How to Add a New Sensor or Component to Adafruit IO WipperSnapper

Created by Brent Rubell

New Component

Which component would you like to set up?

Pin Components

Your Sensor Here!  
Flat Vibration Switch


Last updated on 2023-08-29 04:47:45 PM EDT
Table of Contents

Overview
• How Components Work in Adafruit IO

Get Set Up
• Fork the WipperSnapper_Components Repository

Adding a Pin Component
• Identifying the Type of Pin Component to Use
• Add Component JSON
• Modify Component Definition JSON File
• (Optional) Add Component Visualization
• Add Component Image
• (Optional) Add Component Animation
• Commit Your Changes and Push
• Submit a Pull Request
• Testing your Pin Component

Adding an I2C Component
• Add Component JSON Definition
• Add Component Image
• (Optional) Add Component Animation
• Commit Your Changes and Push
• Submit a Pull Request

Add the Definition JSON
• Add Component JSON Definition
• Add an Image of the Sensor
• (Optional) Add an Animation GIF
• Commit Your Changes and Push
• Submit a Pull Request

Build WipperSnapper with PlatformIO
• Fork WipperSnapper Firmware
• Clone WipperSnapper Firmware
• Open WipperSnapper Firmware in PlatformIO
• Upload Locally-Built WipperSnapper Firmware

Adding the I2C Component Driver
• Get Setup
• Create a new Header File
• Add begin()
• About I2C Sensors in WipperSnapper
• Add a function to read a sensor
• Include the new driver in the I2C Base Driver
• Add Handling Code to the I2C Base Driver
• Test the new I2C Sensor Driver
• Create a Pull Request to WipperSnapper Arduino

F.A.Q.
Overview

You've set up an internet-enabled development board with [Adafruit IO Wipersnapper](https://www.adafruit.com/). But, after glancing at the component picker, you don't see the component you need for your IoT project. Let's get your favorite input, output, or sensor added to Adafruit IO WipperSnapper so it can be used by projects designed by you and others!

The process for adding components varies with difficulty depending on what type of component you're adding:

- If you're adding a "pin component", such as a digital output (i.e.: an LED), analog input (i.e.: potentiometer or light sensor), or digital input (i.e.: push-button or PIR sensor) - adding a new component is easy. It involves writing a small amount of descriptive JSON and adding an image!
- If you're adding an "I2C sensor component" - adding a new I2C sensor component is a little bit trickier. The process involves writing a small amount of JSON, adding an image, setting up a build environment for WipperSnapper, and writing/modifying C++ code.

How Components Work in Adafruit IO

Adding a new component to WipperSnapper means your component will show up in this list on Adafruit IO, for everyone using the platform:
It will also automatically generate a web-form like this:
The sensor or component will have its current state either monitored or controlled by the device.

Once created, a WipperSnapper component can work seamlessly with the rest of Adafruit IO's features including Historical Feeds, Dashboards, and Actions!

Get Set Up

This guide assumes you already have a GitHub account and have installed Git.

- If you are unfamiliar with GitHub and Git or would like to refresh before starting, we have a great guide on using the features of GitHub here. (link)
Fork the WipperSnapper_Components Repository

We'll start by creating a fork of the WipperSnapper Components repository on your account. Sign in to your GitHub account and navigate to the WipperSnapper_Components repository.

Click the "Fork" button on the upper-right-hand corner of the website.

Your fork should take a few seconds to complete, once done, let's move on to adding the component type you want.

Adding a Pin Component

In WipperSnapper terminology, pin components are digital outputs (i.e.: an LED, a motor), analog inputs (i.e.: a potentiometer or a light sensor), or digital inputs (i.e.: a push-button or a PIR sensor).

Adding a "pin component" to WipperSnapper involves writing a small amount of descriptive JSON and adding an image.

Identifying the Type of Pin Component to Use

This guide page uses the Flat Vibration Switch sold in the Adafruit store as an example of a pin component that can be added to WipperSnapper. You will need to modify the instructions to fit the needs/specifications of your component.
Flat Vibration Switch - Breadboard friendly
This is a low sensitivity, directional vibration-induced trigger switch. Inside is a very soft spring coiled around a long metal pin. When the switch...
https://www.adafruit.com/product/4081

The first step is to identify what type of pin component this switch is.

When the vibration switch gets bumped, it acts as a closed switch. In order to get the state of the switch, a microcontroller needs to configure the switch as a digital input as it only reports two states: when the switch is either open ('0') or closed ('1').

So, we'll be creating a digital input pin component as we'll want the vibration switch to send data to WipperSnapper when it reports a boolean state (open or closed).

Add Component JSON

This page assumes you've followed the instructions on the Get Setup page and have the WipperSnapper_Components repository forked and locally cloned.

First, create a Git branch to work within. In this case, name it add-flat-vibration-switch.

Next, navigate to the Wippersnapper_Components/components/pin directory and make a copy of a component with similar functionality. The toggle switch component and the vibration switch both operate the same way and send the same values if they're toggled.

Create a copy of the toggle_switch folder.
Component folder names should use underscores and the name should be lower case (i.e: `my_component`).

Rename the `toggle_switch` copy folder to `flat_vibration_switch`.

Modify Component Definition JSON File

The definition.json file within this folder is known as the definition file, it defines the pin-based WipperSnapper component for use in Adafruit IO.

Using a text editor, open the toggle switch's definition.json file.

```json
{
  "displayName": "Toggle Switch",
  "published": true,
  "autoSelectString": "toggle",
  "mode": "DIGITAL",
  "direction": "INPUT",
  "defaultPeriod": 30,
  "visualization": {
    "type": "switch",
    "offLabel": "Off",
    "offIcon": "fa6:regular:light-switch-off",
    "onLabel": "On",
    "onIcon": "fa6:solid:light-switch-on"
  }
}
```
JSON files are written as pairs of keys and values, separated by a semicolon.

The first key/value pair in the JSON definition is the `displayName`, which is the human-friendly name of a component. This field is required.

We'll start by changing the `displayName` from "Toggle Switch" to "Flat Vibration Switch". You should change the `displayName` to reflect the component you're adding.

At this point, the definition.json file looks like the following.

```json
{
  "displayName": "Flat Vibration Switch",
  "autoSelectString": "toggle",
  "mode": "DIGITAL",
  "direction": "INPUT",
  "defaultPeriod": 30
}
```

Next, set the required pin `mode` field. This field may either be ANALOG or DIGITAL. Since the microcontroller reading a vibration switch will either read a digital 1 or a digital 0 value, its mode is digital.

At this point, the definition.json file looks like the following.

```json
{
  "displayName": "Flat Vibration Switch",
  "autoSelectString": "toggle",
  "mode": "DIGITAL",
  "direction": "INPUT",
  "defaultPeriod": 30
}
```

Finally, the last of the required fields is the pin's `direction` which may be either INPUT or OUTPUT. Data from the vibration switch will be read into the board, so define an INPUT direction.

```json
{
  "displayName": "Flat Vibration Switch",
  "autoSelectString": "toggle",
  "mode": "DIGITAL",
  "direction": "INPUT",
  "defaultPeriod": 30
}
```

The optional `autoSelectString` field is a hint for automatically looking up pin names. For example, an LED component will automatically select a pin labeled "LED" if the pin exists.
In the vibration switch case, this field is not required so it can be removed.

```json
{
    "displayName": "Flat Vibration Switch",
    "mode": "DIGITAL",
    "direction": "INPUT",
    "defaultPeriod": 30
}
```

The `defaultPeriod` field describes the default value of the Period field on the "Component Settings" form on WiperSnapper (in seconds). You may leave this as-is.

However, some sensors may take a longer amount of time than the 30-second default to gather readings and this time period may need to be adjusted.

```json
{
    "displayName": "Flat Vibration Switch",
    "mode": "DIGITAL",
    "direction": "INPUT",
    "defaultPeriod": 30
}
```

(Optional) Add Component Visualization

You may (optionally) add some text to the definition file to describe its "look" on the Adafruit IO device interface.

While optional, a component with visualization not only makes it visually prettier - it adds function! Adafruit IO component visualization includes icons and labels to explain what the component does.

For example, this is a LED switch component without a defined visualization:

![LED switch component without visualization](image)

And this is a vibration switch component on Adafruit IO with a visualization defined in its definition file.

![Vibration switch component with visualization](image)
Adding the following `visualization` object to your component's `definition.json` will define its look and feel:

```json
"visualization": {
  "type": "switch",
  "offLabel": "Still",
  "offIcon": "fa6:regular:bell-slash",
  "onLabel": "Bzzz",
  "onIcon": "fa6:solid:bell-on"
}
```

The component's visualization `type` may be either a `switch` or a `button` (we're working on adding more component types). Each type is omnidirectional, meaning it may be used as either an output or an input.

The `off/onLabel` and `off/onIcons` are specific to the `switch` component type.

- For an example of how the button component type works, check out the definition.json file for the push button component.

The component's labels (`onLabel` and `offLabel`) correspond to its physical state. For example, a vibration sensor when "off" is still (reflected by the `offLabel` above). When the vibration sensor is activated (by movement or touch), we'll change the `onLabel` to "Touched".

The component's icons, `onIcon` and `offIcon`, correspond to the component’s on and off state.

For example, the `onIcon` is defined as:

```
"onIcon": "fa6:solid:bell-on"
```

`bell-on` is the name of the icon, with a bunch of namespaces in front of it. Let's break it down:

- `fa6` means we are using a set of icons called [Font Awesome 6 ()](https://fontawesome.com/icons)
- `solid` is the style of the icon. There are 4 icon styles to pick from - `light`, `thin`, `regular` and `solid`.
- `bell-on` is the name of the icon.

To find an icon, visit [https://fontawesome.com/icons ()] to search for an icon you like.
Clicking on an icon will show you the various icon styles.

Once you've completed adding the `visualization` object to the component's definition, you're finished editing its definition. A complete `definition.json` file should look something like the following.

```json
{
  "displayName": "Toggle Switch",
  "published": true,
  "autoSelectString": "toggle",
  "mode": "DIGITAL",
  "direction": "INPUT",
  "defaultPeriod": 30,
  "visualization": {
    "type": "switch",
    "offLabel": "Off",
    "offIcon": "fa6:regular:light-switch-off",
    "onLabel": "On",
    "onIcon": "fa6:solid:light-switch-on"
  }
}
```
Add Component Image

Next, you'll need to add an image of your component. It's best to get a photo of the component from the manufacturer's website.

First, make sure your image adheres to the following specifications:

- Image file's extension can be any one of: jpg, jpeg, gif, png, svg
- Image file's dimensions must be 300px x 300px
- Image file's size must be at least 3kb and must not exceed 100kb

For the flat vibration switch, it is the first image from the Adafruit Store product page(). Next, this image must be resized. Using a web-based tool such as https://picresize.com() is suggested and ensuring the final image is 300px by 300px.

Add the resized image to the flat_vibration_switch folder and rename it image.png. You may need to delete the existing file, there should only be one file named image.EXTENSION in this folder.
You may also delete the optional animation.gif file within this folder if you are not planning to add one.

(Optional) Add Component Animation

This step is optional due to the amount of work required to produce an animation. The Ruiz brothers have a video on how to create a spinning board animation below:

The optional animation.gif file also must adhere to the following specifications:

- File must ALWAYS be .gif
- File dimensions must be 300px x 300px
- File is between 5kb and 700kb

Commit Your Changes and Push

Next, let’s add all these changes to your fork. Add your components using the `git add` command.

Typing `git status` shows the files you’re about to add.

Then, commit the files by typing `git commit -m "adding new component"`
And push to your forked repository, `git push yourRepo add-component-name`

Next, follow this page to submit a pull request to add your component to WipperSnapper. ()

Submit a Pull Request

Finally, it's time to submit a pull request to add the component to WipperSnapper!

After you've pushed the updated files to your branch, navigate to https://github.com/adafruit/Wippersnapper_Components/pulls () and click "Compare & pull request".

Give your pull request a name and a description. Make it as descriptive as possible!

Click "Create pull request".
The repository will run checks on these files. If the checks pass, Adafruit will review the files.

For reference, an example of this pull request is here >>> ()

Testing your Pin Component

As part of our review process, we'll make your component appear under the component picker as "in development". This allows you to test the component before giving us (Adafruit) the final OK.

On your Adafruit IO Device page, open the component picker. Click the "Show Dev" checkbox.

You should see the pin component you just added.
Make sure everything on the form looks okay. Then, create your component and test it out with a device.

Create Flat Vibration Switch Component

Flat Vibration Switch Name
Flat Vibration Switch

Flat Vibration Switch Pin
Choose a Pin

Return Interval
- On Change
- Periodically

Period
Every 30 seconds

Once you've fully tested your component and are satisfied with how it works, let us know in the Pull Request and we'll make it live on Adafruit IO!
I2C is a popular protocol that uses two wires to communicate with sensors (often called I2C devices). If you need a quick overview (or refresher!) of how I2C works, read through this guide page.

The following page(s) will go over the process of adding your favorite I2C sensor.

You will need to follow this page first, then the "Adding an I2C Component Driver" page after.

The process of adding I2C components is more involved than adding a pin component and you'll be asked to write some C++ code. If you haven't written C++ before - don't worry - this guide will walk you through the process and provide examples.

As an example, the following uses the MCP9808 High Accuracy I2C Temperature Sensor Breakout for this guide since it's a simple breakout that only reports temperature data.

Keep in mind that some I2C breakouts have multiple sensors which report multiple types of data (such as temperature, humidity, altitude, etc.). The more types of data reported by a sensor, the more involved its I2C component driver code will be.
Add Component JSON Definition

This page assumes you've followed the instructions on the Get Setup page and have the WipperSnapper_Components repository forked and locally cloned.

First, create a branch to work within. In our case, we'll name it `add-mcp9808`.

Next, navigate to the Wippersnapper_Components/components/i2c directory. Create a new folder named mcp9808.

Within the mcp9808 folder, add a new file named definition.json. This will hold the definition of the physical sensor for the WipperSnapper website.

JSON files are written as pairs of keys and values, separated by a semicolon.

The first key/value pair in the JSON definition is the `displayName`, which is the human-friendly name of a component. This field is required.

```json
{
  "displayName": "MCP9808",
}
```

Each I2C device has a unique address to identify it by. We have a list of I2C addresses on this page. If it's not on that page, check the sensor's datasheet.

According to its documentation, the Adafruit MCP9808's I2C addresses can range from 0x18 to 0x1C.

These addresses are listed within the `i2cAddresses` field in the JSON definition:

```json
{
  "displayName": "MCP9808",
  "i2cAddresses": [ "0x18", "0x19", "0x1A", "0x1C" ],
}
```

Next, add the list of subcomponents. These are all the sensors on the I2C device.
The MCP9808 breakout only has an ambient temperature sensor. There are two types of ambient temperature sensors on the list of sensor types - "ambient-temp" and "ambient-temp-fahrenheit". The ambient-temp sensor type tells Adafruit IO that a sensor on this breakout will send ambient temperature data back in degree Celsius. Adding ambient-temp-fahrenheit to the list of subcomponents will allow a user on Adafruit IO to select between having the sensor return the temperature in either degree Celsius or degree Fahrenheit.

The final .JSON file for this sensor is below (and here it is on GitHub):

```json
{
    "displayName": "MCP9808",
    "i2cAddresses": ["0x18", "0x19", "0x1A", "0x1C"],
    "subcomponents": ["ambient-temp", "ambient-temp-fahrenheit"]
}
```

Add Component Image

Next, you'll need to add an image of your component. It is recommended you get a photo of the component from the manufacturer's website.

First, make sure your image adheres to the following specifications:

- Image file's extension can be any one of: jpg, jpeg, gif, png, svg
- Image file's dimensions must be 300px by 300px
- Image file's size must be at least 3kb and must not exceed 100kb

Below is the first image from the Adafruit Store product page.

Next, this image must be resized. It is suggested using a web-based tool such as http://picresize.com and ensuring the final image is 300px by 300px.
Add the resized image to the mcp9808 folder and rename it image.EXTENSION. You may need to delete the existing file, there should only be one file named image.EXTENSION in this folder.

You may also delete the optional animation.gif file within this folder if you are not planning to add one.

(Optional) Add Component Animation

This step is optional due to the amount of work required to produce an animation. The Ruiz brothers have a video on how to create a spinning board animation below:

The optional animation.gif file also must adhere to the following specifications:

- File must ALWAYS be .gif
- File dimensions must be 300px by 300px
Commit Your Changes and Push

Next, add all these changes to your fork. Add your components using the `git add` command.

Typing `git status` shows the files you're about to add.

```
~ /D/g/Wippersnapper_Components   | add-mcp9808   | git add
~ /D/g/Wippersnapper_Components   | add-mcp9808   | git status
```

On branch add-mcp9808
Changes to be committed:
(use "git reset HEAD <file>..." to unstage)

new file:  components/i2c/mcp9808/definition.json
new file:  components/i2c/mcp9808/image.png

Then, commit the files by typing `git commit -m "adding new component"`

```
~ /D/g/Wippersnapper_Components   | add-mcp9808   | git commit -m "adding mcp9808 files"
```

2 files changed, 11 insertions(+)
create mode 100644 components/i2c/mcp9808/definition.json
create mode 100644 components/i2c/mcp9808/image.png

And push to your forked repository, `git push yourRepo add-component-name`

```
~ /D/g/Wippersnapper_Components   | add-mcp9808   | git push brenttu add-mcp9808
```

Submit a Pull Request

Finally, to submit a pull request to add the component to WipperSnapper!

After you've pushed the updated files to your branch, navigate to [https://github.com/adafruit/Wippersnapper_Components/pulls](https://github.com/adafruit/Wippersnapper_Components/pulls) and click "Compare & pull request".

Give your pull request a name and a description. Make it as descriptive as possible and click "Create pull request".
The repository will run checks on these files. If the checks pass, Adafruit will review the files.

When approved, they'll be automatically added to WipperSnapper and available under the component picker along with the form options you added to the definition.json file.

For reference, the pull request above is located here.

Add the Definition JSON

Add Component JSON Definition

This page assumes you've followed the instructions on the Get Setup page and have the WipperSnapper_Components repository forked and locally cloned.

First, create a branch to work within. In our case, we'll name it add-mcp9808.

Next, navigate to the Wippersnapper_Components/components/i2c directory. Create a new folder named mcp9808.
Within the mcp9808 folder, add a new file named definition.json. This will hold the definition of the physical sensor for the WipperSnapper website.

JSON files are written as pairs of keys and values, separated by a semicolon.

The first key/value pair in the JSON definition is `displayName`, which is the human-friendly name of a component. This field is required.

The next key/value pair is `published`. This pair should be set to false until the component driver is distributed with a release of WipperSnapper.

```
{
  "displayName": "MCP9808",
  "published": false,
}
```

Each I2C device has a unique address to identify it by. We have a list of I2C addresses on this page. If it's not on that page, check the sensor's datasheet.

According to its documentation, the Adafruit MCP9808's I2C addresses can range from 0x18 to 0x1C.

These addresses are listed within the `i2cAddresses` field in the JSON definition:

```
{
  "displayName": "MCP9808",
  "i2cAddresses": [ "0x18", "0x19", "0x1A", "0x1C" ],
}
```

Add an Image of the Sensor

Next, you'll need to add an image of the sensor. It is recommended to obtain a photo of the component from the manufacturer's website.

First, make sure your image adheres to the following specifications:

- The Image file's extension can be any one of: jpg, jpeg, gif, png, svg
- The image file's dimensions must be 300px by 300px
- The image file's size must be at least 3kb and must not exceed 100kb

For this example, we are using the first image from the Adafruit Store product page.
Next, this image must be resized. It is suggested to use a web-based tool such as [http://picresize.com](http://picresize.com) and ensure the final image is 300px x 300px.

Add the resized image to the mcp9808 folder and rename it image.EXTENSION. You may need to delete the existing file, there should only be one file named image.EXTENSION in this folder.

You may also delete the optional animation.gif file within this folder if you are not planning to add one.

(Optional) Add an Animation GIF

This step is optional due to the amount of work required to produce an animation. The optional animation.gif file also must adhere to the following specifications:

- File must ALWAYS be .gif
The Ruiz brothers have a video on how to create a spinning board animation below:

Commit Your Changes and Push

Next, add all these changes to your fork. Add your components using the `git add` command.

Typing `git status` shows the files you're about to add.

Then, commit the files by typing `git commit -m "adding new component"

And push to your forked repository, `git push yourRepo add-component-name`

Submit a Pull Request

Finally, submit a pull request to add the component to WipperSnapper!

After you've pushed the updated files to your branch, navigate to https://github.com/adafruit/Wippersnapper_Components/pulls and click "Compare & pull request".

Give your pull request a name and a description. Make it as descriptive as possible and click "Create pull request".
The repository will run checks on these files. If the checks pass, Adafruit will review the files. When approved, they’ll be automatically added to WipperSnapper and available under the component picker along with the form options you added to the definition.json file.

For reference, the pull request above is located here.

---

**Build WipperSnapper with PlatformIO**

The easiest way to build WipperSnapper firmware on your computer is by using a combination of [Microsoft Visual Studio Code](https://code.visualstudio.com) (VSCode) and [PlatformIO](https://platformio.org).

Navigate to [https://code.visualstudio.com](https://code.visualstudio.com) and install the latest version of the VSCode editor.

Once downloaded and installed, open VSCode and navigate to the Extensions tab (If you do not have this tab, you can find it using the toolbar by navigating to Settings->Extensions).

Search for the PlatformIO extension.
Select the PlatformIO extension and click Install.

Fork WipperSnapper Firmware

To contribute to WipperSnapper, you will need to create a copy of the WipperSnapper project's repository. This copy is called a fork and you will use it to make changes to WipperSnapper before submitting them to the official project repository.

Sign in to your Github.com account and navigate to the WipperSnapper firmware repository.

Click the "Fork" button to fork your own copy of the adafruit/Adafruit_Wippersnapper_Arduino repository.
Click "Create Fork" and wait a few seconds for GitHub to create a copy of the WipperSnapper repository on your account.

Clone WipperSnapper Firmware

Next, let’s clone your fork of the Wippersnapper firmware so we can download it, make changes locally, compile it, and eventually send your changes up to the official repository.

On the page for your forked repository, click the green Code button. The dropdown should present a link like the one below:

https://github.com/brentru/Adafruit_Wippersnapper_Arduino.git()

Copy the link to your clipboard.

Using a terminal program, enter the following command to clone the forked repository to your computer. Be sure to replace youruserid with your GitHub user ID, and paste the URL from your clipboard.

```
git clone -o youruserid https://your-fork-URL
```
Now that the WipperSnapper firmware is local on your computer, check out a new branch and move on to the new step - building WipperSnapper.

If you are unfamiliar with how to check out a new branch, Kattni wrote a great overview of the process here (...)

Open WipperSnapper Firmware in PlatformIO

Open Visual Studio Code and click Open to bring up the file explorer.

Navigate to the Folder containing your local copy of Adafruit_Wippersnapper_Arduino and click Open.

VSCode should open to WipperSnapper's platformio.ini file. A notification on the bottom-right of your VSCode instance shows a message that PlatformIO is configuring the project and installing the required libraries to build WipperSnapper.
When the libraries finish installing, you should see a notification (or terminal output) that the "Project has been successfully updated!".

Click the PlatformIO logo (it looks like an alien's head) on the left-hand side of the VSCode editor to open PlatformIO.

Under PlatformIO's Project Tasks tab, every board WipperSnapper supports is listed as a folder.

Select the board you'd like to build for. For example, I have an Adafruit Feather ESP32 v2 and I will select the `featheresp32v2` board from the dropdown.
When PlatformIO is loading tasks for a new environment, a blue clock appears on top of the logo. Wait until the blue clock disappears before continuing the build process.

When PlatformIO loads the featheresp32v2 environment, you should see a Build step underneath featheresp32v2->General->Build.

Click Build and PlatformIO should begin to build WipperSnapper firmware for the Feather ESP32 V2.
Once WipperSnapper builds successfully, the terminal will show SUCCESS along with the time it took to build.

Congrats- you've successfully built WipperSnapper locally!

Upload Locally-Built WipperSnapper Firmware

To upload and test WipperSnapper on your device, you will need to specify the upload_port belonging to your device.

Within VSCode, open the platformio.ini file and navigate to the board you are using. Using this guide as an example, we previously compiled for the Adafruit Feather ESP32 V2. The build environment looks like the following.
Next, add the `upload_port()` for your board. This port name varies but will take the form of:

- `/dev/ttyUSB0` - Serial port (Unix-based OS)
- `COM3` - Serial port (Windows OS)

On the VSCode sidebar, click the PlatformIO logo.

Under Project Tasks-> Your Board (for this example, the board is featheresp32v2), click Upload to upload WipperSnapper firmware to your development board.
After the PlatformIO upload succeeds, press the RESET button on your board and navigate to https://io.adafruit.com/devices()

If the upload was successful, you should see your board connected to Adafruit IO!

Adding the I2C Component Driver

Now that you have built WipperSnapper, it's time to add the driver for the I2C sensor.

Get Setup

As an example, we'll be creating a new WipperSnapper I2C component for the MCP9808 High Accuracy I2C Temperature Sensor Breakout. This breakout is perfect to use as an example as it only reports the temperature reading.

You will need to alter/edit the code below to reflect the sensor you're adding, keeping in mind that sensors that report more readings (temperature, humidity, air quality, etc) require more code.
Adafruit MCP9808 High Accuracy I2C Temperature Sensor Breakout

The MCP9808 digital temperature sensor is one of the more accurate/precise we've ever seen, with a typical accuracy of ±0.25°C over the sensor's -40°C to...

https://www.adafruit.com/product/5027

WipperSnapper firmware is based on Arduino. You will need an Arduino library that already contains the driver code for your sensor.

This page is using the Adafruit MCP9808 sensor breakout as an example. The corresponding Arduino library for this sensor is listed on the Arduino Library Manager () and publicly available on GitHub ()

In your IDE/text editor of choice, open the src folder within Adafruit_WipperSnapper_Arduino.

Create a new Header File

WipperSnapper's I2C drivers are stored as header (.h) files within the src/components/i2c/drivers folder and each driver follows the naming convention WipperSnapper_I2C_Driver_SensorName.h.

For our example, we'll create a new header file called WipperSnapper_I2C_Driver_MCP9808.h within src/components/i2c/drivers and add the following skeleton code to the file:

```c
/*!  
 * @file WipperSnapper_I2C_Driver_MCP9808.h
 * 
 * Device driver for the MCP9808 Temperature sensor.
 * 
 * Adafruit invests time and resources providing this open source code, 
 * please support Adafruit and open-source hardware by purchasing 
 * products from Adafruit!
 * 
 * Copyright (c) Brent Rubell 2023 for Adafruit Industries.
 * 
 * MIT license, all text here must be included in any redistribution.
 * */

#ifndef WipperSnapper_I2C_Driver_MCP9808_H
#define WipperSnapper_I2C_Driver_MCP9808_H

#include "WipperSnapper_I2C_Driver.h"
```

©Adafruit Industries
#include &lt;Adafruit_MCP9808.h&gt;

/**************************************************************************
@brief  Class that provides a driver interface for a MCP9808 sensor.
**************************************************************************

class WipperSnapper_I2C_Driver_MCP9808 : public WipperSnapper_I2C_Driver {
public:
	/**************************************************************************
	@brief Constructor for a MCP9808 sensor.
	@param i2c The I2C interface.
	@param sensorAddress 7-bit device address.
	**************************************************************************
	WipperSnapper_I2C_Driver_MCP9808(TwoWire *i2c, uint16_t sensorAddress) {
		_i2c = i2c;
		_sensorAddress = sensorAddress;
	}
		
	/**************************************************************************
	@brief Destructor for an MCP9808 sensor.
	**************************************************************************
	~WipperSnapper_I2C_Driver_MCP9808() {
		delete _mcp9808;
	}
		
	/**************************************************************************
	@brief Initializes the MCP9808 sensor and begins I2C.
	@returns True if initialized successfully, False otherwise.
	**************************************************************************
	bool begin() {
		// TO-DO: Initialization code goes here!
	}
	
	protected:
	Adafruit_MCP9808 *_mcp9808; //&lt; Pointer to MCP9808 temperature sensor object
};
#endif // WipperSnapper_I2C_Driver_MCP9808

At the top of the code are include guards () which prevent the compiler from including multiple instances of this class.

Next, we’ll include the base I2C class (include”WipperSnapper_I2C_Driver.h“) and the Arduino driver library for the I2C sensor (in this example, include <Adafruit_MCP9808.h>).

The constructor (WipperSnapper_I2C_Driver_MCP9808()) and destructor (~WipperSnapper_I2C_Driver_MCP9808()) are called when the component is either initialized or deleted via the WipperSnapper website. The constructor is passed the I2C object and the sensor address from the new component form.
At the bottom, underneath `protected`, you'll need to create a pointer to access the sensor driver (in this example, `_mcp9808`).

Add `begin()`

The `begin()` function contains the code required to initialize the sensor's driver and returns True upon successful initialization.

The MCP9808 has a very simple initialization routine:

```cpp
bool begin() {
  _mcp9808 = new Adafruit_MCP9808();
  return _mcp9808-&gt;begin((uint8_t)_sensorAddress, _i2c);
}
```

About I2C Sensors in WipperSnapper

Before explaining how to add a function to read the sensor, let's first talk a little bit about how this works within WipperSnapper.

WipperSnapper utilizes the Adafruit Unified Sensor Driver library which provides a single type for all sensor readings, `sensors_event_t`, and enforces standardized SI units for each type of sensor.

By reducing all data to a single `sensors_event_t` 'type' and settling on specific, standardized SI units for each sensor family, the same sensor types return values which are compatible with any similar sensor. This enables us to support all types of sensor breakouts without worrying about the units matching and prevents code re-use.

This section of the Adafruit Unified Sensor Driver's GitHub README provides an in-depth explanation of how the driver works.

Add a function to read a sensor

Next, a function needs to be added to read the value of the sensor. The WipperSnapper_I2C_Driver.h file contains base class implementations for reading each type of sensor prefixed by `getEvent_`. 
The MCP9808 contains an ambient temperature sensor. Looking within the `WipperSnapper_I2C_Driver.h`, the `getEventAmbientTemp()` function "reads an ambient temperature sensor".

Let's add an empty `getEventAmbientTemp()` function to our header file:

```cpp
bool getEventAmbientTemp(sensors_event_t *tempEvent) {
    // TODO: Add code here!
    return true;
}
```

Add code to read the sensor's temperature. The temperature is stored within the `sensor_event_t` type's `temperature` field.

```cpp
bool getEventAmbientTemp(sensors_event_t *tempEvent) {
    tempEvent-&gt;temperature = _mcp9808-&gt;readTempC();
    return true;
}
```

Wait - my I2C breakout has more than one sensor reading!

WipperSnapper uses the Adafruit Unified Sensor Driver to handle reading sensors and enforces standardized SI units for every type of sensor. In `WipperSnapper_I2C_Driver.h`, there are functions for each predefined sensor type within the Adafruit Unified Sensor Driver.

The sensor driver header file should implement a `getEventType()` function call for each value to be read.

- For examples of implementations (some might match your sensor!), check out `Adafruit_Wippersnapper_Arduino/tree/main/src/components/i2c/drivers` folder.
Include the new driver in the I2C Base Driver

With the sensor driver complete, the next step is to add it to the base I2C class file, `src/components/i2c/WipperSnapper_I2C.h`

Open the file `src/components/i2c/WipperSnapper_I2C.h`. At the top, add a line to `#include` the MCP9808 driver:

```cpp
#include "drivers/WipperSnapper_I2C_Driver_MCP9808.h"
```

At the bottom of this file, add a new pointer to the MCP9808 driver:

```cpp
private:
    WipperSnapper_I2C_Driver_MCP9808 *_mcp9808 = nullptr;
```

Add Handling Code to the I2C Base Driver

Finally, we'll need to add code to detect and initialize the MCP9808. Within `src/components/i2c/WipperSnapper_I2C.cpp`, locate the `initI2CDevice()` function:

```cpp
bool WipperSnapper_Component_I2C::initI2CDevice(      
    wippersnapper_i2c_v1_I2CDeviceInitRequest *msgDeviceInitReq) {
    WS_DEBUG_PRINT("Attempting to initialize I2C device: ");
    WS_DEBUG_PRINTLN(msgDeviceInitReq->i2c_device_name);
    uint16_t i2cAddress = (uint16_t)msgDeviceInitReq->i2c_device_address;
    ...
}
```
Test the new I2C Sensor Driver

Assuming you have successfully built WipperSnapper using PlatformIO, navigate to the PlatformIO tab and select the board you are using to test.

Click Build.

Once PlatformIO has built WipperSnapper successfully, it'll display SUCCESS at the bottom of VSCode (or terminal):

Next, upload the modified WipperSnapper firmware to your board by navigating to PlatformIO and clicking Upload.
From the PlatformIO tab, click Monitor. A new serial monitor should open. You should see the following debug output stating WipperSnapper is running.

Additionally, you should see your device appear as Online on io.adafruit.com. From the device page on Adafruit IO, click + New Component.
As part of the development process, your component appears under the component picker as "in development". This allows you to test the component before giving us (Adafruit) final approval to make it live.

At the top of the component picker, click the "Show Dev" checkbox. Your sensor should appear in the I2C Components list with a badge showing "In Development"

Click your component. On the monitor, you should see an I2C Scan command being executed on the development board.

```
/dev/cu.SLAB_USBtoUART

16:01:16.179 -> [0xA0], [0x80], [0x0b],
16:01:16.179 -> Looking for subscription len 73
16:01:16.215 -> Found sub #1
16:01:16.215 -> Data len: 12
16:01:16.215 -> Data:
16:01:16.215 -> ff
16:01:16.215 -> MQTT puback packet:
16:01:16.215 -> 2 [0x32], Y [0x59], [0x00], I [0x49],
16:01:16.215 -> Client sendPacket returned: 4
16:01:16.215 -> 12 bytes.
16:01:16.215 -> cbDecodeSignalRequestI2C
16:01:16.215 -> I2C Scan Request
16:01:16.215 -> EXEC: I2C Scan
16:01:16.215 -> Found I2C Device at 0x24
16:01:16.215 -> Found I2C Device at 0x56
16:01:16.251 -> I2C Devices Found: 2
16:01:16.251 -> Publishing Message: I2CResponse...MQTT publish packet:
```

On Adafruit IO, configure your component and click Create Component.
After clicking Create Component, the sensor should initialize. After the period you specified, it should read the sensor's value and publish it to Adafruit IO.

You should see the new value appear on your WipperSnapper device page.

It's good practice to keep the monitor open during this process as its useful for detecting errors and tracing them back.
Almost done - let's add a pull request to the Adafruit WipperSnapper library so others can use this sensor.

Create a Pull Request to WipperSnapper Arduino

First, read over the Doxygen page on this guide and run Doxygen on your code. Then, format your code using clang-format.

Once you've run Doxygen and linted using clang-format, you're ready to submit a pull request adding the sensor to the WipperSnapper Arduino library.

Commit and push your files to a branch on local your fork of Adafruit_WipperSnapper_Arduino and open a new pull request on the Adafruit_WipperSnapper_Arduino repository.

Adafruit has an example of a "perfect" pull request here.
Once reviewed and accepted, Adafruit will include support for your sensor in the latest version of the Adafruit IO Wippersnapper library. We will also remove the "in-development" flag from the component to make it live on Adafruit IO.

F.A.Q.

What about SPI or UART Components?

Wippersnapper does not currently support components that communicate over SPI or UART. It may in the future, though!

I do not feel comfortable writing or editing C++ code but want an I2C sensor component added to Wippersnapper

You may request a component to be added to Wippersnapper by navigating to the Wippersnapper_Components GitHub page and filing a request for a component type (). Please be descriptive as possible, and include any libraries/documentation/URLs you may have. Requests have no ETA regarding when they will be implemented into Wippersnapper.