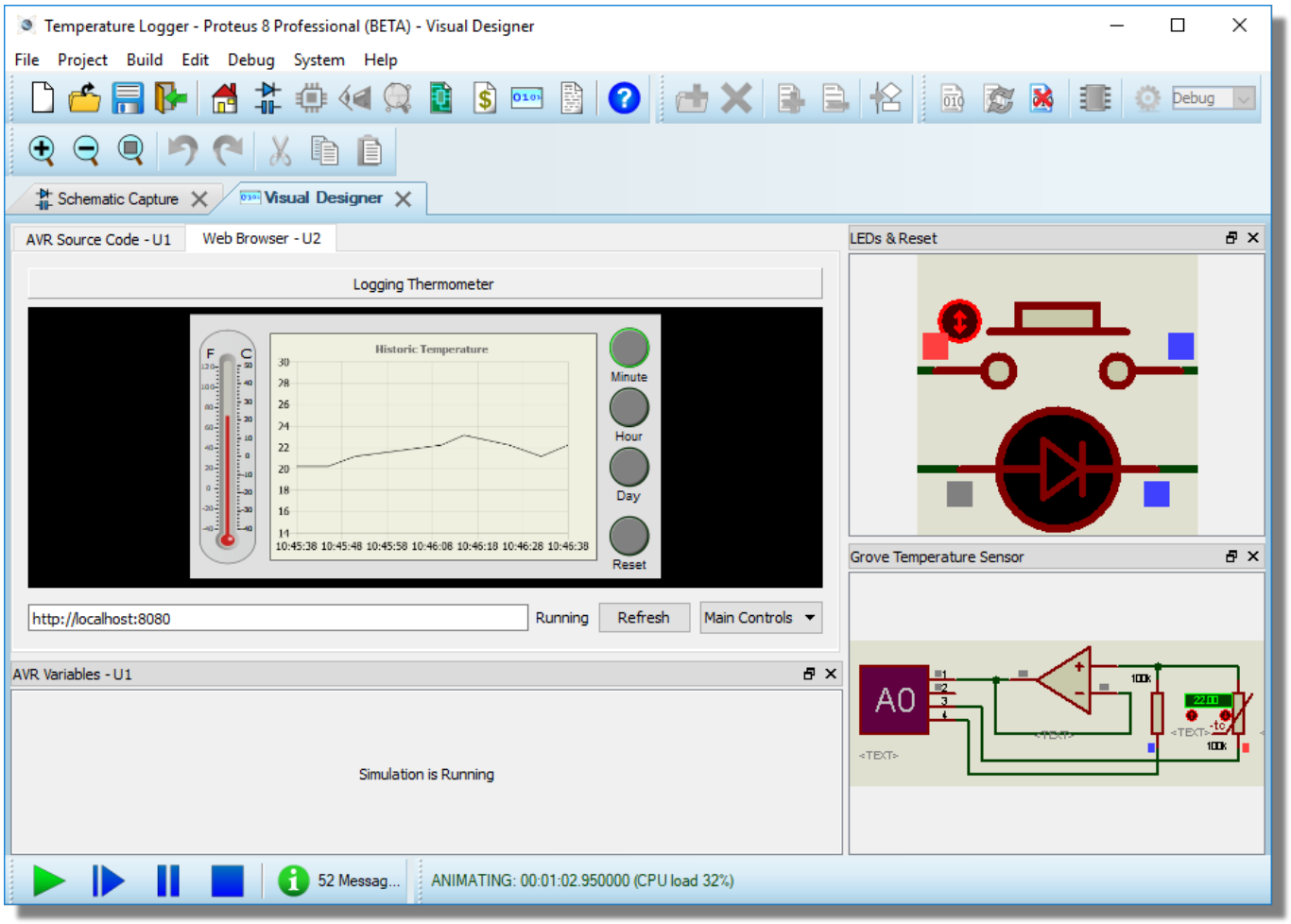
INTRODUCTION

What is IoT Builder ?

IoT Builder 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 various popular Raspberry Pi and Arduino Hardware.

See Also:

- IoT Builder Compatible Hardware



The basic workflow is :

- 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.
- Add the electronic shields, sensors and breakout boards to the schematic via the Visual Designer peripheral gallery.
- 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.
- 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.
- Simulate and debug. Press play and interact with your front panel and your virtual hardware on the schematic. Set breakpoints and single step debug.

- 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.

IoT Builder Targets

While other target platforms are in development IoT Builder is currently targeted at Arduino and Raspberry Pi.

See Also:

- IoT Builder Compatible Hardware
- Tutorial 1 : Blink an Led

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'.

- Source Code Projects

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 Visual Designer is discussed in full in the reference manual linked below:

- Visual Designer

It is also entirely possible to use IoT Builder as an add-on to Proteus VSM for Arduino. This is discussed in the topic on source code projects and the use of the VSM Studio IDE is explored in more depth in that reference manual.

- VSM Studio IDE

COMPATIBLE HARDWARE

Introduction

IoT Builder is designed to work with various pre-made hardware boards. The setup , configuration and programming requirements are dependant on your choice of hardware. Specifically, we support:

- Raspberry Pi 4
- Raspberry Pi 3 / Raspberry Pi Zero
- Arduino Yun / Seeeduino Cloud
- Proteus Playkit
- Arduino Uno plus ESP8266 (in the from of WeMOS D1)

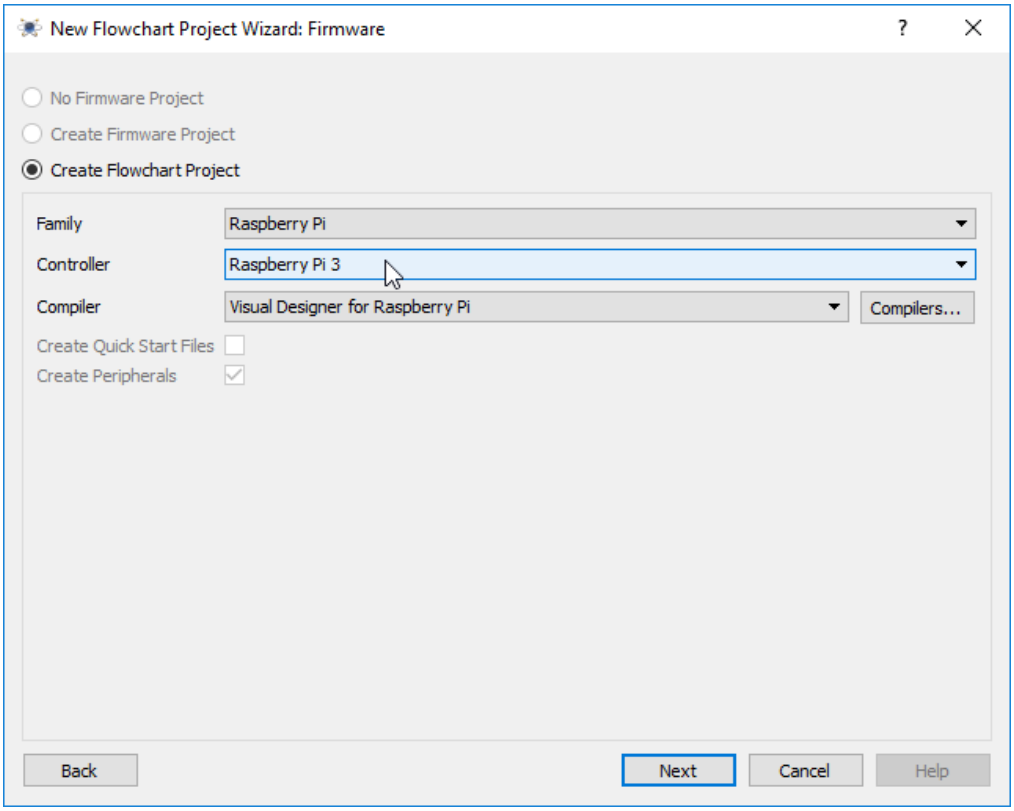
Choosing a Target / Getting Started.

When you create an IoT builder project from the New Project Wizard on the home page you need to specify the target hardware. The process is simple but varies slightly according to target. More information on each target is given below.

Raspberry Pi 3 & Raspberry Pi Zero

Raspberry Pi 3

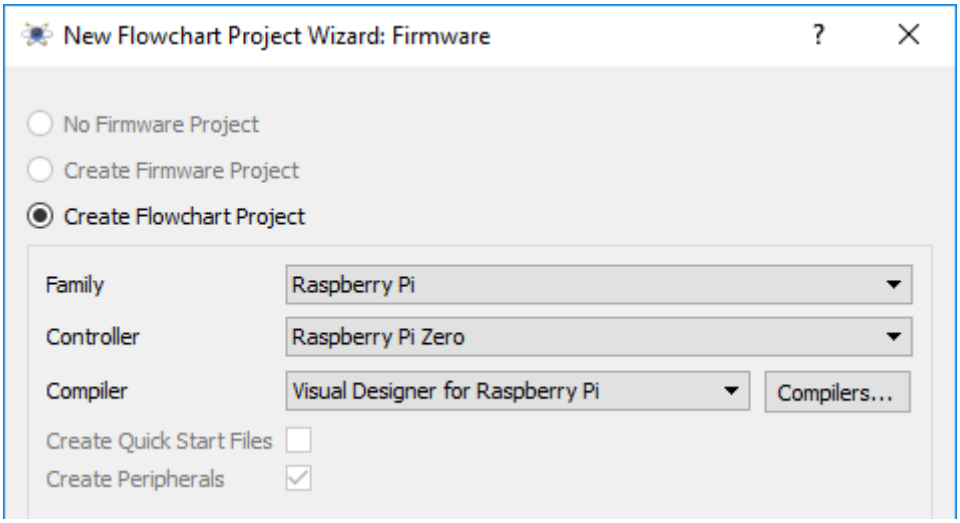
From the homepage select new flowchart project and specify Raspberry Pi 3 as the target.



New Project wizard when creating a Raspberry Pi 3 Project

Raspberry Pi Zero

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

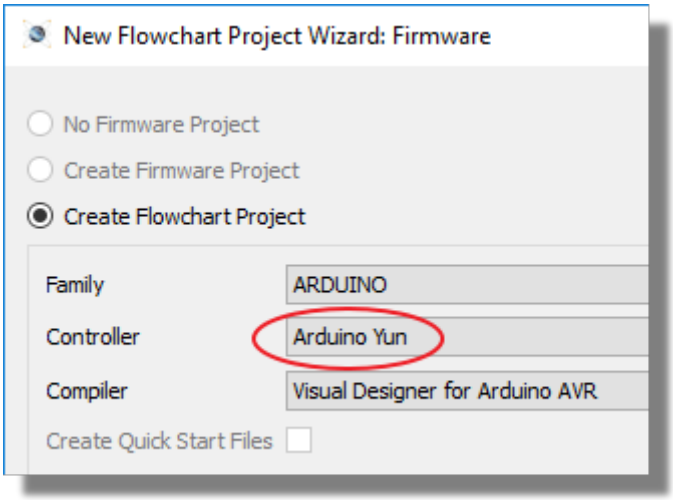


New Project wizard when creating a Raspberry Pi Zero Project

Arduino Yun & Seeeduino Cloud

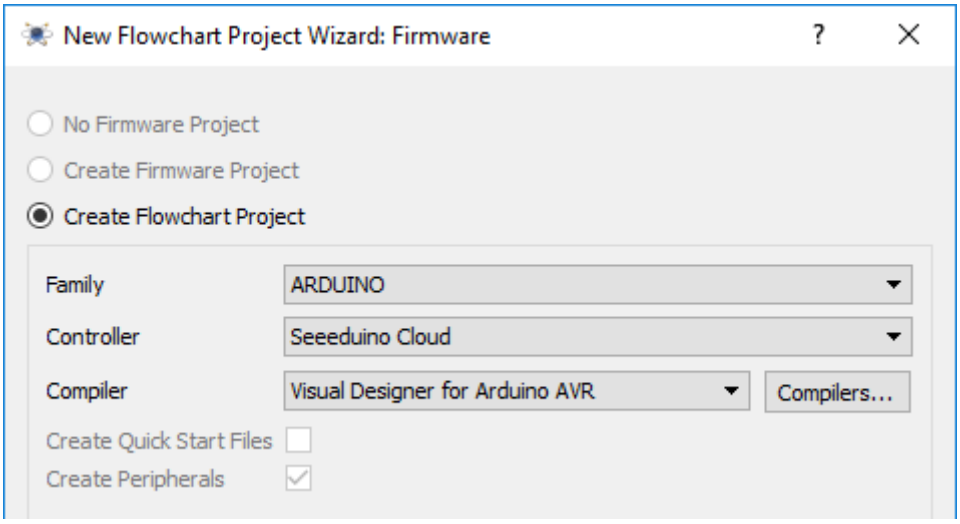
Arduino Yun

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



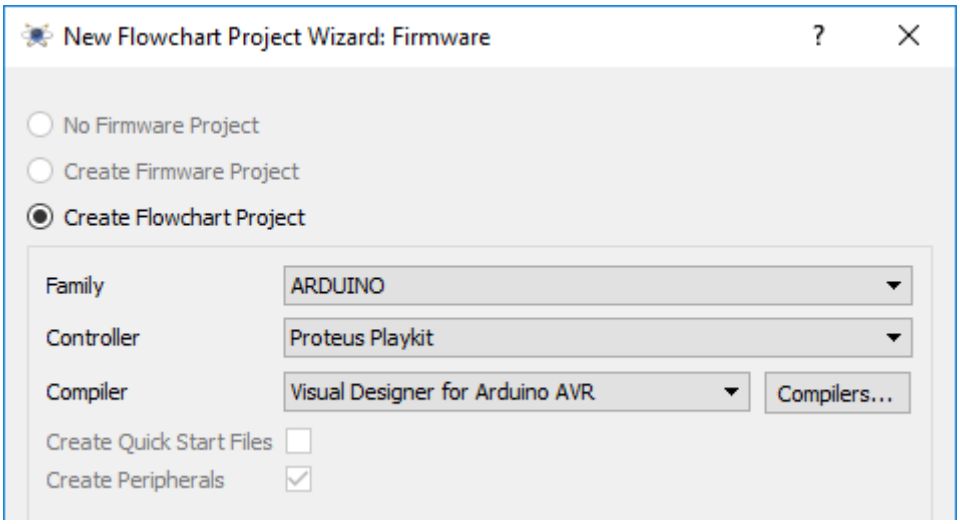
Seeduino Cloud

From the homepage select new flowchart project and specify Seeeduino Cloud as the target.



Proteus Playkit (Ready made ESP8266 + Uno)

From the homepage select new flowchart project and specify Proteus Playkit as the target.

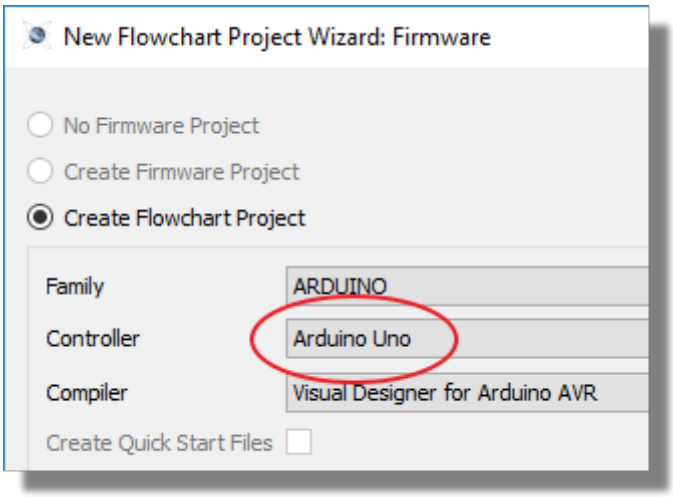


See Also:

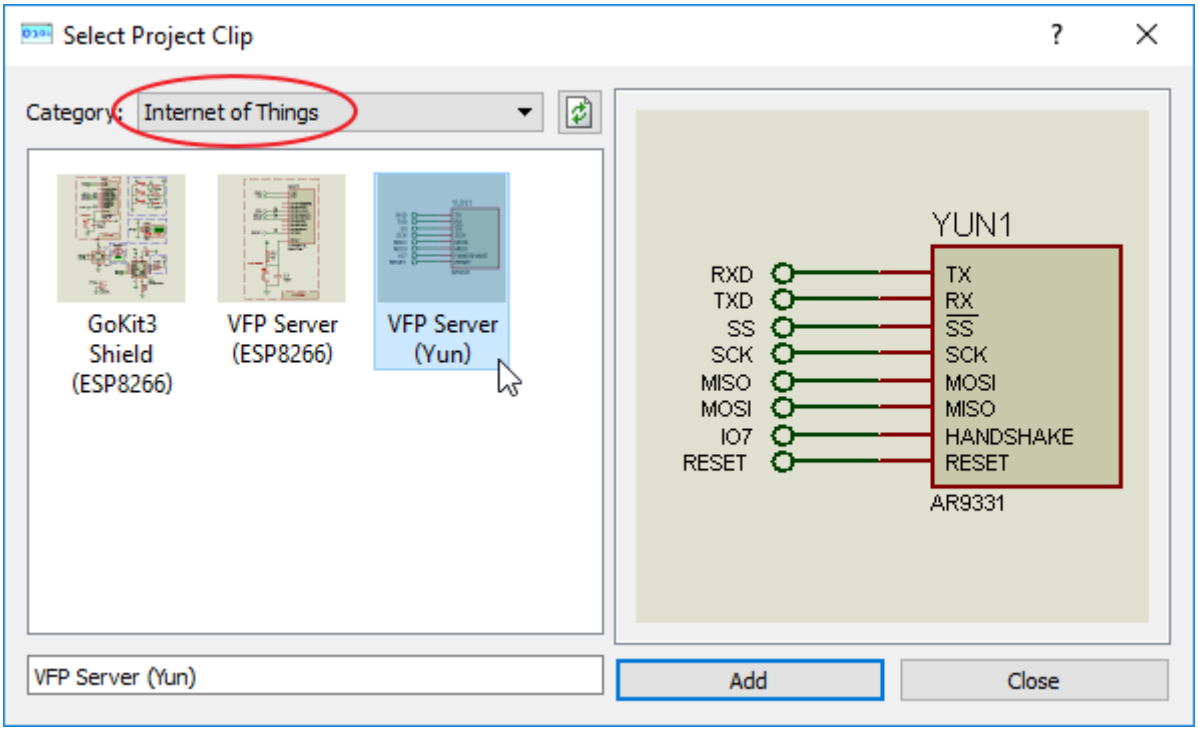
Proteus Playkit

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.

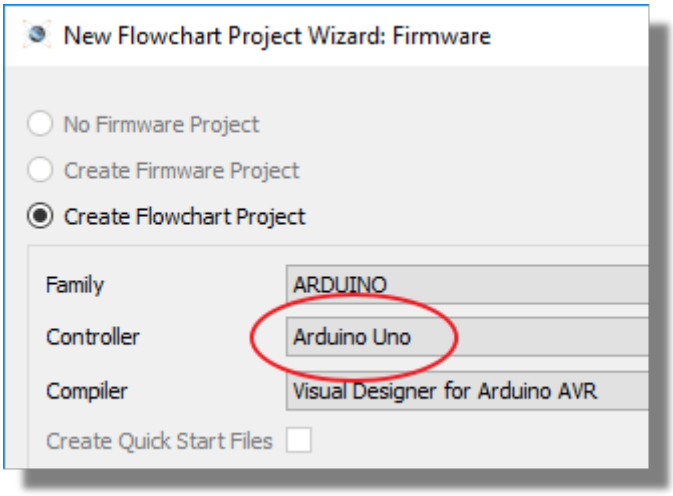


Using an Arduino Yun baseboard is simpler but the Yun shield or ESP8266 tends to have better wifi and allows you to couple to a Mega which has more program ROM.

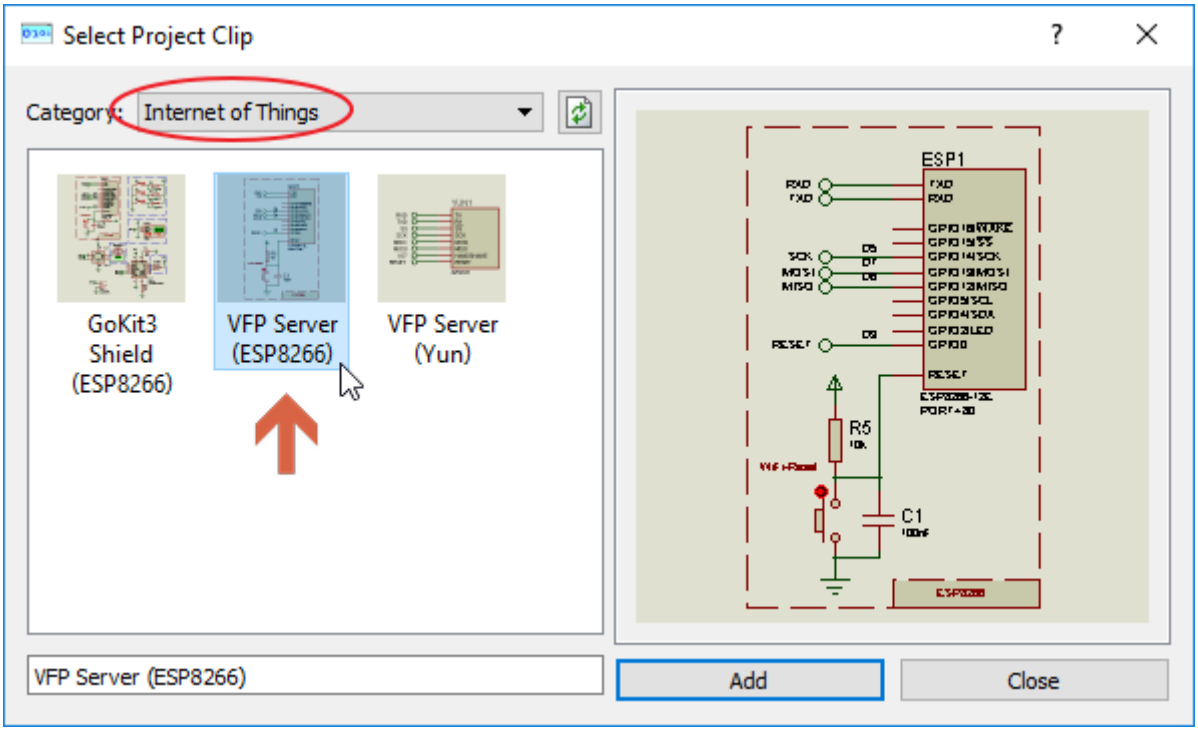
Arduino Uno/Mega + ESP8266

Our support for ESP8266 is based on the Proteus Playkit. You can set up and configure other ESP8266 such as the WeMOS D1 however be advised that the configuration required to make this work is extensive and not for beginners ! Proteus playkit provides an inexpensive ready made alternative.

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 ESP8266



See Also:

- Building and Using an ESP8266 Server with IoT Builder

Programming the Target

You'll find information on configuration and programming the various targets via the link below:

See Also:

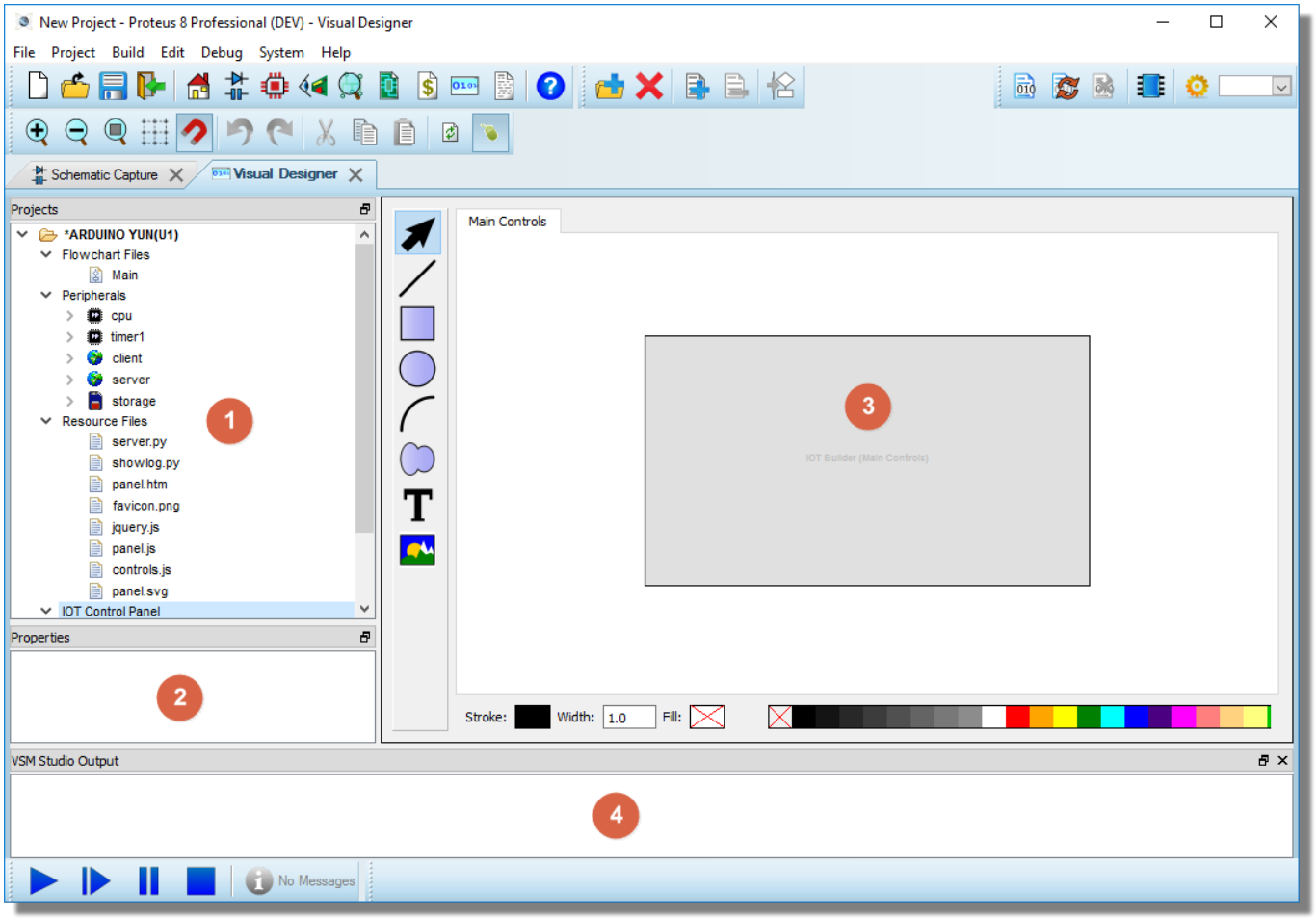
- Programming the Hardware

labcenter Electronics

GUIDED TOUR

Overview

This topic covers in brief the various control sections of the front panel editor in IOTBuilder. 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.

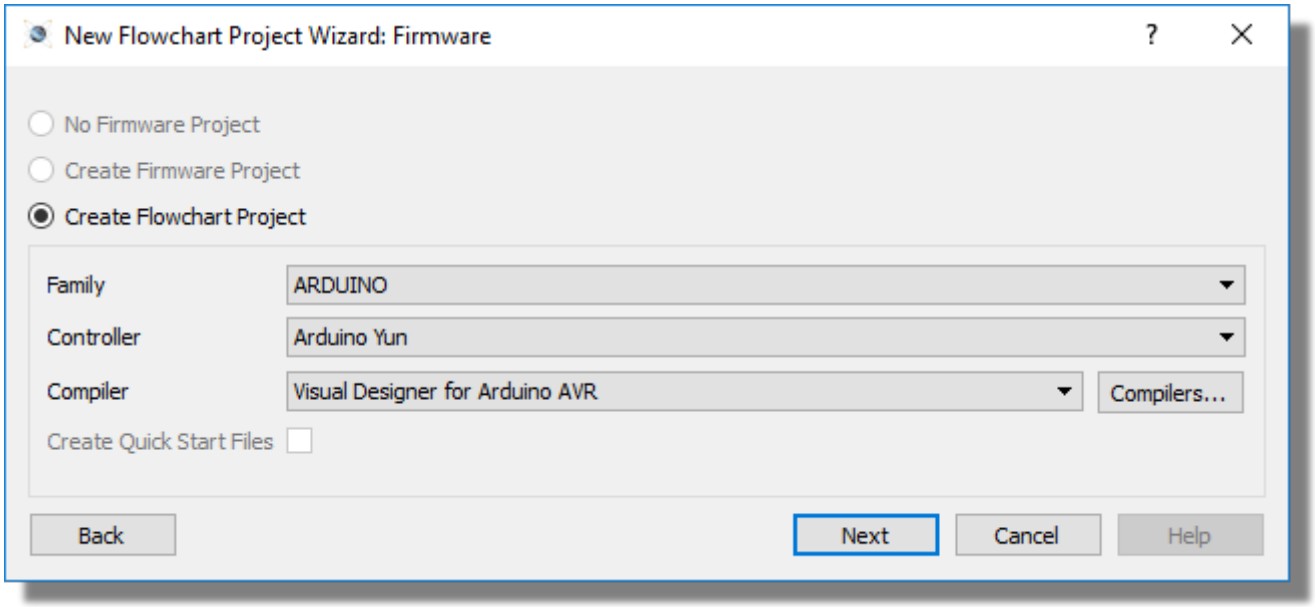
4) Configuration of controls from the property panel at the bottom.

The following short example illustrates these points.

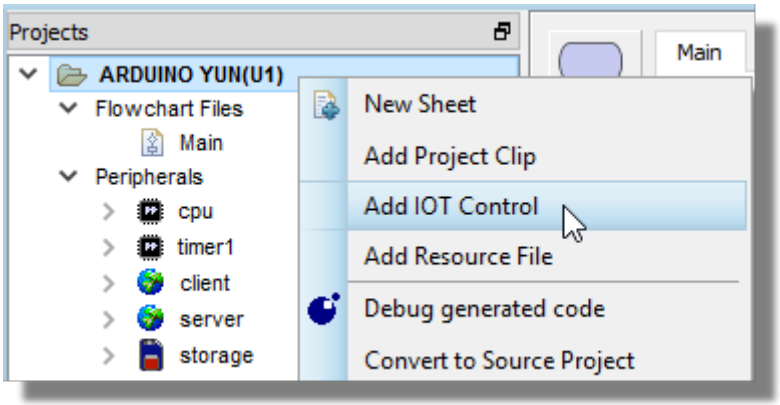
Adding a button

> 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.

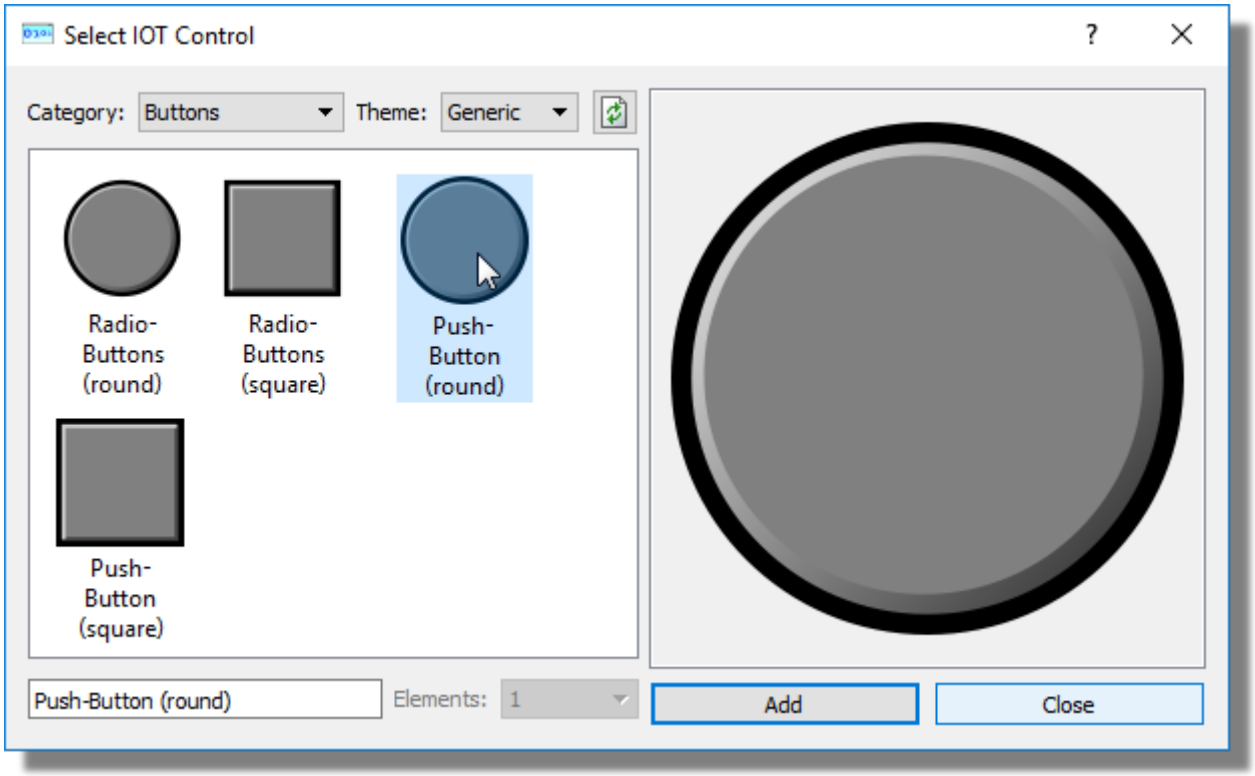
Start a new flowchart project and select Arduino Yun as the target device.

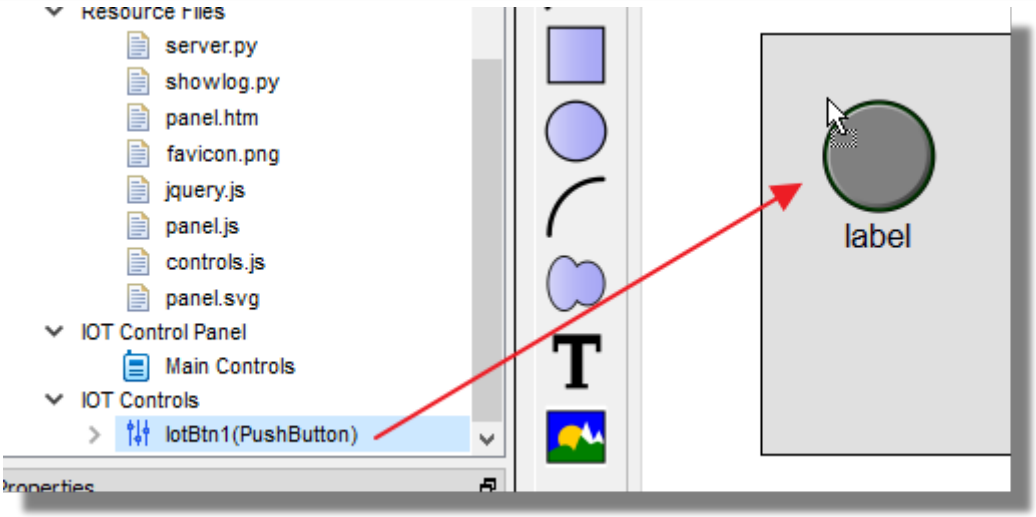


Right click on the Yun in the Project Tree and select 'Add IOT control' from the context menu.

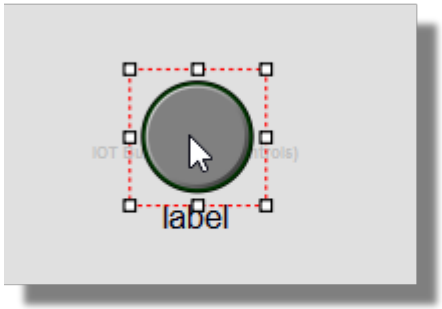


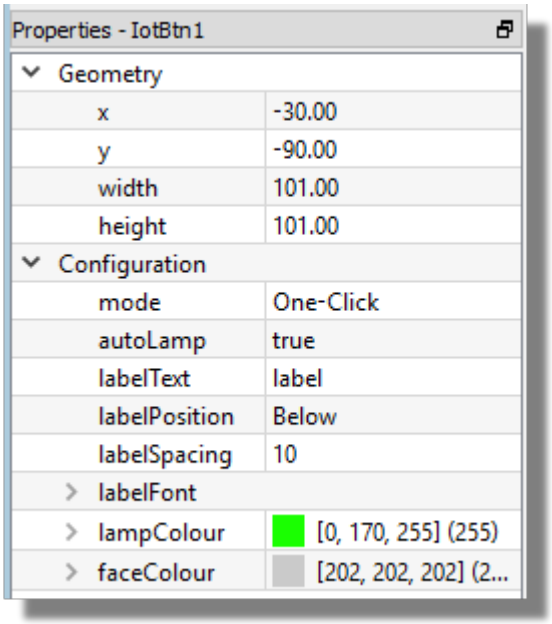
Select a button from the resulting dialogue and then drag and drop it onto the front panel editor.





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

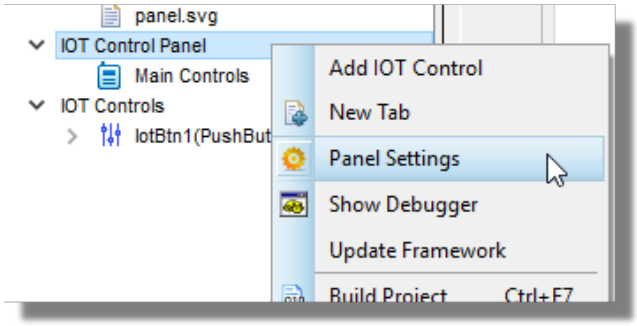


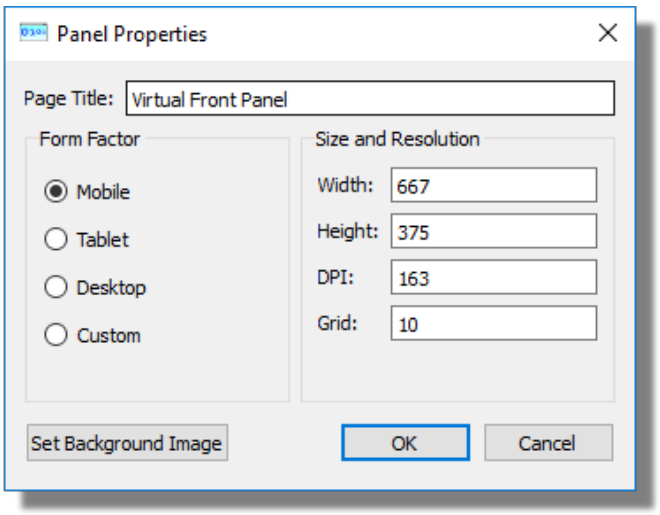


The Editing Window

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:

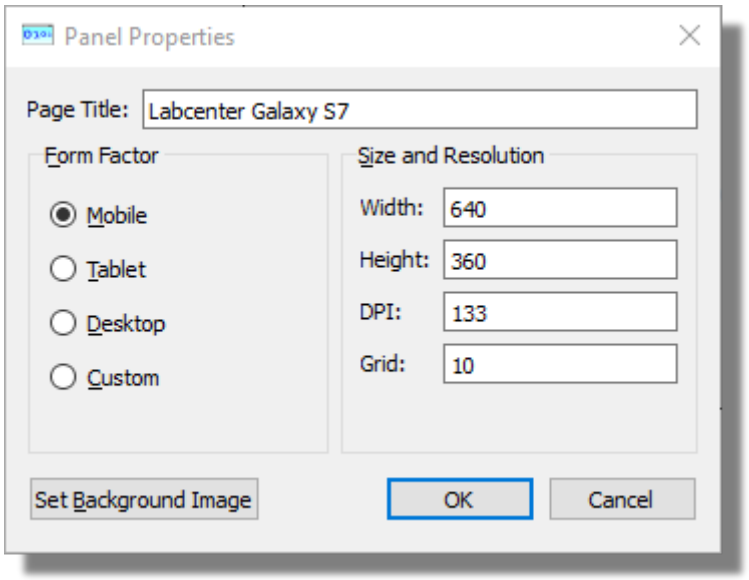
1) Right click on IOT Panel in the Project Tree and select Panel Settings from the resulting context menu.





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 Independant Pixels).



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.

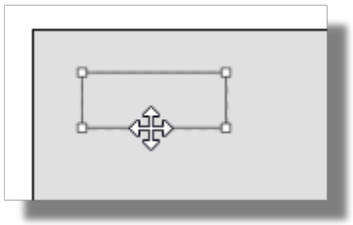
The Graphics Panel

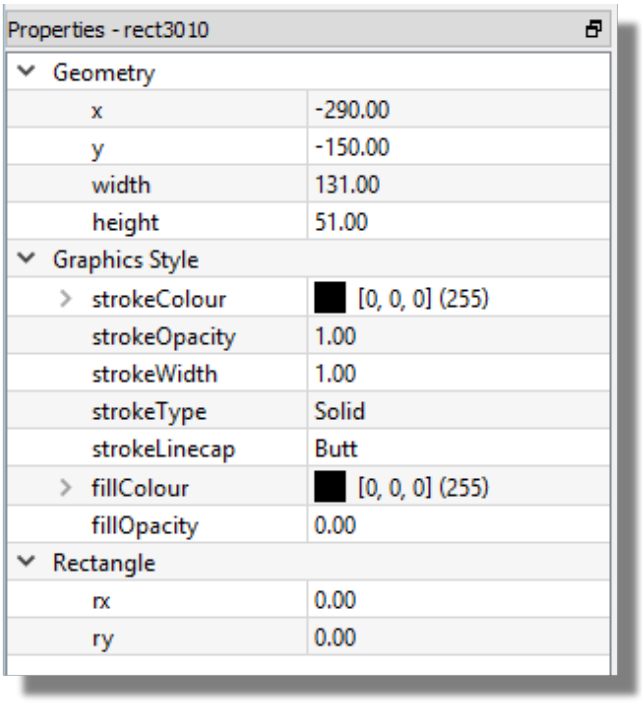
The graphics panel on the left hand side contains a selection mode and then a series of graphics and text primitives as described below:



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.

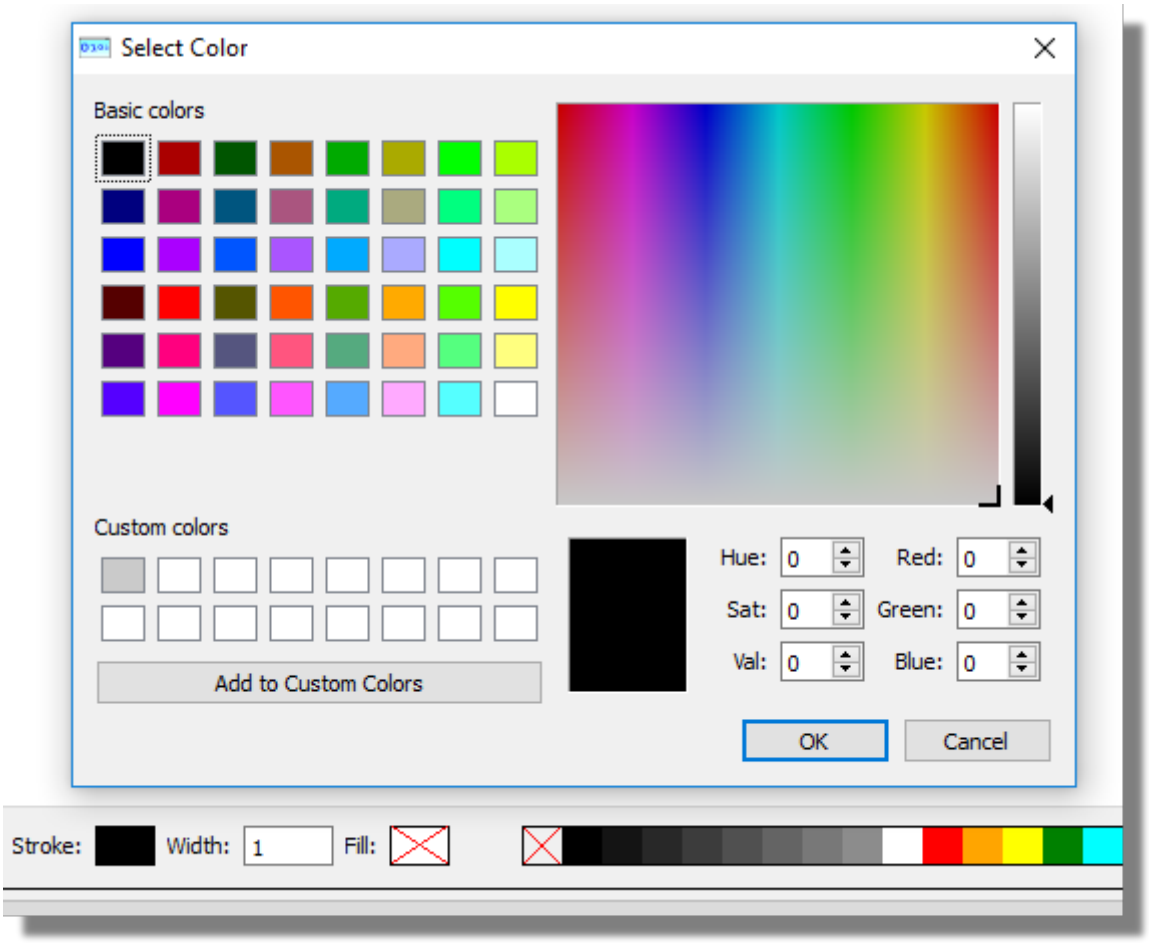
The Colour Palette

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.



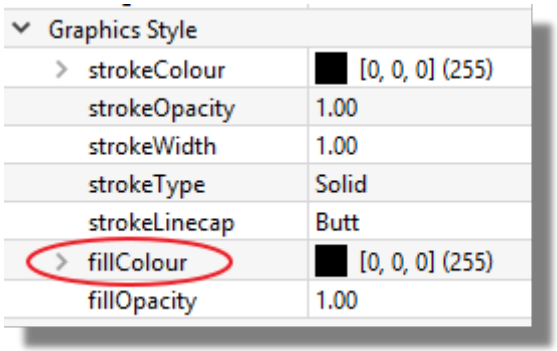
Stroke

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

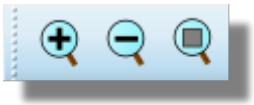
The fill specifies whether the object is wireframe (not coloured in) or in which colour it should be filled. You can change the fill by clicking on the desired colour from the palette to the right of the existing fill colour.



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.

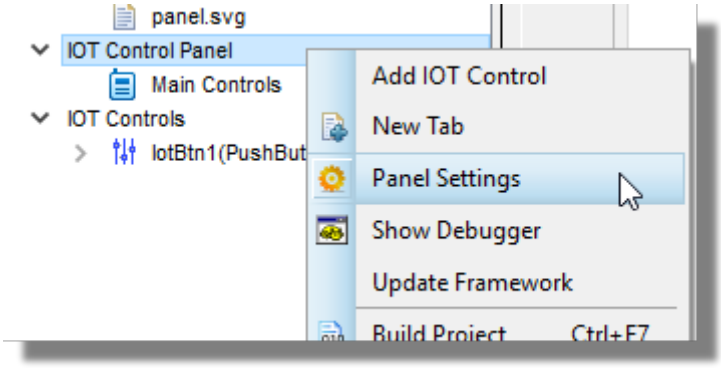


Alternatively, you can use the F6 key to zoom in, the F7 key to zoom back out and the F8 key to return the zoom level to the default view.

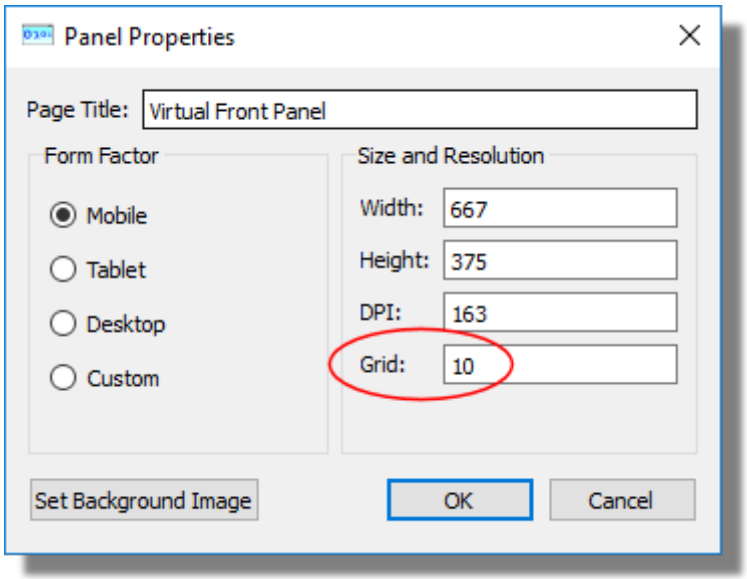
Grids and Snap

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.

See Also:

Visual Designer

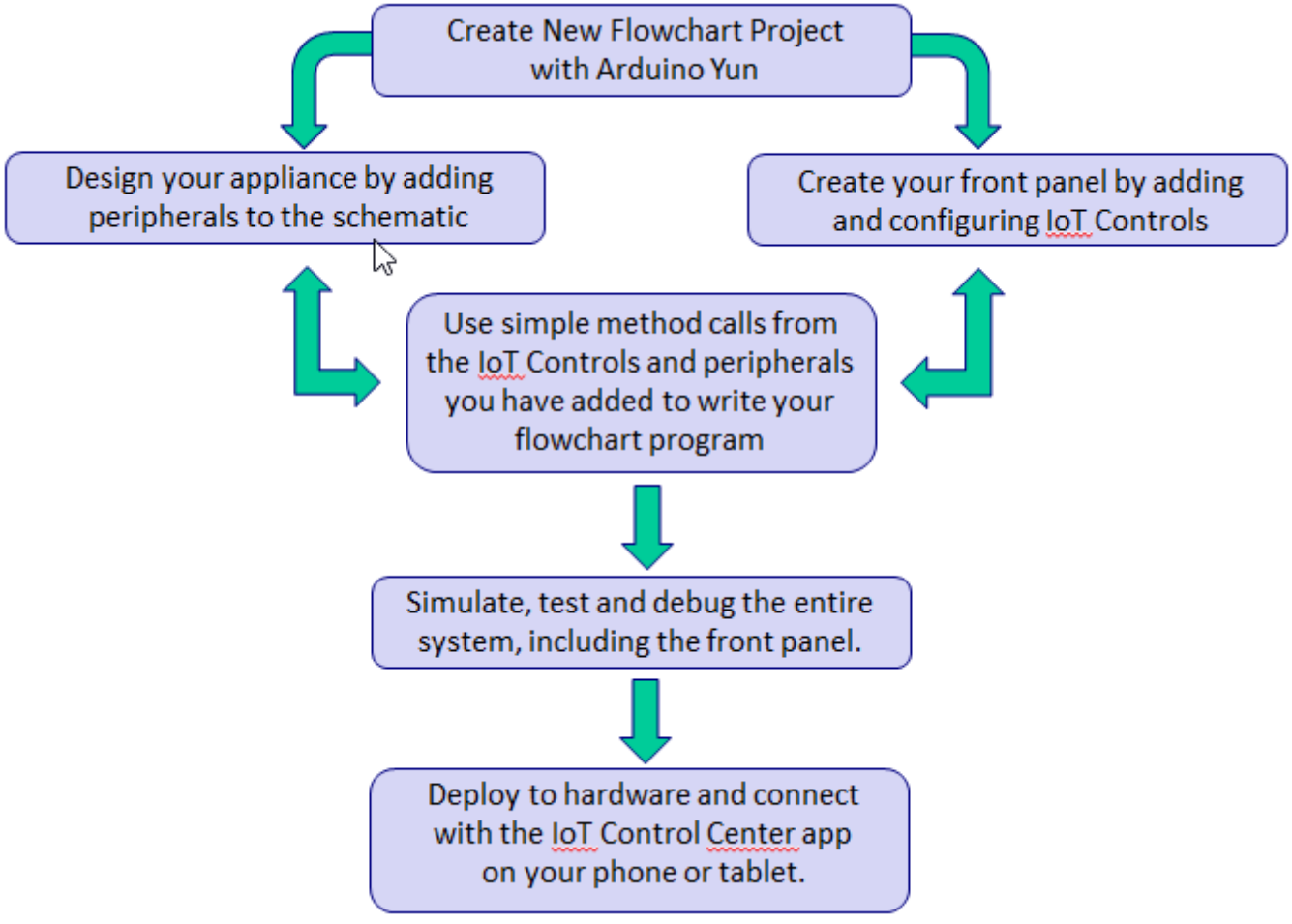
Terminology

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.

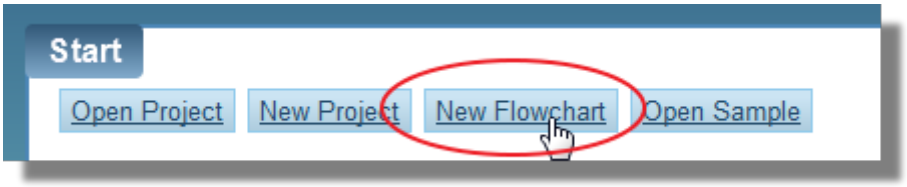
Design Process

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.

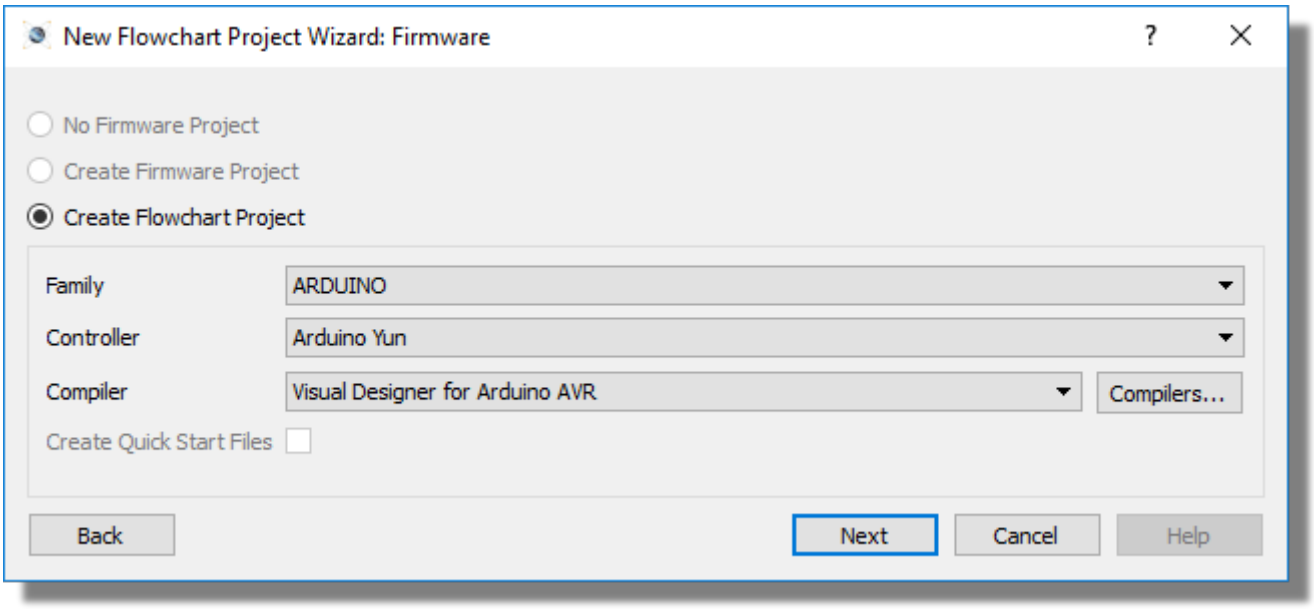


Project Setup

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.

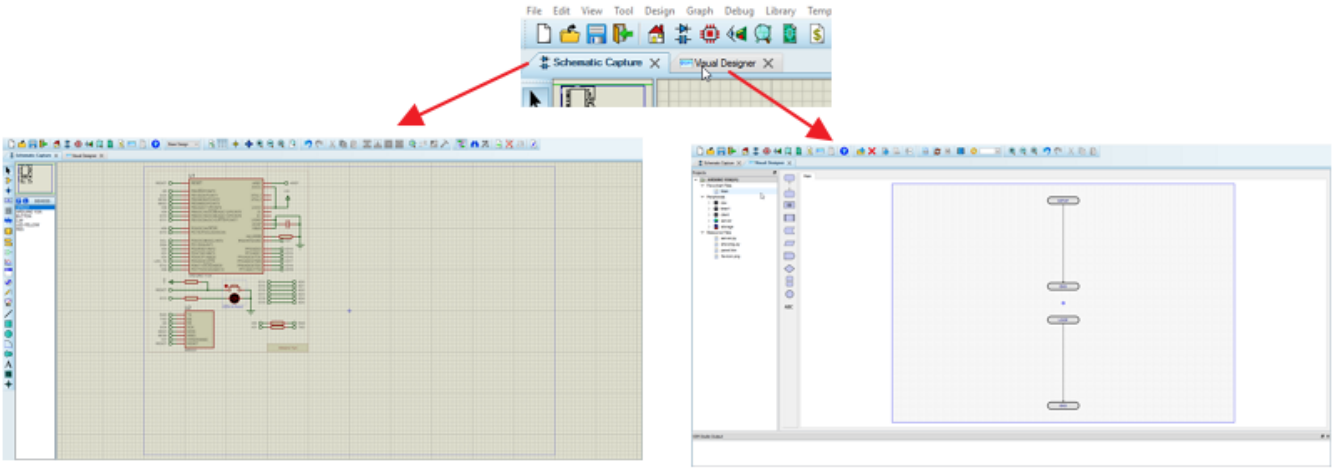


On the next screen we need to create our flowchart project. This creates the firmware which we will be using for the Arduino Yun controller in our connected appliance. This should look like the following:



We'll get a configuration summary page and can 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.

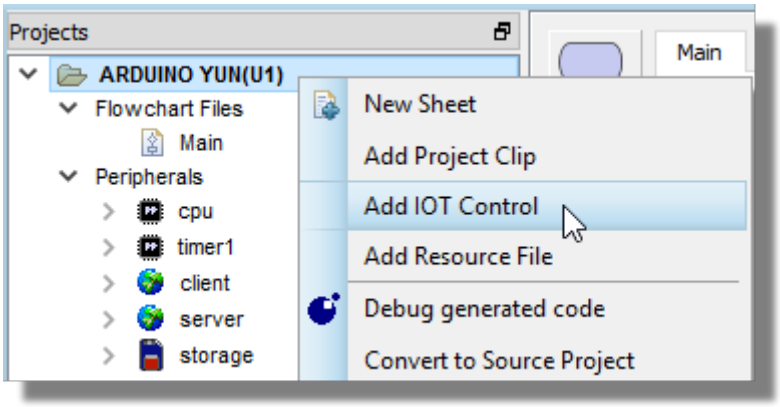


Correct Setup Complete.

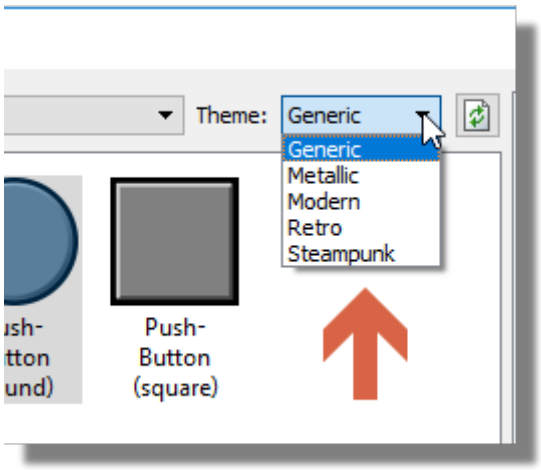
Design the Front Panel

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.

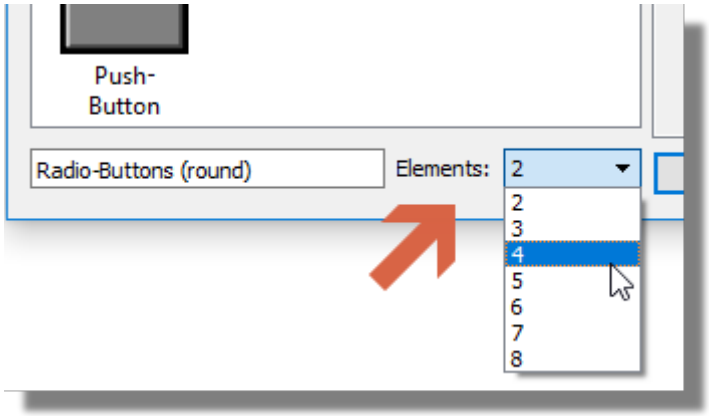


From the resulting dialogue form we need to choose a button and also a number of elements. In our case we need only one button but you could choose to have several elements in a radio button type situation. Select which theme you would like to use:



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

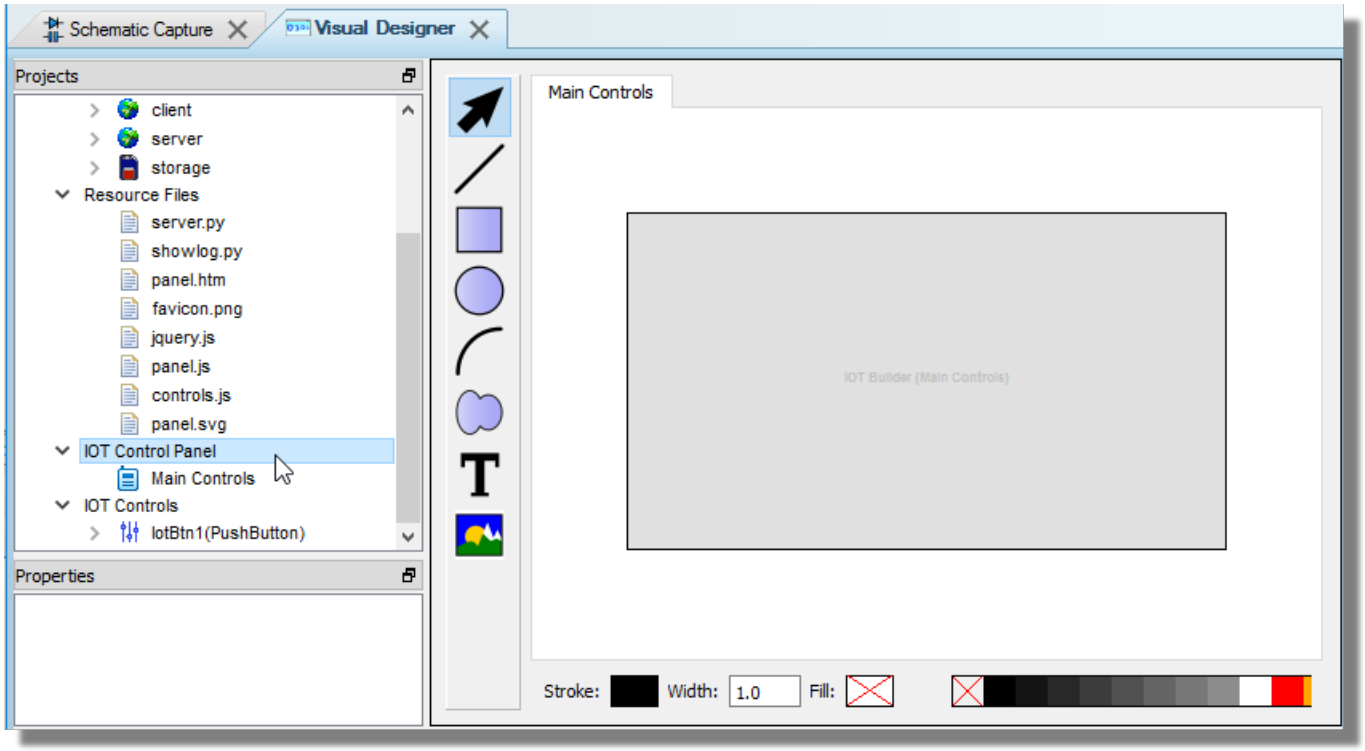
There are two options depending on the number of buttons required on your controller. A Single button (Push-Button Round/Square) or a set of button elements or a set number of buttons (Radio-Buttons Round/Square). If you select Radio-Buttons option, the elements dropdown box will be enabled for you to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



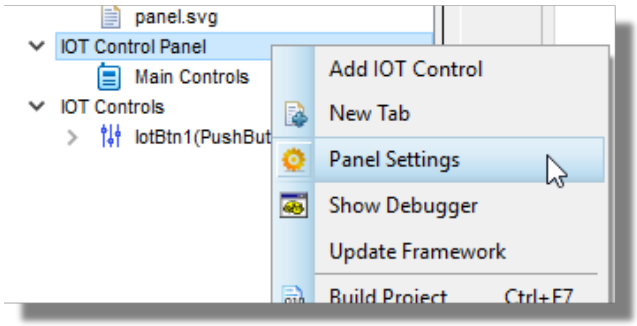
Our themes use what we call *Auto Lamp* which is a property of the push button control. When this is set to *True*, it means that the light on the button will automatically come on when it's clicked, no matter wether there is code associated to it or not. This is graphical / aesthetics only, and nothing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

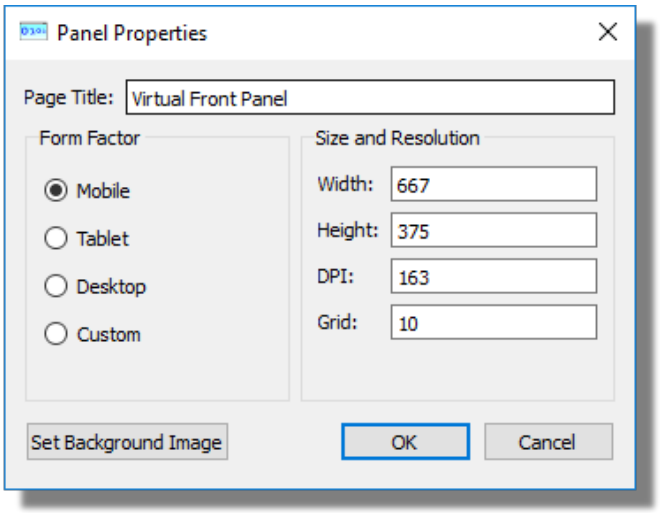
Note the theme selector towards the top right of the dialogue. This allows you to style your front panel in different themes such as retro, steampunk, modern, classic, etc.

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:



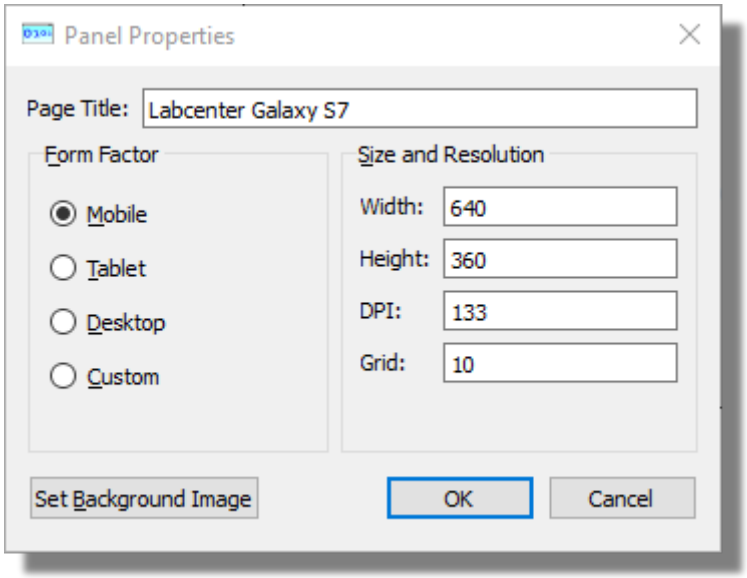
The Editing Window is now showing what will be the user interface for our appliance. By default, it is set to a mobile phone size but you can change this by right clicking on the front panel in the project tree and choosing the panel settings option from the context menu.





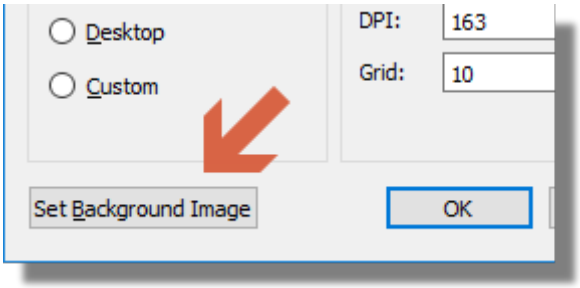
You'll find a list of phone and tablet values here: https://mydevice.io/devices/ . The width and height are the CSS Width and CSS Height in this table and the DPI is the physical PPI divided by the pixel ratio. An iPhone 7 for example is therefore 375x667 and the DPI is 326/2 = 163.

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

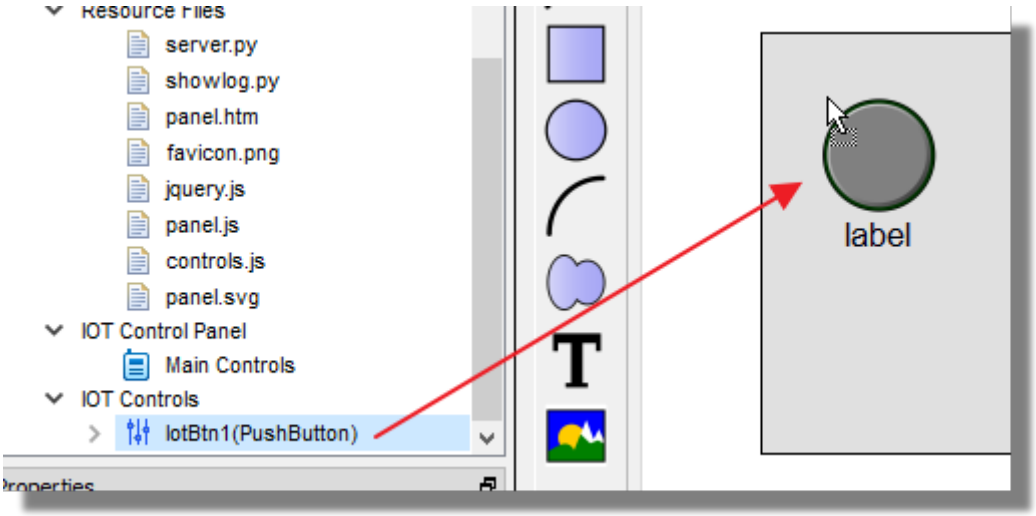


While you can change from a desktop browser to a tablet or a phone at any time it is really best to make a decision early on because any placed controls will not move if you resize the panel. This means if you resize the panel downwards (e.g. tablet to mobile) it is awkward to re-position all of the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

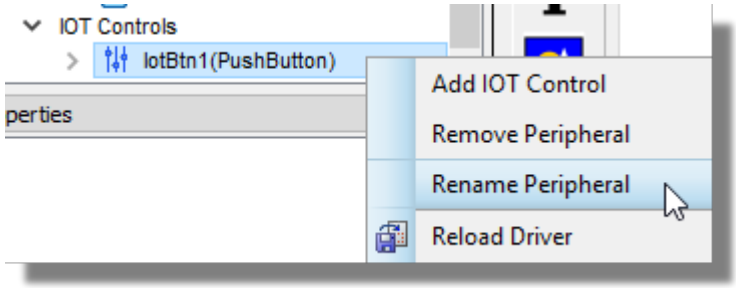
You can also change the background image. The default is a light grey box for simplification, however you may want to upgrade this to something which matches your theme. Simply download an image as a JPG or PNG file in to the Background folder and insert it using the Set Background option:

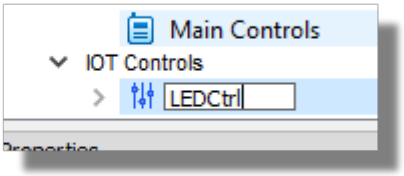


To place our button on the panel simply drag and drop from the button entry on the project tree to the panel.

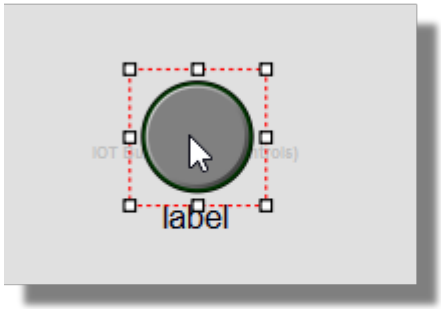


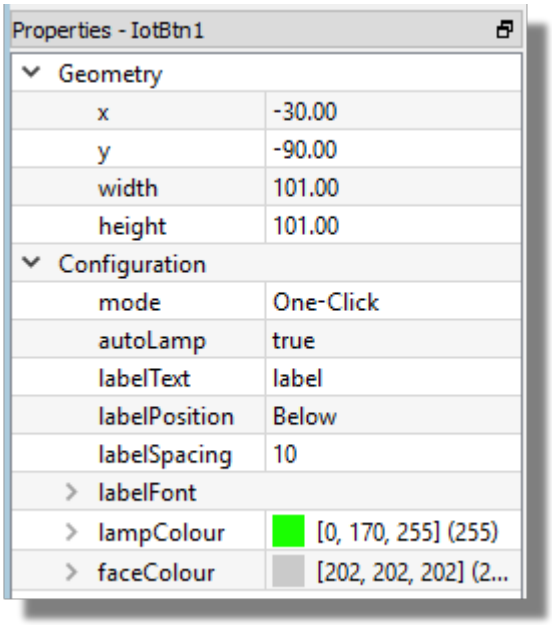
To rename the button right click on the item in the project tree and select rename peripheral from the resulting context menu.



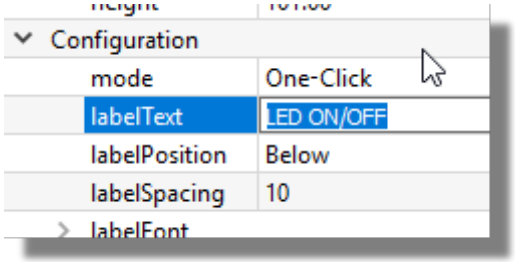


We can edit almost anything we need to with the button via the property panel underneath the project tree. This shows all of the configuration properties available for the currently selected object on the panel. If the button is not selected (does not have a dashed box around it) click left on it once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

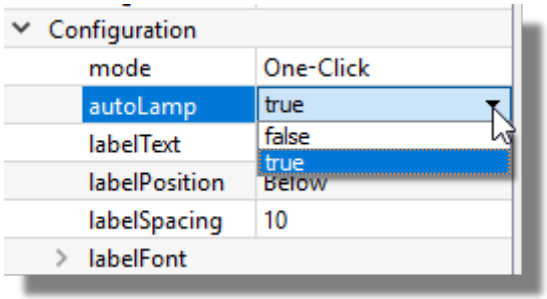




The property section for a button is fairly self-explanatory, consisting of a geometry section for accurate positioning and a configuration section. Here we can specify the type of button (single click is fine) and change the label to something sensible.



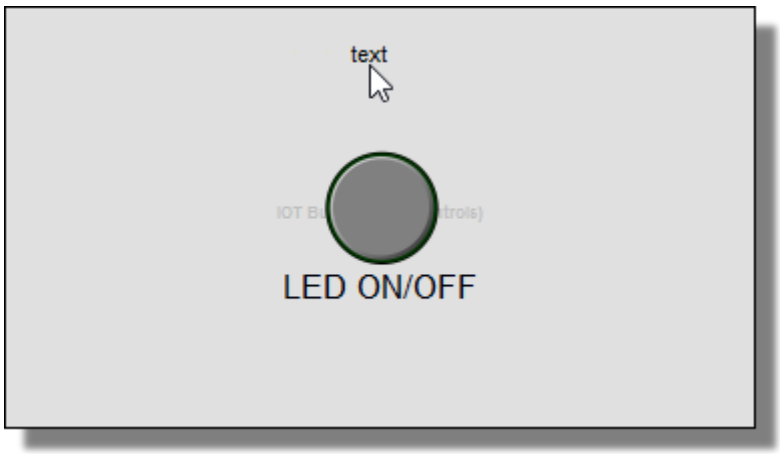
AutoLamp Property



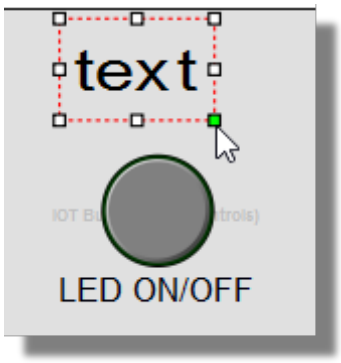
You can set an autolamp property on the IoT button. If so, it becomes a toggling button by default. However, if you turn autolamp off then you will need to use setLamp() on and off in the flowchart - the benefit of this being that it is

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.

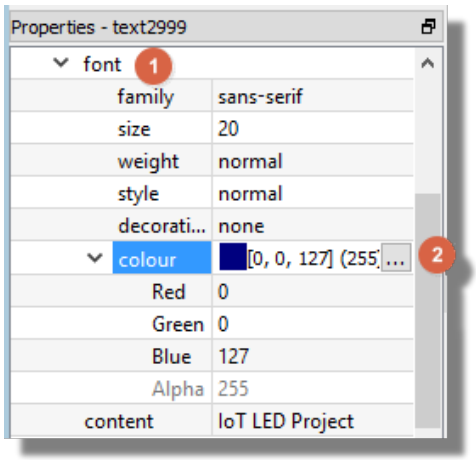
You can add title or graphics to the front panel directly from the primitives on the left hand pane. For example, to add a title we would first select text mode and then place the text indicator in approximately the right position on the panel.

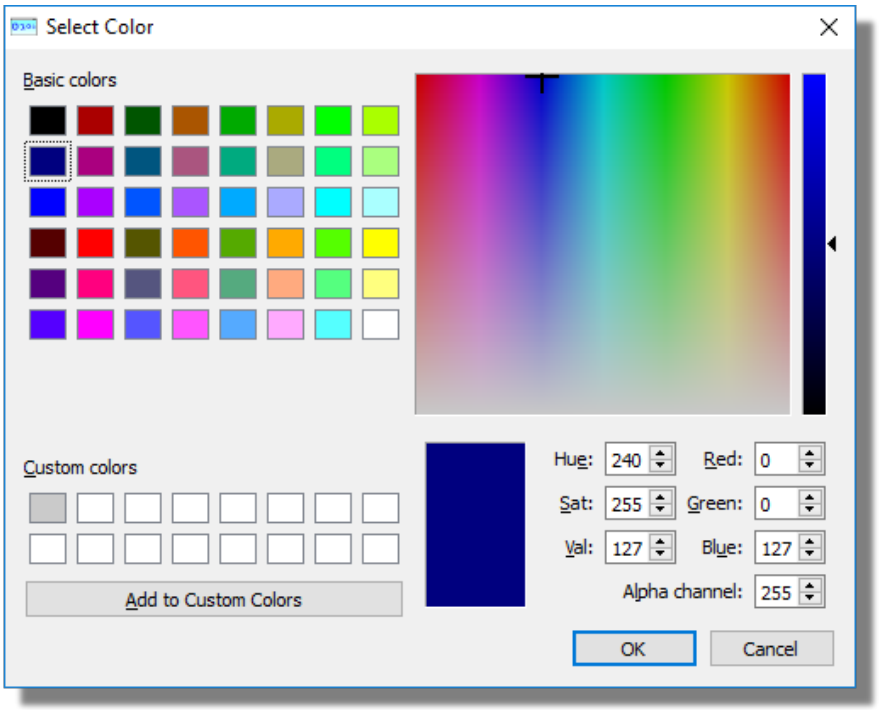


After placement, we stretch the text box to the size we want.



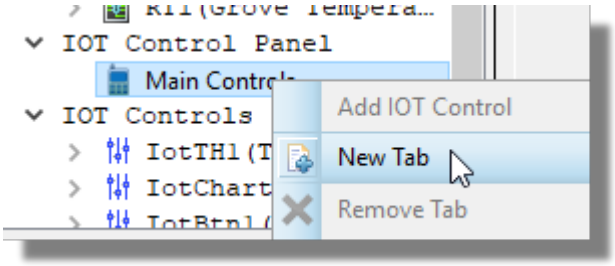
Finally, we can change the font, colour and text content in the property page. In the left hand pane, drop the Font option down and click on the colour.

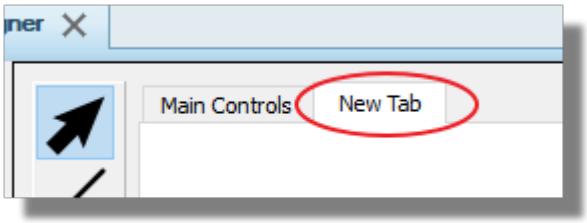




If you have Radio buttons and wish to resize them, Right click on the button > Transfer > Restore Aspect Ration. More information is explained in the Guided Tour section.

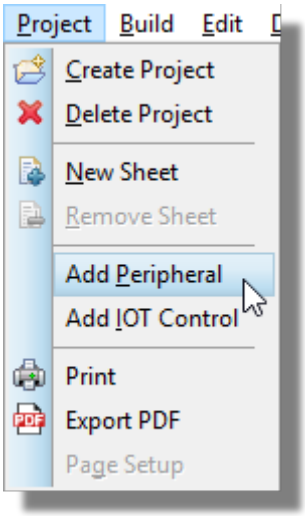
> If you require more than one front panel, you can add multiple tabs by right clicking on the 'Main Controls' in the project tree and select New Tab:



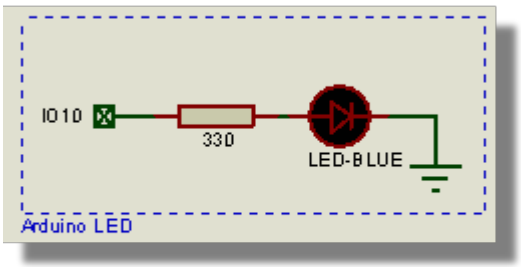


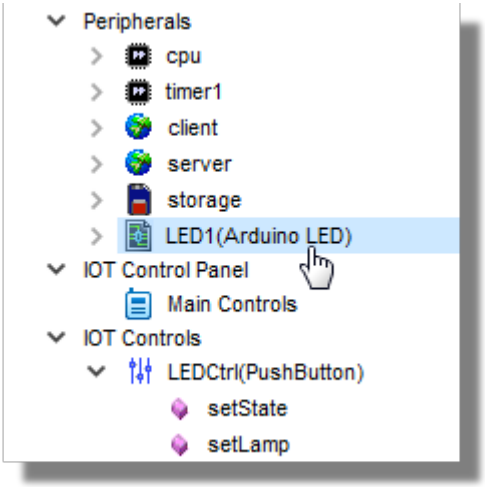
Writing the Program

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.

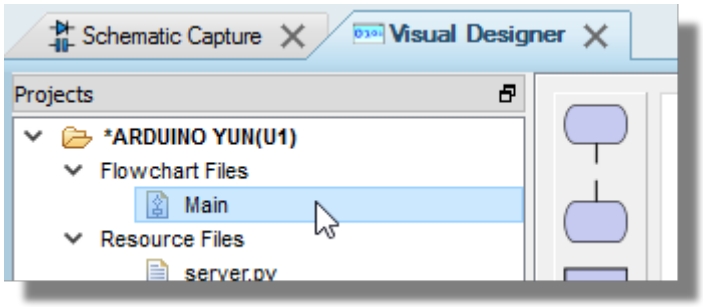


This will both add the LED (Blue, Yellow, Green or Red) to the schematic and it will appear in the project tree in Visual designer.

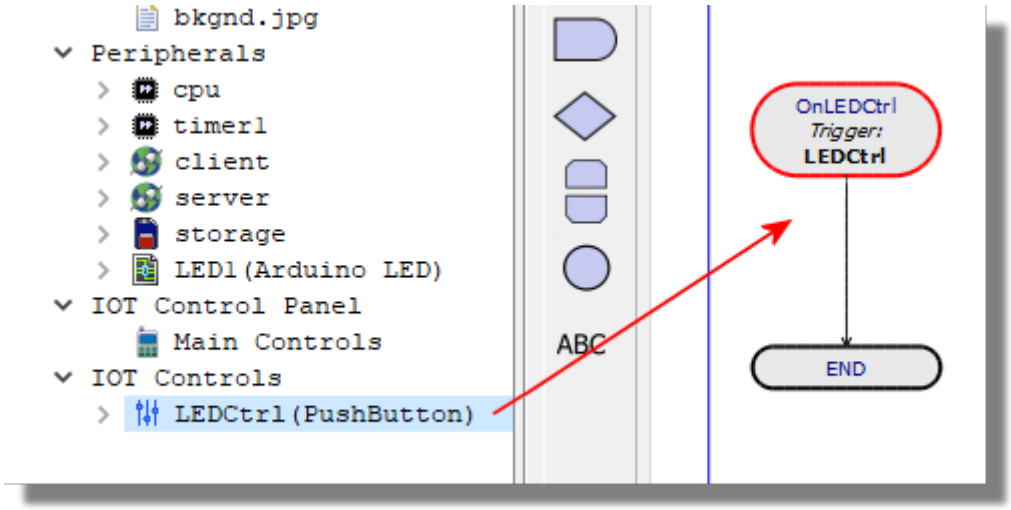




Now double click on the main flowchart page in the project tree to switch to the flowchart program.

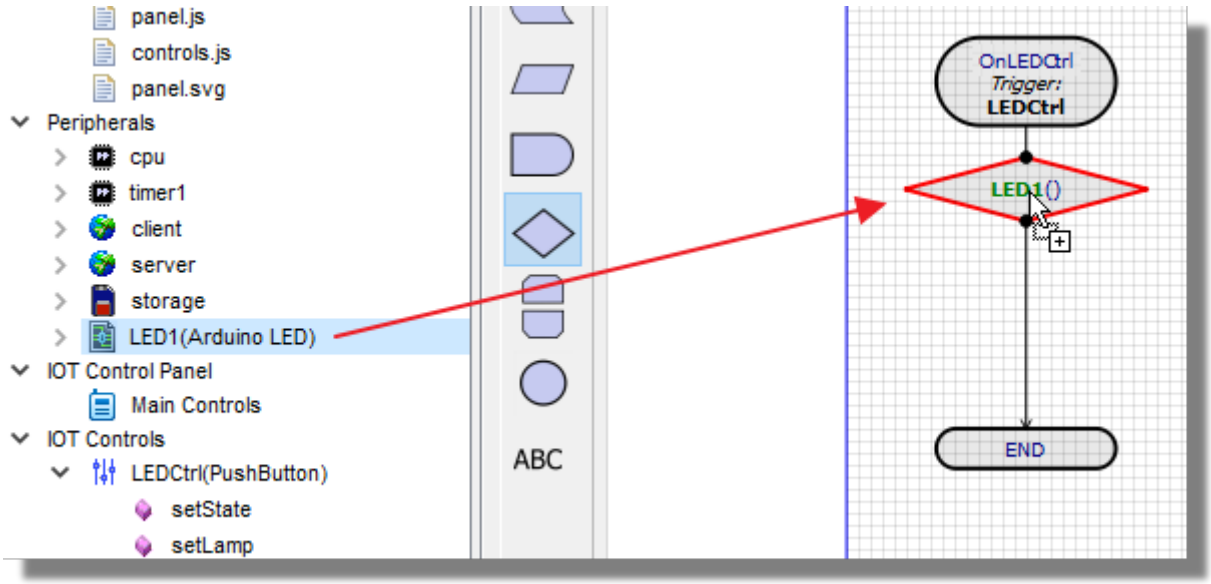


What we need to do in the program is respond to the press of a button and either turn the LED on or turn the LED off. In Visual Designer we do this by creating an event that corresponds to the button click. You can do this really quickly by dragging and dropping from the button in the Project Tree as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



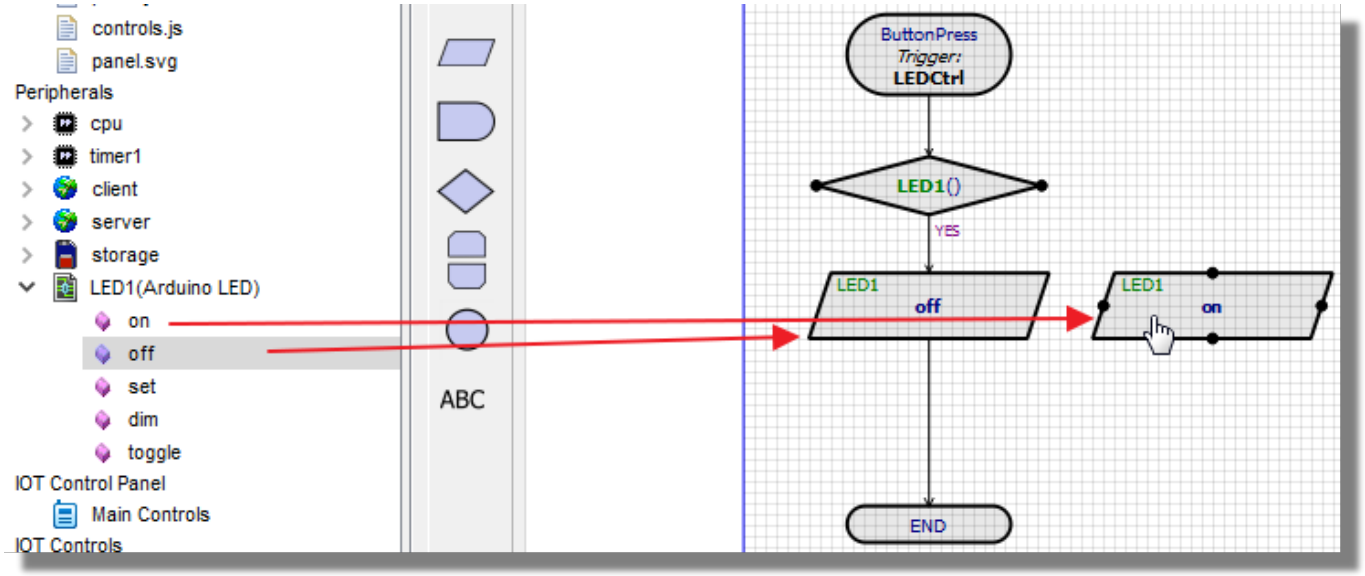
This is a shortcut which adds an event and then specifies it to respond to a button press interrupt.

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.

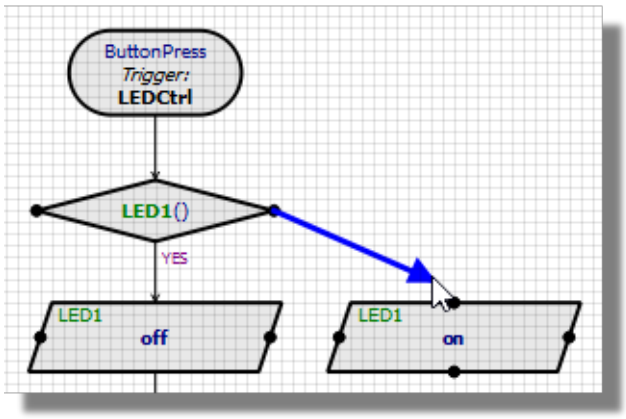


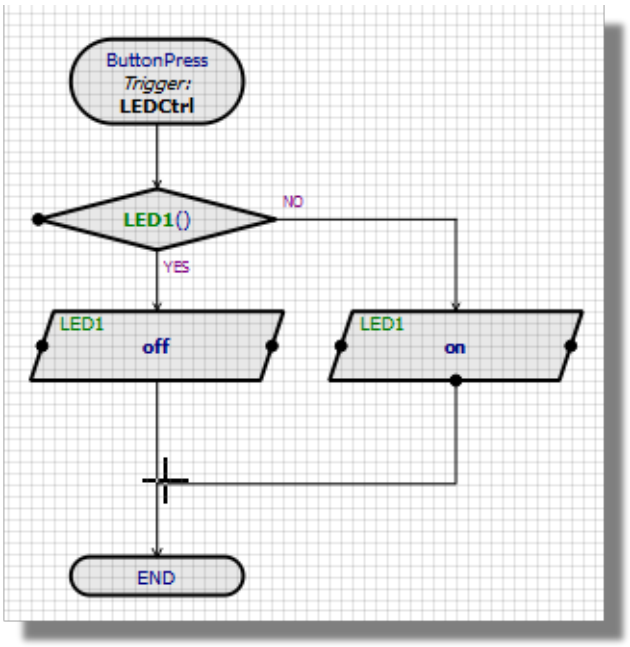
This is actually quite a powerful shortcut. When you drag and drop from the actual peripheral (rather than one of it's methods) you will get what we call the sensor function for the peripheral. In the case of the LED this is a decision that splits on whether the LED is on or off. You'll find more about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

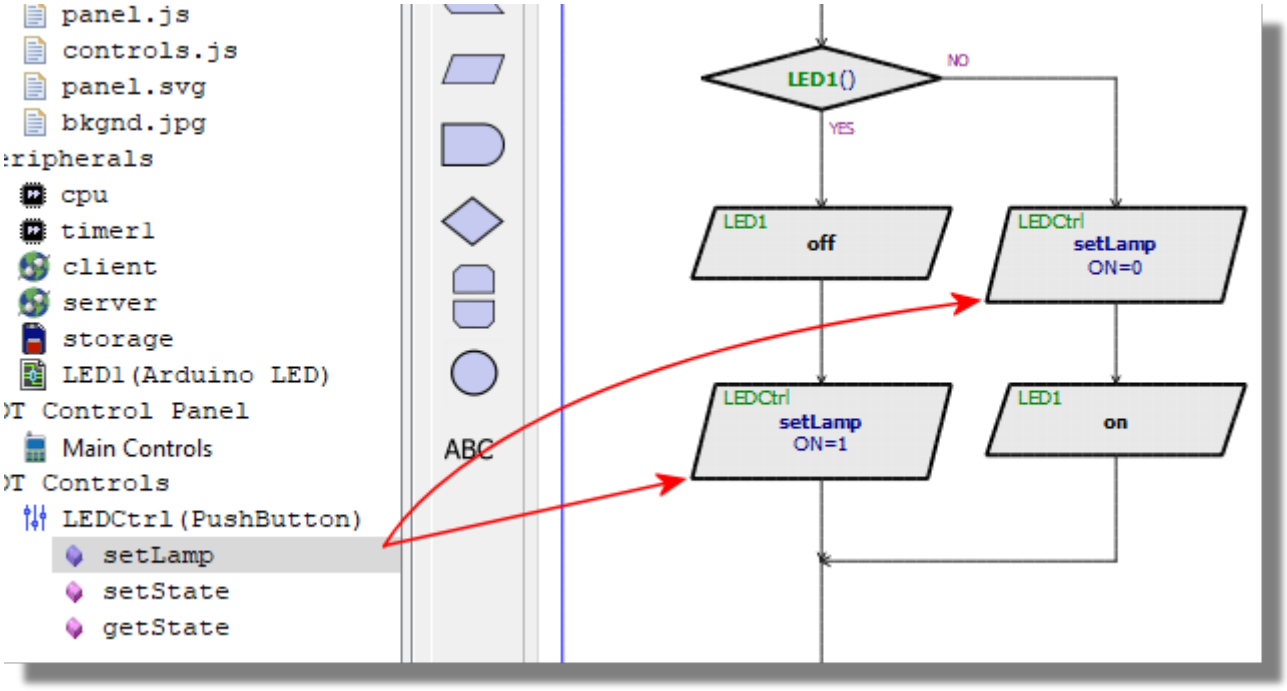


Finally, we'll need to place a couple of extra wires to connect up the ON state.





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.



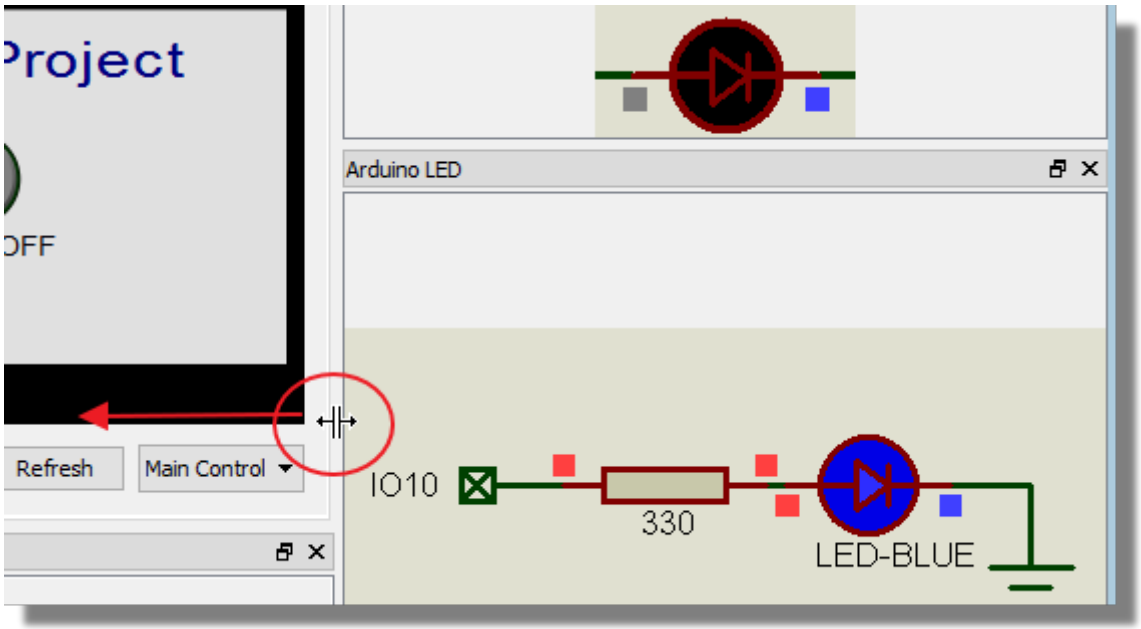
That's actually it - five flowchart blocks to toggle an LED on an Arduino Yun from a mobile phone ! Next we'll look at how we can simulate and test.

Simulate and Test

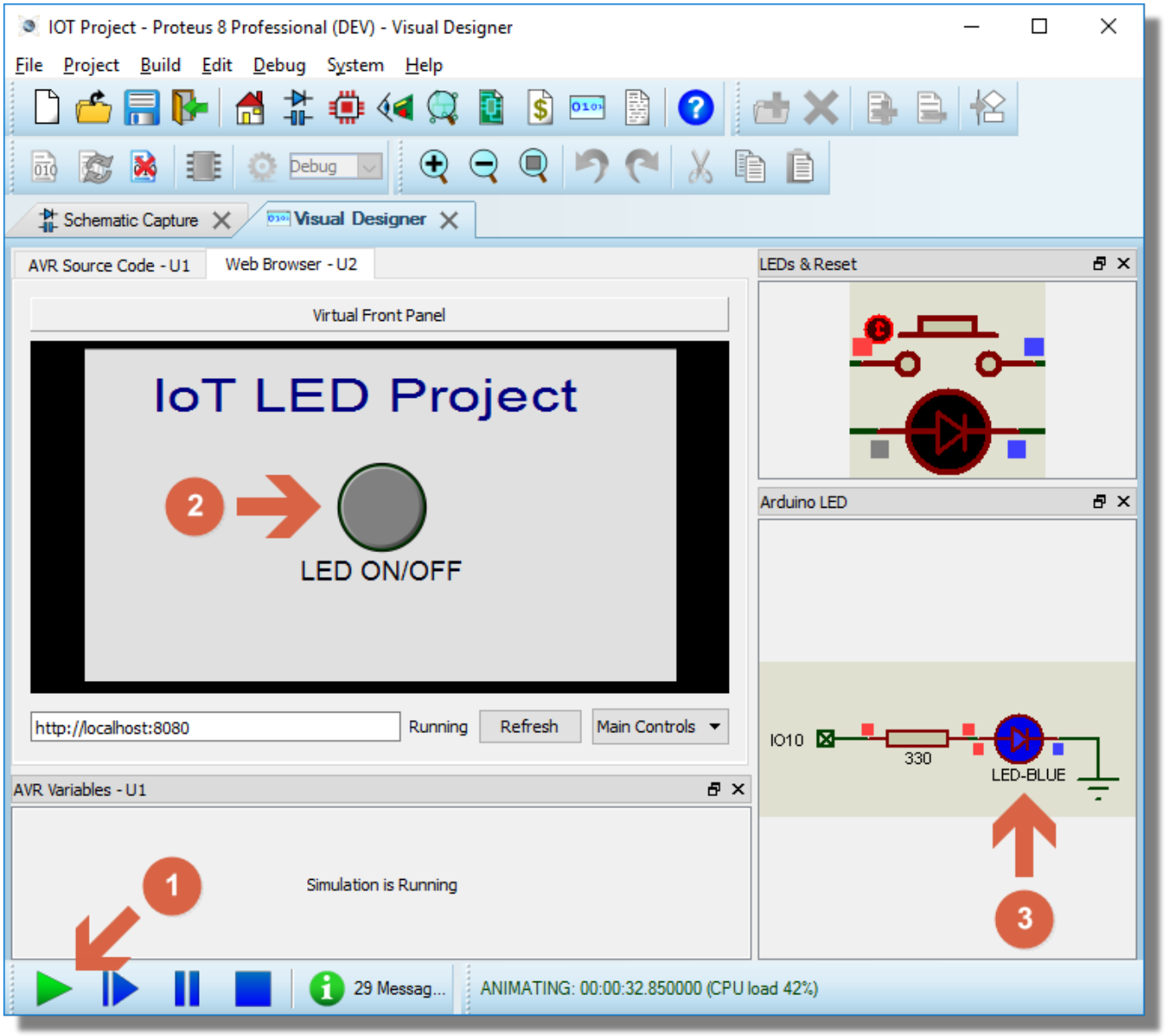
As a quick test that we haven't made any mistakes we can press the play button to simulate the system.



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.



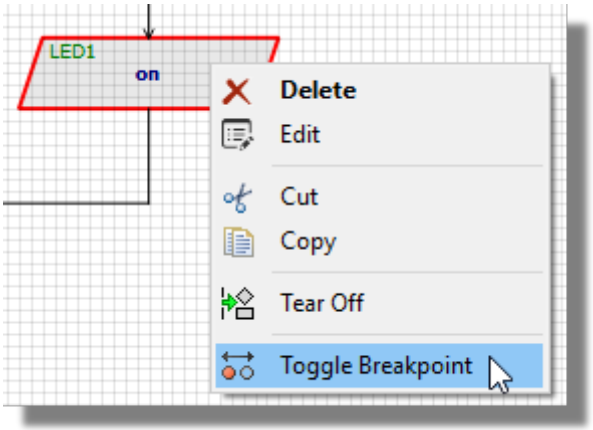
All that remains is to press the button and then wait to see if the LED turns on and then press the button again to turn the LED back off.



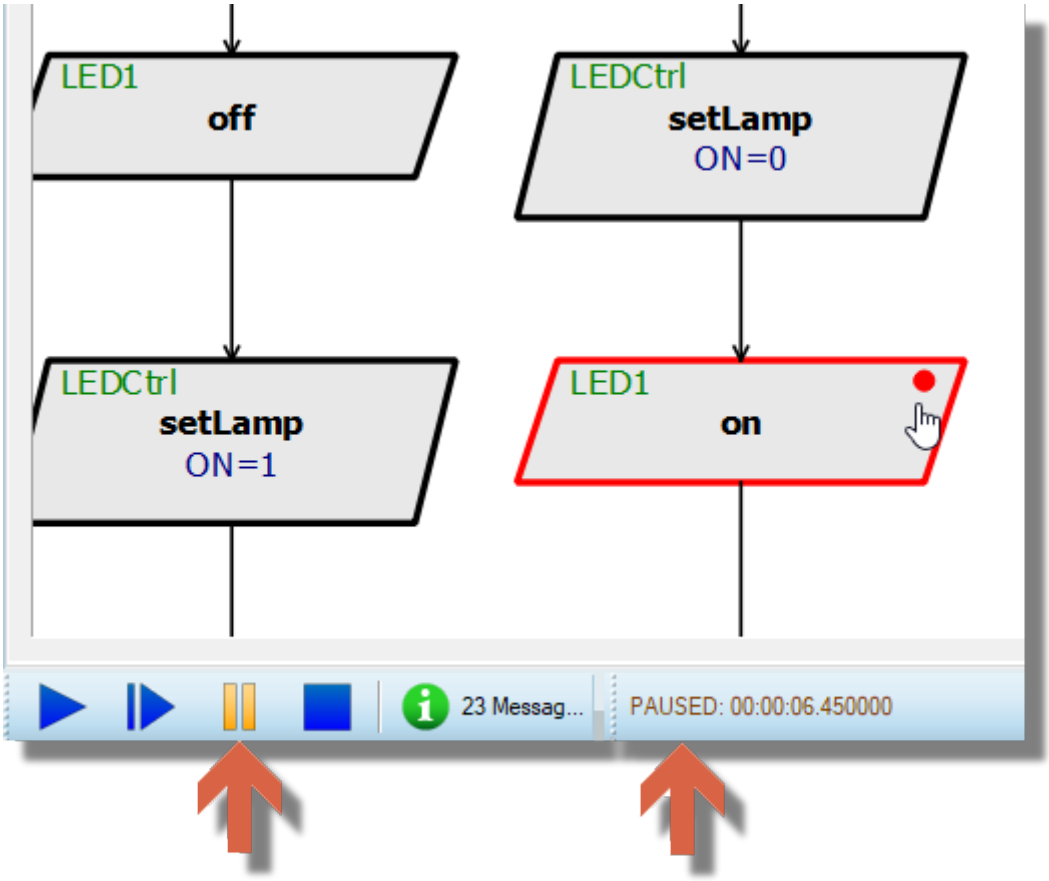
Let's assume however that something wasn't working and that we had to investigate. We'd start by pressing the pause button to halt the simulation.



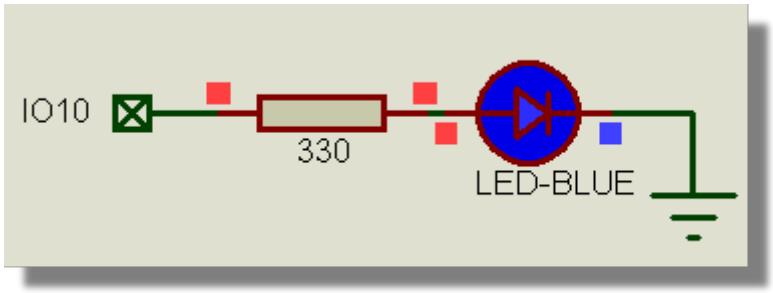
Next we'd set a breakpoint somewhere in our flowchart. In our example, let's do that at the method call to turn the LED on.



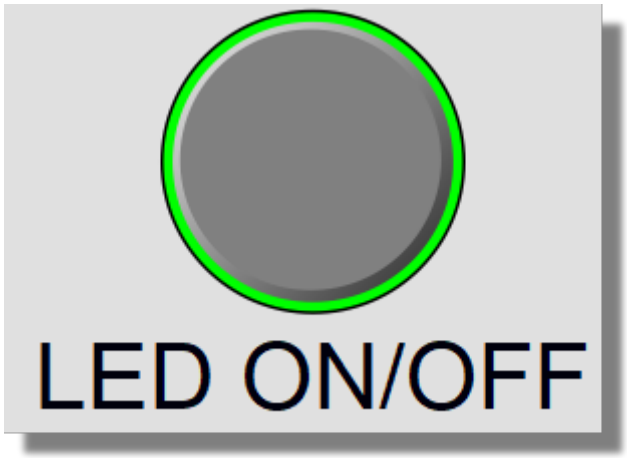
Now we run the simulation and press the button on our panel. The simulation will pause automatically when it hits our breakpoint.



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.

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!

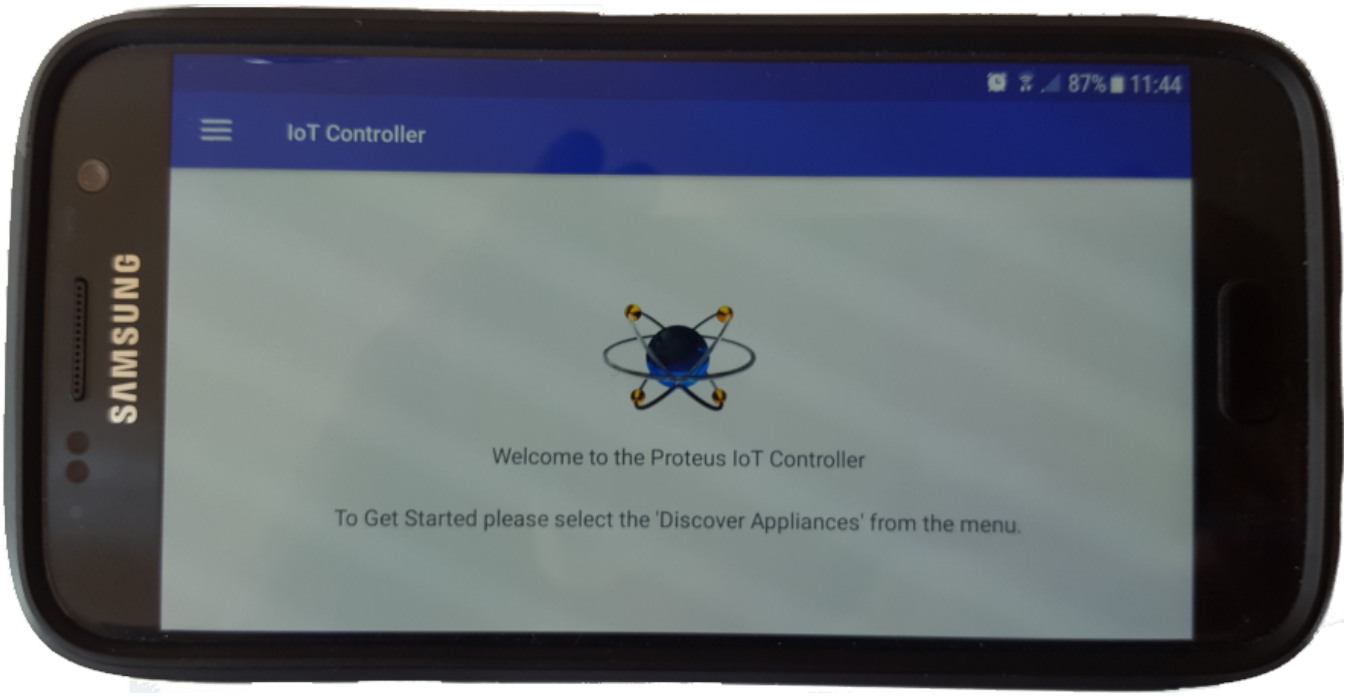
The app serves as a host for the front panel you create and also helps with discovery. The discovery is done by the *Bonjour* utility / Driver. This utility is present on many machines but you can launch the installer directly from the Labcenter Program group on the Start Menu. Now that you have the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

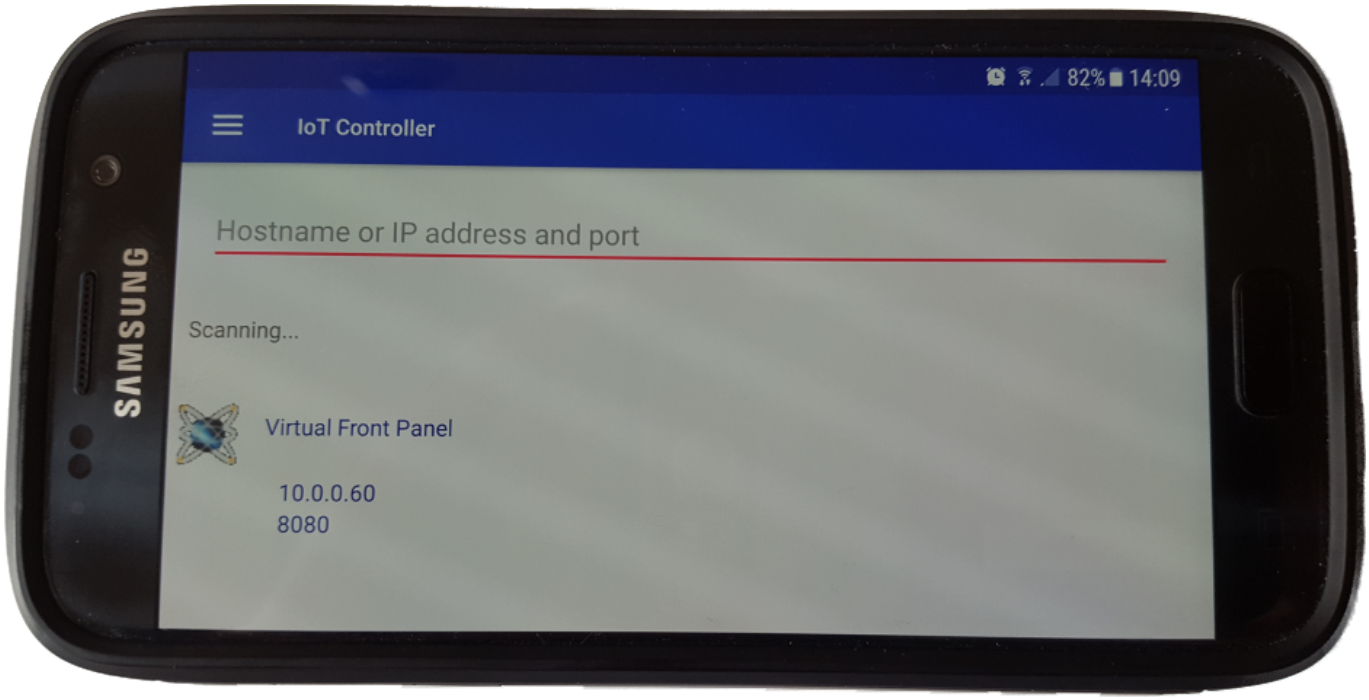
2. Start the running simulation in Proteus, then start the app on your phone.



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.



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:



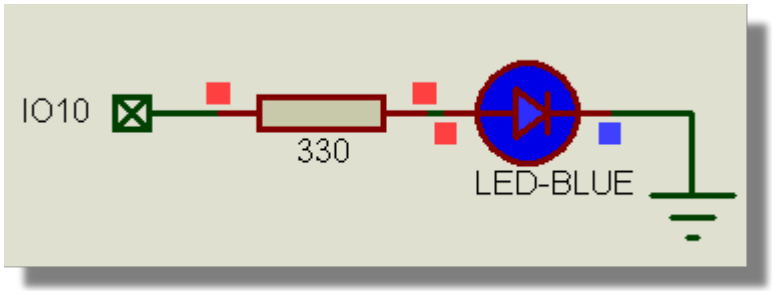
This is because it is now being displayed on the controller.

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

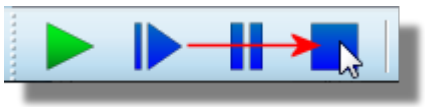
7. Press the button on your phone, the led will come on in the simulation and the indicator lamp will light up on your phone app.



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

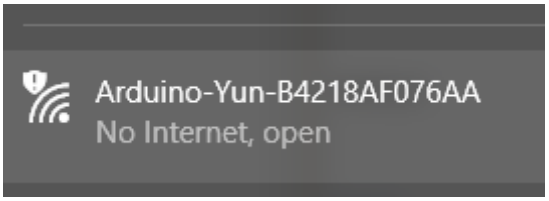
Now that we have the app and our mobile control center is working the final step is to program the physical hardware with our firmware. We can do this either through wireless SSH if your computer has wifi or through USB if not. The process is basically the same in either case. For this example we will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

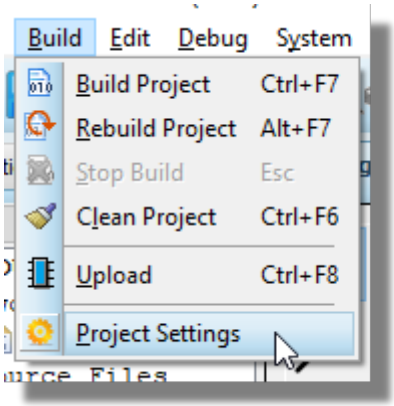


2) After waiting for about 30 seconds for the YUN to boot, Set your PC and your controller to be on the ARDUINO wireless network:



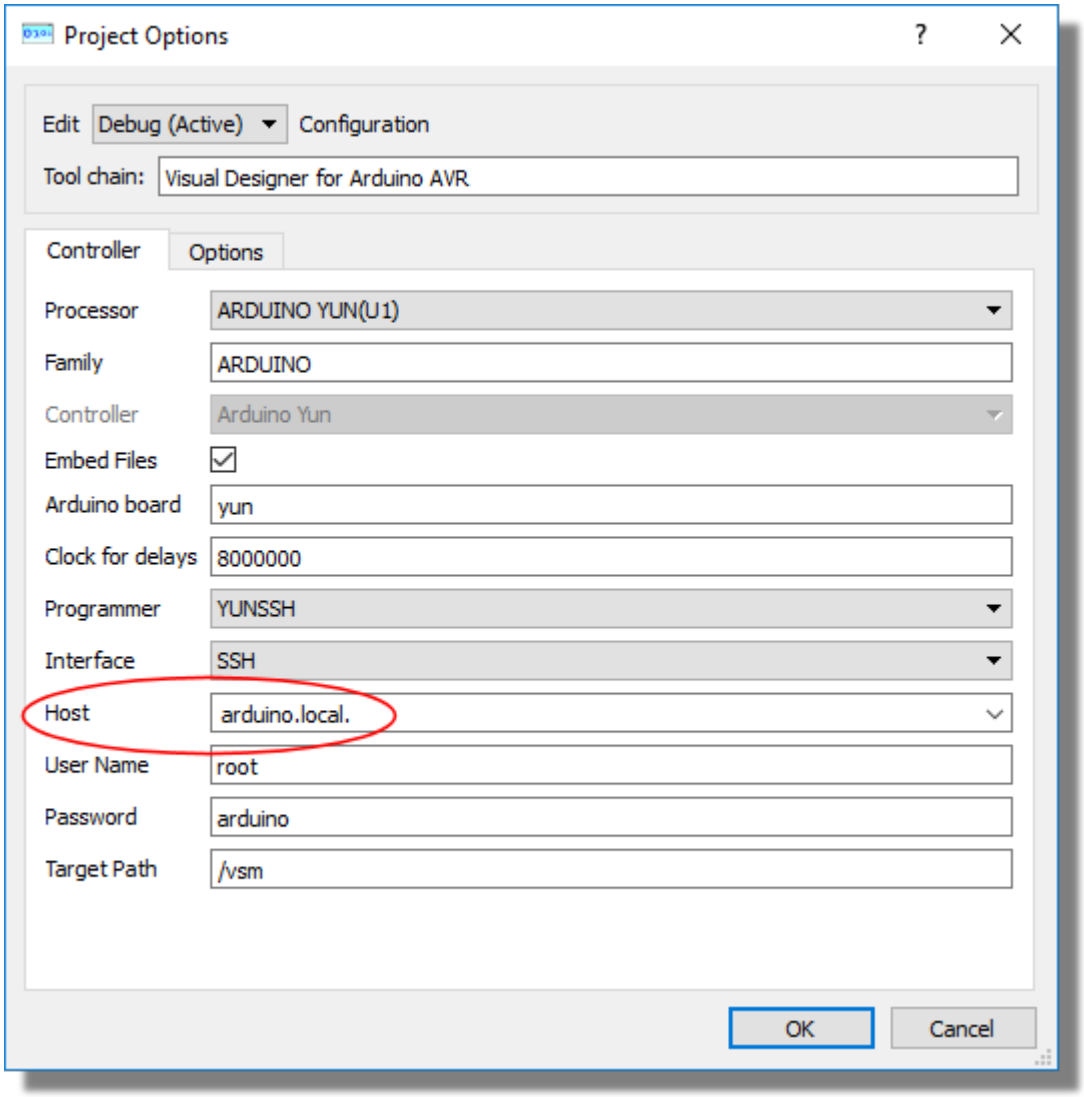
This will take you off the internet but you won't need it for this exercise.

3) Launch the project settings dialogue in Visual Designer.



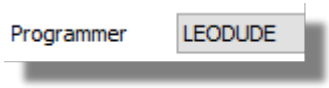


4) If you are programming by WiFi select the YUNSSH programmer and the SSH interface. Your Arduino Yun should be discovered automatically and shown in the host dropdown combo box.

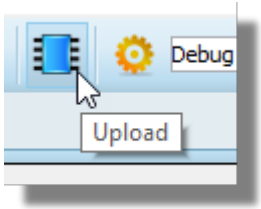


The default name of the Yun has the mac address appended at the end and you should find this written on a sticker on the back of your board.

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.

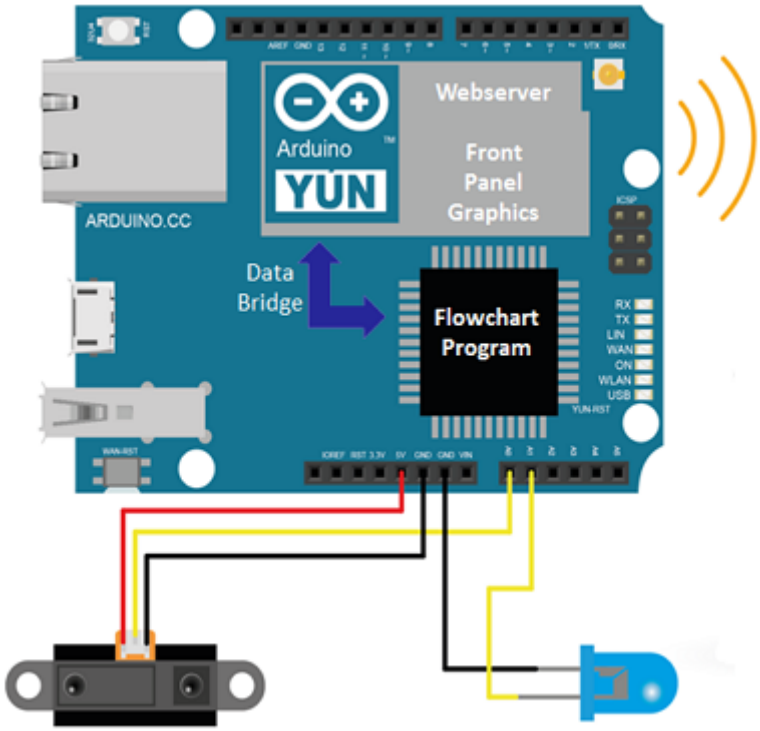
How does it work ?

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.



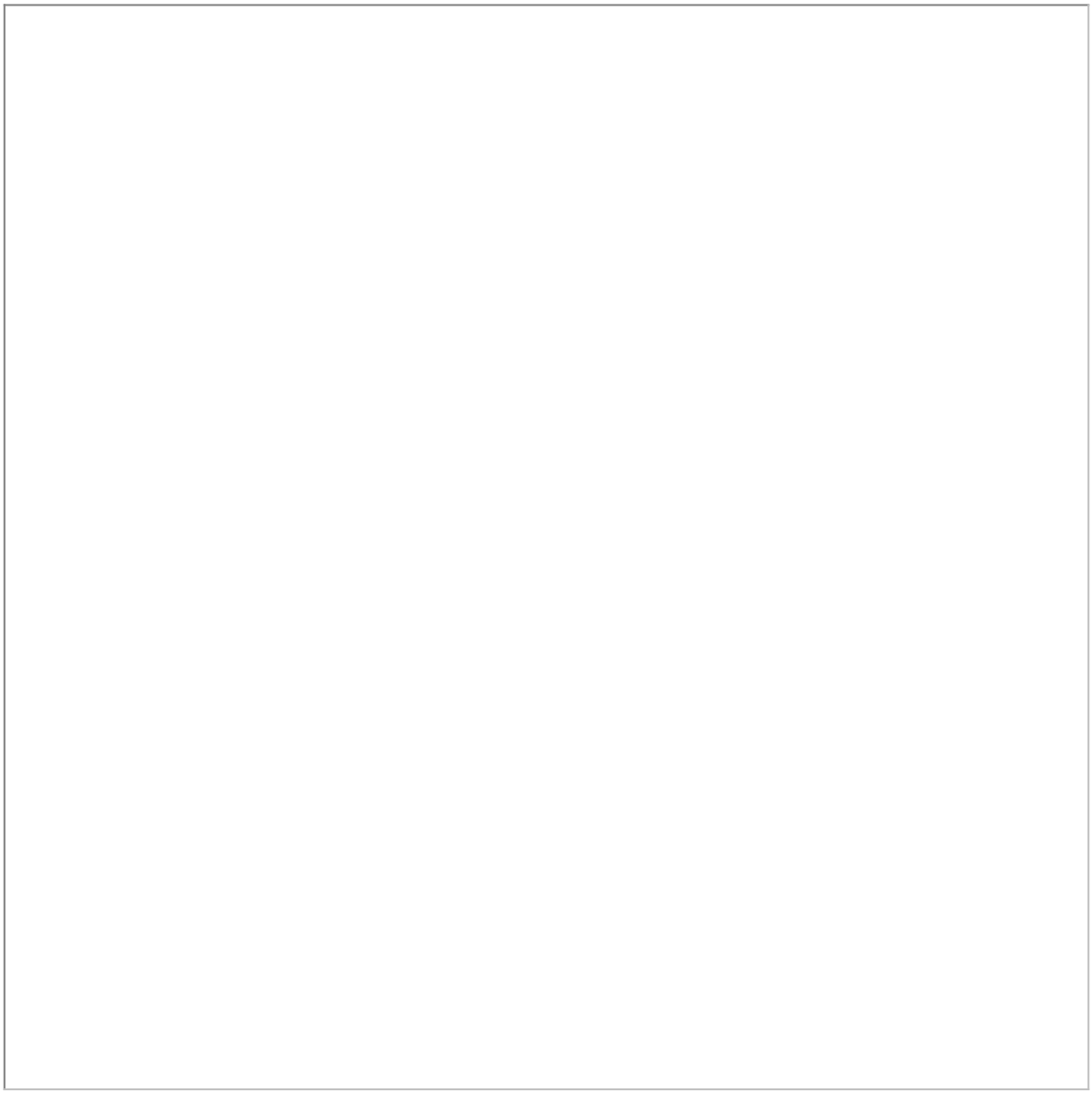
Flowchart program on AVR, webserver and front panel graphics on Atheros.

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.

See Also :

- Tutorial 2 : Temperature Logger.

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.

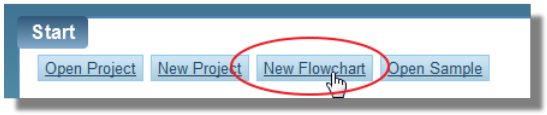
While relatively simple with IoT builder, this is still a bit more involved than our blinking LED tutorial. To avoid information overload we won't go over the same basic techniques that we covered in the first tutorial so it is worth either working through that one first or at least referring to it where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

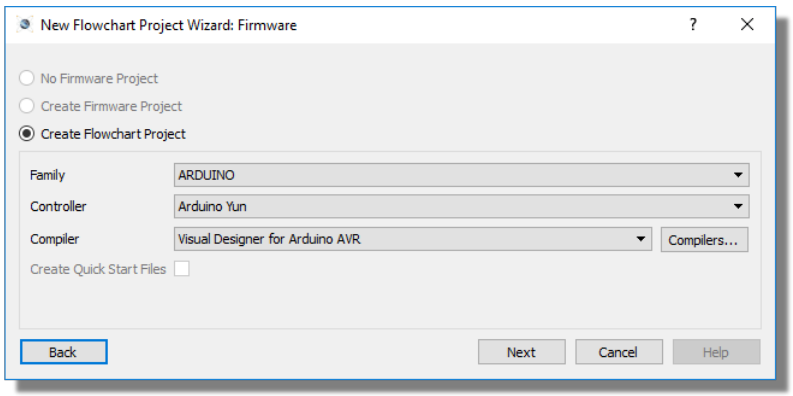
See Also:

- Tutorial 1: Blinking an LED

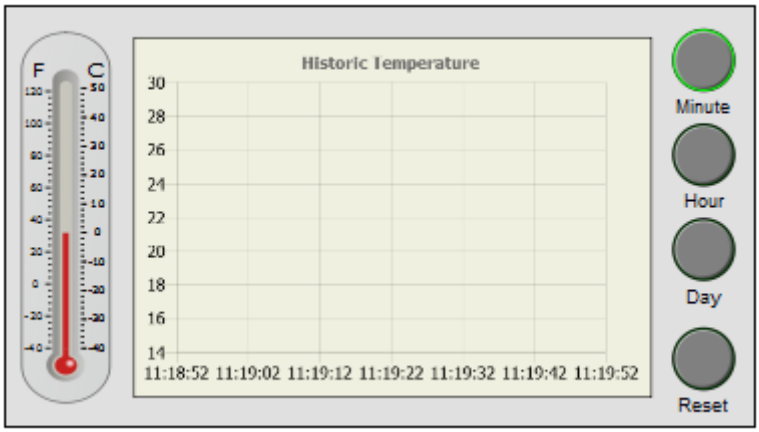
Front Panel Design

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.





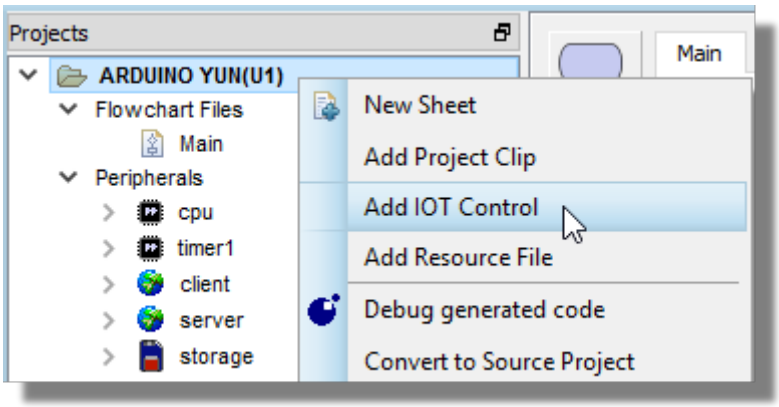
Our front panel for the logging thermometer is going to look like the screenshot below.



Front Panel concept for a logging thermometer application

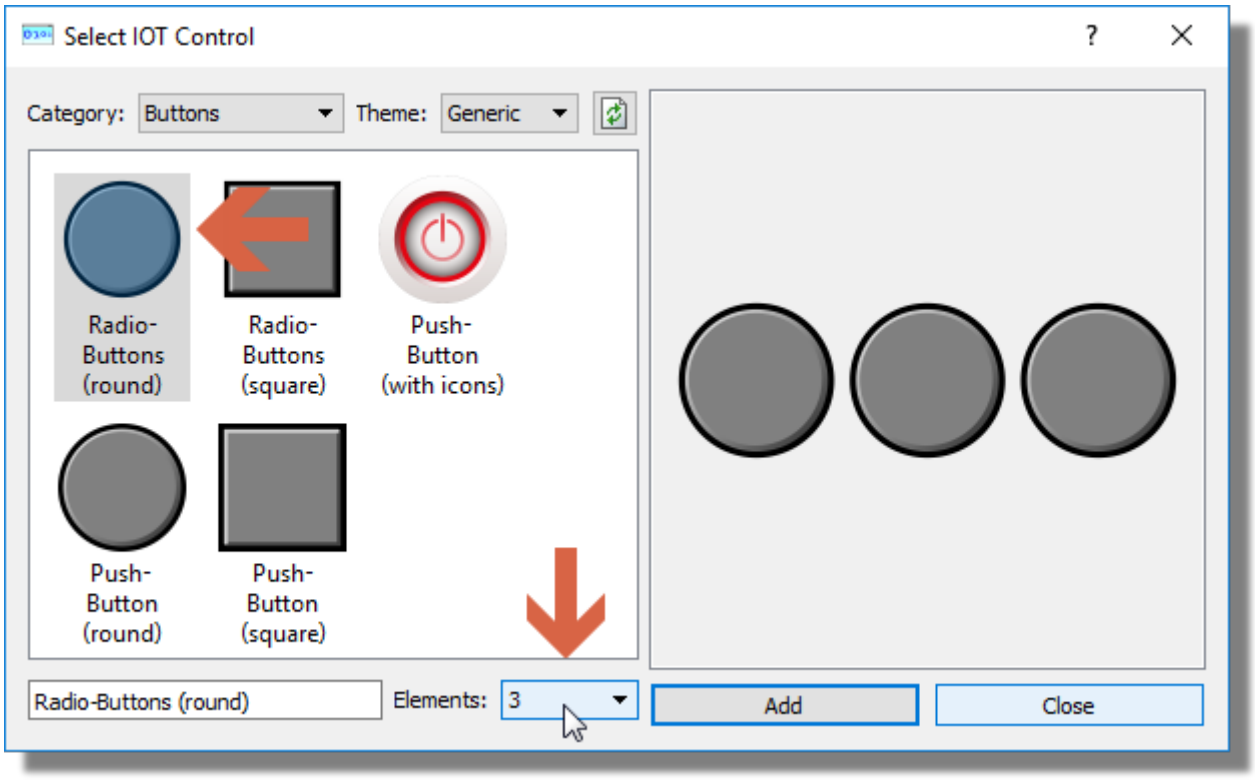
Control Picking

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.

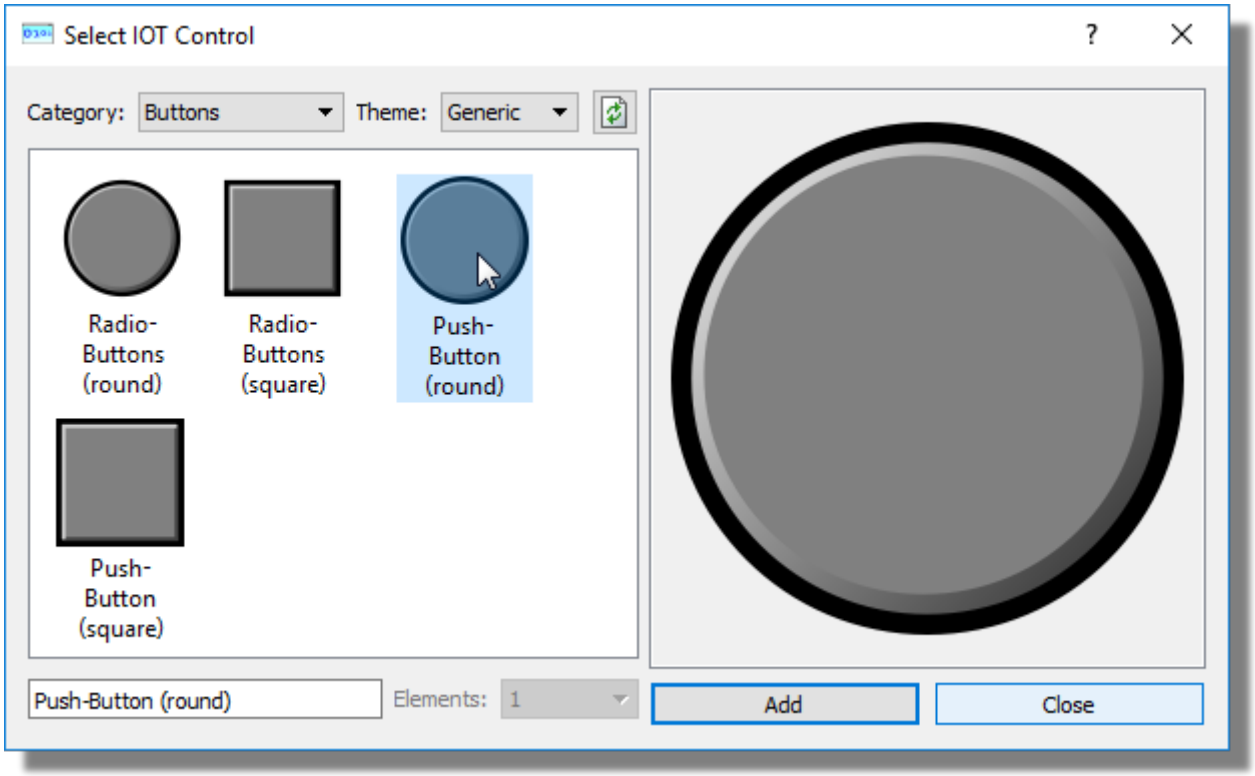


We want a mercury thermometer (-40 to 50 Celsius should be fine !) which can be found in the Display Controls category, a line chart with time x-axis which is under the chart controls category and then we need four buttons from the buttons category.

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.



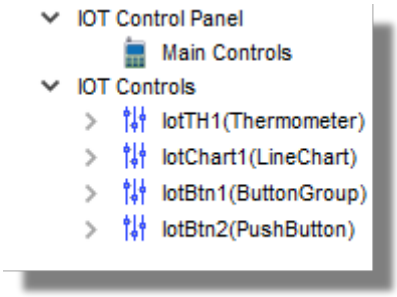
The reset button by contrast is a normal, stand-alone button so we can select push-button and add in the usual way.



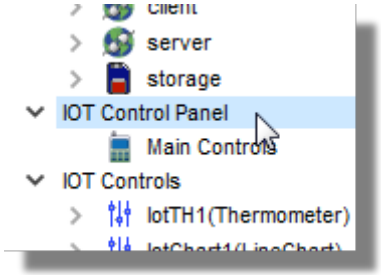
In the screenshot above and in our selections we've chosen to add everything in the basic (generic) theme. You can of course change the theme from the top right and style your front panel in a different way if you like.

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.

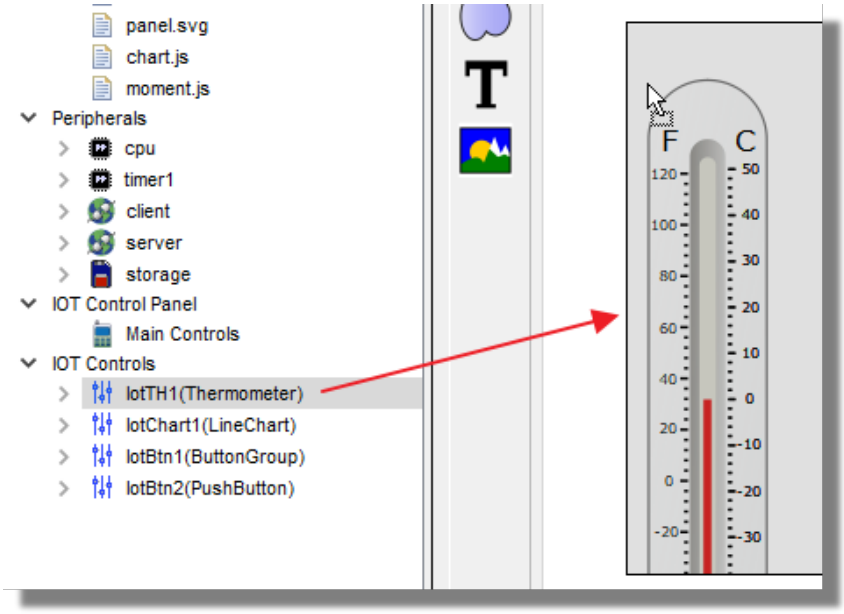


Double click on the IoT Control Panel to open it in the Editor.

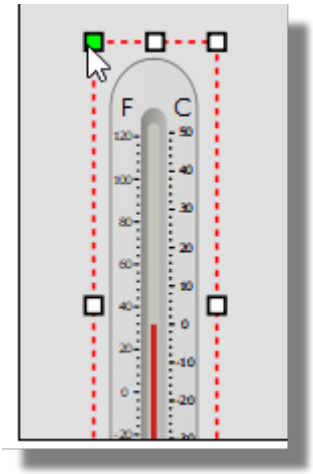


Remember that it is best if you first configure the panel size according to your target (iphone, nexus, galaxy, etc.) before you start placing controls. This is covered in the first tutorial. The default panel size is fairly standard for a modern mobile phone in landscape mode.

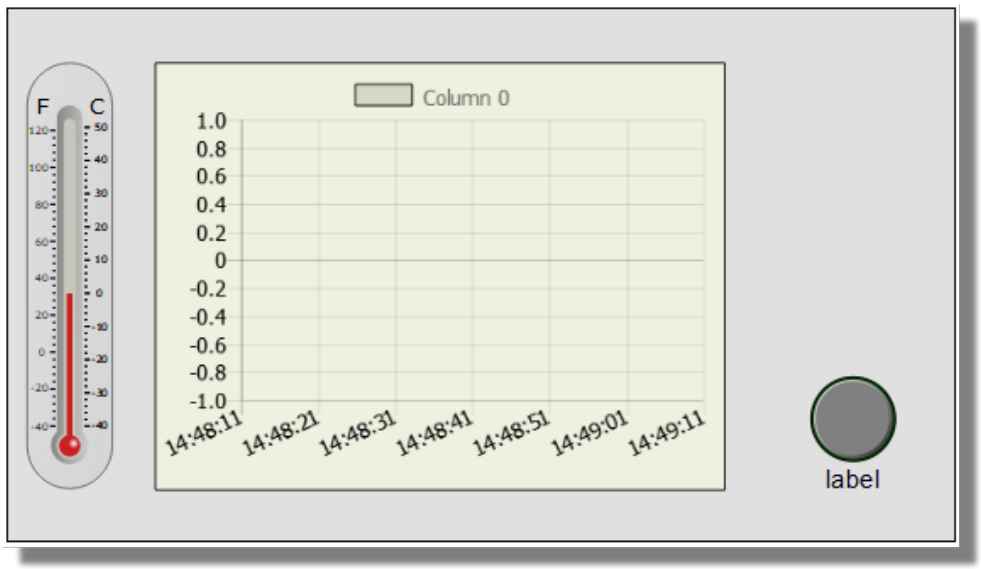
Drag and drop the thermometer onto the panel and position on the left hand side.



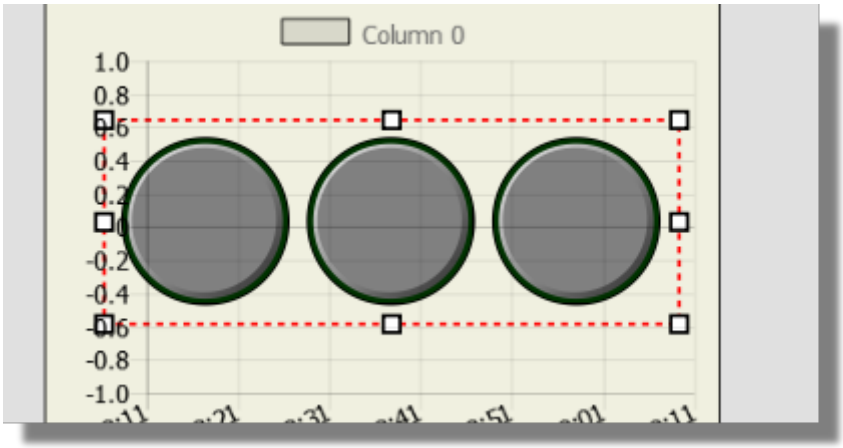
You might find that it is too big for the panel by default in which case simply use the drag handles to resize and then pick it up and re-position as required.



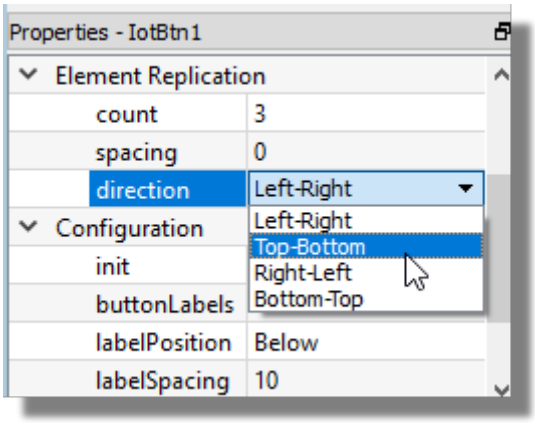
Repeat with the line chart and the reset button.



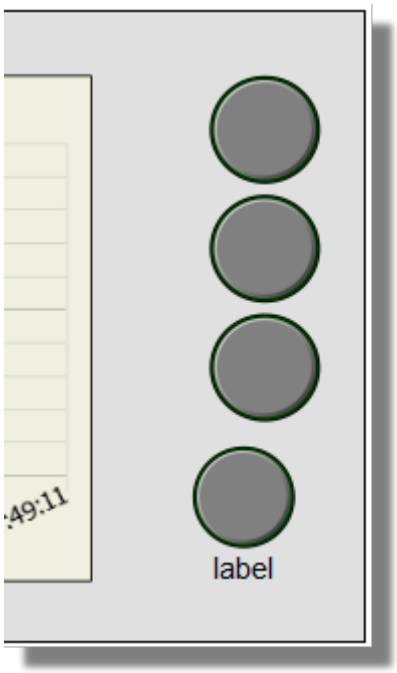
Now place the button group loosely on the panel - over the middle of the line chart is fine.



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

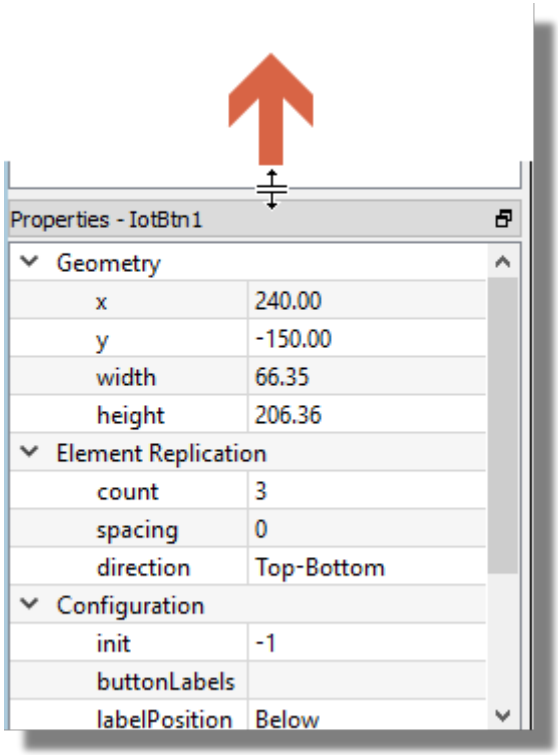


Now resize the button group to fit and then re-position above the reset button.

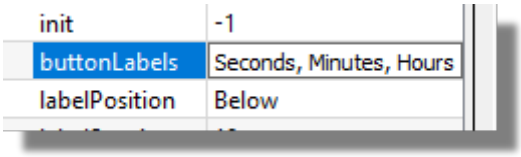


Properties and Alignment

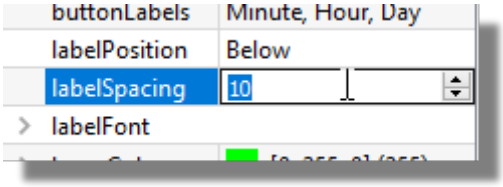
Unless you are very good at this, it's likely that your buttons are looking pretty ugly at this point. They may well be 'squashed' in one dimension and the button group is probably not exactly the same size as the reset button. That's to be expected at this stage and we'll adjust the sizing and placement 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.



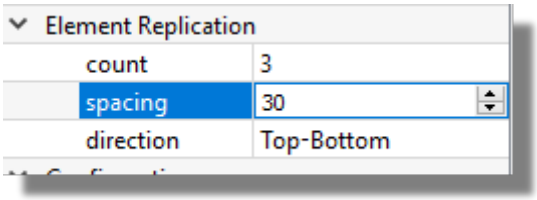
Scroll down to the configuration section and enter the labels as a comma separated list beside the buttonLabels property.



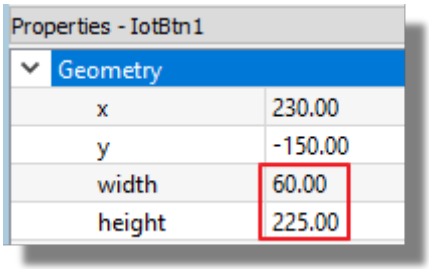
We'll need to include a little bit of label spacing (between the button and the text) so set this to 10



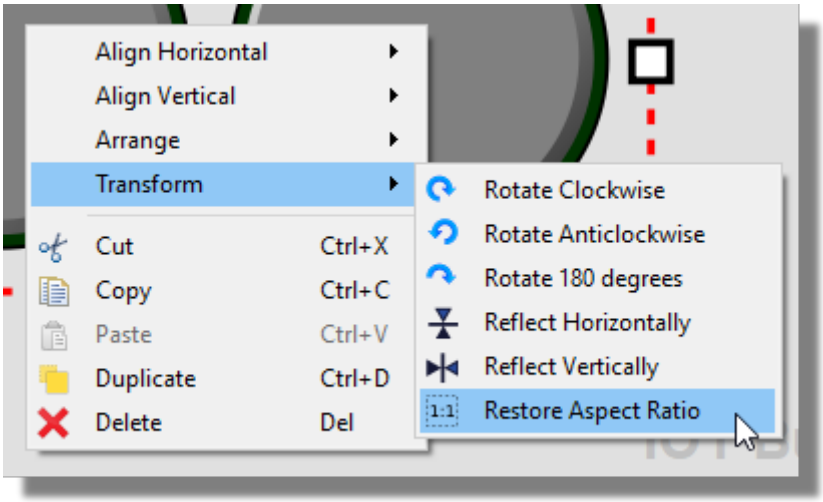
Next, we need to set the separation between the buttons so look for the spacing property under element replication and set this to 30. This is the distance from bottom of one button to top of the next button.



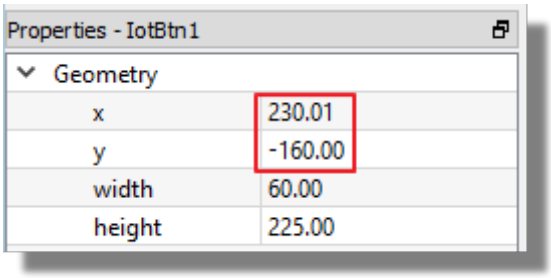
The size of the buttons needs some adjustment, you can resize them by dragging one of the corner nodes, or you can set the size manually. In this case we will set them to be width 60 and the height (of all three buttons) to be 225.



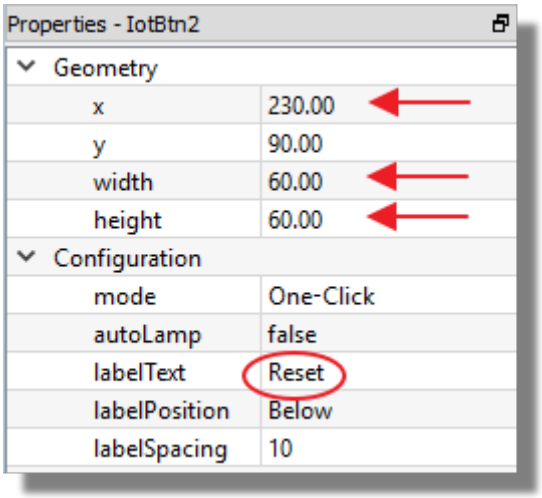
Note that if you have the buttons out of shape, you can right click on the buttons > Transform > Restore Aspect Ration



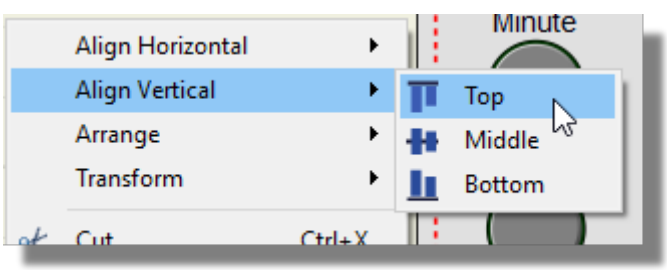
We need to move the button group into position manually or by adjusting the x and y values in the geometry section.



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.



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.

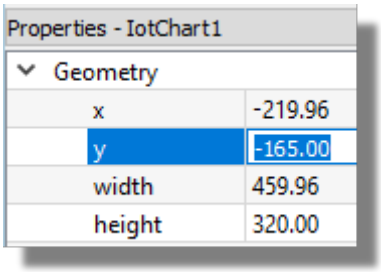
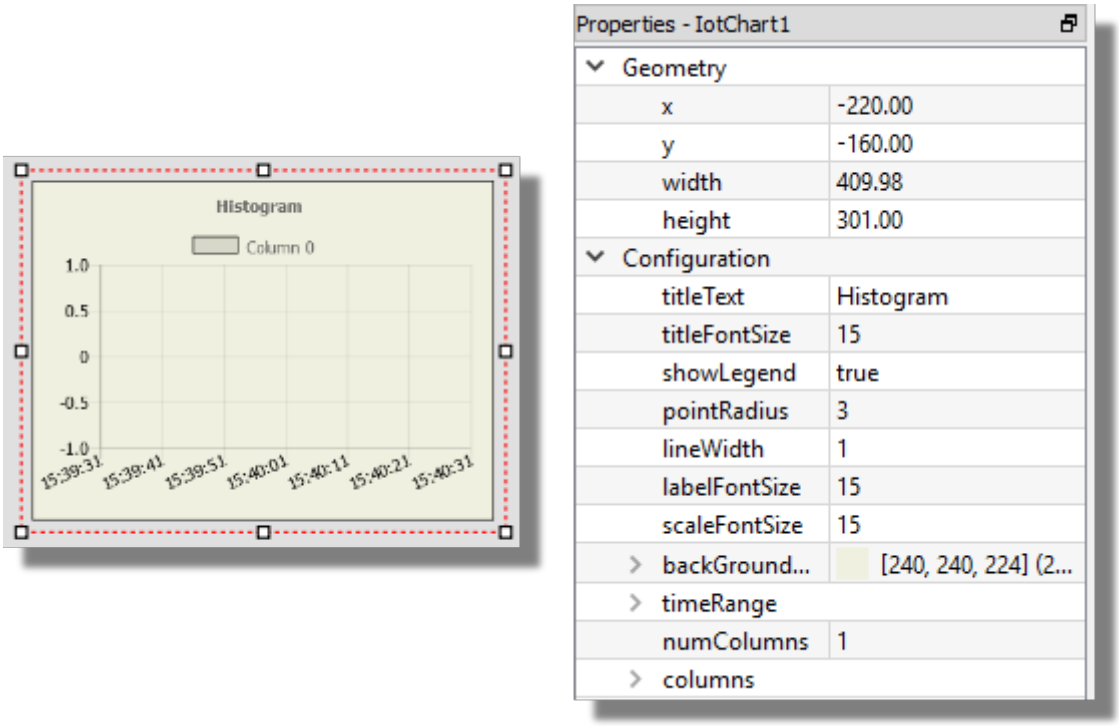
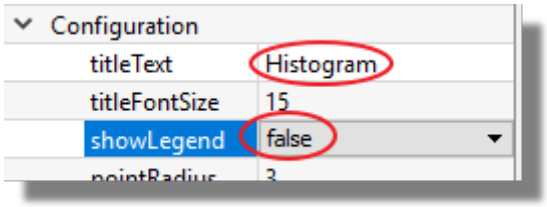


Chart Configuration

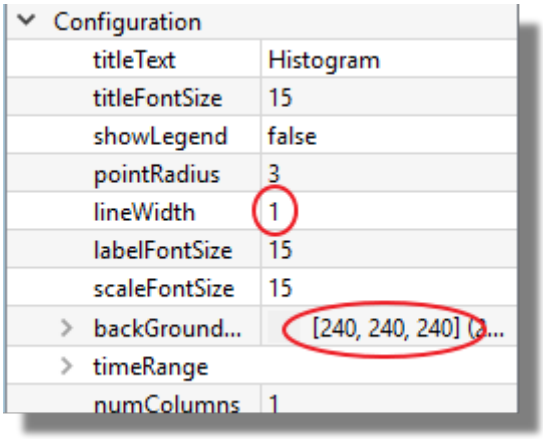
The line chart is a very flexible control and has several configuration options. In our case, we will changing many of these at run time (for example, we will be changing the time axis as the user presses a button to adjust the historic range). However, there are a few aesthetic things we can set up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



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.

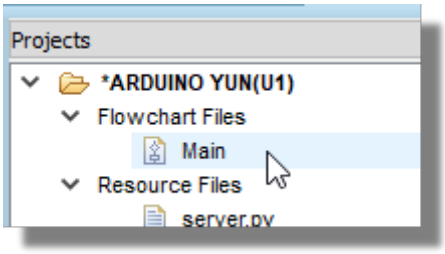


Writing the Firmware (Flowchart)

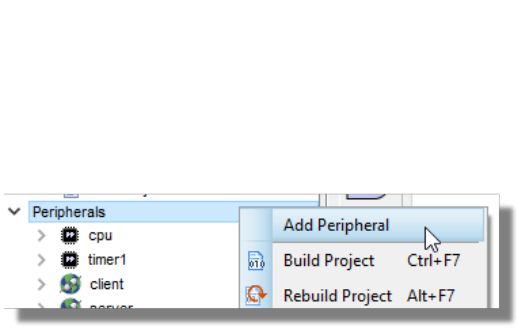
Now that we have our front panel in reasonable shape it's time to think about our program.

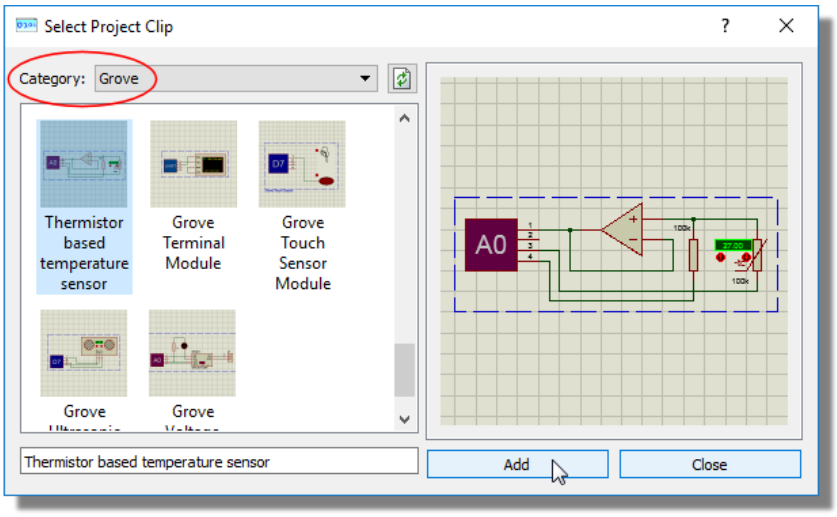
> We will skim quite quickly over the basics of flowchart design in this section. Please refer back to the Visual Designer help file (accessible via the Help menu in Visual Designer) if you need more detailed coverage of anything.

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



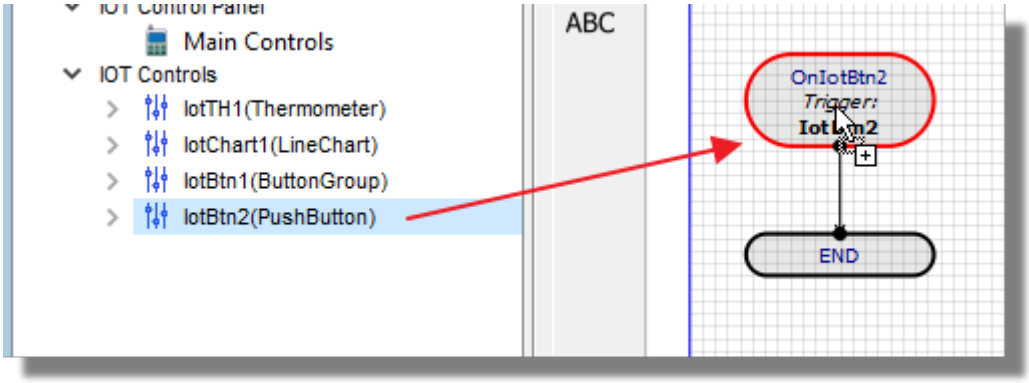
We need first of all to add the temperature sensor to our schematic (virtual hardware). We do this in the usual way via the Add Peripheral command in the project menu and add in the grove temperature sensor.



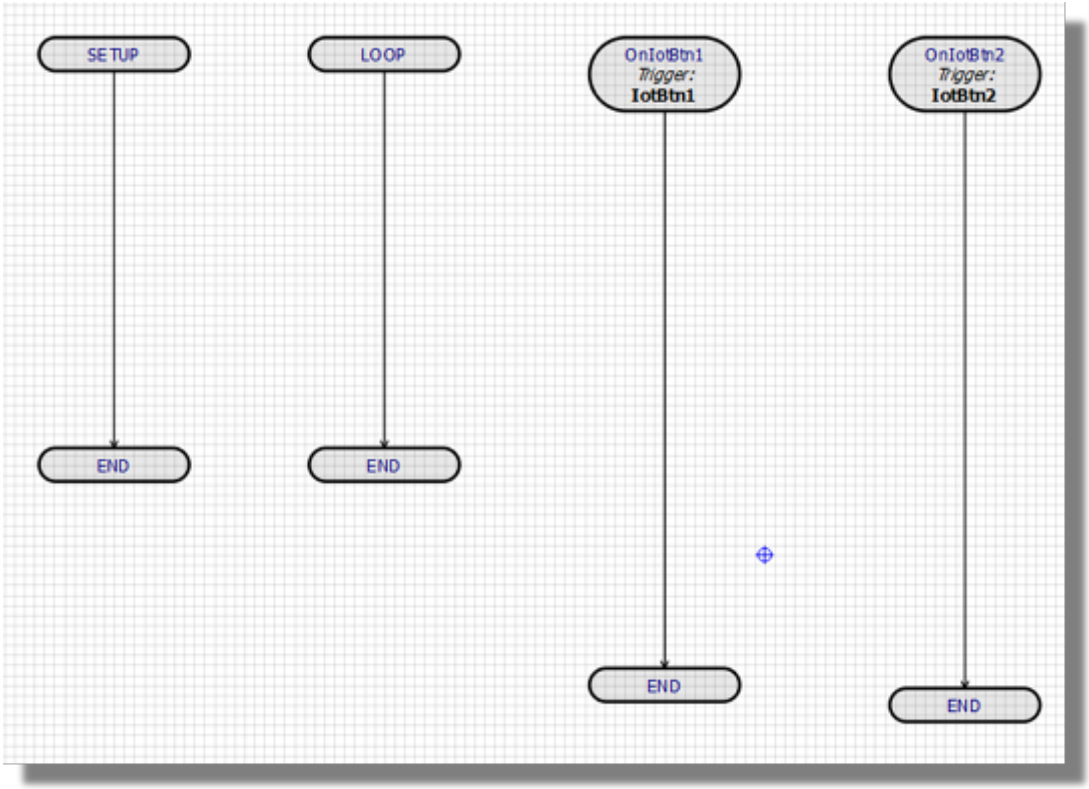


Program Structure

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.

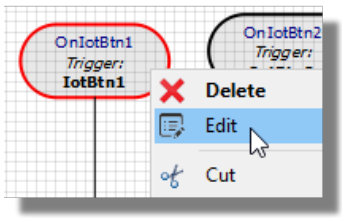


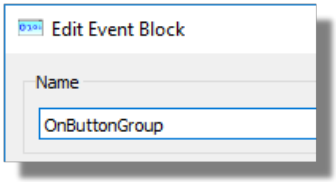
We need to do this both for the button group and for the individual (reset) button. With a little movement of chart elements around our basic program structure should now look like the following.

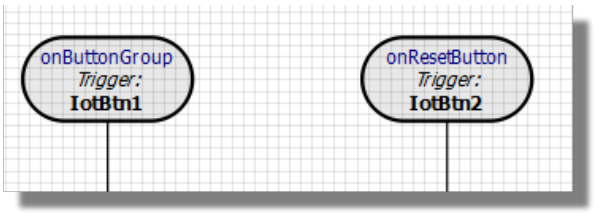


Basic routines for our program

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'

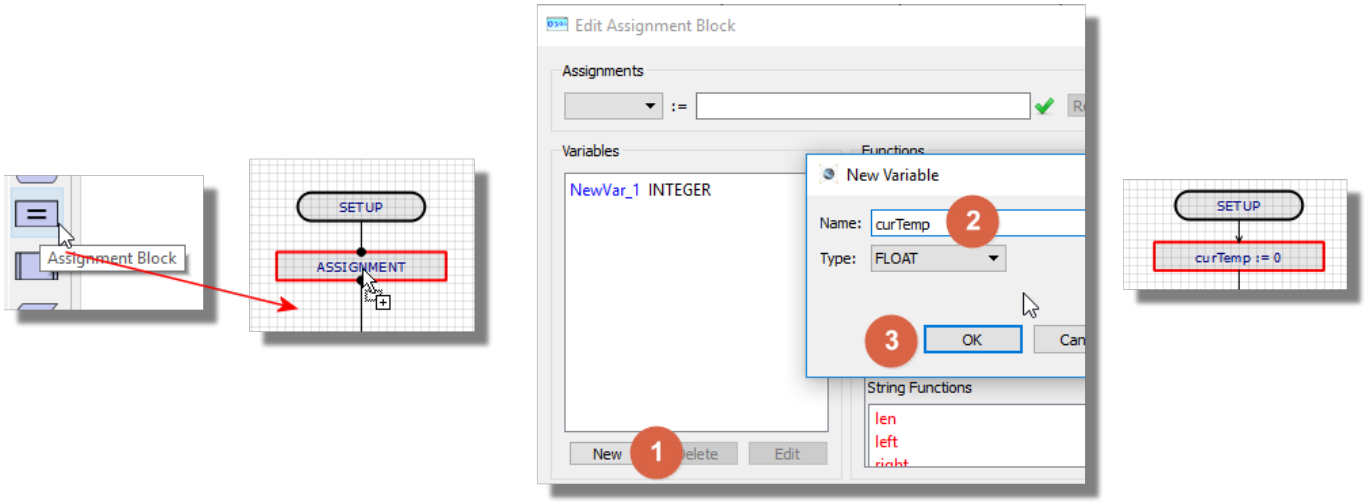




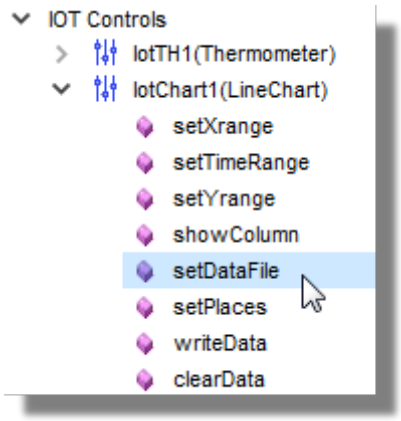


Setup

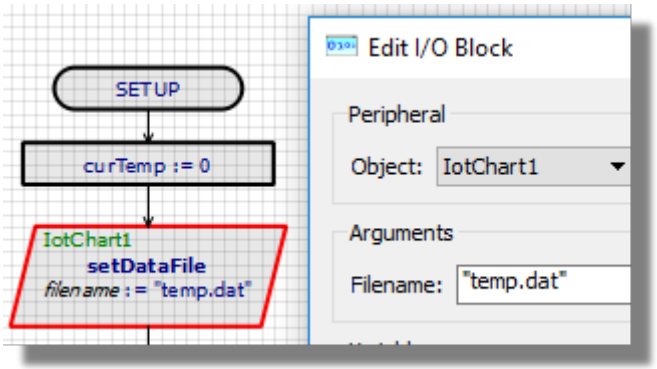
The setup routine is where we do any initialization tasks before the program begins running properly. Since we have a thermometer that will be showing the current temperature we will need a variable into which we will store the temperature. Drag an assignment block onto the setup routine, edit and create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

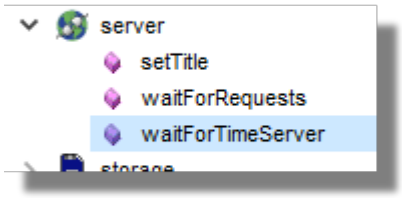


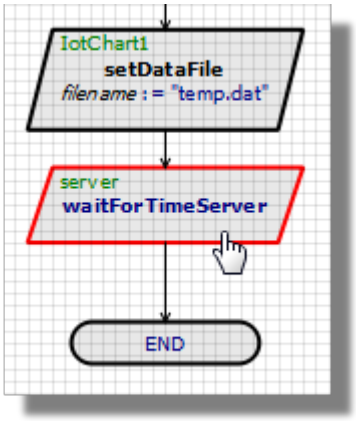
We also need to store our historical temperature data. For small data sets this can be done automatically and will store as session data on the Atheros but for larger data sets you can specify a filename and the data will then be stored to file, either on the Atheros or on the SD Card. We specify a data 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)



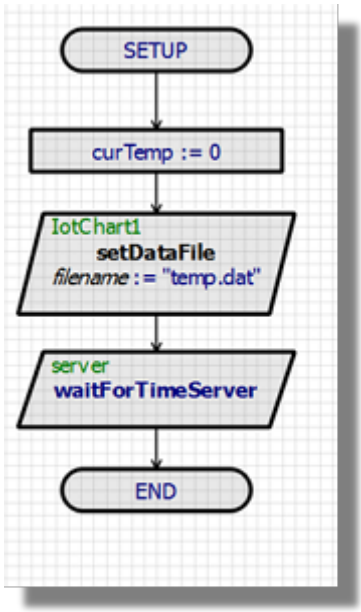




The timeserver is part of the Linux OS running on the Atheros which queries the current time from an internet server.

> ⚠ 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



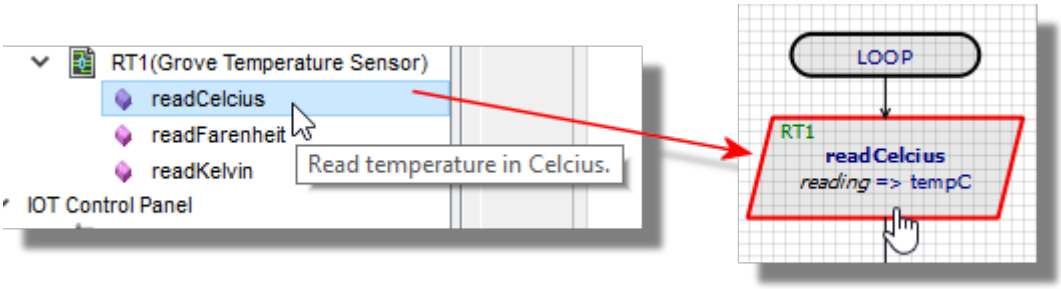
Setup routine for the logging thermometer.

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.



Farenheit or Celcius is a configurable parameter in the property panel in the IoT Builder for the Thermometer. We could easily switch the thermometer into Farenheit 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.

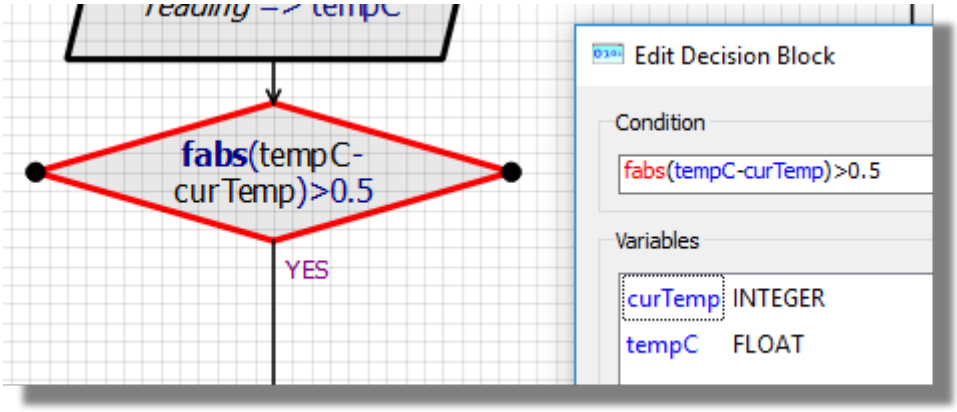
The reason we don't assign directly to the curTemp variable is time. Our loop routine is going to execute extremely fast which means if we don't think about what we are doing three things will happen.

1) Our log file will become enormous very quickly.

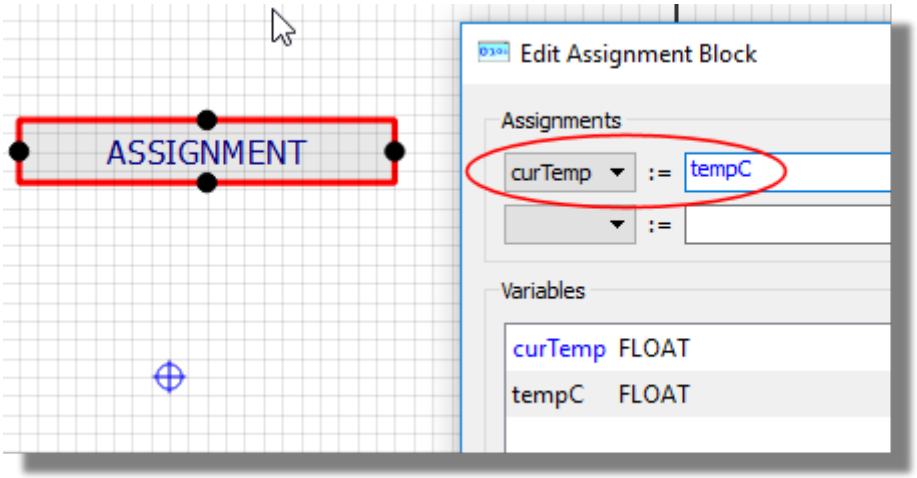
2) We'll be sending huge amounts of unnecessary traffic back to the controller front panel.

3) We'll be unresponsive to UI commands from the GUI because we are executing a tight loop at full speed.

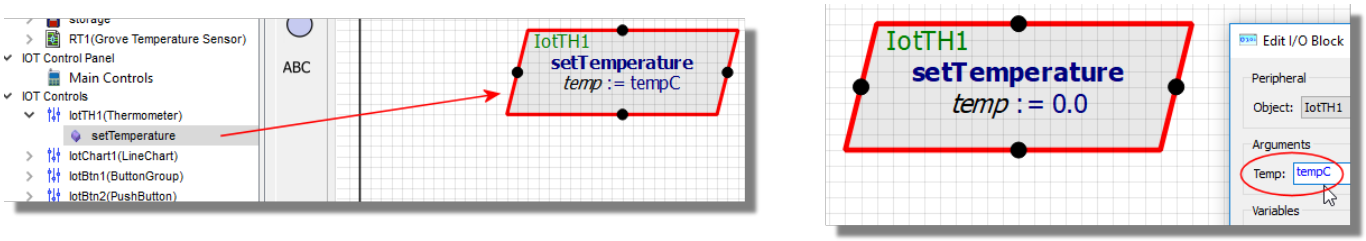
So, when do we need to update the front panel ? Clearly, when the temperature changes we want an update so we'll add a test for temperature change. We can therefore add a decision to the chart that checks whether the temperature difference is greater than half a degree since our last logged temperature.



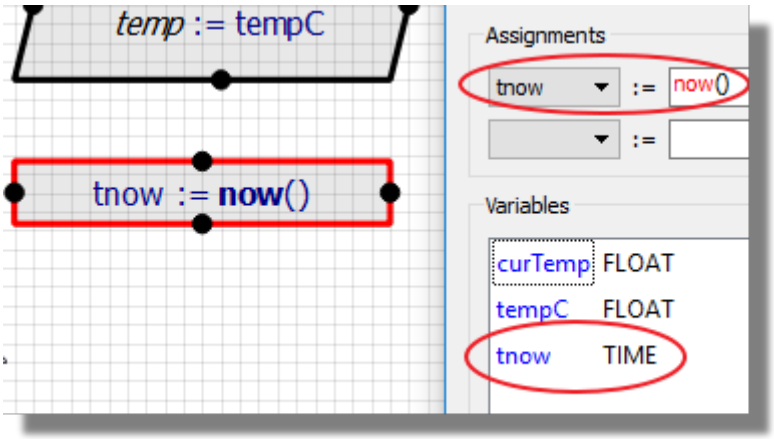
If the temperature has changed we need to assign the temperature reading (tempC) to our temperature variable (curTemp) in order that our test is valid the next time we execute the loop routine. Drag an assignment block next to the fabs decision block and set the following:



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

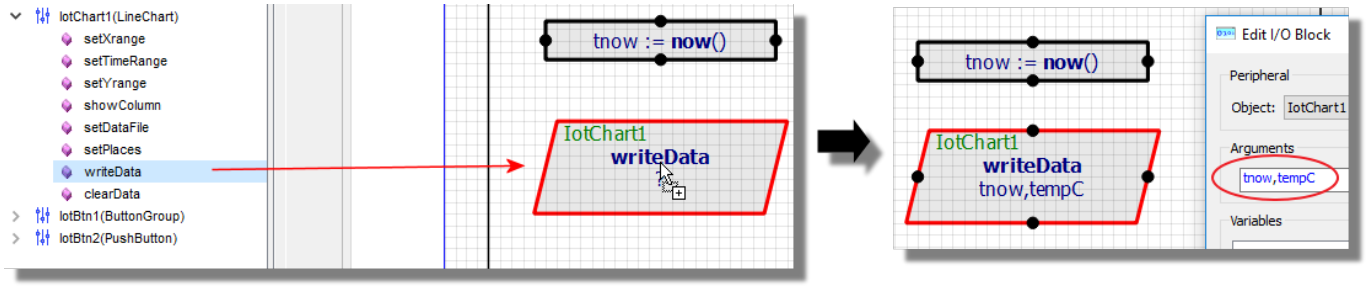


Finally, we need to log the temperature change. In order to do this we need to know the current time, otherwise we can't log a data point properly. We can get the time via the now() method and store it into a new variable (tnow) using a standard assignment block.



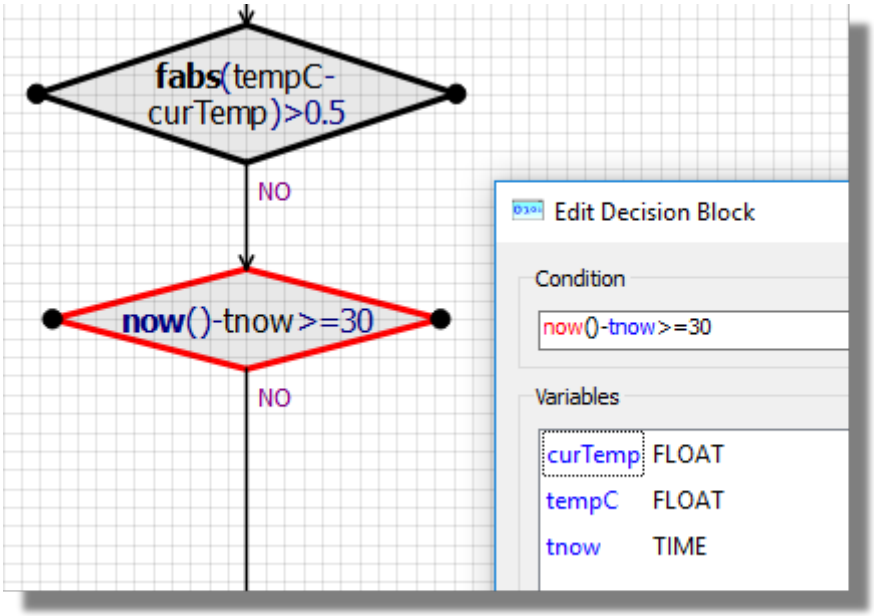
> The now() method is simply a Visual Designer call on the timeserver that returns the time in secs since the EPOC in local time. It therefore enables the AVR side not to worry about time.

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

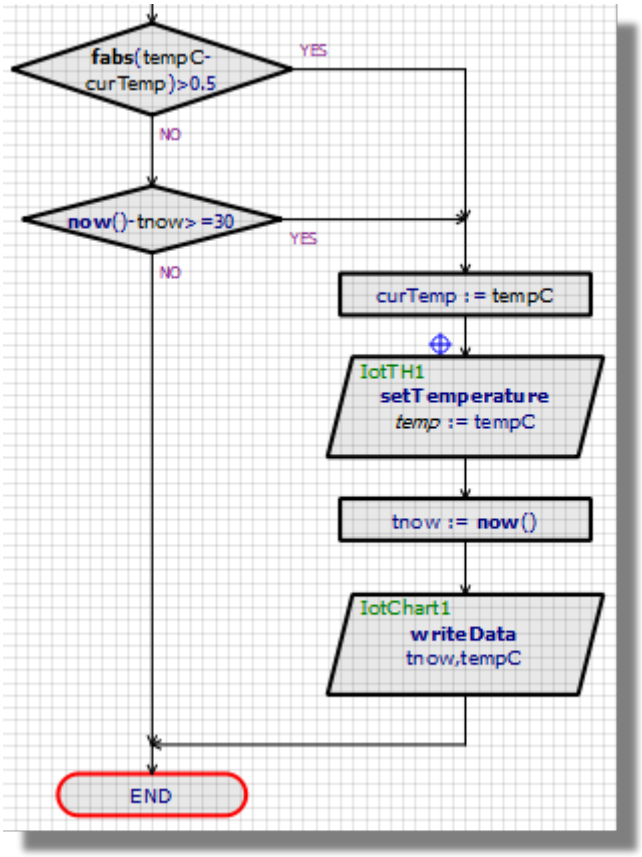


> The writeData() method will both log the data point to file and update the chart with the current data point.

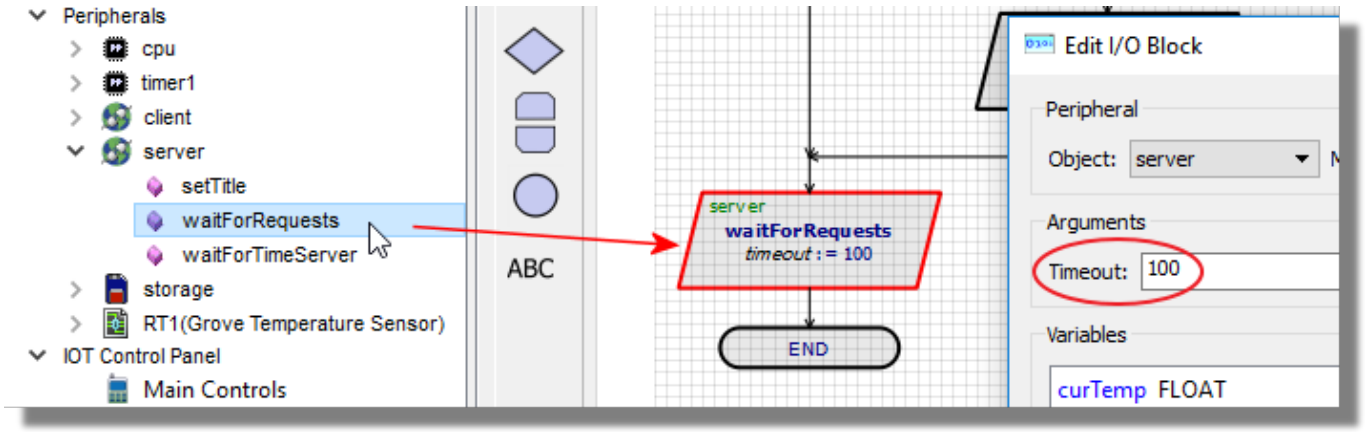
Now, if the temperature doesn't change we'll still want to log data points periodically. Let's say we write every 30 seconds or so if the temperature stays the same. We can do this by adding a simple test to the 'no' branch of our temperature check test. (again you will need to swap the yes/no.)



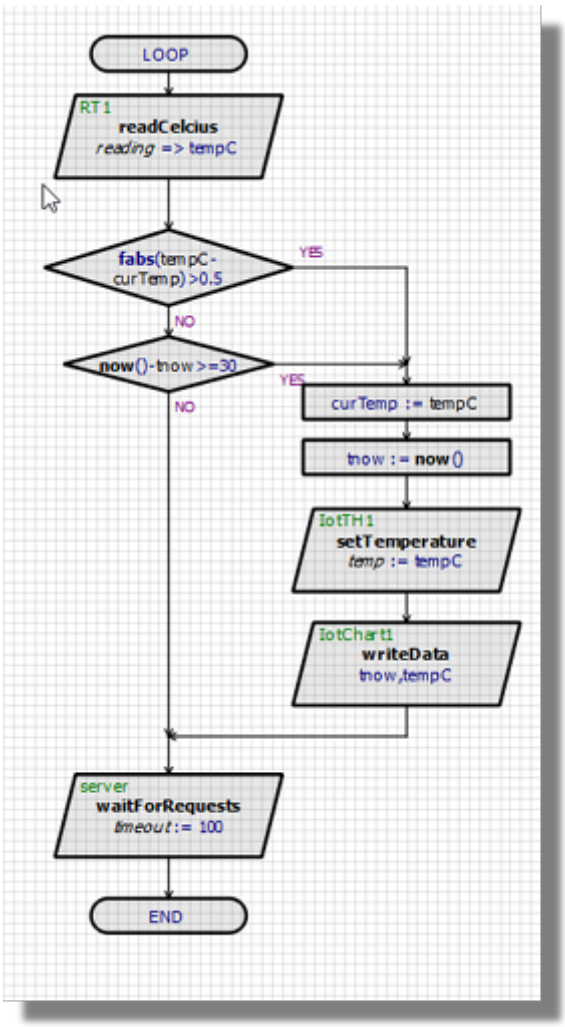
Connect up to the 'action branch', if the result of our time test is positive so that we log if 30 seconds or more has passed since our last log. This is what you should have so far:



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.



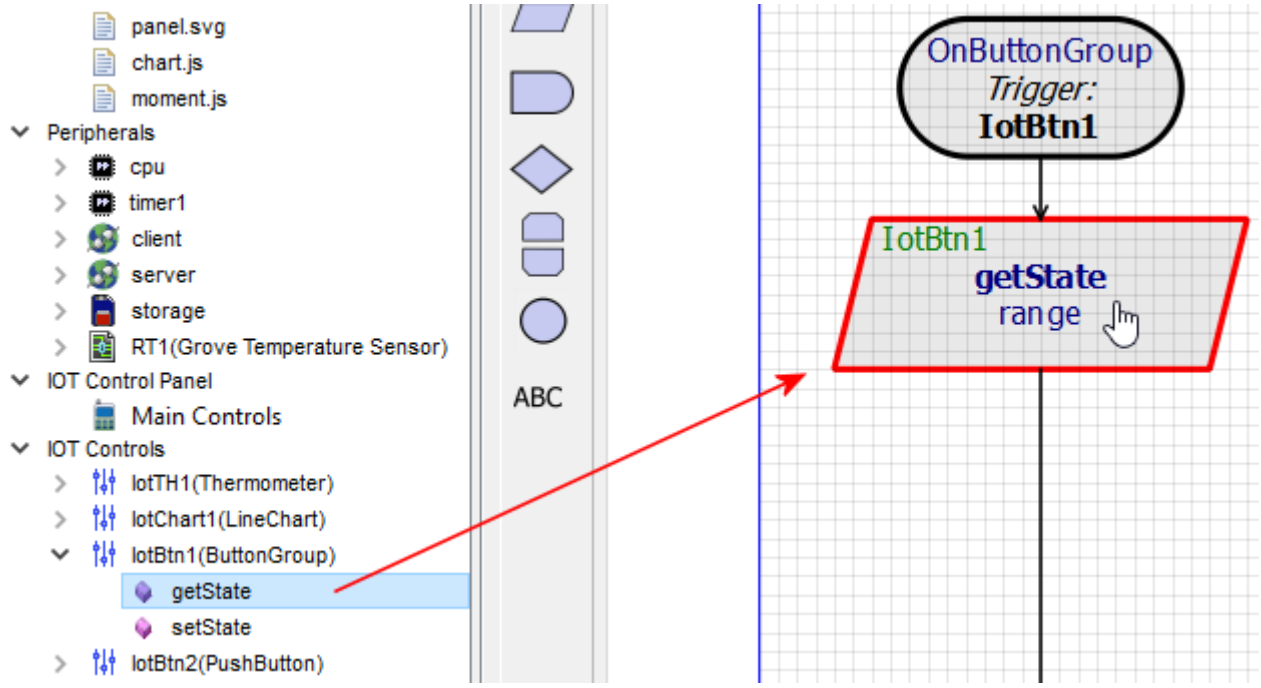
Main execution loop for the logging thermometer.

> ⚠ 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

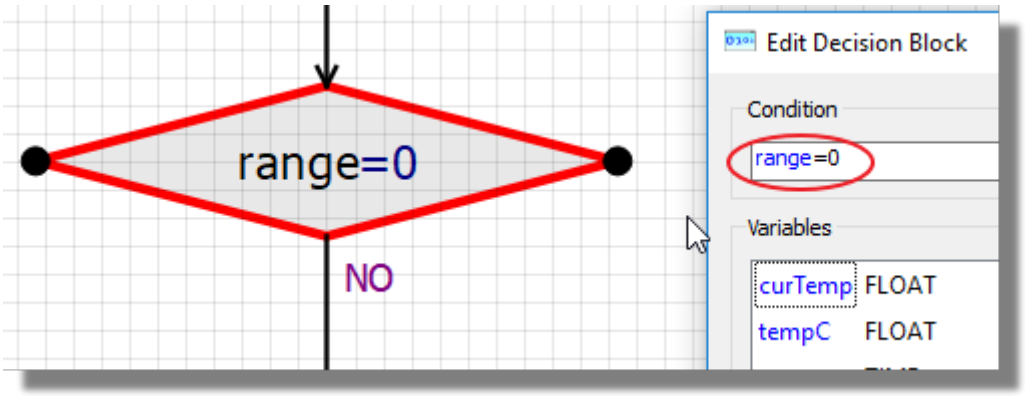
We've now got our program logic set up so the next question is how do we respond to a UI request from the controller. If we first consider the button group then we know that we'll be arriving at the top of our onButtonGroup() routine whenever a button in the group is pressed. So, we need to first determine 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.

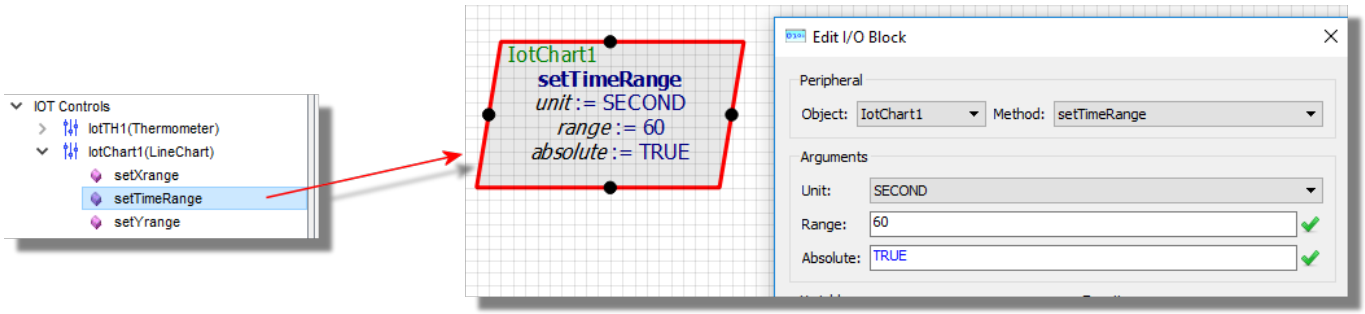


GetState() on a button group returns a zero based index according to the buttons position in the group. The first button is index 0 (Seconds in our case), the second is 1 (Minute) and the third is 2 (Hour). It is set up this way so that a button group can be extended as much or as little as the user 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.



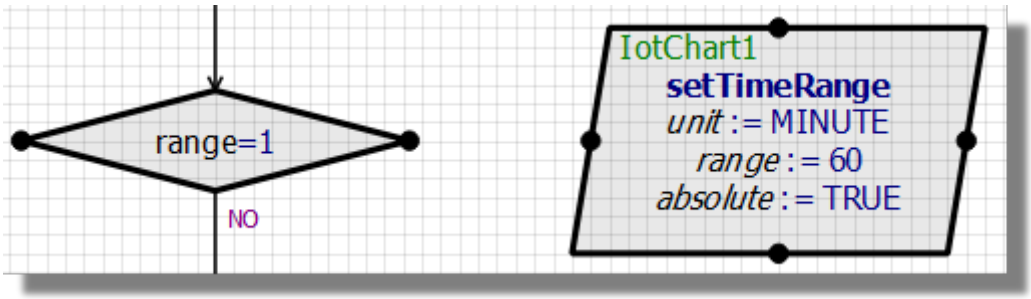
To set the x-axis to span one minute we are specifying a time range so drag and drop the setTimeRange() method from the line chart IoT Control.



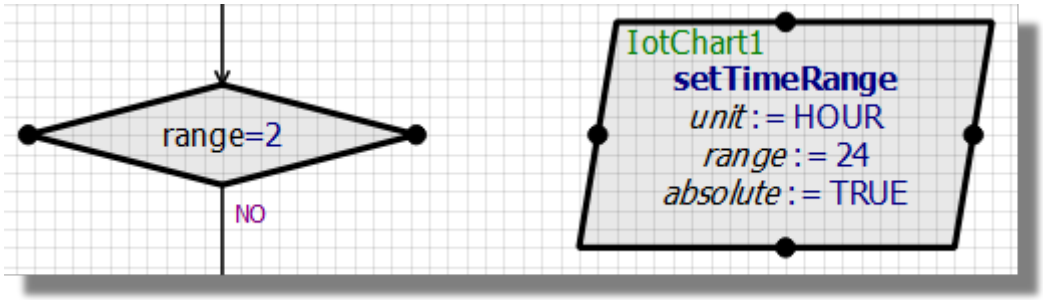
The configuration is then to set the units to seconds and the range to 60 (60 secs = 1 minute). We also want the range to be absolute as above. The time range specifies the amount of time displayed, going backwards from the latest timepoint in the data.

If Absolute=FALSE then the time data is relative.

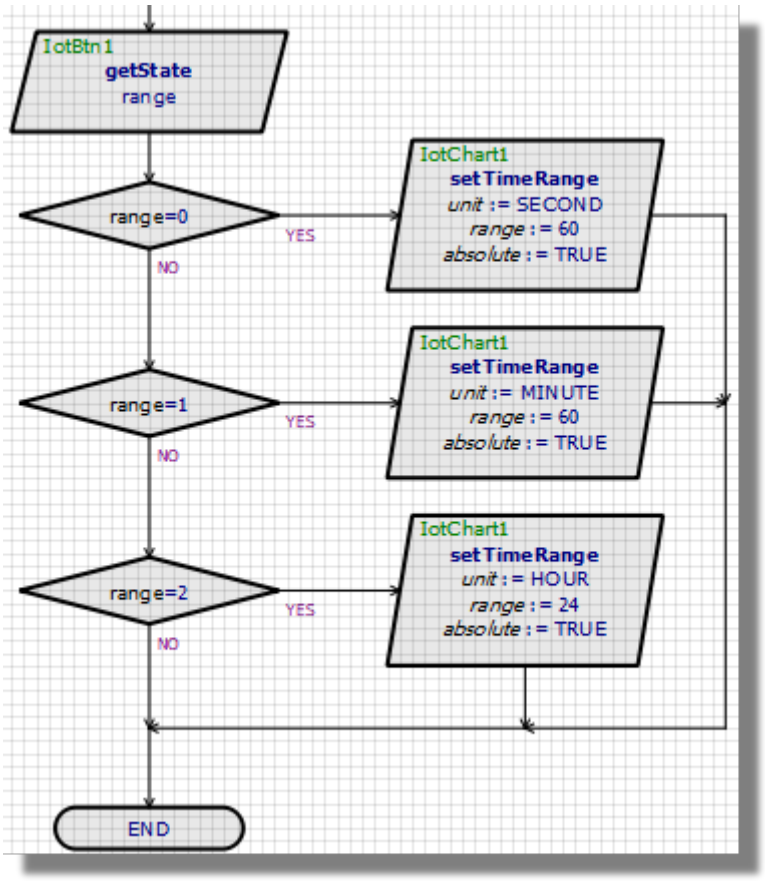
Now, we rinse and repeat with our test for the minute button press.



And finally for the hour range test.

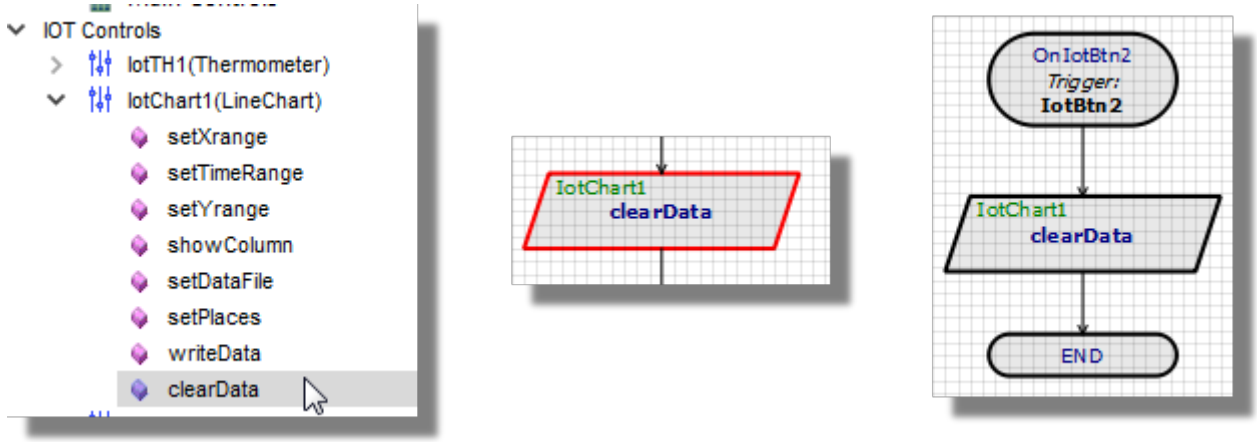


The button group handler should look something like the following when you are finished.



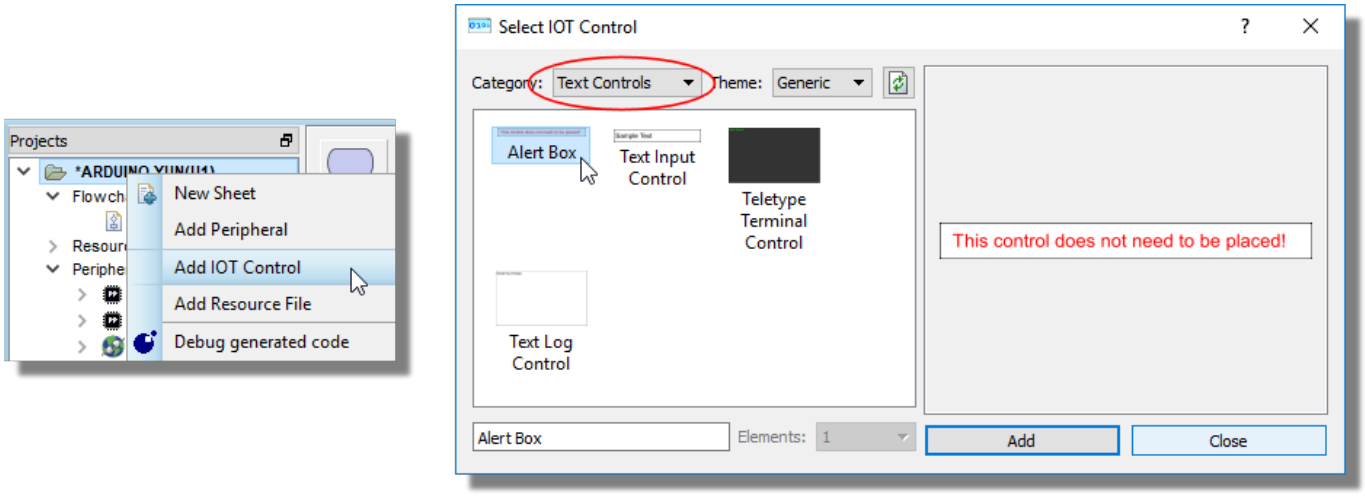
Reset Button Event

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.

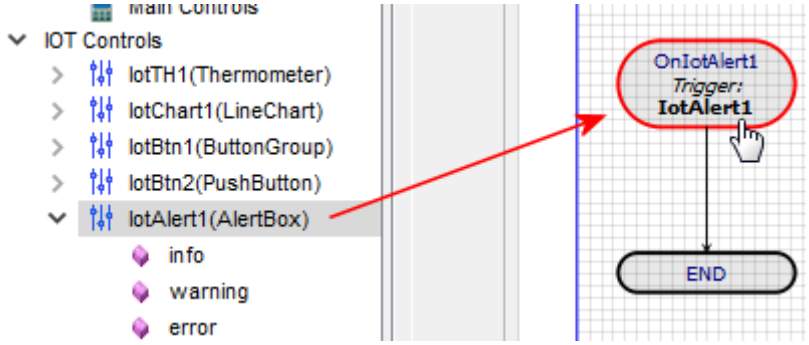


Simple method for handling the reset event.

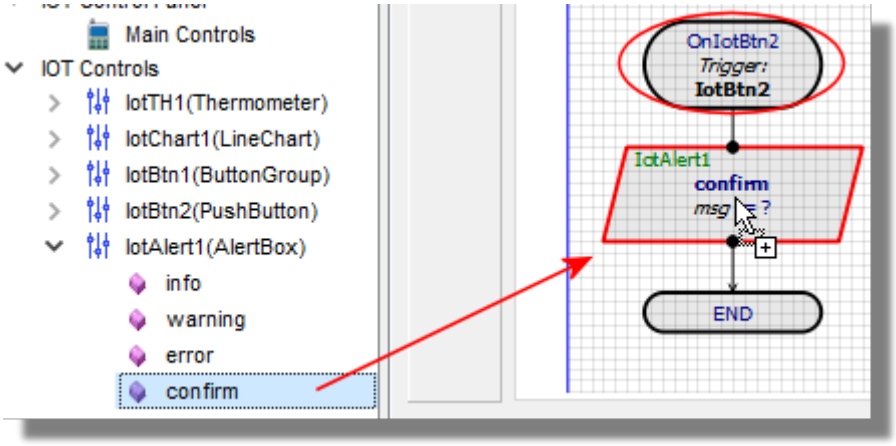
Since we are wiping the entire graph history at this point it may be better to prompt the user to confirm the action before we do this. Fortunately, IOT Builder can provide a way to do this by using the AlertBox control. To use this control, first pick it from the text area of the IoT controls dialogue



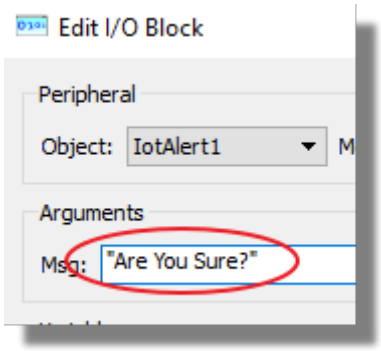
Unlike other controls this one doesn't need to be placed on the panel. We can switch directly to the flowchart and drag the Alert control on to the flowchart and move to a required location:



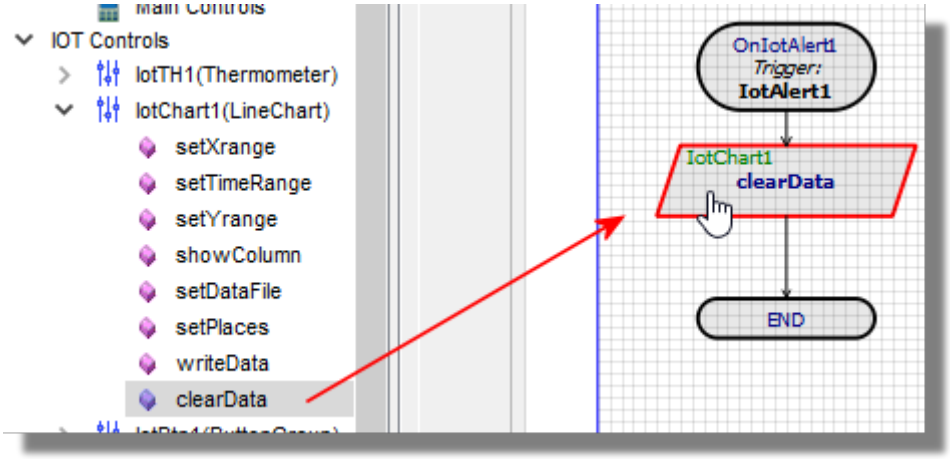
Now drag the 'confirm' method onto the reset button event handler.



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.

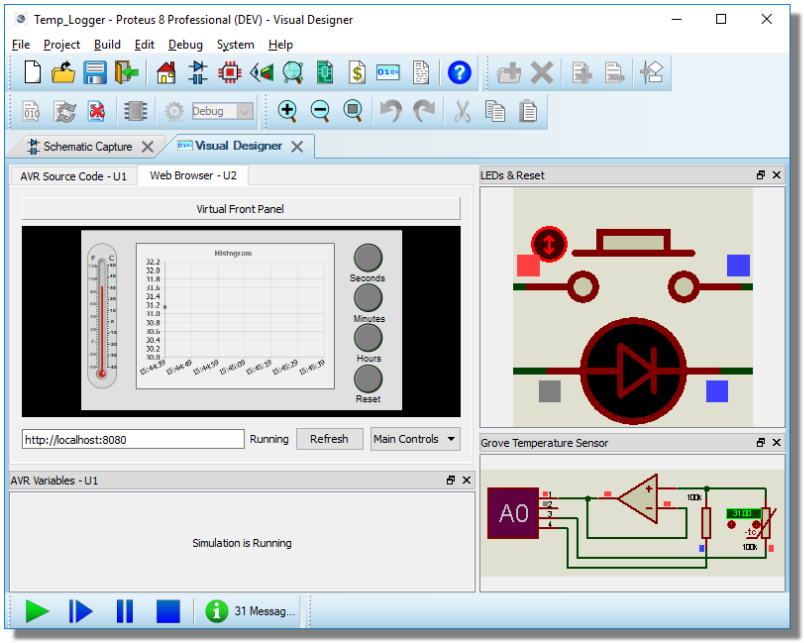
- Your front panel actions (switches, buttons, sliders, etc.) are processed in the flowchart as event routines. Drag and drop from the IoT Control onto the flowchart to place the event routine and then add the required blocks inside it.
- Think about time when it comes to your loop routine. How quickly 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.

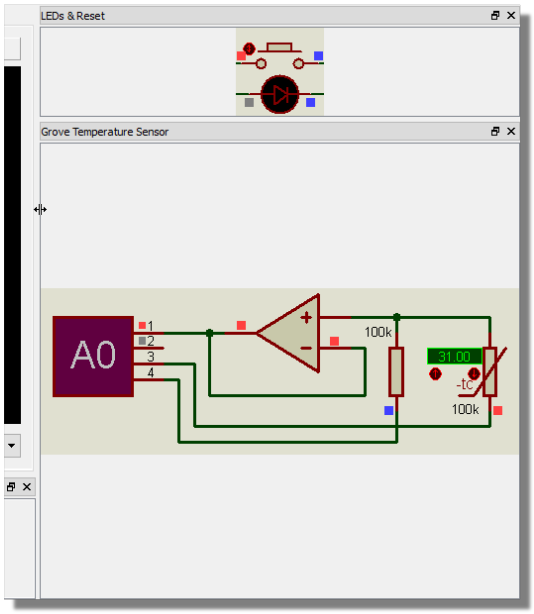
Simulation

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.

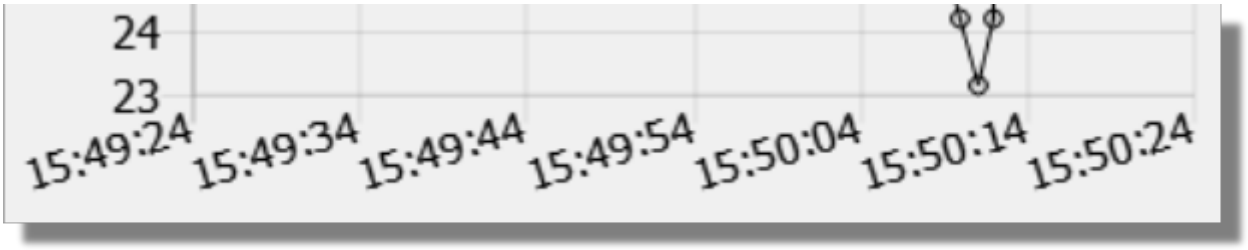


You should end up with the front panel in the main section of the window with the reset button on the Yun at the top and our temperature sensor at the bottom. You can resize these windows as required - in our case it would make sense to make the temp sensor larger so that we can see the readout.



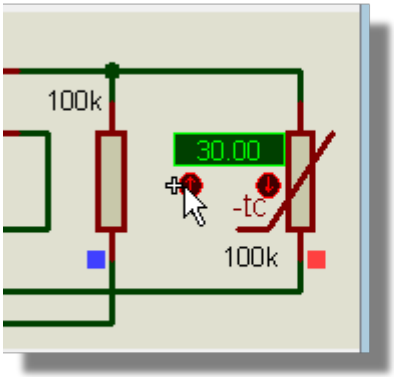


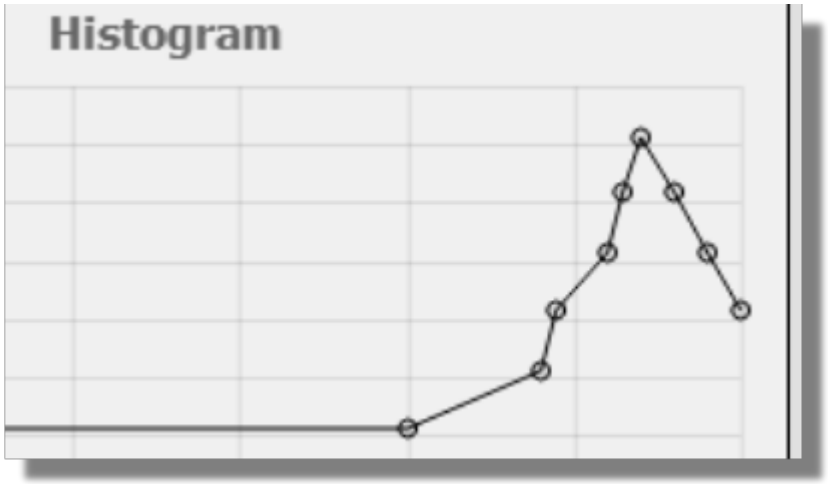
You should notice that the time period on the x-axis matches the default in the button group and that it actually showing the correct time.



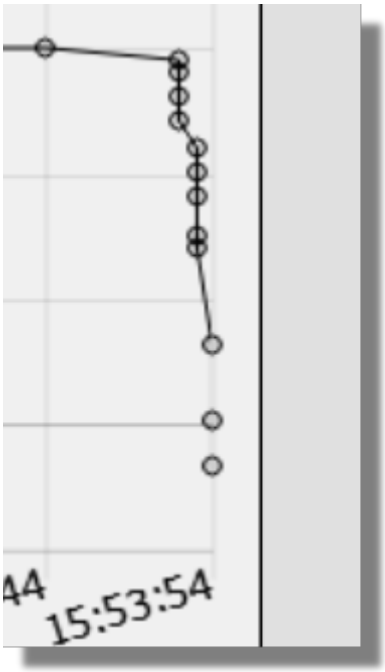
You should also notice that the graph time range advances every minute in real time.

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.



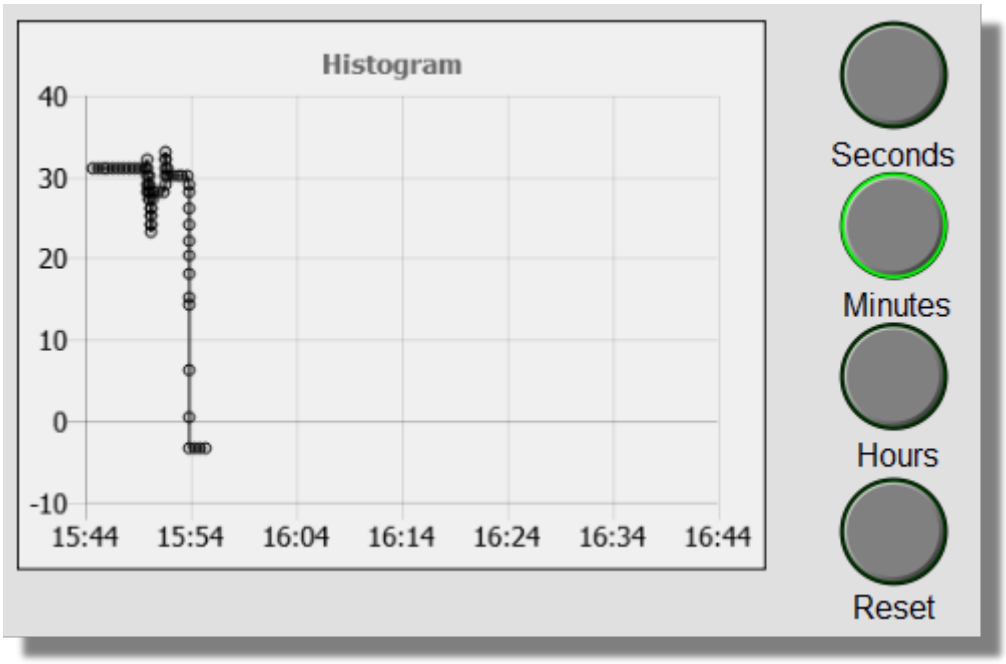


Similarly, we can reduce the temperature on the temp sensor and check our display. We can hold the mouse on the down button to move the temperature down rapidly and examine how responsive our chart and thermometer are.

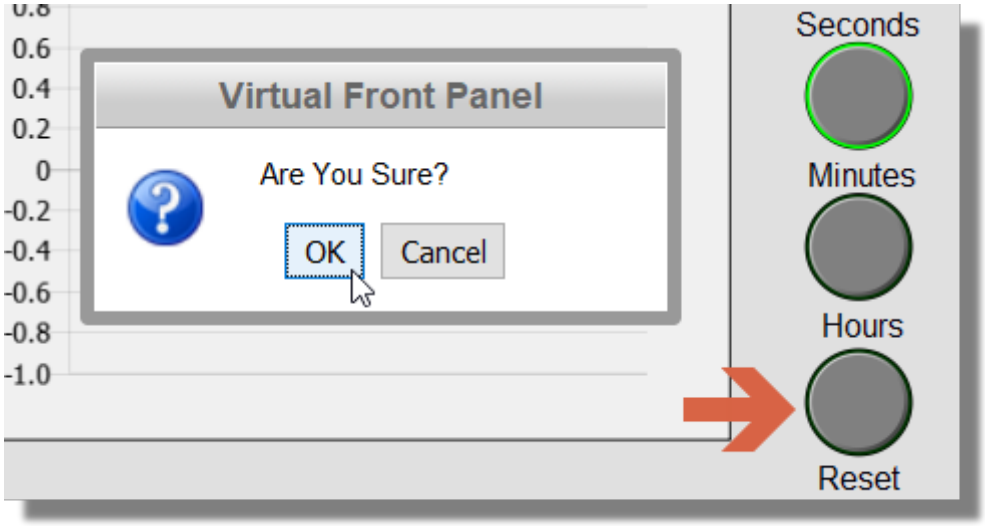


Compare how easy it is to bounds test a temp sensor compared to the real world (hair dryer and fridge ?).

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.

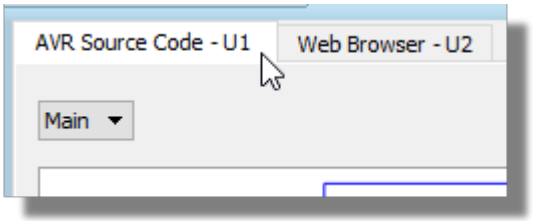


If you want to at this stage you can also check how the front panel looks and performs from your mobile/tablet device. This is discussed in the first tutorial here.

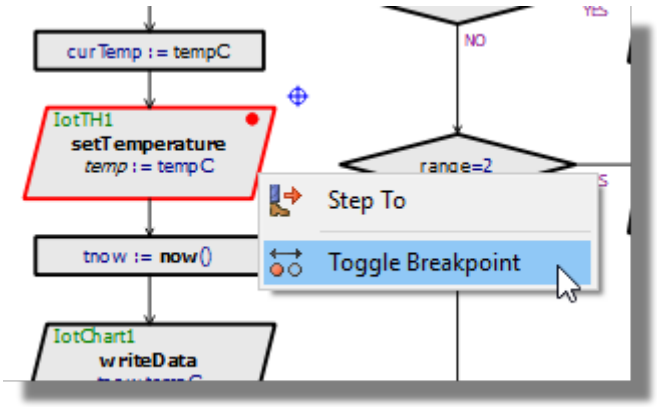
Debug

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.

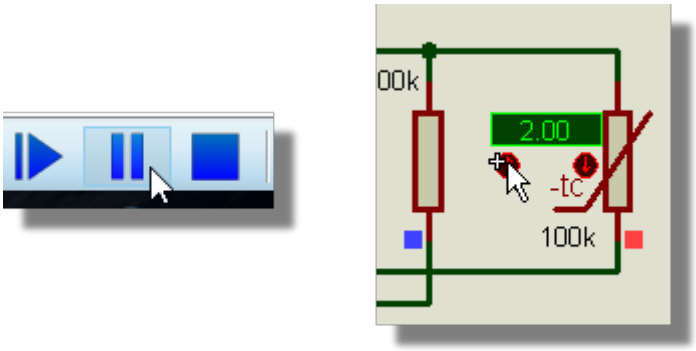




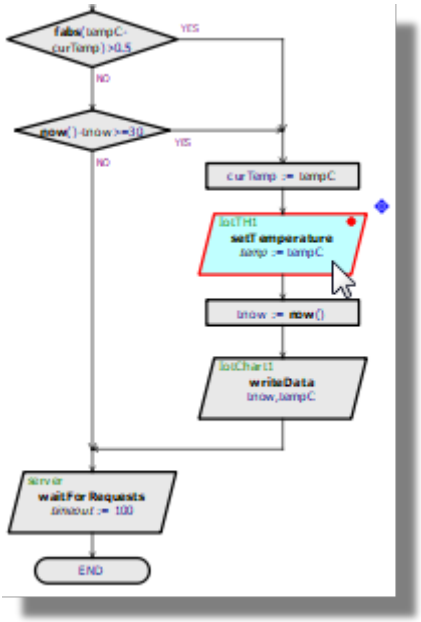
Set a breakpoint on the setTemperature() method in the loop routine.



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

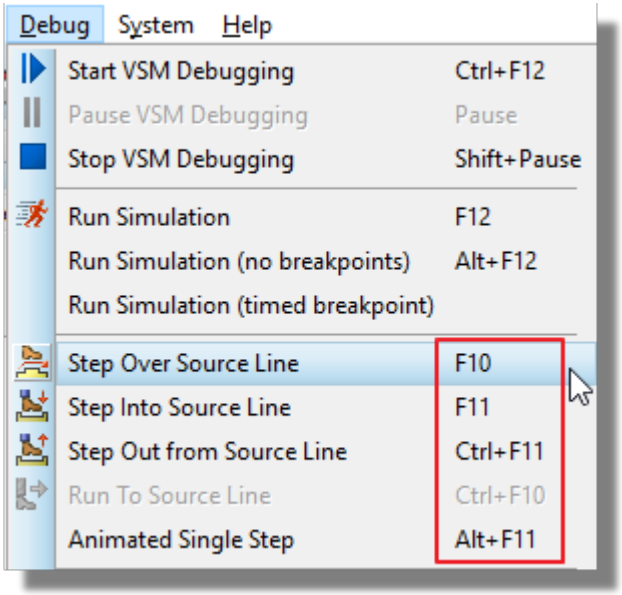


You should find that the program automatically halts and presents you with the source code window with code execution at the breakpoint.

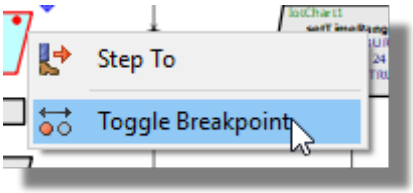


You can now single step your program using the icons at the top, or the keyboard shortcuts shown in the debug menu.

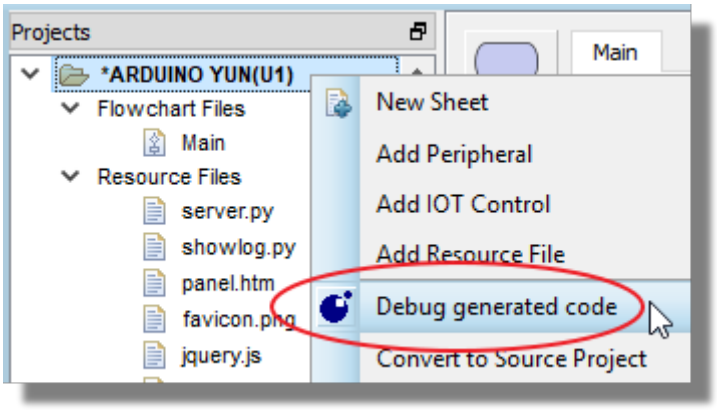




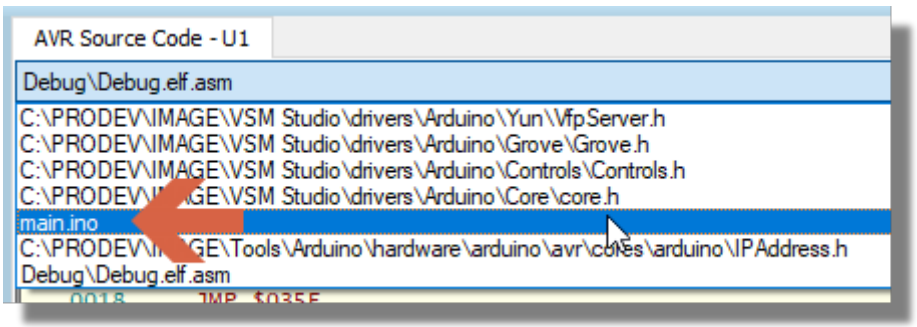
Finally, you can toggle the breakpoint off and stop the simulation when you are finished.



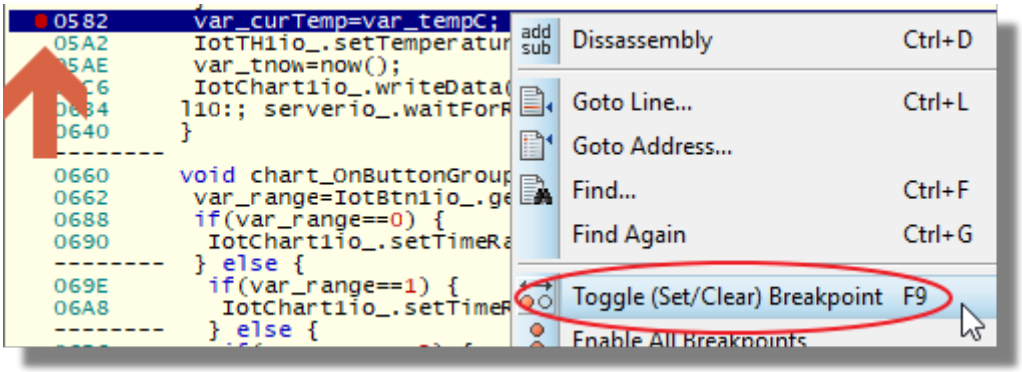
As well as debugging the flowchart it is also possible to debug at source code level. When the simulation is stopped, right click on the Arduino Yun in the project tree and select the debug generated source command from the context menu.



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



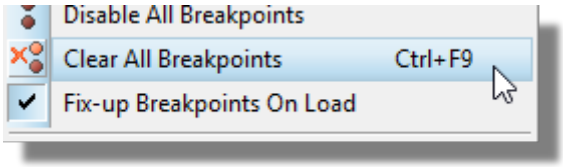
Set our breakpoint in the equivalent source code line, namely var_curTemp = var_tempC;



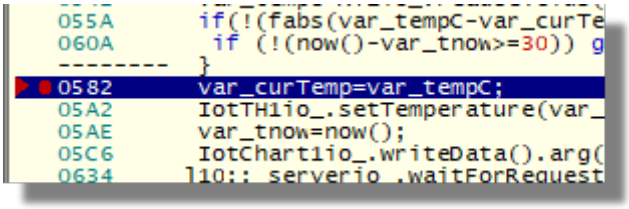
Run the simulation, wait a few seconds for the front panel to load and adjust the temperature sensor. We should pause at our breakpoint and single step exactly as before.

> If you want to 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.

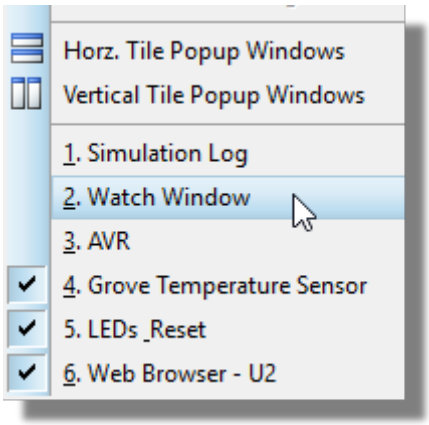


You can return to flowchart debugging by selecting the debug generated source command on the project menu context menu again (it is a toggle).

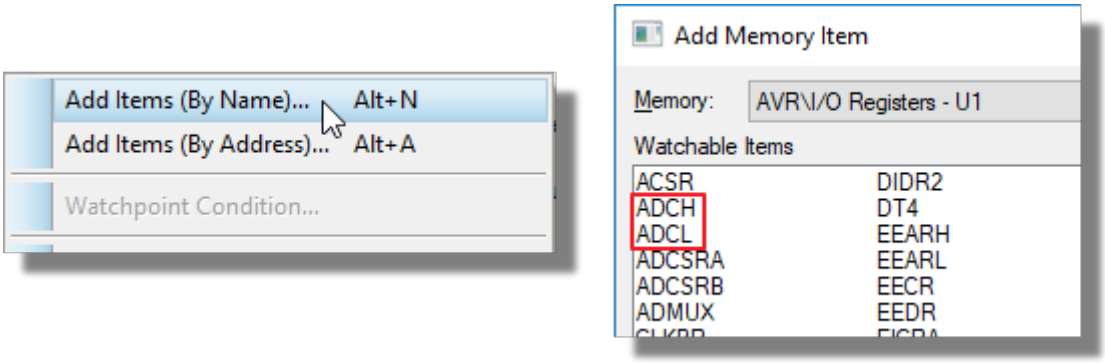


Advanced Debugging

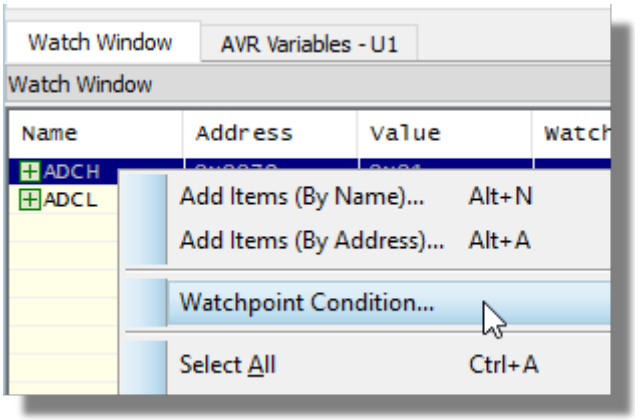
If you are more familiar with the AVR processor and/or the electronics on the schematic then there are a couple of other useful ways to control the simulation. The first is to make use of the watch window to control program flow. The first step is to pause the simulation and then open the watch window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



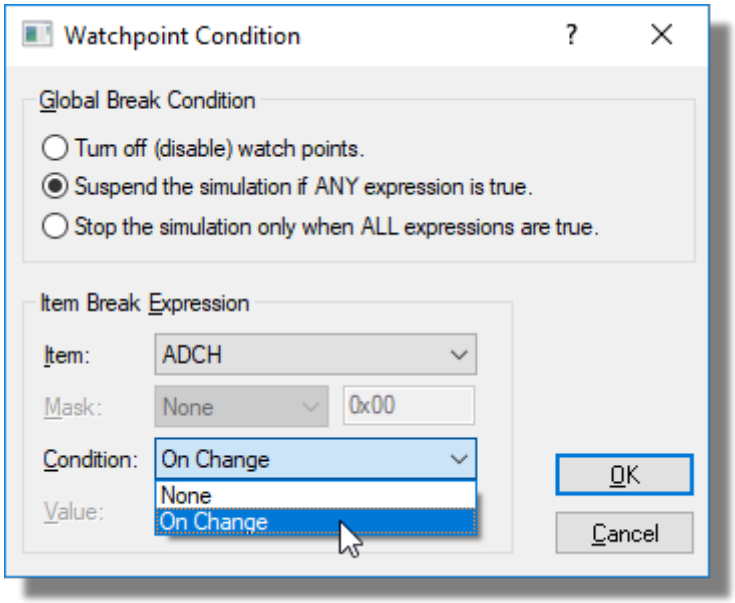
In our case we want to look at the ADC results because the grove temp sensor is connected to the ADC on the AVR. Right click on the watch window and select the add by name command from the resulting context menu.



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

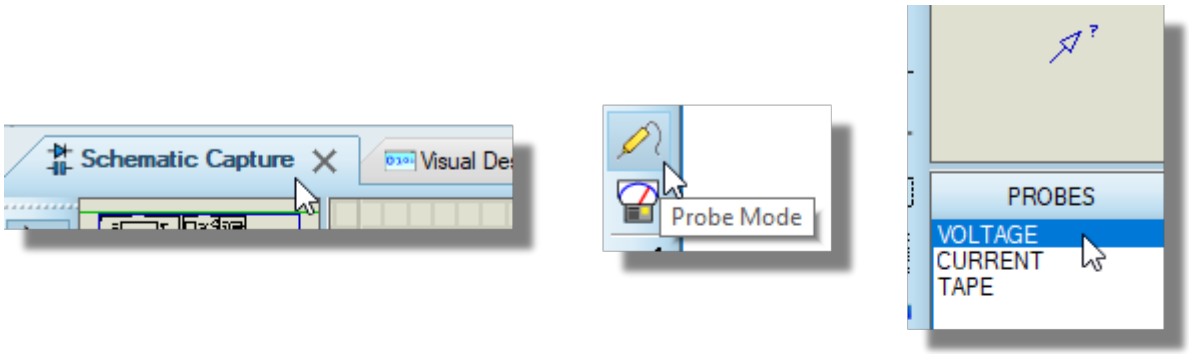
C:\PRODEV\IMAGE\VSM Studio\drivers\Arduino\Grove\Grove.h

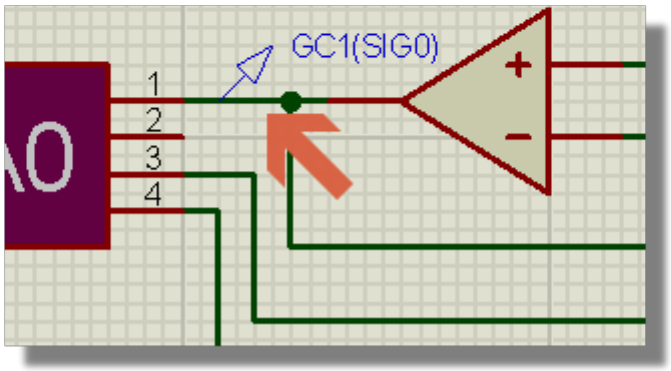
```
--------
-------- class GroveTemperatureSensor
--------  { public:
0368            GroveTemperatureSensor (uint8_t id) { pin = id; pinMode(analogInputToDigitalPin(pin),INPUT); }
--------        bool operator()(float trigger) { return readCelcius() >= trigger; }
037C            float readCelcius()  { return readKelvin()-273.15; }
--------        float readFarenheit() { 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:
--------        uint8_t pin;
--------  };
```

See Also:

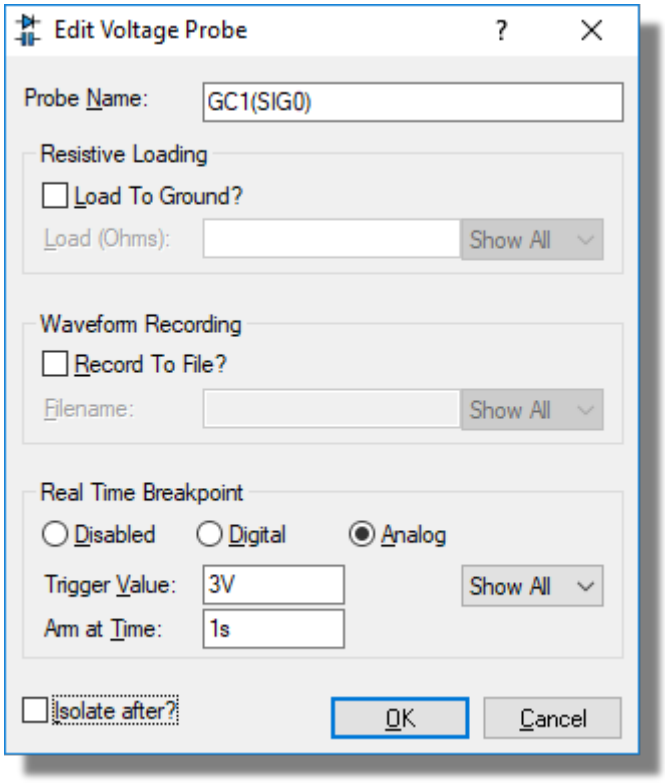
The Watch Window

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.

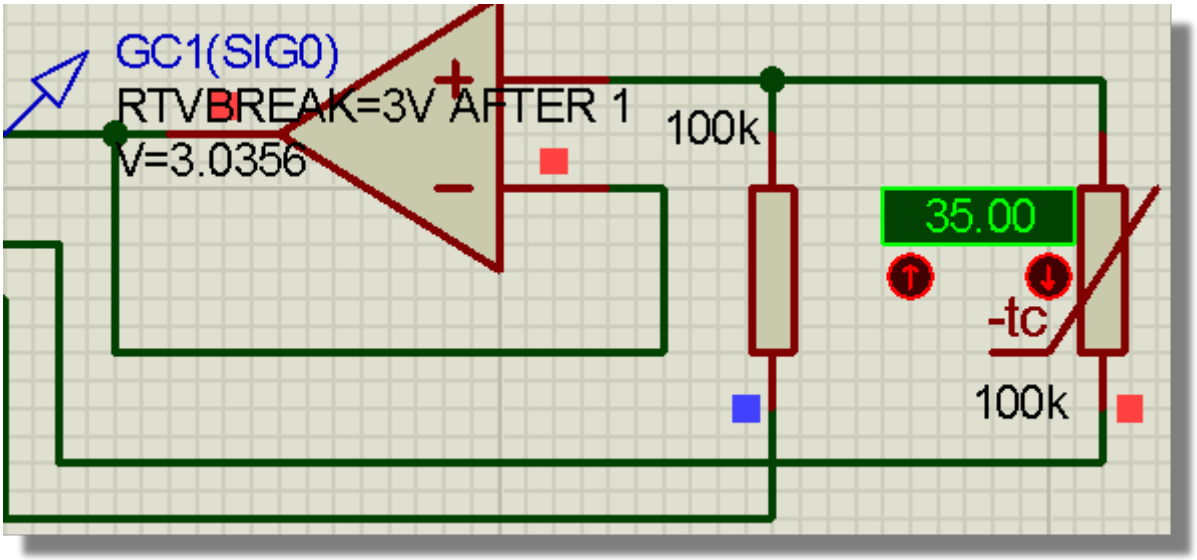




Next, edit the voltage probe and set the breakpoint to be an analogue breakpoint triggering at 3V with arm time of 1 second (to miss any initialisation noise).



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.



See Also:

- Hardware Breakpoints

> This last couple of debugging examples are of course a bit more involved that 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.

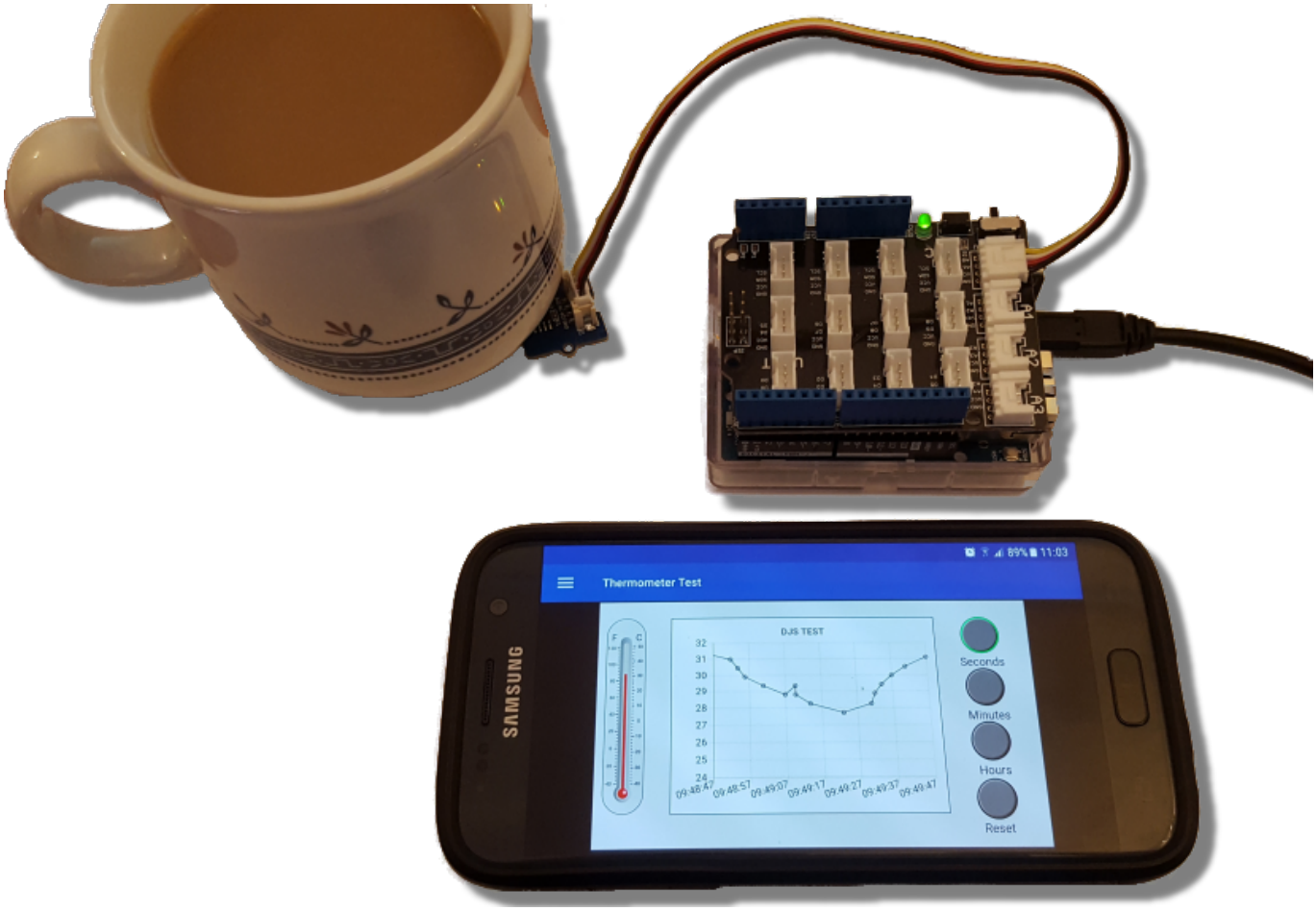
Deployment

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:

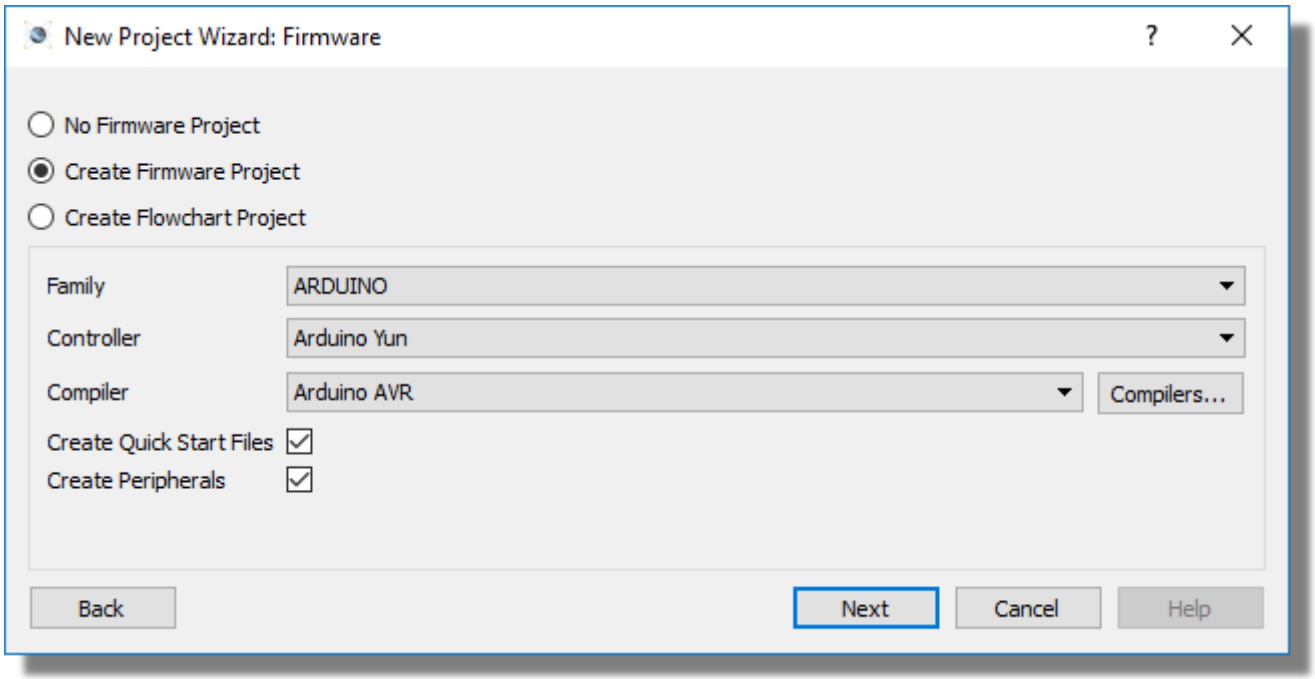
Tutorial 1 : Blink an LED (Visual Designer Project)

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.



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.

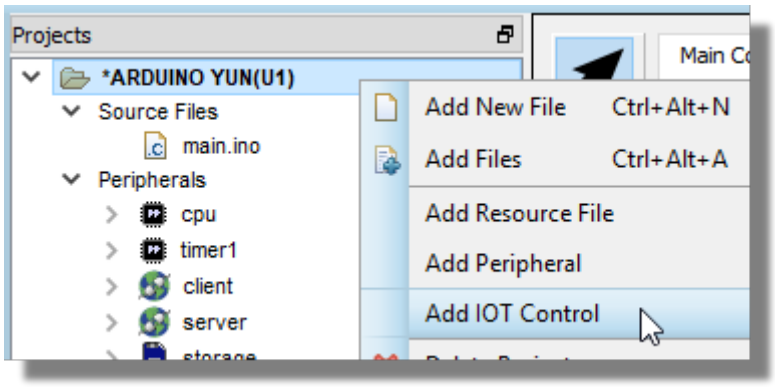
Default settings for the rest of the New Project Wizard are then fine.

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.

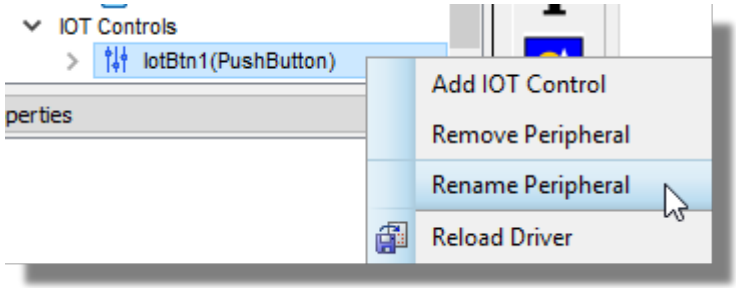
On the schematic you will find a pre-placed Arduino Yun.

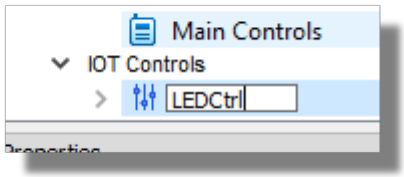
Design Phase

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.

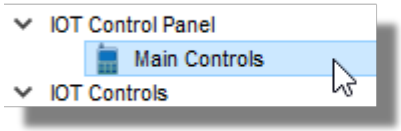


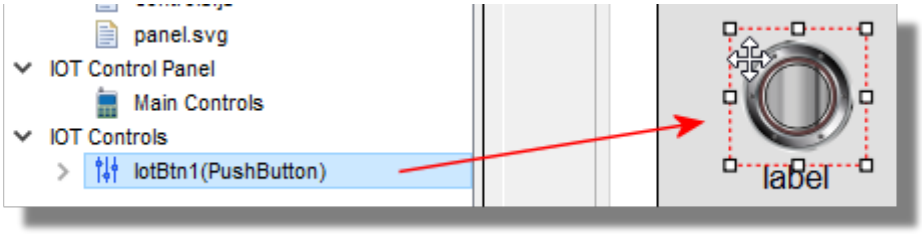
It's always a good idea to immediately rename the button (right click on in in Project tree -> rename command) to something that better reflects it's purpose



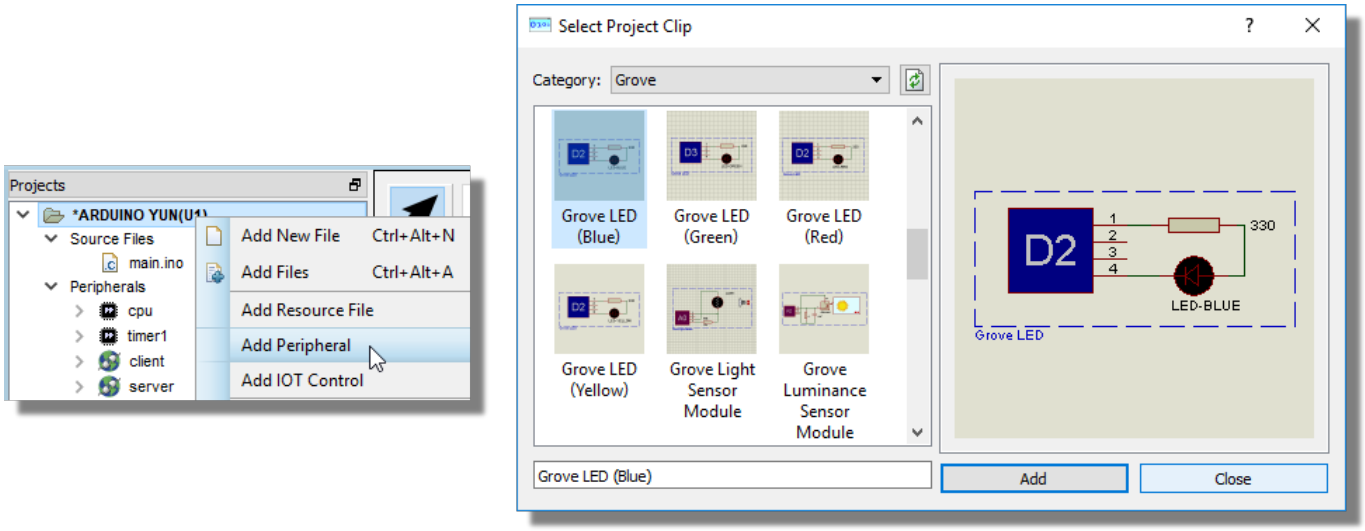


You double click on the front panel to open the panel editor and then you drag and drop to place.

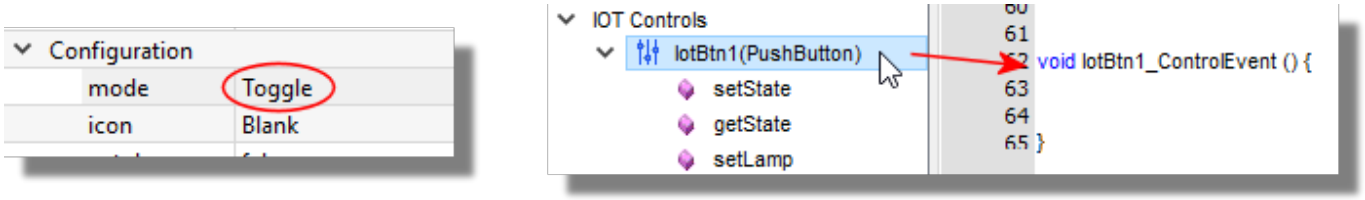




You can also add hardware to your schematic directly from the Add Peripheral dialogue - again on the right click context menu.

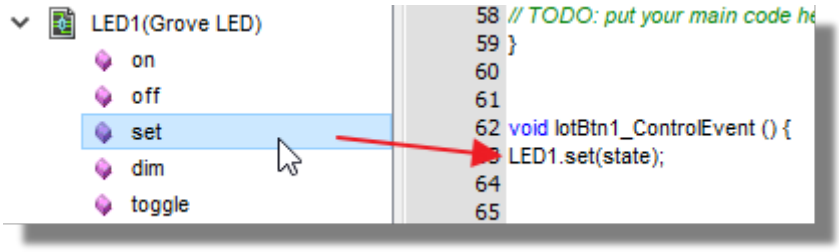


Now in the firmware to respond to the button press we need an event handler. Set the button Mode properties in the Main Control to be 'Toggle'. Now, double click on the Main.ino file and set the mouse caret in an empty space (after the void user_loop function) and then drag and drop from the IoT button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

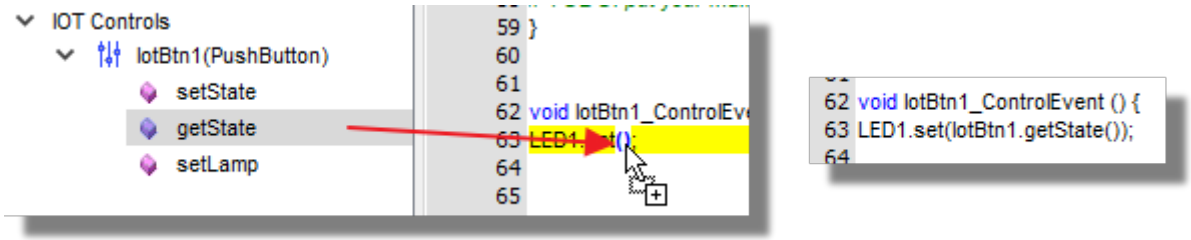


> ⚠ Do not change the names of functions or you will break the program. The exception is when you rename the control or peripheral after placing functions on the source window. In this case you'll need to rename the objects and handler prefixes manually or drag and drop them back out again.

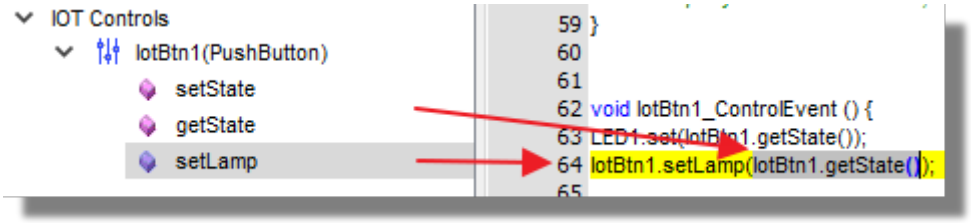
What we want to do is toggle the LED based on the state of the button. From the LED peripheral drag and drop the set routine but this time hold the CTRL button down before you drag. Move the mouse inside the button control event routine and then let go to insert at the current mouse position.



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.

MQTT Protocol

Introduction

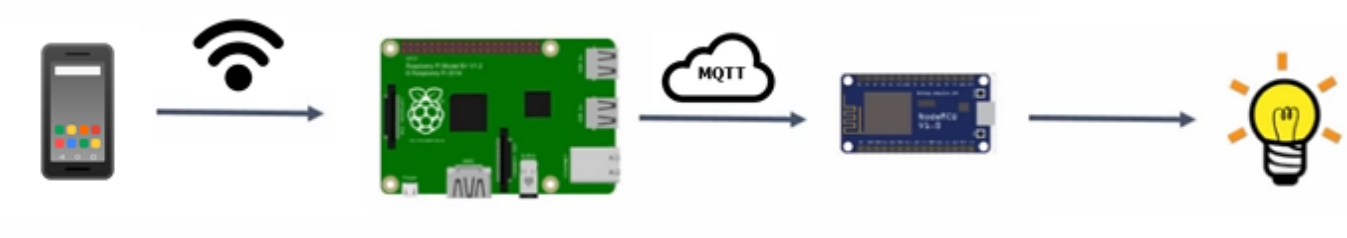
MQTT stands for "Message Query Telemetry Transport". It is a publish/subscribe, extremely simple and lightweight messaging protocol, designed for constrained devices and low-bandwidth, high-latency or unreliable networks. The design principles are to minimise network bandwidth and device resource requirements whilst also attempting to ensure reliability and some degree of assurance of delivery. These principles also turn out to make the protocol ideal of the emerging “machine-to-machine” (M2M) or “Internet of Things” world of connected devices, and for mobile applications where bandwidth and battery power are at a premium.

MQTT is supported directly inside the IoT Builder product, making it easy to set up multi-device systems and manage communications between them.

What can you do with it ?

There are two main things you can do with MQTT and Proteus.

1) You can send a command to control an output.

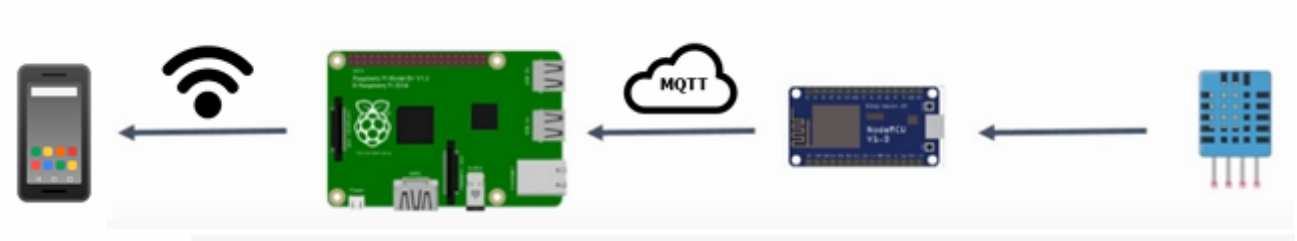


MQTT command from Raspberry Pi tells Arduino client how to control the light.

Here, your mobile device connects with the Raspberry Pi via wi-fi and loads the virtual front panel. It then transmits a command intended for the arduino appliance via the remote user interface. This command is sent to the cloud MQTT server. The Arduino device will receive the command when it wakes up and checks in with the MQTT server and the firmware on the Arduino will interpret and execute the command.

The presence of the MQTT server in the cloud means that the Arduino can sleep for a user determined time and then wake up periodically to receive and execute commands.

2) You can read and publish data from a sensor.



Read data from a sensor appliance and publish to the virtual front panel.

In this case the sensor data is periodically sent to the MQTT cloud server. The Raspberry Pi reads this data, interprets in and displays it on the virtual front panel which is viewable on your connected mobile device.

In both cases the MQTT protocol is used as a bridge between the appliance (arduino) and control unit (raspberry pi). Using a broker in the cloud is not essential but means that the appliance can sleep for long periods of time which saves power.

Basic Concepts

In order to work with MQTT you'll need to first understand a few basic concepts which are explained in brief below. You can also find a wealth of resources on the internet with more detailed information.

See Also:

Wikipedia - MQTT

Publish/Subscribe

A device may publish messages to the other devices in the system and a device may subscribe to a particular topic to receive messages published to that topic. So if Device1 wants to send a command to Device2 it must publish a message to an agreed topic that Device2 has subscribed to.

Messages

Messages are simply information exchanged between your devices. They can consist of either commands or data.

Topics

Topics are the named 'postbox' for messages. They are how a device registers interest in receiving incoming messages and how a device specifies where it want to publish messages. Topics are represented by case sensitive strings with a forward slash '/' to indicate the topic level.

Broker

The broker is responsible for three jobs:

- Receiving all the messages.
- Filtering the messages.
- Publishing the messages to all subscribed clients.

QoS

QoS stands for Quality of Service. It determines how the message is sent. This comes in three forms:

QoS 0: *At most once delivery*. Receiver does not send a response. Sender does not retry and therefore will delete the message locally once it has been sent. The message arrives at the receiver once or not at all.

QoS 1: *At least once delivery*. Receiver acknowledges the PUBLISH packet using the PUBACK packet.

QoS 2: *Exactly one delivery*. Neither loss nor packet duplication is acceptable. This QoS level adds protocol overhead and specifies many rules for both sender and receiver. Packets PUBREC, PUBREL and PUBCOMP are involved instead of PUBACK used in QoS 1.

Mosquitto Broker

Eclipse Mosquitto is an open source (EPL/EDL licensed) message broker that implements the MQTT protocol versions 5.0, 3.1.1 and 3.1. Mosquitto is lightweight and is suitable for use on all devices from low power single board computers to full servers. The installation and setup of the Mosquitto MQTT broker is a required step before you can start designing MQTT systems in Proteus. Fortunately, the process is fairly simple and is covered in some detail in the topic linked below.

See Also:

Mosquitto Broker Setup

Multi-Appliance Systems

It's very common with MQTT systems to have more than one appliance. For example, you might have a raspberry Pi control unit and then one or more arduino sensor units reporting data back to the control unit. In version 8.11 this can only be simulation by having one copy of Proteus open for each appliance. All copies of Proteus would then need to have their simulations running at the same time and be using the same broker to communicate.

We are currently working on a multi-board system whereby more than one appliance can be simulated inside a single copy of Proteus. This will mean that the entire system (rPi, arduino, etc.) can be exercised within one running Proteus simulation. We expect this work to be completed and released with the launch of Proteus v8.12

MOSQUITTO BROKER SETUP

Mosquitto Broker on PC

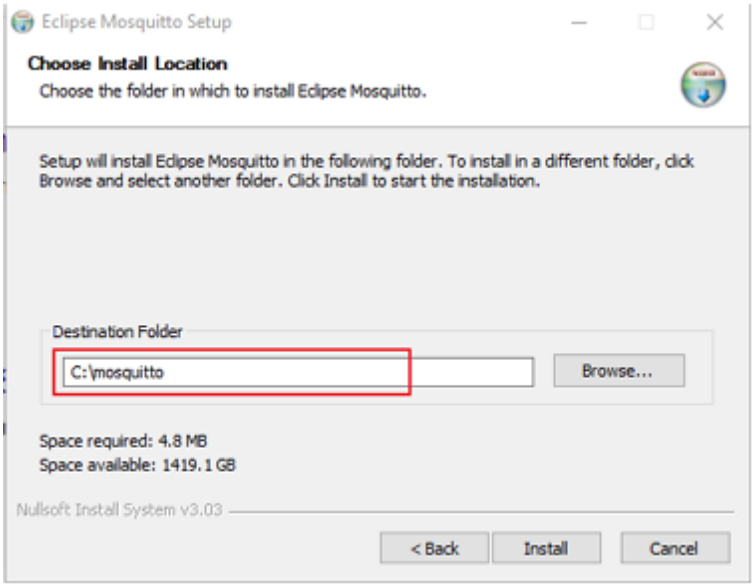
Before we get started with a project in Proteus we need to organise the broker. We will be using the popular Mosquitto broker. As discussed, the broker software can be installed anywhere, so long as the IoT system can see it and communicate with it. Practically speaking, there are three common configurations which we'll cover:

- Broker on PC
- Third party website broker.
- Broker on Raspberry Pi

Broker on PC is the easiest to get started with and is described in this topic.

To start simply visit: https://mosquitto.org/download/

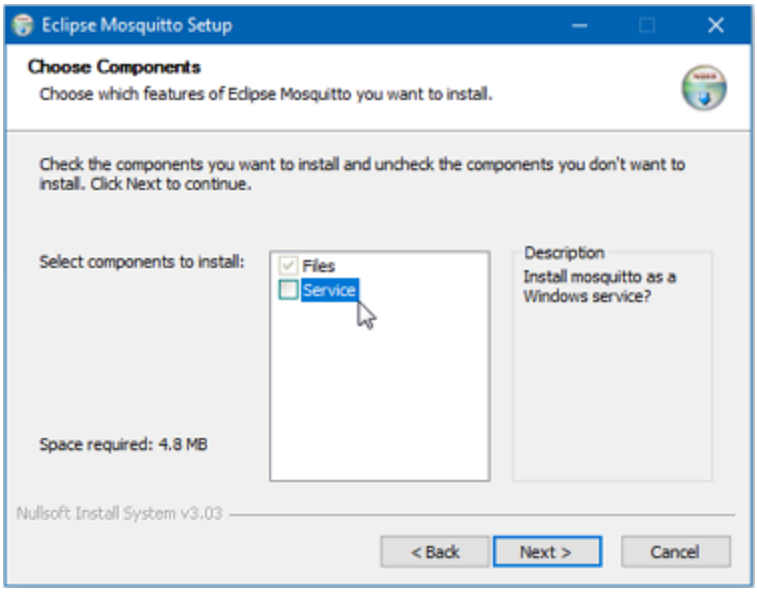
Install whichever version fits your windows machine. Since we will be changing the configuration file we recommend you change the installation directory out of Program Files which Windows does not like to include writable files.



Install to a writable directory.

> ⚠ If you choose to install in the default location (Program Files\mosquitto\) you will need to organise write access to the directory in order to change the config file as discussed below. Tools like Notepad ++ can open files as Admin or you may launch from an administrative command prompt.

Next, we recommend that you uncheck the mosquitto service installation before completing the install.



Do not install the service unless you are confident in your system configuration.

Mosquitto can be run from the console with debugging / logging which is much more useful during system development. Since you cannot have two mosquitto brokers listening on the same port it will also become confusing to have a service broker running as well as console broker. Additionally, the service broker is not suited to servicing multiple Proteus simulation clients because simulation client ID's are not randomised and the broker cannot handle multiple clients with the same ID. Using local console brokers therefore is almost always the better option.

Configuration File

Before we run the Mosquitto service there is one particular configuration change that we recommend.

1) Open up the configuration file in your favourite text editor: `C:\Program Files\mosquitto\mosquitto.conf`

2) Add the following text on a new line within the file: `queue_qos0_messages true`

> QoS 0 messages aren't stored by the broker by default. This setting allows them to be stored which is essential to avoid message being lost in multiple appliance systems. For example, if a subscriber appliance is asleep when a message is published and there is no storage of the message it will be lost. It's important for us to set this in the config file because the Arduino stack only supports the sending of QoS level 0 messages.

Running the Broker

There are two options for running the broker. You can run the service or you can start a process from the command line. If you run from the command line via the procedure below you will get useful logging information during your testing session and the broker will close when you close the command prompt. If you run the service no logging is available and the broker will continue running until the service is stopped.

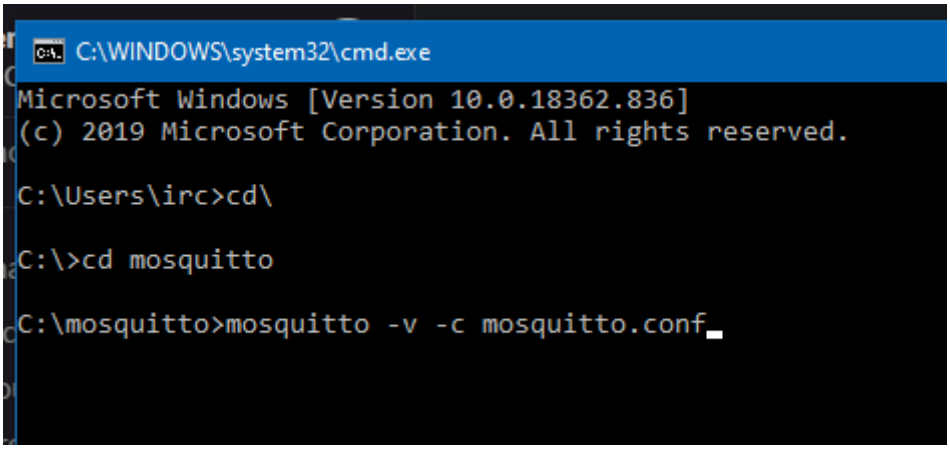
Broker from Command Prompt (Recommended)

Open up a command prompt on the PC, navigate to the mosquitto directory and launch the broker on your machine.

1) Start Menu -> Run -> cmd

2) Navigate to installation directory using the cd command.

3) Type `mosquitto -v -c mosqitto.conf` and it will start a broker on your machine



Starting a console broker session.

> The -v parameter enables logging to the command prompt. In particular, it will tell you which port the broker is listening on.

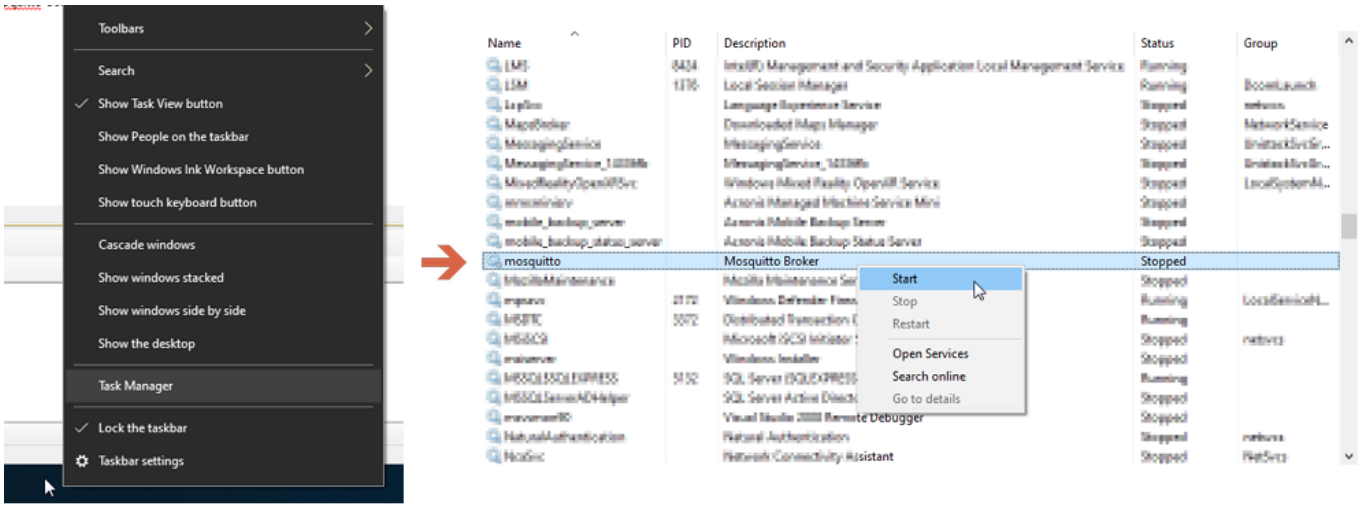
> You may well need to allow mosquitto to pass through the Windows firewall if you get prompted to do so.

Note that closing the command prompt will close the broker and so it should be left open for the session.

Broker Service (Advanced Use - No Debug)

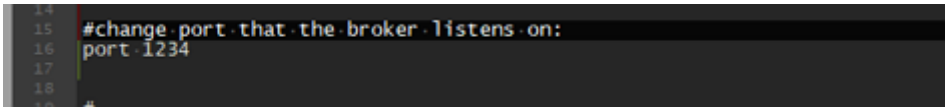
1) Right click on the program bar at the bottom of Windows and select task manager from the resulting context menu.

2) Select the services tab and then scroll to the mosquitto service. Right click on the service and select start.



> You can also use 'net start mosquitto' and 'net stop mosquitto' from the command prompt.

If this doesn't work it's most likely because something else is sitting on port 1883 which the mosquitto service listens on by default. The mosquitto service listens on port 1883 by default but you can change in the config file using the port command.

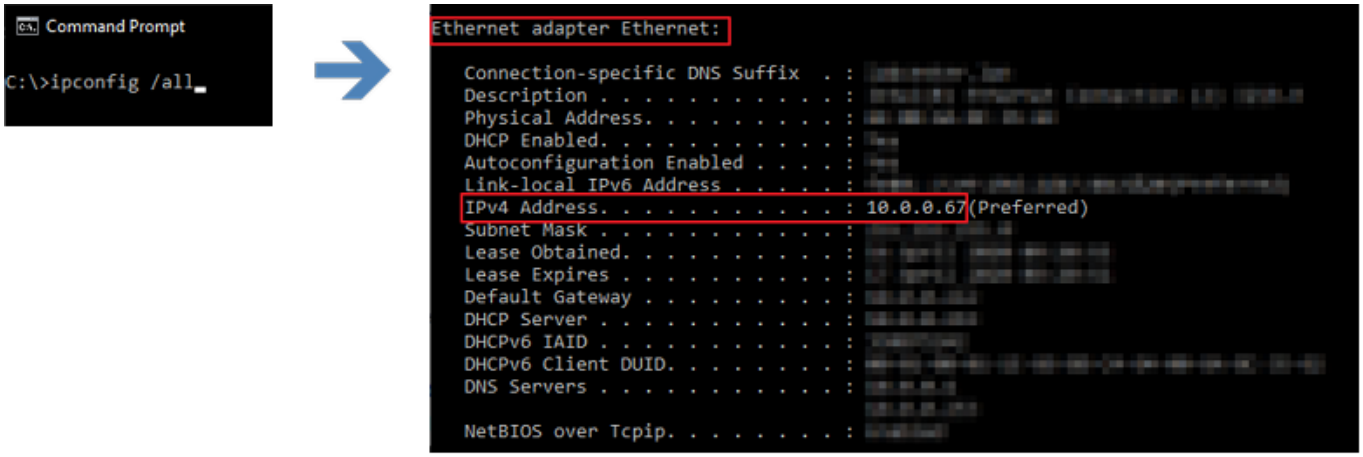


Changing the port in the config file.

IP Address

If you are running Proteus on the same machine as the mosquitto broker you won't need the IP address. In this case we'll simply use the default "LocalHost".

If you are running the broker and Proteus on separate machines, you can get the address by using the ipconfig command on the machine running the broker and look for the “IPv4 Address”. This address will be the broker address.



Retrieving the IP address of the broker machine (necessary only when it is not the same as the machine running the Proteus simulation!)

> If you are plugged in to the router via an ethernet cable, then scroll down the list until your reach the 'Ethernet adapter Ethernet' section. Otherwise find your Wireless Adapter section.

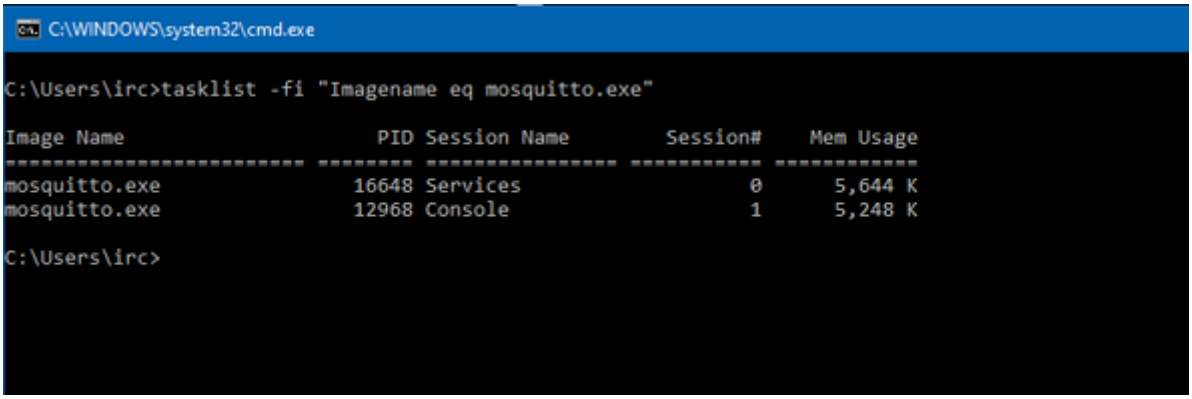
Troubleshooting

Almost all of the problems with setting up the broker will be related to whether a broker is actually running, what port it is on and if it is a service or a console.

Multiple brokers

Since you can change port numbers it's possible to have multiple brokers in play at any time so the first step will be to establish what is going on.

Use the tasklist command at the command prompt as shown below. Note that it not only lists the brokers running but will tell you whether they are service or console based. In my example I have both a service broker and a console broker running. Not at all clever but set up this way for illustration.



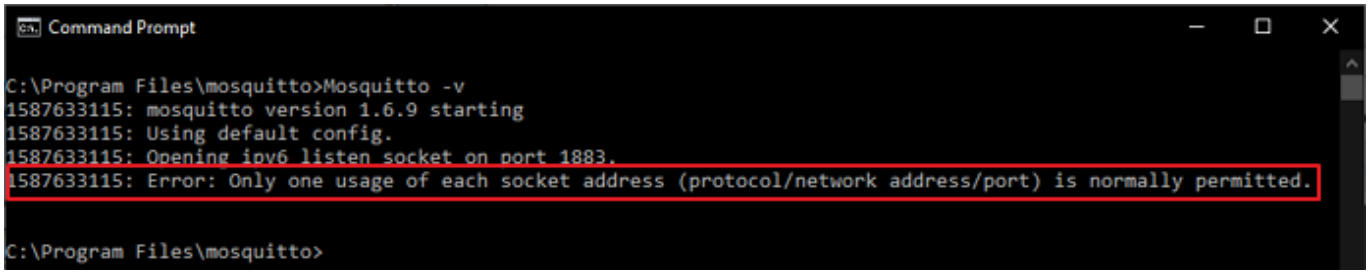
Use the tasklist command to establish what brokers are running on the PC.

To sort out this situation you need to decide whether you want a broker to always be available or whether you want the broker to be available for a session. In the former case you want to keep the service and close the command prompt and in the latter case you want to stop the service and keep the command prompt open.

Multiple brokers on one PC are not recommended as they are likely to cause confusion.

Port Number Conflicts

Another common set of problems is with port numbers. If you get errors at the command prompt or services refusing to start it normally means that something else (usually another broker instance !) is sitting on the port.



Command prompt error when launching a broker and there is a port conflict.

1) Use tasklist as above to make sure you haven't got another broker running. If you do, we'd recommend you close it as running more than broker is going to be confusing unless you are an expert at this type of thing.

2) If another program is using the port (1883 by default) you can change the port used by the broker in the config file.

Accessing the broker from another device

If you are running Proteus on the same PC as the broker then everything will be nice and simple for simulation as you can just use localhost to address the broker. If you are accessing the broker from another device you'll need both IP address and port number.

- The port number will be listed on the command prompt if it's a console broker session.
- The port number will be in the config file it it's running as a service. If there is no mention in the config file then the port number is 1883.

Third Party Website Broker

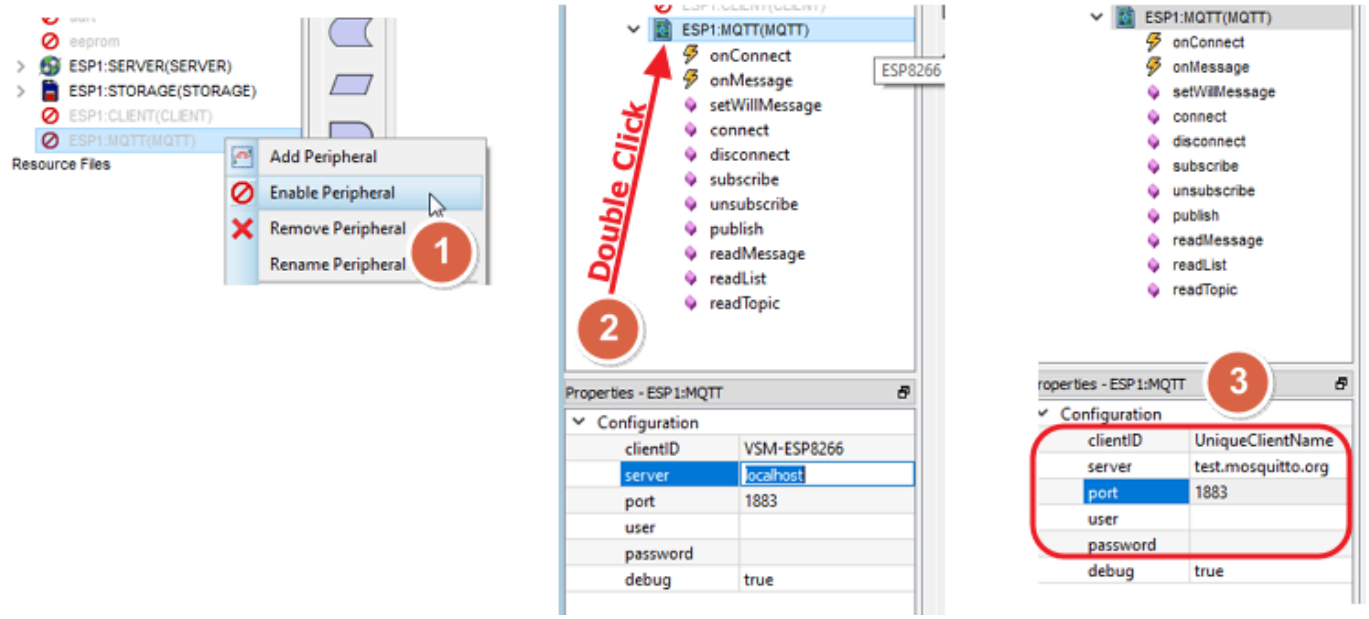
Here we will use a third party website that has Mosquitto installed as the server. There are a few of these but the one we have chosen is test.mosquitto.org.

With a website broker of course no installation or setup is required. All we have to do in our project is specify the broker location in the MQTT peripheral properties. To do this, we would

1) Make sure the MQTT peripheral is enabled and then double click on it to launch the property panel.

2) Set a unique client ID.

3) Specify the URL and port number of the web broker.



We need to tell Proteus where the broker is and how

This can be a fast way to get setup and working and it is particularly useful where two appliances are remote from each other. However, during development especially there are some significant drawbacks:

- No debug. One of the benefits of running a copy of the broker on your PC is that you can launch the broker with debug to the console and see all communications.
- Unique Client ID's. If you are in a classroom environment for example you must make sure each client has a unique client ID for the broker. This isn't a problem when the broker exists on each machine locally but is an issue with a website broker. If you have multiple devices with the same name, every time the new device with that name connects to the broker it will kick the old device with that name off. This can cause issues like loops as the old device attempts to reconnect. If you are planning to use multiple devices with one broker it is essential that each device has a unique clientID. The clientID can be set as a design time configuration property (as shown above) or at run time via a dedicated method call.
- You can't configure the broker. Unlike with the PC configuration the free web broker is set to default configuration. This means QoS 0 messages are not queued and QoS 1 and 2 are. Be aware that this could be a serious problem for Arduino appliances if they can only send QoS 0 messages. It also means that you can't supply credentials to the broker and then log in from the clients.
- Topics are shared across the entire broker user base. Not only is there no privacy at all but perhaps more significantly common topic names may have rogue messages published to them or be shared outside your intended user base.
- The free mosquitto broker is used for debugging and development of Mosquitto itself. This means it may be very slow (if they are debugging) or even suffer from instability and downtime (if they are beta testing).

For these reasons we've opted for installing the broker on a local machine as the preferred development setup and used that in our MQTT Basics Tutorial.

Broker on Raspberry Pi

Here we will use a raspberry PI as the server. To configure the Mosquitto software on a Raspberry Pi:

1) Install the broker using this command:

```
sudo apt install mosquitto mosquitto-clients
```

2) Check to make sure the broker installed correctly

```
sudo systemctl status mosquitto
```

Result:

active (running)

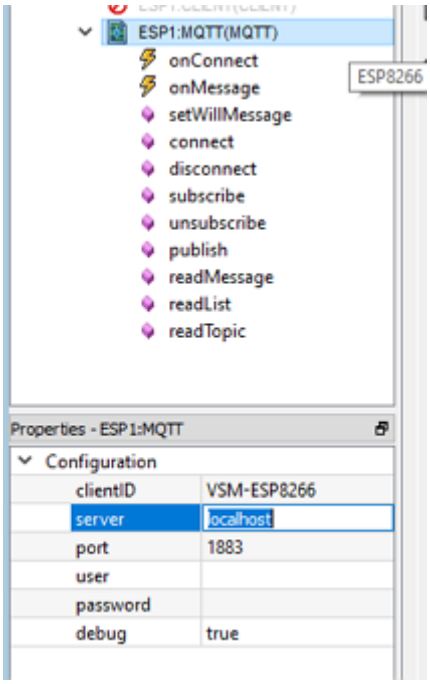
3) To connect, use the command `ifconfig` to identify the IP address of the RPi. You can use localhost when connecting from the RPi but you will need the IP when connecting from an external device.

MQTT Programming Methods

Introduction

Visual Designer makes programming with MQTT simple and straightforward. However, before you can start writing programs you need to follow these three setup steps:

- Make sure your hardware has internet connectivity. If you are working with Arduino go to Add Peripheral and choose an option from the Internet of Things category (e.g. ESP8266). If you are working with Raspberry Pi this isn't necessary because internet connectivity is built in on the rPi.
- Enable the MQTT peripheral. For code saving reasons many on board peripherals are disabled by default. Right click on MQTT in the Project Tree and enable the peripheral.
- Configure the MQTT peripheral. Double click on the MQTT peripheral to open the properties box underneath. This lets us set the IP address and port of the MQTT broker (our computer in the example below) along with other access details.



Design time configuration of the MQTT Peripheral

clientID

clientID will specify the name your device will receive. This should be unique for the broker. If you have multiple devices with the same name, every time the new device with that name connects to the broker it will kick the old device with that name off. This can cause issues like loops as the old device attempts to reconnect.

You can also set the ClientID during runtime with a method.

Server

The server will specify the location of the broker. For example, if the mosquitto broker is running on the same machine as Proteus, you can simply use localhost to connect to it, or you can use the IP address of your machine e.g. 192.168.0.1.

If the broker is running on a machine on the local network, you will need its IP address to replace localhost.

If you wish you can setup or find online brokers such as test.mosquitto.org which will allow you to use the MQTT without your own broker. These brokers are on the internet and can therefore be accessed by any network with internet, allowing your MQTT project to span multiple networks and buildings.

See Also:

Broker Configuration

Port

The default port the broker runs on is 1883, however if you wish you can change this in the broker configuration files. Most online brokers such as test.mosquitto.org use 1883 as the default port also.

User/Password

Some brokers may require you to add credentials. This is simply where you add the credentials to your project to allow them to access a secure broker. Leaving them blank will indicate you don't wish to use them.

Debug

Setting this to true compiles in the MQTT debug statements.

On Arduino, these dump data to the Bridge Console. In simulation it comes out in the simulation log. On ESP8266/ Playkit you can connect to it via telnet and on Yun there is port you can connect to to view it. On Arduino it is a compile time thing so the debug statements only use ROM space if it is enabled.

On Raspberry Pi it dumps to stdout, so if the server program is run from a command prompt you can view it that way.

> Quite apart from the MQTT debug statements described here you can add debug blocks to your own program via the debug method under CPU in the project tree. These will travel out through the same channel to your hardware platform.

That's it. Now the MQTT methods are all available for use in your program and the system knows the location of the broker for communications. Each method is discussed in the topics below and you'll find practical examples of most of them in use in the MQTT Basics tutorial.

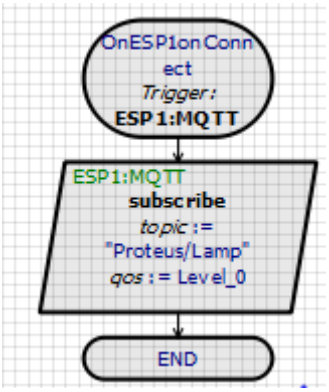
See Also:

MQTT Basics Tutorial

onConnect Event Handler

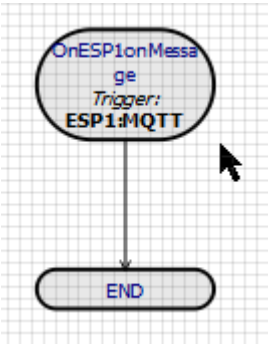
The onConnect event handler is called each time a connection is established.

Library code within the MQTT peripheral will automatically re-connect if the connection to the broker is lost and so this handler may be called periodically if the connection is unstable. If a connection is made with cleansession=TRUE (the default) then all subscriptions will be lost when the connection is re-established. It is therefore normal to place subscriptions in this handler to ensure they are in place and that the application recovers from a broken connection seamlessly.



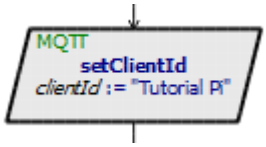
Almost all MQTT programs will have section of flowchart like this.

onMessage Event Handler



The onMessage event is called when a message is sent to a topic the client is subscribed to. This is where the three read methods (readMessage, readList and readTopic) come into play. They can only be used within this event as they directly relate to it.

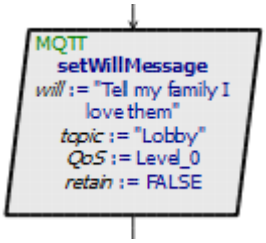
setClientID Method



setClientID allows you to specify the name your device will receive and how it will be identified by the broker. You can also do this via the design time configuration properties.

> ⚠ Remember that each client needs to have a unique Client ID for the broker. If you have multiple devices with the same name, every time the new device with that name connects to the broker it will kick the old device with that name off. This can cause issues like loops as the old device attempts to reconnect. If you are planning to use multiple devices with one broker it is essential that each device has a unique clientID.

setWillMessage Method

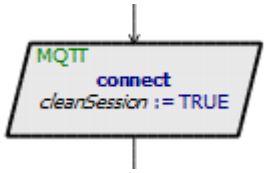


The will message is a publish method that is saved by the broker. This message will be saved by the broker however not sent. In the event that the client disconnects unexpectedly the broker will then remember and publish the will message.

> ⚠ In order to work the will message MUST be put before the connect method in your program.

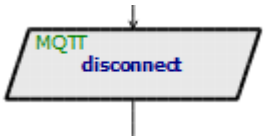
For more on the workings of this function refer to Publish method.

connect Method



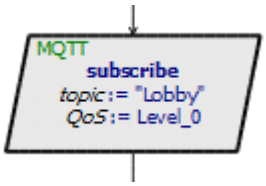
The connect method will connect the client to the MQTT broker. If there are old messages stored by the broker during the time the client was turned off, cleanSession = True will wipe them clean and delete them, whereas cleanSession = False will remember them. It will also remember old subscriptions from when it was last turned on.

disconnect Method



If you want to disconnect from the broker simply use the disconnect method.

subscribe Method

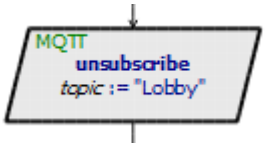


This method simply allows you to subscribe to a topic. Topics are case sensitive so be careful when typing. Topics can also be layered like a filing system. For example, Main/Lobby would be a two-layer topic. Main could contain multiple topics, for example Main/Hallway and Main/Kitchen.

If you want to subscribe to multiple topics easily you can do Main/# and it will subscribe to everything under it.

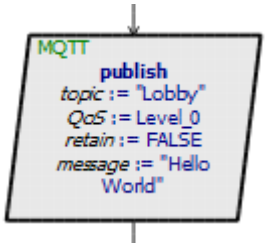
A wild card is were you can get all topics at the same layer with the same name. For example, if all those rooms had a /Temp topic. Main/Lobby/Temp, Main/Hallway/Temp and Main/Kitchen/Temp. To only subscribe to those Temp topics, you can do Main/+/Temp.

unsubscribe Method



If you no longer wish to receive messages from a topic, use the unsubscribe method.

publish Method

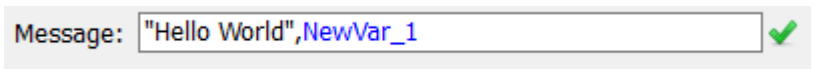


The publish method is how you send a message.

Topic refers to the topic in the broker you wish to publish the message to.

Retain is whether or not you wish the broker to remember the message. If this is set to true, then the message will be sent to every device new and old every time they connect and subscribe to that topic. Even if they have had the message before. You cannot clear retained messages so be careful.

Message is a list that lets you add multiple variables separated by a comma.



This message will look something like ““Hello World”,0” to the broker and any other clients.

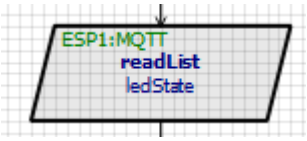
readMessage Method



readMessage will simply return the whole message as a string. This can only be used inside the onMessage event handler.

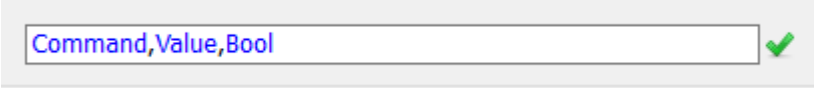
> If the incoming data is not a string it's better to use readList.

readList Method



readList will split the message string by the commas. This can only be used inside the onMessage event handler.

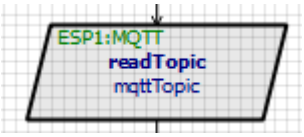
For example, “Hello World”,1,True would become a list of three variables. A String, an Int and a Boolean. You as the user need to know in what order those variables are, create variables of the correct type and add them as a list to this function.



> readList can easily be used with a list of 1 item. It's very useful if the incoming data is not a string as you can read into a variable of any type.

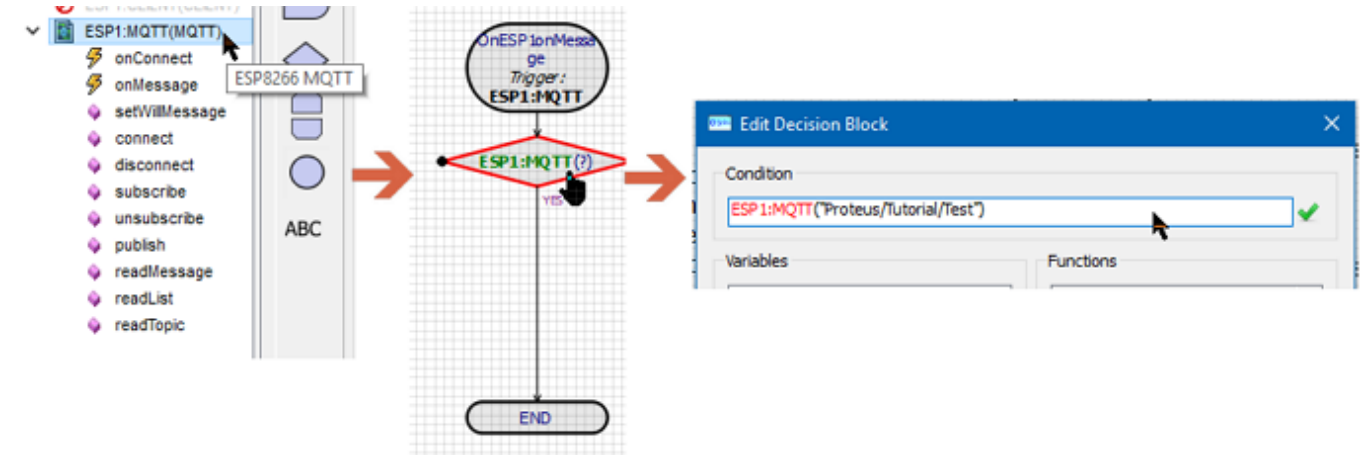
> If you have the wrong variable type or your list is in the wrong order for the incoming data a program crash is quite likely.

readTopic Method



readTopic will return the topic that the message was sent to as a string. This can only be used inside the onMessage event handler.

You can also drag and drop a decision block from the MQTT peripheral directly. This allows you to quickly check to see if the topic on a message is the same as the one you enter



Since the reason for reading the topic is most often to perform different actions based on the topic this is often a really handy shortcut.

See Also:

- MQTT Basics Tutorial (Subscribe)

MQTT BASICS

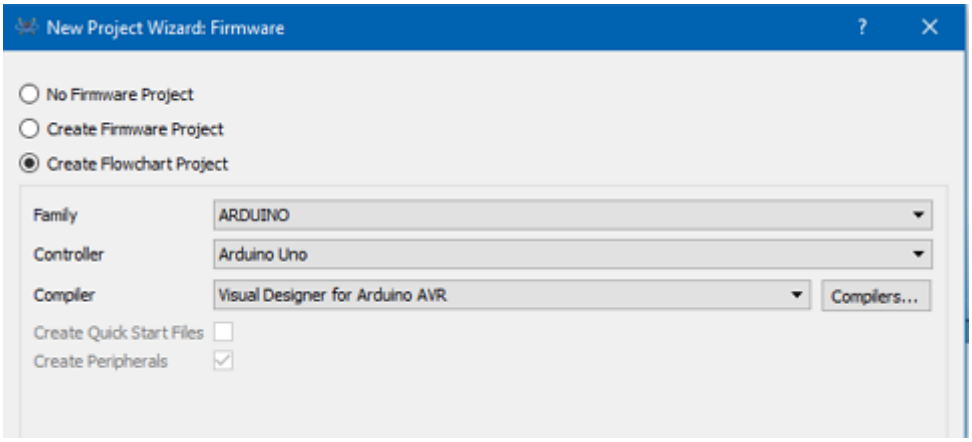
Introduction

This introductory tutorial covers the basics of publishing messages from the appliance to the broker and then subscribing to a topic which has messages published to it. To keep things as simple as possible we will be working with a single appliance and will view published messages directly via the MQTT broker. We are therefore going to assume that before you start this tutorial you have installed the mosquitto broker on your computer.

See Also:

MQTT Broker Setup on PC

Our appliance will consist of Arduino Uno and ESP8266 so let's start with the New Project Wizard on the home page. On the firmware screen create the project with an Arduino Uno target and the Visual Designer compiler.

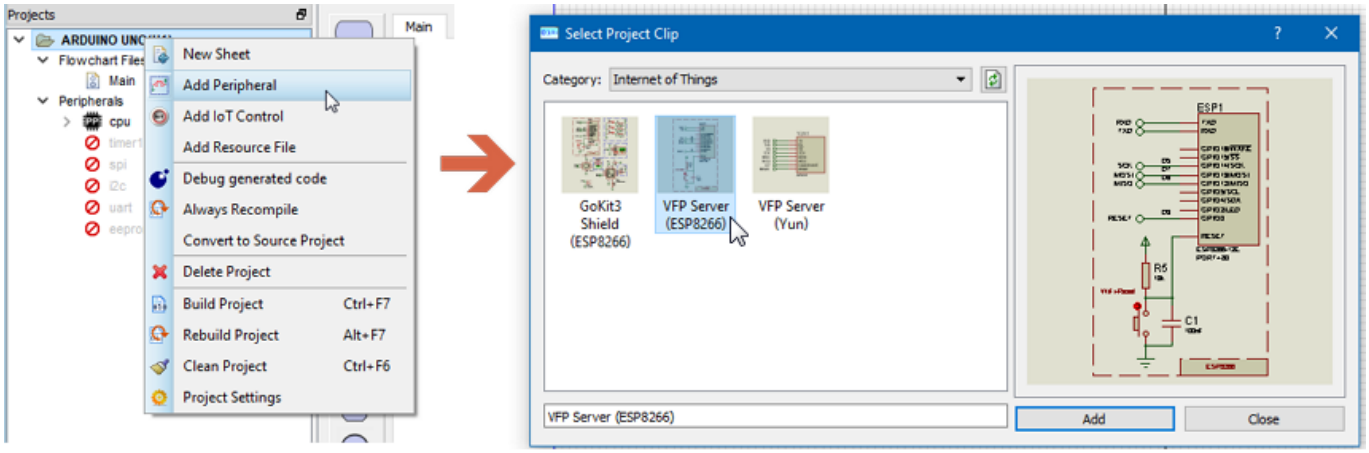


Set up the project with Arduino Uno controller and the Visual Designer Compiler.

Continue on through the wizard until completion. When you are finished you should see that you have a blank flowchart in the Visual Designer tab and a schematic with the Arduino Uno pre-placed.

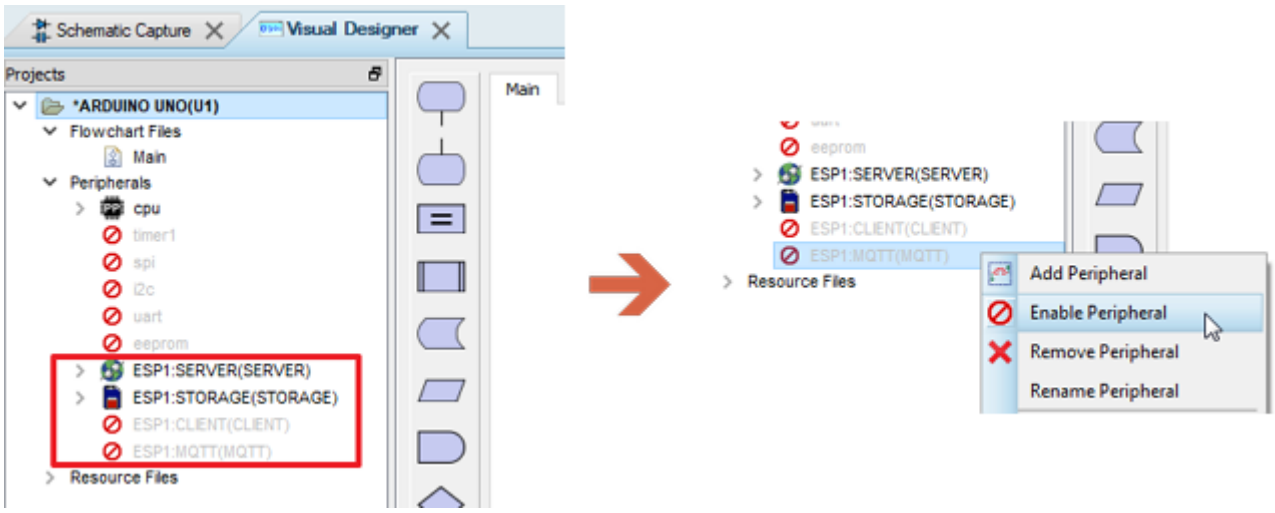
Hardware Configuration

In order for the Arduino to publish messages it needs to be connected to a wireless device. We'll use the ESP8266. To add the ESP8266 to the project first make sure that you are on the Visual Designer tab. Next, right click on the Arduino Uno entry in the project tree and select the Add Peripheral command. Navigate to the Internet of Things category, select the ESP8266 VFP server and click Add to insert it into the project.



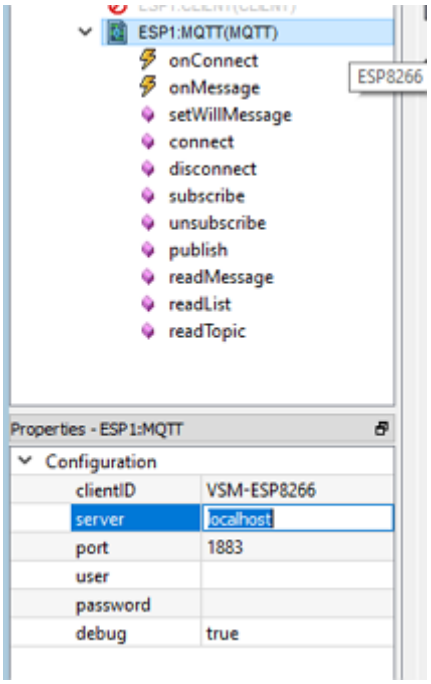
Adding the ESP8266 peripheral into the project.

You'll notice that after adding the ESP8266 peripheral we have extra options in the peripherals project tree. The MQTT functionality is disabled by default to save on Arduino program memory space when it is not in use but we can enable it from the right click context menu.



Enabling MQTT after adding the ESP8266 Peripheral.

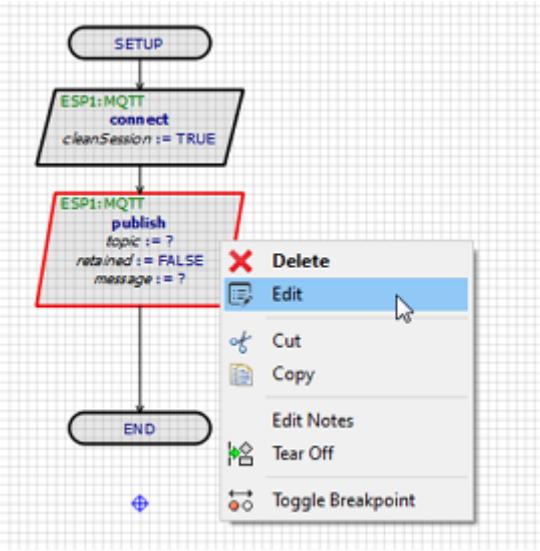
We also need to specify the location of the broker. Double click on the MQTT peripheral entry to open the property panel underneath. Assuming we've installed mosquitto on the same machine we can leave the defaults are localhost. If the mosquitto broker is on another machine you'll need to set the IP address of the machine here. Similarly, port 1883 is the default for mosquitto so you don't need to change it here unless you've made changes to the mosquitto config.



Double click to get the property pane and then set the server and port.

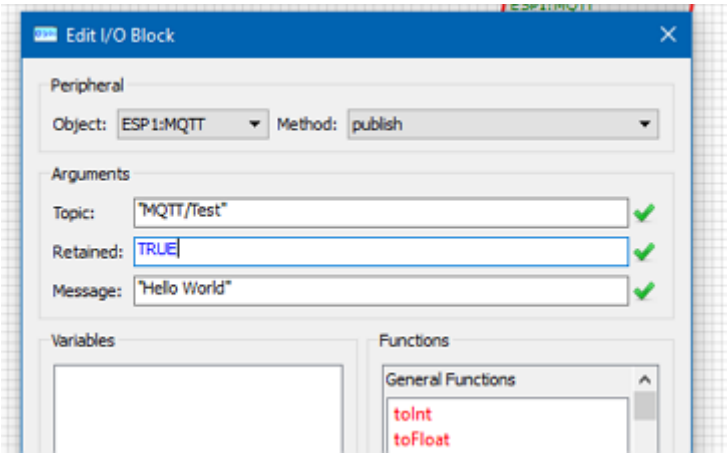
Publishing

Let's turn our attention to the firmware and publish a message to the broker. To keep things as simple as possible we'll do everything in the setup routine and we'll send only one message. Start by dragging across the connect method from the MQTT section of the project tree and drop it inside the setup routine. Once that's in place drag the publish method directly underneath it and right click to edit.



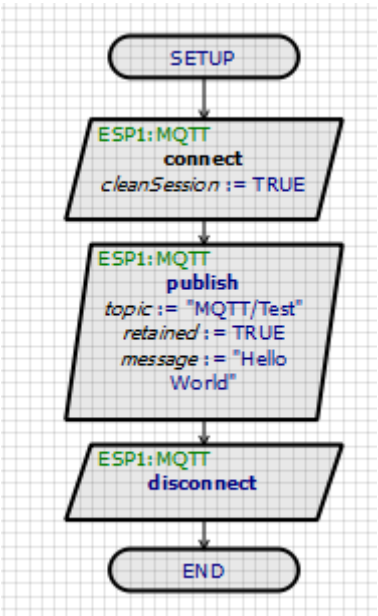
Editing the Publish Method

We need to setup a topic that we are going to publish to and then enter the message that we are going to send. Since we are sending only one message in the Setup routine we need to set the message to be retained so that the broker will store it.



We're going to send a Hello World message to the MQTT/Test topic

Finally, we need to close the connection by dragging the disconnect method to the bottom of the setup routine. Our final firmware flowchart looks like the screenshot below.

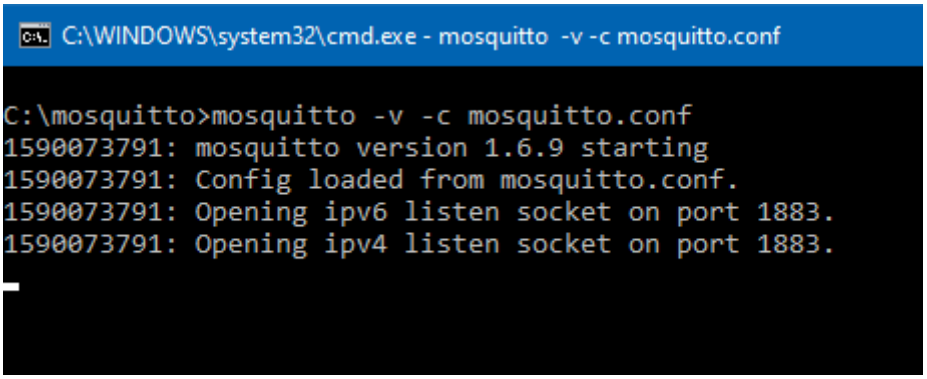


This is how easy it is to publish an MQTT message in Visual Designer !

Simulation (Publish)

Before we run the simulation we need to run the mosquitto broker. This process is described in detail in the broker setup topic but basically you start a command prompt, navigate to the mosquitto directory and then start the session with logging using the following command:

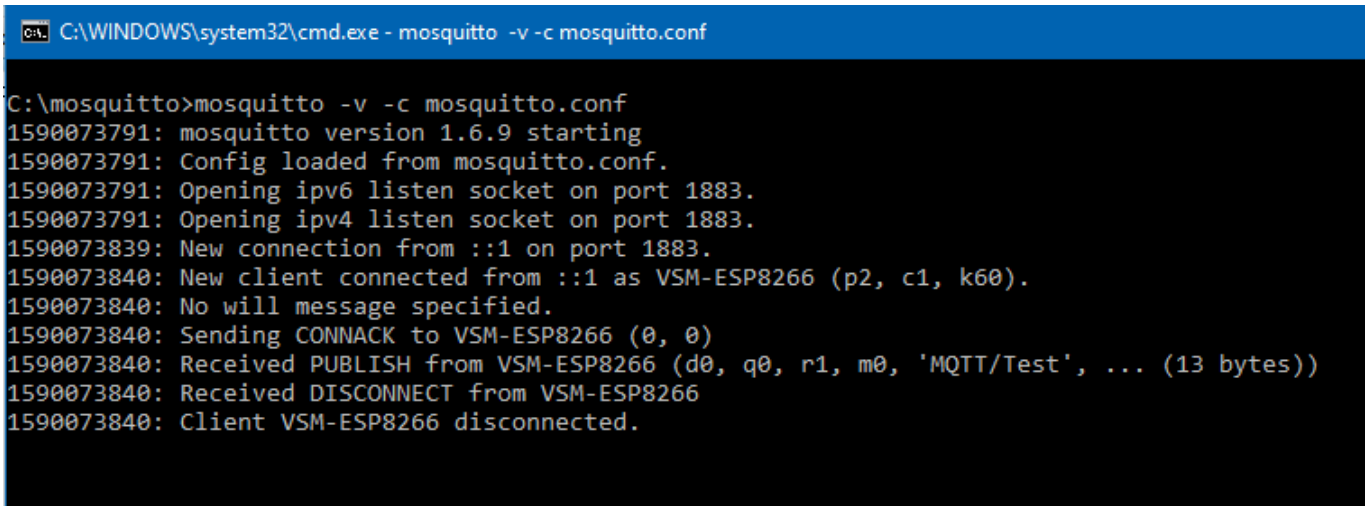
```
mosquitto -v -c mosquitto.conf
```



Console broker is running.

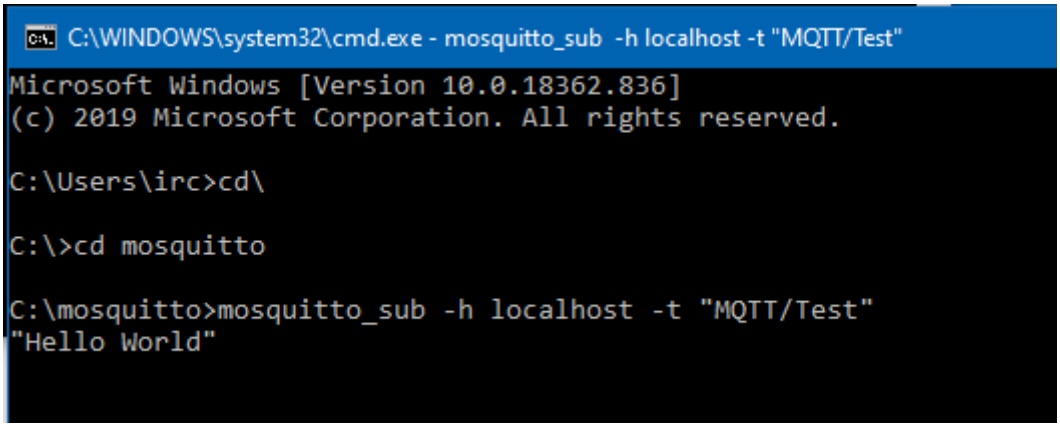
> The -v flag enables logging while the -c flag tells the broker to run the session using configuration from the mosquitto.conf file.

Now the broker is running we can go ahead and run the simulation. You should see that the broker logs the connection being established, the topic being published to and then the session being disconnected.



Session activity shown by the logging on the mosquitto console.

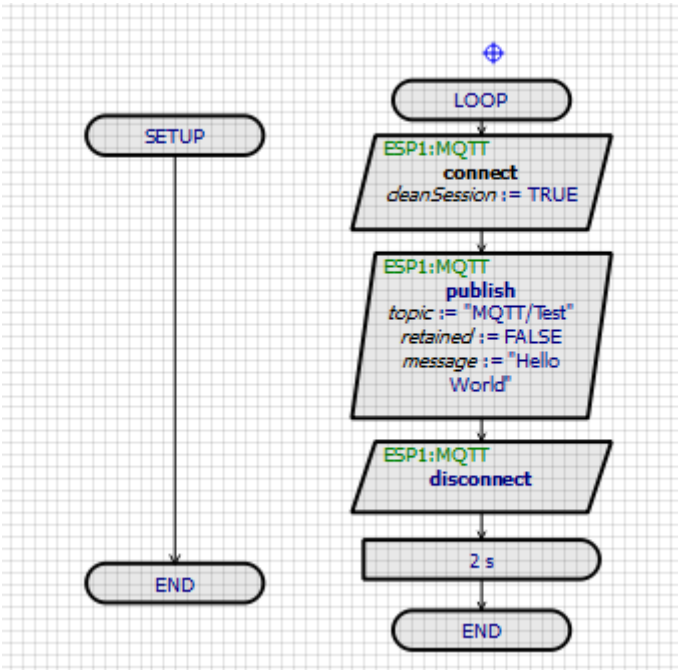
So far, so good. However, to be sure everything is working we need to subscribe to the topic so that we can get the message itself and confirm that transmission was successful. We can do this with a separate command prompt. Launch the command prompt from the Start menu (type cmd in the search bar). Navigate to the mosquitto directory using the cd command and then subscribe to the topic as shown below.



Subscribing to the topic and receiving the message.

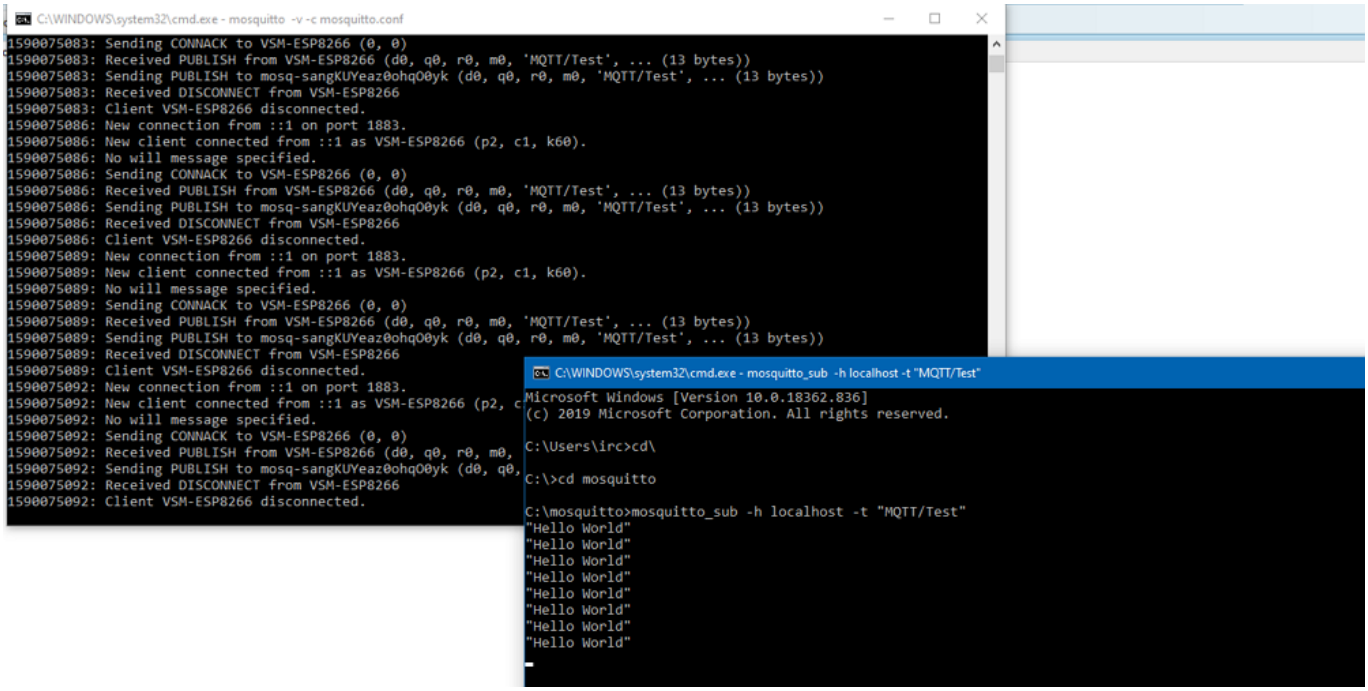
You should also notice at this point that the console broker session has updated to reflect the subscription.

Now, let's leave both command prompts running but stop the simulation and make some changes to our program. Specifically, we'll move the entire program into the loop routine so it is constantly executed. We'll need to add a delay so that we don't hammer the broker with messages and we definitely want to change the message so that it not retained. Our new test program therefore looks like the one below:



Constantly publishing messages to the broker.

Press the play button again and we can see continuous activity on both the console monitor which is logging the broker activity and also on our command prompt which is subscribed to the topic.

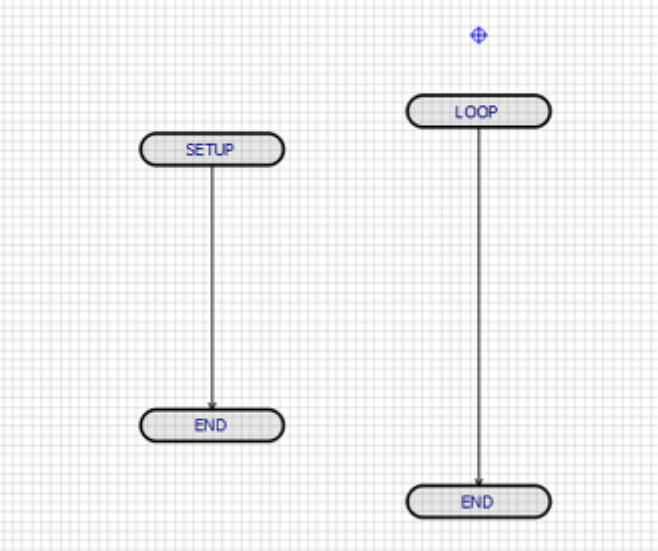


Broker activity on console and command prompt subscribed to the topic.

When you are finished we can stop the simulation. We'll leave both command prompts open as we'll use the broker again when subscribing to a topic in the following section of the tutorial.

Subscribing

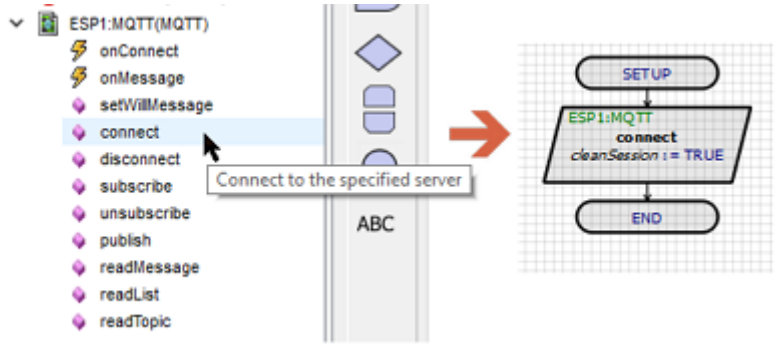
To subscribe to a topic we need to re-write our firmware program so let's return to the visual designer flowchart module and delete the current program blocks



Skeleton Flowchart ready for us to program our Subscribe logic.

Connect

The first step is to establish the connection. We can do this in the SETUP routine as library code within the MQTT peripheral will automatically re-connect if the connection to the broker is lost. Drag and drop the connect method from the MQTT peripheral in the project tree into the setup routine.



Establish a connection in the setup routine.

> We know where the broker is in order to establish a connection to it because we specified the location in the MQTT properties pane. Double click on the MQTT peripheral in the project tree to see or edit the properties.

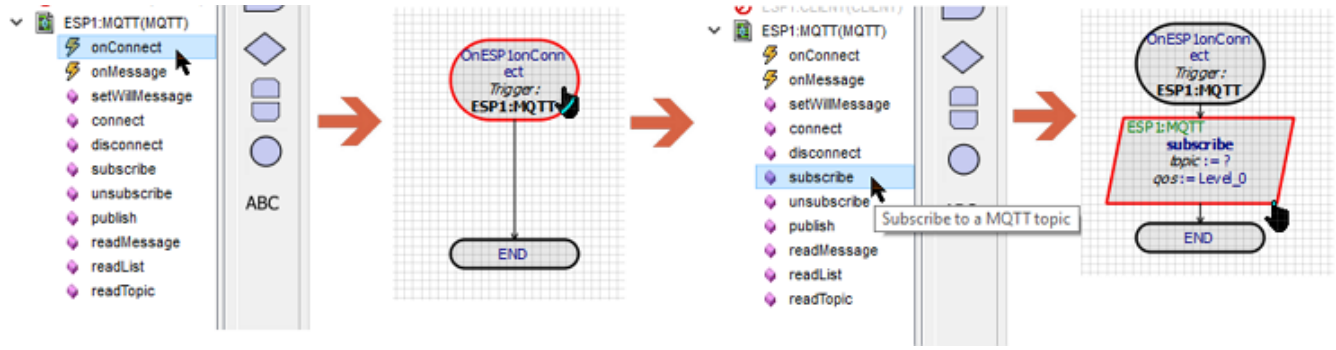
Next, we want to make use of the two event handlers available under the MQTT peripheral. These are called onConnect and onMessage and are shown with a little lightning bolt icon beside them in the project tree. An event handler is a routine in our program that is called when a condition is true. For MQTT event handlers:

- onConnect is called whenever a connection is established.
- onMessage gets called whenever the client receives a message (from a topic the client is subscribed to), which has been forwarded by the broker.

Subscribe

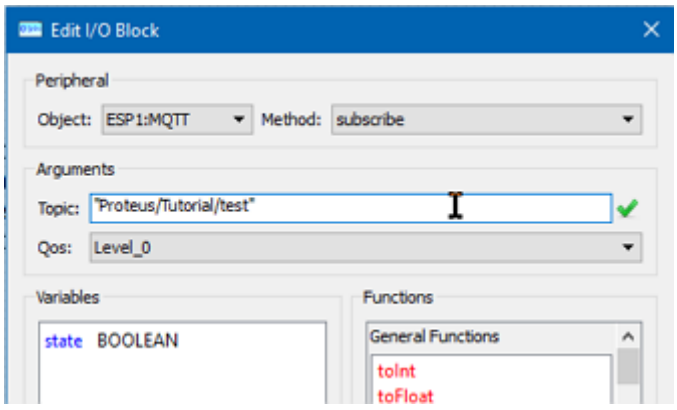
We want to subscribe to topics inside the onConnect event handler. Since we've started a session in the setup routine this will be called for the current session. It will also be called if we ever temporarily lose the connection and then need to re-connect.

Drag and drop the onConnect event handler and then place a subscribe method block inside the event handler.



We want to subscribe to topics inside the onConnect function.

Next, we want to edit the subscribe method and specify the topic that we want to receive messages.



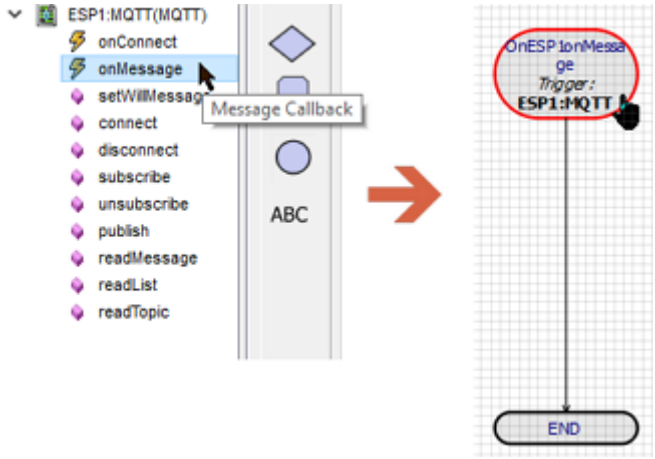
Subscribe to the "Proteus/Tutorial/test" topic.

> Remember the double quotes (" ") around the topic text.

We need to leave the quality of service (Qos) parameter at level 0 as this is the only level currently supported by the Arduino stack.

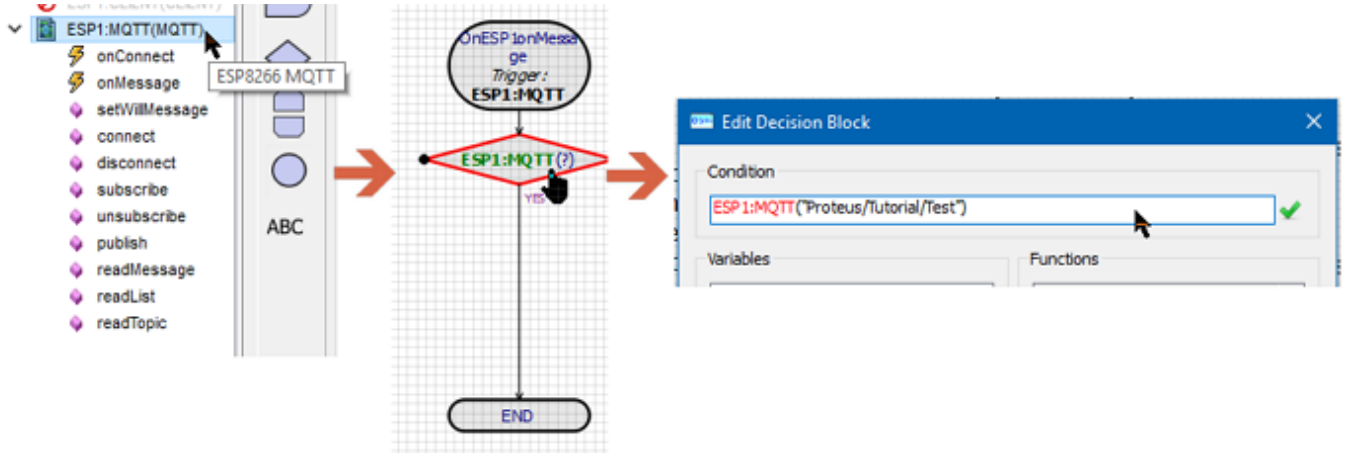
Messages

Finally, we need to decide what to do when a message arrives on a topic that we are subscribed to. We start by placing an onMessage event handler as this is how we'll know that we have a message. Drag and drop the event handler in the usual way onto the flowchart



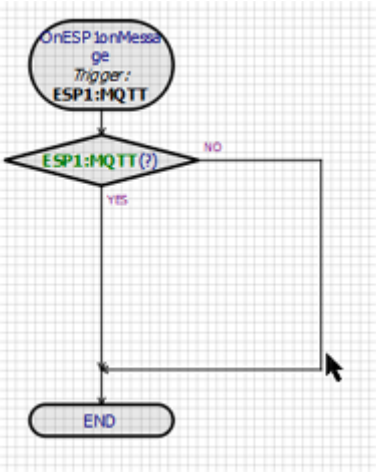
onMessage Event Handler

In an application we may be subscribed to multiple topics so our first job inside the onMessage event handler is to work out which topic the message has come from. The easiest way to do this is to make use of the MQTT sensor function. Drag and drop from the MQTT peripheral itself and you'll see that we get a decision block. Edit the block and enter the topic name. This allows us to gate any message specific logic depending on whether the message belongs to the named topic.



Set up a sensor function to gate messages on the named topic.

At this point it's good practise to draw a wire from the decision to the bottom of the routine to handle those messages not of the topic we're interested in at the moment.



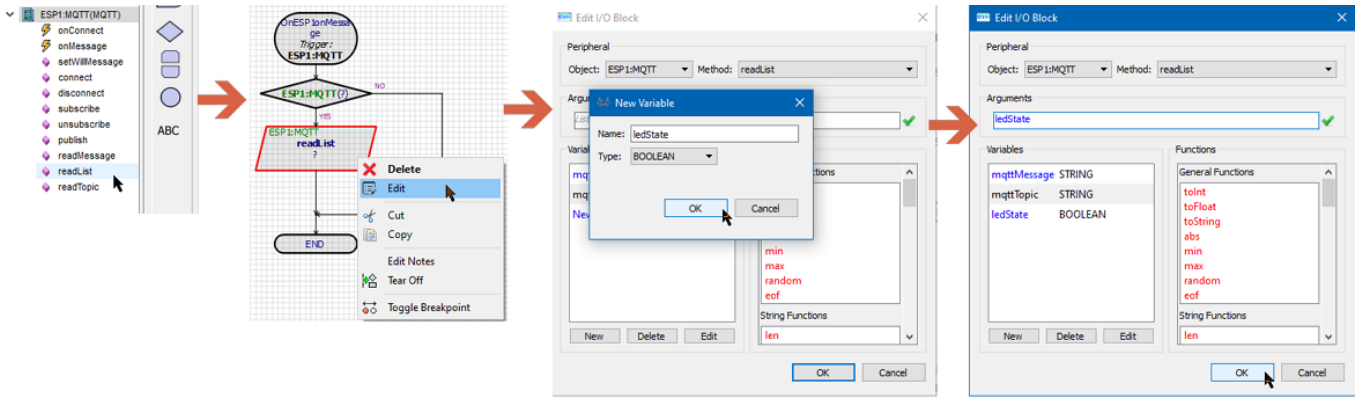
Now, there are two ways in which we can retrieve information from a message, namely:

- readMessage: Retrieves the message as a string into the variable mqttMessage.
- readList: Splits the incoming message at every comma and reads each item into a variable.

readList is really versatile and powerful but you need to know exactly the format of the incoming message. For example, in a sensor report you might have Temp, 190, time, 12:00, humidity, 12.5 and you'd therefore need corresponding variables of type String, int, string, time, string, float, It's also a more dangerous method to parse the message. Having the wrong variable types in your comma separated list is an excellent way to crash your program !!

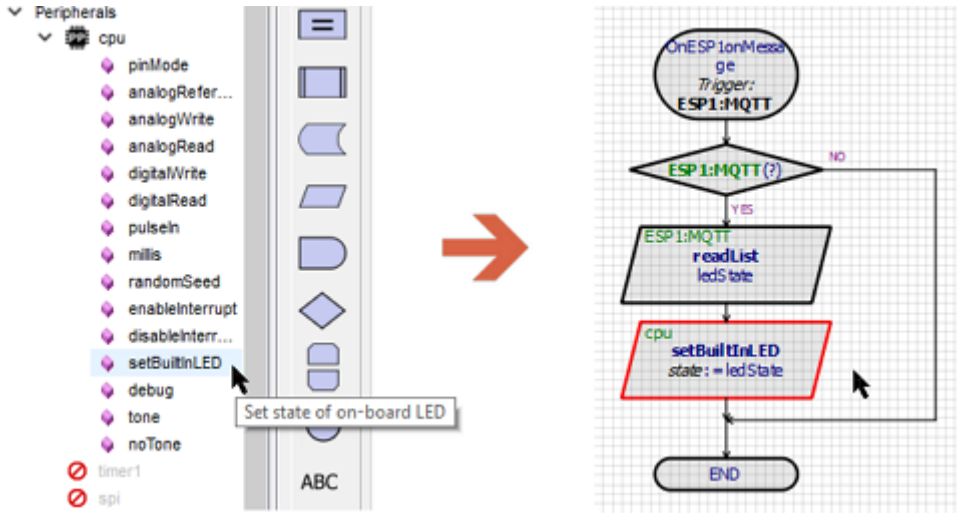
Let's look at a really simple example. We'll assume our incoming messages contain either a 0 or a 1 and that we will turn the onboard LED either off or on depending on the message contents. This is going to be much easier to manage if we treat the incoming message data as boolean which means that we need to use readList instead of readMessage.

Drag and drop the readList method inside the decision gate and then edit the block. We need to create an ledState variable of type boolean and then set that as the variable to receive the incoming message data.



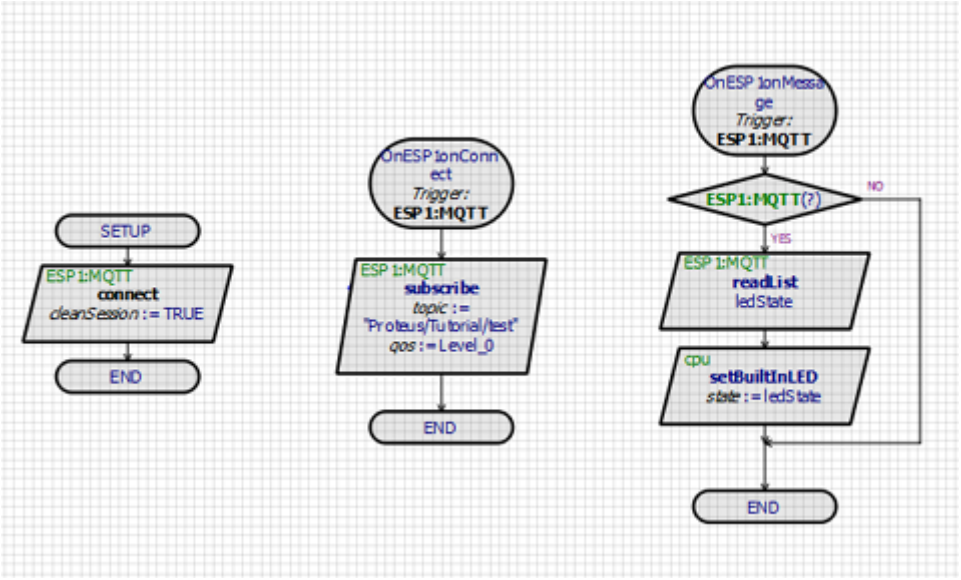
Read the message data into a new boolean variable.

Once we've read in the data we decide what to do with it. In this case, turn the onboard LED either on or off. To do this, expand the CPU section of the project tree and then drag the setBuiltInLED method into place. Then edit the block and set the value to be our variable.



Turn the built in LED on or off depending on the value read from the message.

That's our program ready to go. Your Visual Designer sheet should look something like the following :

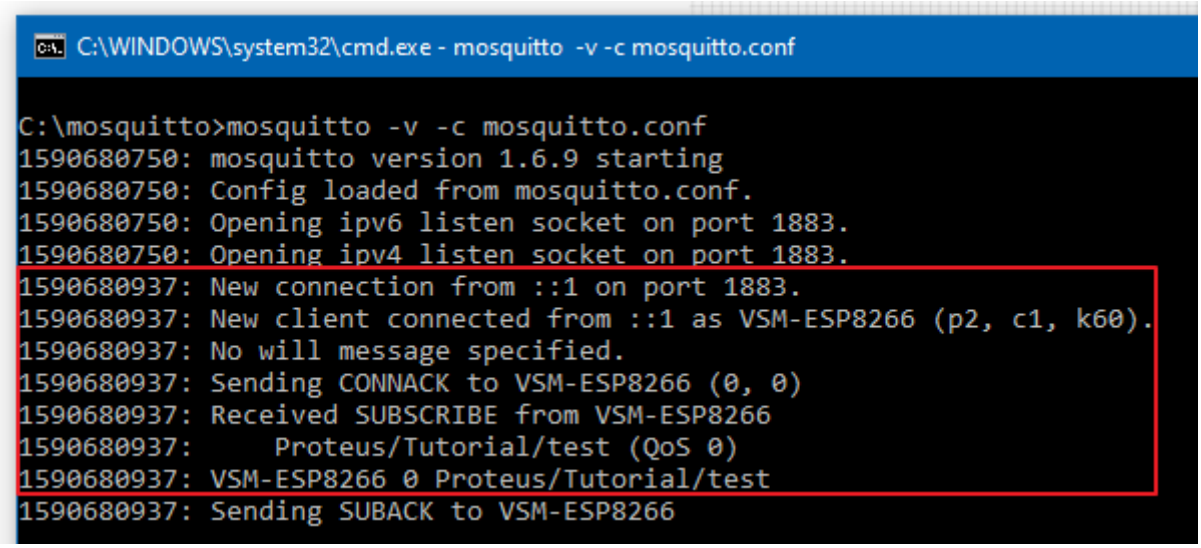


Complete program to subscribe to a topic and turn on/off the LED based on the incoming message data.

Simulation (Subscribe)

Before we run the simulation double check that we have our two command prompt windows up and running. One is running the MQTT broker and the second is running an MQTT console program. If they have been closed down the procedure for launching them is described earlier in the tutorial document.

Now press play to run the simulation and you'll notice the connection being established on the broker log.



Broker command prompt logs activity including the connection and topic subscription from our simulation.

Switch to the MQTT console and let's publish a message to the topic. Type :

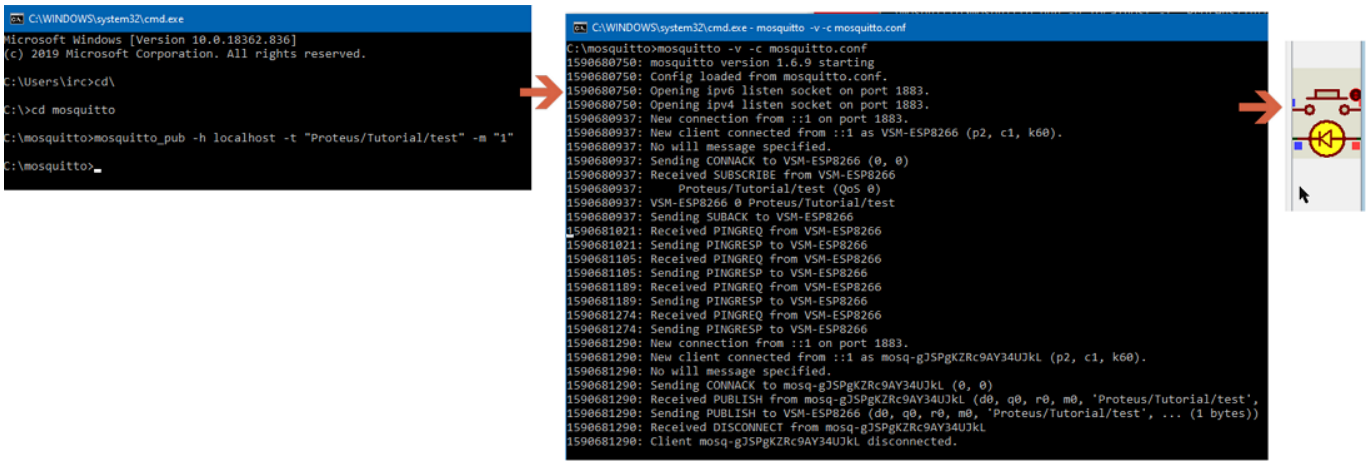
```
mosquitto_pub -h localhost -t "Proteus/Tutorial/test" -m "1"
```

The flags we've used are:

- -h : host location (localhost or IP address).
- -t : topic name
- -m: message

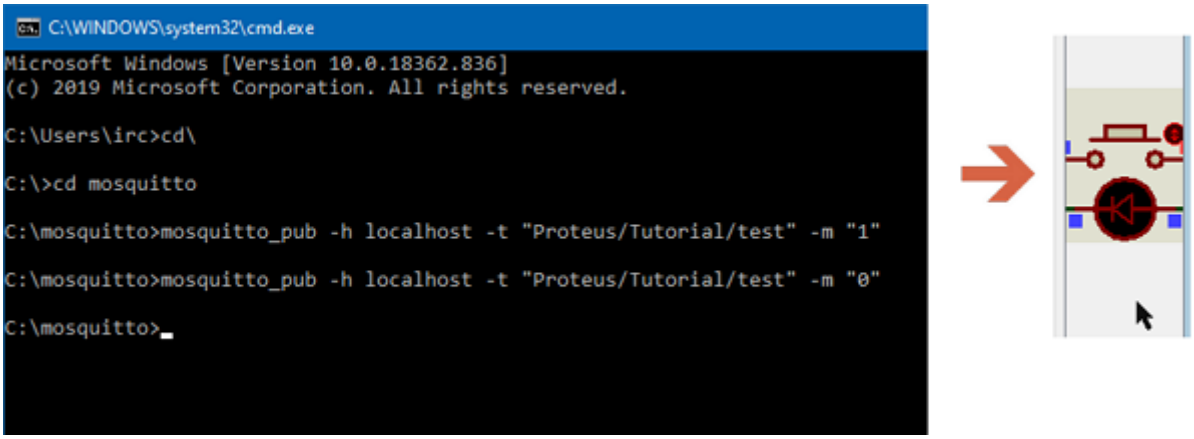
> Remember that strings will be case sensitive !

If everything is working as we'd expect you should see activity on the broker console logging that the message had been published and then the broker sending the message onwards to our client which is subscribed to the topic. Finally, since we sent a "1" you should see the on-board LED in simulation turn on.



Publishing the message is logged on the mosquitto broker and we turn on the in-built LED

Try sending another message with the message as "0" to turn the LED off.



Send a different message to turn the LED back off.

When you are finished stop the simulation and we can close the two command prompts as well. This will shut down the Mosquitto broker and also close our little console app.

Summary

This introductory tutorial was intended to show the basics of how to publish messages and how to subscribe to messages inside a Visual Designer project. We've deliberately avoided using multiple appliances to focus on the basics of using the protocol and have therefore used a command line broker app on the other side of the message.

In a large number of use cases there will of course be more than one appliance. In Proteus, you might typically have a Raspberry Pi controller appliance and one or more Arduino sensor appliances. The next MQTT tutorial covers how to work with multiple appliances in more detail.

Access Control

Introduction

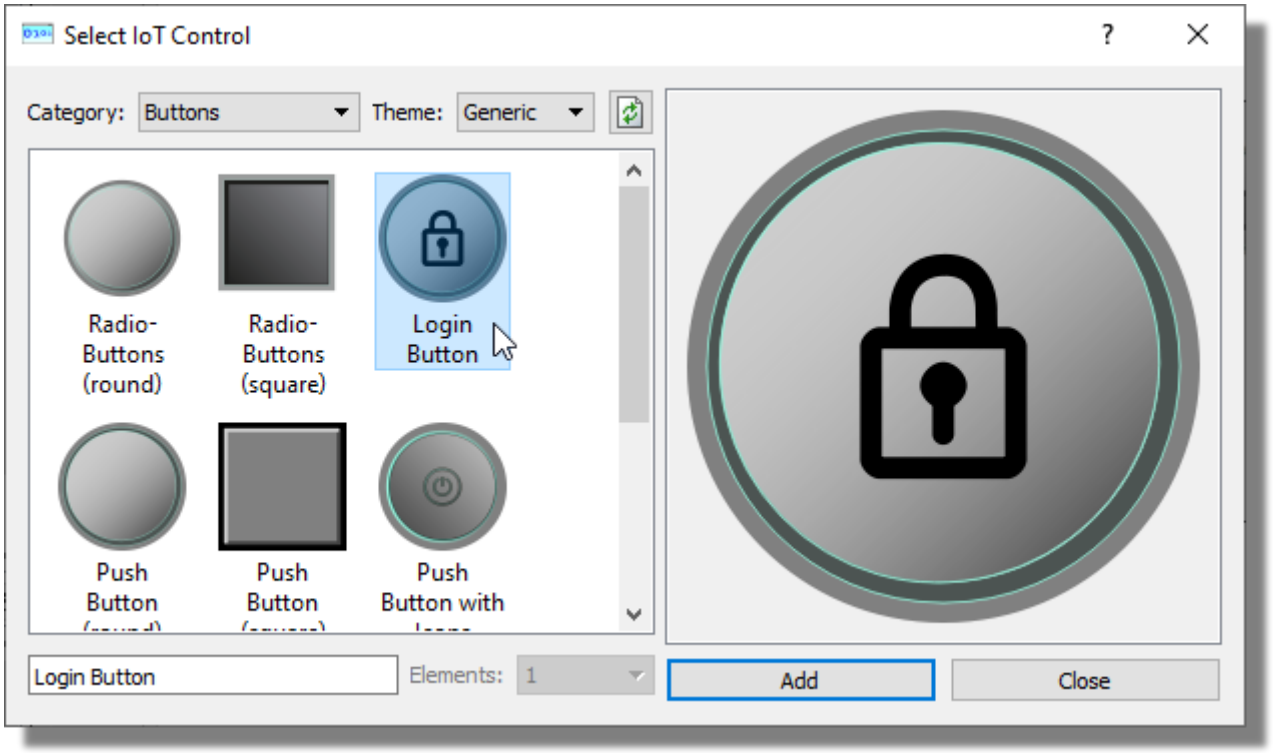
IoT Builder includes the ability to have different levels of access control on the front panel, enabling greater functionality for authenticated users. There are three steps to setting your front panel up with access control, namely:

- Add the login/logout control and set the password.
- Define the behaviour of the other controls when not authenticated.
- Handle the login and logout events in the firmware program

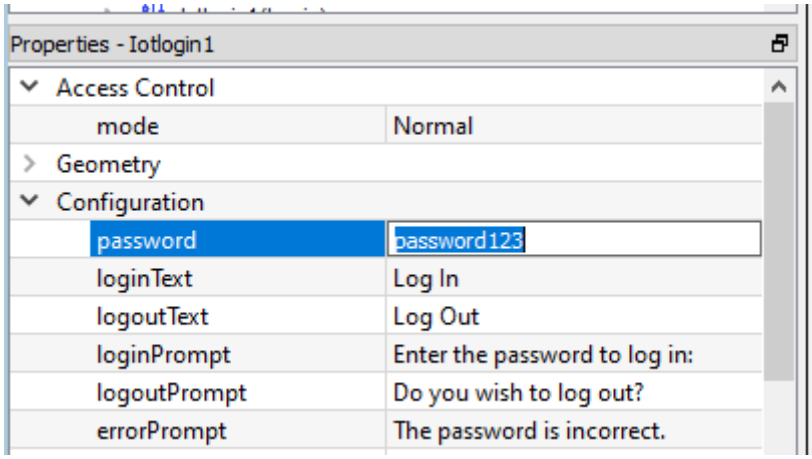
The Login Control

To set up a login control:

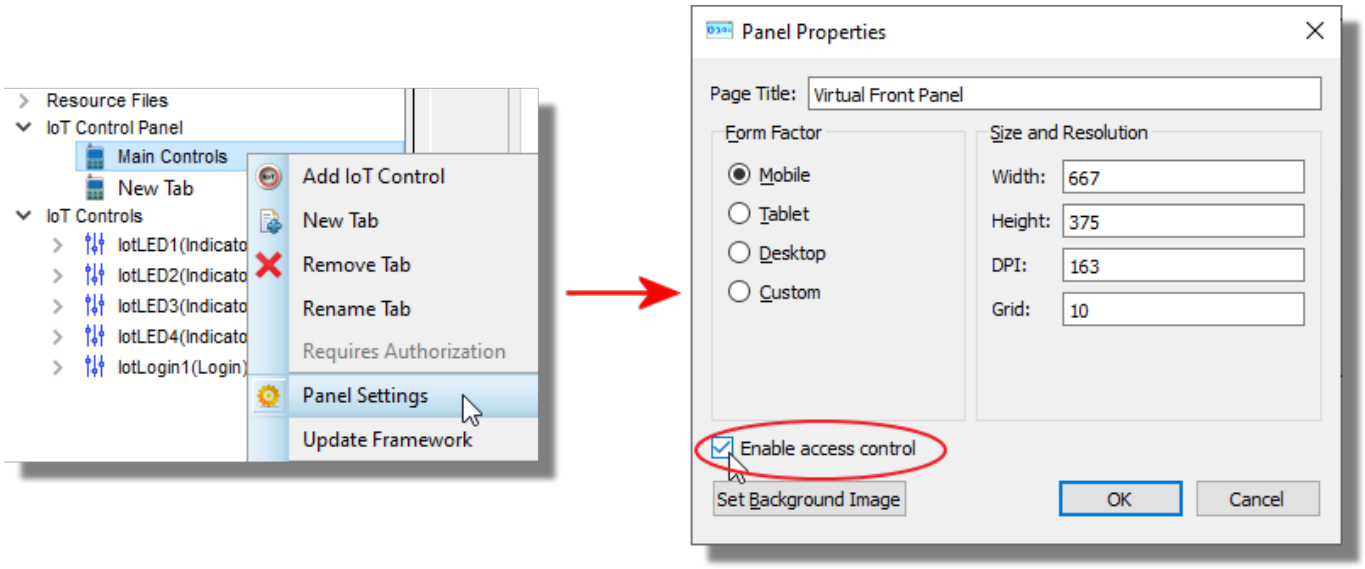
1. Add the login button from the 'Add IoT Control' command on the Project Menu. If you haven't already got a front panel one will be created for you and appear in the Project Tree.



2. Place the Login button on the front panel, make sure it is selected and then change the password and prompt text as required.



3. Turn on the Access Control. This is done by right clicking on the Panel > Panel Settings > Enable Access Control:



Access Control Behaviour

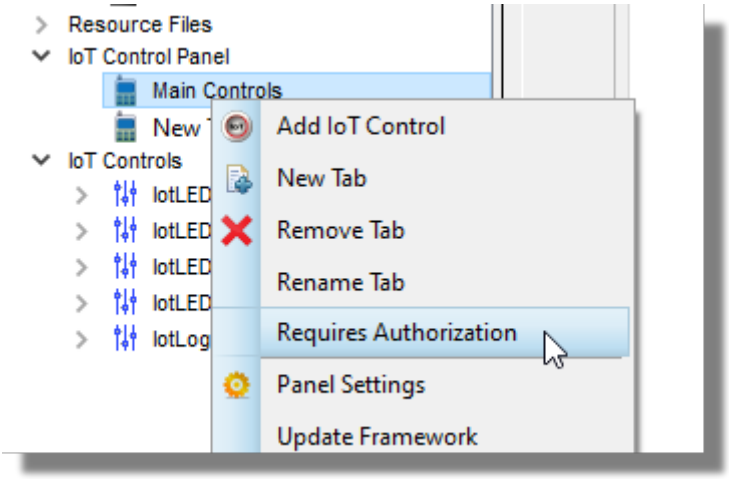
Once you have specified that you want access control you need to decide what the behaviour of the tabs and other controls on the front panel will be.

Tabs

A tab that requires authorisation will be hidden completely, along with all controls on the tab until you are logged in.

To require authorisation for a tab:

Right click on the tab in the Project Tree and select 'Requires Authorization':

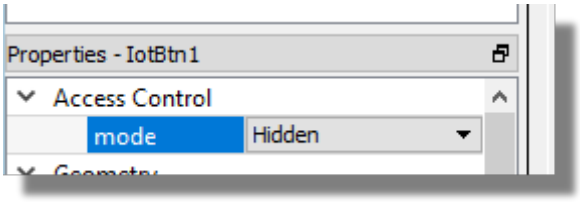


Controls

All controls default to normal behaviour when logged out. This means that they are both visible and enabled. You can change the state of a control to be either disabled or hidden via the property page for that control. This setting determines the behaviour of the control when a user is logged out.

To require authorisation for a control

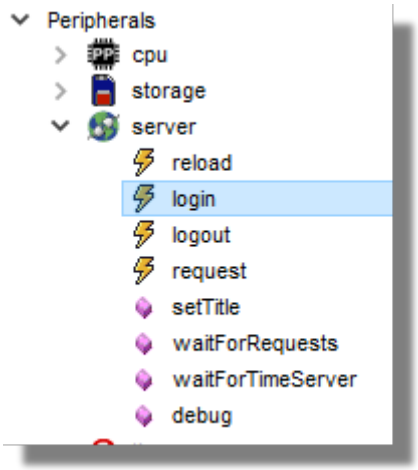
Select the control in question (already placed on the panel) and change it's access control mode property to be either hidden or disabled.



> This is the mode that the control is in before you login. Once logged in, the control will return to be in state 'Normal'. So for example, if you wish for a switch to be shown only when you are logged in, you would place a switch on to the Panel, then set it to be hidden. It will then only appear once you have logged in.

Firmware

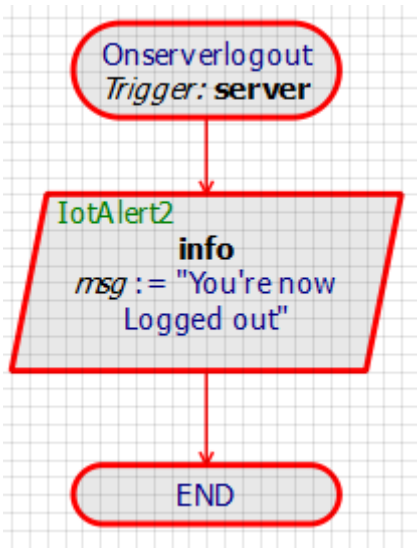
When the front panel is configured for access control you have a working system. Users can login/logout with a password and features in the user interface will become available based on user status. However, it's often useful for the firmware program to also know the status of the user and this is handled via two server events, namely the login and the logout events.



These can be dragged and dropped onto the flowchart in the usual way. The login event will be called automatically on a successful login and the logout event will be called automatically on a successful logout. Then if your program

needs to know about user status you might set up an 'isAdmin' variable that you initialise to FALSE in the setup routine, set to TRUE in the login event and back to FALSE in the logout event.

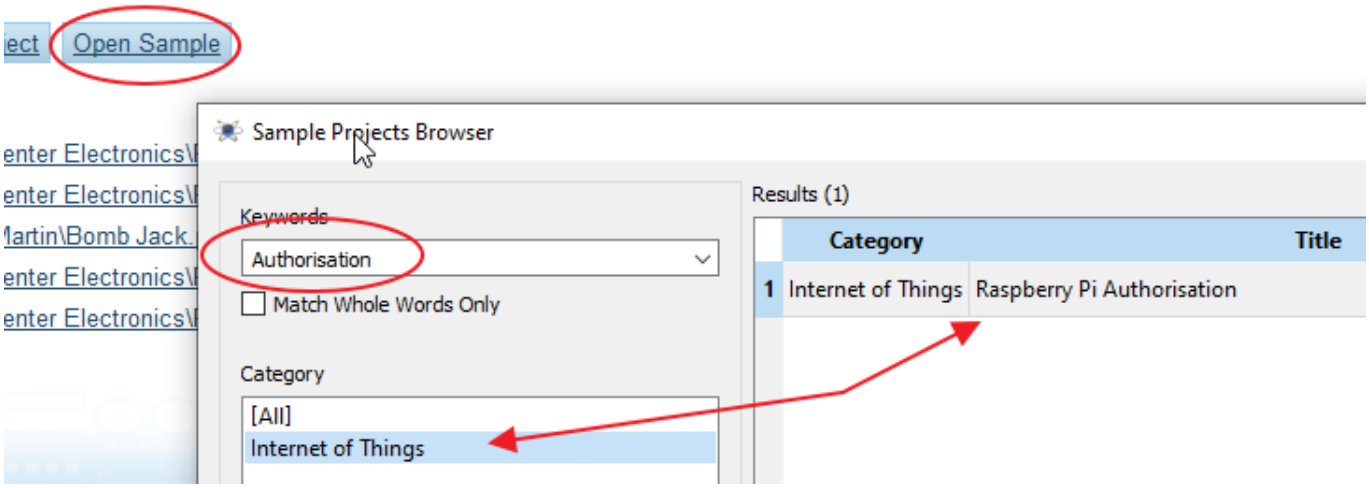
Or, if you would like to display a message stating that the user has successfully logged out, you just add an info box into the logout event.



The server Reload method will force a user log out.

Simulation

Press Play to run the simulation and you should see that your tabs that are authorized are not displayed, and any controls that are set to be hidden are also hidden. Click the Login button and enter the password you have set (password123 if you have not set one) and you should then see all tabs and controls in the project. A ready-made example project can be found in the sample designs:



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. Both are linked below.

- Visual Designer Manual : Raspberry Pi Configuration
- Tutorial Movie : Raspberry Pi Configuration

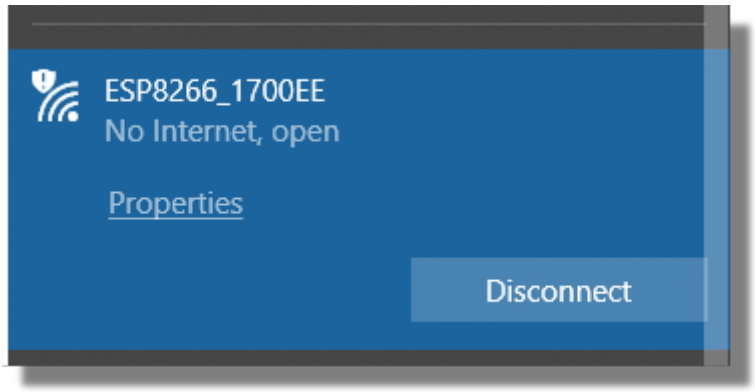
Programming

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

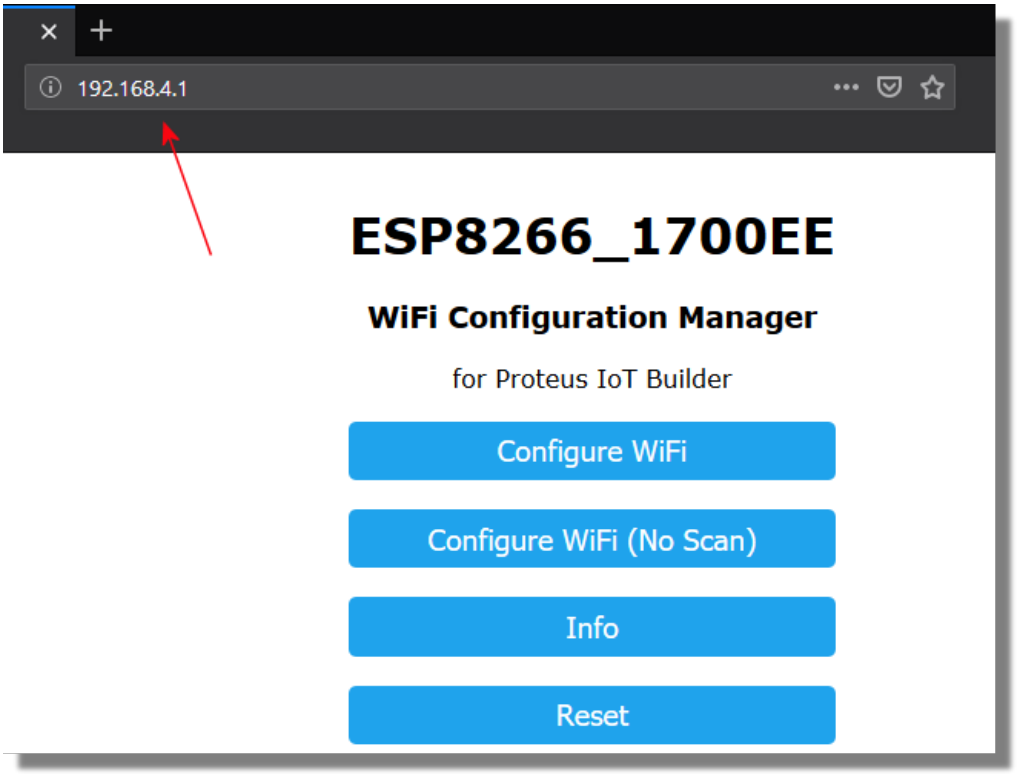
- Visual Designer Manual - Raspberry Pi Programming.

Proteus Playkit

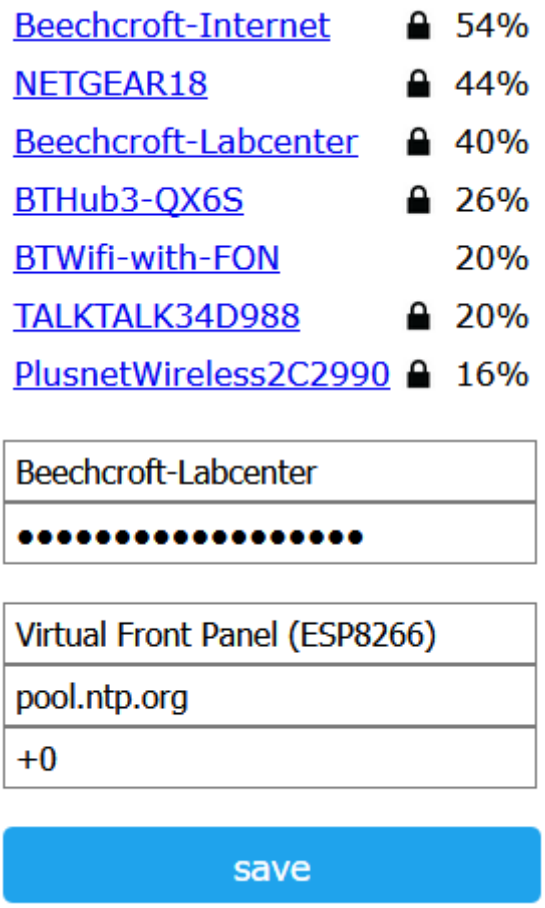
With the Playkit powered on, you can connect your PC's WiFi to the temporary Access Point that it has created.



If you then point a web browser to the IP address 192.168.4.1 you should see a screen similar to the one below:



Click Configure WiFi and select your main WiFi network from the list. You will also need to enter the WiFi password if such is required:



If this screen does not show properly the first time, try refreshing page in your browser

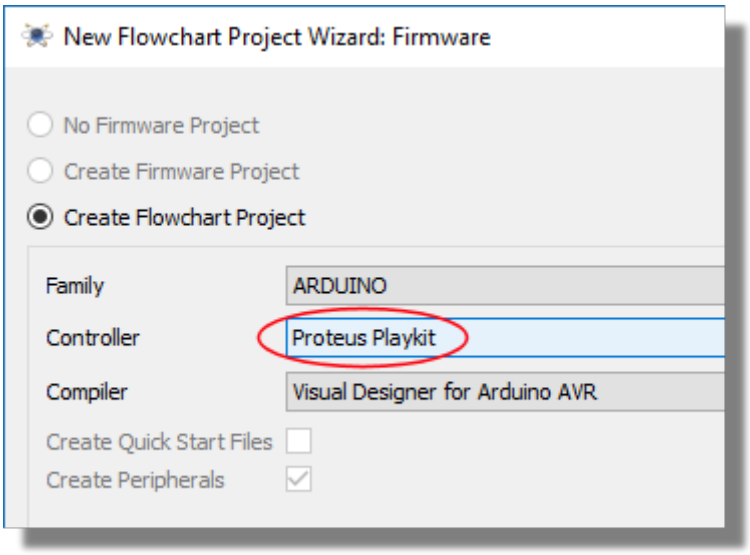
The three fields at the bottom allow configuration of the appliance name, a time (NTP) server and a time offset in hours. You should set this last value to match your current time zone. When you are done, click save. The Playkit

ESP8266 LED should blink for a short while and then become steady. This indicates a successful connection to your WiFi network and you are now ready to wire up the hardware.

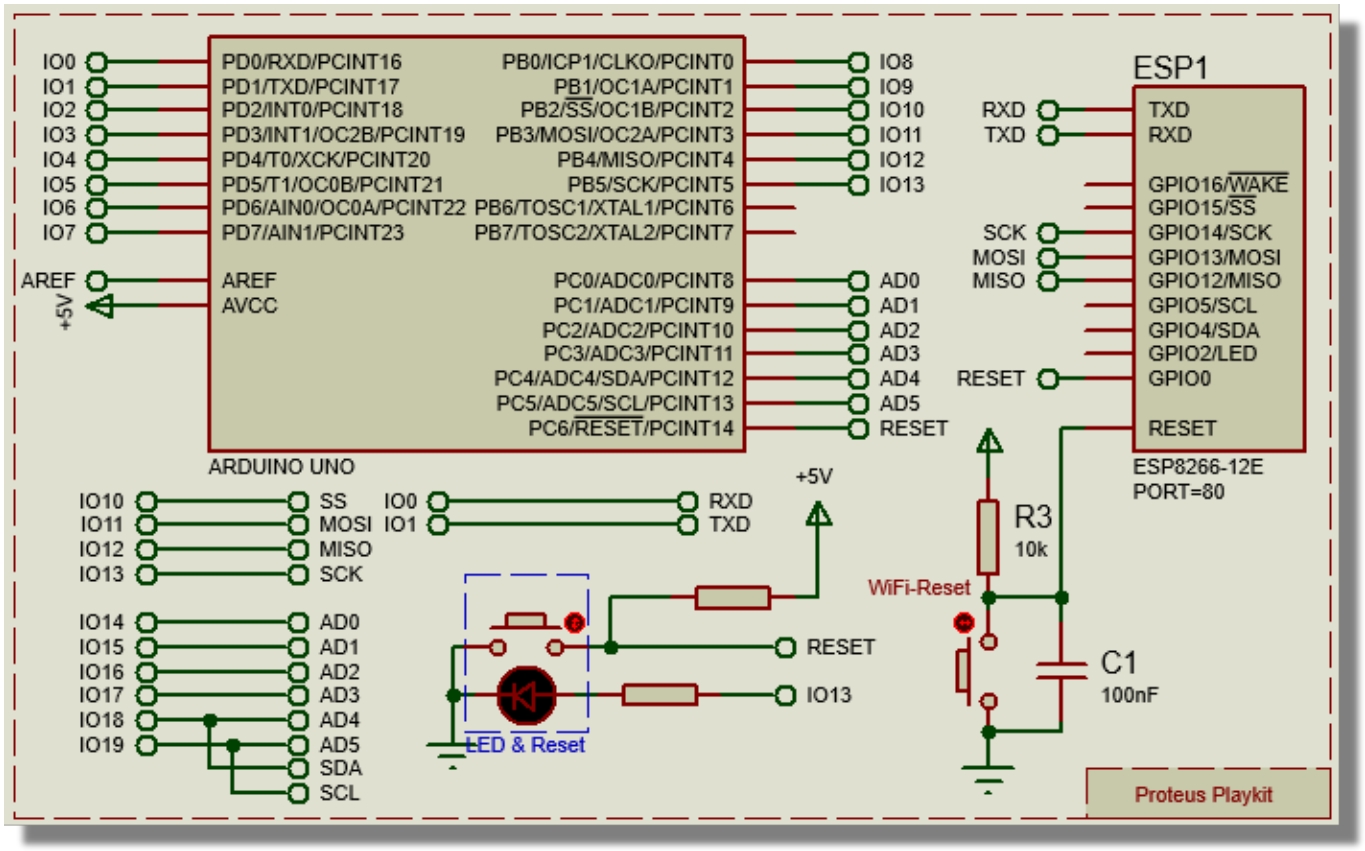
Note that if the appliance fails to connect to the configured WiFi network at start up, it will automatically enter the end-user configuration mode. This makes it possible to re-configure the WiFi settings at a new location, or if either the SSID or password for your network is changed.

The hardware is now ready to be programmed from Proteus. Whereas the previous steps need to be carried out just the once, this last step can be repeated as many times as you like during the process of developing the appliance firmware. Of course, you can also simulate the appliance within Proteus VSM so, in most circumstances you would expect to deploy to the real hardware only right at the end of the software development process.

With your PC re-connected to your network, Open Proteus and select the New Flowchart option. Make sure you set the family to be Arduino and the controller to Playkit.

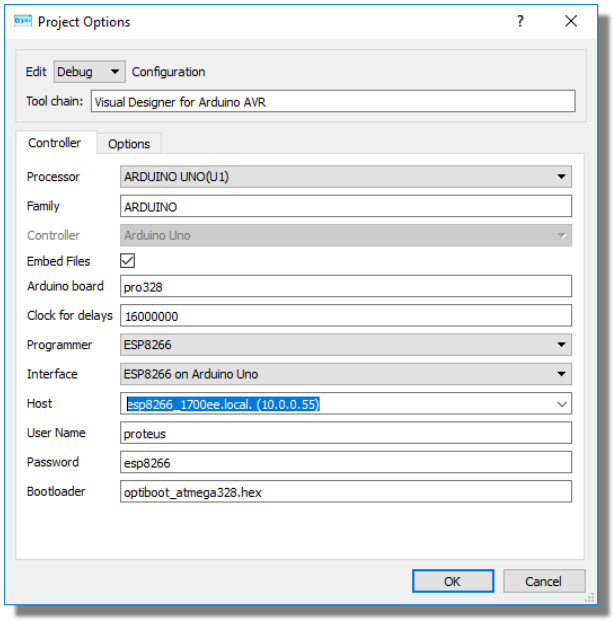


The resulting project will place an instance of the Playkit base board on the schematic for you. The port number can be set as you like and will be needed for both simulation and deployed use.



From the Visual Designer tab, go to the Project Menu > Project Settings to configure IoT Builder for programming. Change the *Programmer* option to ESP8266 and the *Interface* to ESP8266 on Arduino Uno. More options will now

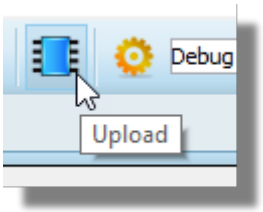
appear. The host option allows you to select the appropriate Playkit from those on your network. Each Playkit is uniquely named and has a sticker on the bottom identifying it.





This name is also the address to use when connecting a browser to the Playkit after deployment of the firmware in the format <name>:<port> e.g esp8266_1700ee.local:80

Once a project is ready to be deployed, it can be uploaded from the top control bar.



Proteus will give you the results of the process in the Output window.

```
avrdude.exe: AVR device initialized and ready to accept instructions
avrdude.exe: Device signature = 0x1e950f (probably m328p)
avrdude.exe: NOTE: "flash" memory has been specified, an erase cycle will be performed
         To disable this feature, specify the -D option.
avrdude.exe: erasing chip
avrdude.exe: reading input file "C:/Users/[redacted]/AppData/Local/Temp/VSM Studio/c6ecc5d481134e33b27670a9dc0915d3/ARDUINO UNO/Debug/with.bootloader.hex"
avrdude.exe: writing flash (32768 bytes):
avrdude.exe: 32768 bytes of flash written

avrdude.exe done.  Thank you.
Firmware upload COMPLETE.
```

> If for any reason you are struggling to upload your firmware or cannot connect to the Playkit, please see the Troubleshooting section.

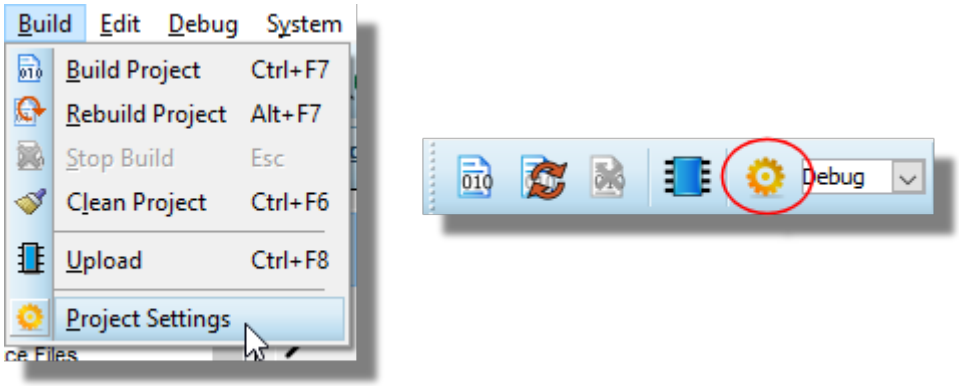
Arduino Yun

Arduino Yun via SSH

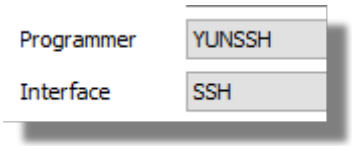
The Arduino Yún contains both an Atmel ATmega32u4 and an Atheros AR9331. Digital pins Digital pins 0 and 1 are used for serial communication between the two processors. Your program firmware will execute on the Atmel CPU while the Atheros processor (running Linux) will host the webserver, front panel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

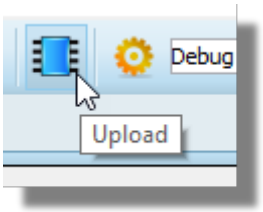


Next, set the programmer to be Yun SSH and the interface to be SSH.

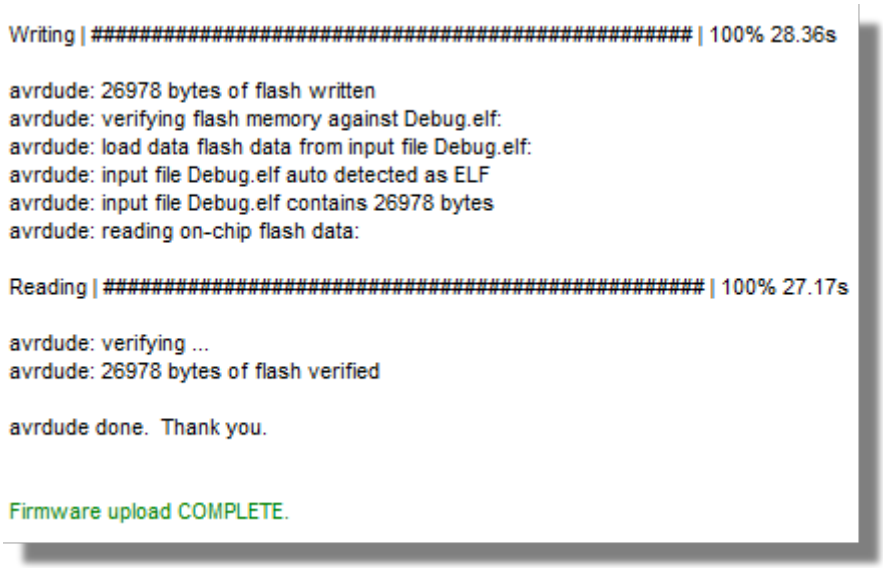


The default address, user and pass are pre-populated but will need changing if the configuration of your device has been changed. If they have been changed you will need to type in the name of the host (arduino.local.) in to a browser, enter in the password and change and then in the settings set up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



How does it work ?

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

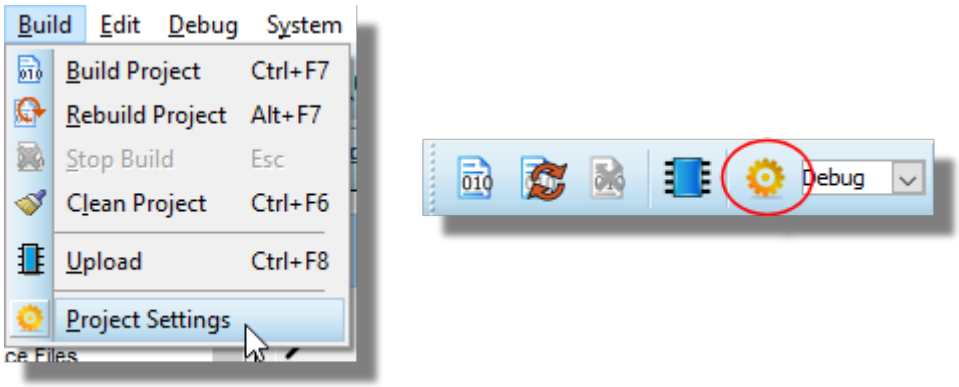
1) The resources are transferred to the target path folder (/vsm by default) on the Atheros.

2) The firmware is transferred to the /tmp folder on the Atheros.

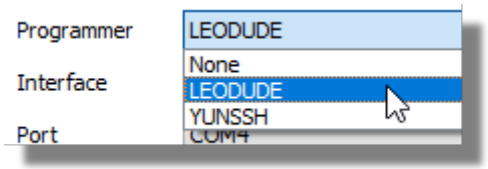
3) AVR-DUDE is invoked (on the Yun side) to program the firmware into the AVR. It does this using the ICP interface (SPI).

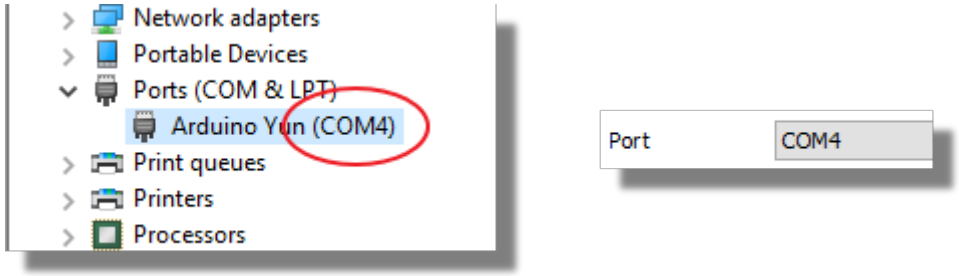
Arduino Yun via USB

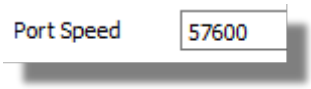
Similarly to the SSH method or programming, configuration of the project takes place from the project settings dialogue. First launch the dialogue form either from the Build menu or from the icon at the top of the editor.



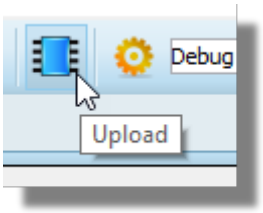
Set the programmer to be LEODUDE and the interface to be Arduino Yun.







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.

> 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.

> 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 is useless as an internet of things appliance. Check with your dealer before purchasing a Yún if you believe you may live in such a country.

Setting The Arduino Yun On To Your Network

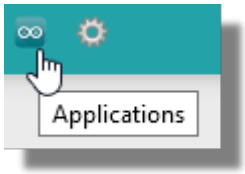
1) Power the Yun with a USB cable

2) Plug an ethernet cable in from your router / network point

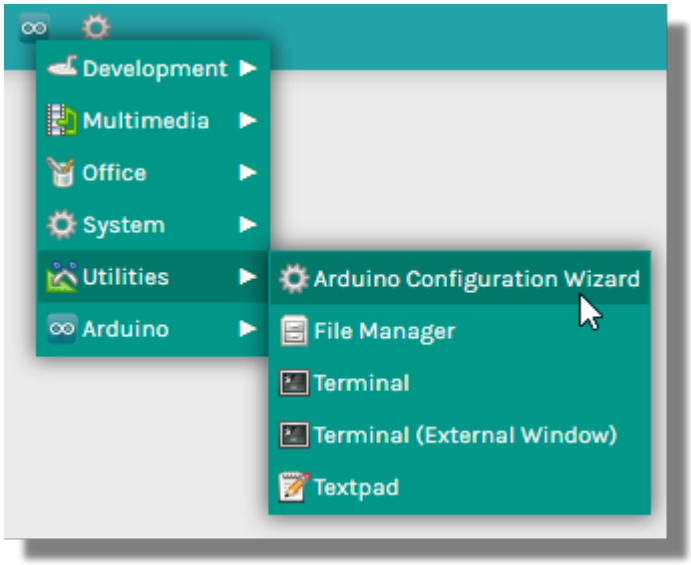
3) Open a web browser and type in its host: arduino.local.' (including the last full stop). This will prompt you for a password

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

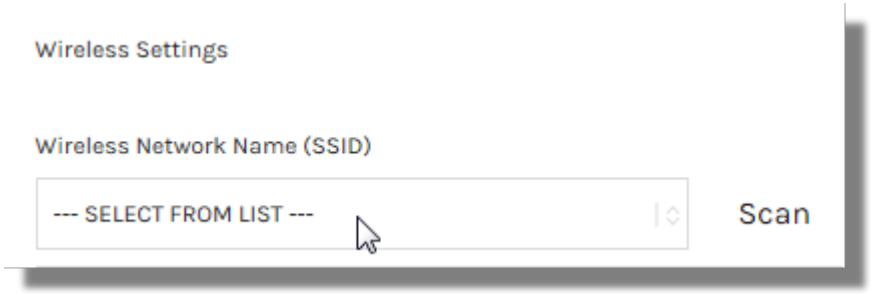


6) Go to the Utilities > Config Wizard



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.

Arduino Uno + ESP8266

Programming the WeMOSD1 will only work once the hardware has been assembled and the ESP8266 has been re-flashed with the Labcenter VFP Server. A full setup guide is provided via the link below:

- Building an ESP8266 Server

ESP8266 Programming : Troubleshooting

Problems with WeMOS D1 Programming

See Also:

- Building an ESP8266 Server

Cannot get a COM Port

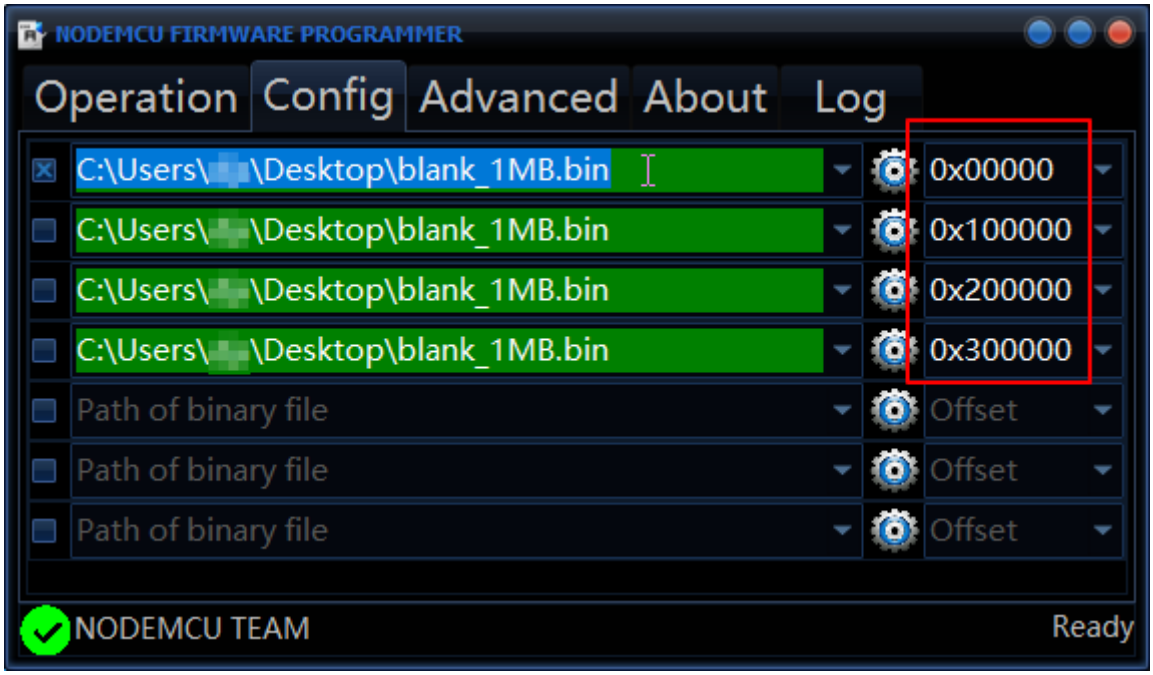
If you have opened the NODEMCU Flasher and no COM port appears, this is most likely to be that you do not have the CH340 drivers installed. You can download the drivers from our website: https://www.labcenter.com/downloads/ Extract them to a folder on your hard drive, and in the Device manager, Right click on the *Unknown device* and select Update Driver. Select Install driver from specific location and navigate to the saved driver files. Click next to install the drivers.

Erasing the flash

If you Flash the WeMOS and the Wifi Modules is not visable, it is likely that there was data already in the Flash. This will need deleting before you can reflash the WeMOS.

Download the *Blank_1MB.BIN* file from our website: https://www.labcenter.com/downloads/

Use the NODEMCU Flasher and open the Config tab. Enter in the Bin file in to 4 pages to wipe ALL the data. Use the following Addresses:

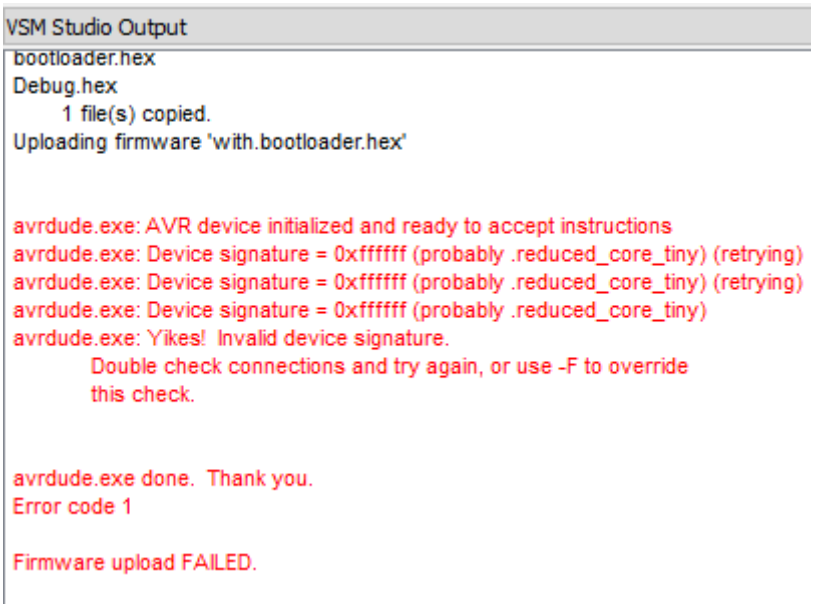


Note that you can only select 1 row to be flashed at any one time, you start at the top and then in the Operation tab Flash the first page, then uncheck page 0 and check page 1, then flash that. Do the same for page 2 and 3.

Once that is complete the Flash is empty, now you can upload the Labcenter BIN file to the flash (C:\ProgramData\Labcenter Electronics\Proteus 8 Professional\VSM Studio\drivers\arduino\Vfp8266\firmware)

Programming Error 1

If you get the following:

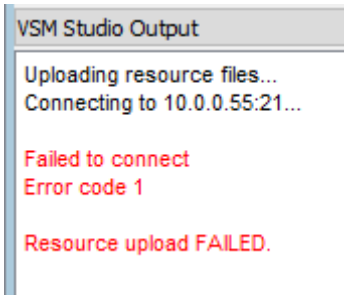


It is likely that the wiring on the breadboard is incorrect.

Playkit or WeMOS Programming

Programming Error 2

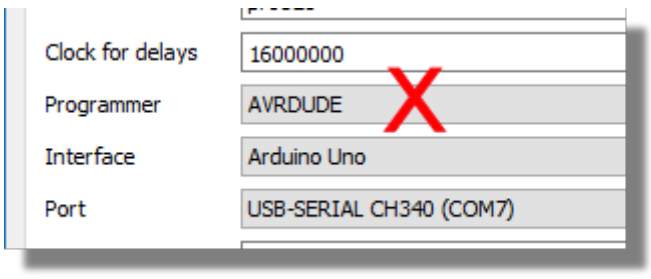
If you get the Following:



1. Check that the hardware is plugged in.
2. Check that the Settings in Project menu > Project Settings are set correctly.
3. Try rebuilding your projects and uploading again.

Programming Error 3

Check the programming options to make sure ESP8266 option is selected, Not AVRDUDE:



Playkit Troubleshooting

Resetting the Playkit

If there are Wifi changes or password mistakes or general problems it can be useful to reset the Playkit hardware. Do this by holding down the AP RST button whilst pressing the 8266 RST button. This will clear the network information and allow you to start the process again. This process is for the V1.3.180625 version of the Playkit (on the reverse of the board) and may change in later versions.



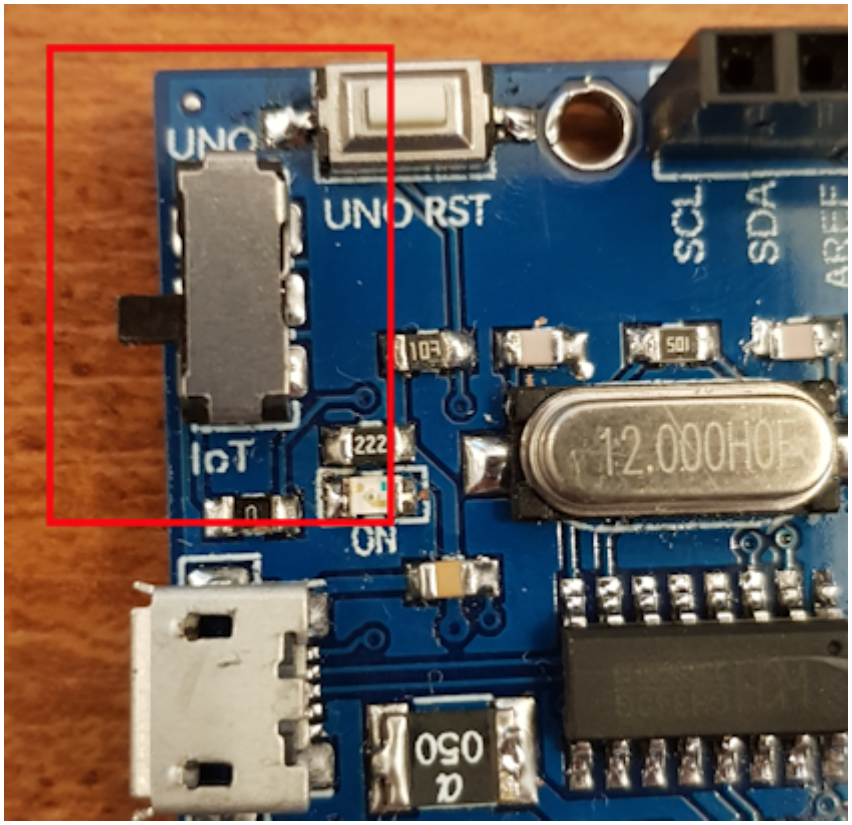
Proteus not recognising Playkit in host option:

1) Check LED on ESP8266. If it is flashing repeat the connection to network operation.

2) Check Playkit and PC are connected to the same network.

Playkit is recognised but fails to upload resources

Make sure the Playkit is switched to IoT instead of UNO. When the switch is set to IoT there is a programming path through to the ESP8266 which allows the resources (notably the front panel graphics) to be stored. When set to Uno the playkit reverts to standard header configuration for Arduino Uno.


UNO
UNO RST
SCL
SDA
IoT
ON
12.000HOE

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.

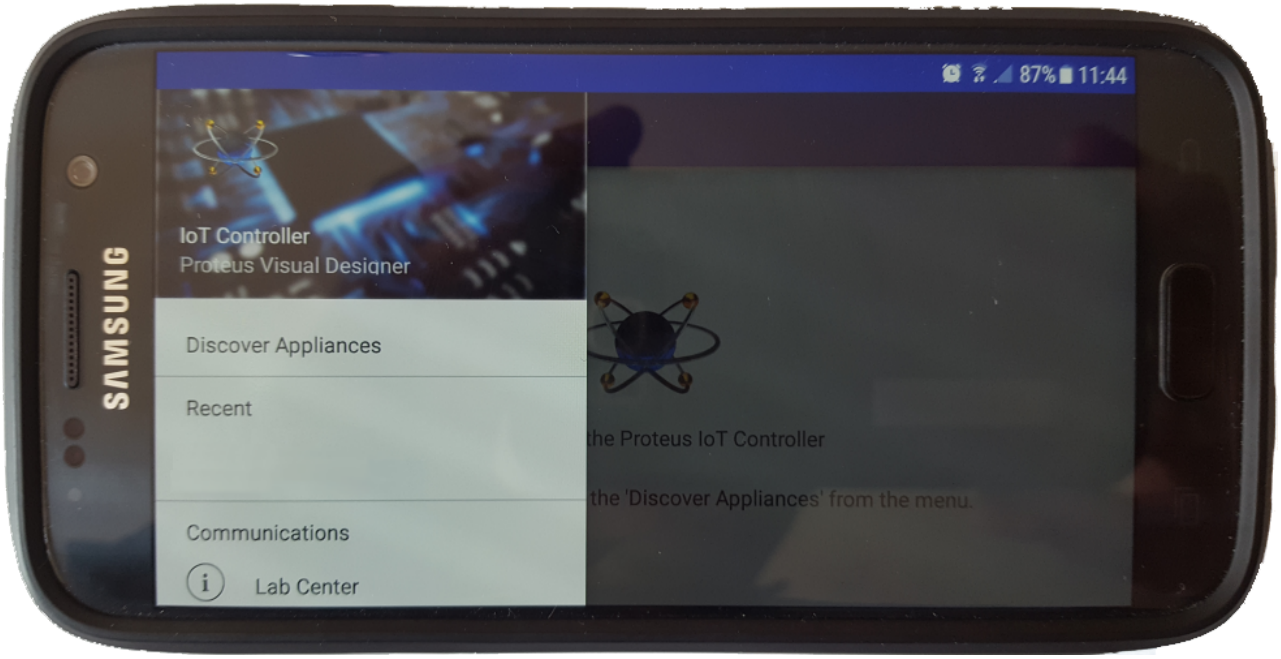
> You need to have the Bonjour utility installed on your PC in order for the App to control the running simulation. If it is not already installed you can install it directly from the Labcenter Program Group on the Start Menu.

Discovery

There is one menu which is at the top left:



This will open the following menu:



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 loads 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.

ℹ 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.

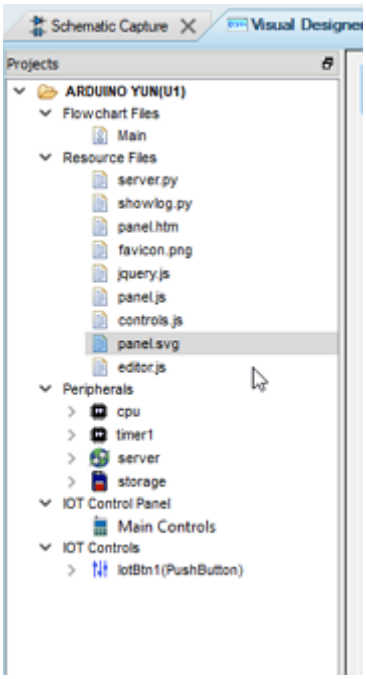
Using Inkscape to Edit the Panel

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.

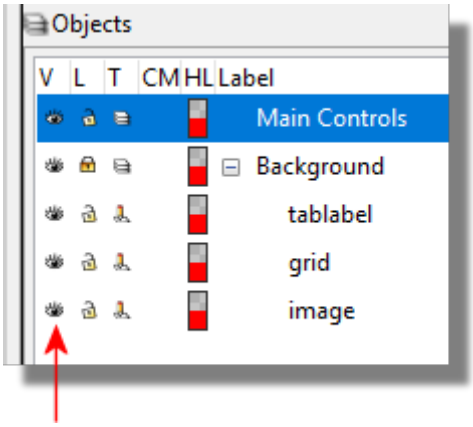
Assuming that you have created a file association between InkScape and SVG files, then double clicking panel.svg under the resource section of the Project Tree will launch InkScape to edit it.



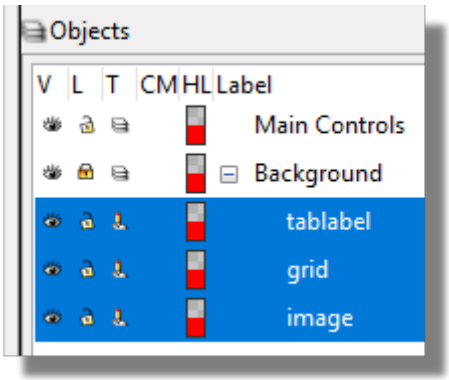
Double Click on the panel resource in Proteus to launch it in Inkscape.

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.

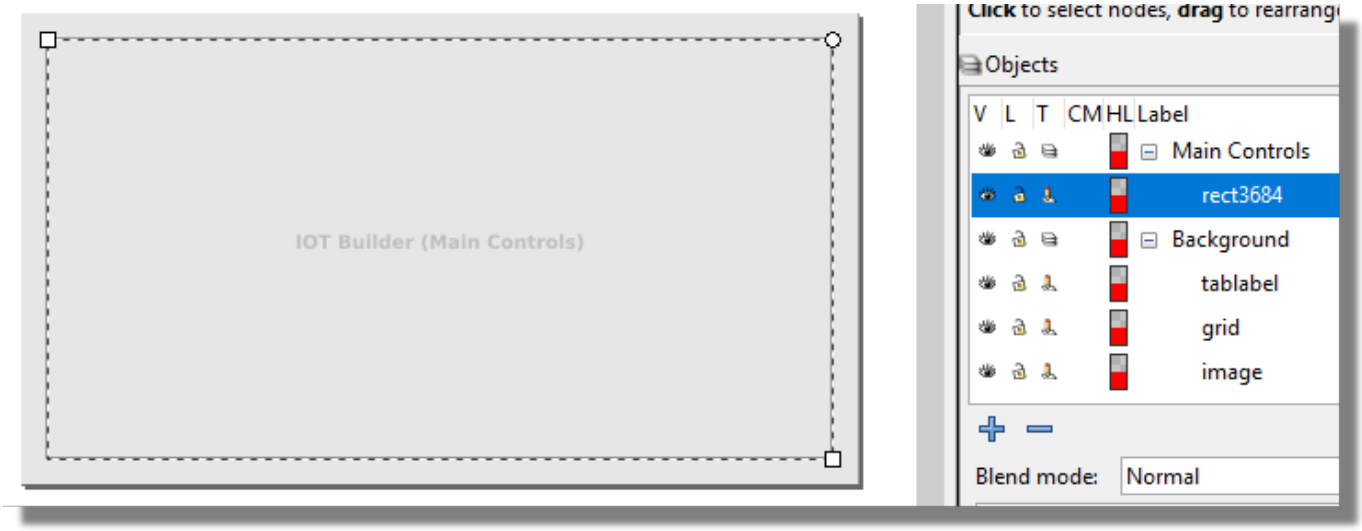
When you first open the panel.svg file in InkScape, you'll see that each tab of the panel is held on a separate layer, these corresponding to top level groups in the SVG file structure. Note that you may need to use the visibility flags in the Layers Dialog to show the other tabs.



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

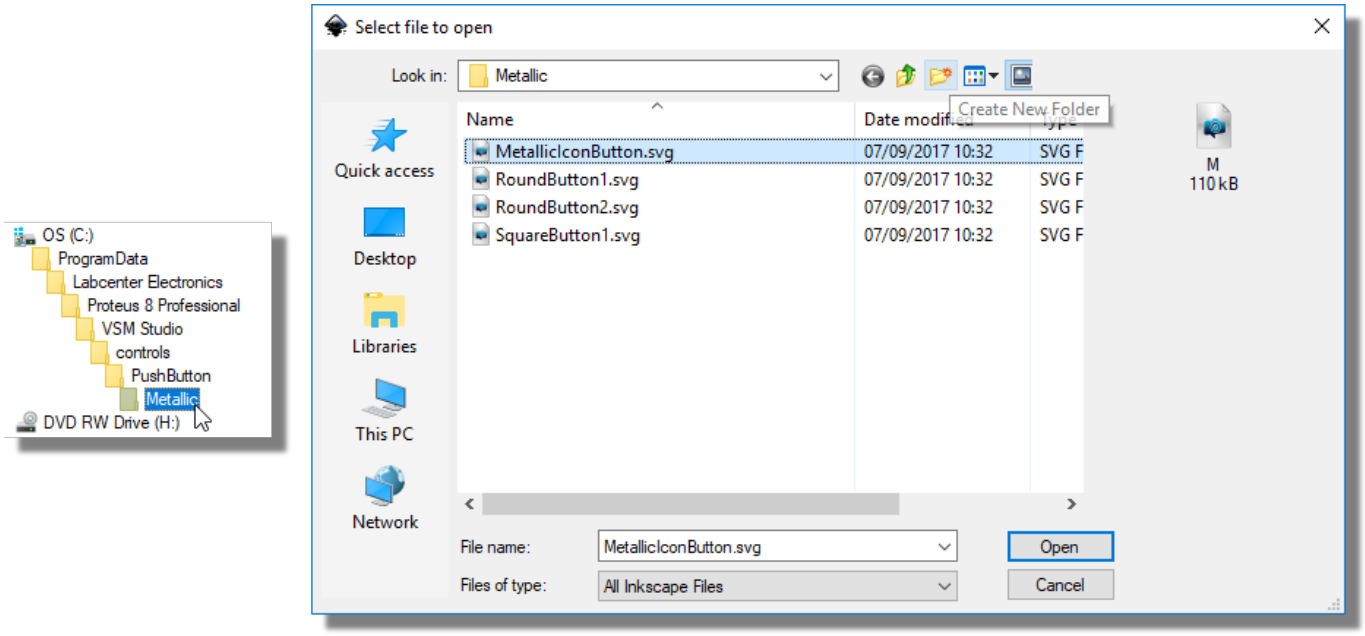
Unlike editing of the panel, the editing of controls is quite involved and requires an understanding of how the control is constructed. If you are interested in creating your own controls then please e-mail us at support@labcenter.com and we can provide you with more detailed technical information.

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

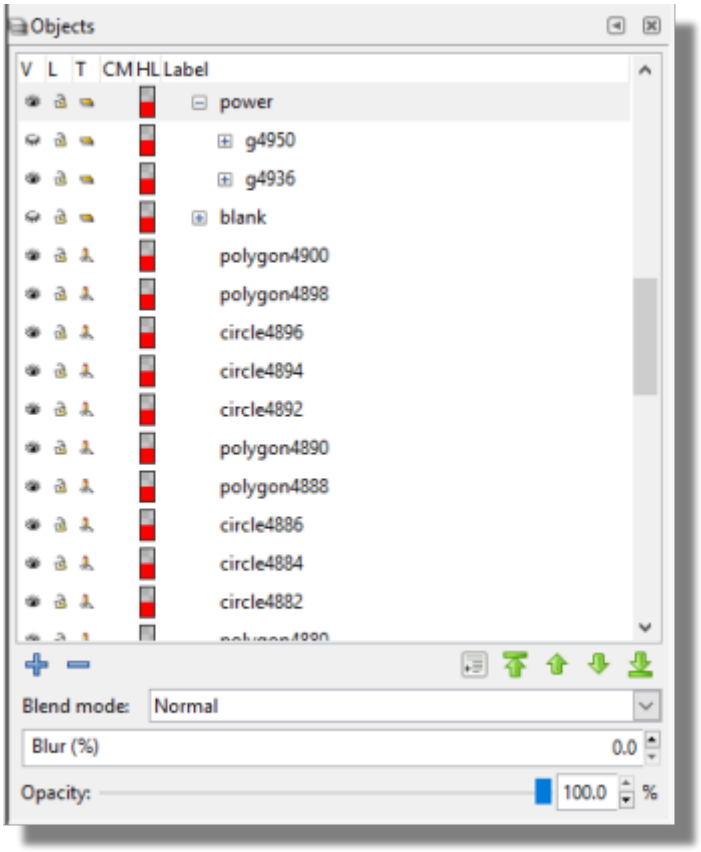
We'll assume at this point that we are in the IoT Builder editor and have added an icon button to our panel. We've looked through the different icons and actually we want a happy face icon in the button. This isn't here but we do have custom icon option so we can make our own

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.



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

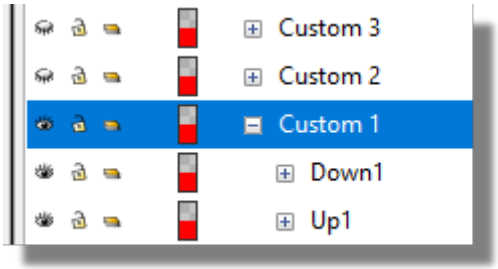
Open the objects panel to view the root structure of the SVG file. Expand the structure tree for a full view of groups and layers.



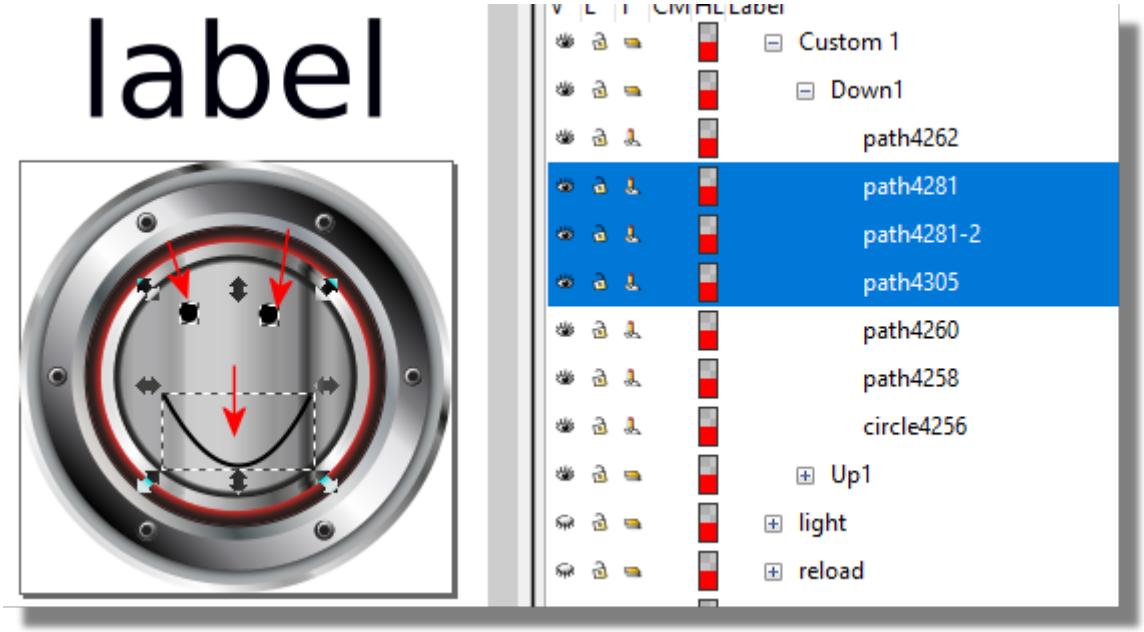
Do not ungroup any objects. If you do so the control will either not load in Proteus or not work properly if it does.

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.

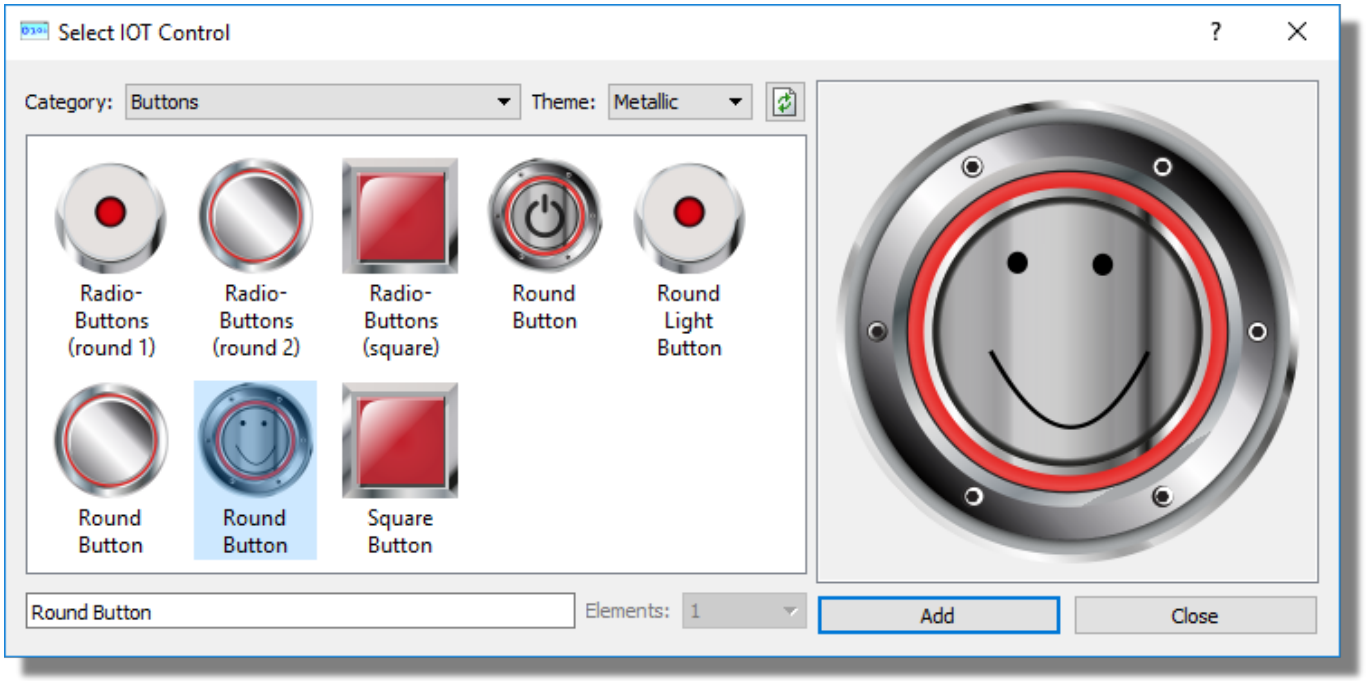


Here you can see the arrows pointing to the added graphics, with them selected on the right hand side:



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:



ℹ 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.