on an LED the first thing we need to do is to add an LED (whichever colour you choose) to our schematic (remember, think of the schematic as a virtual version of your hardware). We do this from the Add Project Clip command on the Project Menu.



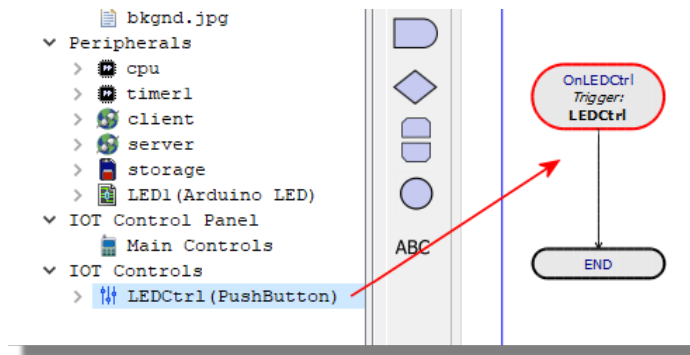
This will both add the LED (Blue, Yellow, Green or Red) to the schematic and it will appear in the project tree in Visual designer.



Now double click on the main flowchart page in the project tree to switch to the flowchart program.

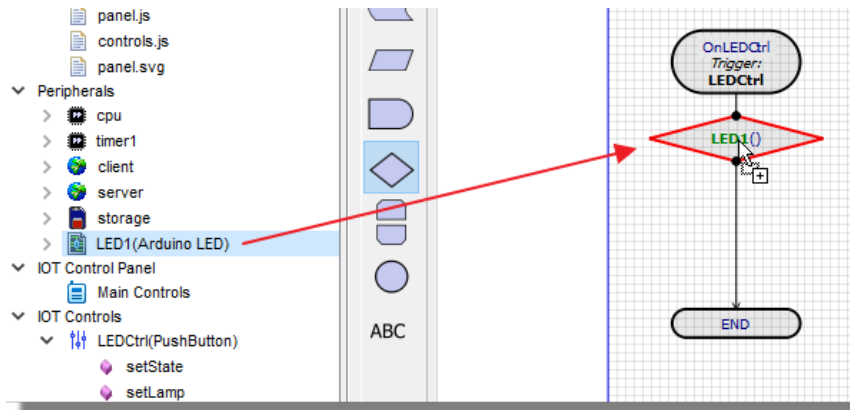


What we need to do in the program is respond to the press of a button and either turn the LED on or turn the LED off. In Visual Designer we do this by creating an event that corresponds to the button click. You can do this really quickly by dragging and dropping from the button in the Project Tree as shown below.



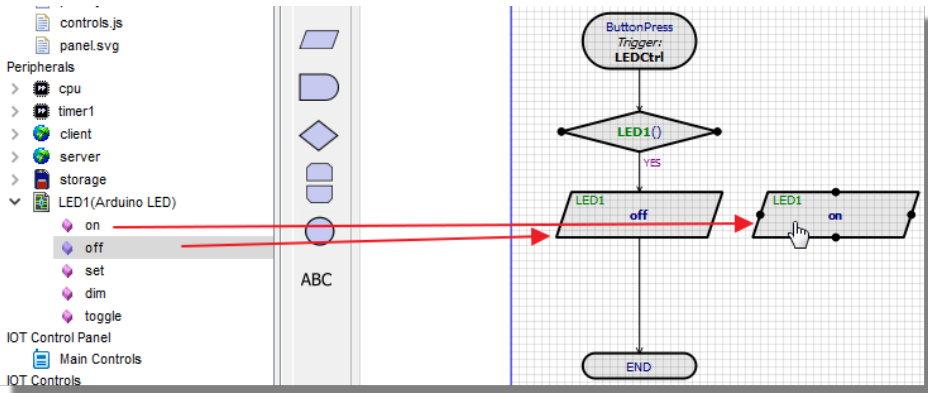
④ This is a shortcut which adds an event and then specifies it to respond to a button press interrupt.

What this means is that each time this particular button is pressed code execution will enter this routine. What we do when this happens depends on whether our LED is currently on or off. Drag and drop from the LED peripheral onto the flowchart and place the decision block in the new event.

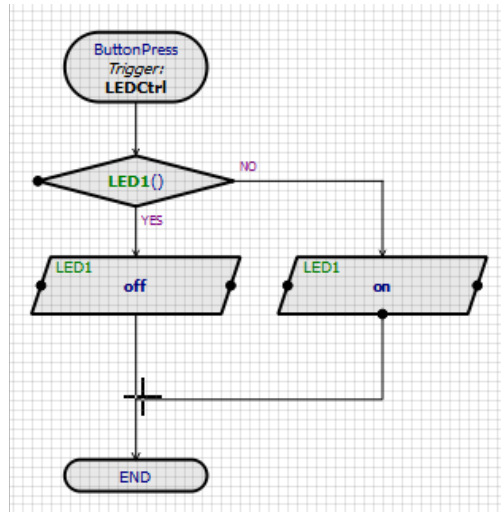
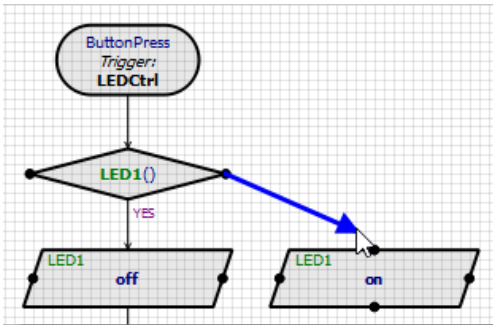


④ This is actually quite a powerful shortcut. When you drag and drop from the actual peripheral (rather than one of it's methods) you will get what we call the sensor function for the peripheral. In the case of the LED this is a decision that splits on whether the LED is on or off. You'll find more about sensor functions in the Visual Designer help file.

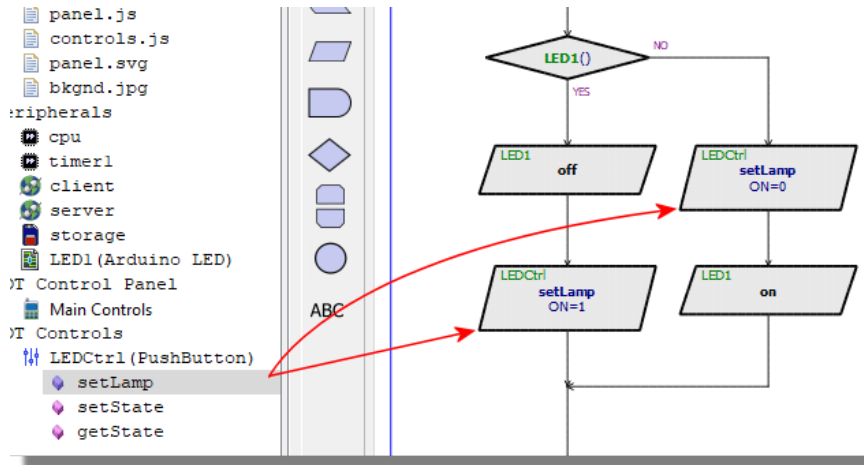
Now that we have a test for whether the LED is on or off all we need to do is turn it on when it is off and off when it is on. Expand the LED in the project tree and then drag and drop the off and on methods into the appropriate places.



Finally, we'll need to place a couple of extra wires to connect up the ON state.



As a final touch, we'll add the front panel indicator when the led is on and off. This amounts more drag and drop placement, this time of the setLamp method from the project tree as shown below.



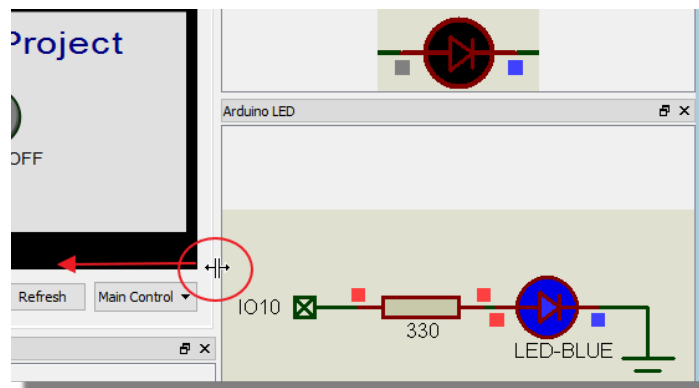
That's actually it - five flowchart blocks to toggle an LED on an Arduino Yun from a mobile phone! Next we'll look at how we can simulate and test.

## Simulate and Test

As a quick test that we haven't made any mistakes we can press the play button to simulate the system.

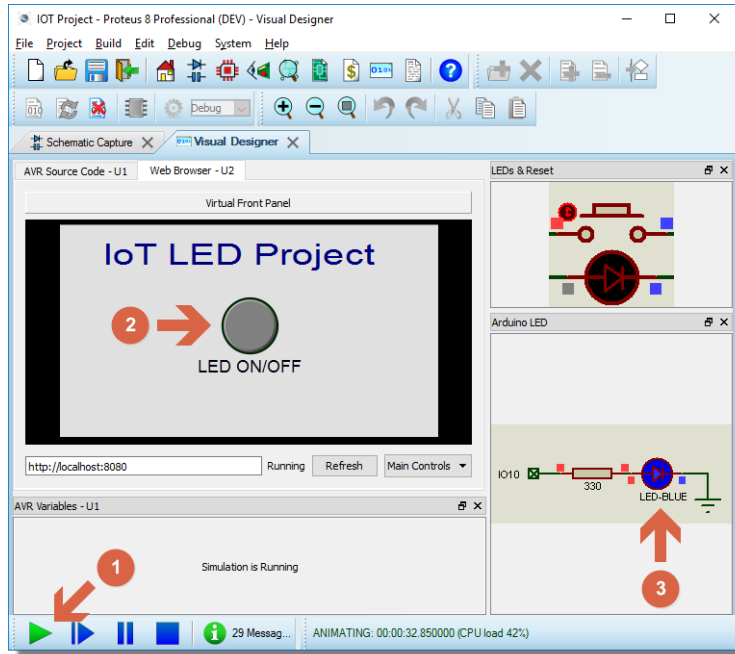


When the simulation is running on our virtual hardware we will end up with the front panel in the main section of the screen and any active areas of electronics on the right hand side. We can drag these out to suitable sizes - in our case to make the LED easily visible.





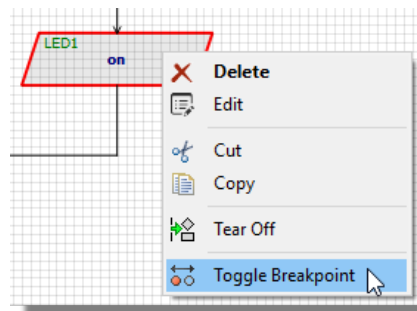
All that remains is to press the button and then wait to see if the LED turns on and then press the button again to turn the LED back off.



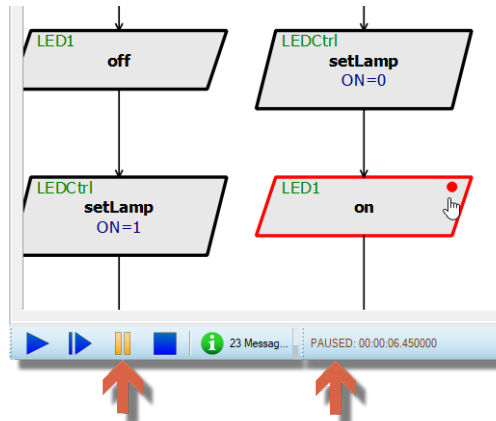
Let's assume however that something wasn't working and that we had to investigate. We'd start by pressing the pause button to halt the simulation.



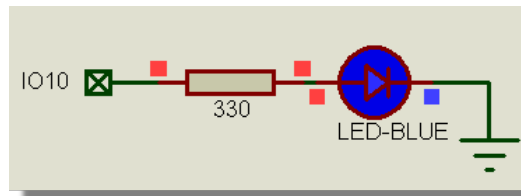
Next we'd set a breakpoint somewhere in our flowchart. In our example, let's do that at the method call to turn the LED on.



Now we run the simulation and press the button on our panel. The simulation will pause automatically when it hits our breakpoint.



We can then use the step commands to single step debug our flowchart while watching it's effect on the circuitry. In our trivial example a single step will turn the led on.



Another single step will illuminate the lamp indicator on our front panel



When we've finished simulation and we've confirmed everything is working as intended we can proceed to deployment.

**i** This is of course as simple a test case as it's possible to get and there isn't a lot that can be done in terms of debugging. You'll find more coverage in other tutorials and in the Visual Designer reference documentation.

## The App

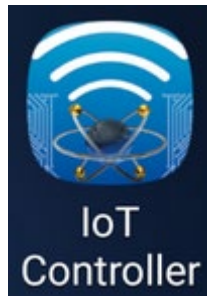
The free IoT Controller app is the easiest way to get the front panel onto your mobile device.

- If you have Android device you can download the APP from Google Play. Search for Proteus IOT Builder and click on install, It's free!

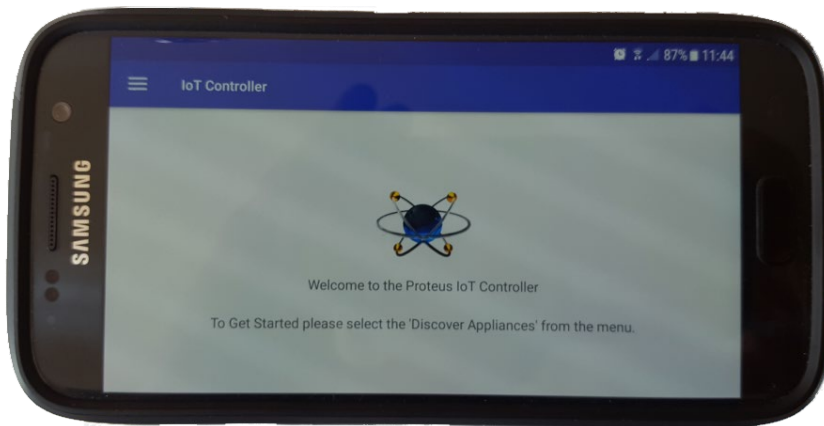
- If you have an iOS device you can download the APP from Apple Store. Search for Proteus IOT Builder and click on install, It's free!

The app serves as a host for the front panel you create and also helps with discovery. The discovery is done by the Bonjour utility / Driver. This utility is present on many machines but you can launch the installer directly from the Labcenter Program group on the Start Menu. Now that you have the app you can control your running IoT builder simulation from your app as follows:

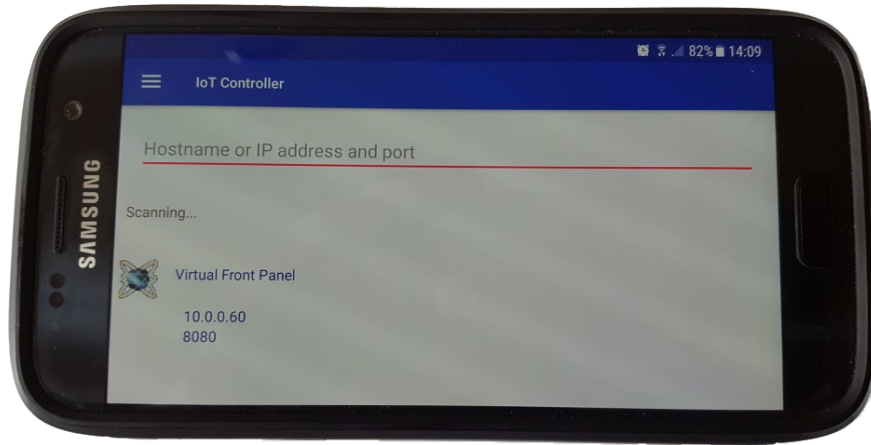
1. Make sure your mobile or tablet is on the same wifi network as your PC is on.
2. Start the running simulation in Proteus, **then** start the app on your phone.



3. The app will open and you will see the following:

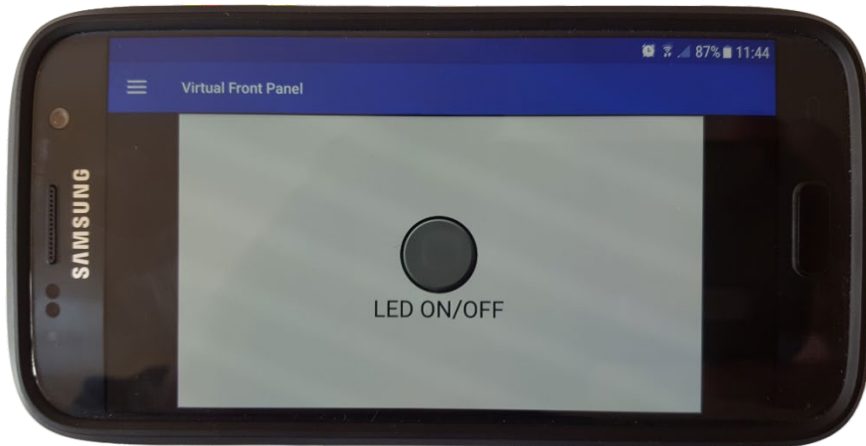


4. Click on the menu option at the top left and select Discover Appliances. Find Virtual Front Panel and select it with your finger.



**i** To change the name of the panel, go to Panel Settings (right click on Main Controls in the project tree) in Proteus and change the Page Title.

5. The Running simulation should automatically appear in the App:



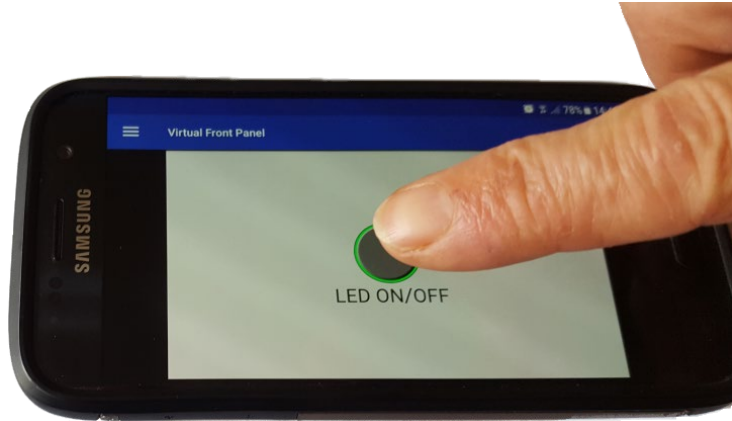
6. A Message will appear in the Front panel in **Proteus** that says the following:

**Lost connection to the 'Virtual Front Panel'.**

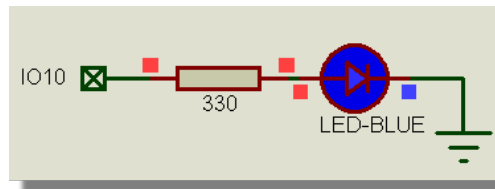
This is because it is now being displayed on the controller.

**i** Obviously this won't work if your computer doesn't have wifi or isn't on the same network as the phone.

7. Press the button on your phone, the led will come on in the simulation and the indicator lamp will light up on your phone app.



Which results in the LED on the schematic turning on:



8. Stop the simulation and/or disconnect when you are finished. Closing the App on your controller will not end the simulation on the PC, you will need to press the STOP button in Visual Designer.




 More information is in the Mobile App help section [The Mobile App](#)

## Programming

---

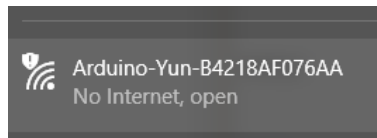
Now that we have the app and our mobile control center is working the final step is to program the physical hardware with our firmware. We can do this either through wireless SSH if your computer has wifi or through USB if not. The process is basically the same in either case. For this example we will do this wirelessly.

 The Arduino Yun Shield (mounted on an Uno or Mega) cannot be programmed via USB because the serial connection is already in use. The Yun board can be programmed either via SSH or via USB.

1) Switch on the Yun by plugging the power supply in (if you are programming via USB make sure it is plugged in, the USB cable to PC can act as both power and programming).

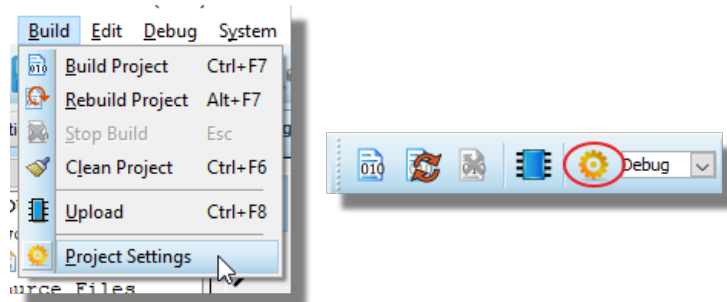


2) After waiting for about 30 seconds for the YUN to boot, Set your PC and your controller to be on the ARDUINO wireless network:

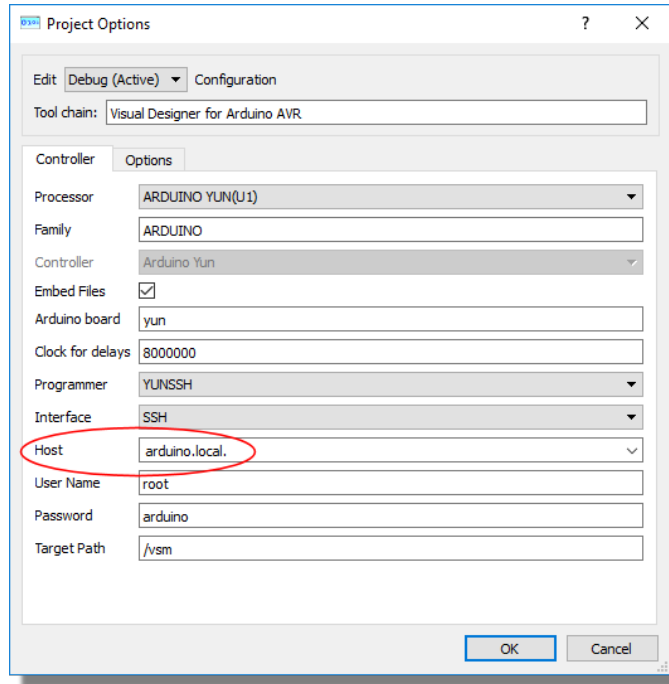


This will take you off the internet but you won't need it for this exercise.

3) Launch the project settings dialogue in Visual Designer.



4) If you are programming by WiFi select the YUNSSH programmer and the SSH interface. Your Arduino Yun should be discovered automatically and shown in the host dropdown combo box.



The default name of the Yun has the mac address appended at the end and you should find this written on a sticker on the back of your board.

If you are programming by USB you need to select the LEODUDE programmer, the COM port and the speed of the arduino. The YUN is 57600 by default.



Now click on the Firware icon to upload your program to the Yun:



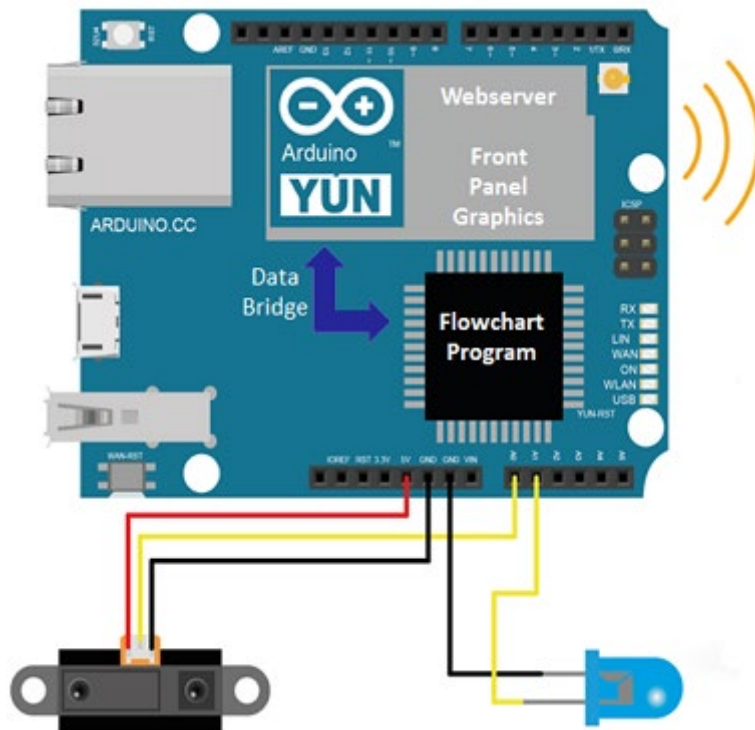
More on programming via wireless, USB and ethernet is explained [here](#).

### How does it work ?

If you are programming by SSH then the firmware and resources (webserver, front panel graphics etc.) will arrive at the Atheros chip. This will automatically run AVRDUDE and program the firmware onto the AVR processor via the ICP interface.


By contrast, if you are programming via USB then the bootloader on the AVR will be used to pass all of the resources across the data bridge to the Atheros chip and then AVRDUDE on the PC will be used to program the firmware onto the AVR processor. This method can be significantly slower if the front panel design is reasonably complex.

In either case, you will end up with firmware, webserver and front panel deployed as shown below.



*Flowchart program on AVR, webserver and front panel graphics on Atheros.*

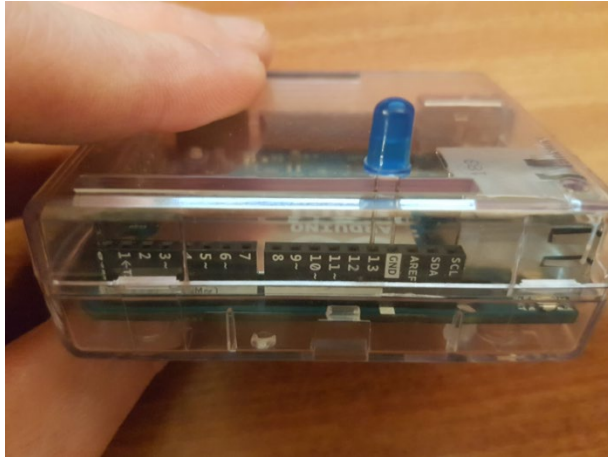
### See Also:

 [Programming the Hardware](#)

## **Controlling the Hardware**

Assuming that we have an LED wired on the real device to the same pin (IO10 in our case) and that the Yun is powered on then the first step is simply to use our app to discover the real Yun in the same way we did earlier for the simulated Yun.





Now we can press the button on our phone and control the LED on the real hardware, exactly as we did in simulation



That completes the basic tutorial and hopefully shows just how simple it is to get things working with IoT Builder. For something a little more useful/realistic it is also well worth working through the Temperature Logger tutorial.

# TUTORIAL 2: LOGGING THERMOMETER

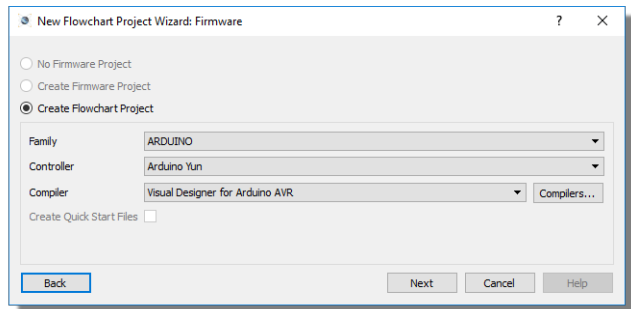
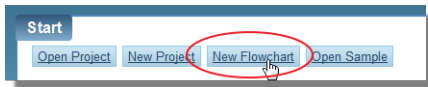
## Introduction

In this tutorial we will be setting up a logging thermometer application. Our front panel will consist of a graph with some buttons to adjust the range of the display and a thermometer to show the current temperature. On the hardware side we'll use the Arduino Yun and a grove temperature sensor.

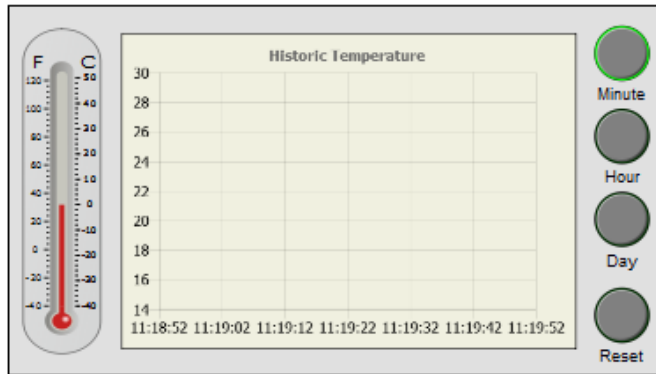
While relatively simple with IoT builder, this is still a bit more involved than our blinking LED tutorial. To avoid information overload we won't go over the same basic techniques that we covered in the first tutorial so it is worth either working through that one first or at least referring to it where needed.

## Front Panel Design

First, we'll create a new project with Arduino Yun and Visual Designer. This means that we will be designing our program with flowchart blocks.



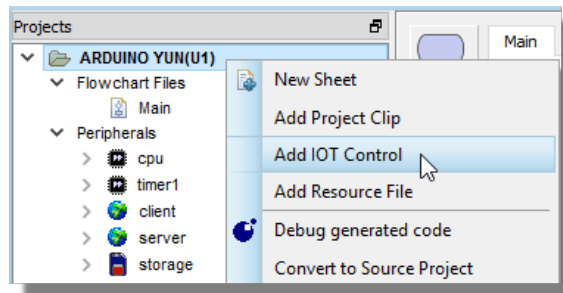
Our front panel for the logging thermometer is going to look like the screenshot below.



*Front Panel concept for a logging thermometer application*

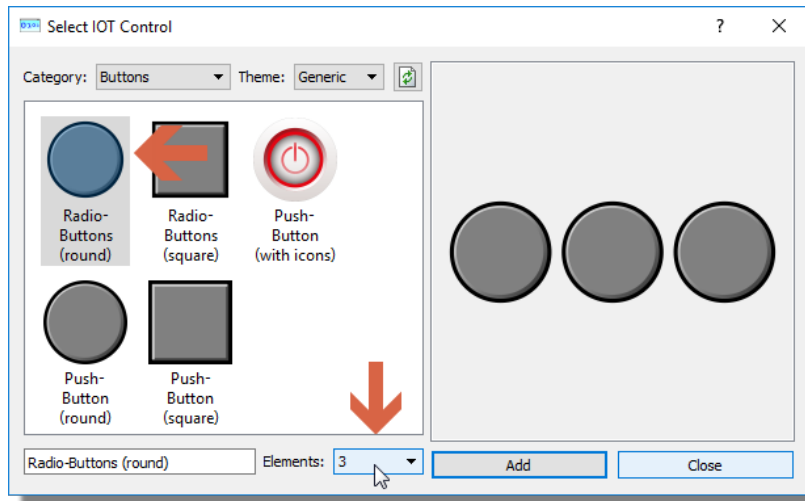
### Control Picking

All of the constituent parts are added from the IoT controls library which we access either from the Project Menu or by right clicking on the Arduino Yun in the Project Tree.

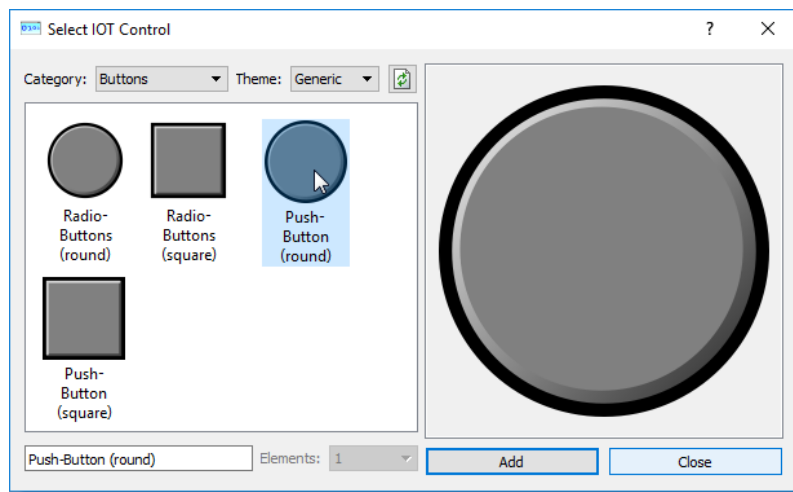


We want a mercury thermometer (-40 to 50 Celsius should be fine !) which can be found in the Display Controls category, a line chart with time x-axis which is under the chart controls category and then we need four buttons from the buttons category.

The buttons are worth further discussion because what we actually want is a group of buttons, much like radio buttons, that correspond to minute, hour and day. These are mutually exclusive - there will only be one selected at a given time - and so we can add them as a button group. To this, select radio buttons (round) and then change the number of elements to three at the bottom before adding to the project.



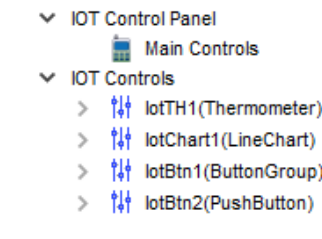
The reset button by contrast is a normal, stand-alone button so we can select push-button and add in the usual way.



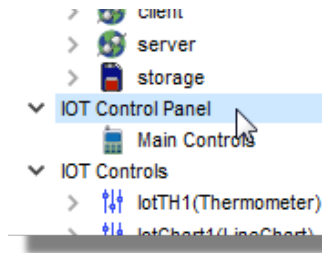
**i** In the screenshot above and in our selections we've chosen to add everything in the basic (generic) theme. You can of course change the theme from the top right and style your front panel in a different way if you like.

### Control Placement

Now that we've added everything we need into our project the next step is to lay out the controls on the panel itself. Your project tree should look something like the following.

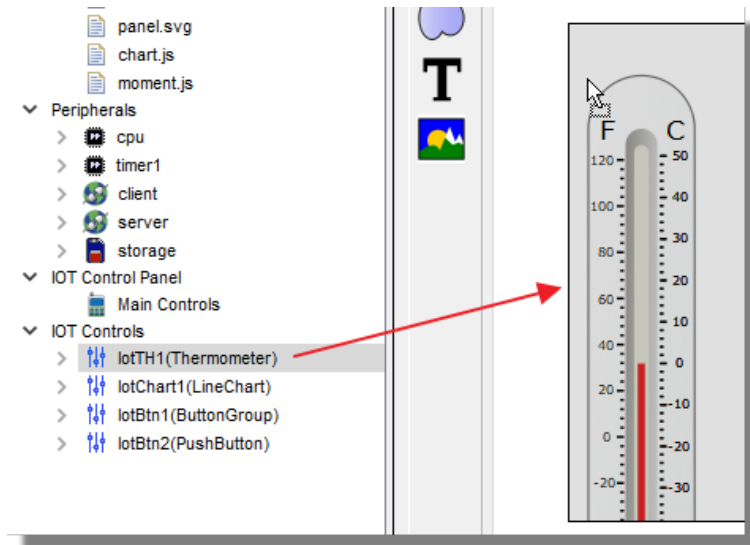


Double click on the IoT Control Panel to open it in the Editor.

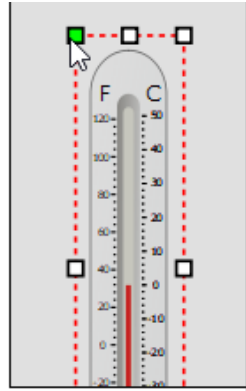


⚠ Remember that it is best if you first configure the panel size according to your target (iphone, nexus, galaxy, etc.) before you start placing controls. This is covered in the first tutorial. The default panel size is fairly standard for a modern mobile phone in landscape mode.

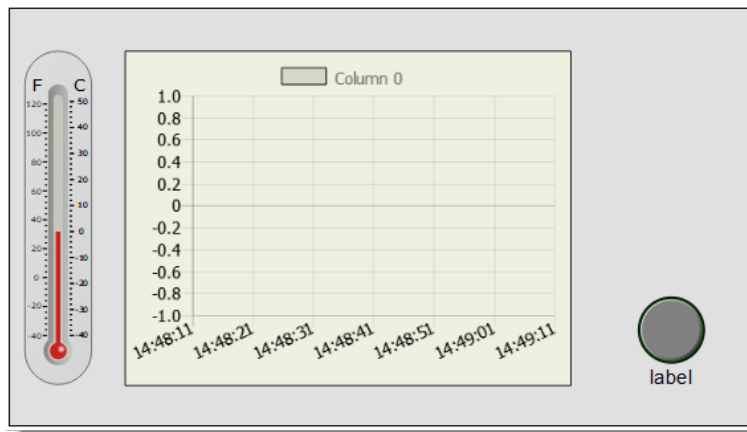
Drag and drop the thermometer onto the panel and position on the left hand side.



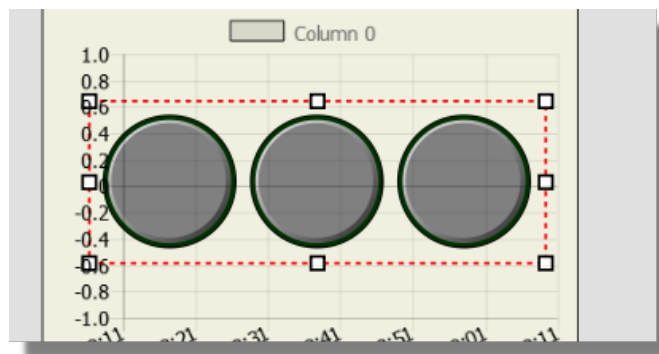
You might find that it is too big for the panel by default in which case simply use the drag handles to resize and then pick it up and re-position as required.



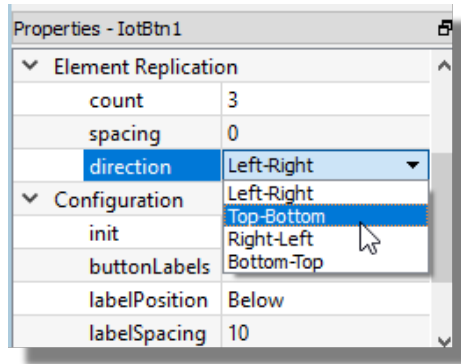
Repeat with the line chart and the reset button.



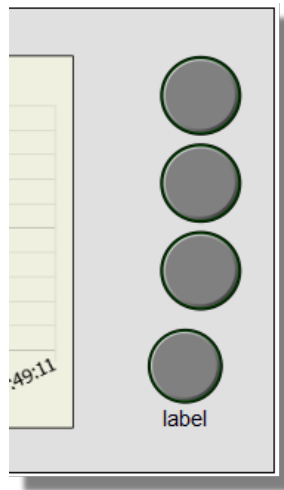
Now place the button group loosely on the panel - over the middle of the line chart is fine.



Switch to the property pane and find the element replication properties. Change the property direction from left-right to top-bottom.

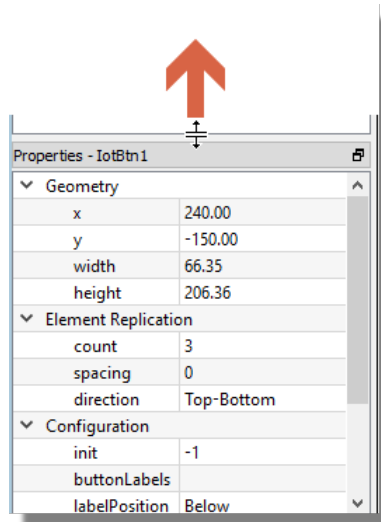


Now resize the button group to fit and then re-position above the reset button.

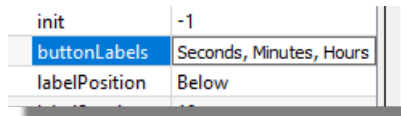


### ***Properties and Alignment***

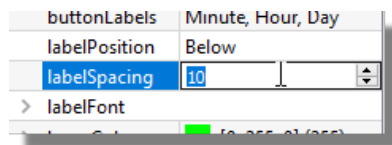
Unless you are very good at this, it's likely that your buttons are looking pretty ugly at this point. They may well be 'squashed' in one dimension and the button group is probably not exactly the same size as the reset button. That's to be expected at this stage and we'll adjust the sizing and placement as we add labels in the property pane. Select the button group first and then drag up the property pane so we more easily make changes.



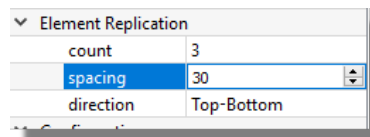
Scroll down to the configuration section and enter the labels as a comma separated list beside the buttonLabels property.



We'll need to include a little bit of label spacing (between the button and the text) so set this to 10

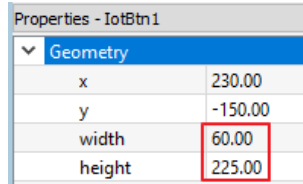


Next, we need to set the separation between the buttons so look for the spacing property under element replication and set this to 30. This is the distance from bottom of one button to top of the next button.



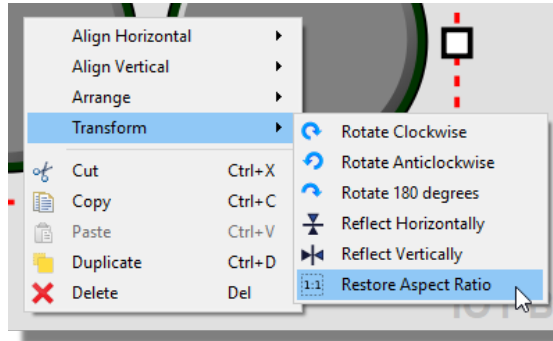
The size of the buttons needs some adjustment, you can resize them by dragging one of the corner nodes, or you can set the size manually. In this case we will set them to be width 60 and the height (of all three buttons) to be 225.



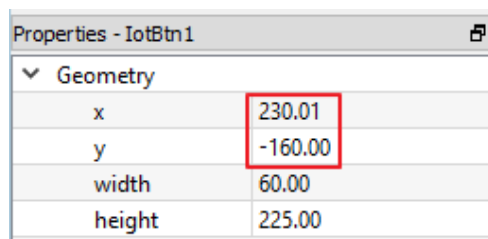


Properties - IotBtn1	
▼ Geometry	
x	230.00
y	-150.00
width	60.00
height	225.00

Note that if you have the buttons out of shape, you can right click on the buttons > Transform > Restore Aspect Ratio

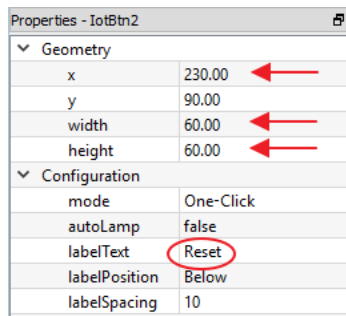


We need to move the button group into position manually or by adjusting the x and y values in the geometry section.



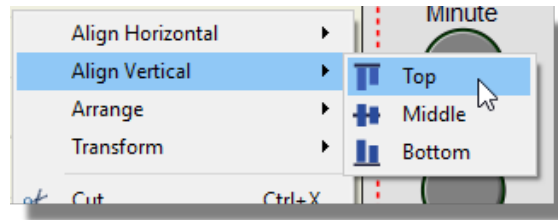
Properties - IotBtn1	
▼ Geometry	
x	230.01
y	-160.00
width	60.00
height	225.00

Now, switch to our reset button, set the label to be Reset, the width and height to be 60 and the x-value to be the same as the button group in order to align.



Properties - IotBtn2	
▼ Geometry	
x	230.00
y	90.00
width	60.00
height	60.00
▼ Configuration	
mode	One-Click
autoLamp	false
labelText	Reset
labelPosition	Below
labelSpacing	10

You can align multiple objects by holding the CTRL key and clicking on the objects, then right click > Align Hoz or Align Vert.

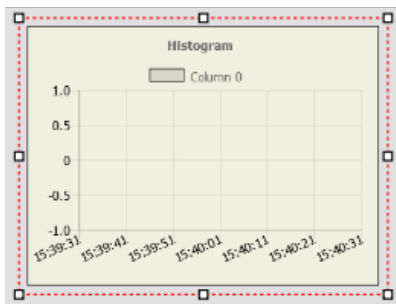


We could continue to align the line chart and the thermometer in a similar way. For example, if we set them all to have the same y-value (e.g. -165) the top of all the elements will be perfectly aligned.

Properties - IotChart1	
▼ Geometry	
x	-219.96
y	-165.00
width	459.96
height	320.00

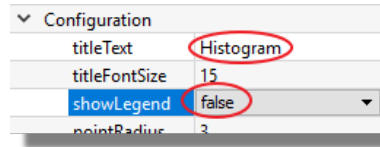
### Chart Configuration

The line chart is a very flexible control and has several configuration options. In our case, we will be changing many of these at run time (for example, we will be changing the time axis as the user presses a button to adjust the historic range). However, there are a few aesthetic things we can set up now. As normal, this is done via the property page once the control has been selected.

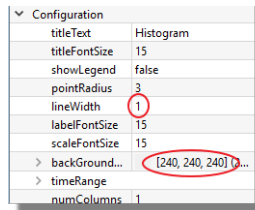


Properties - IotChart1	
▼ Geometry	
x	-220.00
y	-160.00
width	409.98
height	301.00
▼ Configuration	
titleText	Histogram
titleFontSize	15
showLegend	true
pointRadius	3
lineWidth	1
labelFontSize	15
scaleFontSize	15
background...	[240, 240, 224] (2...
timeRange	
numColumns	1
columns	

We can start by giving the chart a title and, since there will be but one dataset, we can remove the legend.



The line width can be dropped down to single pixel and we can change the default background colour to something more complementary to our panel.



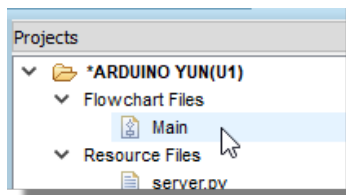
## **Writing the Firmware (Flowchart)**

---

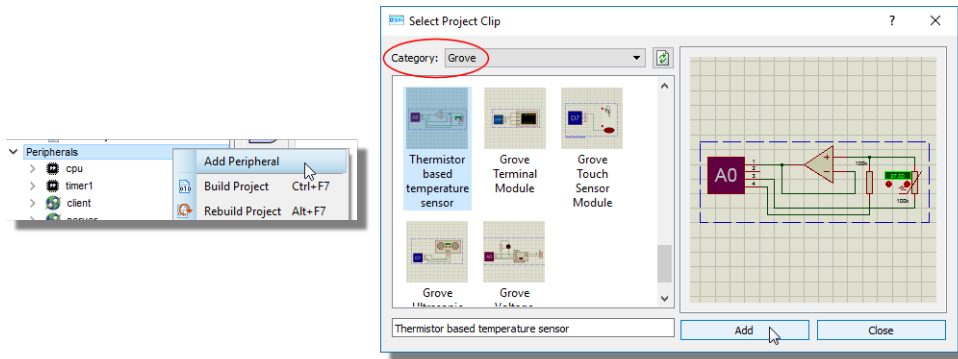
Now that we have our front panel in reasonable shape it's time to think about our program.

**i** We will skim quite quickly over the basics of flowchart design in this section. Please refer back to the Visual Designer help file (accessible via the Help menu in Visual Designer) if you need more detailed coverage of anything.

Switch to the flowchart by double clicking on the main flowchart file in the project menu.

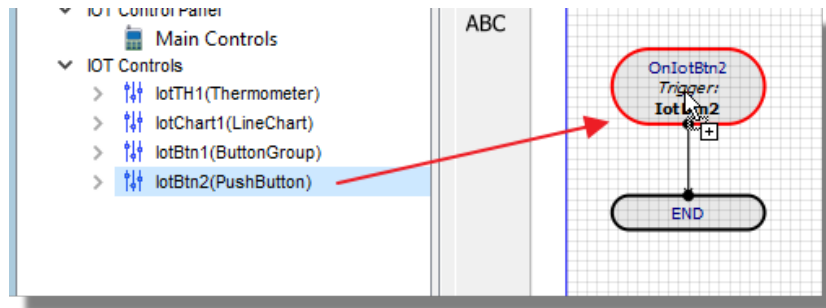


We need first of all to add the temperature sensor to our schematic (virtual hardware). We do this in the usual way via the Add Peripheral command in the project menu and add in the grove temperature sensor.

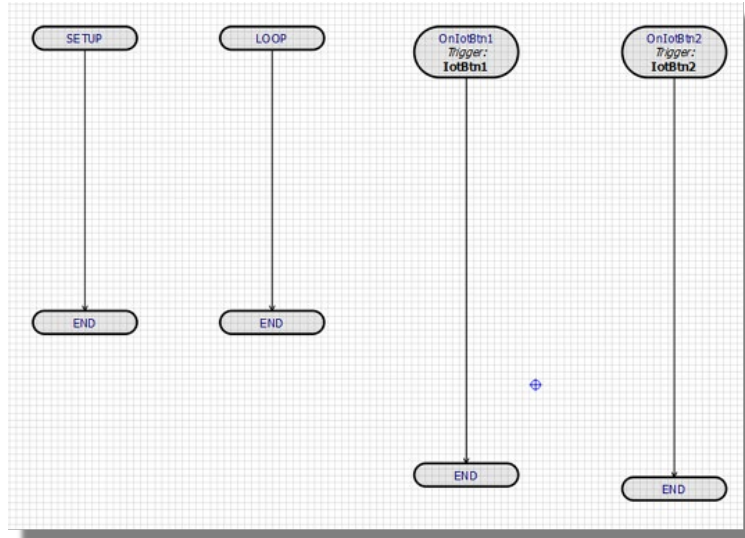


## Program Structure

Broadly speaking, our program is going to need both a setup phase and an operating phase and it is going to have to respond to user input in the form of a button press. The button press can either be a change in the history range for the chart (a press on one of the button group) or it can be a press of the reset button. We already have flowchart constructs for the setup phase (setup) and the operating phase (loop). We can respond to a button press with an event in the flowchart and, as we've seen in the first tutorial, we can add this simply by dragging and dropping from the IoT Control in the project tree onto the flowchart.

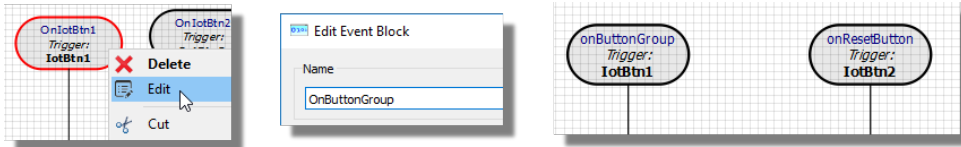


We need to do this both for the button group and for the individual (reset) button. With a little movement of chart elements around our basic program structure should now look like the following.



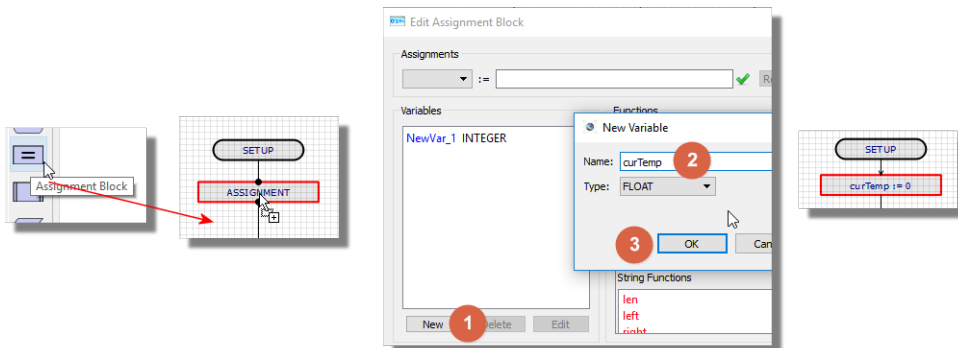
Basic routines for our program

It's a good idea to edit and change the names of the two button handlers to something more readable. For example, we can change to 'onButtonGroup' and 'onResetButton'

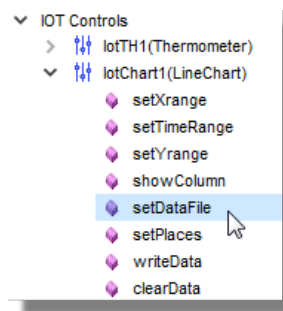


## Setup

The setup routine is where we do any initialization tasks before the program begins running properly. Since we have a thermometer that will be showing the current temperature we will need a variable into which we will store the temperature. Drag an assignment block onto the setup routine, edit and create a new variable for curTemp. Assign the variable to zero and as type FLOAT here.

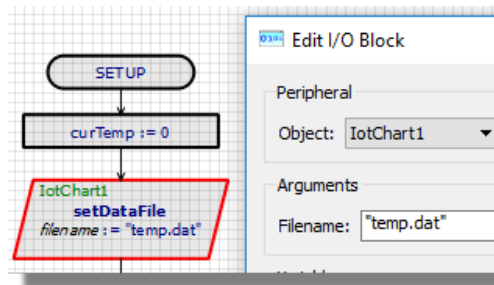


We also need to store our historical temperature data. For small data sets this can be done automatically and will store as session data on the Atheros but for larger data sets you can specify a filename and the data will then be stored to file, either on the Atheros or on the SD Card. We specify a data file for logging by expanding the methods of the chart IoT Control and then dragging and dropping the setDataFile method onto the flowchart.

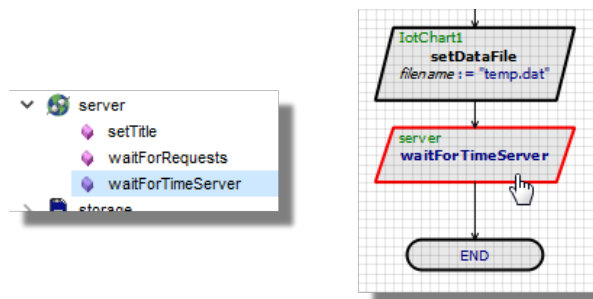


The log file for chart data points is by default stored in the VSM folder on the Atheros (Linux) chip on the Yun. You can mount it on an SD card by prefixing the file name with a path (e.g. /MMT/SDA1 for Yun shield or /MMT/SDB1 for Yun)

Edit the flowchart block and specify the filename.



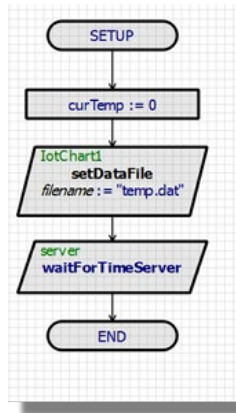
The other thing we need to do before we start logging is to make sure that the timeserver is up and running. We can do this by expanding the server peripheral and then using drag and drop in the normal way to position on the flowchart. This method takes no parameters as it is essentially a smart delay and will exit when the server tells it that the timeserver is active.



The timeserver is part of the Linux OS running on the Atheros which queries the current time from an internet server.

**⚠ The waitForTimeServer() method ensures that the time server is running and that we are receiving valid time points before it returns. This means that if your physical Yun is not connected to the internet (i.e. not receiving valid time) then this command will never exit and your program will not work.**

Your setup routine should now look something like the following



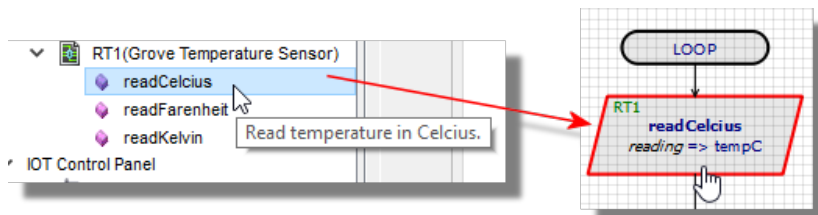
Setup routine for the logging thermometer.

## Loop

Now that we have the initialization out of the way we need to think about what our running program is actually going to do. We need to do three things in normal operation:

- Read the temperature
- Update the thermometer
- Write the chart data
- Check if the temperature has changed

The first of these is the easiest. Our thermometer is working in Celcius so we can simply drag and drop the readCelcius method from the grove temperature sensor into the top of the loop routine.



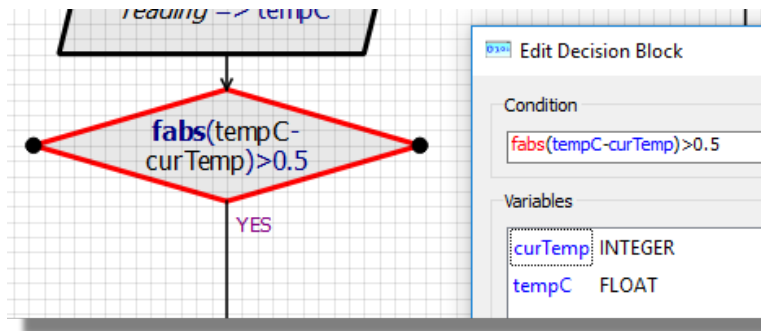
**i** Fahrenheit or Celcius is a configurable parameter in the property panel in the IoT Builder for the Thermometer. We could easily switch the thermometer into Fahrenheit and then use the corresponding method on the grove temp sensor to read the value.

Note that a new float variable (tempC) is automatically created and assigned to reading it.

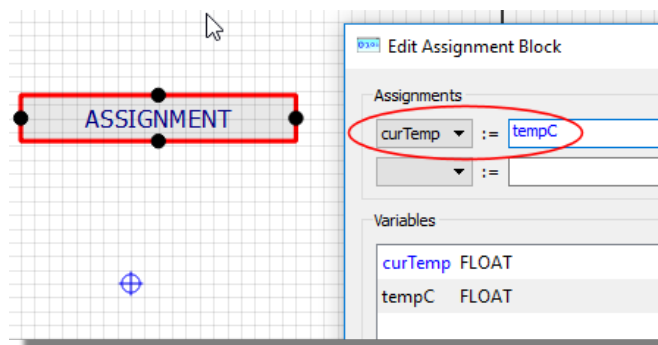
The reason we don't assign directly to the curTemp variable is time. Our loop routine is going to execute extremely fast which means if we don't think about what we are doing three things will happen.

- 1) Our log file will become enormous very quickly.
- 2) We'll be sending huge amounts of unnecessary traffic back to the controller front panel.
- 3) We'll be unresponsive to UI commands from the GUI because we are executing a tight loop at full speed.

So, when do we need to update the front panel ? Clearly, when the temperature changes we want an update so we'll add a test for temperature change. We can therefore add a decision to the chart that checks whether the temperature difference is greater than half a degree since our last logged temperature.

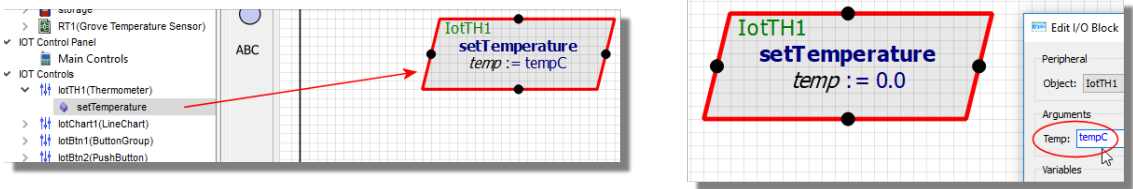


If the temperature has changed we need to assign the temperature reading (tempC) to our temperature variable (curTemp) in order that our test is valid the next time we execute the loop routine. Drag an assignment block next to the fabs decision block and set the following:

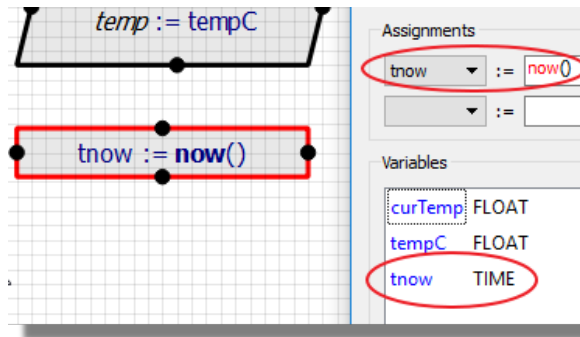




Next, we need to update the thermometer on the front panel with the temperature. We can do this by drag and drop of the setTemperature method of the thermometer IoT Control

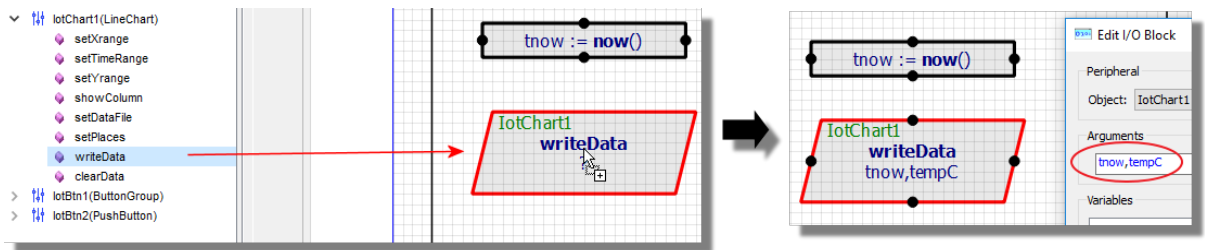


Finally, we need to log the temperature change. In order to do this we need to know the current time, otherwise we can't log a data point properly. We can get the time via the now() method and store it into a new variable (tnow) using a standard assignment block.



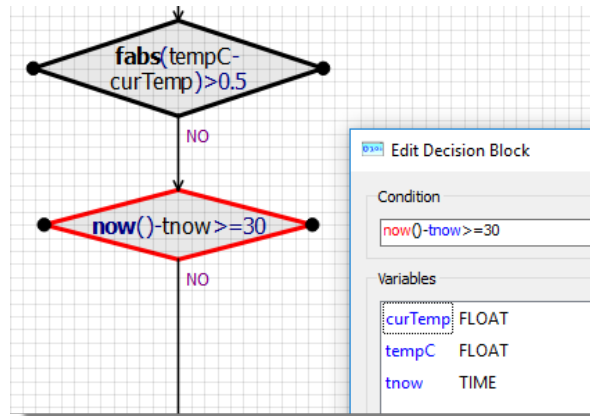
The now() method is simply a Visual Designer call on the timeserver that returns the time in secs since the EPOCH in local time. It therefore enables the AVR side not to worry about time.

Now we can use the writeData method of the IoT Chart control to log the data point.

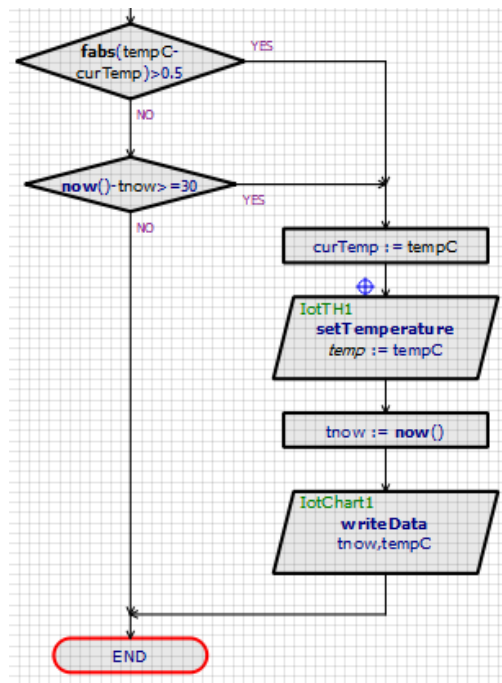


The writeData() method will both log the data point to file and update the chart with the current data point.

Now, if the temperature doesn't change we'll still want to log data points periodically. Let's say we write every 30 seconds or so if the temperature stays the same. We can do this by adding a simple test to the 'no' branch of our temperature check test. (again you will need to swap the yes/no.)

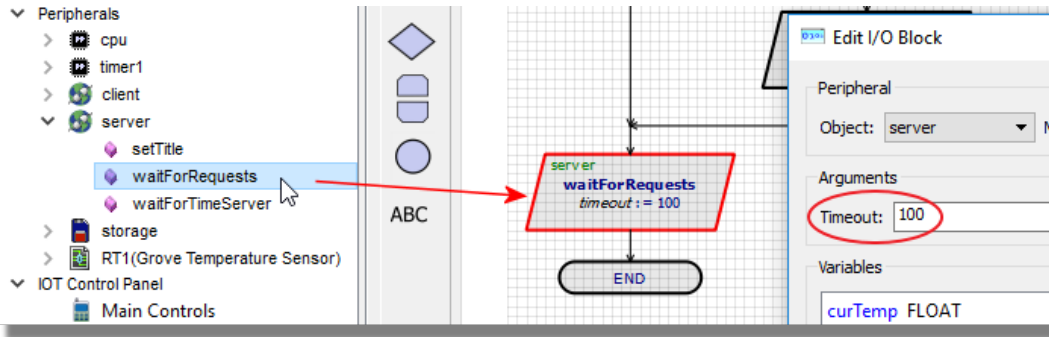


Connect up to the 'action branch', if the result of our time test is positive so that we log if 30 seconds or more has passed since our last log. This is what you should have so far:

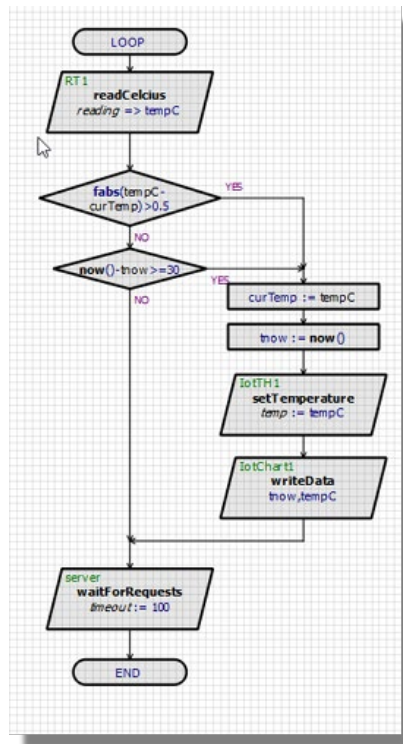


The last thing we need to do is add a pause. This will slow down the execution of the loop routine and allow time for pending UI commands such as a button press to trigger. It is extremely important here that we add a `WaitforRequests` server method rather than a normal delay method. A delay method is blocking and button presses etc will not be processed while we are in a delay routine. By contrast a `waitforRequests` routine will yield to an interrupt such

as a button press. To address problem (3) above and keep an IoT application responsive to the GUI therefore you should always use waitForRequests as shown below.



Your loop routine should now look something like the following.



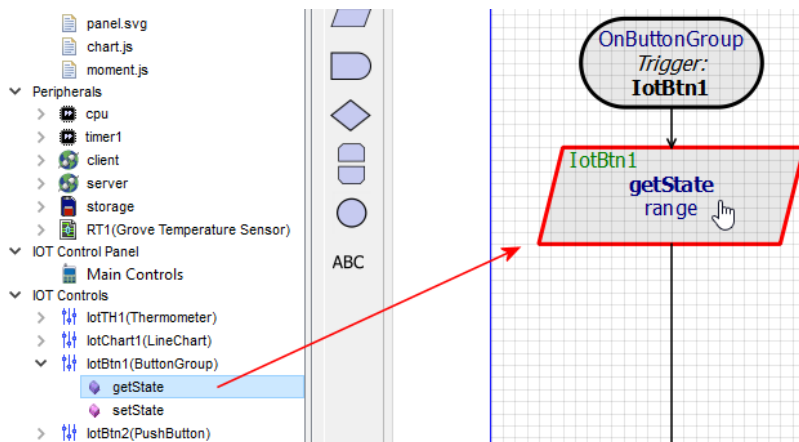
Main execution loop for the logging thermometer.

⚠ Not considering execution time versus update time to the controller is a very common mistake with this kind of program. Always remember that the loop routine will execute very quickly and anything you log/display/store is not going to work nearly as fast.

## Button Group Event

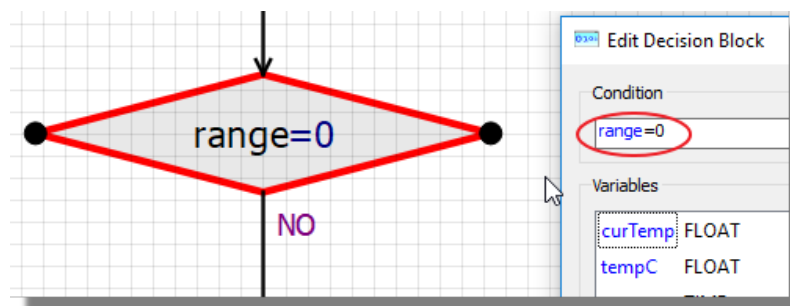
We've now got our program logic set up so the next question is how do we respond to a UI request from the controller. If we first consider the button group then we know that we'll be arriving at the top of our `onButtonGroup()` routine whenever a button in the group is pressed. So, we need to first determine which button it is and then set the chart x-axis to span one minute, one hour or one day.

Start by dragging and dropping the `getState` method from the button group IoT control. Add an integer variable called 'range' and assign the result to it.

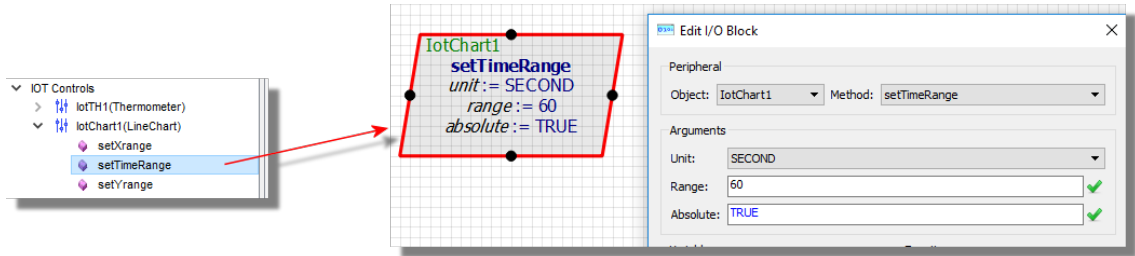


**GetState()** on a button group returns a zero based index according to the buttons position in the group. The first button is index 0 (Seconds in our case), the second is 1 (Minute) and the third is 2 (Hour). It is set up this way so that a button group can be extended as much or as little as the user requires. For example, you can specify a 10 button group and then test between 0 and 9 to determine which button has been pressed.

Now we have the state we need a test for each button state and we need to set the x-axis range accordingly in each case. So, in the case of the seconds button we want to test for `range=0`. drag a decision block on to the flowchart and add `range=0` having swapped the Yes/No.



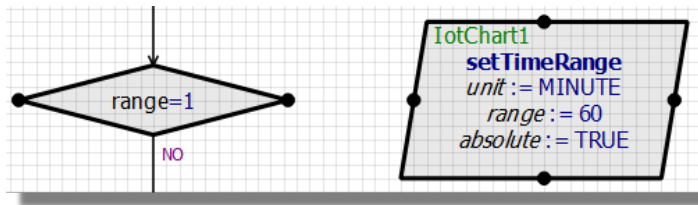
To set the x-axis to span one minute we are specifying a time range so drag and drop the `setTimeRange()` method from the line chart IoT Control.



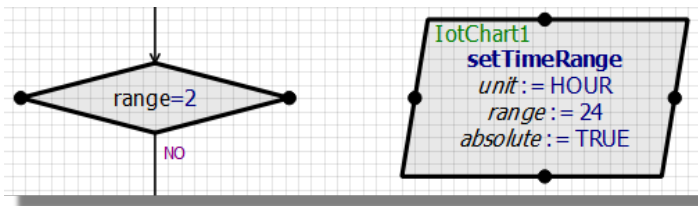
The configuration is then to set the units to seconds and the range to 60 (60 secs = 1 minute). We also want the range to be absolute as above. The time range specifies the amount of time displayed, going backwards from the latest timepoint in the data.

If Absolute=FALSE then the time data is relative.

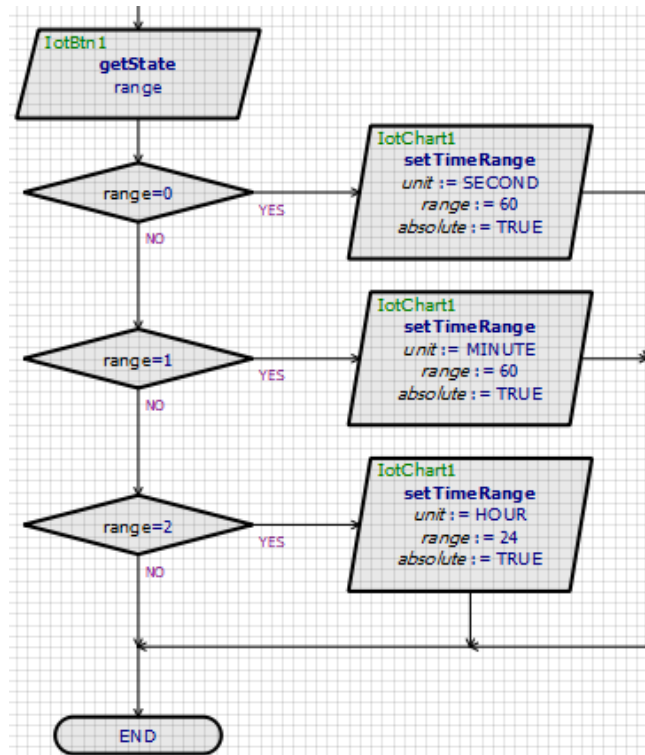
Now, we rinse and repeat with our test for the minute button press.



And finally for the hour range test.

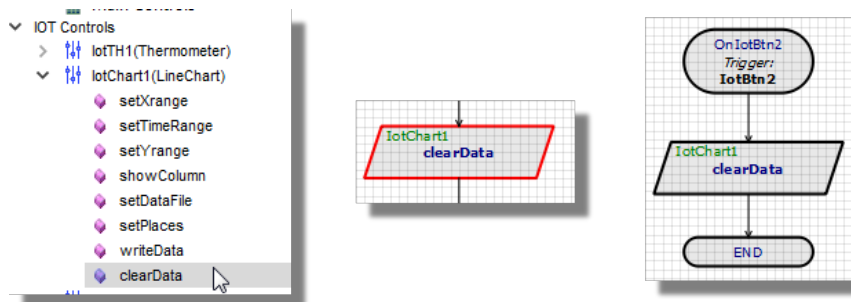


The button group handler should look something like the following when you are finished.



### Reset Button Event

Our last job is to respond to a reset button push inside the onResetButton() routine again, the IoT control for the line chart has the method we need so all we need do is drag and drop the clearData method into the button handler.

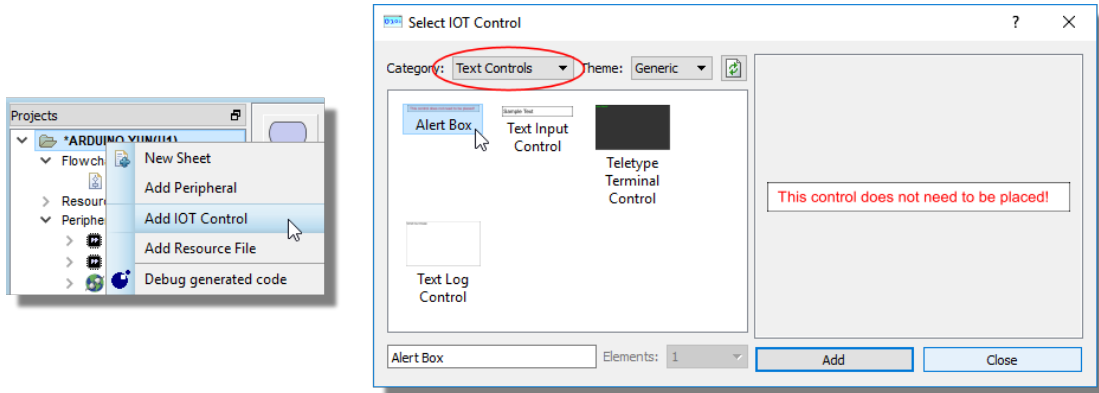


Simple method for handling the reset event.

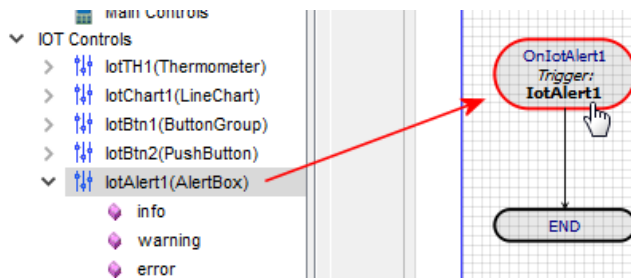
Since we are wiping the entire graph history at this point it may be better to prompt the user to confirm the action before we do this. Fortunately, IoT Builder can provide a way to do this by

## LABCENTER ELECTRONICS LTD.

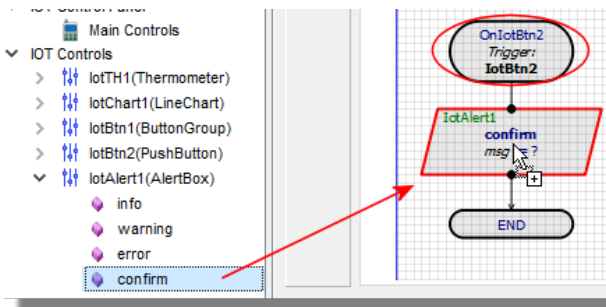
using the AlertBox control. To use this control, first pick it from the text area of the IoT controls dialogue



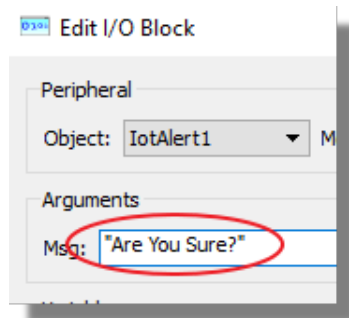
Unlike other controls this one doesn't need to be placed on the panel. We can switch directly to the flowchart and drag the Alert control on to the flowchart and move to a required location:



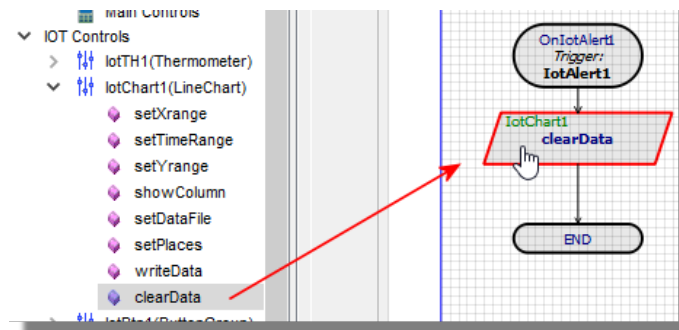
Now drag the 'confirm' method onto the reset button event handler.



Edit this and provide suitable text for the message.



What we want to do here is clear the data because the user has confirmed that this is the action they want.



### Notes:

An alert box is a two stage process. First you have an alert method which pops up the box for user input. Second you have an alertbox handler which is called if the user presses the OK button.

The alertbox handler is called when user presses the OK button only !! The text of the alertbox is irrelevant - if the user hits OK the event handler will be called and if they don't it won't be called.

Certain types of alertbox (e.g. info) only have an OK button and so the event handler will always be called. Others, such as confirm, give the user a cancel option. Note that, even in the case of the confirm box you are not giving the user a binary choice because you cannot respond directly to a cancel in your program - the alertbox handler is called only when the OK button is pressed.

**⚠ The call to the alert box handler returns immediately and program execution continues ! You need to therefore consider what code will be executing while the user is reading the message and responding.**



## Summary

That's everything we need for a IoT enabled logging thermometer in a single sheet of flowchart program. There are some important principles and tips from this process that will apply to almost all of your own projects which are summarized below.

- Your front panel actions (switches, buttons, sliders, etc.) are processed in the flowchart as event routines. Drag and drop from the IoT Control onto the flowchart to place the event routine and then add the required blocks inside it.
- Think about time when it comes to your loop routine. How quickly will it execute, how quickly does it need to execute and is there time to respond to GUI commands.
- Always use `waitForRequests()` instead of a normal `delay()` if you want your program to respond to UI events.
- If you are in doubt about how to drive a peripheral on the schematic look at it's methods in the peripherals section of the project tree.
- If you are in doubt about how to do something with an IoT control look at it's methods in the IoT controls section of the project tree.

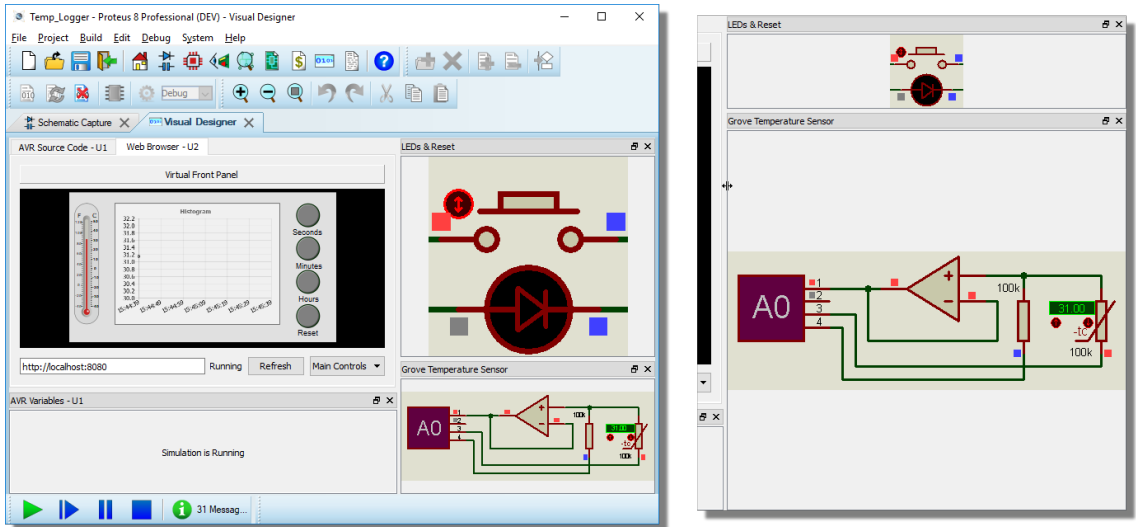
## Simulation

---

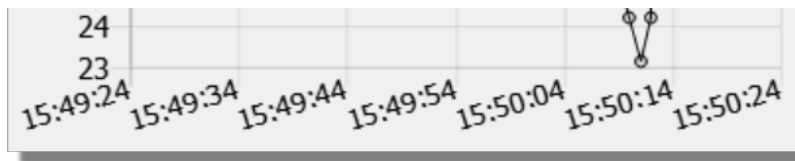
Now that we've worked through the front panel design and the programming we can give the entire system a trial and see how it works. Start by pressing the play button to run the simulation.



You should end up with the front panel in the main section of the window with the reset button on the Yun at the top and our temperature sensor at the bottom. You can resize these windows as required - in our case it would make sense to make the temp sensor larger so that we can see the readout.

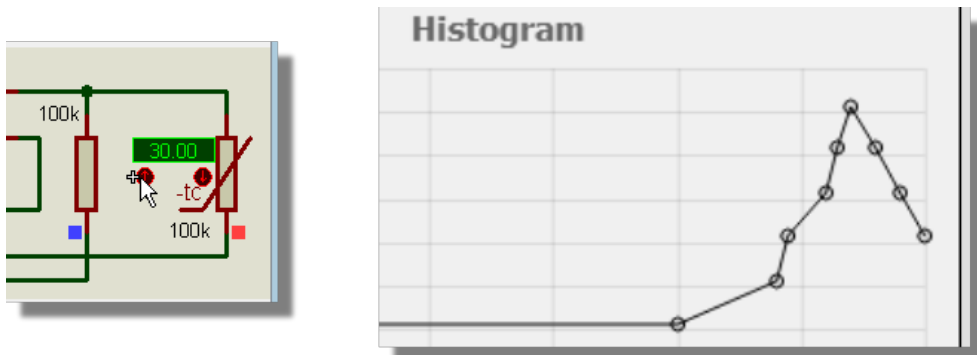


You should notice that the time period on the x-axis matches the default in the button group and that it actually showing the correct time.

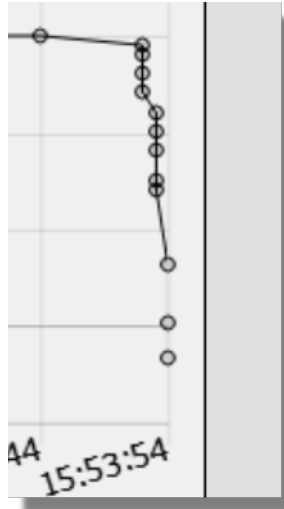


**i** You should also notice that the graph time range advances every minute in real time.

Our first test is to increase the temperature on the temperature sensor by clicking on the activation button and check whether the live display on the thermometer and our chart responds accordingly.

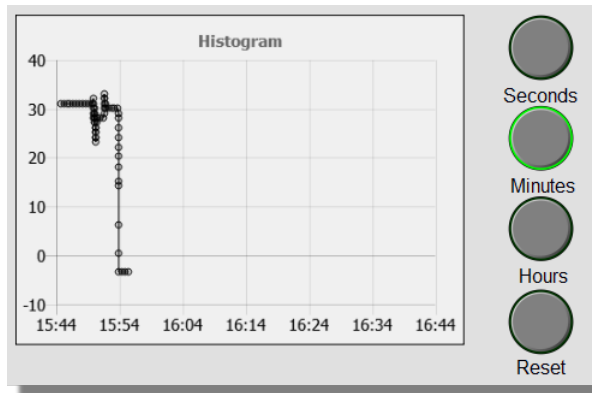


Similarly, we can reduce the temperature on the temp sensor and check our display. We can hold the mouse on the down button to move the temperature down rapidly and examine how responsive our chart and thermometer are.

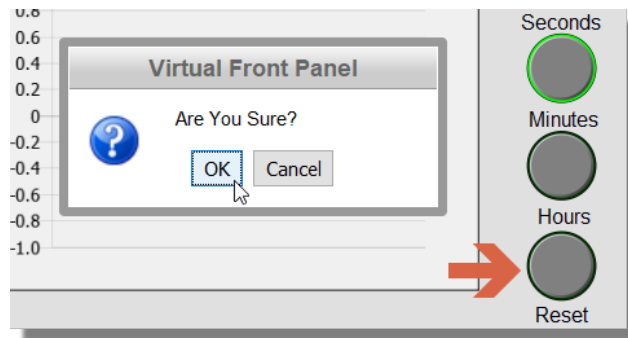


Compare how easy it is to bounds test a temp sensor compared to the real world (hair dryer and fridge ?).

After a minute or so our data changes will be moving off the chart so we can change the range to one hour and check that the time range is correct and that the data from our temperature testing has been correctly stored.



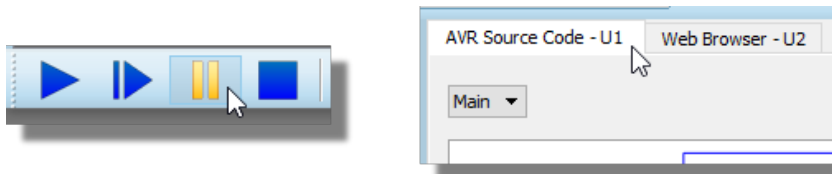
Finally, we can reset the display and confirm that our data set is wiped by checking the chart display.



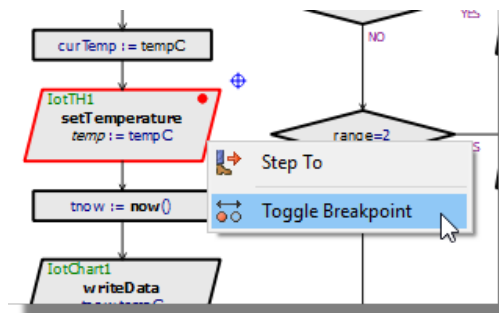
If you want to at this stage you can also check how the front panel looks and performs from your mobile/tablet device. See the first tutorial

## Debug

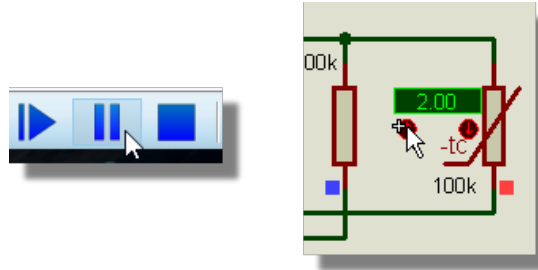
Let's look at a few different ways we can pause our program and single step to look closer at our program flow if something has gone wrong. Press the pause button to halt the running simulation and switch to the source code tab.



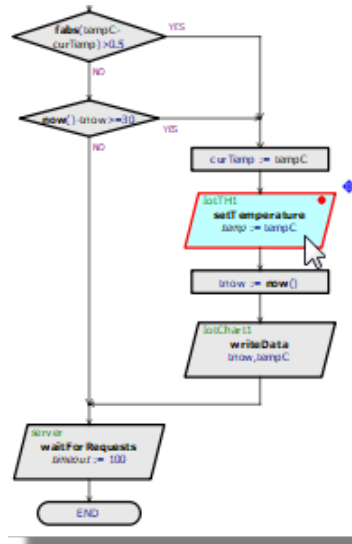
Set a breakpoint on the `setTemperature()` method in the loop routine.



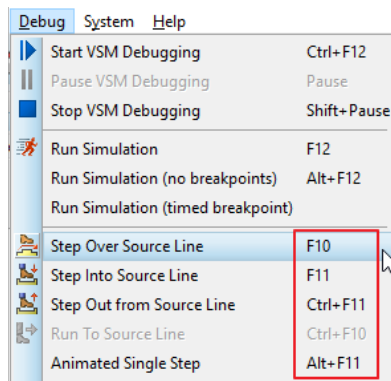
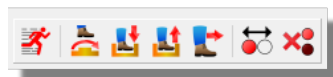
Now run the simulation and change the temperature on the temp sensor.



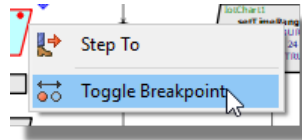
You should find that the program automatically halts and presents you with the source code window with code execution at the breakpoint.



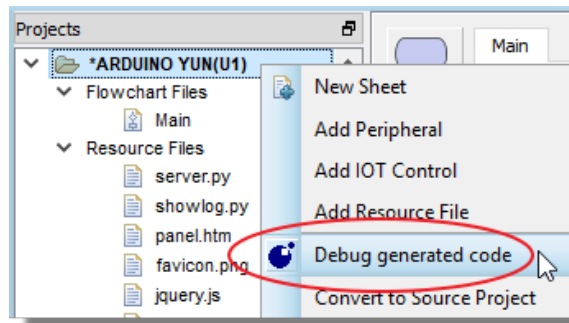
You can now single step your program using the icons at the top, or the keyboard shortcuts shown in the debug menu.



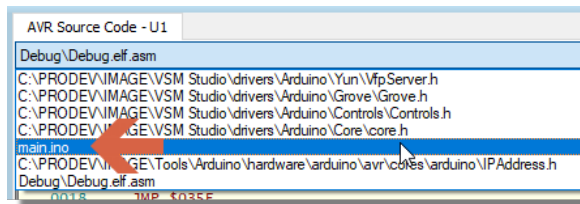
Finally, you can toggle the breakpoint off and stop the simulation when you are finished.



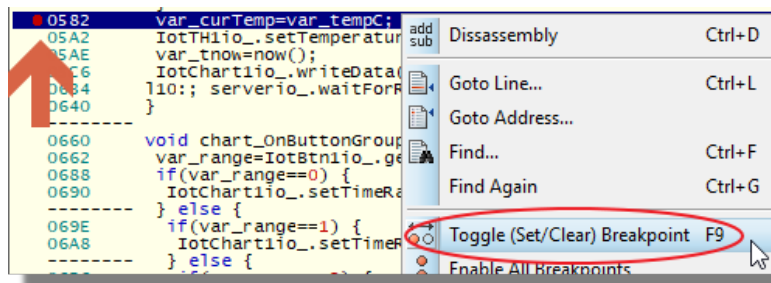
As well as debugging the flowchart it is also possible to debug at source code level. When the simulation is stopped, right click on the Arduino Yun in the project tree and select the debug generated source command from the context menu.



Pause the simulation and select the main.ino source file from the file selector at the top.



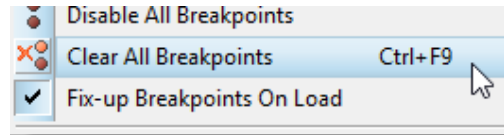
Set our breakpoint in the equivalent source code line, namely `var_curTemp = var_tempC;`



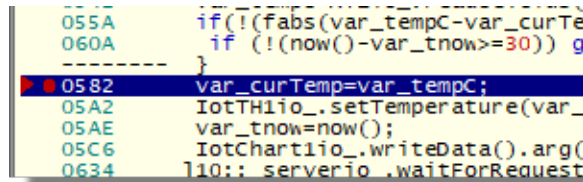
Run the simulation, wait a few seconds for the front panel to load and adjust the temperature sensor. We should pause at our breakpoint and single step exactly as before.

**i** If you want you can even single step at machine code level by using the disassembly command on the right click context menu.

Clear the breakpoint via the context menu command and stop the simulation when you are done.

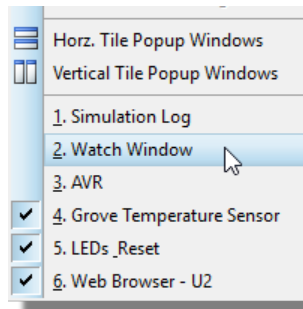


You can return to flowchart debugging by selecting the debug generated source command on the project menu context menu again (it is a toggle).

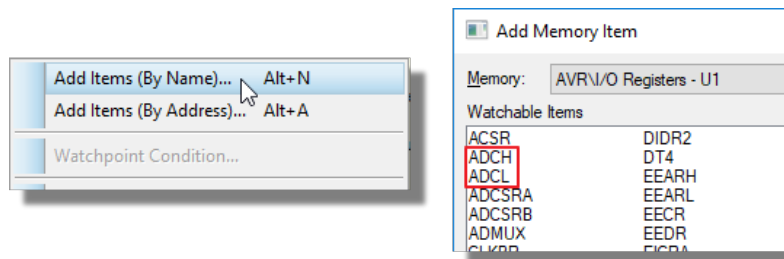


### Advanced Debugging

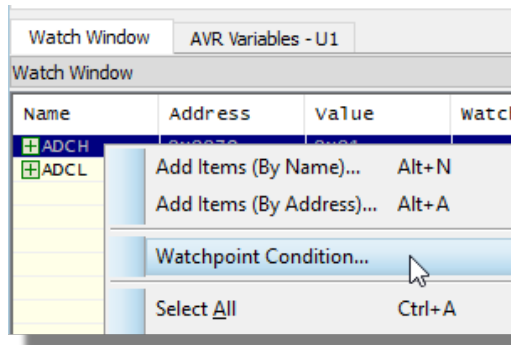
If you are more familiar with the AVR processor and/or the electronics on the schematic then there are a couple of other useful ways to control the simulation. The first is to make use of the watch window to control program flow. The first step is to pause the simulation and then open the watch window from the debug menu.



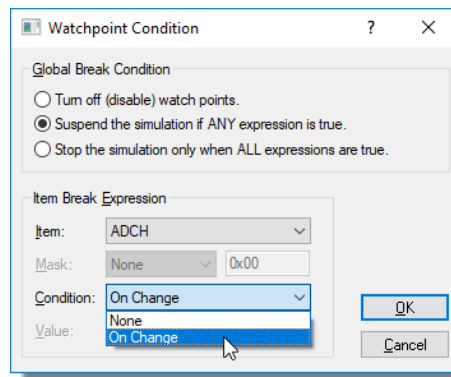
In our case we want to look at the ADC results because the grove temp sensor is connected to the ADC on the AVR. Right click on the watch window and select the add by name command from the resulting context menu.



Add both ADCH (high byte) and ADCL (low byte) to the watch window and then run the simulation. Notice that, as you adjust the temperature on the sensor the value of ADCH will increase. You can choose to set a breakpoint (called a watchpoint condition) on or above a particular value. Right click on the ADCH line on the watch window and select watchpoint condition from the resulting context menu.



Select on change and then run the simulation. The program will pause each time you change the temperature.



If you really want to understand what is happening here you can switch to source code debugging and repeat the experiment pressing F10 to step once you hit the breakpoint. You'll actually find the ADC conversion code in the grove.h file for the temperature sensor (select from the drop down at the top).

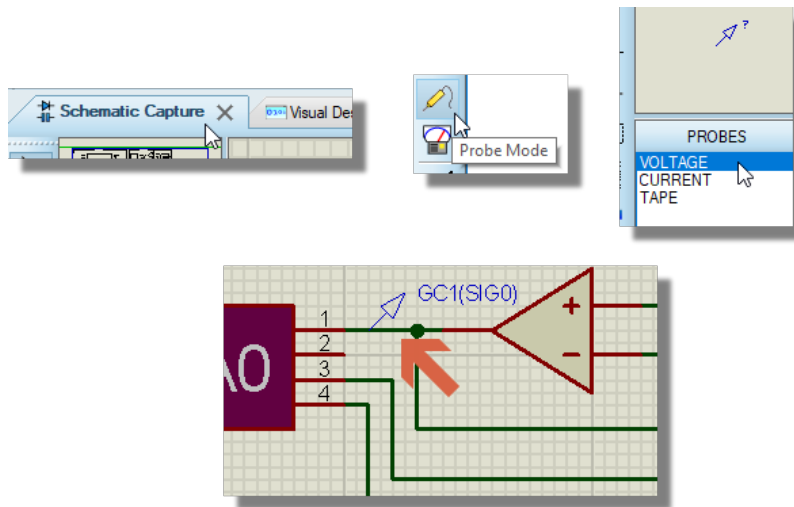


```
AVR Source Code - U1  Web Browser - U2
C:\PRODEV\IMAGE\VSM Studio\drivers\Arduino\Grove\Grove.h

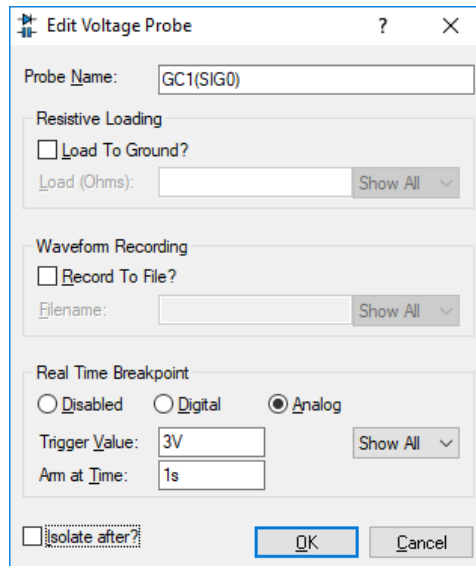
-----
----- class GroveTemperatureSensor
-----
0368 { public:
-----
0369     GroveTemperatureSensor (uint8_t id) { pin = id; pinMode(analogInputToDigitalPin(pin),INPUT); }
-----
037C     float readCelcius() { return readkelvin() >= trigger; }
-----
038E     float readFahrenheit() { return 9*readkelvin()/5 - 459.67; }
-----
038E     float readkelvin() { return 1.0/(Log(resistance()/10000)/3975+1/298.15); }
-----
----- private:
03CC     float resistance() { int a = analogRead(pin); return (1023-a)*10000.0/a; }
-----
----- private:
-----
----- };
```

More information on the Watch Window can be found in the Proteus VSM Help file

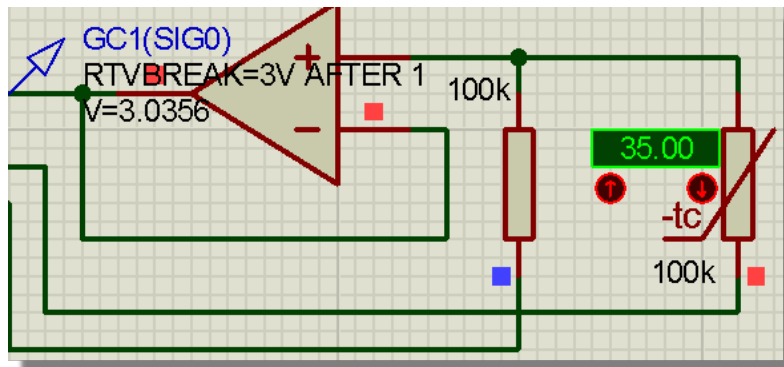
You can even set a breakpoint to trigger on a hardware condition. For example, we can trigger on a voltage change on the ADC input line. First, switch to the schematic, select a voltage probe and drop it onto the line.



Next, edit the voltage probe and set the breakpoint to be an analogue breakpoint triggering at 3V with arm time of 1 second (to miss any initialisation noise).



Press play and run the simulation. You should find that as you increase the temperature the voltage across the probe increases and the breakpoint triggers around 35deg C.



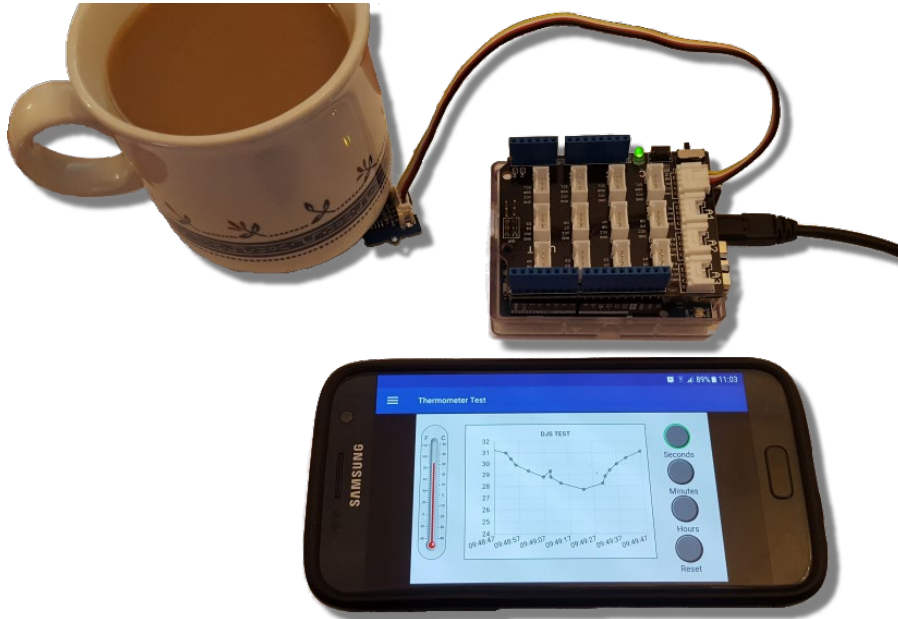
- i This last couple of debugging examples are of course a bit more involved than what we have shown previously but it serves as an example that the full professional debugging tools of Proteus VSM are available to you if need be.

## Deployment

When we are satisfied with our system we can deploy to the real hardware exactly as discussed in the first tutorial.

See Also: [Programming](#)

Here's a picture of the running hardware controlled from our mobile phone. We placed the temp sensor beside the coffee cup for a little while to graph some changes.



# SOURCE CODE PROJECTS

## Introduction


---

For more advanced users, IoT Builder is a great product to work with in C++ source code projects as well. Since the target hardware (the appliance) is Arduino you'll need to have a product license for one of the following Proteus products to use source code projects:

- Visual Designer
- Proteus VSM for AVR
- Proteus VSM for Arduino AVR
- Proteus Platinum

You'll also need a license for the IoT Builder product unless you have Proteus Platinum which includes everything.

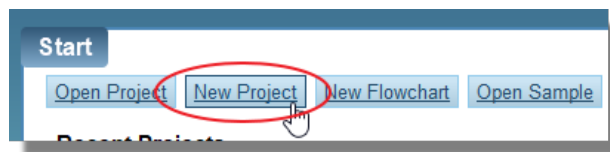
The workflow is largely similar to that discussed in the Visual Designer tutorial linked below but there are a couple of important differences which are covered in the topics below.

 See Also: [Tutorial 1 : Blink an LED \(Visual Designer Project\)](#)

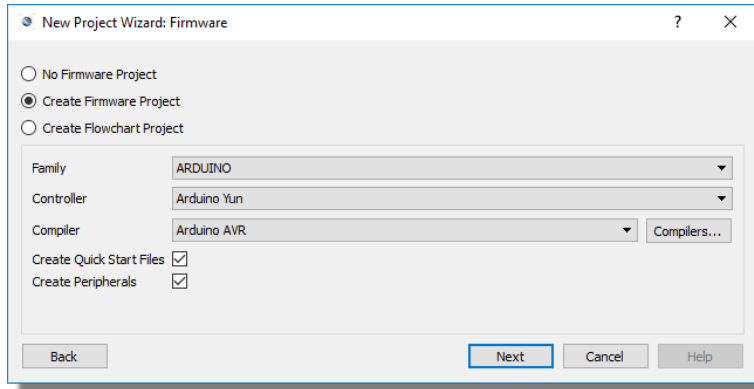
## New Project Wizard

---

When you are creating a source code project you start the New Project Wizard with the New Project button.



The default options will suffice until the firmware page when we want to select the Create Firmware Project option (not the Create Flowchart Project option). The target family is Arduino and let's choose the controller as the Arduino Yun. The other important options is to make sure that both the Create Quick Start Files and the Create Peripherals options are selected.



**i** The Create Peripherals checkbox is vital here as it will allow you to choose from all of the prepared shields and breakout boards and will initialize all of the Labcenter peripheral drivers that work with them. If this box is unchecked you will not easily be able to use IoTBuilder in your design.

Default settings for the rest of the New Project Wizard are then fine.

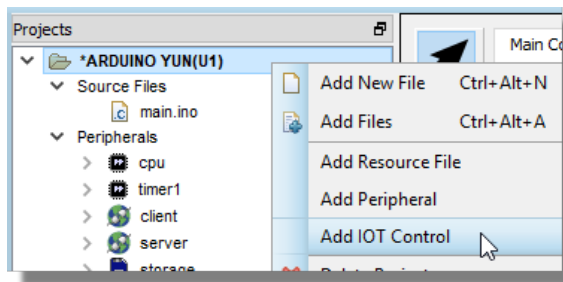
Once complete, you will end up with a source code project in VSM Studio which includes a sizeable amount of pre-generated code. Those who work with MFC or similar will find this familiar but the key thing is not to disturb, change or move the generated code. This is used by the Labcenter drivers and will update as you add peripherals so needs to be managed by the Proteus software. You should ignore this section of code and write your code inside the setup() and loop() routines as normal.

On the schematic you will find a pre-placed Arduino Yun.

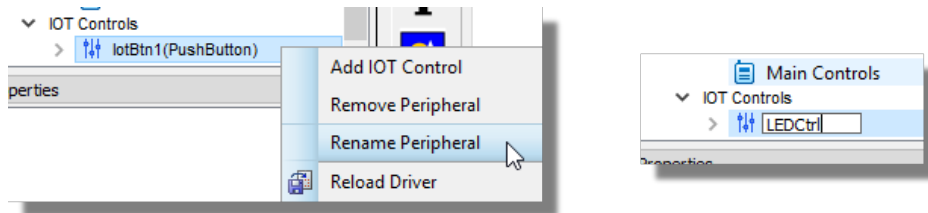
## **Design Phase**

---

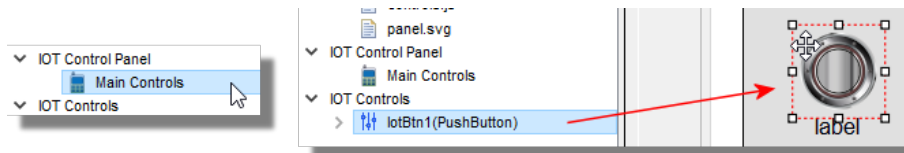
The design of the front panel is identical to that described in the Visual Designer tutorials. You add IoT Controls via the right click context menu on the project tree.



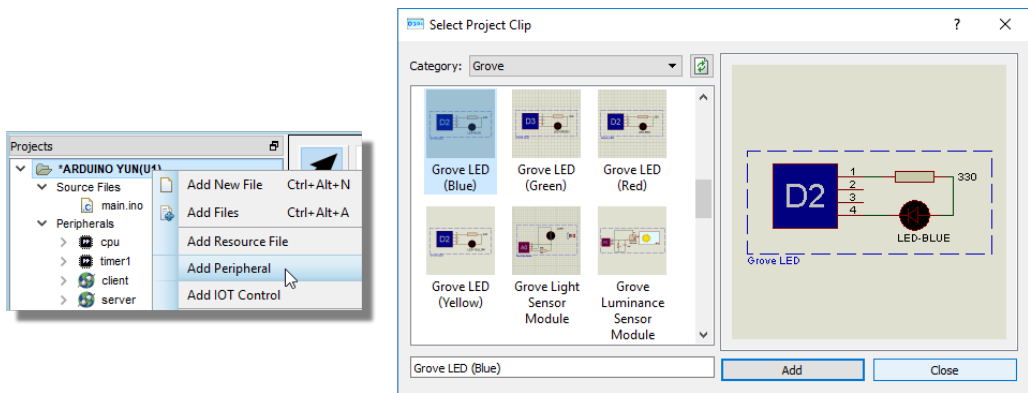
It's always a good idea to immediately rename the button (right click on in in Project tree -> rename command) to something that better reflects it's purpose



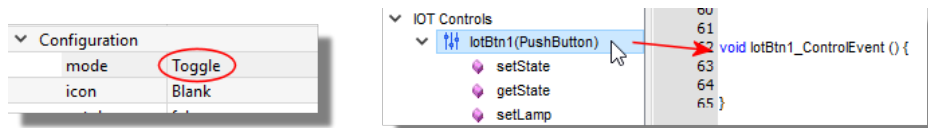
You double click on the front panel to open the panel editor and then you drag and drop to place.



You can also add hardware to your schematic directly from the Add Peripheral dialogue - again on the right click context menu.

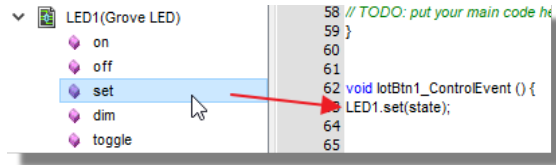


Now in the firmware to respond to the button press we need an event handler. Set the button Mode properties in the Main Control to be 'Toggle'. Now, double click on the Main.ino file and set the mouse caret in an empty space (after the void user\_loop function) and then drag and drop from the IoT button control onto the source code window. You should get a skeleton control event function into which we can add our button processing code.

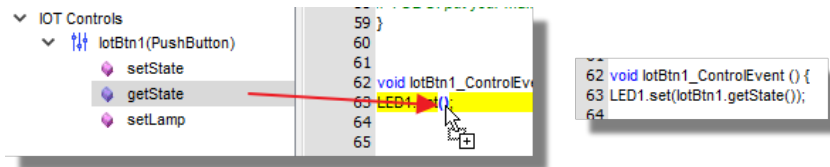


**⚠ Do not change the names of functions or you will break the program. The exception is when you rename the control or peripheral after placing functions on the source window. In this case you'll need to rename the objects and handler prefixes manually or drag and drop them back out again.**

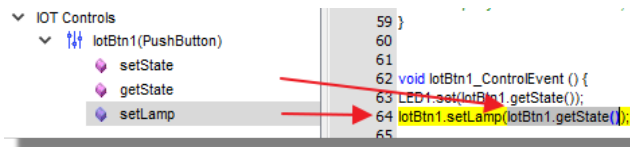
What we want to do is toggle the LED based on the state of the button. From the LED peripheral drag and drop the set routine but this time hold the CTRL button down before you drag. Move the mouse inside the button control event routine and then let go to insert at the current mouse position.



Having dropped the set routine we need to add the boolean parameter depending on whether we are turning the LED on or off. The simplest thing to do is to insert the getState() method of the button which returns the bool that we want - again you can do this by deleting the word 'state' from inside the brackets and then drag and drop the getState() method from the IoT Button control in the project tree.



Having not already set the autoLamp() method on the button itself, we can do this in the code. Drag the setLamp() method below the above code and then again insert the getState() method on to the brackets and set as follows:



This enables the lamp light to do the same as the LED indicating to the user whether the led is on or off.

That's pretty much it. There are other things we could add but basically this will toggle the grove LED on the Arduino Yun as you press the button on your controller (e.g. mobile phone). We strongly recommend you review the tutorials as the simulation, debug and deployment phases are all near identical to those for a flowchart program.

### Summary:

- Don't change or move the generated code at the top of the file.
- If you drag and drop normally the code snippet will insert wherever the mouse caret is at in the file.
- If you hold the CTRL button down while dragging then the mouse caret will move with the drag and the code snippet will be inserted at the caret position when you release the mouse.

# PROGRAMMING THE HARDWARE

## Overview

When the program has been written, the front panel designed and the system simulated and tested the final step is to program the physical hardware. You can do this directly from inside Proteus as discussed below.

## Raspberry Pi 3

Raspberry Pi programming involves an initial, one time configuration phase and then a programming procedure that you follow every time you want to change the firmware.

### Configuration

The one-time configuration process is documented in full in the Visual Designer manual and a [short video](#) tutorial is also available.

### Programming

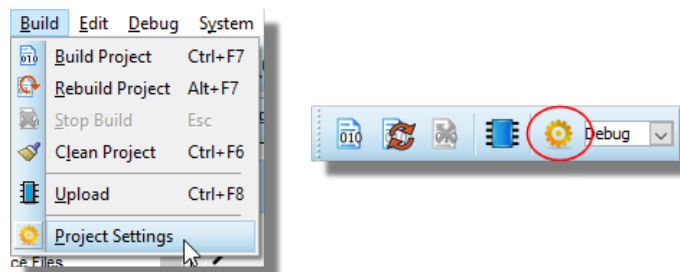
Once configured the actual act of programming is really simple and is documented in the Visual Designer manual as per the link below

## Arduino Yun via SSH

The Arduino Yún contains both an Atmel ATmega32u4 and an Atheros AR9331. Digital pins 0 and 1 are used for serial communication between the two processors. Your program firmware will execute on the Atmel CPU while the Atheros processor (running Linux) will host the webserver, front panel controls and associated IOT resources.

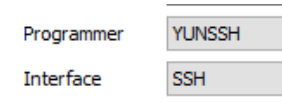
It is important at this point to make sure that your Yun and your PC are on the same wireless network. If this is a Yun 'straight out of the box' you and the Arduino is plugged in to a power source, it will appear under your WiFi settings as "Arduino-Yun-XXXXXXX" where XXXXXXX is the serial number of the Yun.

Configuration of the project for programming takes place from the project settings dialogue. First launch the dialogue form either from the Build menu or from the icon at the top of the editor.



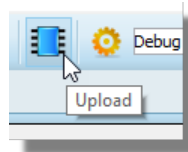
Next, set the programmer to be Yun SSH and the interface to be SSH.





The default address, user and pass are pre-populated but will need changing if the configuration of your device has been changed. If they have been changed you will need to type in the name of the host (arduino.local.) in a browser, enter in the password and change and then in the settings set up the Arduino correctly.

After configuration, make sure the Yun device is switched on and is out of reset and then program via the button at the top of the editor.



You will see progress updates in the output window as programming takes place - note that, depending on the size of the resources involved, this could take a little time.

```
Writing | ##### | 100% 28.36s

avrdude: 26978 bytes of flash written
avrdude: verifying flash memory against Debug.elf:
avrdude: load data flash data from input file Debug.elf:
avrdude: input file Debug.elf auto detected as ELF
avrdude: input file Debug.elf contains 26978 bytes
avrdude: reading on-chip flash data:

Reading | ##### | 100% 27.17s

avrdude: verifying ...
avrdude: 26978 bytes of flash verified

avrdude done. Thank you.

Firmware upload COMPLETE.
```

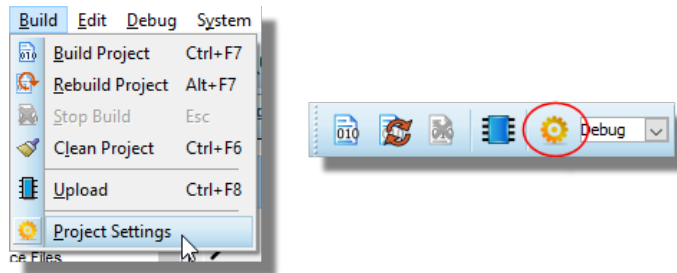
### **How does it work ?**

When you press the program button the following actions take place:

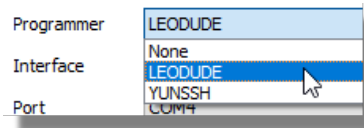
- 1) The resources are transferred to the target path folder (/vsm by default) on the Atheros.
- 2) The firmware is transferred to the /tmp folder on the Atheros.
- 3) AVR-DUDE is invoked (on the Yun side) to program the firmware into the AVR. It does this using the ICP interface (SPI).

## Arduino Yun via USB

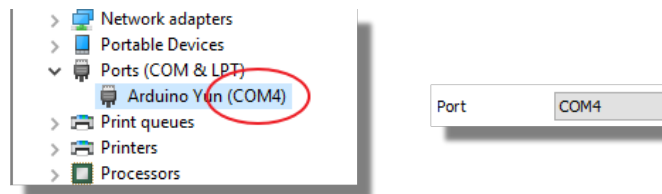
Similarly to the SSH method or programming, configuration of the project takes place from the project settings dialogue. First launch the dialogue form either from the Build menu or from the icon at the top of the editor.



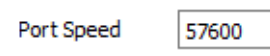
Set the programmer to be LEODUDE and the interface to be Arduino Yun.



At this point the only other fields available are the Port and the Speed. The port will be whatever your PC creates for you. i.e. COM4. If there are more than one COM ports in the list, you can find which one the Yun is using from the Windows Device Manager.



The port speed by default for the Yun is 57600 bps.



After configuration, make sure the Yun device is switched on and is out of reset and then program via the button at the top of the editor.



You will see progress updates in the output window as programming takes place - note that, depending on the size of the resources involved, this could take a little time.

```
Writing | ##### | 100% 28.36s
avrdude: 26978 bytes of flash written
avrdude: verifying flash memory against Debug.elf:
avrdude: load data flash data from input file Debug.elf:
avrdude: input file Debug.elf auto detected as ELF
avrdude: input file Debug.elf contains 26978 bytes
avrdude: reading on-chip flash data:

Reading | ##### | 100% 27.17s

avrdude: verifying ...
avrdude: 26978 bytes of flash verified

avrdude done. Thank you.

Firmware upload COMPLETE.
```

### *How does it work ?*

When you press the program button the following actions take place:

- 1) The bootloader on the AVR will be used to pass all of the resources across the data bridge to the Atheros chip.
- 2) Resources will be deposited in the target path folder (/vsm by default).
- 3) AVRDUDE on the PC will be invoked to program the firmware onto the AVR processor.

### **Computer / Network / Lab Setup**

---

Depending on the features of your computer and/or the number of PC's and Yun boards in the room It may be necessary to do some configuration.

- 1) If a PC has wi-fi then it can connect to the Yun directly via the procedure described above. The MAC address for the Yun (normally found as a sticker on the shield/board) will be appended as part of the device name allowing you to identify it.



2) If the PC's do not have wi-fi then some configuration is going to be needed. We'd suggest you have a wireless access point/router connected to the office/class LAN. Plug the Yun in to the network and it will obtain a dynamic IP address from a DHCP server. After that, you should be able to see the Yun hardware from the PC and program according to the procedure above.

3) Alternatively, you can cable the individual Yun boards to the PC's and program via USB as described above. Assuming the Yun is in default configuration (Access point) then the phones / tablets will connect to the real hardware after programming. You may end up with a lot of micro-networks (mobile device and Yun) and people will have to check mac addresses to connect to the correct Yun but it will work fine.

4) If all fails remember that you don't need each computer to program physical hardware. Proteus VSM has complete simulation and debug of the electronics and the front panel so a single programming station is both common and perhaps more practical.

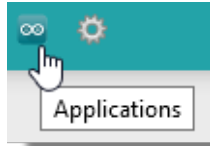
**i** If you are using an Access point bear in mind that the wifi on the Yun baseboard is fairly poor and, with a noisy network, the accessible range between the device and the access point may be short. Our experience is that a Yun shield on an Uno board is better than a Yun baseboard but see also note 3) above.

**i** Note that In some countries, it is prohibited to sell WiFi enabled devices without government approval. While waiting for proper certification, some local distributors are disabling WiFi functionality. Needless to say, this will prevent the programming of the Yun via this method and also render it useless as an internet of things appliance. Check with your dealer before purchasing a Yun if you believe you may live in such a country.

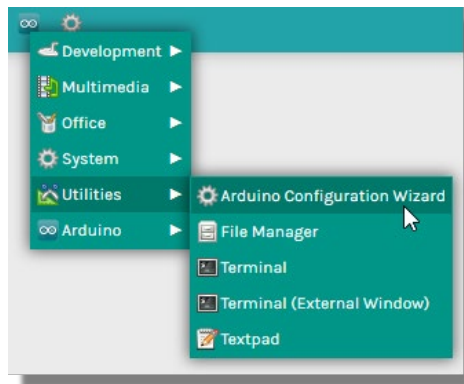
## Setting The Arduino Yun On To Your Network

---

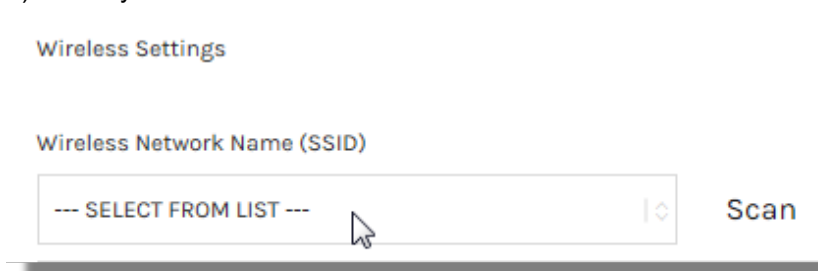
- 1) Power the Yun with a USB cable
- 2) Plug an ethernet cable in from your router / network point
- 3) Open a web browser and type in its host: arduino.local.' (including the last full stop). This will prompt you for a password
- 4) Enter the password: arduino or doghunter to enter the Arduino's settings
- 5) If the Arduino Configuration Wizard does not appear automatically, click on the Arduino Applications icon



- 6) Go to the Utilities > Config Wizard



- 7) Click Next until you reach the Wireless settings page
- 8) Select your network from the list



- 9) The security will be automatically selected and you can enter in the password of your Wireless connection as you would if you were connecting your phone to the same network.
- 10) Click next until you get the save option, and save. Your Arduino is now connected to your network and will automatically connect to it when turned on.

# THE MOBILE APP

## Overview

---

In this day and age, most people have a SMART phone which runs an app / program. We have design the Proteus IoT Builder App which is available in the Apples APP Store or Googles Play store.

## Download and Installation

---

### *Android App*

On your Android phone, open Google Play and search for 'IoT Builder' and install

### *iOS App*

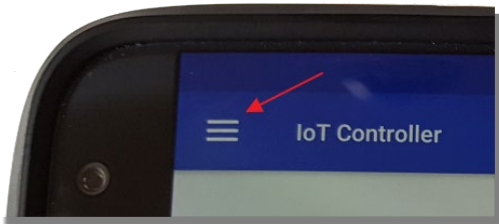
On your iPhone phone, open your App Store and search for 'IoT Builder' and install.

**i** You need to have the Bonjour utility installed on your PC in order for the App to control the running simulation. If it is not already installed you can install it directly from the Labcenter Program Group on the Start Menu.

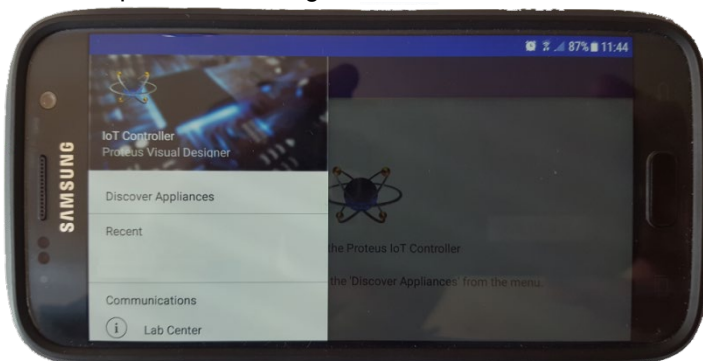
## Discovery

---

There is one menu which is at the top left:



This will open the following menu:




## LABCENTER ELECTRONICS LTD.

---

Click on 'Discover Appliances' and it will begin searching for any running Proteus simulation on the same network. It uses Zeroconf/Bonjour (as supported by Android or iOS) to look for any appliances (built with IoT Builder) on the local network. Both actual appliances and VSM simulations announce themselves using Zeroconf so hence the app can find either.

When it locates a simulation, it will load its panel on to the main screen. Select this to run and control the simulation on your phone. To close the app, simply click the back button or your home screen button.

 Note that this does not stop the simulation on the Hardware or the PC. This needs to be done either by unplugging the Yun or stopping the simulation in Proteus.

 Contact Information and the App version number is found at the bottom of the menu under 'Labcenter'.

# ADVANCED PANEL DESIGN

## How a Virtual Front Panel Works

---

While IoT Builder contains quite powerful editing tools and a set of themes for different control styles, you can if you want customize both the panel and the controls even further in a dedicated graphics package. The virtual controls on the front panel are implemented using SVG graphics elements controlled by Javascript classes. The root file (panel.htm) contains a javascript framework which pulls in panel.svg, panel.js and controls.js from the server. The entire of the front panel – even if it has multiple tabs – is held in a single SVG file – panel.svg. The SVG format used in panel.svg is augmented with additional data that is stored under the vfp namespace. However, for all this, the panel file is just a normal SVG file and can be edited using an external SVG editor.

## Using Inkscape to Edit the Panel

---

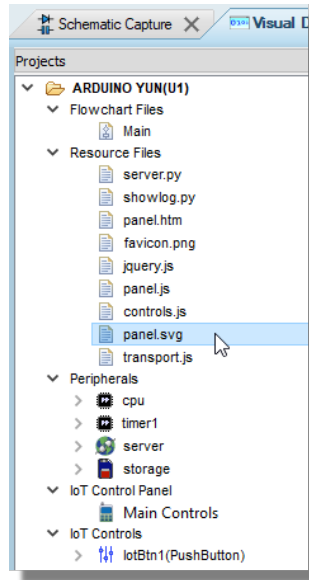
Although it is possible to use any general purpose SVG editor to edit your panel file, we would strongly recommend that you use InkScape. It is free, and it is the tool we use internally to create the controls themselves so we know that it works well.

There are several reasons why you might want to edit the panel externally:

- To add complex graphics such as gradients which are not supported by the built in editor.
- To reposition the controls using more sophisticated snapping tools and guidelines.
- To make minor changes to the appearance of the controls.

Assuming that you have created a file association between InkScape and SVG files, then double clicking panel.svg under the resource section of the Project Tree will launch InkScape to edit it.

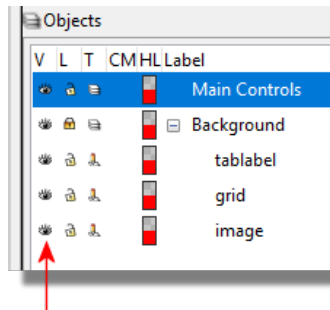




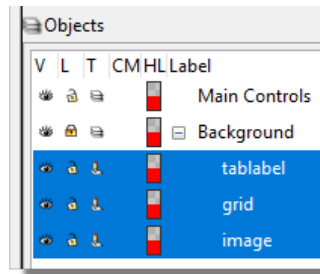
*Double Click on the panel resource in Proteus to launch it in Inkscape.*

However, before you do this, it's important to understand that it possible to destroy your front panel if you do the wrong things with the external editor. The key thing to avoid is disturbing the group structure of the SVG file. Put simply, this means that in no circumstances must you ungroup the controls. If you do that, you will need to re-place them manually inside Proteus.

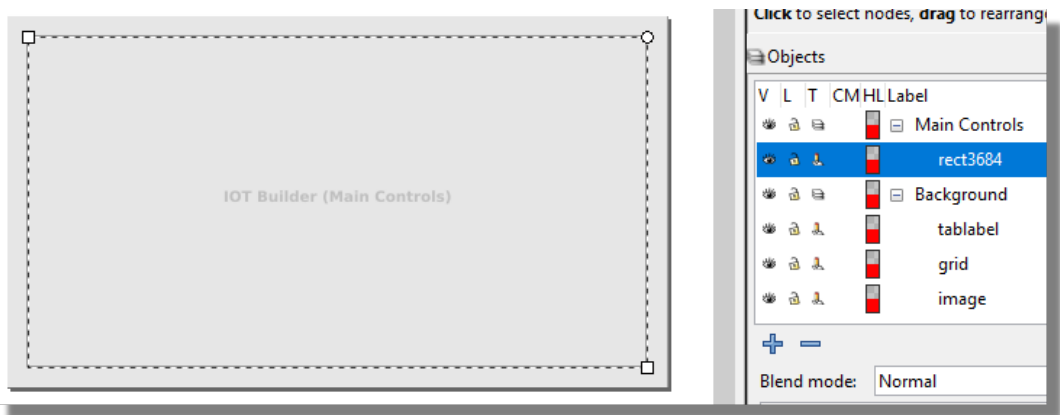
When you first open the panel.svg file in InkScape, you'll see that each tab of the panel is held on a separate layer, these corresponding to top level groups in the SVG file structure. Note that you may need to use the visibility flags in the Layers Dialog to show the other tabs.



Then, on each layer the controls will generally appear as second level groups, at least unless you have used the grouping function in the build in editor. As long as you retain this basic structure, the panel will reload back into Proteus in full working order.



If you want to add extra graphics or text to the panel, you can just draw them using the Inkscape tools. If you place them on the background layer, they will appear on all the tabs, otherwise they will appear on the tab corresponding to the layer that you place them on. Basically, if you draw graphics on the the Main Controls layer, it will appear on the main panel in Proteus. If you draw graphics on the Background layers, it will appear on all panels that are added to the Proteus project.



On the other hand, if you want to make changes to elements within a control you can do this by holding down ctrl whilst clicking on the element. This selects an individual element within the group structure without the need to ungroup it first. You can then change stroke/fill styles, font attributes and so on for that particular element.

As soon as you save the panel.svg file back from Inkscape, Proteus will detect the change and reload it automatically.

**⚠ Do NOT ungroup any of the IoT controls that are on the panel. If you do, the resulting SVG will not work in Proteus.**

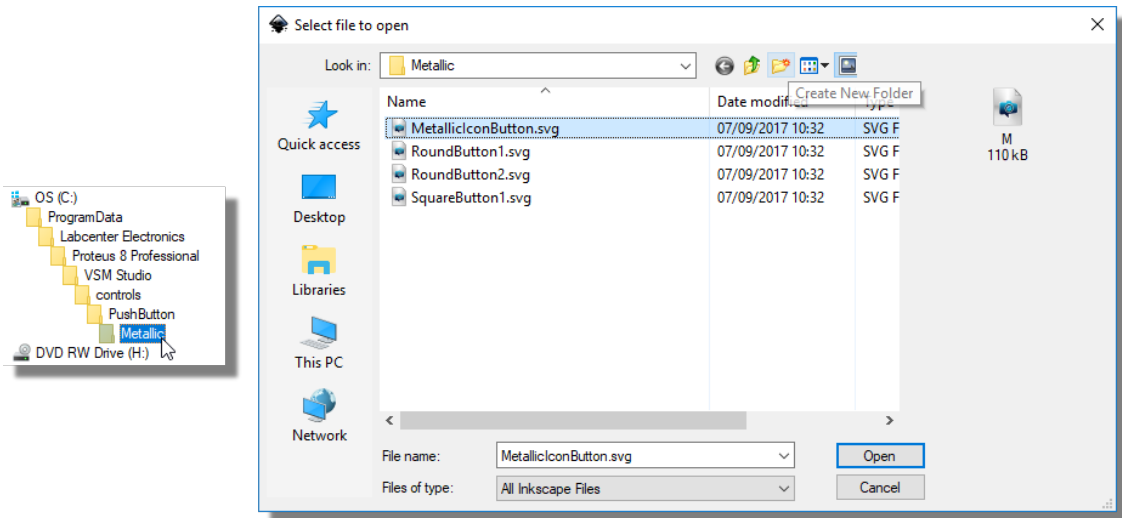
## Control Editing

Unlike editing of the panel, the editing of controls is quite involved and requires an understanding of how the control is constructed. If you are interested in creating your own controls then please e-mail us at [support@labcenter.com](mailto:support@labcenter.com) and we can provide you with more detailed technical information.

Meanwhile, the following example shows how you can make a minor change to change the iconography on the icon buttons.

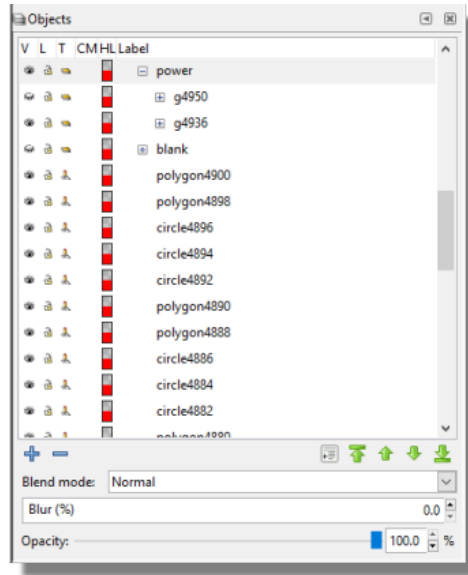
### Creating a Custom Icon Button

We'll assume at this point that we are in the IoT Builder editor and have added an icon button to our panel. We've looked through the different icons and actually we want a happy face icon in the button. This isn't here but we do have custom icon option so we can make our own. Open your editor and then navigate to the Controls directory (inside \ProgramData\Labcenter Electronics\VSM Studio\ of your installation), select Pushbutton, then the theme you want and finally open the SVG file for the icon button.



**i** You need to open the icon button, not the standard button. It will have icon in it's name.

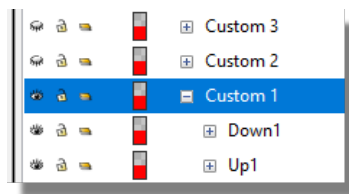
Open the objects panel to view the root structure of the SVG file. Expand the structure tree for a full view of groups and layers.



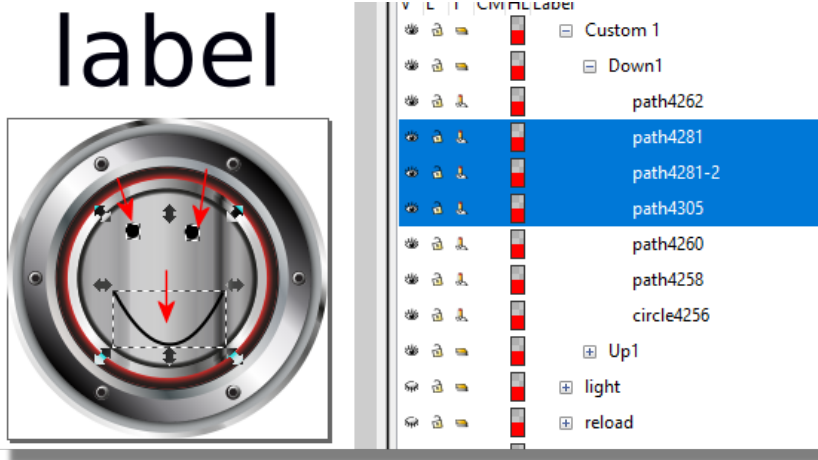
**⚠ Do not ungroup any objects.** If you do so the control will either not load in Proteus or not work properly if it does.

**ℹ Hide any layers (i.e. Power) that are currently visible and unhide your Custom1 layer so that any graphics that you add will appear.**

Locate the Custom layer (1, 2 or 3) you wish to modify from the structure tree and expand it. You will find two more groups inside this Custom group, the up and down states of the button. Expand either the Up or Down group and add a new box, line, arc, Bezier etc. objects as needed to create your icon. You can then copy these in to the other group and move accordingly to match the first groups positioning. That way the button will appear stationary when pressed. Multiple objects can be used but they must not be grouped together.

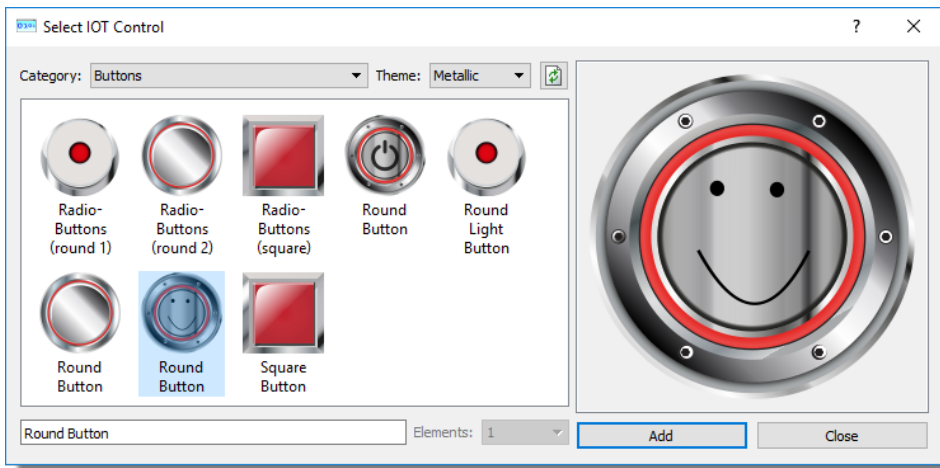


Here you can see the arrows pointing to the added graphics, with them selected on the right hand side:



Save the SVG as a new file. It is strongly recommended you create a new folder within the control folder to store all of your custom SVG's. Doing so will mean all custom SVGs will show as their own skin group when viewing in the IoT control panel. This skin group will get it's name from the Folder name e.g Modern, Retro etc.

**⚠ Hide the Down state before exporting the SVG file otherwise your button will not appear correctly in the IoT Control form:**



**ⓘ Any changes/new icons saved in the installation folders provided with Proteus will be overwritten and updated with new releases of Proteus. Custom folders will remain intact.**

# IoT CONTROLS

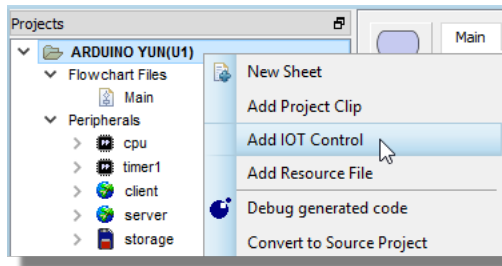
## BUTTONS

### Introduction

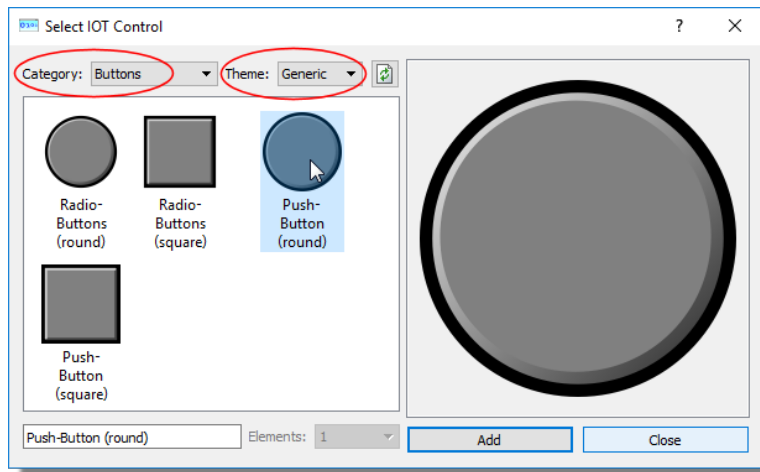
The button is probably the most common and certainly one of the most versatile controls in the IoT Control Library. This topic discusses the different types of button and various configuration options but the basic process for adding a button is the same in all cases.

### To add a button

1) Right click on the Arduino Yun in the project tree and select the Add IoT Control command from the resulting context menu



2) Make sure the category selector shows buttons and choose your theme from the selector on the right.



**i** The theme determines the available styles of buttons but not their functionality and lets you create a modern looking GUI or a classic retro panel according to your preference.

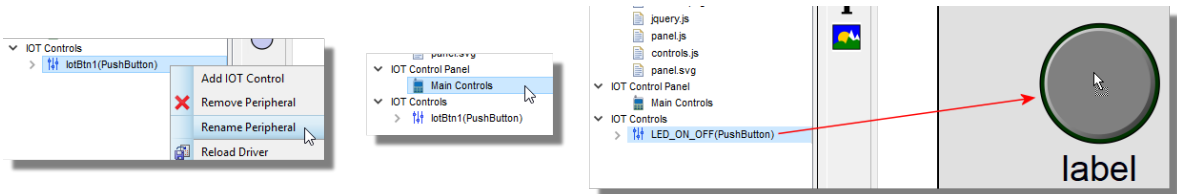
3) There may be three types of button, namely radio buttons, icon buttons and standard buttons.

- Radio buttons are a mutually exclusive option group which you can easily query to see which option is selected. If you want radio buttons select the button type and the number of elements and then click to add.
- Icon buttons are normal buttons pre-loaded with a selectable icon such as power, play, stop, etc. If you want an icon button just click to select and then click the add button to bring it into your project.
- Standard buttons are all of the other buttons and may come in different shapes and styles. If you want a standard button follow the usual steps of click to select and then add button to bring into your project.

## **Design Time Configuration**

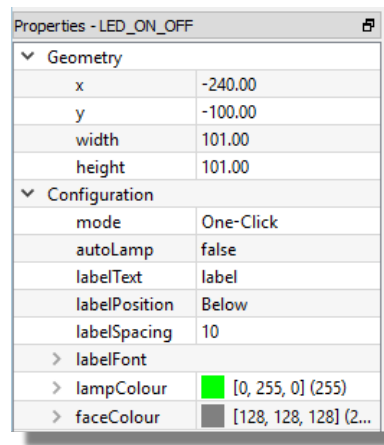
---

Configuration of the buttons can be done primarily at design time once the button is placed on the panel. We'd suggest you first rename your button to suit it's purpose, then double click on the Mains Control panel to show and finally drag and drop the button control onto the panel.



### *Renaming and placing a button*

Select the button by left clicking on it and then view the properties in the property pane at the bottom left of your editor.



### *Button properties*

The following properties are commonly available:

### [Geometry Properties \(x, y, width, height\)](#)

Properties - LED_ON_OFF	
▼ Geometry	
x	-240.00
y	-100.00
width	101.00
height	101.00

These properties enable positioning and alignment. For example you could set a series of controls to have the same x value to align them vertically, The width and height properties allow for sizing of the control, although you may find it just as easy to use the 'restore aspect ratio' command on the right click context menu for the control after sizing with the mouse.

### [Mode Property](#)

Configuration	
mode	One-Click
autoLamp	One-Click
labelText	Momentary
labelPosition	Toggle
	Relay

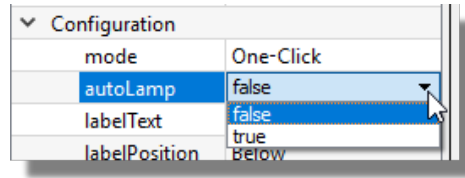
There are three types of button:

- **One-Click Mode:** The button has a single state (clicked). Your event handler in the firmware is called after the click is completed.
- **Momentary Action:** The button can be depressed and held down (the momentary action) but will return to its default state on release. Your event handler in the firmware is called when the button is depressed and again when it is released.
- **Toggle:** A binary button with two states (e.g. on and off) which are toggled on click. Your event handler in the firmware is called whenever the button is clicked and you can use the `getState()` method to query whether state is 1 or 0.

The choice of button mode depends on what you want to happen when you press the button. If for example you have a reset button then you would have a one-click button because each time you press that button you want to reset the app. If you want to turn the volume up you might use a momentary action button which will respond while the button is depressed. If you want to toggle something on and off then you want to use a toggle button because you can query the state of the button at any time.



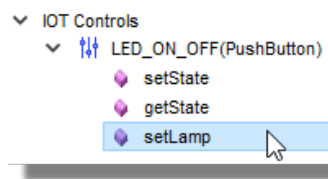
### autoLamp Property



The buttons all have a lamp indicator ring which can be illuminated when they are pressed/active. This provides useful visual feedback on the UI device and can be automated via this property. The behaviour of the autolamp property depends on the type of button chosen as follows:

- One-Click Button: The lamp indicator will come on when the button is depressed and go off when the button is released.
- Momentary Action Button: The lamp indicator will come on when the button is depressed and go off when the button is released.
- Toggle Button: The lamp indicator will come on when the button is clicked and will stay on until the button is clicked again.

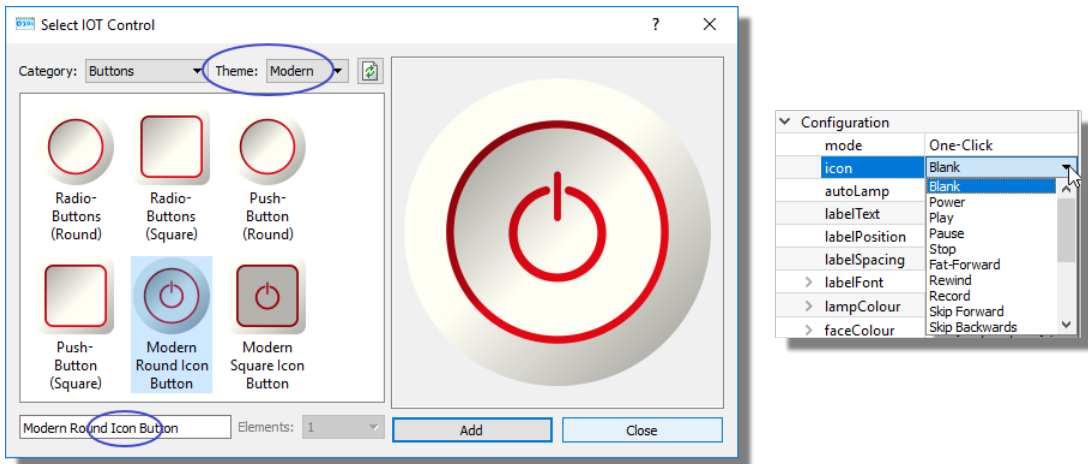
It follows that using the autoLamp property makes the most sense when you have a toggle button because the lamp indicator can automatically reflect an on and off state. The default behaviour however is to have autolamp turned off because a more complete approach to setting the lamp indicator is to do so explicitly in your program via the setLamp() method.



#### *setLamp() method*

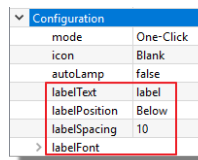
The benefit of this is that it is the firmware running on the appliance that dictates the state of the lamp which is then guaranteed correct. While it is very convenient to use the autolamp property here be aware that it will work with no interaction with your appliance (i.e. entirely in the GUI).

### Icon Property (only available with Icon buttons !)



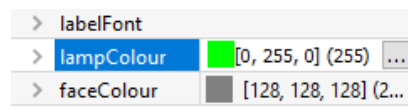
If you have chosen to use an icon button then you can specify which icon to use from a pre-selected list or - with some effort - you can add a custom icon of your own. To add a custom icon you will need to edit the SVG file natively. This is not a trivial operation but the process is discussed in some detail here.

### Label Configuration



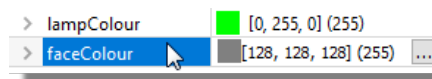
These properties are all fairly self-explanatory, allowing you to enter label text, position the label relative to the control and then control the spacing of the label from the control. You can also choose from a small selection of font types and set font size, decorations and colours.

### Lamp Colour



Specify the colour of the indicator lamp on the button. This is used when you execute the setLamp() method in your program or if you set the autolamp property to be true.

### Face Colour

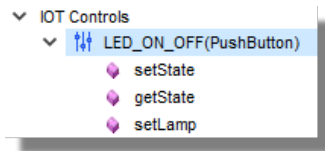


Specify the colour of the button face, excluding the lamp indicator ring.

## Programming Methods

---

Having configured the buttons we still need to access them in our program. There are three available methods (only two for radio buttons) which we can see by dropping the tree down a level in the project tree.




*All 3 possible methods*

Depending on what product you have these methods can be dragged and dropped into your flowchart program (as a method call block) or into your source code project (as a skeleton function).

### [setState\(\) Method](#)

The setState() Method applies only to the toggle button type (because that is the only type that has two states) and allows you to set a pre-defined state without executing an actual button press. For example, if you had a logging button which you had set to be of type toggle (on and off) then you might decide that the default behaviour should be to log data. In this case you would use the setState() method in your setup routine so that your appliance was logging and the first user press of the button served to disable logging.

 See Also: [Mode Property](#)

### [getState\(\) Method](#)

The getState() method again applies only to the toggle button and returns the current state of the button as a boolean.

### [setLamp\(\) Method](#)

The setLamp() method serves to turn on or off the indicator lamp on the button. This is intended for use when the autoLamp property is set to false and allows the firmware to control the feedback to the user on the controller.

If instead you use the autoLamp property then you don't need to use this method at all but the indicator on the users panel will simply toggle, irrespective of whether the button press command is received and processed successfully.

## SWITCHES

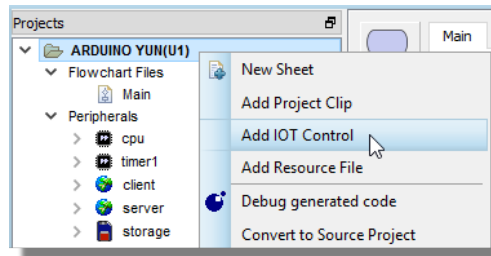
### Introduction

Depending on which theme you choose there are three possible types of switches:

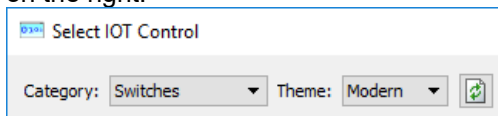
- Simple two state switches (flick switches, toggle switches etc.)
- Slide switches (multi-state)
- Rotary switches (multi-state)

### To add a switch

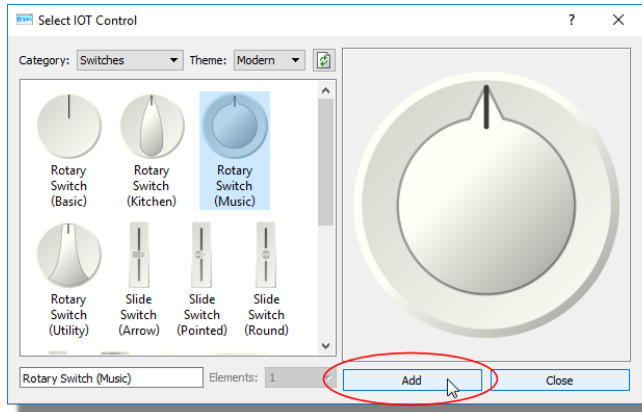
1) Right click on the Arduino Yun in the project tree and select the Add IoT Control command from the resulting context menu



2) Make sure the category selector shows Switches and choose your theme from the selector on the right.

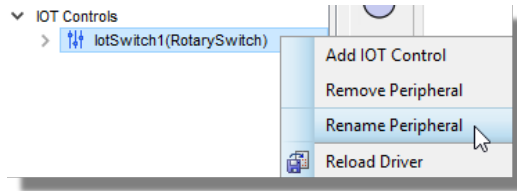


3) Either double click on the switch to add it directly or click to select and then press the add button to add it to the project.

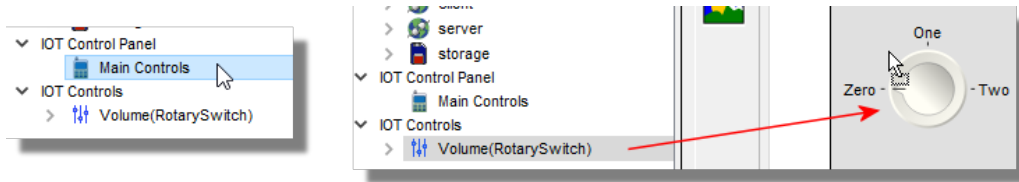


## Design Time Configuration

The first thing to do is to rename the switch to reflect its purpose in your design. You can do this via the right click context menu.



To place the switch on your panel first open the panel editor by double clicking on the IoT 'Main Control' Panel in Project Tree. Next, drag and drop the switch onto the panel and position as required.



To configure the behaviour of the switch select it on the panel and then adjust the properties in the property editor pane.

Properties - Volume	
Geometry <ul style="list-style-type: none"> <li>x: -107.00</li> <li>y: -165.00</li> <li>width: 213.00</li> <li>height: 136.00</li> </ul>	
Configuration <ul style="list-style-type: none"> <li>init: 0</li> <li>numStates: 3</li> <li>tickLabels: Zero,One,Two</li> <li>tickLength: 5</li> <li>amin: -90.00</li> <li>amax: 90.00</li> <li>showTicks: true</li> <li>showLabels: true</li> <li>tickColour: [0, 0, 0] (255)</li> <li>labelFont: &gt;</li> </ul>	

### Toggle Switch

The toggle switch has geometry configuration allowing to precisely size the control using width and height fields and also to align with other controls by applying a common x or y value.

Properties - IotToggleSwitch1	
Geometry <ul style="list-style-type: none"> <li>x: 180.50</li> <li>y: -69.50</li> <li>width: 67.00</li> <li>height: 95.00</li> </ul>	

It also includes two labels which allows you to add descriptive text for both states of the switch (e.g. on and off )

Configuration	
labelText1	label1
labelText2	label2

You can also change the orientation between vertical and horizontal and specify a spacing for the labels from the control body.

Configuration	
labelText1	label1
labelText2	label2
orientation	Vertical
labelSpacing	Vertical
labelFont	>

It is much better to set this property than to rotate the control because the text labels will position themselves correctly when you adjust with the property.

Finally, you can configure the font size and font type for the text labels via the labelFont properties.

▼ labelFont	
family	sans-serif
size	27
weight	normal
style	normal
decoration	none
▼ colour	■ [0, 0, 0] (255)
Red	0
Green	0
Blue	0
Alpha	255

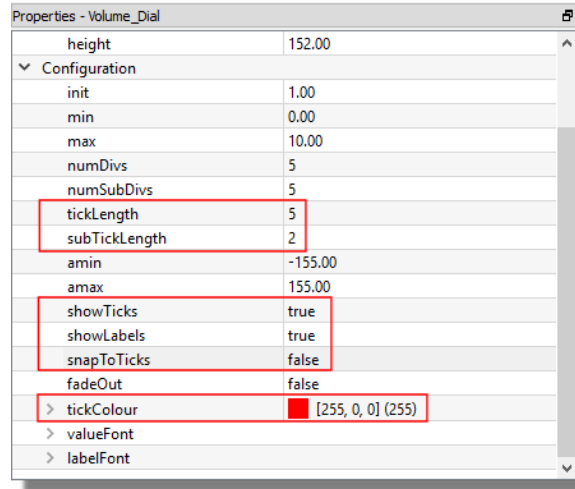
### Rotary Switch

In addition to the properties for toggle switch the rotary switch contains additional configuration for the multiple states:

The init property lets you set the state at which the dial should start, the numStates property defines how many states the switch will have and the amin/amax properties define the angle range for the dial. For the angles, consider straight up to be angle=0, left is negative and right is positive. So for example amin=-90 and amax=90 uses the top half of the dial.

Properties - Volume_Dial	
height	152.00
▼ Configuration	
init	1.00
min	0.00
max	10.00
numDivs	5
numSubDivs	5
tickLength	5
subTickLength	2
amin	-155.00
amax	155.00

We also have some more aesthetic properties for specifying the length of the little ticks leading out of the dial (tickLength), changing their colour (tickColour) and for disabling them altogether (showTicks), along with the tickLabels property to change the text of the labels as a comma separated list or choose to hide the labels (showLabels).



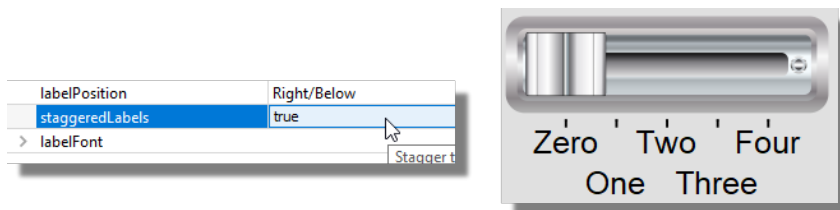
### Slide Switch

The slide switch is very similar to the rotary switch. It has the basic set of properties we saw in the toggle switch, including the orientation property which allows us to set the bar either vertically or horizontally. There are then a couple more properties related to label positioning. You can choose to position to the right if vertical which will then be below if horizontal or you can choose to position to the left if vertical which will position above when the orientation is horizontal.

orientation	Vertical
labels	Zero, One, Two, Three, Four
labelPosition	Right/Below

**i** The movement and positioning of the labels will work properly only if you change the orientation of the control via this property. Rotating the control will not adjust the labels.

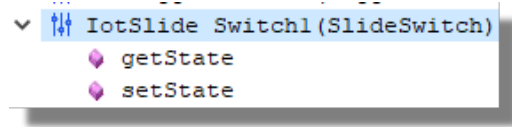
You can also choose to stagger the labels onto two rows which can make it much easier to make text readable with horizontal orientation.



## Programming Methods

The programming methods allow us to access and control the peripheral at run time. For the switch there are only two methods, namely `getState()` and `setState()`.





### [getState\(\) Method](#)

Reads the value of the switch position as an integer value starting from 0. eg a toggle switch has two states 0 and 1. A slide switch could have multiple in the format of 0,1,2,3..... with 0 being the bottom most state. A rotary switch has a 0 state on the location furthest in the negative degree direction.

### [setState\(\) Method](#)

Programmatically changes the state of the switch and causes the control to move to indicate the new condition.

## DISPLAYS

---

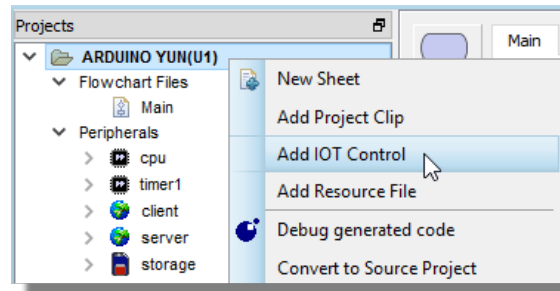
### *Introduction*

There are basically six types of display controls that you can select:

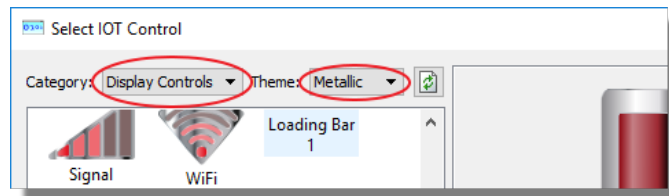
- Moving dial controls (e.g. panel meter, speedometer).
- 7-Seg display control
- Mercury thermometer control
- Loading / Percentage bar.
- LED Strip
- Gauge

## To add a display control

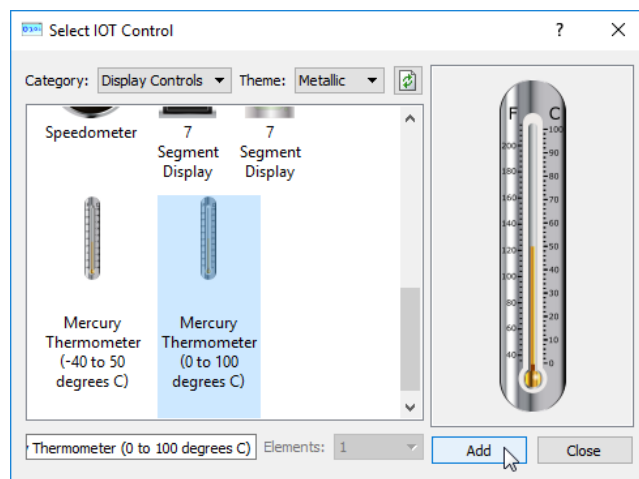
1) Right click on the Arduino Yun in the project tree and select the Add IoT Control command from the resulting context menu



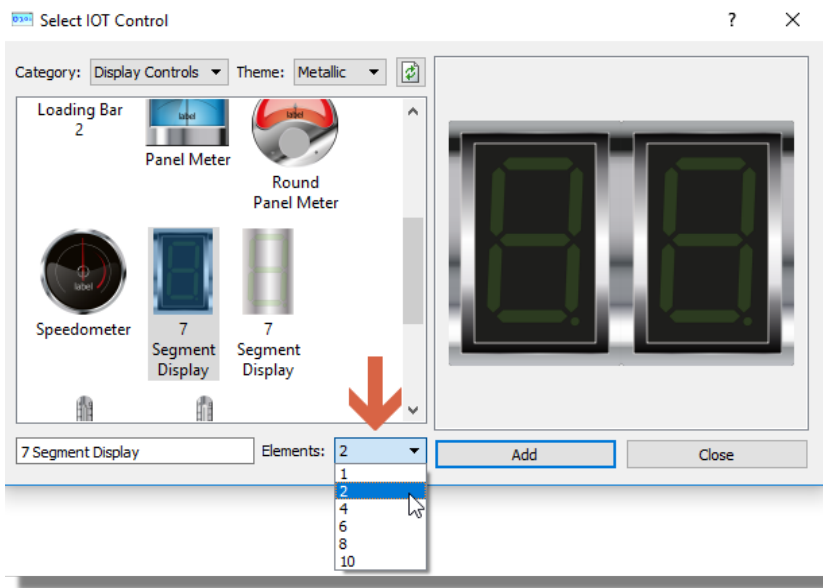
2) Make sure the category selector shows Display Controls and choose your theme from the selector on the right.



3) If you are adding a moving dial control or a thermometer you simply click to select and then press the add button to add it to the project.

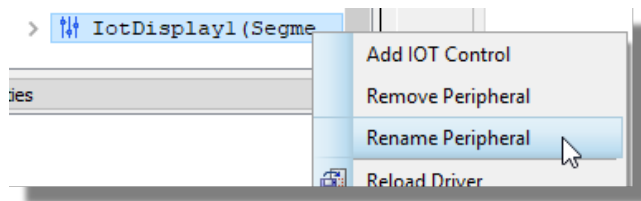


If you are adding a 7 segment display then you need to select the control, specify the number of elements at the bottom and then add it to the project.



### Design Time Configuration

The first thing you should do is to rename the control to reflect its purpose in your design. You can do this via the right click context menu.



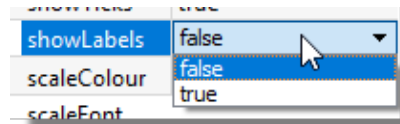
### Common Properties

There are a number of common properties that can be configured regardless of the type of display you have selected. First, we have the usual geometric options, allowing you to position and align the control

Properties - IotMeter1	
▼ Geometry	
x	-120.00
y	-90.00
width	201.00
height	201.00

- It is often easier and quicker to resize the control with the mouse (drag the handles) and then right click on the control in the panel and select transform -> restore aspect ratio from the context menu.

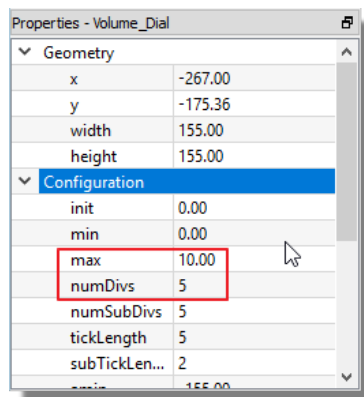
We can change the label by using the label property or choose to hide it altogether by setting the showLabels property to false.



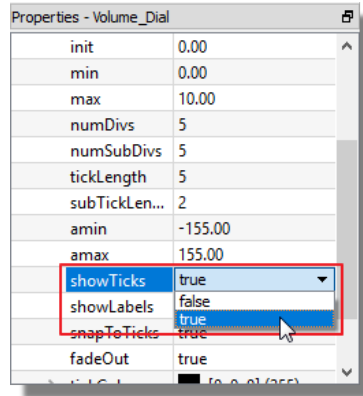
### Moving Dial Control Configuration

Depending on the theme you've chosen there may be a couple of different moving dial controls such as panel meters or speedometer dials but the configuration options are the same.

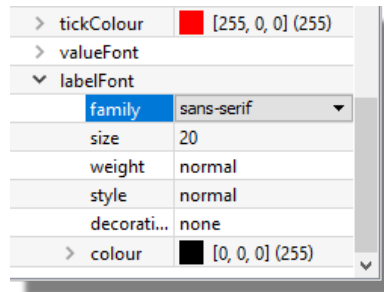
Configuration of the number range is a matter of setting the min and max properties and then the number of divisions between them. For example, if we set the range from 0 to 10 with numDivs=5 we would increment by 2 in each division as shown below



The other aspect to the number range is the sub-divisions. Here you can set the number of sub-divisions (numSubDivs) and the length of the indicator for each one (subtickLength). You can also choose not to show the tick indicators at all via the showTicks property



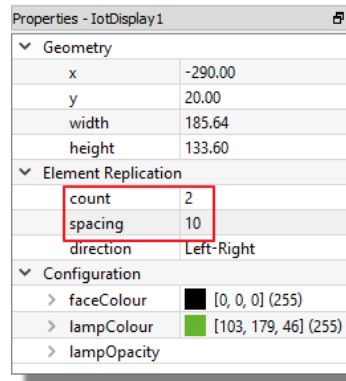
You can adjust both the colour and the font used in the number scale via the tickColour and scaleFont properties



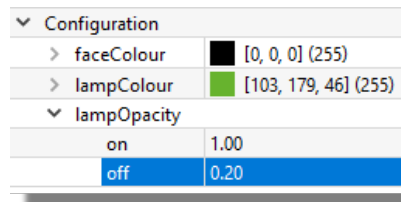
Finally, you can disable animation of the needle by setting animateNeedle to false. This adds a touch of realism to the display when in use by factoring in some basic acceleration and momentum effects.

### [Seven Segment Display Controls](#)

This type of control includes configuration options for the elements and the for the lamp used to illuminate the segments. You can choose (or change) the number of elements used in the control as well as the spacing between them and the direction of reading.



In terms of colour you can change the colour of the face, the colour of the lamp and also the opacity of the lamp in both it's on and off state. The opacity is a value between 0 and 1 with 1 being fully illuminated and 0 being fully off.



### [Thermometer Control](#)

The mercury thermometer display controls are trivial and, apart from geometric configuration, have a single property to specify the units as either Celcius or Farenheit.

### [Loading Bar Control](#)

This reads back a % value based on computational mathematics performed in the program code. A value between 0 and 100 can be represented. In addition to the common properties the following additional properties are available:

- init - the starting value of the loading/progress bar
- barColour - sets the colour of the indicator bar as it progresses
- animateBar - decides whether to animate the bar progress or not

### [LED Bar \(including WiFi, signal, battery indicators\)](#)

The following properties are unique to the LED bar type of display.

units	Decides whether bar lights up as percentage return or as a set number of steps. If using a percentage return, the % value of each LED is worked out from the total number on the strip. The LED will only turn on when the value reaches mid point between two LED percentages. The only exception to this is
-------	---

	the first LED which will always turn on as soon as it has a value assigned.
LEDonColour	Sets the on colour of the LED
LEDOffColour	Sets the off colour of the LED

### Gauge

The gauge displays have the following additional properties:

**tickDis** - Sets the distance of the value ticks from the centre of the gauge

**labelDis** - Sets the distance of the step values from the indicator ticks

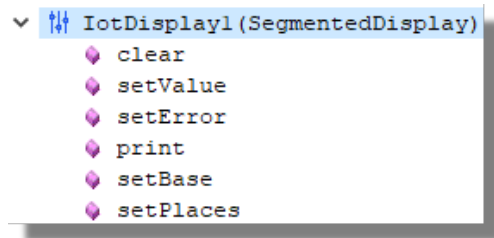
**emptyColour** - Sets the colour of the gauge when it is empty.

**gaugeColour** - Sets the colour as the gauge fills.

### Programming Methods

---

The programming methods allow us to access and control the peripheral at run time. The following methods are available in Visual Designer:



### Segmented Display

**clear** - clears all display segments

**setValue** - allows you to set a numerical value to be printed to the display

**setError** - allows you to display 'E' error warning on all segments

**print** - allows you to set a numerical, boolean or string argument to display on segments

**setBase** - allows you to configure display for Decimal, Binary, Octal or Hex values

**setPlaces** - allows you to define the number of decimal places the display allows

### Thermometer:

**setTemperature** - allows you to specify the temperature in a numeric value

### Dial Controls:

**setLabel** - Allows you to set a name for the display. This will override any label name you specify in the design properties when program is running

**setValue** - Same as segmented display

### [Loading Bar, LED bar, Gauge:](#)

**setValue** - As above

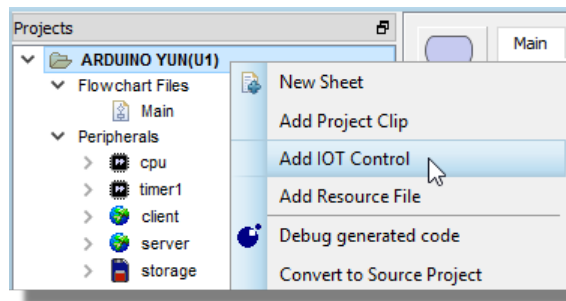
## INDICATORS

### *Introduction*

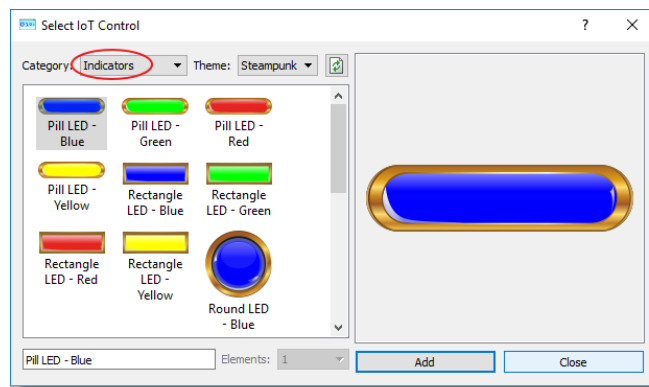
The indicators section includes LED's of various shapes and sizes for use in front panel design. They all share the same property set.

### To add an indicator

1) Right click on the Arduino Yun in the project tree and select the Add IoT Control command from the resulting context menu



2) Make sure the category selector shows Indicator and choose your theme from the selector on the right.

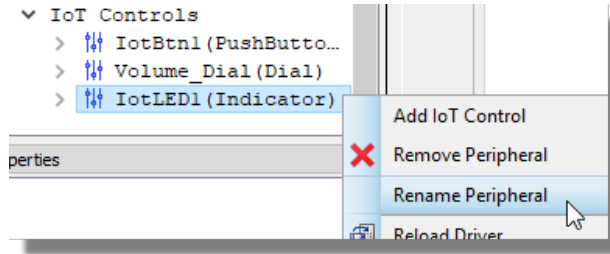


3) Either double click on the indicator to add it directly or click to select and then press the 'Add' button to add it to the project.



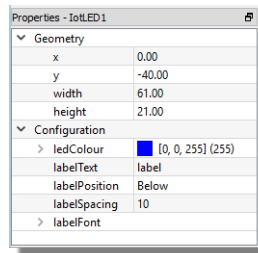
## Design Time Configuration

Configuration of the buttons can be done primarily at design time once the indicator is placed on the panel. We'd suggest you first rename the indicator to suit it's purpose, then double click on the Mains Control panel to show and finally drag and drop the button control onto the panel.



*Renaming and placing an indicator*

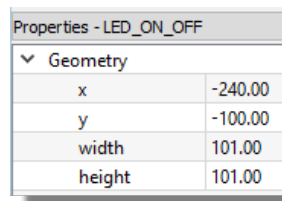
Once placed, select the indicator by left clicking on it and then view the properties in the property pane at the bottom left of your editor.



*Indicator properties*

The following properties are commonly available:

### Geometry Properties (x, y, width, height)



These properties enable positioning and alignment. For example you could set a series of controls to have the same x value to align them vertically, The width and height properties allow for sizing of the control, although you may find it just as easy to use the 'restore aspect ratio' command on the right click context menu for the control after sizing with the mouse.

### Other Properties

**ledColour** - where available this allows you to set a custom colour for the lamp

**labelText** - Allows you to set a text label to give meaning to the lamp, e.g. "On"

**labelPosition** - Allows you to determine where the label will show relative to the lamp, or not to show it at all

**labelSpacing** - sets the distance of the label from the lamp

**labelFont** - allows you to customise the text style of the label

## Programming Methods

The programming methods allow us to access and control the peripheral at run time. For an indicator there is only one method in Visual Designer:

**setValue** - allows you to set the state of the lamp in boolean terms. True is on and False is off.

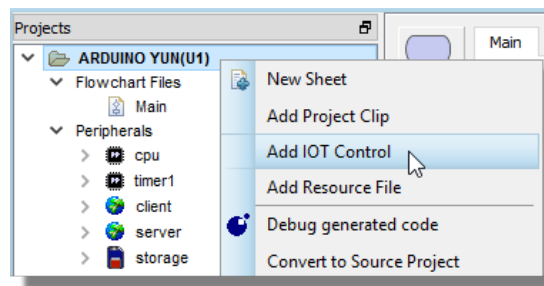
## DIALS AND SLIDERS

### Introduction

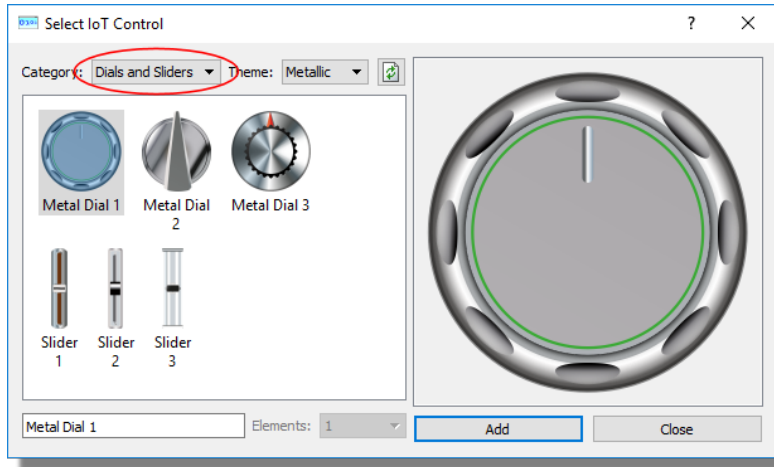
The IoT control library contains a library of both rotary and slider controls. This topic discusses the configuration and runtime properties of these controls.

### To add a dial/slider

1) Right click on the Arduino Yun in the project tree and select the Add IoT Control command from the resulting context menu



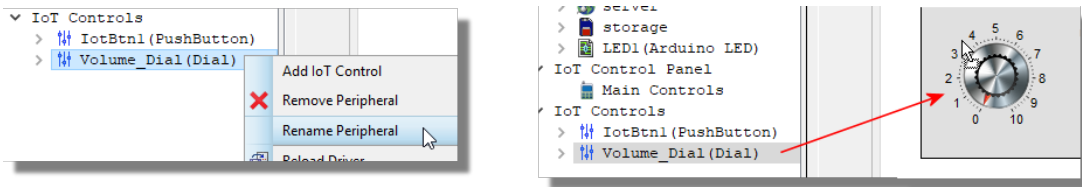
2) Make sure the category selector shows dials/sliders and choose your theme from the selector on the right.



The theme determines the available styles but not their functionality and lets you create a modern looking GUI or a classic retro panel according to your preference.

## Design Time Configuration

Configuration of the controls can be done primarily at design time once they have been placed on the panel. We'd suggest you first rename your control to suit its purpose, then double click on the Mains Control panel to show and finally drag and drop onto the panel.



*Renaming and placing a dial*

Select by left clicking on it and then view the properties in the property pane at the bottom left of your editor.

Properties - Volume_Dial	
▼ Geometry	
x	-197.00
y	-125.36
width	155.00
height	185.52
▼ Configuration	
init	0.00
min	0.00
max	10.00
numDivs	10
numSubDivs	5
tickLength	5

### Dial Properties

**i** If you want a text label on a dial or slider to give it a name, this must be done by adding a text element to your front panel.

The geometric properties are available regardless of which type of dial or slider you choose:

#### Geometry Properties (x, y, width, height)

Properties - LED_ON_OFF	
▼ Geometry	
x	-240.00
y	-100.00
width	101.00
height	101.00

These properties enable positioning and alignment. For example you could set a series of controls to have the same x value to align them vertically, The width and height properties allow for sizing of the control, although you may find it just as easy to use the 'restore aspect ratio' command on the right click context menu for the control after sizing with the mouse.

#### Dial Controls

**init** - sets the initial values of the dial

**min, max** - sets the range of numerical values for the dial

**numDivs, numSubDivs** - sets the number of intervals along the range of the dial.

**tickLength, subTickLength** - sets the size of the individual ticks

**amin, amax** - sets the distance, in degrees, how far round the dial the labels are placed. Top of the dial is at 0 degrees, with the positive being a clockwise direction

**showTicks, showLabels** - toggle the visibility of the ticks and labels around the edge of the dial

**snapToTicks** - Snaps the dial to the closest sub tick or full tick to give precise control of the movement. This can be set to on or off, and is On by default.

**fadeOut** - sets whether the read out of the dial position stays constantly visible or fades out after a short display following a change in dial position

**tickColour, valueFont, labelFont** - customise the colour, font and size of the labels and ticks on the dial

### Slider Controls

Mainly the same as the dial controls with a couple of extra positional options:

**orientation** - allows to flip between horizontal and vertical. Recommended over a rotate transform as labels and values will rotate and reposition accordingly.

**valuePosition, labelPosition** - determine where the values and label will be positioned relative to the position/orientation of the slider.

## **Programming Methods**

---

The programming methods are the primary means by which you can interact with the control in your firmware program. For dials and sliders we have two exposed methods in Visual Designer.

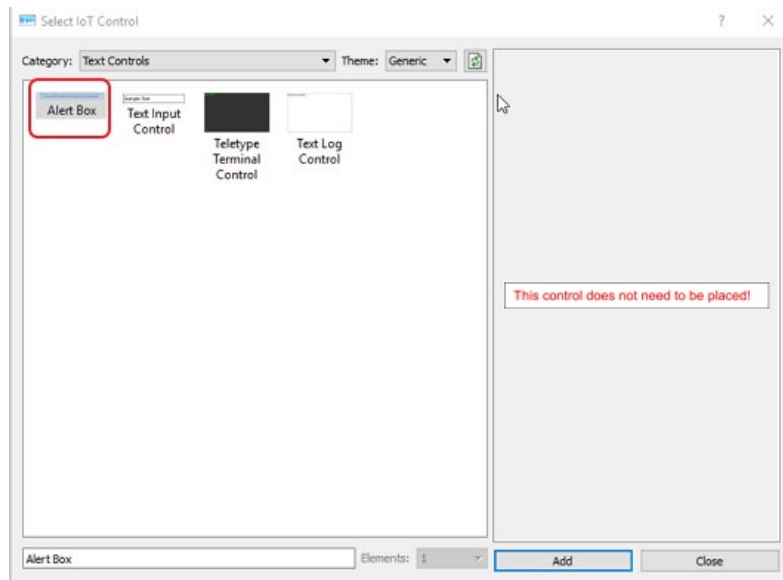
**setValue** - allows you to set the value for the dial or slider via the program rather than manually

**getValue** - reads the value of the dial or slider at its current position.

## ALERTS

### Using Alert Boxes


An alert control is one which can present a message box to the user with information and a request to confirm. Unlike all other control it does not need to be placed on the front panel itself but it does still need to be picked into the project. As always this is done from the add IoT control command on the Project menu or via the right click context menu in the project tree.

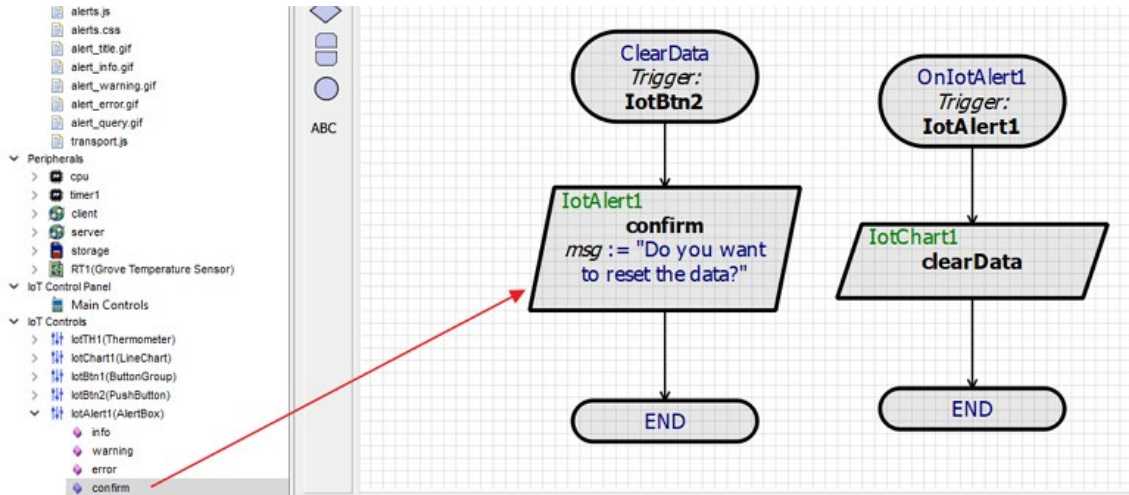


*Adding an Alert box.*

The basic procedure for using an alert control is:

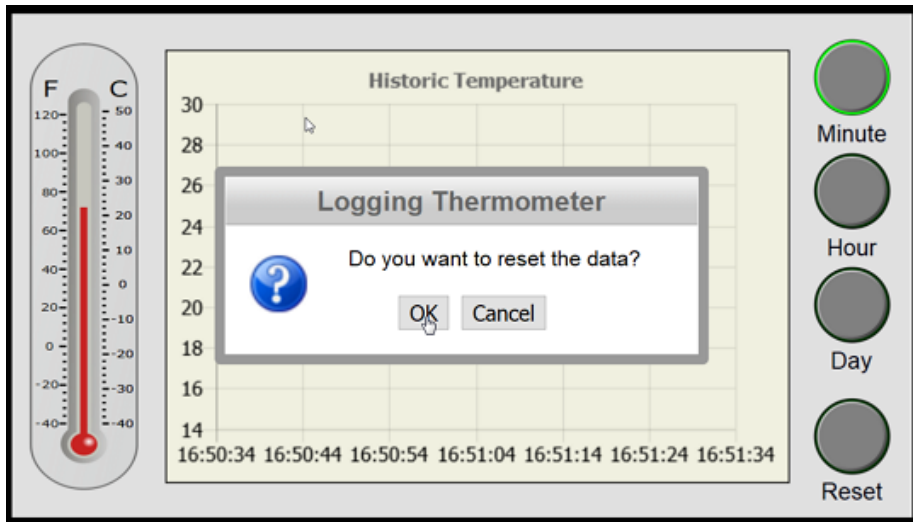
- 1) Add it to your project as shown above.
- 2) Drag the method for the type of alert you want into the appropriate place in your code and type your message.
- 3) Drag out an event, set the trigger to be the alert and then respond to the users mouse click inside the event.

 You'll find more information on events in the IoT Builder help file and event triggers are also discussed in some detail in the [Clocks and Timers](#) topic.



Using a confirm message box to present info to the user and then clearing the chart data when the user confirms.

The code above will present a message box to the user **in the remote user interface** (phone/tablet/browser) when executed as shown below:



**i** You'll find a good example of this in the temperature logging sample design which you can launch from the Open Sample button on the Proteus homepage.

## Design Time Properties and Programming Methods

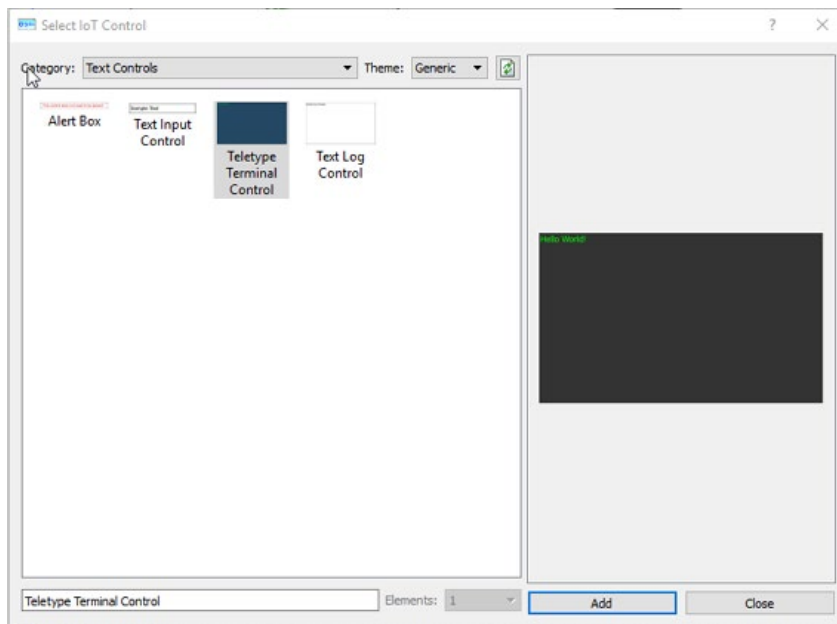
There are no design time configuration properties for an alert box and the four types of alert boxes are self-explanatory. It's important to be aware that program execution continues while an alert box is up and that your trigger event is only called when a positive user input is detected.

## TEXT BOXES AND TERMINALS

### Introduction

IoT Builder includes support for various text entry controls. The simplest is a single line text box but you can also have a multiline teletype terminal or a text logger. Regardless of your choice you add the control in the usual way, namely:

- 1) Select Add IoT control from the Project Menu (or the right click context menu on the project tree).
- 2) Select the text controls category from the resulting context menu.
- 3) Choose your control and click add to insert into the project.





## Text Input Control

---

### *Design Time Properties*

**Geometry:** - Standard geometric properties for sizing and positioning.

**readOnly:** False allows entered text to be fed back in to the program as a string. True prevents text being entered, good for a notification box.

**placeholder** - text to display in the box to prompt the user on the desired input.

**maxLength** - maximum length of the string allowed to be entered.

### *Programming Methods*

**setText:** allows the text to be entered into the box which the program can either accept or wait for the user to edit.

**getText:** reads the string value of the text entered in to the box

**clear:** removes any entered text from the display, returning it to the place holder value

**setError:** set an error message in red under the textbox - ideal if a value entered does not meet the required format/length etc of the program

## Teletype Terminal Control

---

### *Design Time Properties*

**Geometry:** Standard geometric properties for sizing and positioning.

**textFont:** Sets the font family/size/colour etc for the text printed on the display

**placeholder:** text to text to display in the box to prompt the user on the desired input.

**maxLength:** maximum length of the string allowed to be entered.

**clsCmd:** instruction required to clear the screen of any previous text.

### *Programming Methods*

**print:** Print the specified value to the screen. Repeated calls will start printing where the last string finished (i.e. on the same line where possible)

**println:** Print the required value to the screen on a new line

**setBase:** Set the unit base for printing interger vlaues (Decimal, Binary, Octal,Hex)

**setPlaces:** Set how many decimal places are allowed when printing our floating point values

**setPrompt:** Allows you to set a string as a prompt for text entry - this overwrites the place holder.

**getCommand:** returns the value in the command variable, or an empty string if nothing is set.

**cls:** Clears the terminal screen

---

## Text Log Control

---

### *Design Time Properties*

**textFont:** set the font to be used when writing to the display

**warnColour:** sets the colour for any warning text written

**errorColour:** sets the colour for any error text written

**backgroundColour:** sets the background colour of the display

### *Programming Methods*

**setFile:** set the file name for the data to be stored to

**setBase:** set the unit base for printing interger vlaues (Decimal, Binary, Octal, Hex)

**setPlaces:** set how many decimal places are allowed when printing our floating point values

**info:** write a normal log message to the screen

**warning:** Set an warning text to be displayed on the screen

**error:** Set an error text to be displayed on the screen

**clear:** Clears the screen

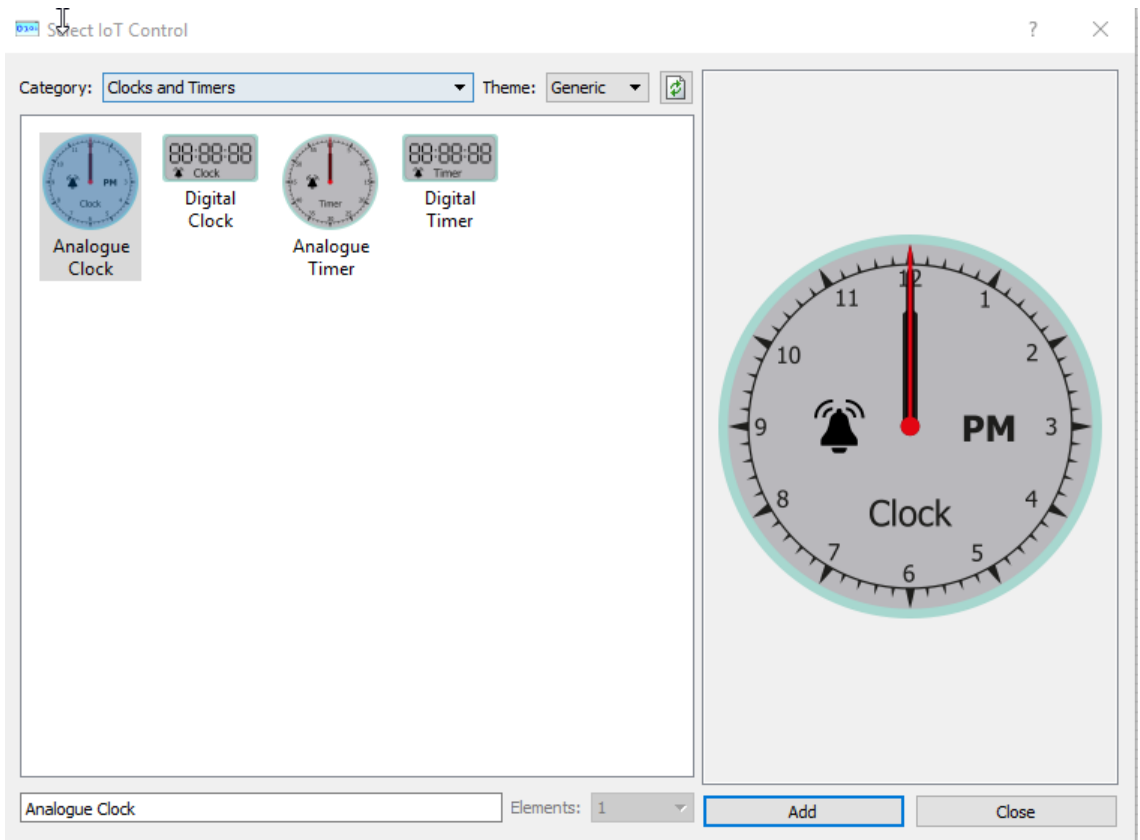


# ADVANCED CONTROLS

## CLOCKS AND TIMERS

### Introduction

There are a range of clocks and timer peripherals that can be added to your front panel to allow you to work with time in your application. As always, these can be picked from the 'Add IoT Control' command on the project menu.



*Adding clocks or timers in the generic skin.*

## Understanding Time

---

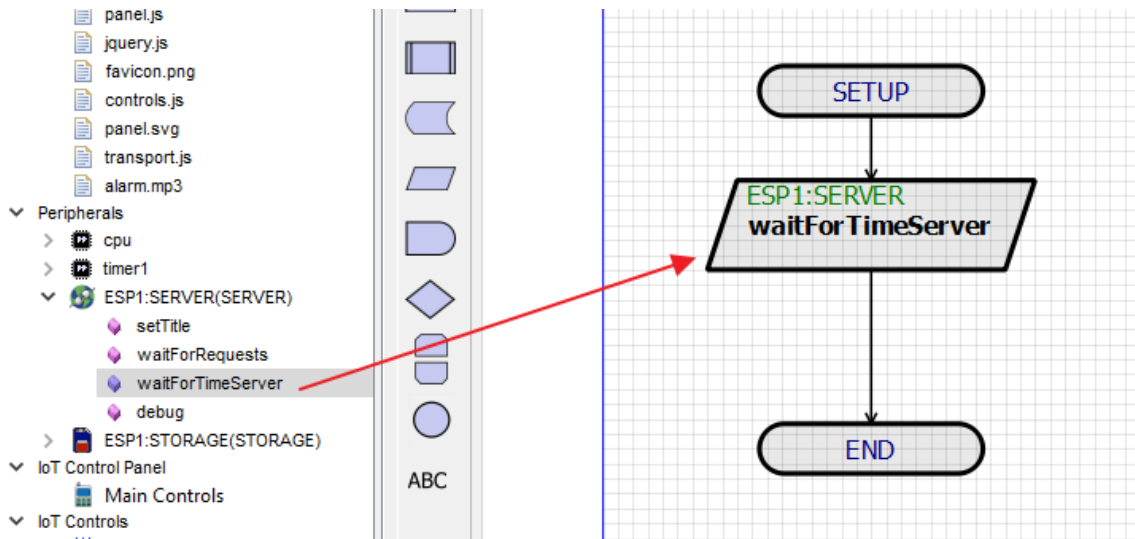
Regardless of whether you are working with a clock or a timer It is really important to understand how your appliance and your GUI acquire time.

### Getting / Setting Time

There are two methods to set the time.

#### 1) Linux Time server

You call the **waitForTimeServer()** method from the server methods in your code - normally first thing in the setup() routine. This will cause the AVR to query the Atheros chip (running Linux) and will return the time.



You then use the **now()** method in the expression editor to assign the current time to the clock or timer. This is shown below.

The screenshot shows the IoT Builder interface. On the left is a project tree with folders like 'Peripherals', 'ESP1:SERVER(SERVER)', and 'IoT Controls'. A red arrow points from the 'setTime' method in the 'IoT Controls' folder to a flowchart. The flowchart starts with a 'SETUP' block, followed by an 'ESP1:SERVER' block containing 'waitForTimeServer', then an 'IoTClock1' block containing 'setTime' and 'time := now()', and finally an 'END' block. To the right, the 'Edit I/O Block' window is open, showing 'Peripheral: IoTClock1' and 'Method: setTime'. The 'Arguments' section has 'Time: now0'. The 'Functions' list on the right includes 'now', which is circled in orange.

Note in this method your appliance itself is aware of the time.

## 2) Browser time

The alternative is to automatically retrieve the time from the browser which will then apply the time to the GUI control. This is done at design time via the property panel.

The screenshot shows the 'Properties - IoTClock1' panel. Under the 'Configuration' section, the 'useBrowserTi...' property is set to 'false' and is circled in red.

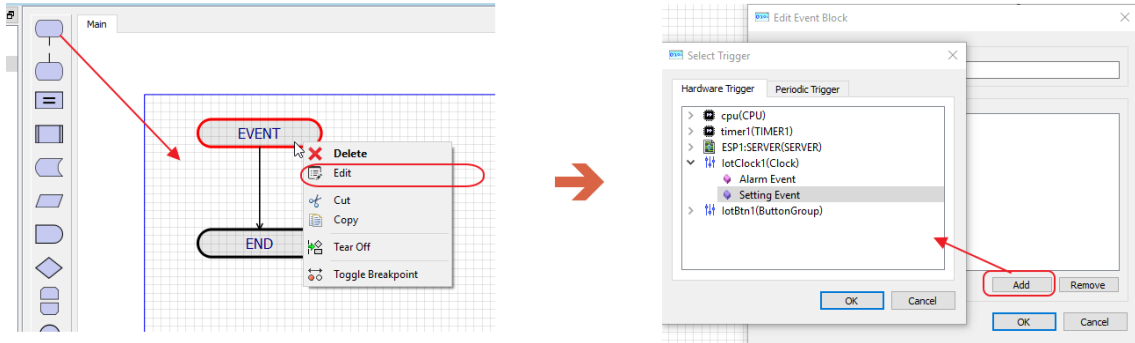
Properties - IoTClock1	
▼ Geometry	
x	-300.00
y	-120.00
width	426.03
height	239.81
▼ Configuration	
label	Clock
> labelFont	
> segColour	■ [0, 0, 0] (255)
useBrowserTi...	false

⚠ If you use the browser time your appliance doesn't necessarily know the time. The first method is safer because your hardware is directly driving the GUI.

## Time Events

Each time the time is set you get a set event which you can use to perform action in your program. Amongst other things this would allow you to pass the time to your appliance if you use the browser time method of acquiring time (although note your appliance will still have no idea of time until the browser is live or the app is opened !!!)

You can respond to a set event by dragging out the event handler and then adding the trigger as the time set.



You'll notice that the other event trigger that you can set is an alarm trigger. If you set an event with an alarm trigger it will be called every time an alarm goes off.

**i** You can see all of this in action in the sample design for the alarm clock which you'll find by clicking on the open sample button on the Proteus home page and then typing 'alarm' in the search box. The sample you want is the 'IoT Alarm Clock'

## Clock Properties

---

### Design Time Properties

**Geometry** - Set position and size of control

**label** - Title to display on clock face

**labelFont** - Set font colour, style etc

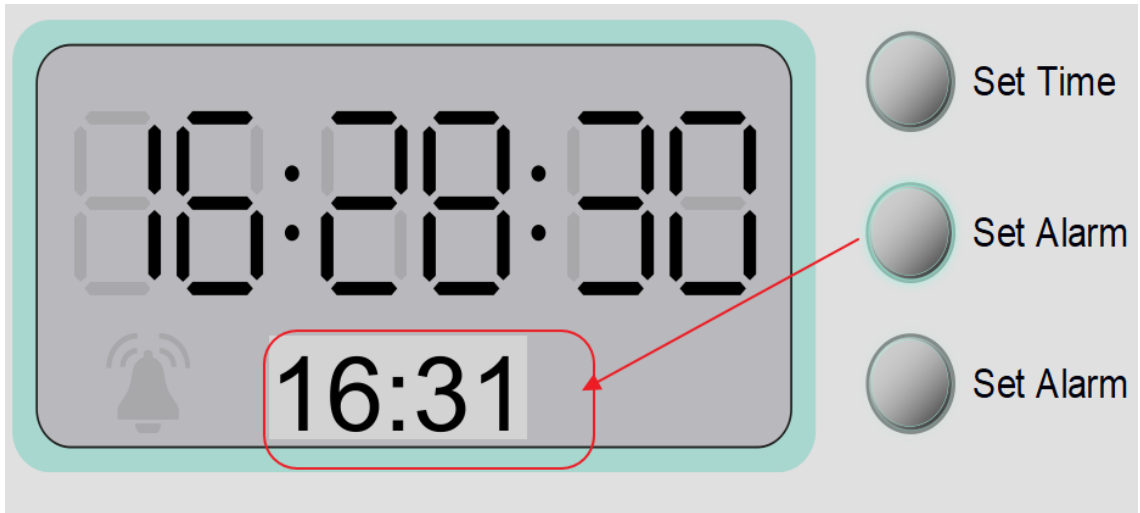
**useBrowserTime** - true/false - user browser time instead of time server for display See explanation of different methods to acquire time detailed above.

**segColour (digital clocks only)** - set digit segment colours

**showPM (analog clocks only)** - true/false - show or hide the PM indicator

### Programming Methods

**setTimeMode** - allows the user to enter a time on the clock face itself. If passed with parameter 0 the user is setting the clock time and if passed with parameter value 1 through 8 the user is setting an alarm time. So, setTimeMode(0) sets the clock time, setTimeMode(1) sets the time for alarm 1, setTimeMode(2) sets the time for alarm 2 and so on. This can be seen in the alarm clock sample design.



*The setTimeMode() method is called when the button is pressed to enable the user to type in a time which will either be an alarm or the clock time itself.*

**cancelMode** - cancel the user entry of the time as entered with the setTimeMode method.

**setTime** - Set the time of the clock to a specified value (typically now()) for real time). read in seconds from midnight 1st January 1970.

**getTime** - reads the current time of the clock

**setAlarm** - set the time, in hours and minutes, for each of the alarms

**enableAlarm** - turns alarm functions on or off for each of the clocks alarms

**getAlarm** - reads the alarm time

## Timer Properties

[Please refer to the sample design 'Countdown Timer' for a working example. You'll find this via the Open Sample button on the Proteus Home Page.](#)

### Design Time Properties

**Geometry:** The usual geometry properties enable accurate positioning and sizing for the control.

**label:** Specify the label for the control that indicates it's function.

**dir:** Specify the direction of counting for the timer. This defaults to down so a ten second period would result in the timer starting at 10 and counting down to 0. Direction set to up would be the opposite and would start at 0 and count up to 10.

**mode:** Oneshot mode represents a single timer (unless you stop->restart in code) whereas repeat mode gives you a way to receive periodic callbacks to the timer trigger event in your code.



**period:** The three period controls let you set the number of hours, minutes and seconds for the timer period. You can also do this in code if you need to change or set at run time.

**enableAlarm:** this provides a way to disable the alarm when the timer condition is met. Can also be set/unset in code to handle programmatically.

**segColour:** set the colour of the segments in the display.

### *Programming Methods*

**start()** : This starts the timer count.

**stop()**: The stops the timer count

**restart()**: This will restart the timer count.

**setPeriod()**: set the length of time to elapse before the alarm goes off.

**enableAlarm()**: turn the alarm on or off. If enableAlarm is set to FALSE then the alarm will not trigger and you will not receive an alarm trigger event call. By default, the timer alarm is enabled.

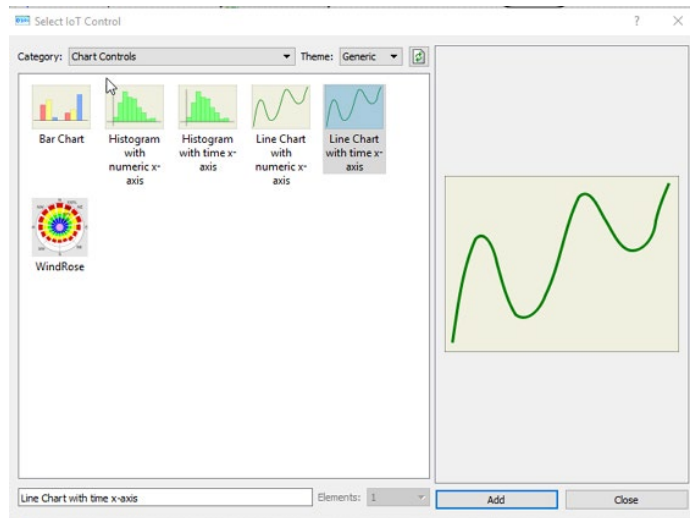
**getTime()**: Gets the time from the clock.

## LINE CHART

### Introduction

The line chart is a control which allows you to log and store data. You can select either a numerical or a time based line chart in the usual way:

- 1) Select Add IoT control from the Project Menu (or the right click context menu on the project tree).
- 2) Select the chart controls category from the resulting context menu.
- 3) Choose your control and click add to insert into the project.



If you choose to have the x-axis as time it is essential that you use the `waitForTimeServer()` server method in your setup routine so that your appliance actually knows the time. This is discussed in more detail in the topic on [clocks and timers](#).

**i** You'll also find a good working example of the line chart in the temperature logger sample design. You can open this from the Open Sample button on the Proteus home page. In particular, note how the x-axis (time) can be adjusted in code and the chart view updates dynamically.

### Design Time Properties

**titleText:** This property simply lets you set the title for the line chart.

**FontSizes:** This simply lets you change the size of the fonts on the chart control.

**showLegend:** This will show or hide the colour key. Without this it becomes hard to tell the difference between lines however it will also take up less space.

**pointRadius:** This simply allows you to change the size of the different points on the chart. Setting this to 0 will remove the points entirely just leaving the line.

**lineWidth:** This allows you to change the thickness of the line. Setting this to 0 will hide the line and leave only the points on the chart.

**backgroundColour:** This will allow you to set the background colour for the selected chart.

**timeRange:** This property will set the x axis allowing you to display time. This is only available on the time-based line chart. The unit will set what the chart will display in and the range will set how much is shown. If absolute is true the histogram will work off the current time, so if you start it a 1:00 it will also begin at 1:00. If Absolute is false, then it will start at time 00:00 and will increment from there.

**Column settings:** numColumns simply sets the number of columns available. Then you can go to the individual column settings and change their name, which side the axis is displayed on, colour and range on the y axis.

## Programming Methods

---

**setXRange:** This allows the user to set the x range, this only works on the numeric line chart control however is available on both.

**setTimeRange:** Using this you can set the x axis to a time. Unit will determine which time unit is used, then range will determine how many of those units are displayed. If absolute is true the histogram will work off the current time, so if you start it a 1:00 it will also begin at 1:00. If Absolute is false, then it will start at time 00:00 and will increment from there.

**setYRange:** This will allow you to set the y range on either side of the chart. However if the value provided goes over the designated max or below the designated min then the chart will automatically adjust.

**showColumn:** This will allow you to show or hide lines. You cannot show a line which isn't set within the properties.

**setDataFile:** setDataFile allows you to set which file the graph will read and write to. This is set by entering the file name as a string.

**setPlaces:** This will set the number of decimal places the histogram will work too.

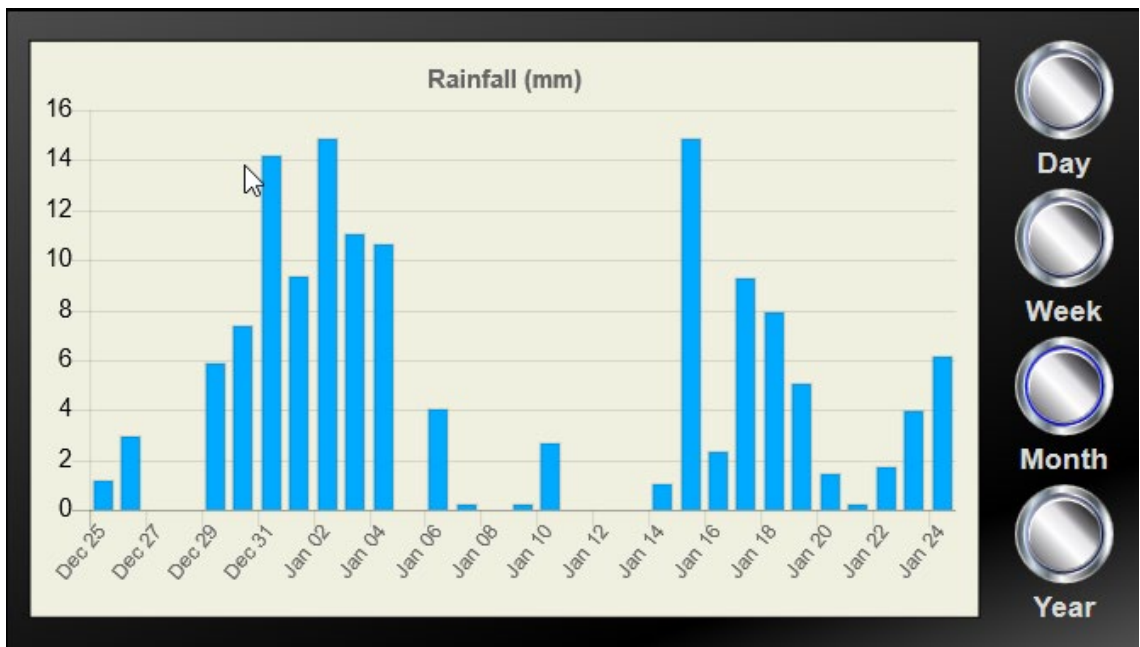
**writeData:** Write data will allow you to write a data point to the chart, this will not only update the chart but also store that data in a file if you have linked it.

**clearData:** This will clear all the data from the chart and delete the data file linked to it.

## BAR CHART

### Introduction

The bar chart is a data collection and storage control allowing you to set multiple pieces of data and then store them inside multiple files. Files can then be loaded in and displayed next to one another allowing for the comparison of the data. The data files cannot be accessed directly and you must use the read and write commands to interact. You can easily control the colour and labels to make the bar chart stand out.



*Example of a single series barchart*

## Design Time Configuration

### numCategories

This Property allows you to set the number of categories. A category is a collection of data for one subject. Categories don't interact unless told to and each Category will contain data from different times.

### categorySet

This allows you to set the names of the different categories making it easier to see which is which.

### [numSeries](#)

This will set the number of Series. A series is a set of data within a Category. It is important to understand that there is difference in behaviour between series0 and all the other series. series0 is held in memory and can be manipulated programmatically via method calls such as setData(). In order to preserve resources all other series are stored in files and can only be manipulated via the writeSeries() and loadSeries() method calls. So, when you have multiple series in your chart your program will become quite a bit more complicated as you will need to switch data into other series to populate the chart.

### [setSeries](#)

This allows you to set the colour of each series and the labels which correspond.

### [titleText](#)

Simply a string to be used as a title

### [titleFont,scaleFontSize,labelFontSize](#)

Simple properties allowing you to change the size of the text.

### [showLegend](#)

This allows you to show and hide the Series keys.

### [minY and minX](#)

This allows you to set the limits of each axis. The graph will however auto adjust if the values go over.

## **Programming Methods**

---

### [showSeries](#)

This method allows you to show and hide series. This is useful when displaying different amounts of data. You cannot create series here and the number of series must be set in the design time properties before you can reveal them using this method.

### [setYrange](#)

This simply allows you to control the Y-range from your program

### [setData](#)

setData() allows you to set any value to series 0 of a user set category. Note that you can only manipulate series0 with this method. With multiple series you would typically:

- manipulate series0
- writeData() to save series0
- loadData() to load series0 into series1 for example.
- resetSeries() and then manipulate series0 again.

 Only series0 can be manipulated in memory. This preserves resources on the 8-bit AVR.

### [incrementData](#)

This allows you to increase the 0 series on any category by a set value.

[decrementData](#)

The reverse of incrementData will remove a set value from the 0 series of any category.

[resetSeries](#)

This allows you to reset the data on an individual series. It will apply to all categories.

[writeSeries](#)

This will write all the 0 series to a file which you set by inputting a string. This will also overwrite a file if you use the same file name twice allowing you to re-use files.

[loadSeries](#)

This is the next step of writeSeries and allows you to read data from a file and apply it to a series. Most basic example would be you have read data from series 0 and then wiped it. You can then apply that read data to the next series along series 1.

[clearData](#)

clearData will simply clear all the data files and wipe the graphic.

## HISTOGRAM

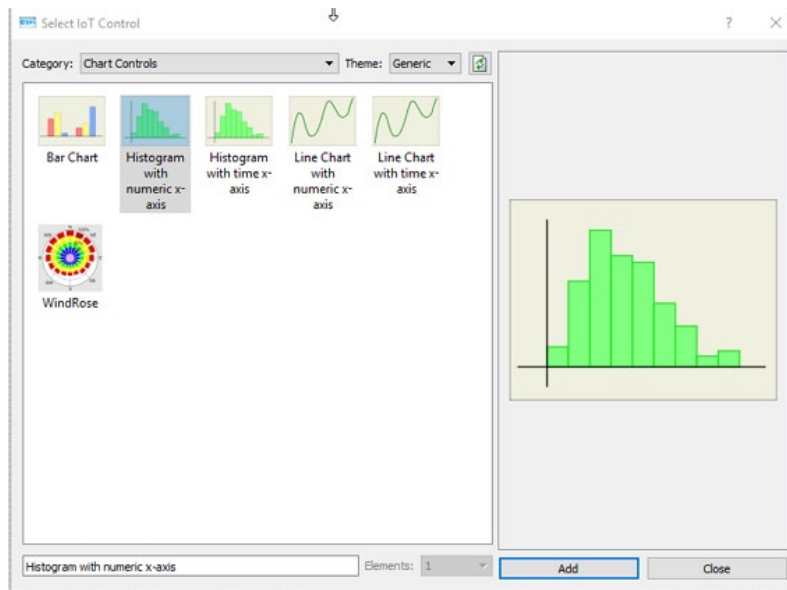
---

### Introduction

The Histogram is a chart control that allows you to display and store data. It may look more like the bar chart but it behaves more like the line chart with the user setting an x value opposed to a category.

### To add a histogram control:

- 1) Select 'Add IoT Control' from the Project Menu.
- 2) Select the Charts Category.
- 3) Choose either the numeric or the time based version of the control.
- 4) Click Add to insert into the project.



*Adding a histogram control.*

The control can then be dragged and dropped from the project tree into the virtual front panel in the normal way.

### Design Time Properties

**Title:** Simply allows you to set the title for the histogram.

**titleFontSize, labelFontSize and scaleFontSize:** Simply allows you to change the font sizes for all the labeled areas.

**Background and Bar Colour:** Allows the user to set the colours of the background and bars.

**X and Y range:** Allows you to set the min and max ranges on the x and y axis, the min can be a negative or 0. The x range property will also allow you to set the number of bars displayed on the histogram.

**timeRange (Time histogram only)** :The time range allows you to set the range of time shown by the histogram. You can also set the number of bars displayed by the histogram. If absolute is true the histogram will work off the current time, so if you start it a 1:00 it will also begin at 1:00. If Absolute is false then it will start at time 00:00 and will increment from there.

### Programming Methods

**setXrange** : setXrange allows you to set the range on the x axis allowing you expand or decrease it and set the number of bars.

**setTimeRange (To be used on time histogram only)** :This allows you to set the range of time displayed on the x axis. This will not only allow you to set the number of bars available but also set the range of time displayed. If absolute is true the histogram will work off the current time, so if you start it a 1:00 it will also begin at 1:00. If Absolute is false then it will start at time 00:00 and will increment from there.

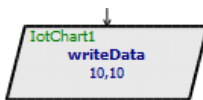
**setYrange** :This allows you to set the range of the y axis. However, if the data is larger than the y axis the histogram will automatically adjust and set to the correct range.

**showColumn**: This allows you to show or hide individual bars on the histogram and therefore filtering your data.

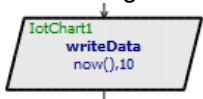
**setDataFile**:setDataFile allows you to set which file the graph will read and write to. This is set by entering the file name as a string.

**setPlaces**:This will set the number of decimal places the histogram will work too.

**writeData**:This will write a line of data to the graph. In addition, if a data file is set it will also write that data to the data file and increment on top of existing data.



When using a time-based histogram use this configuration.



**i** You'll find more information about time methods like `now()` in the topic on [clocks and timers](#)

**clearData**: This will clear all data from the histogram and corresponding data file.

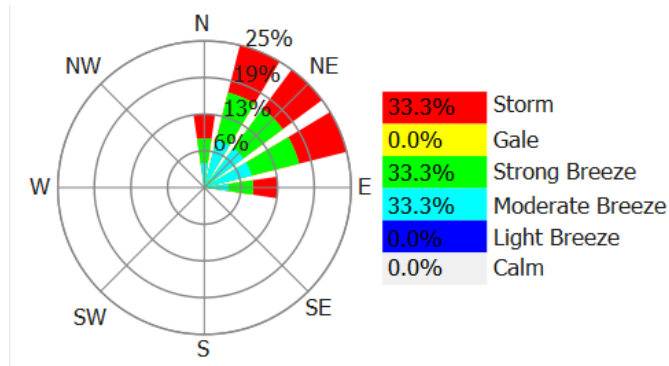


## WIND ROSE CONTROL

---

### Introduction

The wind rose is a specialised chart control which has been designed to show data about wind speed and direction.



📌 Please refer to the weather station sample design for a working example on how to use and configure the wind rose

### Design Time Configuration

#### Calm and Petal/Band Properties

The wind rose uses petals to show the direction and wind speed. Because calm wind has no direction it is shown as a circle in the centre. Each wind speed is given a different colour with the top speed at the far end of the petal and the low wind speed at the centre. You as the user can customise the colour of each petal by editing its “band” and the corresponding key will also change. You can also change the label of each band key allowing you to change it from storm to 20m/s if that’s how your software stores the data.

Due the fact not all black text can be seen on a colour you have the option to change the text colour on each key individually. This property is also in the band property group.

Configuration	
calm	
label	Calm
bandColour	[240, 240, 240] (255)
textColour	■ [0, 0, 0] (255)
band1	
label	Light Breeze
bandColour	■ [0, 0, 255] (255)
textColour	■ [0, 0, 0] (255)
band2	
band3	
band4	
band5	

### Time Properties

The time range property controls how much data is shown by the chart. In this example it will show 24 hours' worth of data. The unit has 4-options minute, hour, day or month which are the unit. Then simply put the number of units you would like to show.

timeRange	
unit	Hour
range	24

### Colour and Font Properties

These properties are simple aesthetics controlling the font and colour of the text and the colour of the background.

▾ backgroundColour	[255, 255, 255] (255)
Red	255
Green	255
Blue	255
Alpha	255
▾ directionFont	
family	sans-serif
size	12
weight	normal
style	normal
decoration	none
▸ colour	■ [29, 29, 27] (255)
▾ percentageFont	
family	sans-serif
size	12
weight	normal
style	normal
decoration	none
▸ colour	■ [29, 29, 27] (255)
▾ keyFont	
family	sans-serif
size	12
weight	normal
style	normal
decoration	none
▸ colour	■ [29, 29, 27] (255)

**Filename Property**

File name will simply create a storage file with that name for your data. You cannot see or access that file directly but by changing it you can be running multiple different files for different wind roses.

filename	windrose.dat
----------	--------------

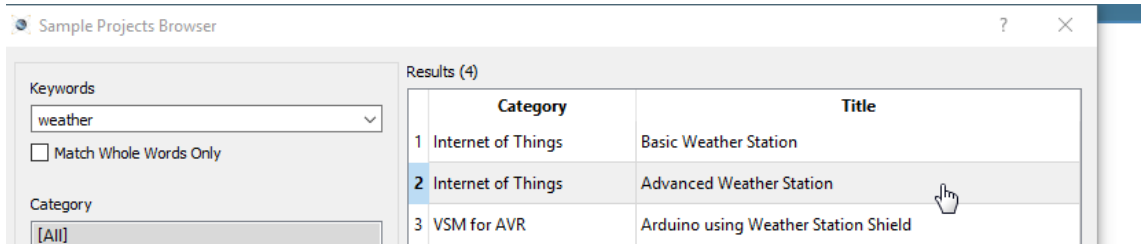
**Bands Property**

Here you can set which band a certain speed will enter. The example here 0-4 mph will be calm, 5-13mph will be band one and so on. This property will always need 5 numbers and cannot operate with any more or less.

bands	4,13,25,39,55
-------	---------------

## Programming Methods

Often the best way to see how something works is to look at an example. The weather station example (accessed via open sample button on home page) makes use of the wind rose and is well worth reviewing before starting your own project.



**SetTimeRange** - The set time range allows you to set the

**StoreReading** - Store reading will store the direction and band into a temporary array. The data will not enter the main data file until storeRecord is used.

**StoreRecord** - storeRecord will transfer all data from the temporary array into the main data file and give it a time stamp.

**clearData** - clearData will clear the entire data file leaving no data to display.

**reload** - Reload simply updates the graphic.