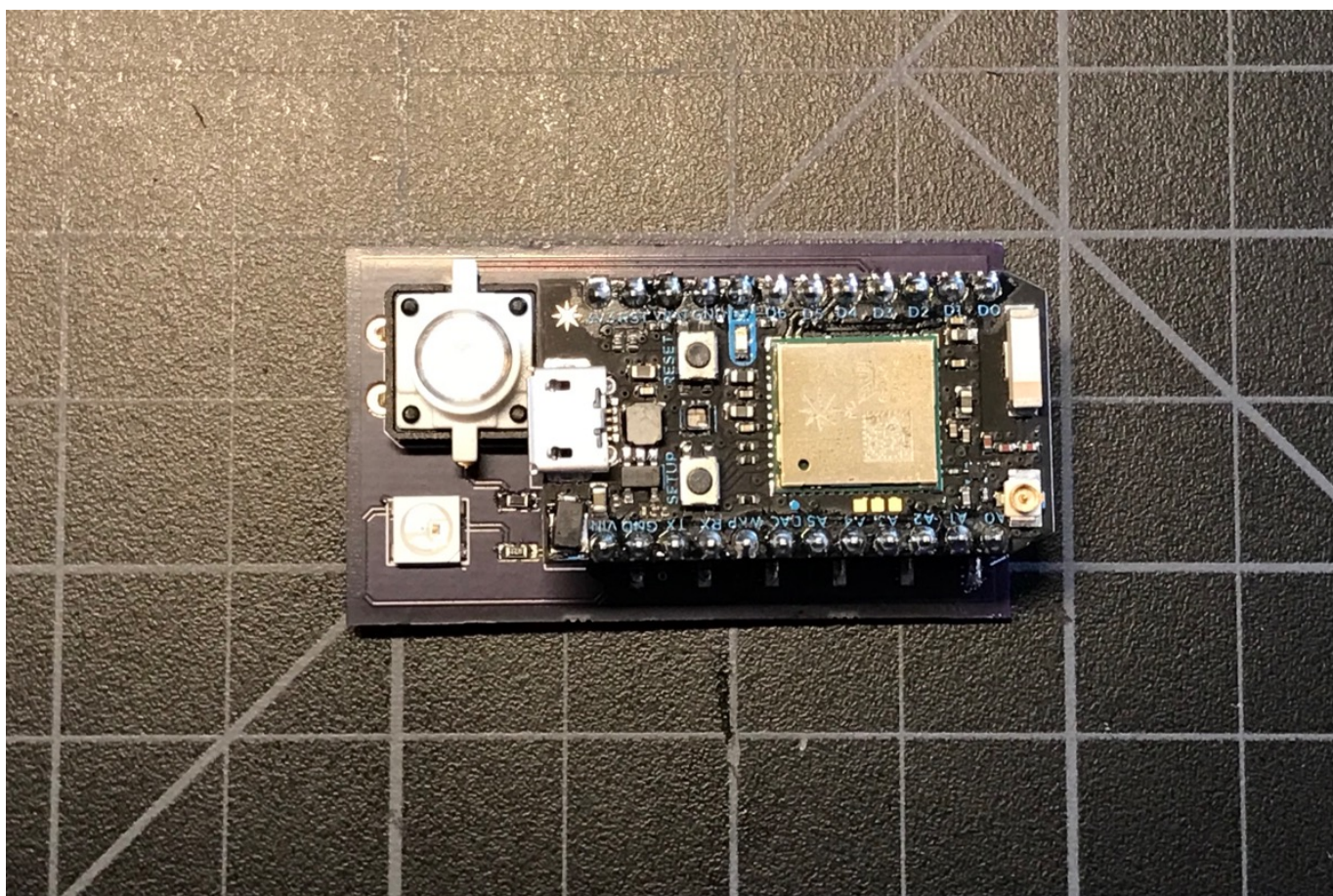


DYNAMIC FIRMWARE MANAGEMENT WITH RULES ENGINE

In many cases, adding business logic around when over-the-air (OTA) firmware updates are delivered can lead to a better end-user experience. This automation can make OTA updates more seamless and intelligent, with less need for manual intervention.

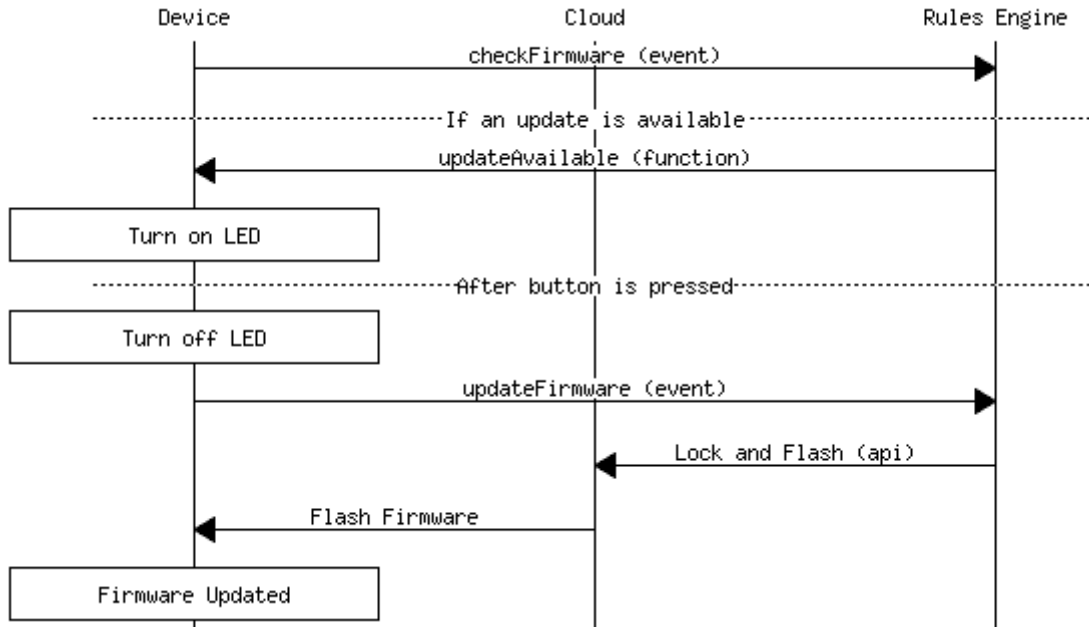
This tutorial shows how to dynamically update application firmware running on a device using the Rules Engine.

The sample hardware is a Photon with an illuminated button. When there's a firmware update available, the button lights up. Pressing the button installs the update!



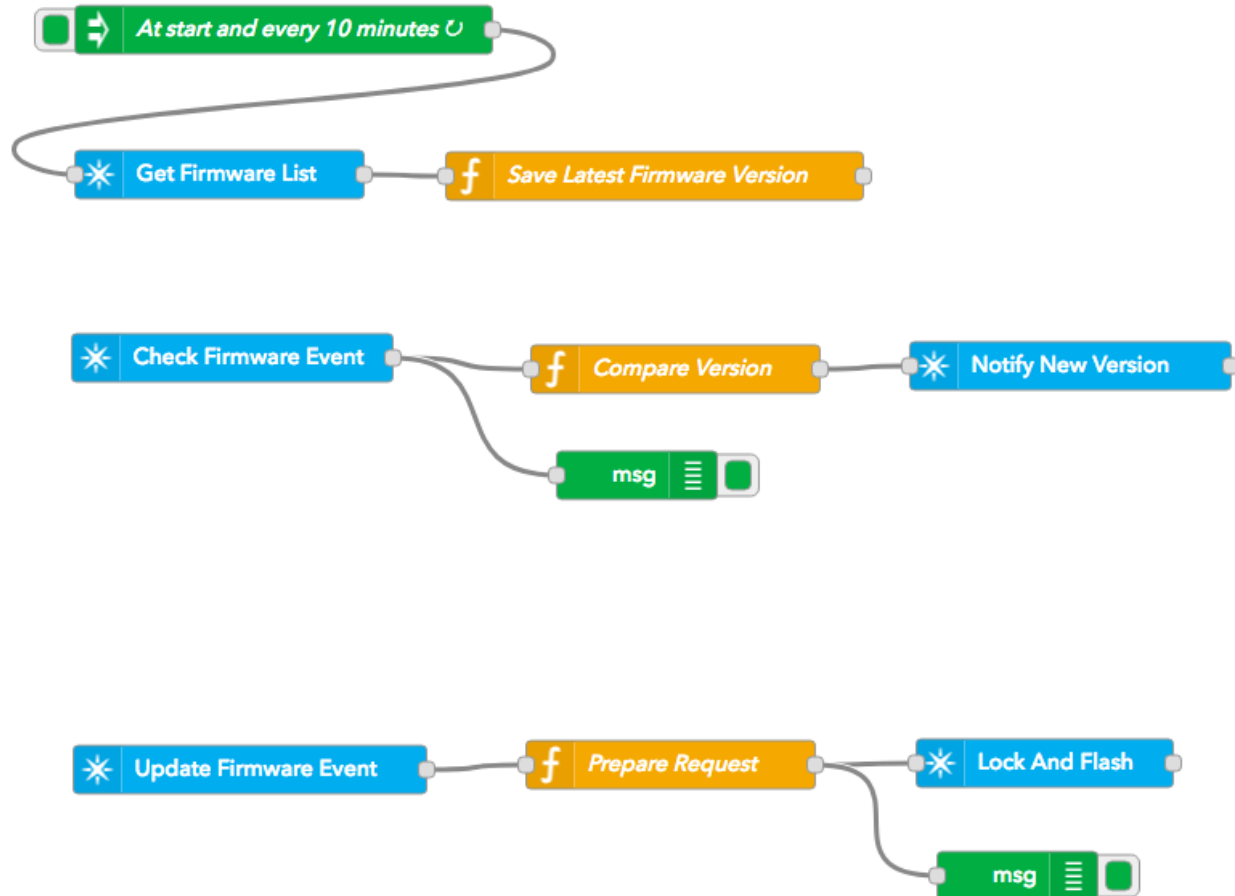
- At startup and every 10 minutes, the Photon sends a `checkFirmware` event to the cloud with its current product firmware version number.
- If there is a newer version of product firmware available, the Rules Engine calls the device's `updateAvailable` function.

- The function handler turns on the LED within the button. This is the there's a firmware update available for this device signal.
- As soon as the user presses the illuminated button, the Rules Engine will initiate a lock and flash for the device, causing it to be updated.



The Rules Engine flow has three parts:

- Getting the firmware version list
- Handling the `checkFirmware` event
- Handling the `updateFirmware` event

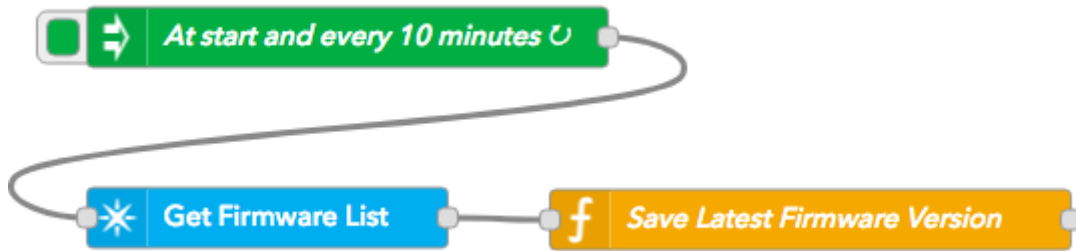


Drag the Copy Rules button into the Rules Engine window to create the flow automatically, or you can create the flow from scratch with the steps below.

Copy Rules ▶

Getting the firmware version list

In this section we'll get the list of firmware versions using the Particle cloud API and save the version number of the most recent version.



- In the Input section, drag an **inject** node into the workspace.
- Double click to edit
- Set **Inject once after 2 seconds**
- Set **Repeat interval**
- Set **every 10 minutes**

This will retrieve the list of firmware versions at startup and every 10 minutes thereafter.

Edit inject node

DELETE
CANCEL
DONE

▼ **NODE PROPERTIES**

✉ Payload a_z

☰ Topic

Inject once after 2 seconds, then

↻ Repeat interval

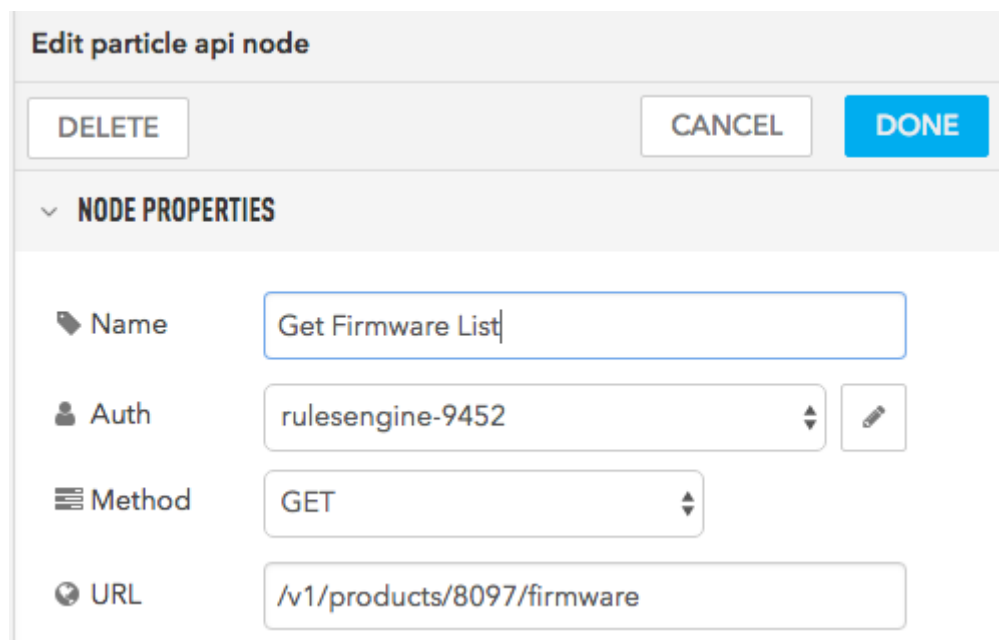
every 10 minutes

🔖 Name At start and every 10 minutes

- 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/8097/firmware**

Make sure you select your product client ID and secret, not your personal account. Also make sure you change 8097 to your product ID.



Edit particle api node

DELETE CANCEL DONE

▼ **NODE PROPERTIES**

Name Get Firmware List

Auth rulesengine-9452

Method GET

URL /v1/products/8097/firmware

- 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:

```
var latest = 0;

for(var ii = 0; ii < msg.payload.length; ii++) {
  if (msg.payload[ii].version > latest) {
    latest = msg.payload[ii].version;
  }
}

flow.set('latestFirmwareVersion', latest);

msg.payload = latest;

return msg;
```

This saves the latest version number the flow variable `latestFirmwareVersion` so it can be used by other parts of the flow.

Edit function node

DELETE CANCEL DONE

▼ NODE PROPERTIES

Name

Save Latest Firmware Version

Function

```
1 var latest = 0;
2
3 for(var ii = 0; ii < msg.payload.length; ii++) {
4     if (msg.payload[ii].version > latest) {
5         latest = msg.payload[ii].version;
6     }
7 }
8
9 flow.set('latestFirmwareVersion', latest);
10
11 msg.payload = latest;
12
13 return msg;
```

Handling the checkFirmware event

In this section we'll handle the `checkFirmware` event that the device sends to the cloud when it wants to know if there's new firmware.



- From the Particle group, drag a **subscribe** node to the workspace.
- Double click to edit.
- Set the Name, Auth, and Event ("checkFirmware").
- Leave the Device field blank (allow all devices)
- Set the Scope to **Product** and set your Product ID (mine is 8097).

Edit subscribe node

DELETE CANCEL DONE

▼ **NODE PROPERTIES**

Name

Auth

Event

Device

Scope User Product

Product

- 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:

```
var deviceFirmwareVersion = parseInt(msg.payload);

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

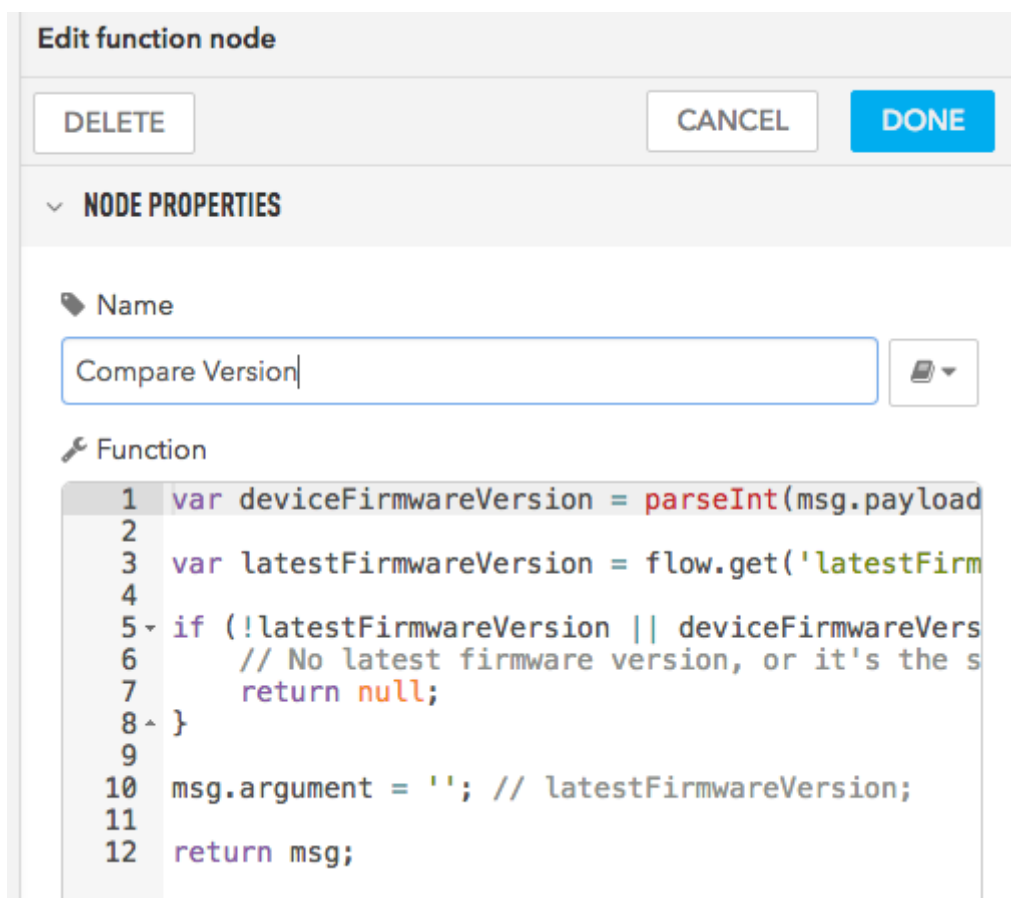
if (!latestFirmwareVersion || deviceFirmwareVersion >= latestFirmwareVersion)
{
    // No latest firmware version, or it's the same or newer
    return null;
}
```

```
}  
  
msg.argument = ''; // latestFirmwareVersion;  
  
return msg;
```

What this does is compare the firmware version sent from the device (deviceFirmwareVersion) to the latestFirmwareVersion that the previous flow generated.

If there is no update available, the flow returns null, which means the remainder of the nodes in the flow will not be executed.

If there is an update, control will pass to the next node.



- From the Particle group, drag a **function** node to the workspace. Note that this one is a Particle function, not a Function function!
- Double click to edit.
- Set the Name, Auth, and function ("updateAvailable")

- Set the Scope to **Product** and set your Product ID (mine is 8097).



This node uses the Particle function to call the `updateAvailable` function handler in the device firmware. This turns on the LED in the button so the user knows there is an update available.

Edit function node

DELETE CANCEL DONE

▼ **NODE PROPERTIES**

Name

Auth  

Function

Argument

Device

Scope User Product

Product

Handling the updateFirmware event



This section handles the request from the device to update its firmware now.

- From the Particle group, drag a **subscribe** node to the workspace.
- Double click to edit.
- Set the Name, Auth, and Event ("updateFirmware").
- Leave the Device field blank (allow all devices)
- Set the Scope to **Product** and set your Product ID (mine is 8097).

Edit subscribe node

DELETE CANCEL DONE

▼ NODE PROPERTIES

Name

Auth

Event

Device

Scope User Product

Product

- 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:

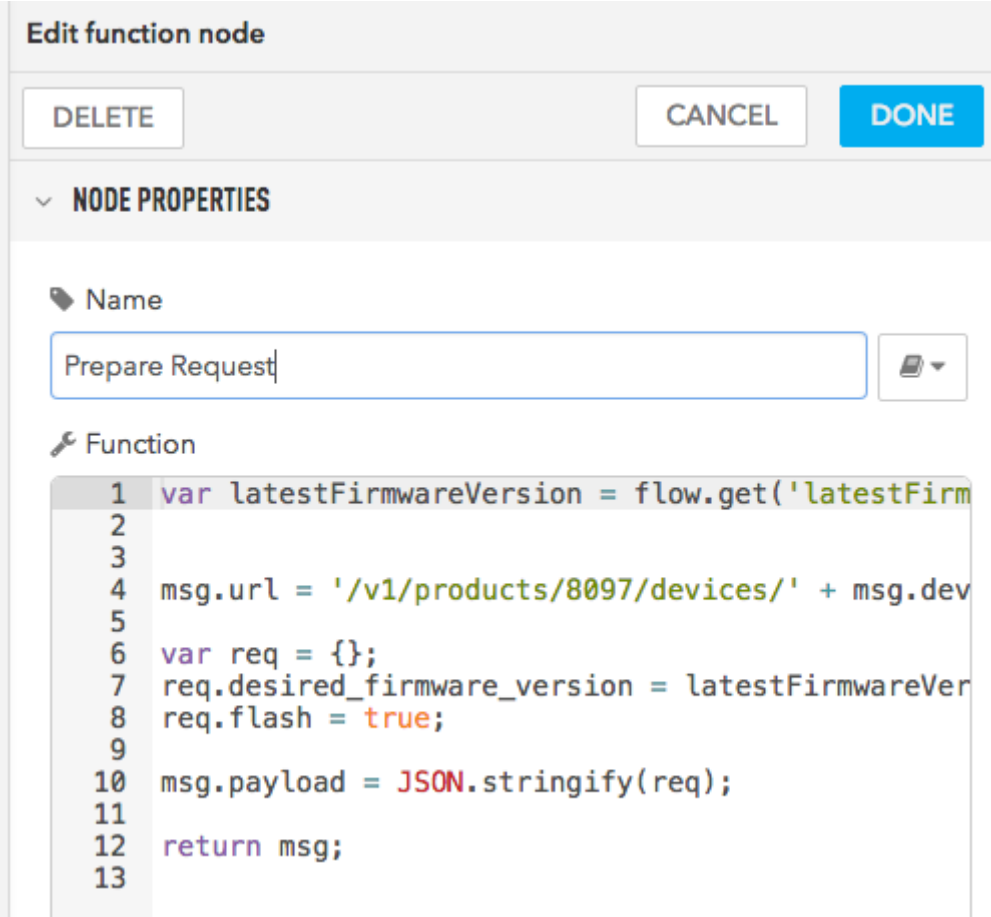
```
var latestFirmwareVersion = flow.get('latestFirmwareVersion');  
  
msg.url = '/v1/products/8097/devices/' + msg.device;  
  
var req = {};  
req.desired_firmware_version = latestFirmwareVersion;
```

```
req.flash = true;

msg.payload = JSON.stringify(req);

return msg;
```

This gets the latest firmware version and then creates a Particle API request to lock and flash that version.



Edit function node

DELETE CANCEL DONE

▼ **NODE PROPERTIES**

Name

Prepare Request

Function

```
1 var latestFirmwareVersion = flow.get('latestFirm
2
3
4 msg.url = '/v1/products/8097/devices/' + msg.dev
5
6 var req = {};
7 req.desired_firmware_version = latestFirmwareVer
8 req.flash = true;
9
10 msg.payload = JSON.stringify(req);
11
12 return msg;
13
```

- In the Particle section, drag a **particle api** node into the workspace.
- Double click to edit.
- Set your product authentication.
- Set **Method PUT**.

You can leave the rest blank as the URL and request are set by the previous node.

Edit particle api node

DELETE CANCEL DONE

▼ NODE PROPERTIES

Name

Auth

Method

URL

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"

// particle compile photon --target 0.8.0-rc.10 --saveTo firmware.bin

PRODUCT_ID(8097);
PRODUCT_VERSION(6);
SYSTEM_THREAD(ENABLED);

#include "neopixel.h"
#include "Debounce.h"

SerialLogHandler logHandler;

const int NEOPIXEL_PIN = D2;
const size_t NEOPIXEL_COUNT = 1;
const int NEOPIXEL_BRIGHTNESS = 15;
Adafruit_NeoPixel strip(NEOPIXEL_COUNT, NEOPIXEL_PIN, WS2812B);
Debounce button;
```

```
const int SWITCH_IN_PIN = D5;
const int SWITCH_LED_PIN = D3;

const unsigned long VERSION_CHECK_PERIOD_MS = 10 * 60 * 1000; // Every 10
minutes
unsigned long lastVersionCheck = 0;
unsigned long versionCheckPeriod = 10000; // Initially 10 seconds
bool updateAvailable = false;

extern uint16_t __system_product_version; // This is where PRODUCT_VERSION is
saved to

int updateAvailableHandler(String param);

void setup() {
    Serial.begin();

    Particle.function("updateAvailable", updateAvailableHandler);

    button.attach(SWITCH_IN_PIN, INPUT_PULLUP);

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

    strip.setBrightness(NEOPIXEL_BRIGHTNESS);
    strip.begin();
}

void loop() {
    if (button.update() && button.fell()) {
        Log.info("button pressed");
        if (updateAvailable && Particle.connected()) {
            updateAvailable = false;
            digitalWrite(SWITCH_LED_PIN, LOW);

            Particle.publish("updateFirmware", "", PRIVATE);
        }
    }

    if (millis() - lastVersionCheck >= versionCheckPeriod) {
        lastVersionCheck = millis();
    }
}
```

```
if (Particle.connected()) {
    // Check firmware version
    char buf[32];
    snprintf(buf, sizeof(buf), "%d", __system_product_version);
    Particle.publish("checkFirmware", buf, PRIVATE);
    Log.info("requesting checkFirmware %s", buf);

    versionCheckPeriod = VERSION_CHECK_PERIOD_MS;
}
else {
    // Not connected, check again in 10 seconds
    versionCheckPeriod = 10000;
}
}

int updateAvailableHandler(String param) {
    Log.info("updateAvailable %s", param.c_str());

    digitalWrite(SWITCH_LED_PIN, HIGH);
    updateAvailable = true;

    return 0;
}
```

The project.properties file adds these libraries:

```
dependencies.neopixel=0.0.14
dependencies.Debounce=0.0.1
```