Beyond IFTTT-Talking To Web Services

Dr. M.P. Rogers
44-599-IoT

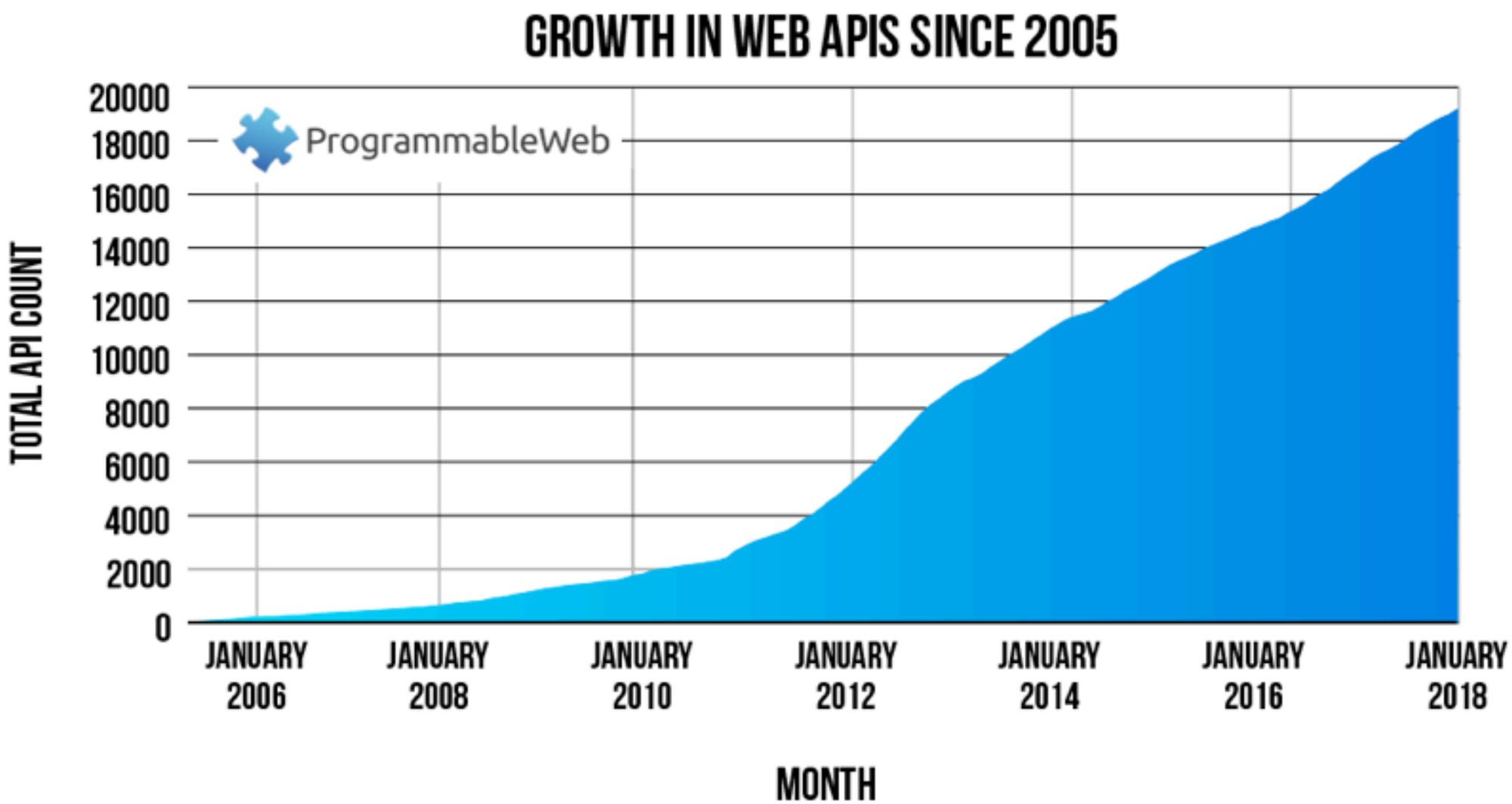
Objectives

- Students will be able to:
  - explain what a web service is
  - explain what a webhook is
  - construct webhooks to connect to utilize particular web services
  - create mustache templates
  - inspect web service requests with requestb.in

Web Services

- A **web service** is an application that provides functionality, over HTTP, that another application can utilize
- The web services that we will examine are RESTful, and work by sending HTTP messages GET, POST, PUT, DELETE, etc. to exchange information
- Examples:
  - openrates.io's web service provides current exchange rates that a mobile app might display. It uses GET messages to specify currencies and time periods
  - Uber's web service allows other apps to make ride requests, get estimated arrival times, etc.
  - WeatherUnlocked gives access to current and forecast weather

ICE: Useful Web Services

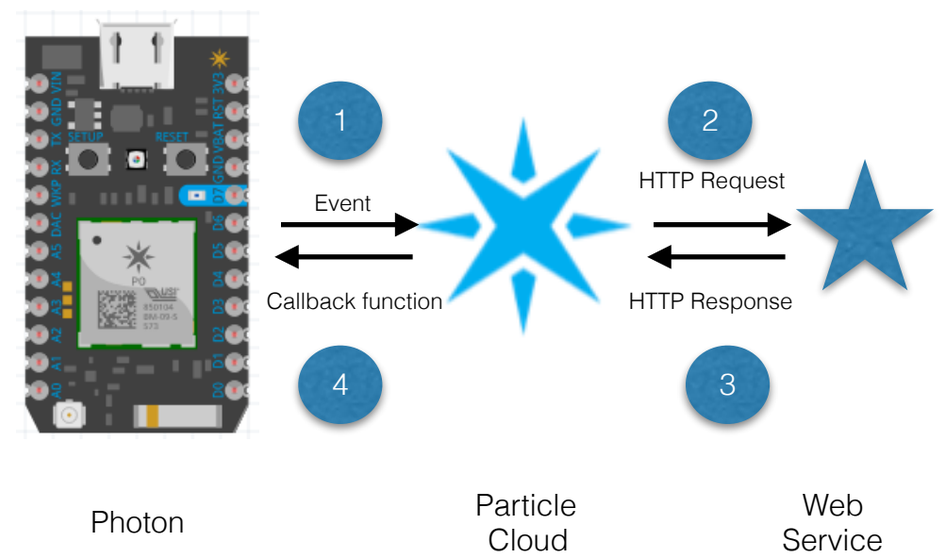


Source: https://www.programmableweb.com/news/research-shows-interest-providing-apis-still-high/research/2018/02/23

ICE: Find a useful/interesting web service at programmableweb.com

Webhooks

- A **webhook** is a mechanism for interacting with a web service
- The webhook, triggered when a Photon publishes an event (1), sends an HTTP request — GET, POST, PUT or DELETE — to a web service (2)
- Results from the web service (3) can also be returned to a Photon (4) that subscribes to an appropriate event
- Q: How does this relate to what IFTTT does?
- Q: Is this secure?
- Q: What are other examples of webhooks?

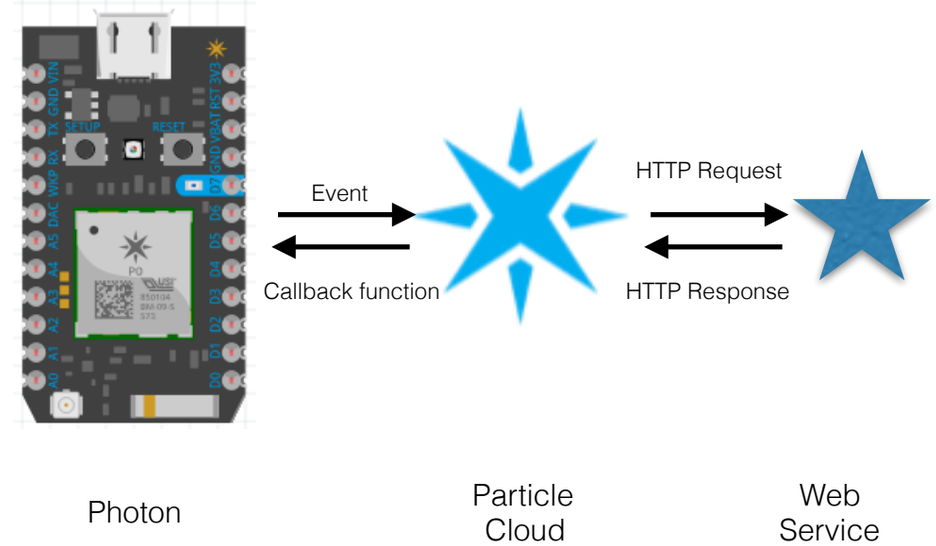


Webhooks

- Q: How does this relate to what IFTTT does?
- A: IFTTT is basically a user-friendly webhook manager
- Q: Is this secure?
- A: As secure as the web service we are using.
- Q: What are other examples of webhooks?
- A: GitHub, Microsoft Azure, Stripe

Webhooks

- Each webhook invocation produces 3 events in the log
  - **eventName**, your original event
  - **hook-sent/eventName**, when the request is sent
  - **hook-response/eventName/0**, when a response is received
- Webhooks can use POST, GET, PUT, etc., as dictated by the web service
- Since a webhook communicates with another web service, an API key for that service may be required
- Since a web service response shows up as an event, you can subscribe to it, then process the data from that response.



EVENT NAME	DATA	PUBLISHED AT	DEVICE
hook-response/tempertemper/0	10	*July 27th at 10:36:35 am*	particle-internal
hook-sent/tempertemper		*July 27th at 10:36:35 am*	particle-internal
tempertemper	25.96	*July 27th at 10:36:34 am*	trochee_wombat

An Example: ThingSpeak

- thingspeak.com can be used to store, display, and analyze sensor data
- Since thingspeak.com is run by the same people who made MATLAB, its visualization/analysis tools are first class
- Aside: Like IFTTT, ThingSpeak provides the ability to evaluate data and when certain criteria are met, trigger another web service. This is accomplished through the ThingSpeak apps. The ThingHTTP App is most general, it allows you to communicate with any web service

ThingSpeak as a Web Service

- To store data on thingspeak.com, a Write API Key is necessary. To access data stored in a private channel, a Read API key is required

- Sending a **GET** request to api.thingspeak.com/update, specifying two parameters: api_key and field1, will add the value of field1 to your channel.

- Sending a **POST** request, with the two parameters appearing in the body of the POST message, will do the same

API Requests

Update a Channel Feed

```
GET https://api.thingspeak.com/update?api_key=xxxxx&field1=0
```

Get a Channel Feed

```
GET https://api.thingspeak.com/channels/12345/feeds.json?api_key=xxxxx&results=2
```

Get a Channel Field

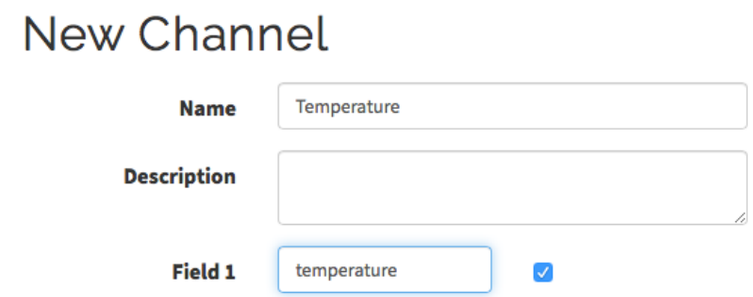
```
GET https://api.thingspeak.com/channels/12345/fields/1.json?api_key=xxxxx&results=2
```

Get Channel Status Updates

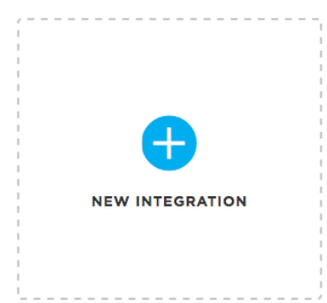
```
GET https://api.thingspeak.com/channels/12345/status.json?api_key=xxxxx
```

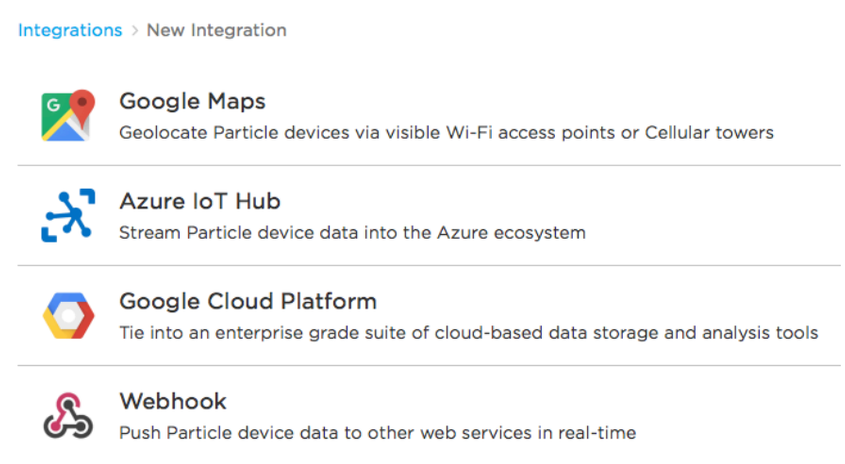
Creating a Webhook to ThingSpeak

1. In ThingSpeak, create a New Channel named **Temperature** with a field label called **temperature** — the latter the name that will appear in the visualization

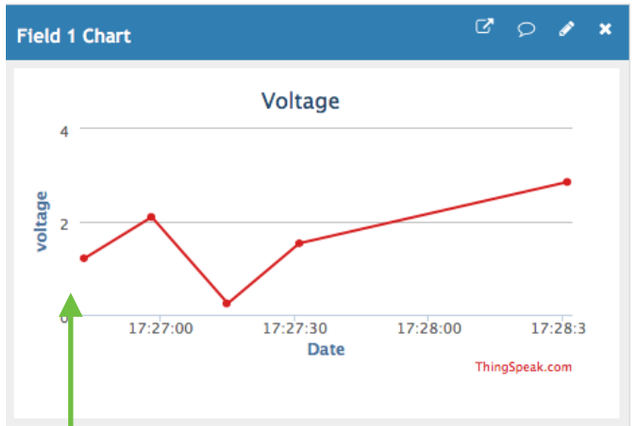


2. In particle.io, on the Console/ Integrations tab, click on New Integration and select Webhook





Creating a Webhook to ThingSpeak



1. In your newly created webhook, configure the following:

    1. event name: **tempertemper** — the name of the event that will trigger the webhook, the 1st arg in Particle.publish())

    2. URL: https://api.thingspeak.com/update — where we send a request to

    3. Request type: POST

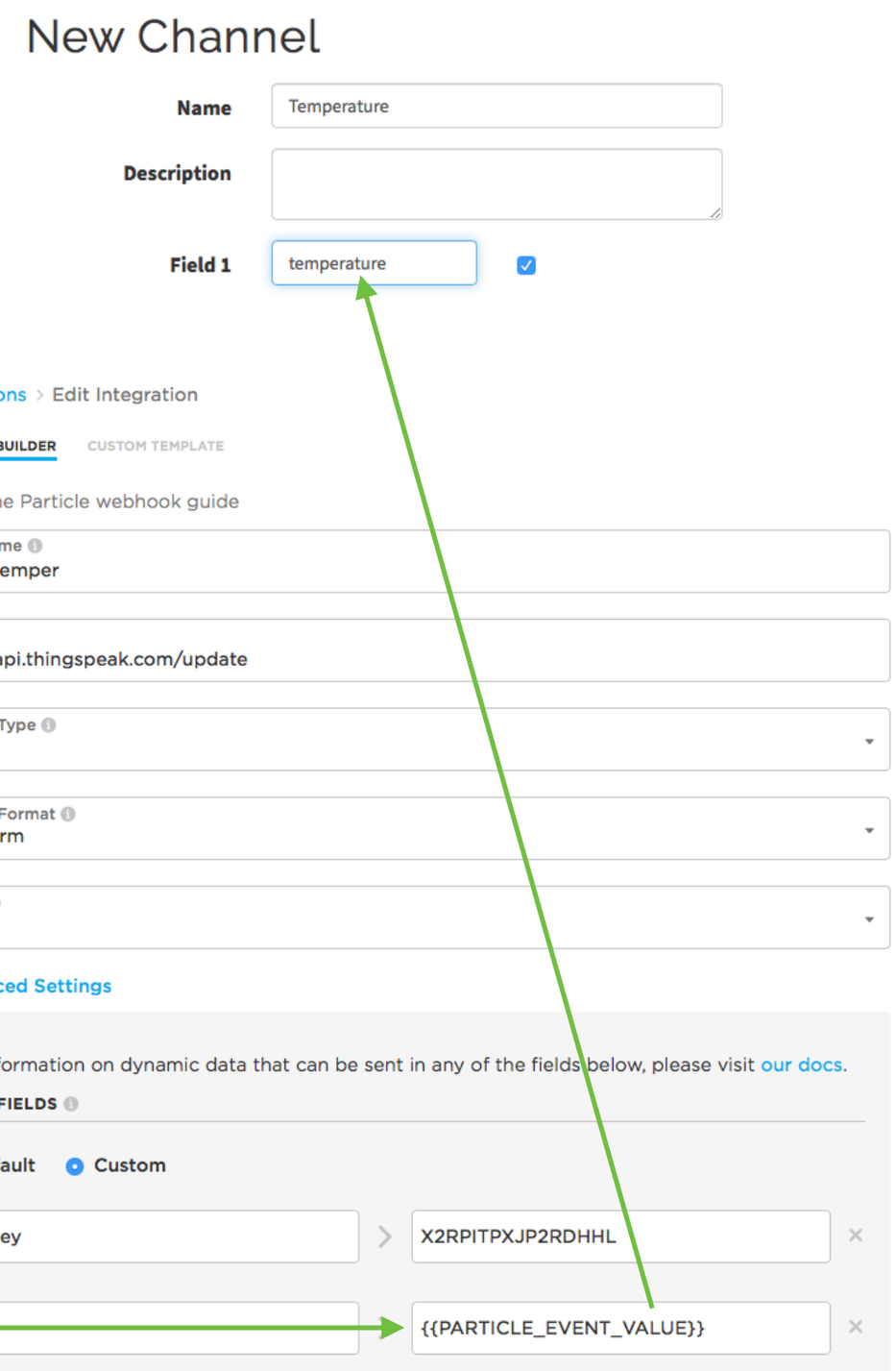
    4. Request Format: Web Form

    5. Device: Any — we could also specify a single device

2. Click on Advanced Settings, then on Custom. Here you'll provide data to appear in the message body, by defining:

    1. api_key: Your API KEY (from ThingSpeak),

    2. field1: {{PARTICLE_EVENT_VALUE}} — the 2nd arg in Particle.publish()




```
Particle.publish("tempertemper",String(temperature));
```

What Gets Sent

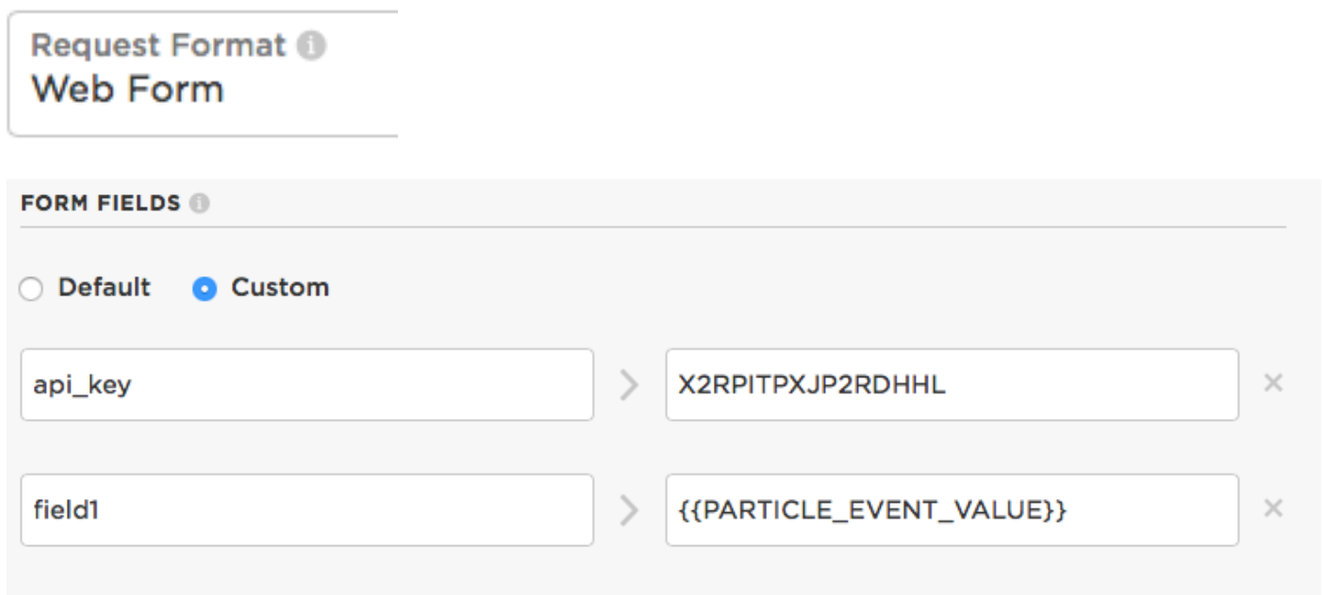
- A POST message gets sent to thingspeak.com
- The body of the post message depends on the request format.
- It can be either url-encoded (ampersand-delimited name=value pairs) or JSON
- ThingSpeak will accept both.

What Gets Sent: Web Form Custom

- Web Form Custom encodes the data as &-delimited field=value pairs, as shown.
- This is how data gets sent from an html form, hence the name: Web Form.

POST /update
User-Agent: ParticleBot/1.1 (https://docs.particle.io/webhooks)
Content-Type: application/x-www-form-urlencoded
Host: api.thingspeak.com
X-Request-Id: aebecc04-2240-4c5e-b7f6-51850ab678ef
Total-Route-Time: 0
Via: 1.1 vegur
Content-Length: 37
Connect-Time: 1
Connection: close

```
field1=26.55&api_key=YOUR_WRITE_API_KEY_RIGHT_HERE
```



What Gets Sent: Web Form Default

POST /update
User-Agent: ParticleBot/1.1 (https://docs.particle.io/webhooks)
Content-Type: application/x-www-form-urlencoded
Host: api.thingspeak.com
X-Request-Id: 61f671e7-8226-4fa3-969b-2628d543f02d
Total-Route-Time: 0
Via: 1.1 vegur
Content-Length: 141
Connect-Time: 0
Connection: close

```
event=tempertemper&data=26.63&published_at=2018-07-27T16%3A07%3A32
&coreid=ddddddddddddddd&field1=26.63&api_key=
YOUR_WRITE_API_KEY_RIGHT_HERE
```

Web Form Default includes 4 default fields

- event name,
- time,
- device ID
- data.

Request Format
Web Form

FORM FIELDS

Default Custom

event	{{{PARTICLE_EVENT_NAME}}}
data	{{{PARTICLE_EVENT_VALUE}}}
coreid	{{{PARTICLE_DEVICE_ID}}}
published_at	{{{PARTICLE_PUBLISHED_AT}}}
api_key	X2RPITPXJP2RDHHL
field1	{{PARTICLE_EVENT_VALUE}}

What Gets Sent: JSON Custom

POST /update
User-Agent: ParticleBot/1.1 (https://docs.particle.io/webhooks)
Content-Type: application/json
Connection: close
X-Request-Id: 85226991-7d30-4e0e-b8f5-c91700bfa9c1
Total-Route-Time: 0
Via: 1.1 vegur
Content-Length: 47
Accept: application/json
Connect-Time: 0
Host: api.thingspeak.com

```
{"field1":"26.72","api_key":"YOUR_WRITE_API_KEY_RIGHT_HERE"}
```

- JSON custom sends the data as a JSON object.
- Some web services require JSON, others will also accept web forms (url encoding): their API will tell you (if not, don't use it!)

Request Format
JSON

JSON DATA

Default Custom

```
1 {"api_key":"X2RPITPXJP2RDHHL",
2  "field1":"{{PARTICLE_EVENT_VALUE}}"
3 }
```

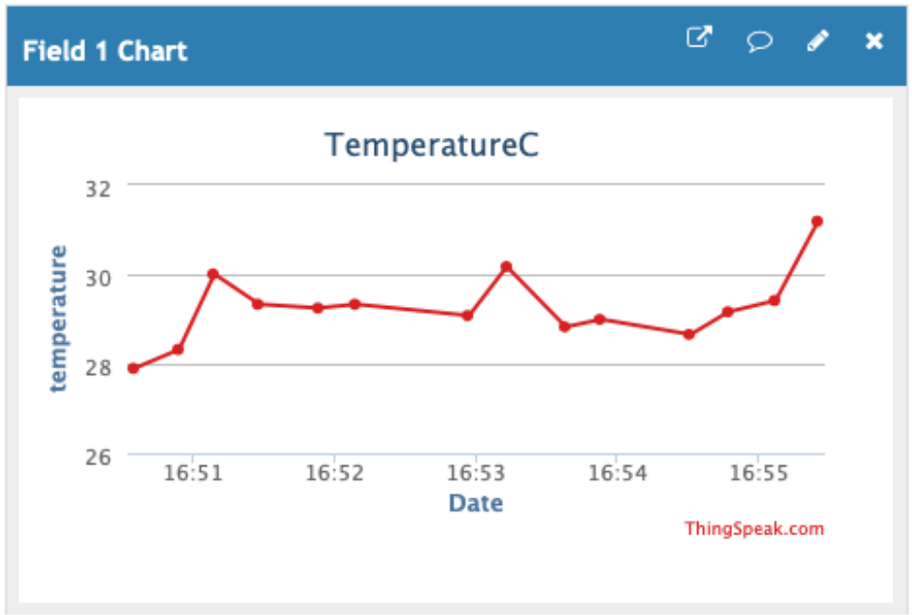
What Gets Returned: Web Form

Events

Search for events

ADVANCED

NAME	DATA	DEVICE ID	PUBLISHED AT
hook-response/pushed/0	11	particle-internal	10/3/18 at 4:48:07 pm
hook-sent/pushed		particle-internal	10/3/18 at 4:48:06 pm
pushed	28.320549	240038000347353138383138	10/3/18 at 4:48:06 pm
hook-response/pushed/0	10	particle-internal	10/3/18 at 4:47:43 pm
hook-sent/pushed		particle-internal	10/3/18 at 4:47:42 pm
pushed	28.067429	240038000347353138383138	10/3/18 at 4:47:42 pm



As noted, each web service interaction involves 3 events:

- the initial event that triggered the webhook
- "hook-sent", corresponding to the service request
- "hook-response" corresponding to the service response

Note that

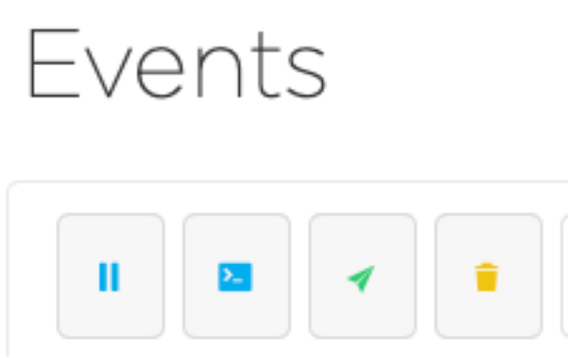
- Data is empty in the hook-sent event: that is normal.
- thingspeak returns the number of data points in the current channel

What Gets Returned: JSON

```
event: pushed
data:
{"data":"28.658044","ttl":60,"published_at":"2018-10-03T21:56:44.111Z","coreid":
"240038000347353138383138"}

event: hook-sent/pushed
data:
{"data":"","ttl":60,"published_at":"2018-10-03T21:56:44.238Z","coreid":"particle
-internal"}

event: hook-response/pushed/0
data:
{"data":"15","ttl":60,"published_at":"2018-10-03T21:56:44.305Z","coreid":"partic
le-internal"}
```



- Clicking on the 2nd button to see events in the console
- Since we posted the data as JSON, ThingSpeak decided to send us back a JSON object, containing a fuller description of the channel, not just the number of data points

Particle Console Fun Facts

- The console includes
  - Devices
  - Products
  - SIM Cards
  - Events
  - Integrations
  - Authentication
  - Billing
  - Build (a link to build.particle.io)
- To see webhook activity, look in **Events**.
- If you look in Devices >> Your Device, you won't see the hook-sent and hook-response events, even if you spend **hours** looking for them, because these are associated with a device called "particle internal", not your actual device

ThingSpeak Data Rate

- You get what you pay for: with a free account, ThingSpeak will accept at most 1 message every 15 seconds

Webhook Variables

- When your event triggers a webhook, 4 variables are at your disposal for injection into the JSON or form data being sent to the web service:

- {{PARTICLE_DEVICE_ID}} - the originating Photon's id

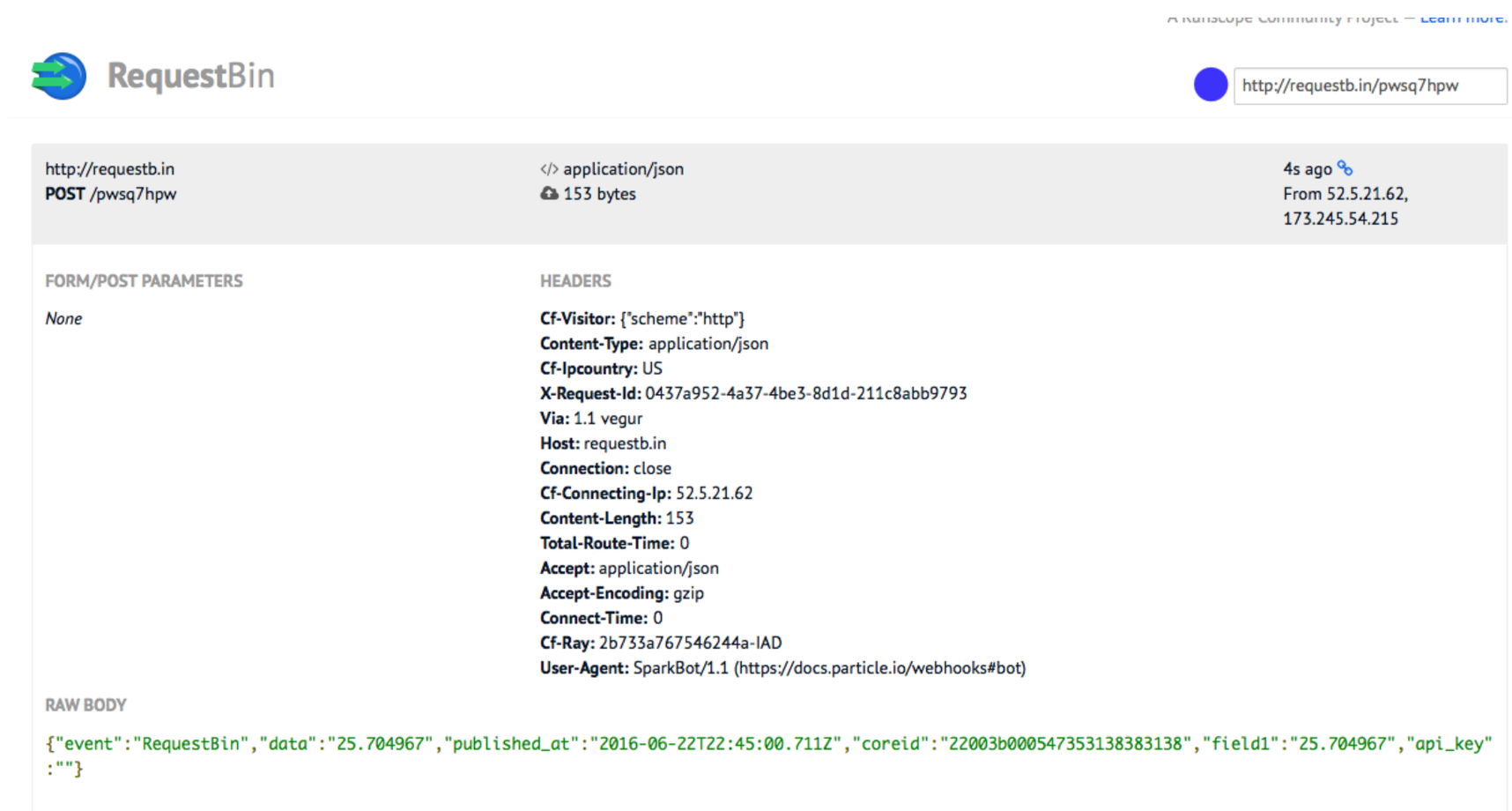
- {{PARTICLE_EVENT_NAME}} - the name of the triggering event

- {{PARTICLE_EVENT_VALUE}} - the data being published

- {{PARTICLE_PUBLISHED_AT}} - time stamp, when the event occurred

Debugging Webhooks

- Using RequestBin (requestb.in), it is possible to see what particle.io is sending out. Merely change the URL to that provided by requestb.in (keeping the rest of the fields constant)



Debugging Webhooks

- Although requestb.in has been shutdown, an alternative is available here:
  - https://requestbin-mpr.herokuapp.com

Completing the Circle: Getting Info from a Web Service

- ThingSpeak talked back to us: if we sent it a url-encoded form, it returned the number of data points; if we sent it JSON, it sent back a JSON description of the channel

- This is not ThingSpeak-specific: a POST message always generates a response (IETF RFC 7230), and much of the time the entire *point* of a web service is to get information from it

- So ... how do we do that?

Completing the Circle: Getting Info from a Web Service

- **Particle.subscribe(*eventName*, *handler,* MY_DEVICES)** lets us specify a callback function — *handler* — that will be invoked in response to the triggered webhook
  - *eventName:* hook-response/*triggering-event-name*/*index*
  - handler*:* void *handler*(const char * event, const char * data)
  - MY_DEVICES is required ... otherwise it doesn't work. You're welcome 😎
- Recall that Particle.subscribe() does prefix matching. Hence, if the original event was tempertemper, you could subscribe to hook-response/tempertemper (or just hook-response/tem, if ambiguity was not an issue), rather than the full hook-response/tempertemper/0

OpenWeatherMap

- As an example, OpenWeatherMap can be used to fetch weather. Its API uses a query string (use GET) and the response is in JSON

Maryville's zip code is 64468

http://api.openweathermap.org/data/2.5/weather?zip=64468,us&appid=12bb498c3677cdfe5285a277361e412f


```
{"coord":{"lon":-94.87,"lat":40.34},"weather":
[{"id":500,"main":"Rain","description":"light
rain","icon":"10d"}],"base":"stations","main":
{"temp":298.66,"pressure":1012,"humidity":44,"temp_min":298.15,"temp_max":2
99.15},"visibility":16093,"wind":
{"speed":7.7,"deg":160,"gust":9.8},"clouds":
{"all":1},"dt":1506894780,"sys":
{"type":1,"id":1664,"message":0.006,"country":"US","sunrise":1506860226,"su
nset":1506902380},"id":0,"name":"Maryville","cod":200}
```

Parsing Responses

- The entire response can be delivered to our callback function, and we could use C++ to parse it. An alternative is to provide a Response Template, using the mustache templating system — in this case particle.io parses the String for you and extracts the relevant portions.

- For instance, to obtain the temperature, we would put {{main.temp}} in our response template


```
{
  "coord":{
    "lon":-94.87,
    "lat":40.34
  },
  "weather":[
    {
      "id":701,
      "main":"Mist",
      "description":"mist",
      "icon":"50d"
    }
  ],
  "base":"stations",
  "main":{
    "temp":291.88,
    "pressure":1013,
    "humidity":88,
    "temp_min":291.15,
    "temp_max":292.15
  },
  "visibility":16093,
  "wind":{
    "speed":6.2,
    "deg":290,
    "gust":9.3
  },
  "clouds":{
    "all":1
  },
  "dt":1502976900,
  "sys":{
    "type":1,
    "id":857,
    "message":0.0051,
    "country":"US",
    "sunrise":1502969596,
    "sunset":1503018761
  },
  "id":0,
  "name":"Maryville",
  "cod":200
}
```

openweathermap webhook

Event Name *The Particle event name that triggers the webhook*	`fetchTemperature`
Full URL *The target endpoint that is hit when the webhook is triggered*	http://api.openweathermap.org/data/2.5/weather
Request Type *The standard web request method used when the webhook is triggered*	GET
Device *The device that will trigger the webhook*	*any device*
Query Params *Parameters to append the URL string when hitting the webhook endpoint*	`{ "zip": "64468,us", "APPID": "xxxxx" }`
Include Default Attributes *Whether your webhook will include the data included in your Particle.publish()*	Yes
Enforce SSL *Whether your webhook will will validate the certificate against its certificate authority chain*	No
Response Template *A custom template to be returned back to your device when the webhook returns*	`{{main.temp}}`

The Webhook in Action

```
void handleTemp(String event, String data){
    Serial.println("Event: " + event);
    Serial.println("Data: " + data);
}

void setup(){
    pinMode(D0, INPUT_PULLUP);
    Particle.subscribe("hook-response/fetchTemperature", handleTemp, MY_DEVICES);
    Serial.begin(9600);
}

void loop(){

    delay(500);                          // another way to stop multiple invocations
    if(digitalRead(D0) == LOW) {
       Particle.publish("fetchTemperature");
       Serial.println("Event published");
   }
}
```

```
Event published
Event: hook-response/fetchTemperature/0
Data: 293.65
```

EVENT NAME	DATA	PUBLISHED AT	DEVICE
hook-response/fetchTemperature/0	293.65	*August 17th at 10:04:29 am*	particle-internal
hook-sent/fetchTemperature		*August 17th at 10:04:29 am*	particle-internal
fetchTemperature	null	*August 17th at 10:04:29 am*	adorable-jetpack

More on Mustache

- Mustache takes a JSON object and extracts bits and pieces of it. It works by using tags -- names in {{ }} -- that reference keys in JSON
- Text not in {{ }} is passed verbatim into the handler
- These examples show the response template text and the output from handleTemp()
  - {{main.temp}} -- {main.humidity}} // 291.88 -- 88
  - {{weather.0.main}} // mist
  - {{sunrise}} // error
  - {{sys.sunrise}} // 1502969600 (# of seconds since Jan 1, 1970)

weather.0 ==> 0th array element


```
{
  "coord":{
    "lon":-94.87,
    "lat":40.34
  },
  "weather":[
    {
      "id":701,
      "main":"Mist",
      "description":"mist",
      "icon":"50d"
    }
  ],
  "base":"stations",
  "main":{
    "temp":291.88,
    "pressure":1013,
    "humidity":88,
    "temp_min":291.15,
    "temp_max":292.15
  },
  "visibility":16093,
  "wind":{
    "speed":6.2,
    "deg":290,
    "gust":9.3
  },
  "clouds":{
    "all":1
  },
  "dt":1502976900,
  "sys":{
    "type":1,
    "id":857,
    "message":0.0051,
    "country":"US",
    "sunrise":1502969596,
    "sunset":1503018761
  },
  "id":0,
  "name":"Maryville",
  "cod":200
}
```

Exercises

1. Capture and display the data coming back from thingspeak.com
2. Capture conversion rates between two currencies - exchangerate-api.com
    - fetch and print the USD/AED currency rate using bulk ;
    - fetch and print the USD/INR currency conversion rate using pair
3. Determine the current moon phase - http://aa.usno.navy.mil/data/docs/api.php
4. Set up webhooks with some other interesting web services
    1. twilio.com
    2. pushbullet.com
    3. ?

Examples

- exchangerate-api.com
  - https://v3.exchangerate-api.com/bulk/*apikey*/USD
  - https://v3.exchangerate-api.com/pair/*apikey*/USD/INR
- http://aa.usno.navy.mil/data/docs/api.php

Resources

- http://www.service-architecture.com/articles/web-services/index.html [web services]
- http://community.thingspeak.com/2015/10/official-thingspeak-library-for-arduino-and-particle/
- https://github.com/rickkas7/particle-webhooks
- requestb.in
- https://jsonformatter.curiousconcept.com
- http://mustache.github.io