Chapter 8 – Websites and Services

Contents

[Installing and Running Node.js Webserver 2](#_Toc456520444)

[Installing Node.js 2](#_Toc456520445)

[Running Node.js for the first time 2](#_Toc456520446)

# 1. Node.js Webserver

## 1.1 Installing Node.js

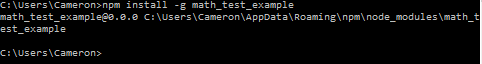
1. Navigate to <https://nodejs.org/en/> using the browser of your choice.
2. Click “Download” on that web page that displays in your browser.
3. Download nodejs.
4. Run the installer that you downloaded.
5. Install it to any location on your PC.
6. Congratulations you have installed Node.js

## 1.2 Running Node.js for the first time

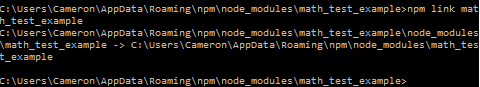
1. Open command prompt
2. Type npm adduser
3. You will be prompted to enter a username. Input any username you desire and press enter.
4. You will be prompted to enter a password. Input any password you desire and press enter.
5. You will be prompted to enter an email. Input your email address and press enter.

## 1.3 Install Node.js math package example

1. Open command prompt
2. Type npm install -g math\_test\_example and press enter.
3. You should see something similar to the following for a successful install.



1. Open command prompt and navigate to the node\_modules directory shown in command prompt. In the example above, the node\_modules directory is “C:\Users\Cameron\AppData\Roaming\npm\node\_modules\math\_test\_example”.
2. Type npm link math\_test\_example and press enter. You should see something similar to the following for a successful run of this command.



## 1.4 Test that the package is installed properly

1. Open command prompt and navigate to the directory you used in step 1.3.4. Mine is “C:\Users\Cameron\AppData\Roaming\npm\node\_modules\math\_test\_example”.
2. Create a file called main.js.
3. The contents of main.js are as follows:

var math\_example = require('math\_test\_example');

var result = 0;

console.log();

result = math\_example.addition(5,10);

console.log('addition(5,10) = ' + result);

console.log();

result = math\_example.subtraction(50,10);

console.log('subtraction(50,10) = ' + result);

console.log();

result = math\_example.multiplication(3,7);

console.log('multiplication(3,7) = ' + result);

console.log();

result = math\_example.division(27,3);

console.log('division(27,3) = ' + result);

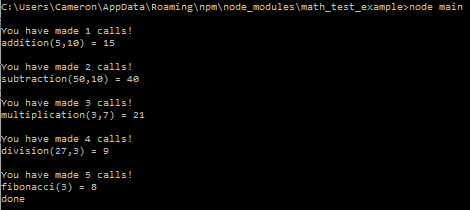
console.log();

result = math\_example.fibonacci(3);

console.log('fibonacci(3) = ' + result);

console.log('done');

1. Type the command node main and press enter
2. The output should be as follows:



1. Your test is successful!

# Web Services

Welcome to the fabulous world of web services! It’s a pretty fantastic place to be! It makes web sites much cooler and more fun to use! You might be asking yourself, I’ve never heard of a web service, that’s why I’m reading this section, so what makes them so cool and fun? Well I’ll tell you!

## When to use a web service?

* One problem with websites is that performing a post causes a response that repaints the page with the HTML result. This becomes a problem when the HTML response looks like the existing page because you waste bandwidth and resources to repaint the page.
* Web services fix this problem by providing the ability to send data to the server and receive data back from the server without requiring a repaint of the browser screen.

## Classes of Web Service

There are two major classes of web services listed below and described in the following sections.

* Representational State Transfer (REST), and;
* Arbitrary Web Services.

### Representational State Transfer (REST)

A description of REST is as follows:

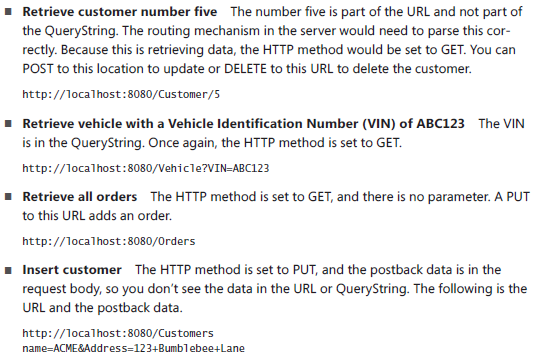
* The purpose of REST is to manipulate web resources by using a uniform set of **stateless** operations.
* Clean URLs are tightly associated with the REST concept.
* URLs use nouns ONLY.

#### REST Operations

REST maps create, retrieve, update and delete (CRUD) operations to HTTP methods. The mappings are as follows:

|  |  |
| --- | --- |
| **HTTP Method** | **CRUD Operation** |
| GET | * Retrieve data, or; * Performs an operation that does not change server-side data. |
| POST | * Updates data on the server, or; * Retrieves data when complex parameter values need to be sent to the server (e.g. a form) |
| PUT | * Inserts new server-side data |
| DELETE | * Deletes server-side data |

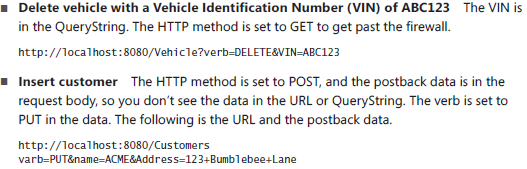
#### REST Examples



#### Problems with REST

* Some browsers only support GET and POST methods.
* Many firewalls only allow GET and POST methods to pass through them.

You can **fix** these problems by passing actions (verbs) in the query string. An example is shown below:



#### Advantages of REST

* Easy to connect to!
* Lightweight and thin client!
* Send and receive data in simple ways!

### Arbitrary Web Services

Exposes an arbitrary set of operations! A description of an arbitrary web service is as follows:

* Don’t attempt to map aspects of the protocol to operations
* Focused on the ability to offer more functionality such as message routing and various means of security
* Have an interface that enables the client to read and parse information
* Require the client to assemble a request, or message by using Simple Object Access Protocol (SOAP) message
* Not required to utilize HTTP protocol

## Calling a Web Service

You can call a Web Service using AJAX! AJAX utilizes a JavaSCript object called XMLHttpRequest, which is used to send and receive data.

### JavaScript synchronous example

// creates a new object

var xmlhttp=new XMLHttpRequest();

// This sets up the request by generating a relative URL with the QueryString x=5&y=10. False indicates that the operation will be performed synchronously.

xmlhttp.open("GET","/addition?x=5&y=10",false);

// communicates with the server by sending the request

xmlhttp.send();

// gets the returned response and parses it into a JSON object

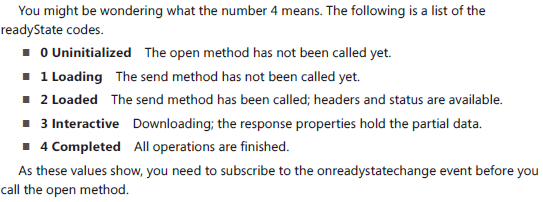
var jsonObject = JSON.parse(xmlhttp.response);

### JavaScript asynchronous example

// creates a new object

var xmlhttp = new XMLHttpRequest();

// this function handles the asynchronous call. It will be called five times based on the state of the asynchronous operation. Each of these are described as follows:



**xmlhttp.onreadystatechange = function () {**

**if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {**

**var jsonObject = JSON.parse(xmlhttp.response);**

**result.innerHTML = jsonObject.result;**

**}**

**}**

// This sets up the request by generating a relative URL with the QueryString x=5&y=10. True indicates that the operation will be performed asynchronously.

xmlhttp.open("GET", "/addition?x=" + x + "&y=" + y , **true**);

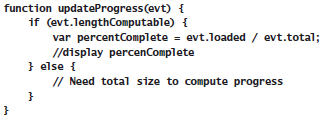
// communicates with the server by sending the request

xmlhttp.send();

### Progress of a Web service call

You can see the progress of a web service call by subscribing to the progress event. The code below can be added to any asynchronous call to subscribe to the “progress” event.

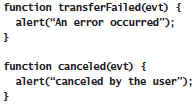




### Handle web service call errors

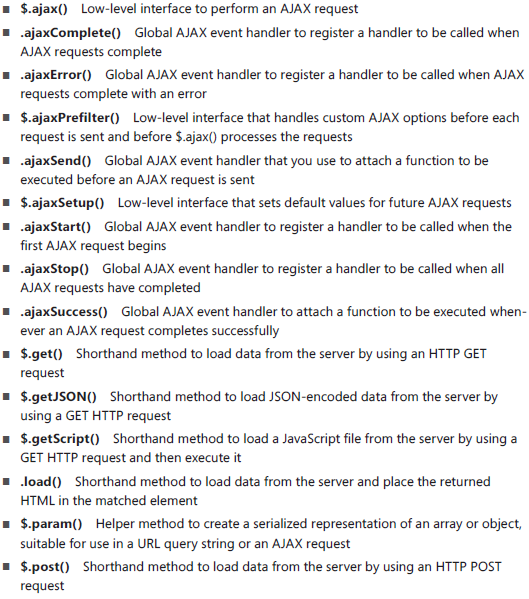
You can handle web service call errors by subscribing to the following “error” and “abort” events. The code below shows how you can subscribe to those events.

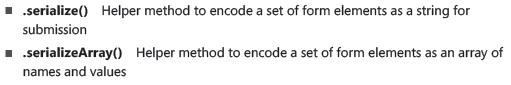




### JQuery XMLHttpRequest Wrappers

Use jQuery for your web service calls. It’s much easier than using the XMLHttpRequest object. The following wrappers are available for your use.

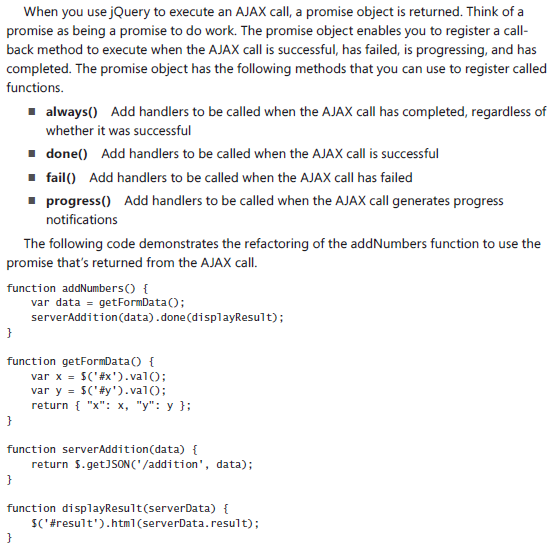




### Special Mention: Promises!

jQuery objects return promises and promises should be used when making any web service call. Chapter 9 describes promises in detail, so please wait until then to understand their usage.

If you must learn about them right now. Here is a small primer on their usage with web service calls.



## Cross-origin resource sharing

Cross-origin resource sharing occurs when you have a web service call to a URL that is different from the domain of the page requesting it.

For example,

If example.com makes a web service call to google.ca, then this is an example of cross-origin resource sharing.

The above example of a web service call would fail because it could result in a cross-site scripting attack. Basically the server “google.ca” won’t respond to example.com’s web service call because the request from example.com could be malicious.

The way to fix this is to implement cross-origin resource sharing on the “google.ca” web server. This can be done by adding “Access-Control-Allow-Origin: <http://google.ca> <http://example.com>” to the header of all web service requests received.

# Putting it all together

## RESTful Math Web Service

The example below uses Node.js as the server.

**App.js**

var express = require('express');

var app = express(); var formidable = require('formidable');

var math = require('math\_example');

app.use(express.static(\_\_dirname + '/public'));

app.get('/addition', function (request, response) {

var x = Number(request.query.x),

y = Number(request.query.y),

result = math.addition(x, y);

response.writeHead(200, { 'Content-Type': 'application/json' });

response.end('{ "result": ' + result + '}');

console.log('Handled addition request for x=' + x + ' : y=' + y);

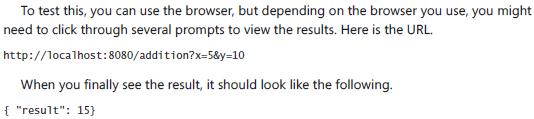
});

var port = 8080;

app.listen(port);

console.log('Listening on port: ' + port);

**Output:**



## Call the Math Web Service with jQuery

