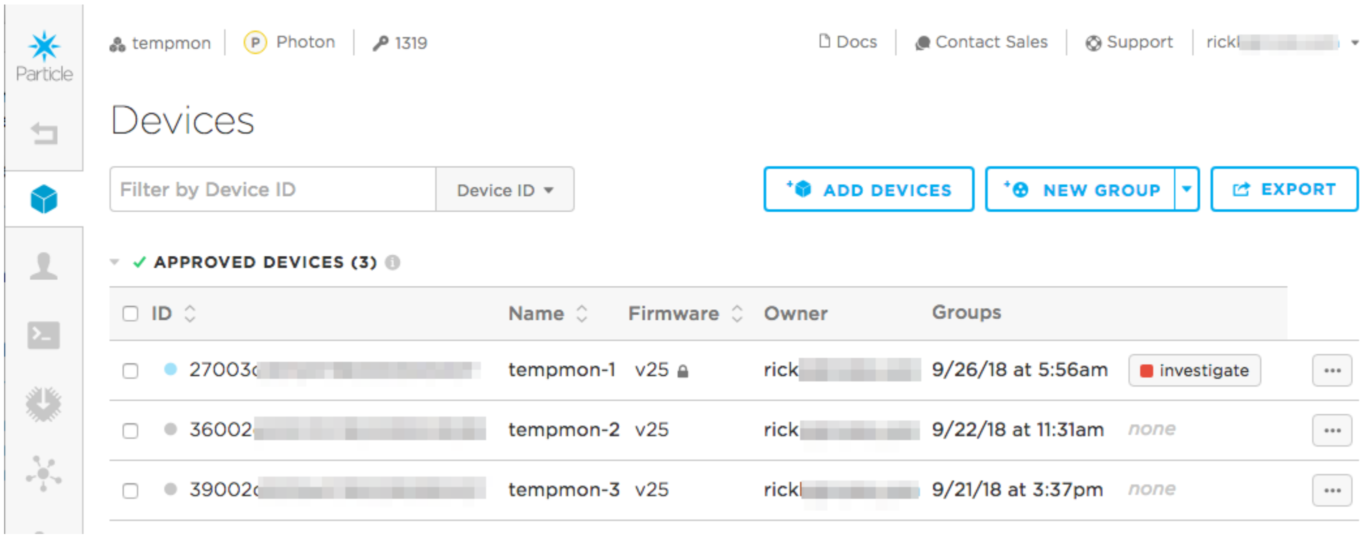
FLEET-WIDE REMOTE DIAGNOSTICS WITH RULES ENGINE

Remote Diagnostics allow you to monitor the health of your fleet of devices in real time. The Rules Engine can help you understand device health indicators in aggregate.

Flagging devices based on diagnostic data

In this section, we'll use the diagnostic data to automatically add devices to a group "investigate" when they report bad signal strength.

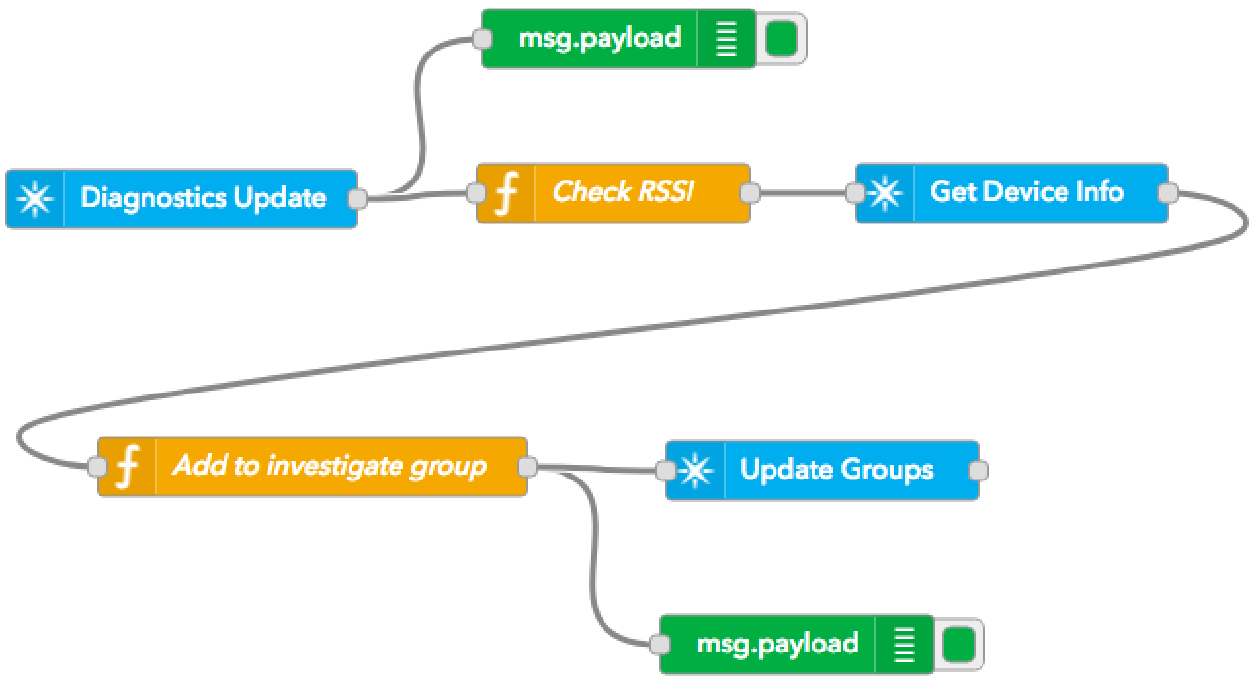


ID	Name	Firmware	Owner	Groups	
27003[redacted]	tempmon-1	v25	rick[redacted]	9/26/18 at 5:56am	investigate
36002[redacted]	tempmon-2	v25	rick[redacted]	9/22/18 at 11:31am	none
39002[redacted]	tempmon-3	v25	rick[redacted]	9/21/18 at 3:37pm	none

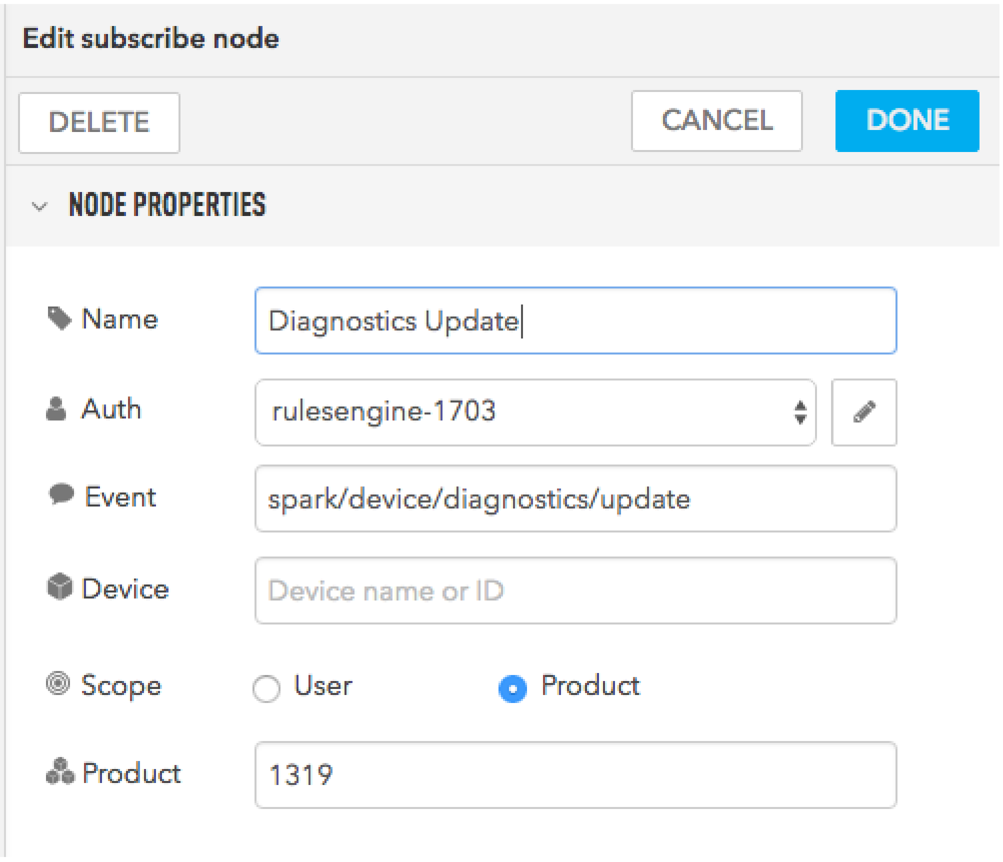
Drag the Copy Rules button into the Rules Engine window to create the flow automatically, or you can create the flow from scratch with the steps below.

Copy Rules ▶

We'll be creating this flow:



- From the Particle group, drag a **subscribe** node to the workspace.
- Double click to edit.
- Set the Name, Auth, and Event ("spark/device/diagnostics/update").
- Leave the Device field blank (allow all devices)
- Set the Scope to **Product** and set your Product ID (mine is 1319).

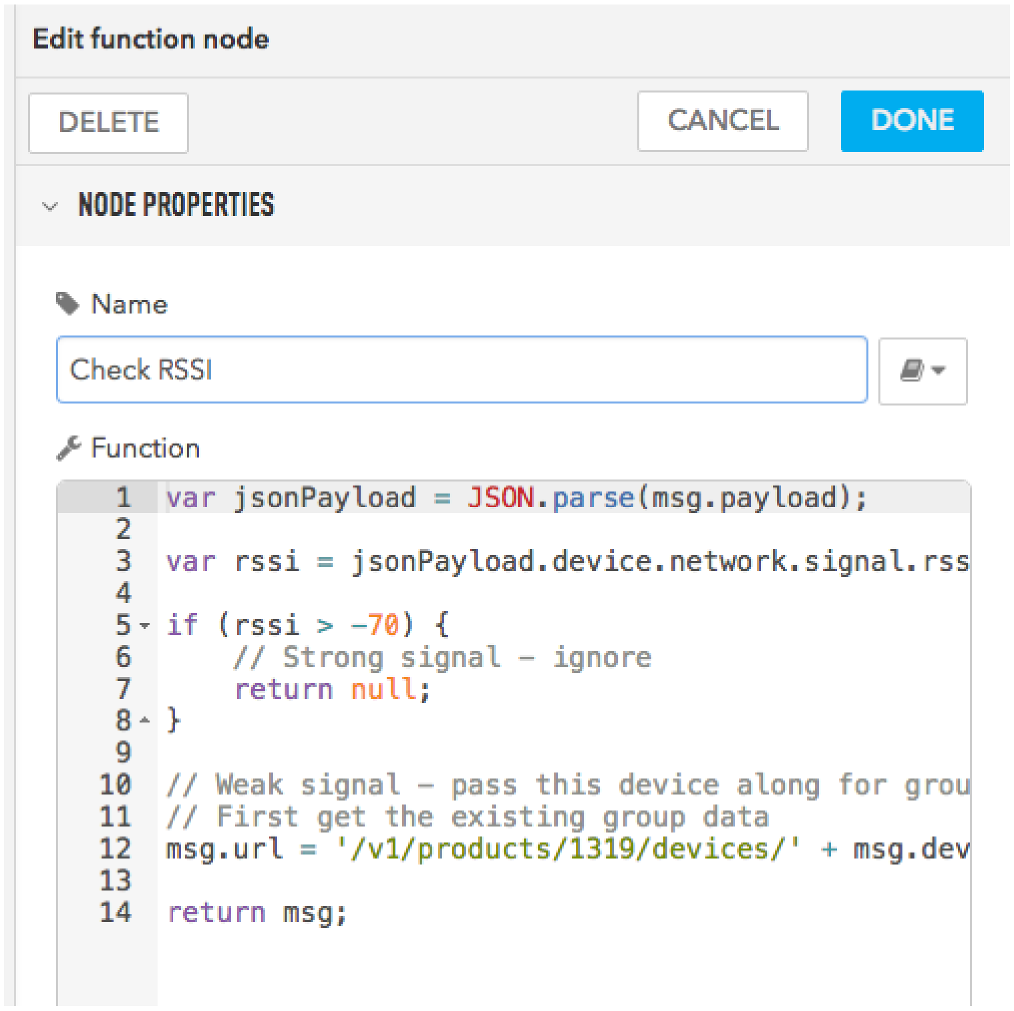


Name: Diagnostics Update
Auth: rulesengine-1703
Event: spark/device/diagnostics/update
Device: Device name or ID
Scope: User [✓ Product]
Product: 1319

- From the Function group, drag a function node to the workspace. Note that this is a Function function, not a Particle function!
- Double click to edit.
- Set the function body to:

```
var jsonPayload = JSON.parse(msg.payload);

var rssi = jsonPayload.device.network.signal.rssi;

if (rssi > -70) {
    // Strong signal - ignore
    return null;
}

// Weak signal - pass this device along for grouping
// First get the existing group data
msg.url = '/v1/products/1319/devices/' + msg.device;

return msg;
```

This checks the RSSI (signal strength), and if it's low, starts by querying the existing information for this device.



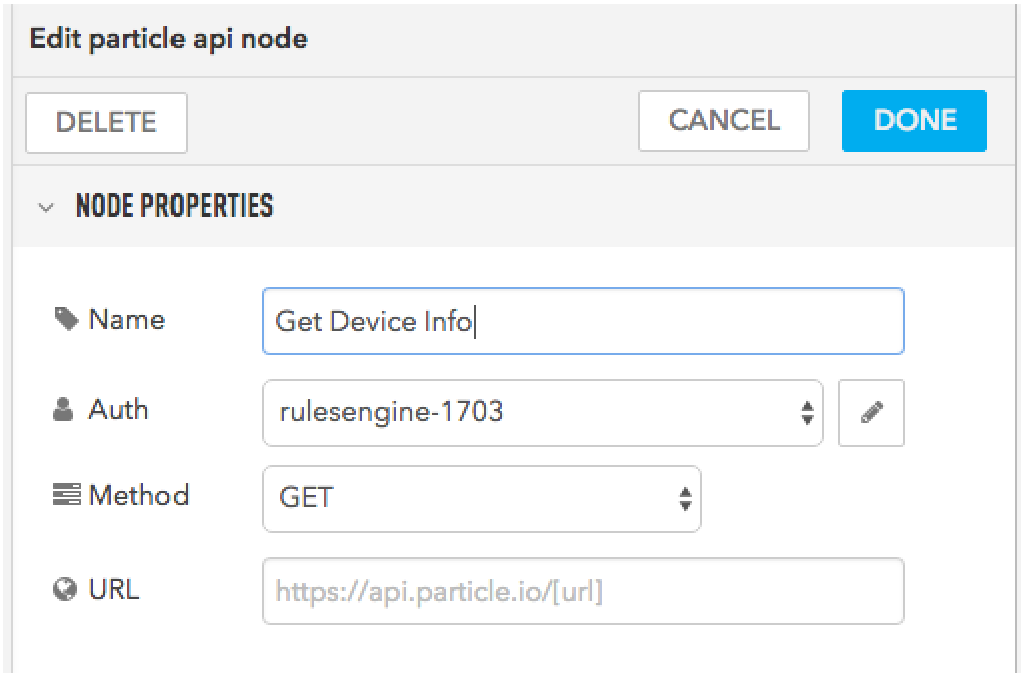
Name

Check RSSI

Function

```
var jsonPayload = JSON.parse(msg.payload);

var rssi = jsonPayload.device.network.signal.rss

if (rssi > -70) {
    // Strong signal - ignore
    return null;
}

// Weak signal - pass this device along for grou
// First get the existing group data
msg.url = '/v1/products/1319/devices/' + msg.dev

return msg;
```

- In the Particle section, drag a **particle api** node into the workspace.
- Double click to edit.
- Set your product authentication.
- Set **Method GET**.
- Leave the URL field blank, as it's set by the previous function node.

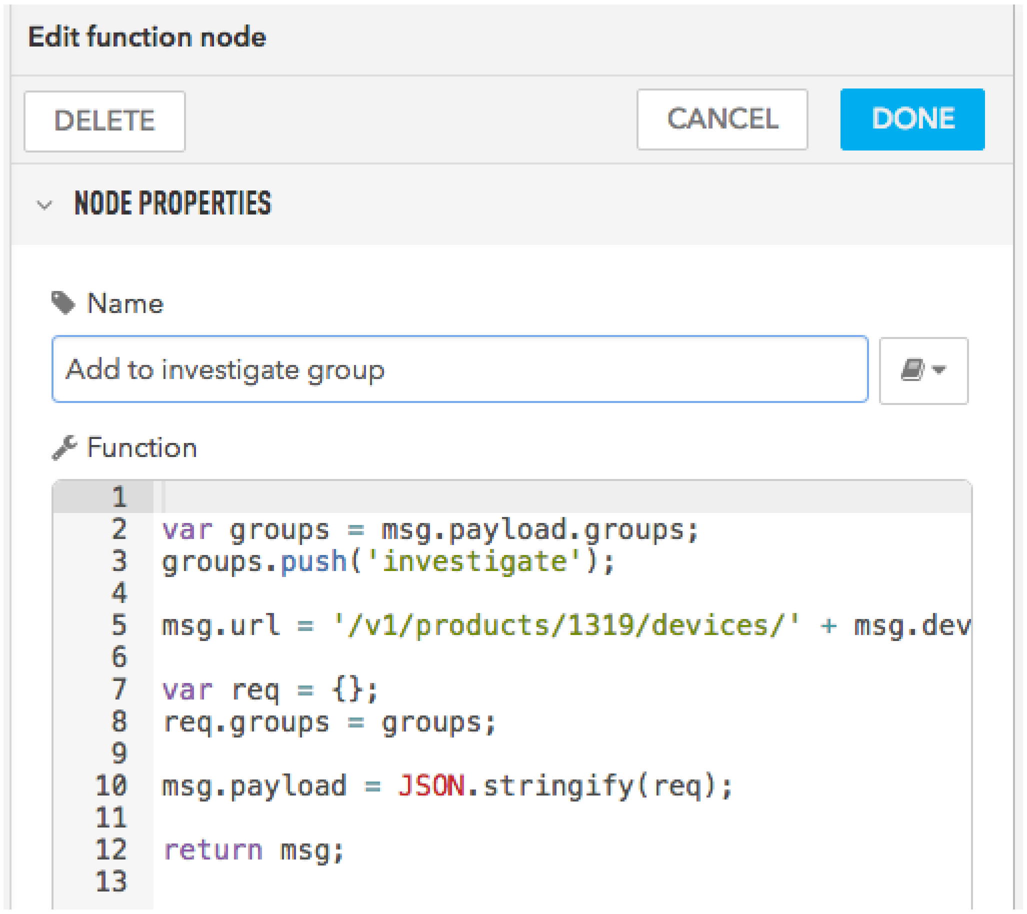

Edit particle api node

Name: Get Device Info
Auth: rulesengine-1703
Method: GET
URL: https://api.particle.io/[url]

- From the Function group, drag a function node to the workspace. Note that this is a Function function, not a Particle function!
- Double click to edit.
- Set the function body to:

```
var groups = msg.payload.groups;
groups.push('investigate');

msg.url = '/v1/products/1319/devices/' + msg.device;

var req = {};
req.groups = groups;

msg.payload = JSON.stringify(req);

return msg;
```

This adds "investigate" group to the existing device groups for this device and prepares the Particle API request.



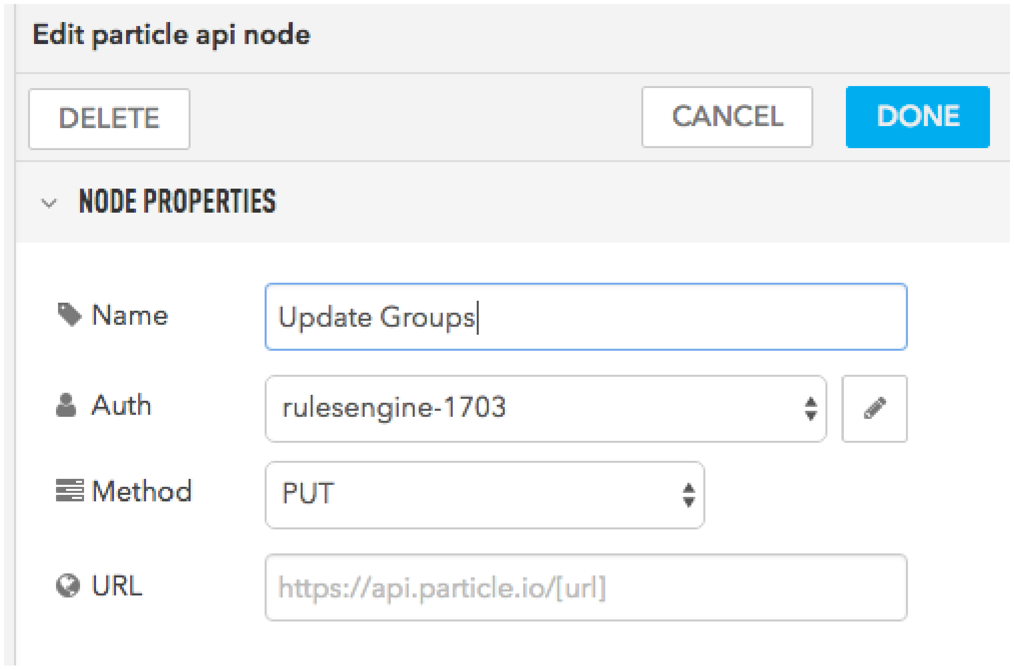
Name

Add to investigate group

Function

```

var groups = msg.payload.groups;
groups.push('investigate');

msg.url = '/v1/products/1319/devices/' + msg.dev

var req = {};
req.groups = groups;

msg.payload = JSON.stringify(req);

return msg;

```

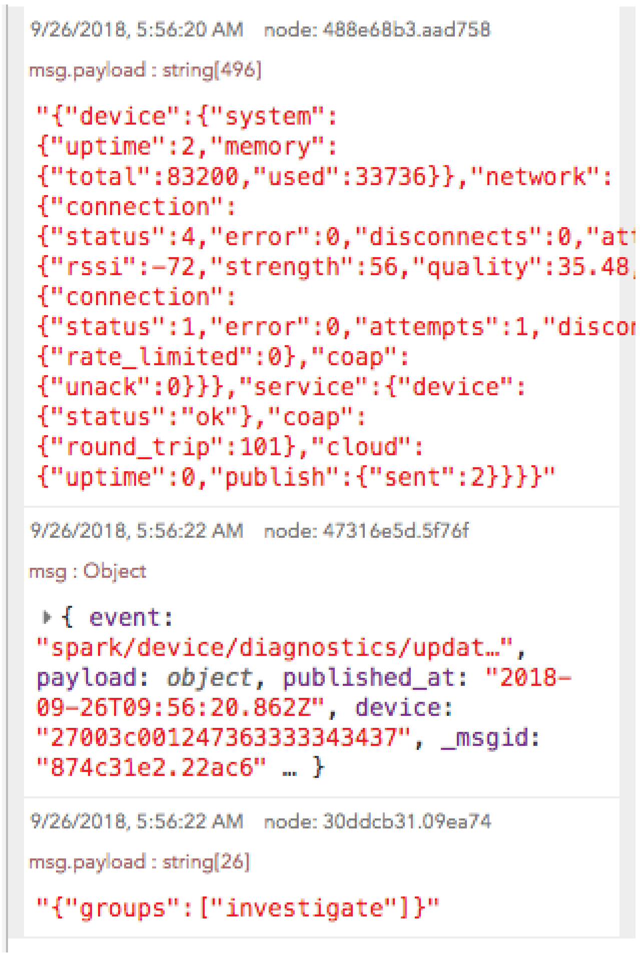
- In the Particle section, drag a **particle api** node into the workspace.
- Double click to edit.
- Set your product authentication.
- Set **Method PUT**.
- Leave the URL field blank, as it's set by the previous function node.



Name: Update Groups
Auth: rulesengine-1703
Method: PUT
URL: https://api.particle.io/[url]

- Connect the nodes as pictured above.
- Deploy!

The Rules Engine debug log should show something like this:



```
9/26/2018, 5:56:20 AM   node: 488e68b3.aad758
msg.payload : string[496]
"{"device":{"system":
{"uptime":2,"memory":
{"total":83200,"used":33736}},"network":
{"connection":
{"status":4,"error":0,"disconnects":0,"at
{"rssi":-72,"strength":56,"quality":35.48
{"connection":
{"status":1,"error":0,"attempts":1,"disco
{"rate_limited":0},"coap":
{"unack":0}}},"service":{"device":
{"status":"ok"},"coap":
{"round_trip":101},"cloud":
{"uptime":0,"publish":{"sent":2}}}}"

9/26/2018, 5:56:22 AM   node: 47316e5d.5f76f
msg : Object
▸{ event:
"spark/device/diagnostics/updat…",
payload: object, published_at: "2018-
09-26T09:56:20.862Z", device:
"27003c001247363333343437", _msgid:
"874c31e2.22ac6" … }

9/26/2018, 5:56:22 AM   node: 30ddcb31.09ea74
msg.payload : string[26]
"{"groups":["investigate"]}"
```

Showing tables of devices in the dashboard

In this section we will:

- Show a table of devices in the dashboard
- Automatically update the temperatures from all of the sensors

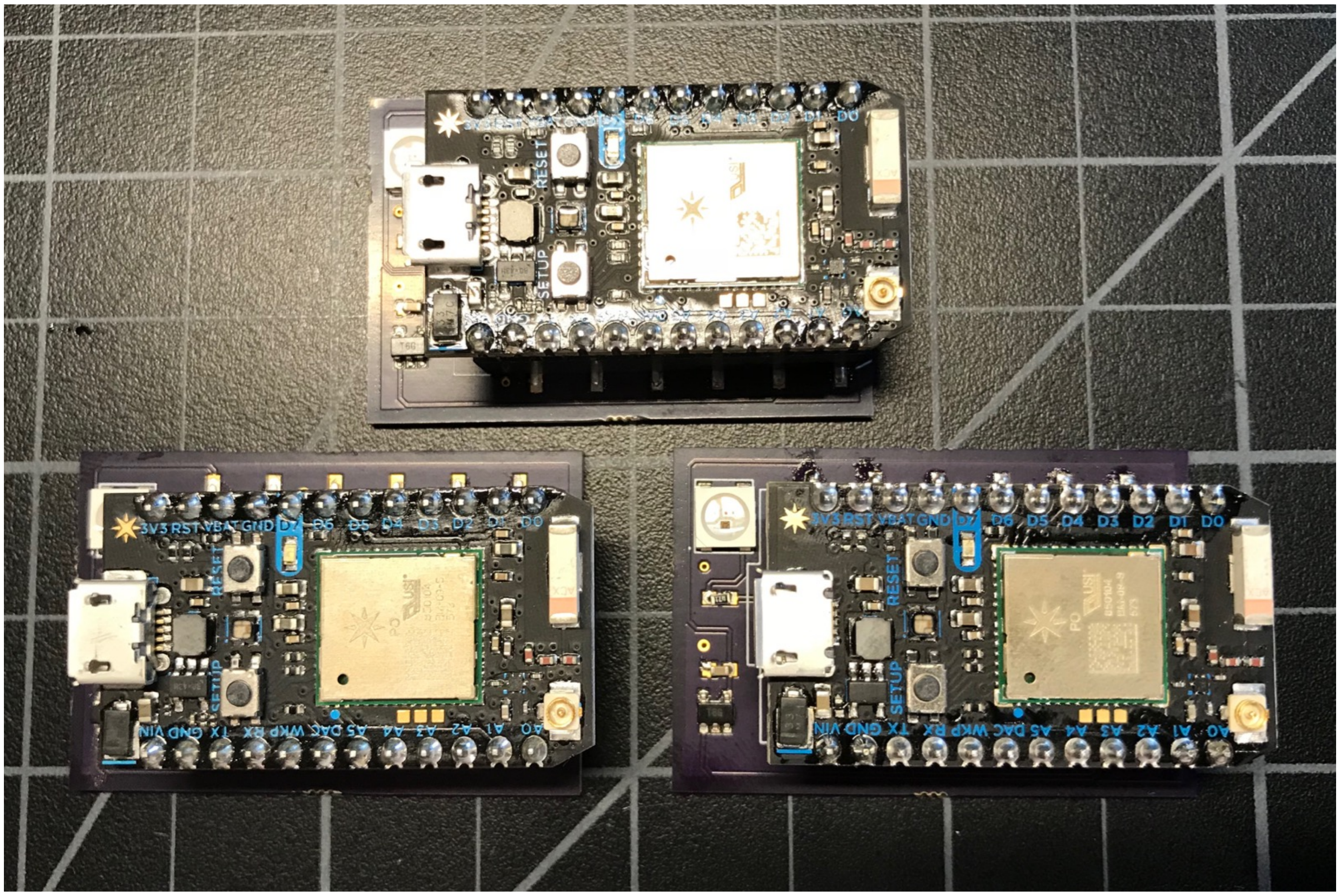
Adding on this, in the second part we will:

- Add rules so when the temperature is out of range, the LED turn red

And in the third part we will:

- Add device diagnostics to the table, to show the current Wi-Fi signal strength (RSSI) for the sensor

We'll be using our demonstration temperature monitor product. The hardware consists of a Photon, a temperature sensor, and a bright RGB LED for alerting.



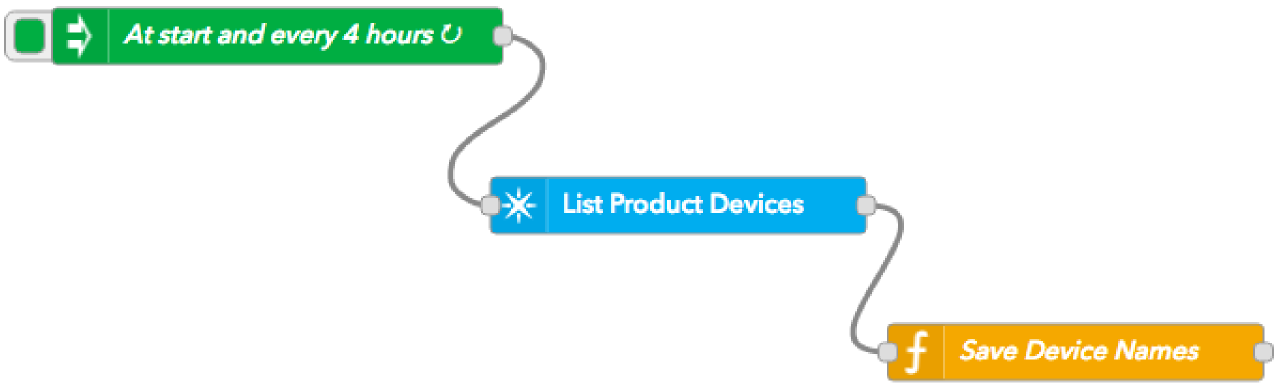
The device firmware can be found at this link.

This tutorial also shows how to use product mode to interact with a Particle product vs. individual developer devices.

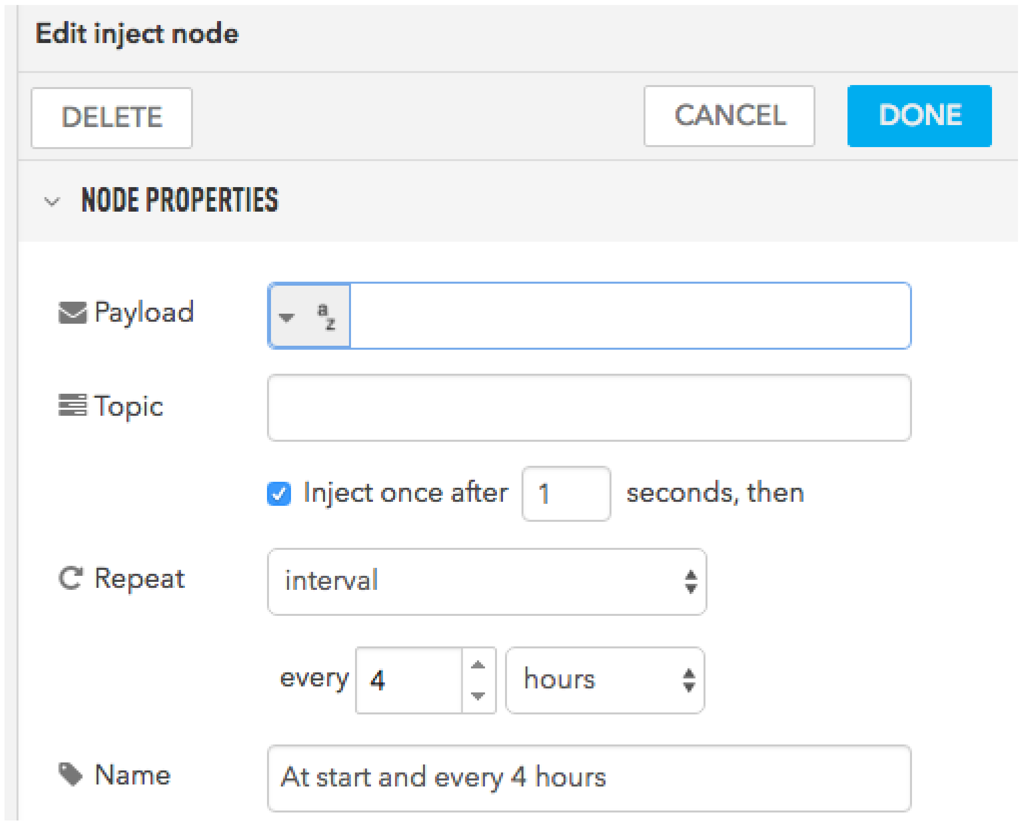
Product name flow

All of the flows in this section use this recipe to map the device IDs in the events into device names. It's a handy technique to include in your scripts.

We'll be setting up this flow:



- In the Input section, drag an **inject** node into the workspace.
- Double click to edit
- Set **Inject once after 1 seconds**
- Set **Repeat interval**
- Set **every 4 hours**



Payload

Topic

☑ Inject once after 1 seconds, then

Repeat: interval

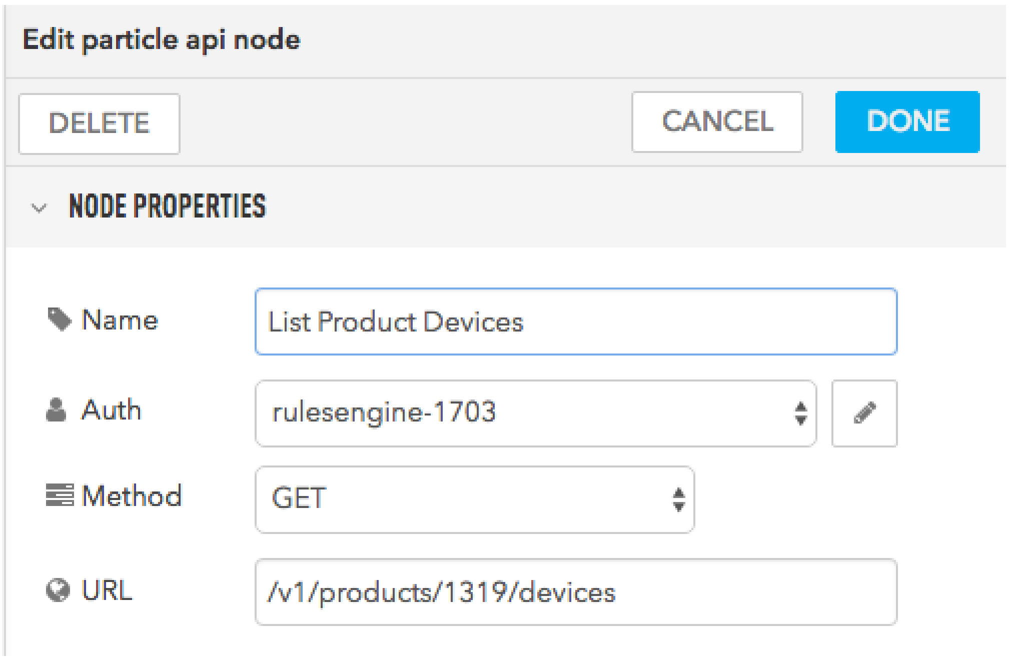
every 4 hours

Name: At start and every 4 hours

- In the Particle section, drag a **particle api** node into the workspace.

- Double click to edit
- Set your product authentication
- Set **Method GET**
- Set **URL /v1/products/1319/devices**

Make sure you select your product client ID and secret, not your personal account. Also make sure you change 1319 to your product ID.



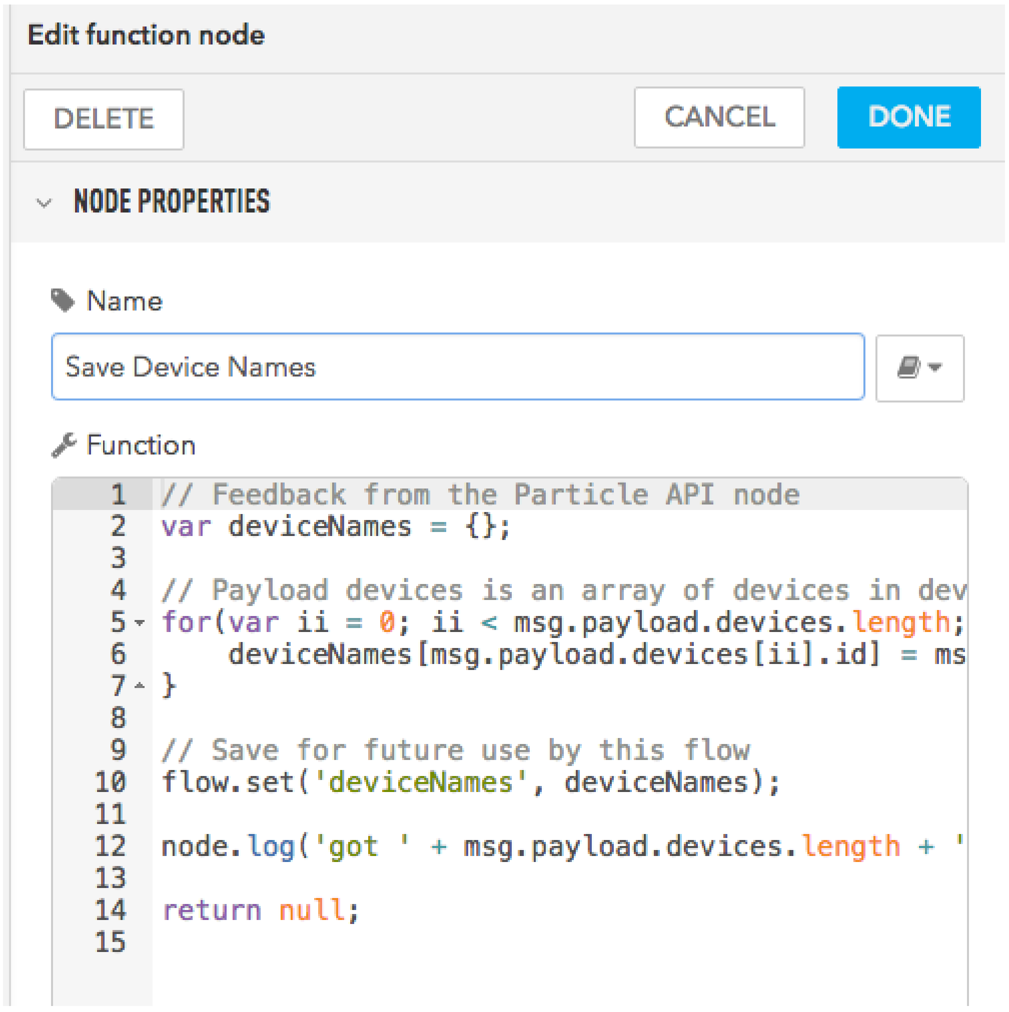
Name: List Product Devices
Auth: rulesengine-1703
Method: GET
URL: /v1/products/1319/devices

The last node in the flow saves the data so it can be used by other nodes in the flow.

- 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:

```
// Feedback from the Particle API node
var deviceNames = {};

// Payload devices is an array of devices in devices for this product
for(var ii = 0; ii < msg.payload.devices.length; ii++) {
    deviceNames[msg.payload.devices[ii].id] = msg.payload.devices[ii].name;
}

// Save for future use by this flow
flow.set('deviceNames', deviceNames);
flow.set('deviceNames', deviceNames);

node.log('got ' + msg.payload.devices.length + ' device names');

return null;
```

What this does is that the response from the list devices API request, and build a table (hash) of mappings from device ID to device name and saves it so other nodes in the flow can access the information.



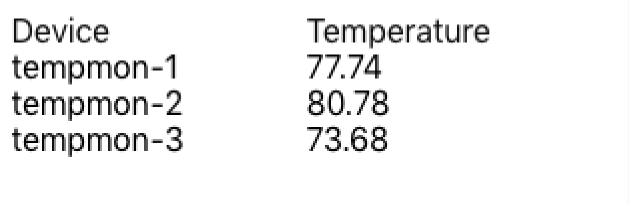
Name: Save Device Names

Function

```
// Feedback from the Particle API node
var deviceNames = {};

// Payload devices is an array of devices in dev
for(var ii = 0; ii < msg.payload.devices.length;
    deviceNames[msg.payload.devices[ii].id] = ms
}

// Save for future use by this flow
flow.set('deviceNames', deviceNames);

node.log('got ' + msg.payload.devices.length + '

return null;
```

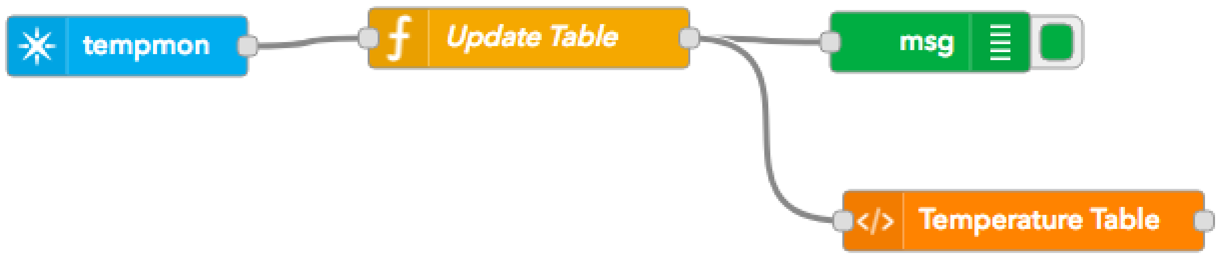
Show temperature in dashboard

While this example uses temperatures, you could use this technique for displaying a table of any values generated by your sensors.

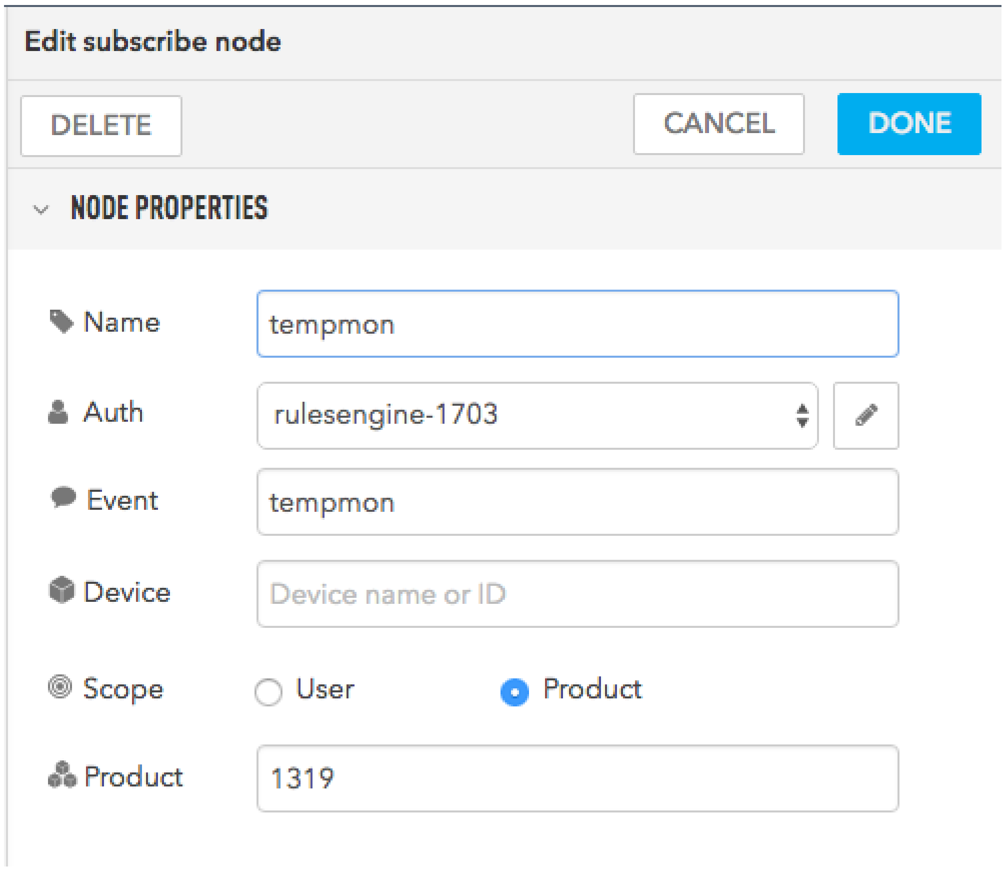


Device	Temperature
tempmon-1	77.74
tempmon-2	80.78
tempmon-3	73.68

We'll be creating this flow to process the tempmon events.



- From the Particle group, drag a **subscribe** node to the workspace.
- Double click to edit.
- Set the Name, Auth, and Event.
- Leave the Device field blank (allow all devices)
- Set the Scope to **Product** and set your Product ID (mine is 1319).



Name: tempmon
Auth: rulesengine-1703
Event: tempmon
Device: Device name or ID
Scope: User [✓ Product]
Product: 1319

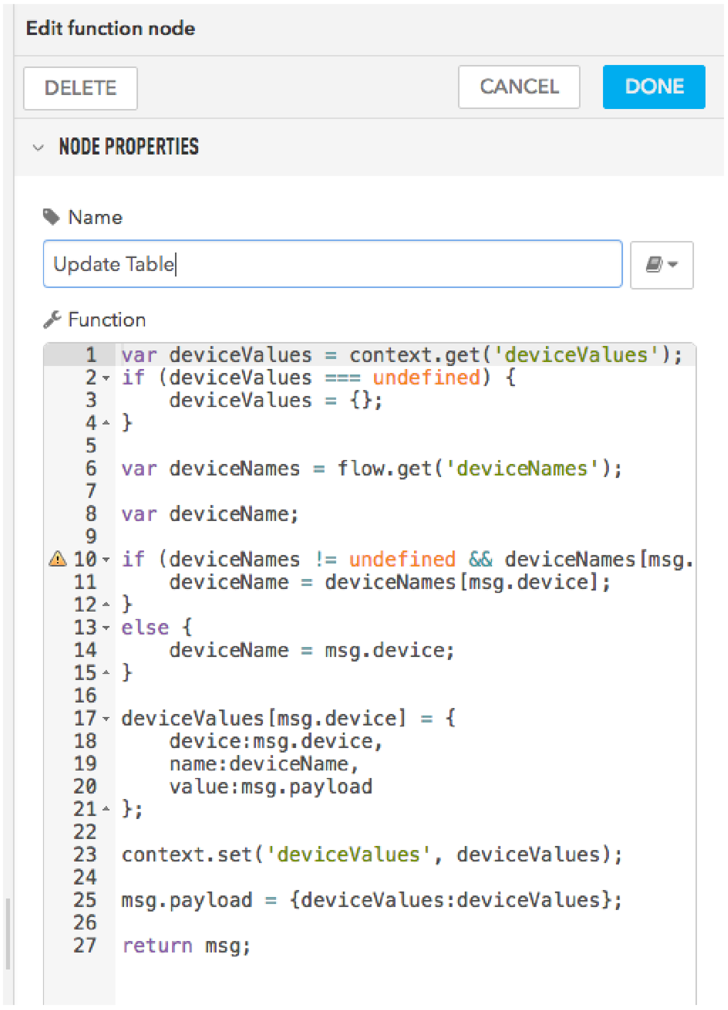
- From the Function group, drag a function node to the workspace. Note that this is a Function function, not a Particle function!
- Double click to edit.
- Set the function body to:

```
var deviceValues = context.get('deviceValues');
if (deviceValues === undefined) {
    deviceValues = {};
}

var deviceNames = flow.get('deviceNames');

var deviceName;

if (deviceNames != undefined && deviceNames[msg.device]) {
    deviceName = deviceNames[msg.device];
}
else {
    deviceName = msg.device;
}
```

```
deviceValues[msg.device] = {
    device:msg.device,
    name:deviceName,
    value:msg.payload
};

context.set('deviceValues', deviceValues);

msg.payload = {deviceValues:deviceValues};

return msg;
```

What this does is:

- Get the saved deviceNames object for this flow, creating it if necessary.
- Convert the device ID to a name (if possible).
- Save the device ID, device name, and the value



Name: Update Table

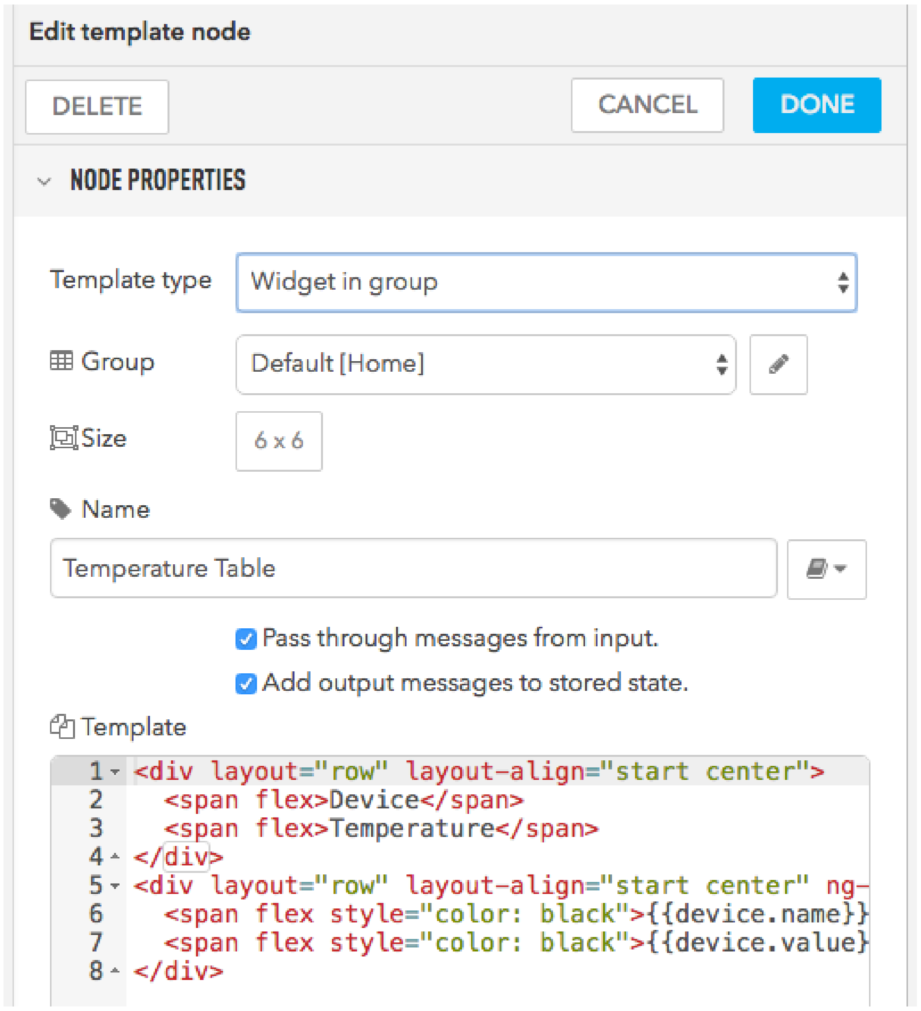
Function

```
var deviceValues = context.get('deviceValues');
if (deviceValues === undefined) {
    deviceValues = {};
}

var deviceNames = flow.get('deviceNames');

var deviceName;

if (deviceNames != undefined && deviceNames[msg.
    deviceName = deviceNames[msg.device];
}
else {
    deviceName = msg.device;
}

deviceValues[msg.device] = {
    device:msg.device,
    name:deviceName,
    value:msg.payload
};

context.set('deviceValues', deviceValues);

msg.payload = {deviceValues:deviceValues};

return msg;
```

- From the Dashboard group, drag a **template** node to the workspace.
- Set the template as follows:

```
<div layout="row" layout-align="start center">
  <span flex>Device</span>
  <span flex>Temperature</span>
</div>
<div layout="row" layout-align="start center" ng-repeat="device in
msg.payload.deviceValues">
  <span flex style="color: black">{{device.name}}</span>
  <span flex style="color: black">{{device.value}}</span>
</div>
```

What this does is create an automatically expanding table of devices.



Template type: Widget in group

Group: Default [Home]

Size: 6 x 6

Name: Temperature Table

☑ Pass through messages from input.

☑ Add output messages to stored state.

Template

```
<div layout="row" layout-align="start center">
  <span flex>Device</span>
  <span flex>Temperature</span>
</div>
<div layout="row" layout-align="start center" ng-
  <span flex style="color: black">{{device.name}}
  <span flex style="color: black">{{device.value}
</div>
```

- Deploy the flow and view the dashboard

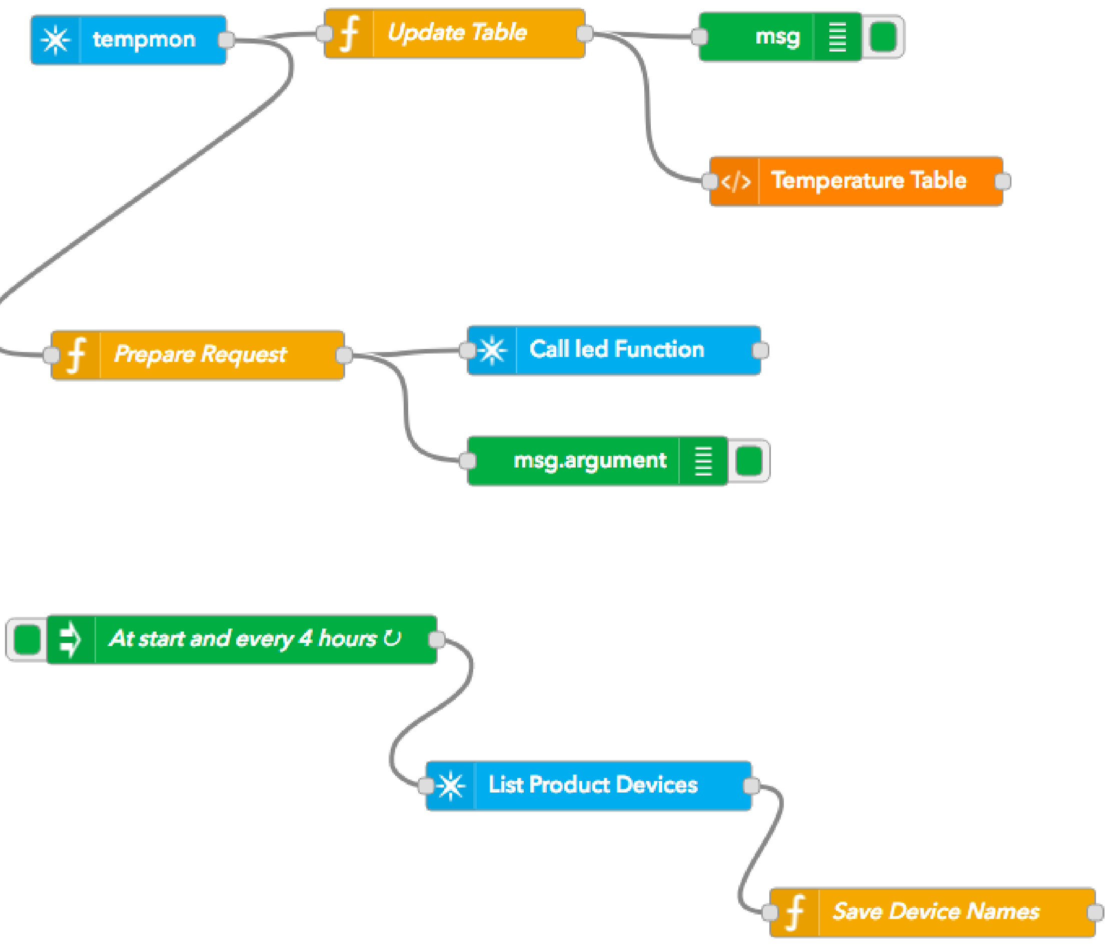
The output should look like the table in the beginning of this section.

Add business logic and feedback

One thing you can do with the Rules Engine is put your business logic in a more easily edited Rules Engine. For example, we'll add a feature to turn the LED on the board red when the temperature is too high.

Rather than embedding the temperatures and logic in the device firmware, we can put it in the Rules Engine. This makes it easy to change the limits without rebuilding and deploying code.

Also, you have complete flexibility. Want to have multiple levels so when the temperature gets really high it blinks red? It's a simple change in the Rules Engine!



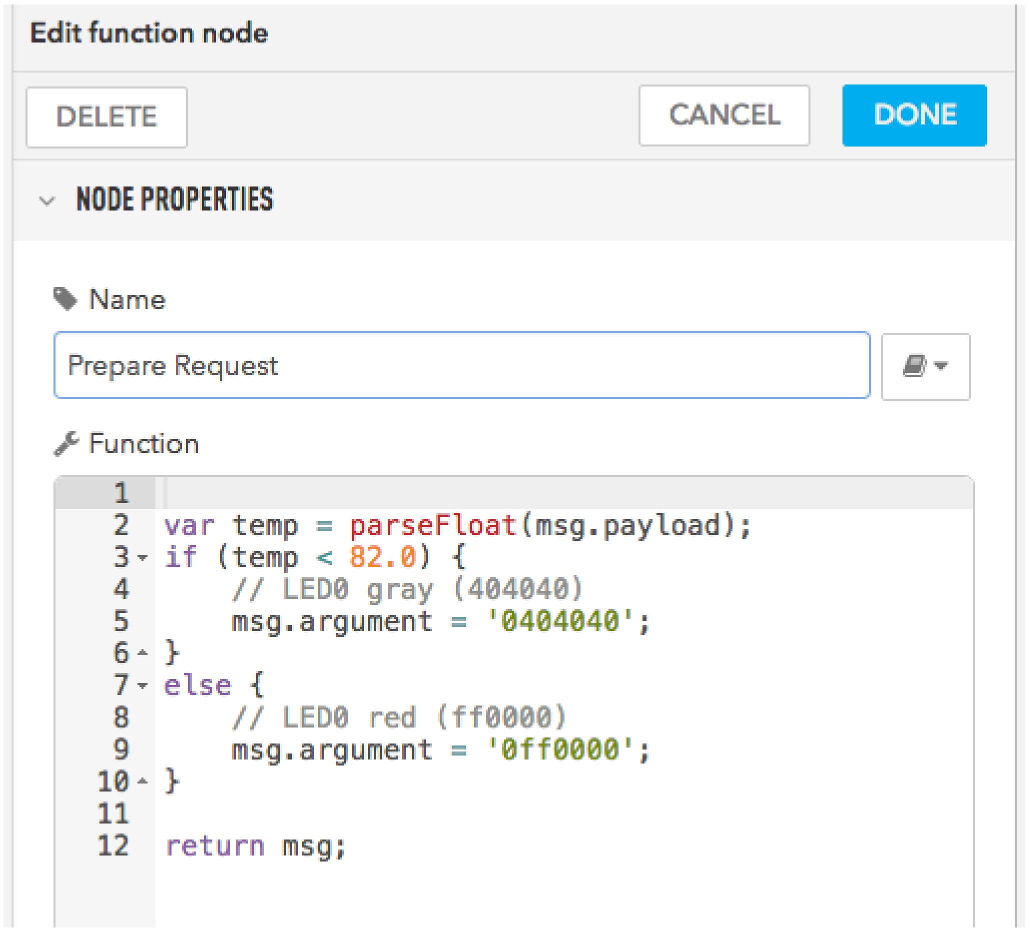
This builds on the previous example.

- From the Function group, drag a function node to the workspace. Note that this is a Function function, not a Particle function!
- Double click to edit.
- Set the function body to:

```
var temp = parseFloat(msg.payload);
if (temp < 82.0) {
    // LED0 gray (404040)
    msg.argument = '0404040';
}
else {
    // LED0 red (ff0000)
    msg.argument = '0ff0000';
}

return msg;
```

This is the business logic. It maps a temperature threshold (82.0) to a command to device firmware.

- Setting the argument to 0404040 sets the LED to light gray.
- Setting the argument to 0ff0000 sets the LED to red.

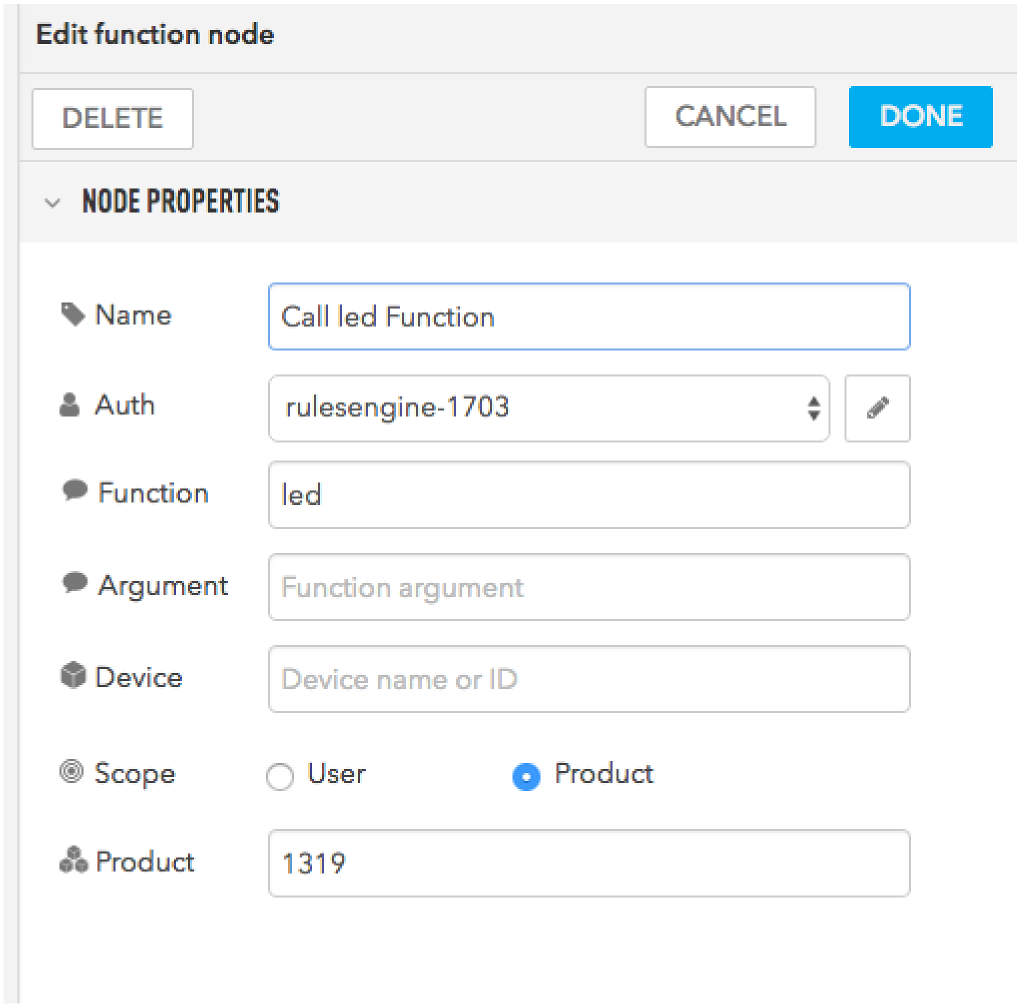


Name: Prepare Request

Function

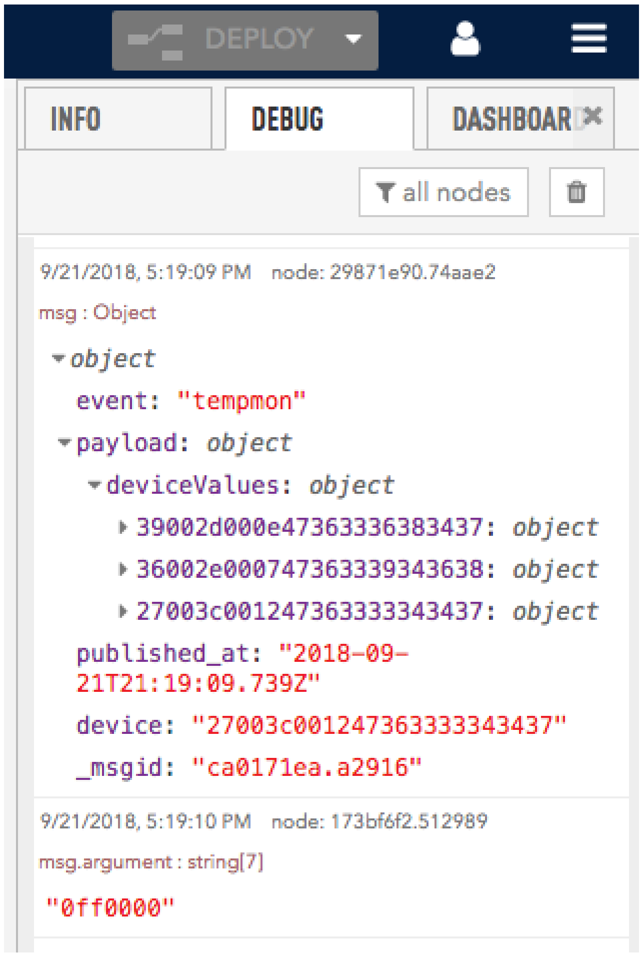
```

var temp = parseFloat(msg.payload);
if (temp < 82.0) {
    // LED0 gray (404040)
    msg.argument = '0404040';
}
else {
    // LED0 red (ff0000)
    msg.argument = '0ff0000';
}

return msg;
```

- From the Particle group, drag a **function** to the workspace.
- Set the parameters as follows.
- Make sure you set the Product to your Product ID



Name	Call led Function
Auth	rulesengine-1703
Function	led
Argument	Function argument
Device	Device name or ID
Scope	User [✓ Product]
Product	1319

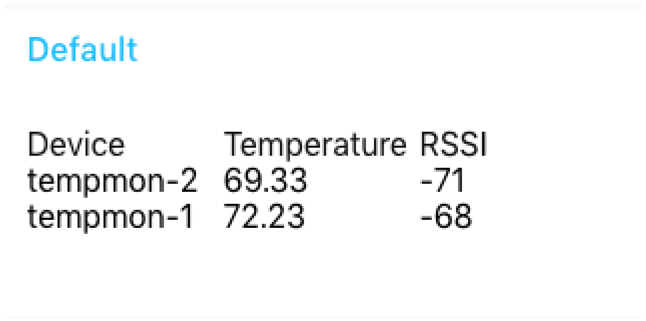
The Debug log in the Rules Engine can be helpful in diagnosing any problems you encounter.



```
9/21/2018, 5:19:09 PM   node: 29871e90.74aae2
msg : Object
▾object
  event: "tempmon"
 ▾payload: object
   ▾deviceValues: object
     ▸39002d000e47363336383437: object
     ▸36002e000747363339343638: object
     ▸27003c001247363333343437: object
  published_at: "2018-09-
  21T21:19:09.739Z"
  device: "27003c001247363333343437"
  _msgid: "ca0171ea.a2916"
9/21/2018, 5:19:10 PM   node: 173bf6f2.512989
msg.argument : string[7]
"0ff0000"
```

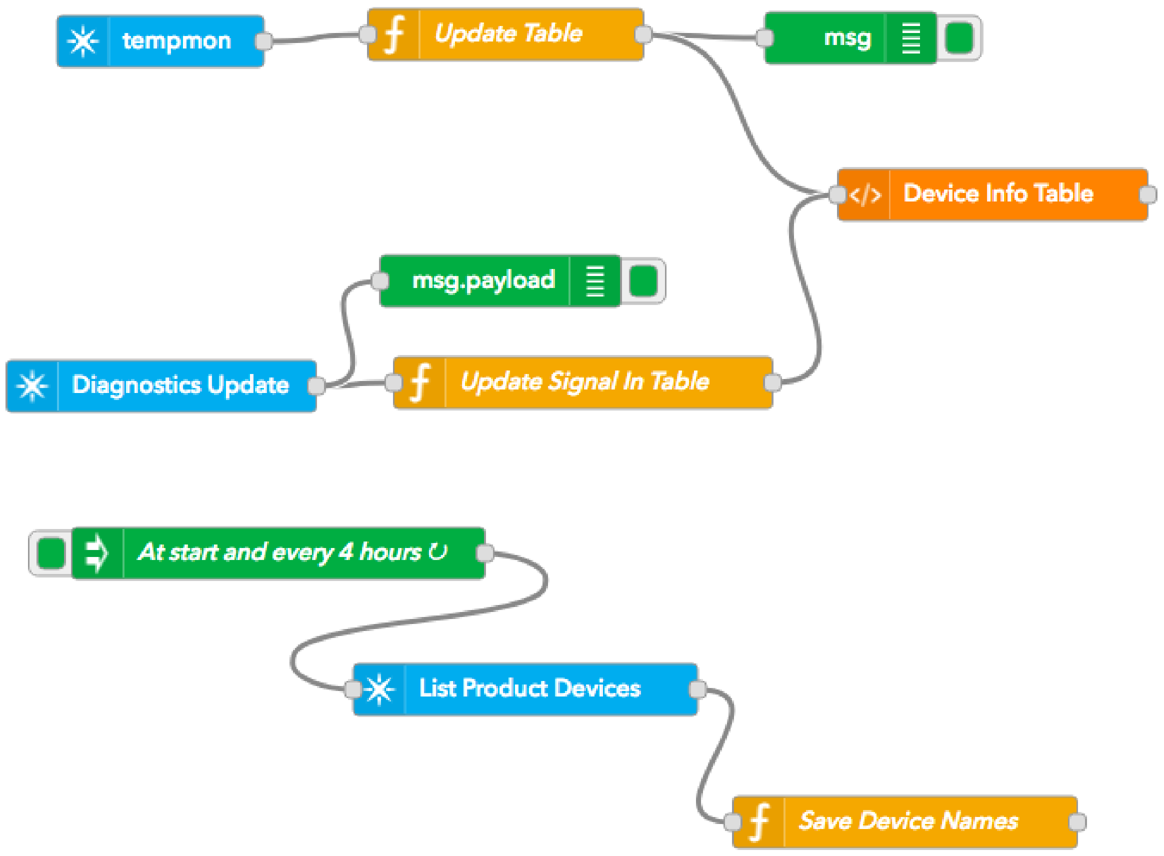
Add device diagnostics

While the table of temperatures is nice, you can use the Rules Engine to mix in all sorts of other information. In this example, we use the device diagnostics feature to log the Wi-Fi signal strength (RSSI) in the table as well.

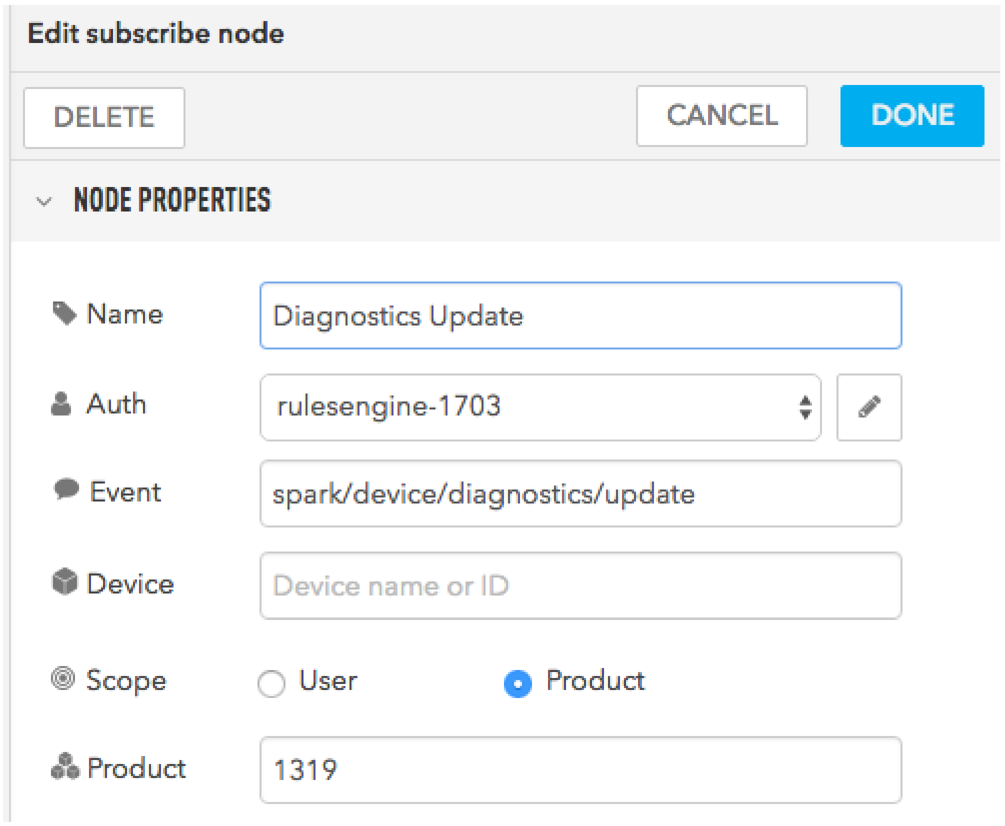

Default

Device	Temperature	RSSI
tempmon-2	69.33	-71
tempmon-1	72.23	-68

We'll just add a few nodes on the table example.



- From the Particle group, drag a **subscribe** node to the workspace.
- Double click to edit.
- Set the Name, Auth, and Event ("spark/device/diagnostics/update").
- Leave the Device field blank (allow all devices)
- Set the Scope to **Product** and set your Product ID (mine is 1319).



Name: Diagnostics Update

Auth: rulesengine-1703

Event: spark/device/diagnostics/update

Device: Device name or ID

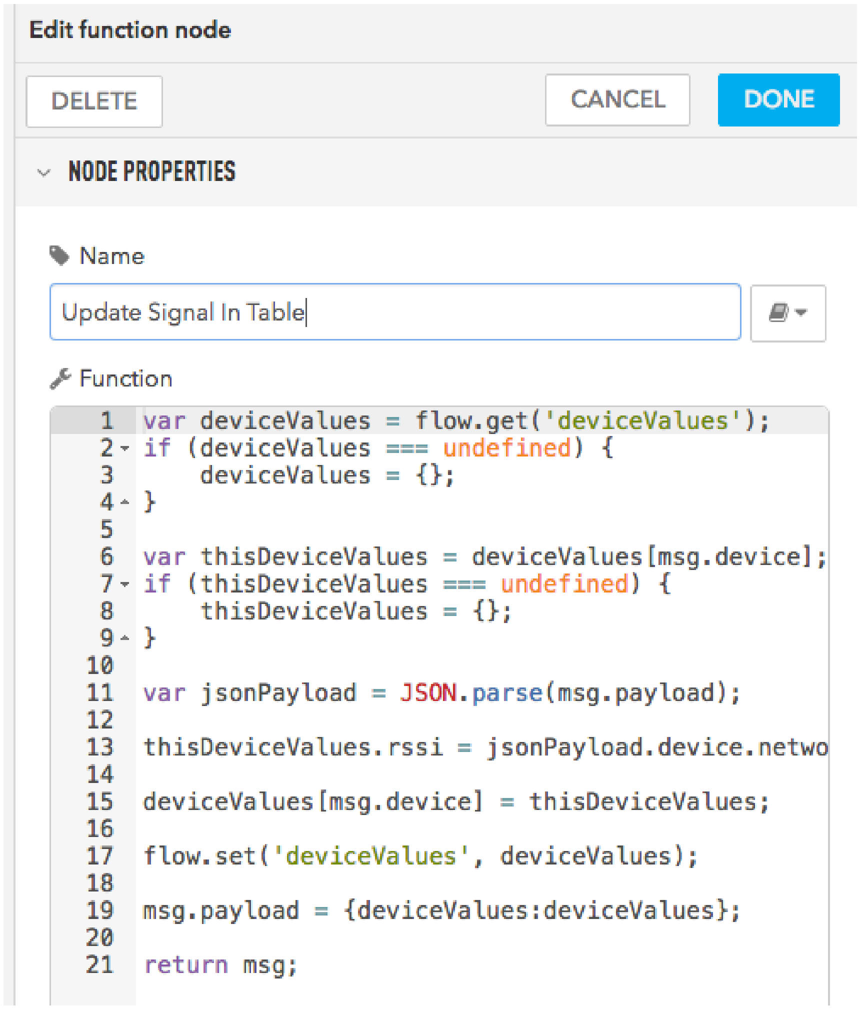
Scope: User [✓ Product]

Product: 1319

- From the Function group, drag a function node to the workspace. Note that this is a Function function, not a Particle function!
- Double click to edit.
- Set the function body to:

```
var deviceValues = flow.get('deviceValues');
if (deviceValues === undefined) {
    deviceValues = {};
}

var thisDeviceValues = deviceValues[msg.device];
if (thisDeviceValues === undefined) {
    thisDeviceValues = {};
}

var jsonPayload = JSON.parse(msg.payload);

thisDeviceValues.rssi = jsonPayload.device.network.signal.rssi;

deviceValues[msg.device] = thisDeviceValues;

flow.set('deviceValues', deviceValues);
```

```
msg.payload = {deviceValues:deviceValues};

return msg;
```

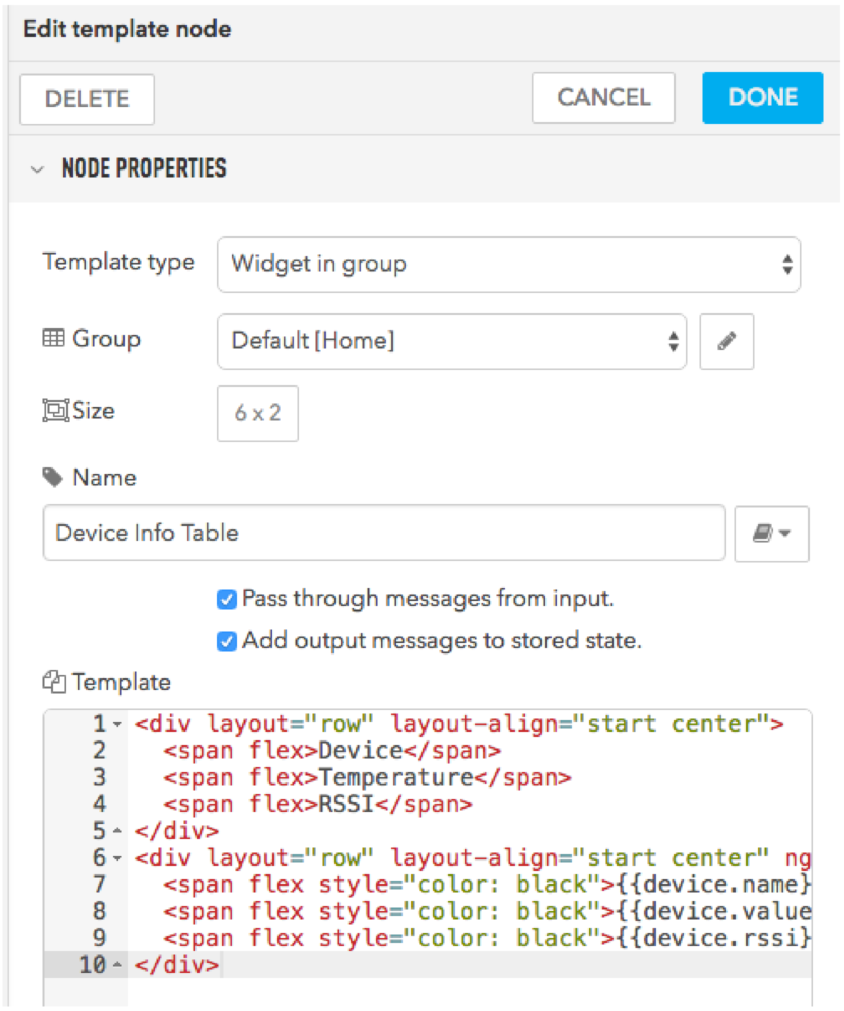


```
var deviceValues = flow.get('deviceValues');
if (deviceValues === undefined) {
    deviceValues = {};
}

var thisDeviceValues = deviceValues[msg.device];
if (thisDeviceValues === undefined) {
    thisDeviceValues = {};
}

var jsonPayload = JSON.parse(msg.payload);

thisDeviceValues.rssi = jsonPayload.device.netwo

deviceValues[msg.device] = thisDeviceValues;

flow.set('deviceValues', deviceValues);

msg.payload = {deviceValues:deviceValues};

return msg;
```

- Connect the output of the function node Update Signal In Table into the Device Info Table node.
- Edit the Device Info Table node to add the new columns to the template:

```
<div layout="row" layout-align="start center">
  <span flex>Device</span>
  <span flex>Temperature</span>
  <span flex>RSSI</span>
</div>
<div layout="row" layout-align="start center" ng-repeat="device in
msg.payload.deviceValues">
```

```
    <span flex style="color: black">{{device.name}}</span>
    <span flex style="color: black">{{device.value}}</span>
    <span flex style="color: black">{{device.rssi}}</span>
  </div>
```



Template type: Widget in group

Group: Default [Home]

Size: 6 x 2

Name: Device Info Table

☑ Pass through messages from input.

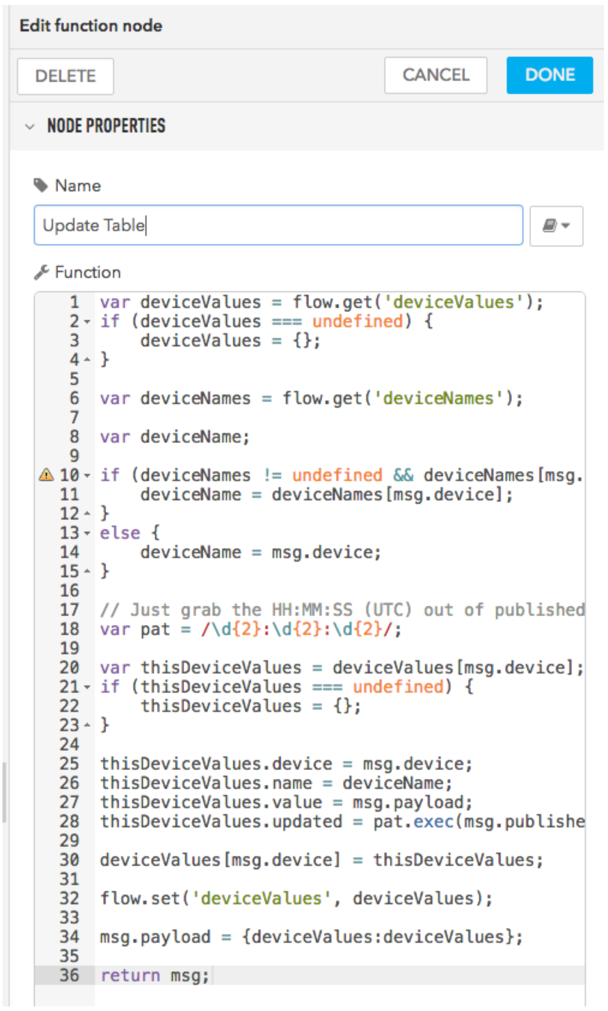
☑ Add output messages to stored state.

Template

```
<div layout="row" layout-align="start center">
  <span flex>Device</span>
  <span flex>Temperature</span>
  <span flex>RSSI</span>
</div>
<div layout="row" layout-align="start center" ng
  <span flex style="color: black">{{device.name}
  <span flex style="color: black">{{device.value
  <span flex style="color: black">{{device.rssi}
</div>
```

- Edit the Update Table node code
- This code allows the deviceValues data to include both the temperature and RSSI data, coming from two different sources.

```
var deviceValues = flow.get('deviceValues');
if (deviceValues === undefined) {
    deviceValues = {};
}

var deviceNames = flow.get('deviceNames');
var deviceName;

if (deviceNames != undefined && deviceNames[msg.device]) {
    deviceName = deviceNames[msg.device];
}
else {
    deviceName = msg.device;
}

// Just grab the HH:MM:SS (UTC) out of published_at
var pat = /\d{2}:\d{2}:\d{2}/;

var thisDeviceValues = deviceValues[msg.device];
if (thisDeviceValues === undefined) {
    thisDeviceValues = {};
}

thisDeviceValues.device = msg.device;
thisDeviceValues.name = deviceName;
thisDeviceValues.value = msg.payload;
thisDeviceValues.updated = pat.exec(msg.published_at)[0];

deviceValues[msg.device] = thisDeviceValues;

flow.set('deviceValues', deviceValues);

msg.payload = {deviceValues:deviceValues};

return msg;
```



Update Table

```
var deviceValues = flow.get('deviceValues');
if (deviceValues === undefined) {
    deviceValues = {};
}

var deviceNames = flow.get('deviceNames');

var deviceName;

if (deviceNames != undefined && deviceNames[msg.
    deviceName = deviceNames[msg.device];
}
else {
    deviceName = msg.device;
}

// Just grab the HH:MM:SS (UTC) out of published
var pat = /\d{2}:\d{2}:\d{2}/;

var thisDeviceValues = deviceValues[msg.device];
if (thisDeviceValues === undefined) {
    thisDeviceValues = {};
}

thisDeviceValues.device = msg.device;
thisDeviceValues.name = deviceName;
thisDeviceValues.value = msg.payload;
thisDeviceValues.updated = pat.exec(msg.publishe

deviceValues[msg.device] = thisDeviceValues;

flow.set('deviceValues', deviceValues);

msg.payload = {deviceValues:deviceValues};

return msg;
```