Quick Lesson -

Dynamic Web Pages for Photon

“A javascript template for Particle variables and functions”

**Table of Contents**

[Introduction](#_q78nx5y34fjb)

[Creating a Javascript Web Page](#_2rob8fxt05v0)

# Introduction

The original specification HTML 3.0 was a static markup language. It could describe how a document was to look, but couldn’t react to actions by the user. Very early on, this was recognized as a drawback and many solutions were created to allow web pages to respond to user input (early examples included Java, Flash, Javascript, etc.). These solutions allow the browser to download a piece of code within the HTML web page and run it.

This quick lesson will walk you through the basic template of a Javascript web page that responds to button clicks. This Quick Lesson is not intended to get you started programming in Javascript, but just to provide a basic editable template for you to get a functional website up and running.

# Creating a Javascript Web Page

Creating a webpage that interfaces with your microcontroller allows you to control the microcontroller and read the values of program variables from anywhere you can access the web. Let’s learn how to write a webpage that can read the value of a cloud variable and call a cloud function.

The basic language of the web is called **hypertext markup language (HTML)**.HTML is made up of **tags** (text inside pointy-brackets) and **content**. Tags typically structure the document and describe how to display the content. For example, the content could be text and a tag could specify that the text should be displayed in **bold**.

Every HTML document starts with <HTML> and ends with </HTML>.In general, for every tag <TAG>, there is a matching closing tag </TAG>. There are two basic fields inside an HTML document. The <HEAD> (header), which is *optional*, defines things like the title bar (<TITLE>) and metadata for search engines. The <BODY> contains the contents of the document. Many HTML elements exist to make webpage contents look nice, but for this tutorial, we will ignore most of these and stick to the basics. You can find links at the bottom of the document for further reading on this topic if you want to improve the appearance of your web pages.

HTML was originally intended to describe the contents of a webpage and provide links to other web pages. HTML in its natural state doesn’t deal well with information that changes (also referred to as dynamic web pages). One way to improve a web page's ability to process data and change its output is by using a **scripting language** called Javascript.Anything inside <script> and </script> tags is processed by the web browser as a script rather than an HTML document. A Javascript is a programming language, like C++, and a script is a type of program, like the ones you write in C++.

The description of our webpage will start with the following code.

|  |
| --- |
| **<!DOCTYPE HTML>**  **<html>**  **<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js" type="text/javascript" charset="utf-8"></script>**  **<body>** |

The <script src=”http://…”> above simply loads a library of functions for use in this webpage (like a #include statement). You can simply copy and paste this html code at the top of the file for your webpage.

Let’s learn how to display the value of Javascript variables in our webpage. The **<SPAN>** tag creates a blank field in a webpage that can be populated with text and updated dynamically. The text around the **<SPAN>** tags describes the text surrounding the field. For example, the first statement below will display “Current Temperature \_\_\_\_\_℉”, where the blank can be filled in by updating the value of the current\_temp variable.

|  |
| --- |
| **Current Temperature:<span id="current\_temp"></span>&deg; F<br>**  **Desired Temperature:<span id="desired\_temp"></span>&deg; F<br>** |

The **<br>** tag at the end of each line creates a line break. This behaves just like the new line character (‘\n’).The spans will eventually be accessed in Javascript by their IDs (current\_temp and desired\_temp).

The next item we will place in our webpage is a button. When the button is clicked, it will run a function called **start()**.The text on the button will read **Refresh Data**.

|  |
| --- |
| **<button id="connectbutton" onclick="start()">Refresh Data</button>** |

Now, let’s see how to write data into our spans. We do this by using the **jquery** Javascript library (initialized above), **document** class, and the **innerHTML** function.Here’s a simplified example of how innerHTML works:

|  |
| --- |
| **<script type="text/javascript">**  **document.getElementById("current\_temp").innerHTML = "1234";**  **document.getElementById("desired\_temp").innerHTML = "5678";**  **</script>** |

When embedded in the HTML above, this Javascript would produce the following webpage.

|  |
| --- |
|  |

However, we don’t want the temp to be a static number (like 1234 or 5678), but a dynamic number retrieved from our Photon. To do this, we need to replace the <script> above with one that does more work.

Now let’s write the start() function that gets called when we push the button. Inside this function, we will issue **get requests** to get the values of our cloud variables from the microcontroller. Remember that each cloud variable has a unique URL, and we can get the value of the variable by accessing the URL. To do this, current\_temp and desired\_temp must be declared as cloud variables in the microcontroller code. After issuing get requests for the variables, we will take the values returned from the cloud and display them in our spans. As discussed in **Cloud Variables**, you can get the value of a cloud variable by accessing the URL where the variable is stored in the cloud. Our get requests will simply create the appropriate URL and use the URL to get the variable’s value. Don’t forget, every microcontroller has its own **device ID** and **access token** that must be used in the URL. The code below appends several text fields together to create a URL called **requestURL** that contains the device ID, the access token, and the requested cloud variable name. Then, the URL is queried using the **getJSON** function. When the variable value is returned, it is placed into the appropriate span by the **innerHTML**.

|  |
| --- |
| **<script type="text/javascript">**  **function start(objButton) {**  **var deviceID = "YOUR\_DEVICE\_ID\_GOES\_HERE";**  **var accessToken = "YOUR\_ACCESS\_TOKEN\_GOES\_HERE";**  **var baseURL = "https://api.particle.io/v1/devices/"**  **var varName = "current\_temp";**  **requestURL = baseURL + deviceID + "/" + varName + "/?access\_token=" + accessToken;**  **$.getJSON(requestURL, function(json) {**  **document.getElementById("current\_temp").innerHTML = + json.result;**  **});**  **var varName = "desired\_temp";**  **requestURL = baseURL + deviceID + "/" + varName + "/?access\_token=" + accessToken;**  **$.getJSON(requestURL, function(json) {**  **document.getElementById("desired\_temp").innerHTML = json.result;**  **});**  **}**  **</script>** |

Notice the **</script>** tag at the end of the HTML code above. This ends the javascript script.

Now that we know how to get the values of cloud variables and put them in our webpage, let’s see how to set the value of a variable on the Photon by calling a cloud function. We will do this using a simple HTML form that makes a **post request**. Like the get request, the post request uses a URL. This URL calls a cloud function called **set\_temp** and passes the form’s input value (**args**) as the string argument of the function. Initially, the form’s input field will show the text **(50-90)**, reminding the user that the input temperature for the set\_temp function should be between 50-90 degrees. The text on the button will read **Set Temperature**. Since our webpage is now done, we can use the **</body>** tag to end the body of the webpage and the **</html>** tag to end the webpage.

|  |
| --- |
| **<form action="**[**https://api.particle.io/v1/devices/**](https://api.particle.io/v1/devices/YOUR)**YOUR\_DEVICE\_ID\_GOES\_HERE/set\_temp?access\_token=YOUR\_ACCESS\_TOKEN\_GOES\_HERE" method="POST">**  **<input type="text" name="args" value="(50-90)"><br>**  **<input type="submit" value="Set Temperature">**  **</form>**  **</body>**  **</html>** |

**HINT:** You can view Javascript errors on the “Console” in Google Chrome by clicking**** → More Tools → Developer Tools, then click on the >> and selecting Console. See the picture below:

|  |
| --- |
|  |

Our webpage should end up looking something like this:

|  |
| --- |
|  |

And the code looks like this:

|  |
| --- |
|  |

While you may not understand all the details of HTML and Javascript, the good news is that it is relatively simple to copy the existing HTML code, like the examples above, and modify it to suit your needs. Instead of writing the code from scratch, copy and paste the existing code and modify it to work for the particular cloud variables and cloud functions in your program.

You can read more about dynamic webpages that update automatically and handle post request without refresh in [Quick Lesson - Particle Cloud Webpage without Refresh](https://drive.google.com/open?id=1oWcVHBH-aO5xOA9czefNScoRL6T4nCE-Bfi-zBKRQJ4)