DEVICE COMMAND AND CONTROL WITH RULES ENGINE

There are two communication primitives that you can use to remotely interact with a Particle device in real-time:

- **Functions** allow code on the device to be called when requested by the Device Cloud. A string is passed to the function and an integer is returned.
- **Variables** allow the device to record a value. The value (integer, double, or string) is only retrieved by the cloud when needed.

The Rules Engine makes it possible to call functions and check variables as part of a flow. This can help you tell a device to do something under certain conditions, or check a sensor reading based on a cloud-side trigger.

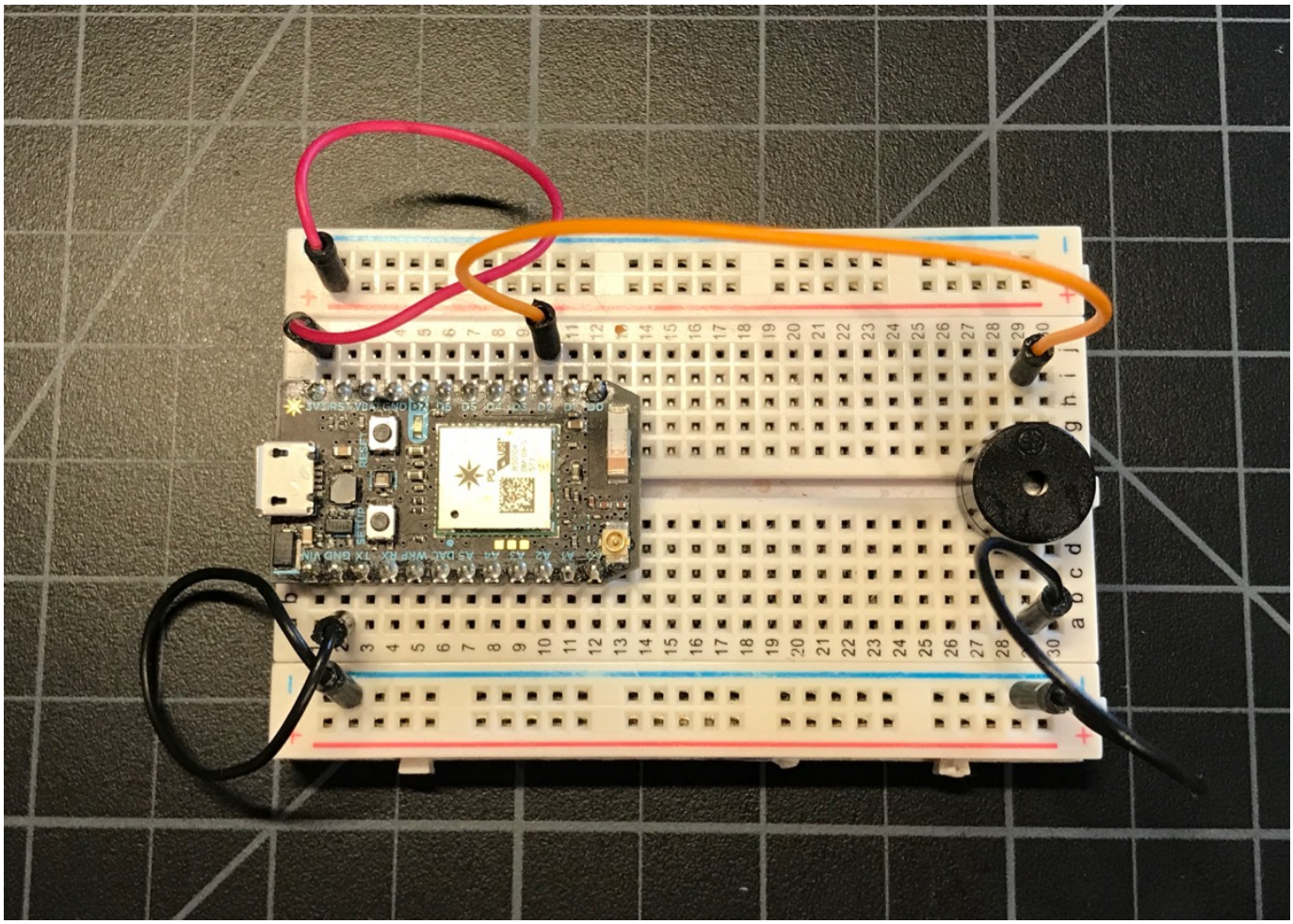
In this tutorial, we'll expand on the previous real-time alerting tutorial and introduce some new nodes for communicating with your Particle devices.

We'll be using the function and publish nodes in this section.

Tutorial Hardware

For the hardware side of this project we're using a second Photon and a buzzer.

If you don't have a buzzer, you can just use a plain Photon because the blue D7 LED turns on whenever the buzzer would be turned on, so you can see the alarm conditions that way.

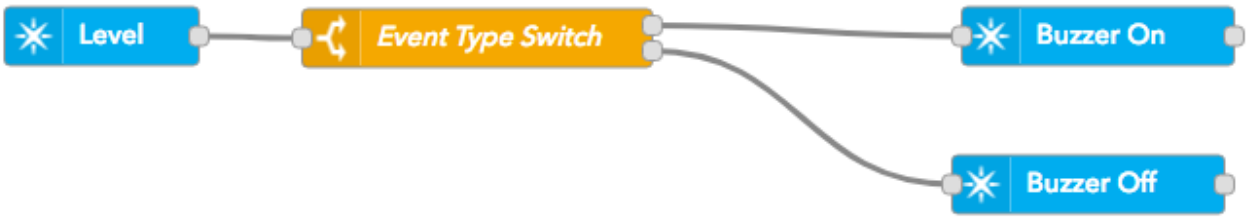


Creating the Flow

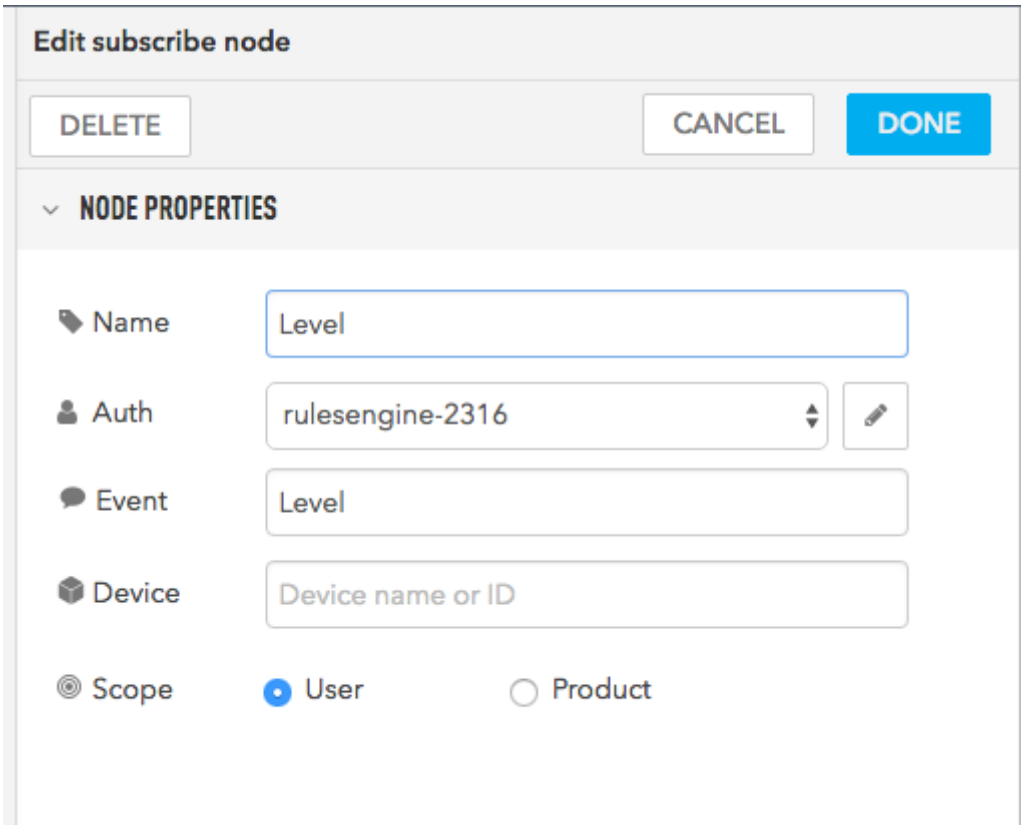
Drag the Copy Rules button into the Rules Engine window to create the flow automatically, or you can create the flow from scratch with the steps below.

Copy Rules ▶

This is the flow we'll be creating:

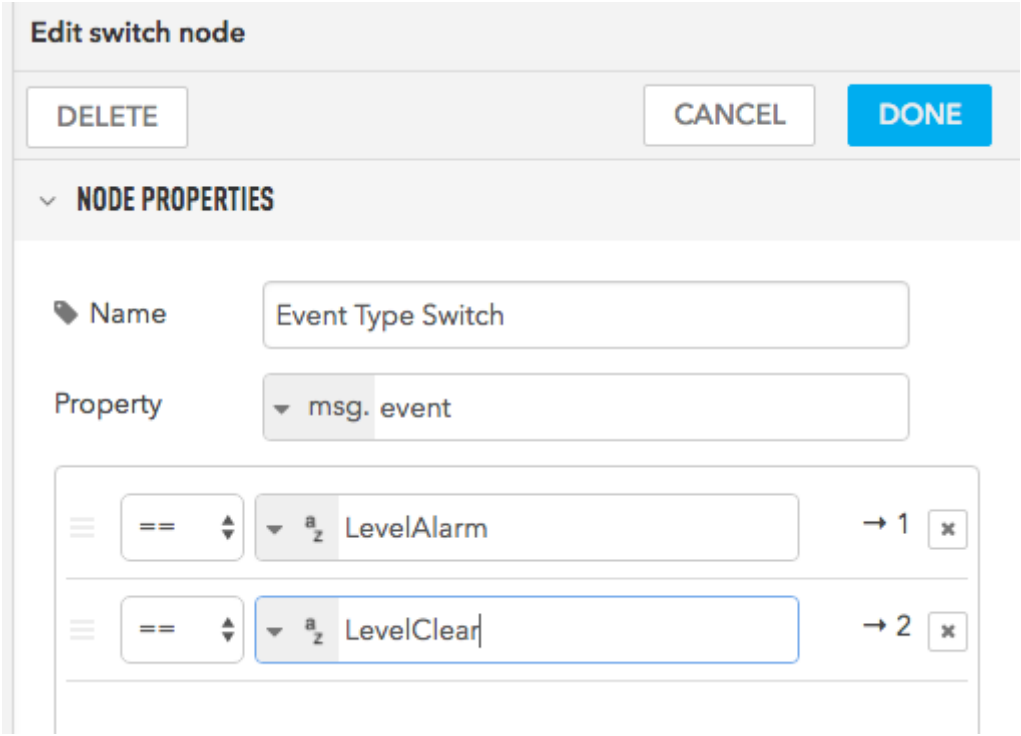


- From the **Particle** group, drag **subscribe** node to your flow.
- Double click to edit it.
- Set **Name** to **Level** (can be anything).
- Set **Auth** to the authentication we created in the real-time alerting tutorial.
- Set **Event** to **Level**. This was LevelAlert in the previous tutorial, but we really only want Level, which will match all of the Level events including LevelAlert, LevelClear, and LevelValue.
- Leave the **Device** field empty
- Leave the **Scope** at **User**.
- Click **Done**.



- From the **function** group, drag a **switch** node to your flow.
- Double click to edit it.
- Set the **Name** to **Event Type Switch** (can be anything).
- Change the *Property **to msg.**event** instead of msg.payload.
- Set the first option to **== LevelAlarm**.

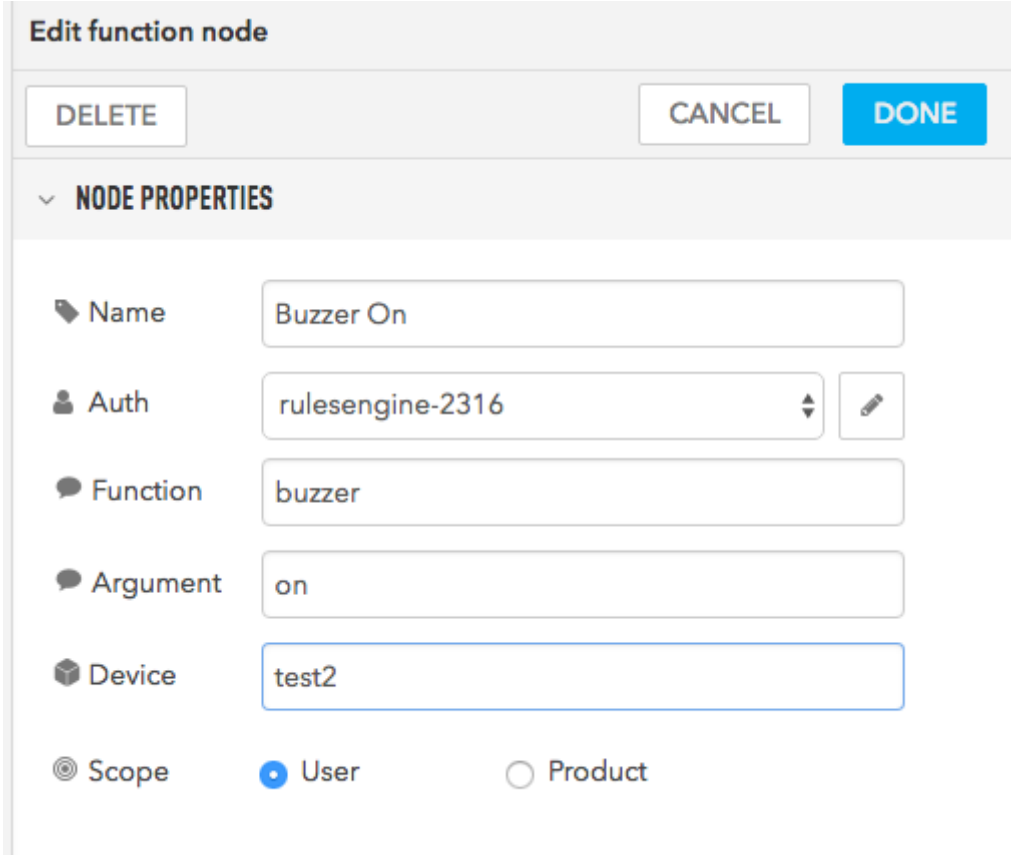
- Click the Add button below the list box.
- Set the second option to **== LevelClear**.
- Click **Done**.



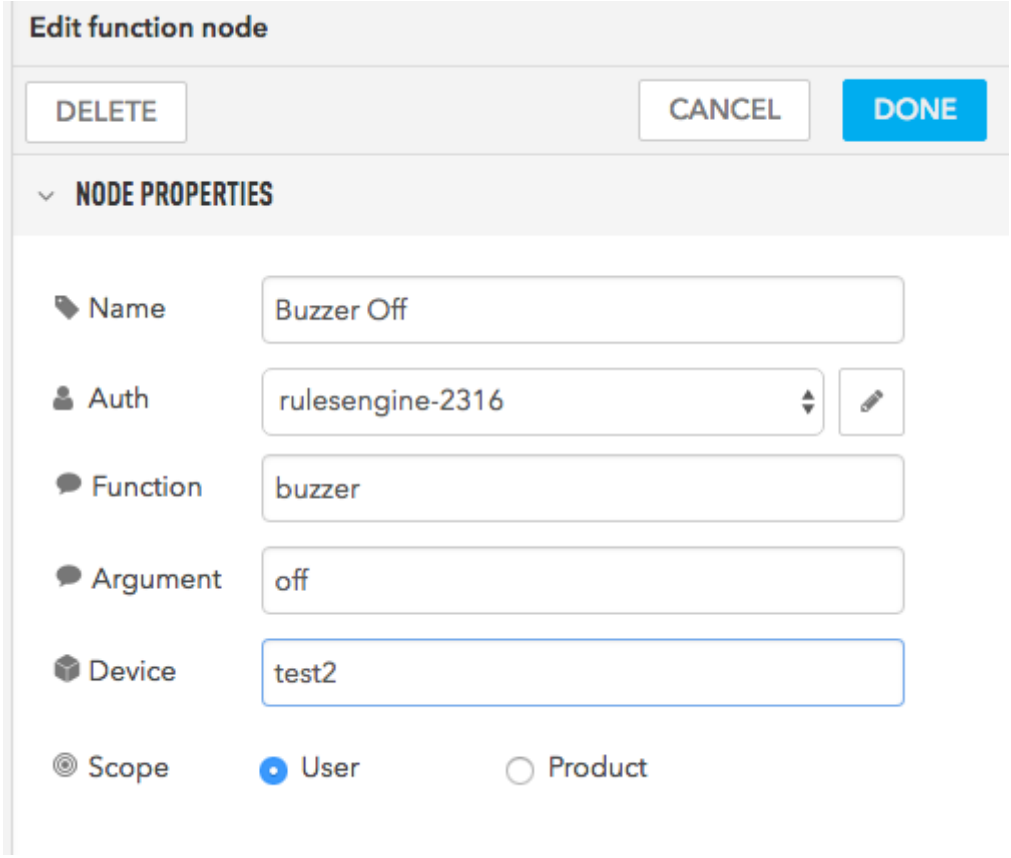
- Unlike the nodes we've used before, you'll notice there are two outputs from our newly created **switch** node.

- From the **Particle** group, drag **function** node to your flow.
- Double click to edit it.
- Set **Name** to **Buzzer On** (can be anything).
- Set **Auth** to the authentication we created in the real-time alerting tutorial.
- Set **Function** to **buzzer** (case-sensitive, must match what the device firmware uses).

- Set **Argument** to **on** (case-sensitive).
- Set **Device** to the name of the device that's running the buzzer.
- Leave the **Scope** at **User**.



- Create another function node, but this time:
- Set **Argument** to **off** (case-sensitive).



- Connect the nodes together as pictured.
- Deploy your flow.
- Causing an level alarm situation should turn on the buzzer.
- Removing the level alarm situation should turn off the buzzer.

Device firmware

The Photon is programmed with the following code. You can also use this link to open it in the Particle Web IDE.

```
#include "Particle.h"

SerialLogHandler logHandler;

// This is the pin the buzzer is connected to
const int BUZZER_PIN = D2;
const int LED_PIN = D7;

void buttonHandler(system_event_t event);
```

```
int buzzerFunction(String cmd);

void setup() {
    Serial.begin();

    pinMode(BUZZER_PIN, OUTPUT);
    digitalWrite(BUZZER_PIN, LOW);

    pinMode(LED_PIN, OUTPUT);
    digitalWrite(LED_PIN, LOW);

    Particle.function("buzzer", buzzerFunction);

    System.on(button_click, buttonHandler);
}

void loop() {
}

int buzzerFunction(String cmd) {
    if (cmd.equals("on")) {
        digitalWrite(BUZZER_PIN, HIGH);
        digitalWrite(LED_PIN, HIGH);
    }
    else {
        digitalWrite(BUZZER_PIN, LOW);
        digitalWrite(LED_PIN, LOW);
    }
    return 0;
}

void buttonHandler(system_event_t event) {
    // Clicking the SETUP button mutes the buzzer.
    // The blue D7 LED will stay lit until the alarm condition is no longer
occurring.
    digitalWrite(BUZZER_PIN, LOW);
}
```

Variables

In the **Particle** section of the palette, a **variable** allows a flow to get a value from a specific device in real-time. This node is used in the **Visualization and Analytics** tutorial, **Querying the**

value.

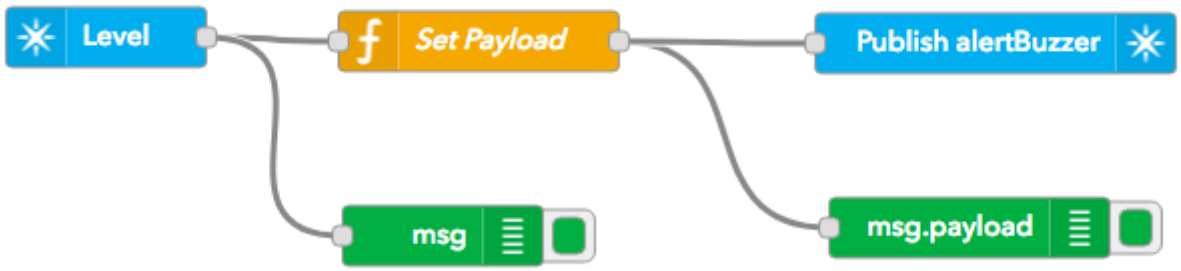
Publish

Instead of using a Particle function to turn on a buzzer on a specific device, you can use Particle publish to notify any buzzer in your account. You can have as many as you want, without changing the Rules Engine code!

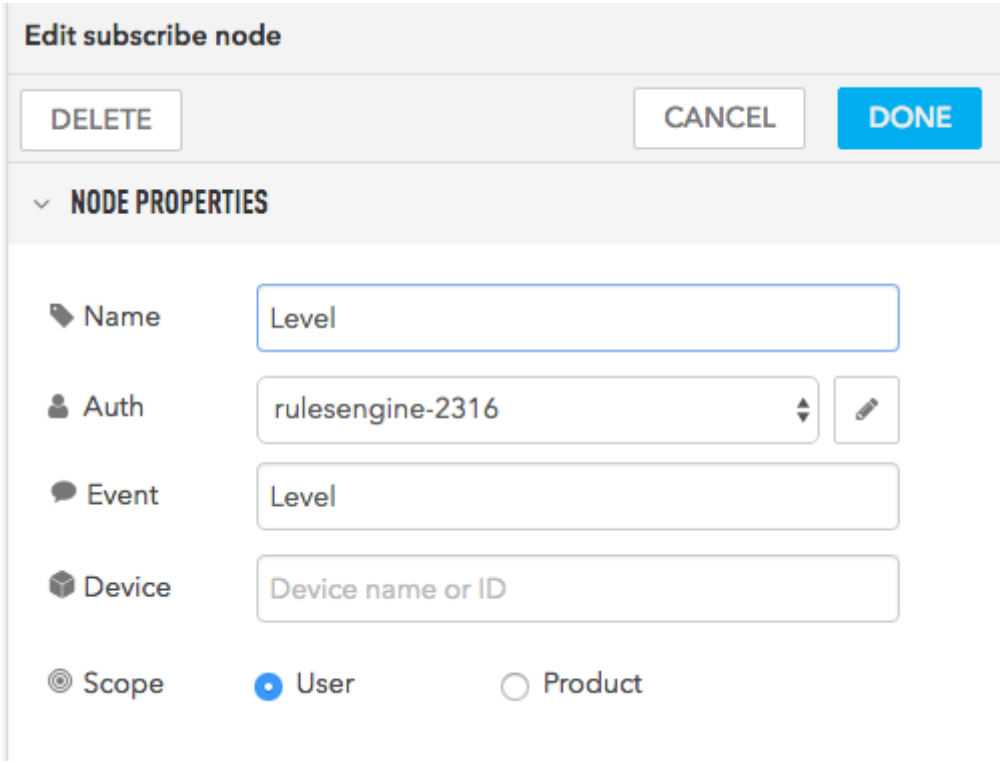
Drag the Copy Rules button into the Rules Engine window to create the flow automatically, or you can create the flow from scratch with the steps below.

Copy Rules ▶

This is the flow we'll be creating:



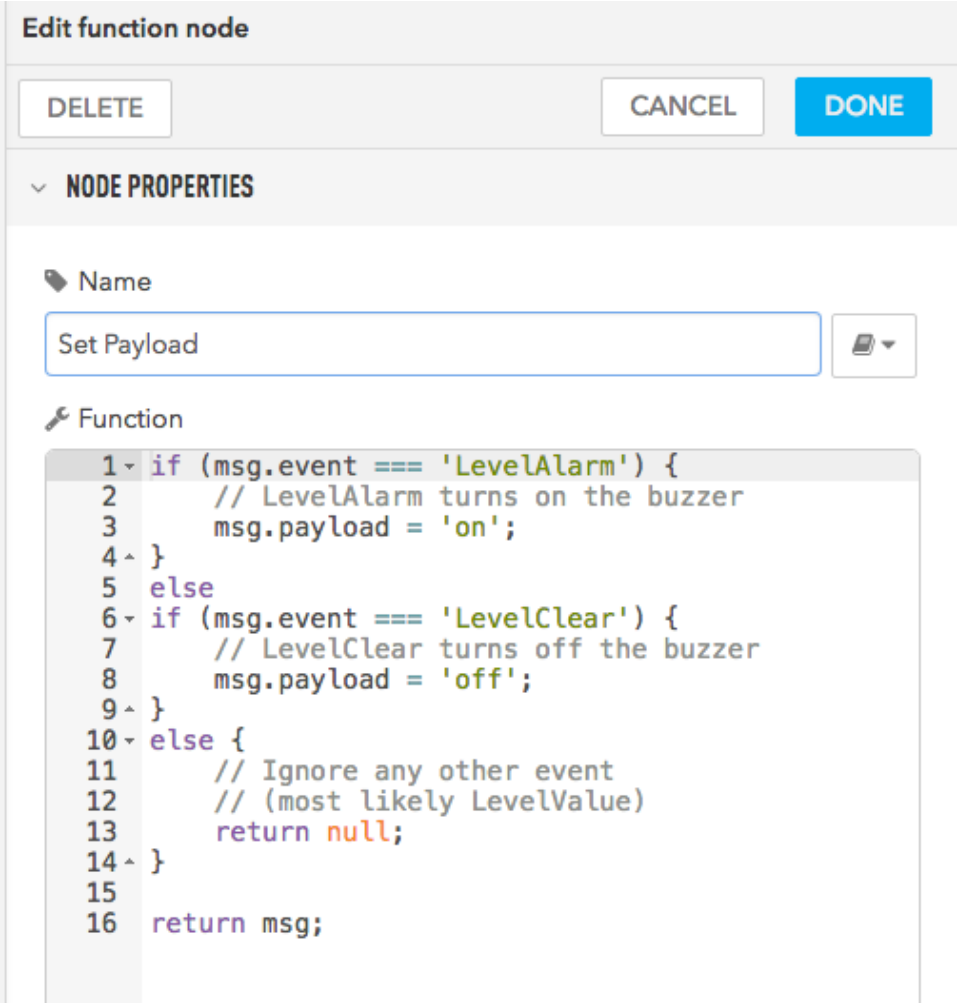
- From the Particle group, drag a **subscribe** node to the workspace.
- Double click to edit.
- Set the Name, Auth, and Event ("Level").
- Leave the Device field blank (allow all devices)



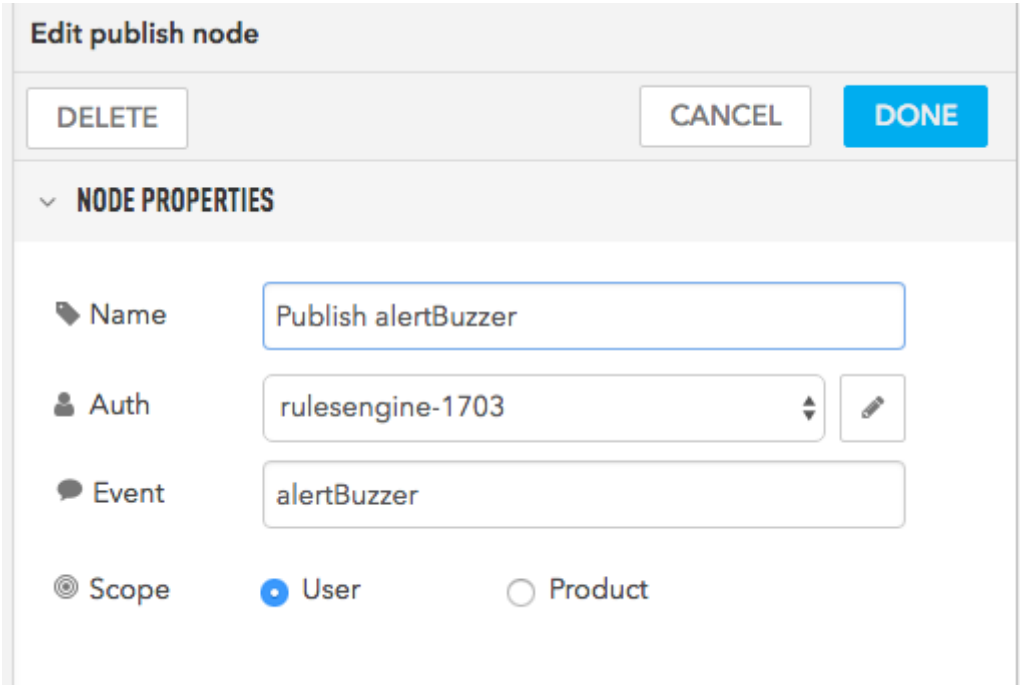
- In the Function section, drag a **function** node into the workspace. Note that this is a function function, not a Particle function!
- Set the function to:

```
if (msg.event === 'LevelAlarm') {
    // LevelAlarm turns on the buzzer
    msg.payload = 'on';
}
else
if (msg.event === 'LevelClear') {
    // LevelClear turns off the buzzer
    msg.payload = 'off';
}
else {
    // Ignore any other event
    // (most likely LevelValue)
    return null;
}

return msg;
```



- From the Particle group, drag a **publish** node to the workspace.
- Double click to edit.
- Set the Name, Auth, and Event ("alertBuzzer").



The firmware on the device is quite similar. You can also use this link to open it in the Particle Web IDE.

```
#include "Particle.h"

SerialLogHandler logHandler;

// This is the pin the buzzer is connected to
const int BUZZER_PIN = D2;
const int LED_PIN = D7;

void buttonHandler(system_event_t event);
void buzzerHandler(const char *event, const char *param);

void setup() {
    Serial.begin();

    pinMode(BUZZER_PIN, OUTPUT);
    digitalWrite(BUZZER_PIN, LOW);

    pinMode(LED_PIN, OUTPUT);
    digitalWrite(LED_PIN, LOW);

    Particle.subscribe("alertBuzzer", buzzerHandler, MY_DEVICES);

    System.on(button_click, buttonHandler);
```

```
}

void loop() {
}

void buzzerHandler(const char *event, const char *data) {
    Log.info("buzzerHandler %s %s", event, data);

    if (strcmp(data, "on") == 0) {
        digitalWrite(BUZZER_PIN, HIGH);
        digitalWrite(LED_PIN, HIGH);
    }
    else {
        digitalWrite(BUZZER_PIN, LOW);
        digitalWrite(LED_PIN, LOW);
    }
}

void buttonHandler(system_event_t event) {
    // Clicking the SETUP button mutes the buzzer.
    // The blue D7 LED will stay lit until the alarm condition is no longer
occurring.
    digitalWrite(BUZZER_PIN, LOW);
}
```

The debug log in the Rules Engine can be helpful as well:


9/22/2018, 11:10:46 AM node: fec85b62.05dc18
msg.payload : string[2]
"on"
9/22/2018, 11:10:46 AM node: 9e88b02d.5ce62
msg : Object
{ event: "LevelAlarm", payload: "2.716728", published_at: "2018-09-22T15:10:46.699Z", device: "1e00320004473431383330 38", _msgid: "8b493706.f12068" }
9/22/2018, 11:10:52 AM node: fec85b62.05dc18
msg.payload : string[3]
"off"
9/22/2018, 11:10:52 AM node: 9e88b02d.5ce62
msg : Object
{ event: "LevelClear", payload: "0.548230", published_at: "2018-09-22T15:10:52.701Z", device: "1e00320004473431383330 38", _msgid: "e82d540e.d0f3d8" }