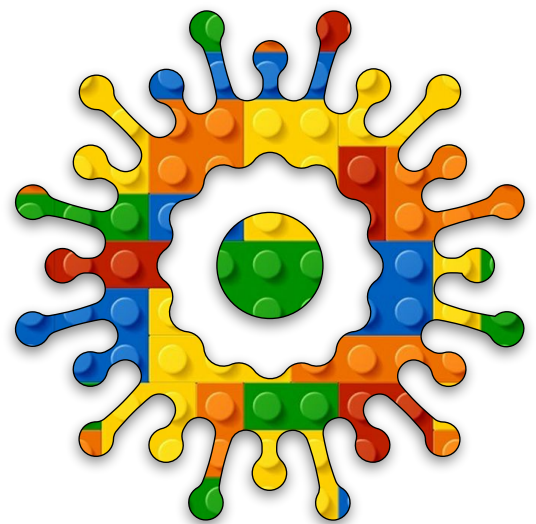


Internet of Things Module B Unit #3 button ON





Module B Unit #3 button ON

The Thing

The Sketch

Sketch B3.1 the full sketch

Sketch B3.2 adding in the new code

Sketch B3.3 simplified sketch

Upload

Dashboard

The Status widget

Customising the widget

Custom section

Toggle widget

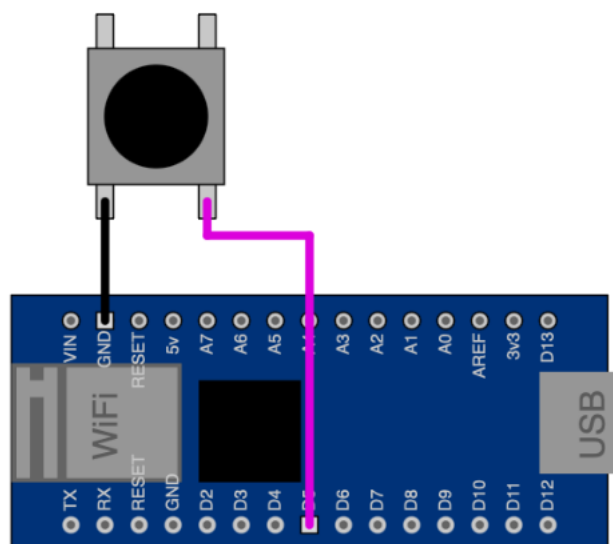


Introduction to button ON

For this unit, we will have a button on pin D5, as we did with the **module A unit #8 the button**. We don't need the LED traffic lights for this unit. We will be sensing when the button is pressed and released with a widget that returns on/off, true/false, or coloured icons. For this scenario, we will press the button to speak and release the button to mute; it will make sense at the end, trust me.

The circuit diagram is shown below.

Figure 1: button wiring diagram





The Thing

- 1 Delete the old **Thing** and create a new one, call it **myButton**.
- 2 Add a boolean (**bool**) variable and call it **button**.
- 3 Click **Read Only**, then click **ADD VARIABLE**.

Figure 2: boolean button and Read Only

Add variable

Name

button



Sync with other Things



Boolean eg. true



Declaration

`bool button ;`



Variable Permission



☐ Read & Write

☒ Read Only

Variable Update Policy



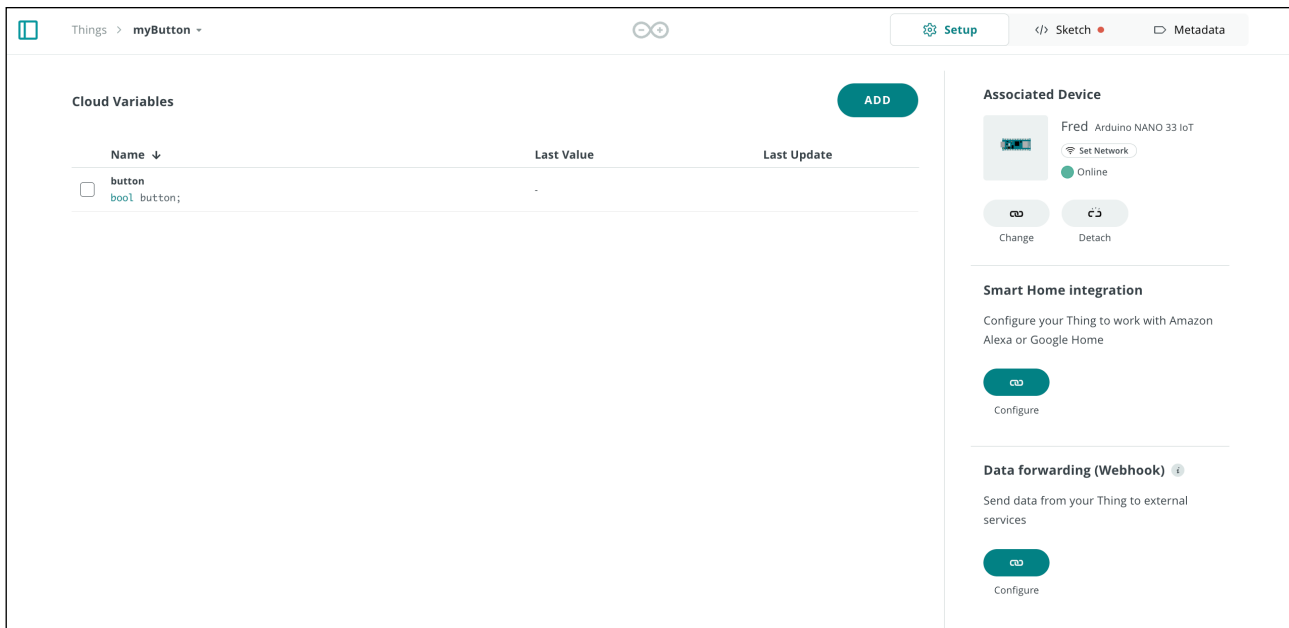
☒ On change

☐ Periodically

CANCEL

ADD VARIABLE

Figure 3: looking like this



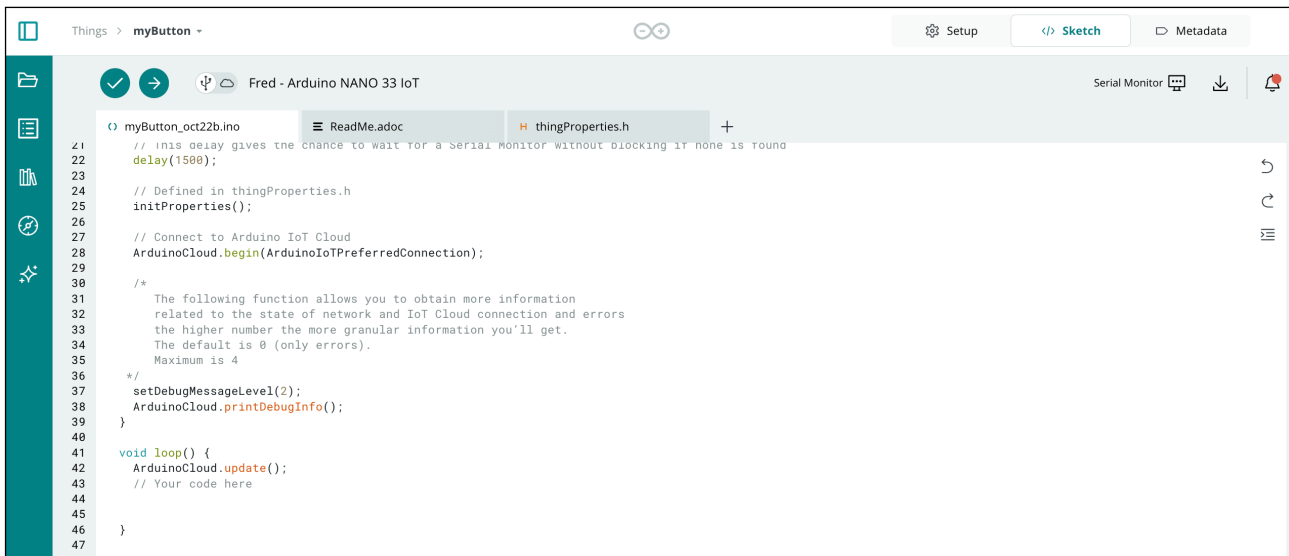


The Sketch

Next, we click on the `</>` **Sketch** tab at the top to see what our generated sketch looks like. You will notice that there is no `onChange()` function. This is because it is read-only. We are reading if the button has been pressed or not.

! Notice that there is no **Sketch Secrets** tab. Fix it by going to the **Devices** and clicking on where it says **WiFi** and make sure the credentials (WiFi name and password) are correct. This will take you back to the sketch but now with the **Sketch Secrets** tab in place.

Figure 4: `</>` Sketch





Sketch B3.1 the full sketch

This is what you should see.

```
/*  
  Sketch generated by the Arduino IoT Cloud Thing "myButton"  
  https://create.arduino.cc/cloud/things/xxxxxxxxxx  
  
  Arduino IoT Cloud Variables description  
  
  The following variables are automatically generated and updated  
  when changes are made to the Thing  
  
  bool button;  
  
  Variables which are marked as READ/WRITE in the Cloud Thing  
  will also have functions  
  which are called when their values are changed from the  
  Dashboard.  
  These functions are generated with the Thing and added at the  
  end of this sketch.  
*/  
  
#include "thingProperties.h"  
  
void setup() {  
  // Initialize serial and wait for port to open:  
  Serial.begin(9600);  
  // This delay gives the chance to wait for a Serial Monitor  
  without blocking if none is found  
  delay(1500);  
  
  // Defined in thingProperties.h  
  initProperties();  
}
```

```
// Connect to Arduino IoT Cloud
ArduinoCloud.begin(ArduinoIoTPreferredConnection);

/*
    The following function allows you to obtain more information
    related to the state of network and IoT Cloud connection and
    errors
    the higher number the more granular information you'll get.
    The default is 0 (only errors).
    Maximum is 4
*/
setDebugMessageLevel(2);
ArduinoCloud.printDebugInfo();
}

void loop() {
    ArduinoCloud.update();
    // Your code here
}
```



Notes

The key point here is that we have the void loop() as the last function. This is where we will put most of our code.



Sketch B3.2 adding in the new code

We add the following code. The `pinMode()` is obvious. We are going to read digital pin **5**, which will either be on/off, true/false, 1/0. The button variable is a boolean data type.

```
/*  
  Sketch generated by the Arduino IoT Cloud Thing "myButton"  
  https://create.arduino.cc/cloud/things/xxxxxxxxxx  
  
  Arduino IoT Cloud Variables description  
  
  The following variables are automatically generated and updated  
  when changes are made to the Thing  
  
  bool button;  
  
  Variables which are marked as READ/WRITE in the Cloud Thing  
  will also have functions  
  which are called when their values are changed from the  
  Dashboard.  
  These functions are generated with the Thing and added at the  
  end of this sketch.  
*/  
  
#include "thingProperties.h"  
  
void setup() {  
  // Initialize serial and wait for port to open:  
  Serial.begin(9600);  
  pinMode(5, INPUT_PULLUP);  
  // This delay gives the chance to wait for a Serial Monitor  
  without blocking if none is found  
  delay(1500);  
}
```

```
// Defined in thingProperties.h
initProperties();

// Connect to Arduino IoT Cloud
ArduinoCloud.begin(ArduinoIoTPreferredConnection);

/*
    The following function allows you to obtain more information
    related to the state of network and IoT Cloud connection and
    errors
    the higher number the more granular information you'll get.
    The default is 0 (only errors).
    Maximum is 4
*/
setDebugMessageLevel(2);
ArduinoCloud.printDebugInfo();
}

void loop() {
    ArduinoCloud.update();
    button = digitalRead(5);

}
```



Notes

We are using the **pull-up** resistor. We are going to read the the data on digital pin **5**, which will be either **HIGH** or **LOW**.



Sketch B3.3 simplified sketch

Taking out all the spaces and comments. Highlighting the additional code.

```
#include "thingProperties.h"

void setup() {
  Serial.begin(9600);
  pinMode(5, INPUT_PULLUP);
  delay(1500);
  initProperties();
  ArduinoCloud.begin(ArduinoIoTPreferredConnection);
  setDebugMessageLevel(2);
  ArduinoCloud.printDebugInfo();
}

void loop() {
  ArduinoCloud.update();
  button = digitalRead(5);
}
```



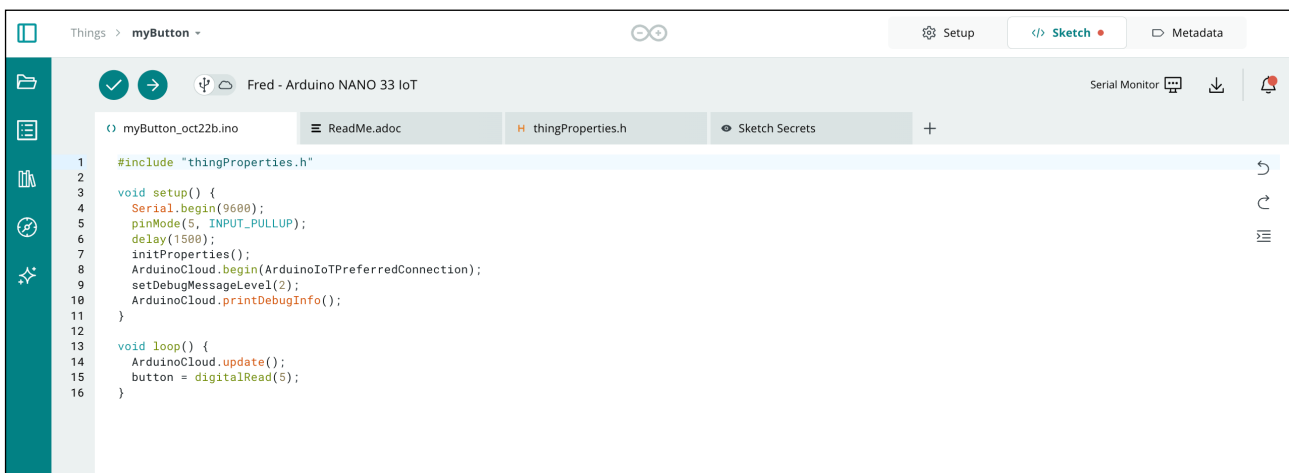
Notes

The code is relatively simple once you take out the functions generated by the Arduino Cloud.

Upload

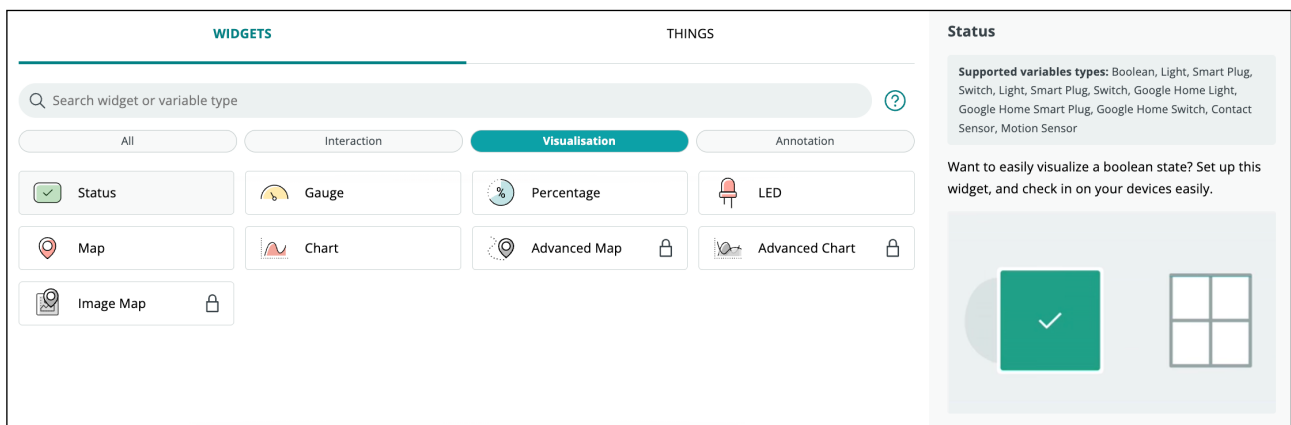
Your next job is to click on the upload button. Hopefully, everything will be in order. I found I had to do it a couple of times. Just check that you have the correct details in the secrets tab. If it is missing, then go back to your device and add them there before continuing.

Figure 5: Sketch Secrets tab is now there



Once uploaded, we dash to the **Dashboard**. Create a **Dashboard** and call it **myButton**. We are going to add a widget called **Status**. Add that widget.

Figure 6: Visualisation widgets





The Status widget

The status widget can be customised, but with the default settings, we can at least check that everything works. The default is on/off. When you press the button, it changes to OFF and the colour red.

Figure 7a: button not pressed

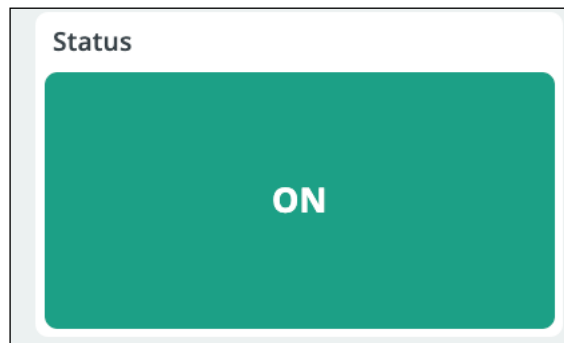
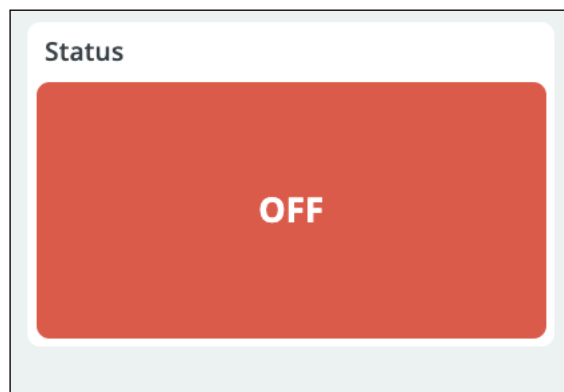


Figure 7b: button pressed

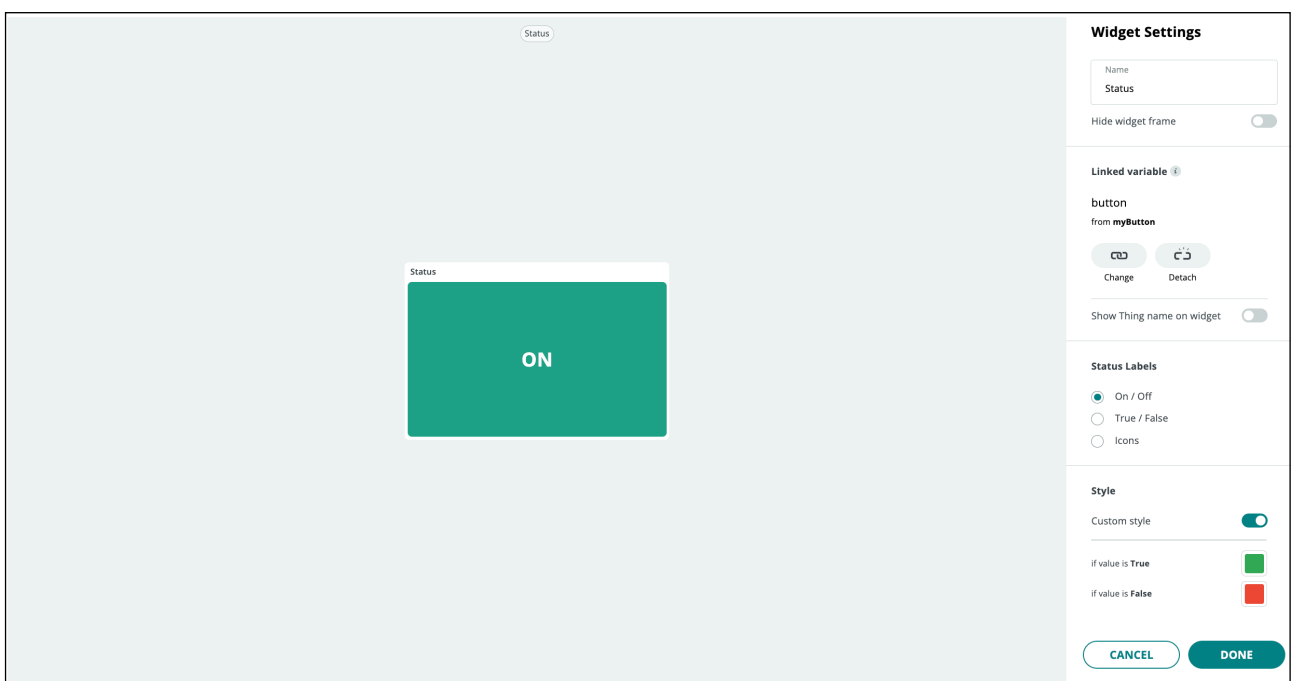


Customising the widget

Just to be a bit creative, we are going to have a microphone icon which will relate to the mic being off (mute) when the button is not pressed, but the mic being on when pressed (recording).

Go to **edit** the widget. Switch the custom style button on, and you get a small extra menu.

Figure 8: Custom Style





Custom section

Click on the icon's radial button. That makes the list of icons available. Select the **X** next to the red square. Search for the icon that is a microphone and select it. Go to the **✓** icon and change that to the microphone with a diagonal line through it. Next, change the green colour to black. Click **DONE**.

Figure 9a

Widget Settings

Name
Status

Hide widget frame ☐

Linked variable ⓘ
button
from myButton

Show Thing name on widget ☐

Status Labels

☐ On / Off
☐ True / False
☒ Icons

Style
Custom style

if value is True
if value is False

Figure 9b

Widget Settings

Name
Status

Hide widget frame ☐

Linked variable ⓘ
button
from myButton

Show Thing name on widget ☐

Status Labels

☐ On / Off
☐ True / False
☒ Icons

Style
Custom style ☒

if value is True
if value is False



Toggle widget

As you press the button, it will toggle between record (red), mute (black). This is just a suggestion.

Figure 10a: record

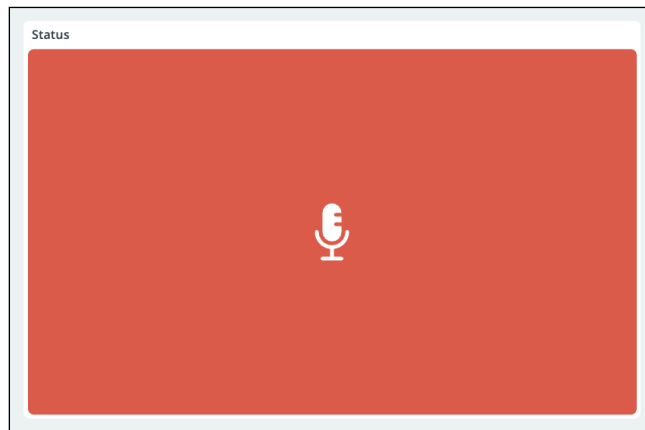


Figure 10b: mute

