

Blue Planet

Introduction to REST APIs and Blue Planet Orchestrator APIs

Student Demo Guide



LEGAL NOTICES

THIS DOCUMENT CONTAINS CONFIDENTIAL AND TRADE SECRET INFORMATION OF CIENA CORPORATION AND ITS RECEIPT OR POSSESSION DOES NOT CONVEY ANY RIGHTS TO REPRODUCE OR DISCLOSE ITS CONTENTS, OR TO MANUFACTURE, USE, OR SELL ANYTHING THAT IT MAY DESCRIBE. REPRODUCTION, DISCLOSURE, OR USE IN WHOLE OR IN PART WITHOUT THE SPECIFIC WRITTEN AUTHORIZATION OF CIENA CORPORATION IS STRICTLY FORBIDDEN.

EVERY EFFORT HAS BEEN MADE TO ENSURE THAT THE INFORMATION IN THIS DOCUMENT IS COMPLETE AND ACCURATE AT THE TIME OF PUBLISHING; HOWEVER, THE INFORMATION CONTAINED IN THIS DOCUMENT IS SUBJECT TO CHANGE.

While the information in this document is believed to be accurate and reliable, except as otherwise expressly agreed to in writing CIENA PROVIDES THIS DOCUMENT "AS IS" WITHOUT WARRANTY OR CONDITION OF ANY KIND, EITHER EXPRESS OR IMPLIED. The information and/or products described in this document are subject to change without notice. For the most up-to-date technical publications, visit www.ciena.com.

Copyright © 2015-2018 Ciena® Corporation – All Rights Reserved

The material contained in this document is also protected by copyright laws of the United States of America and other countries. It may not be reproduced or distributed in any form by any means, altered in any fashion, or stored in a database or retrieval system, without the express written permission of Ciena Corporation.

Ciena®, the Ciena logo, Z22™, Z33®, Z77®, Blue Planet® and other trademarks and service marks of Ciena appearing in this publication are the property of Ciena. Trade names, trademarks, and service marks of other companies appearing in this publication are the property of the respective holders.

Security

Ciena cannot be responsible for unauthorized use of equipment and will not make allowance or credit for unauthorized use or access.

Contacting Ciena

Corporate headquarters	410-694-5700 or 800-921-1144	www.ciena.com
Customer technical support/warranty		
In North America	1-800-CIENA24 (243-6224) 410-865-4961	
In Europe, Middle East, and Africa	800-CIENA-24-7 (800-2436-2247) +44-207-012-5508	
In Asia-Pacific	800-CIENA-24-7 (800-2436-2247) +81-3-6367-3989 +91-124-4340-600	
In Caribbean and Latin America	800-CIENA-24-7 (800-2436-2247) 410-865-4944 (USA)	
Sales and General Information	410-694-5700	E-mail: sales@ciena.com
In North America	410-694-5700 or 800-207-3714	E-mail: sales@ciena.com
In Europe	+44-207-012-5500 (UK)	E-mail: sales@ciena.com
In Asia +81-3-3248-4680 (Japan)		E-mail: sales@ciena.com
In India	+91-124-434-0500	E-mail: sales@ciena.com
In Latin America	011-5255-1719-0220 (Mexico City)	E-mail: sales@ciena.com
Training	877-CIENA-TD (243-6283) or 410-865-8996	E-mail: techtnng@ciena.com

For additional office locations and phone numbers, please visit the Ciena website at www.ciena.com.

Change History

Release	Revision	Publication Date	Reason for Change
1	0.0	12/19/2017	Initial Release
1	b	04/09/2018	Updated rev only

Contents

REST APIs Demo Guide	5
Course Software	5
Purpose of the Demo's	5
Demos Folder Contents	6
Task 1: Install Fiddler.....	6
Task 2: Flickr API Call	8
Task 3: Resource Representations.....	10
Task 4: Service Calls With node.js.....	12

REST APIs Demo Guide

This guide has been provided as a walkthrough guide for the demonstration code used in this course. While publicly accessible APIs are available, we have decided to provide local demonstrations to give you the ability to have greater access to packet inspection, and to combat against the inevitable aging and deprecation of publicly accessible APIs. We encourage you to go online at some point, and experiment with APIs that are salient to your work or interests.

All demos are designed to run locally in conjunction with a packet capture / analysis tool. The bulk of the demos run using HTTP REST API calls using GET requests, and return either text or JSON.

Course Software

- Fiddler
- Node.js
- Internet Explorer (Use Internet Explorer for the jQuery demos)

The packet capture / analysis tool used in the course is the Telerik Fiddler Web Debugger. Fiddler may be downloaded for free at <https://www.telerik.com/download/fiddler>.

This course also uses node.js for a demonstration. Node may be downloaded from <https://nodejs.org/en/download/>. Be sure to choose the installation salient to your platform. Once installed, you may test the installation by running `node -v` in a terminal window.



This course also makes use of Internet Explorer. Not because Internet Explorer is turbo awesome, but because it still allows for the local execution of asynchronous JavaScript code. Internet Explorer does not block asynchronous JavaScript from running locally. Other browsers do block the local execution of asynchronous JavaScript. The JavaScript demos will not work in other browsers unless they are run on a web server. Installing a web server is beyond the scope of this course. Use Internet Explorer to run the JavaScript/jQuery demos.

Purpose of the Demo's

The purpose of the demonstrations is so that developers who are new to REST can work with fundamental concepts such as Client/Server interaction, HTTP Request/Response headers and bodies, HTTP Request Methods, and data transformation. Again, these are demonstrations. A full explanation of any programming language(s) or frameworks is beyond the scope of this course. If you are interested in an in-depth programming course, check SABA or contact Ciena Learning.

Demos Folder Contents

The demos folder contains 2 subdirectories and a readme file

- `http_js_jq`
- `nodedemo`
- `README.txt`

The `http_js_jq` directory contains simple HTML documents that may be run in Internet Explorer. Again, these examples use asynchronous JavaScript calls, which Internet Explorer does not block from running locally. Other browsers do block the local execution of asynchronous JavaScript. Use Internet Explorer to run the demos.

The contents of the `http_js_jq` directory are:

- `css/`
- `js/`
- `60s.xml`
- `70s.js`
- `80s.json`
- `1990s.html`
- `2000s.html`
- `flickrapicall.html`
- `multipleresources.html`
- `resourcerepresentation.html`

The following files are the example files: `flickrapicall.html`, `multipleresources.html`, `resourcerepresentation.html`. The rest of the files and directories are support files. The current directory and file structure is intact and should not be modified. If you modify the file or directory structure, you risk breaking the linkages between the files. Do not modify the structure of this directory.

The `nodedemo` folder contains a simple HTTP Server, and a simple HTTP Client that communicate using HTTP Get Requests and HTTP Responses. Node.js is required to run these examples. Once the server is up and running however, any software (browser, cURL, etc.) may be used to send GET requests. The server is not setup to handle POST requests.

The contents of the `nodedemo` directory are:

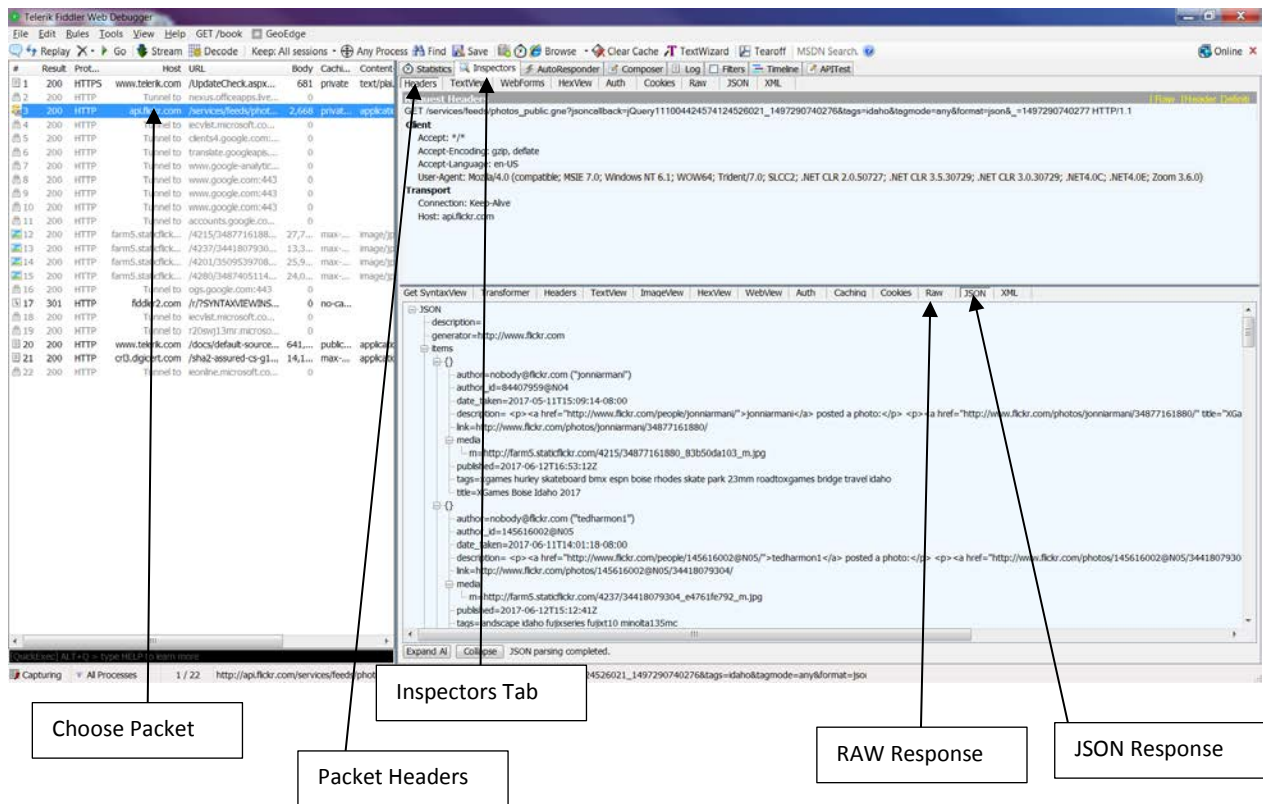
- `getlistener.js`
- `getserver.js`

Task 1: Install Fiddler

Fiddler is a free web debugging proxy. It is a packet capture and protocol analyzer tool. Fiddler is very easy to use and incredibly helpful for debugging, monitoring, or crafting HTTP request/response, body and header data

1. Download Fiddler from <https://www.telerik.com/download/fiddler>.
2. Navigate to the location you saved the fiddler setup file.

3. Run the `fiddlersetup.exe` program. Accept the defaults.
4. From this point forward, you may run fiddler on your system while you test your clients and servers. Fiddler will capture the packets, allow you to choose the packets you wish to analyze, and avail you to header and body information. Use the following figure to assist you in choosing monitor tabs.
5. The following figure is the result of the execution of the flickrapicall.html demo.

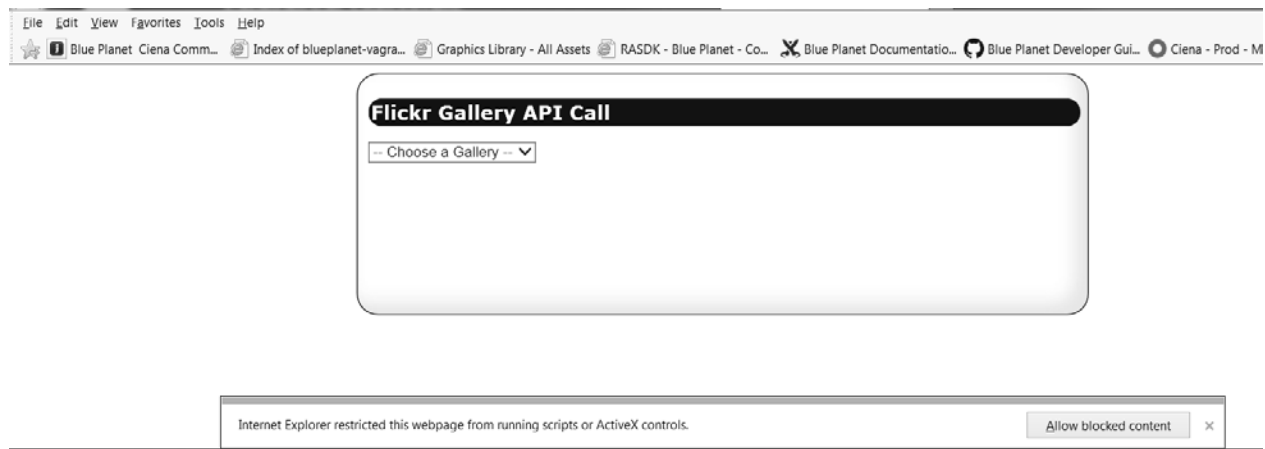


Task 2: Flickr API Call

The flickrapicall.html file is a simple file that uses the jQuery framework to initialize an asynchronous call to Flickr. The Flickr REST API call is an event driven call, catalyzed by clicking on a drop-down menu. The drop-down menu is used to append data to an HTTP GET request to instruct the API about which data will be returned. The API call returns a JSON payload that is parsed by the jQuery framework (custom code) and displays the results in the web page interface.

The notable takeaways from this demo are:

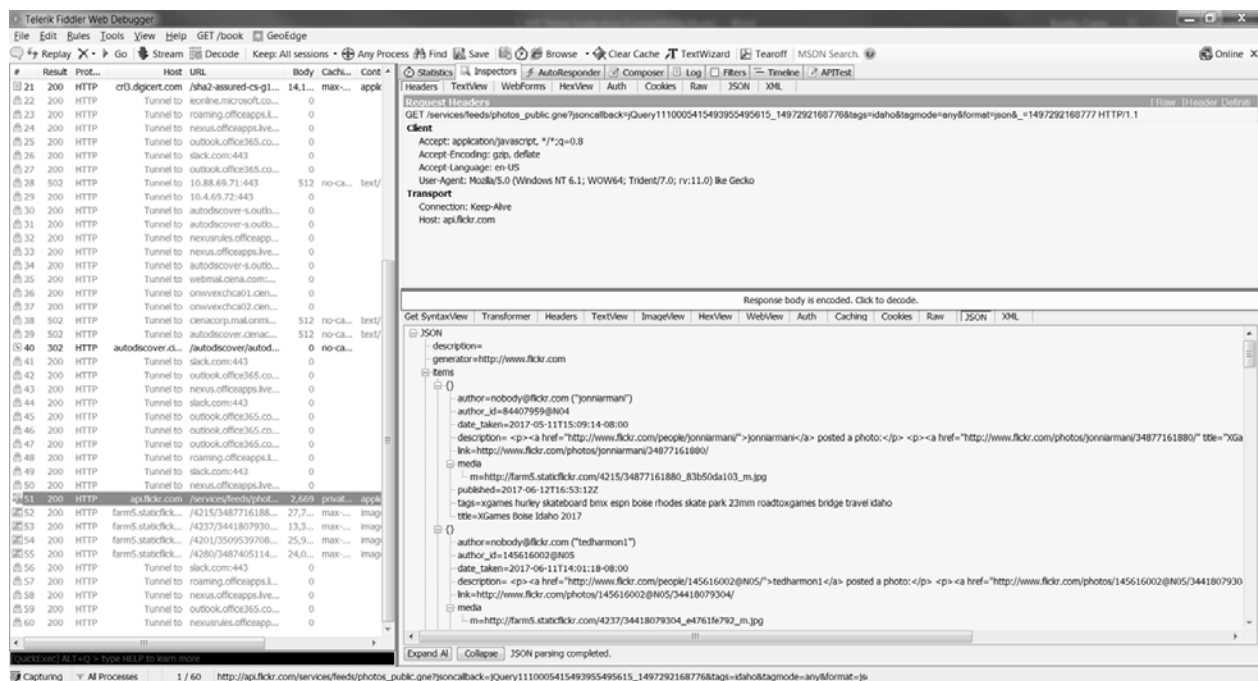
- Client\Server Interaction
 - Event Driven API calls
 - HTTP Request methods may be used to change the resource representation
 - JSON payload responses
 - Binary data responses
 - JSON payload transformation
1. Open fiddler and configure your interface (reference previous figure).
 2. Navigate to the location you stored the demo folder and open the http_js_jq directory.
 3. Locate the flickrapicall.html file and open the file in Internet Explorer. Authorize any security questions if they pop up. (Allow Blocked Content).



4. Click the Choose a gallery drop-down menu, and choose an option. The REST call will be generated and the results will be displayed in the gallery box.



5. In fiddler, locate the Flickr api call, and inspect the payload.



6. (Optional) Feel free to open the HTML file in any text editor to view the code. The following is a notable code snippet. A full explanation is beyond the scope of the course. The jQuery `$.getJSON` call generates a GET request and passes a plain JavaScript object of configuration instruction on the QueryString to the Flickr API. The response body is handled in the anonymous function that iterates through the JSON payload, targets the salient variables, and formats them into a presentational style suitable for the interface. In this case, as a series of images that are appended as content to a division of page space with the id of photos.

This snippet is the HTTP GET request to the Flickr API:

```
$.getJSON("http://api.flickr.com/services/feeds/photos_public.gne?jsoncallback=?",  
    {  
        tags: $dropdown.val(),  
        tagmode: "any",  
        format: "json"  
    },  
    function(data) {  
$.each(data.items, function(i, item) {  
    var img = $("");  
    img.attr("src", item.media.m).appendTo($photos);  
    if (i >= 3) return false;  
});  
});
```

7. Feel free to explore the file. A full explanation of all related code is beyond the scope of the course

Task 3: Resource Representations

There are 2 demo files that are used to help understand the concept of resource representations. The `resourcerepresentation.html` file transforms a JSON object into a drop-down menu in an HTML interface. The `multipleresources.html` file transforms many source objects (JSON, XML, and HTML) into a simple menu driven, informative HTML interface.

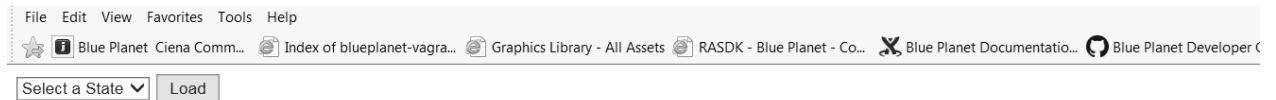
The notable takeaways from this demo are:

- Client\Server Interaction
- Event Driven API calls
- HTTP Request methods may be used to change the resource representation
- JSON payload responses
- JSON payload transformation
- Understanding the difference between the resource, and representation

As in the previous lab, keep fiddler open so you may view the packets

1. In the `demos/http_js_jq` directory, open the `resourcerepresentation.html` file in Internet Explorer.
2. Authorize any necessary security
3. You will be presented with a simple drop-down menu. The menu (unlike in the previous example) has not been populated yet. This is by design, as it helps to reinforce the event driven, and on demand behaviors of REST calls.

- Click the Load button to load the drop-down menu. This will populate the drop-down menu from a JSON object that has been loaded as a dictionary. The same JSON object will be used to alter the interface representation that the user sees, once a choice has been made.



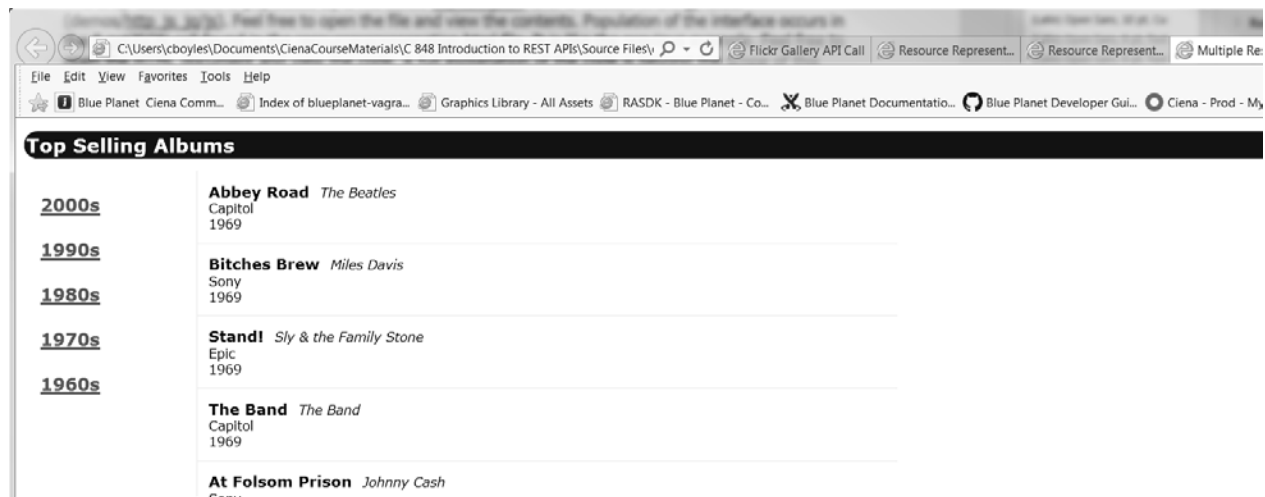
- Click the drop-down menu to return data from the JSON payload.



The drop-down menu is populated by a file called `capitals.json`, and is located in the `js` folder (`demos/http_js_jq/js`). Feel free to open the file and view the contents. Population of the interface occurs in the `$.getJSON` call found in the `resourcerepresentation.html` file. It is like the previous example. Feel free to open the HTML document and view the code. A full explanation of the code is beyond the scope of this course.

You will now continue with another example

- In the `demos/http_js_jq` directory, open the `multipleresources.html` file in Internet Explorer.
- Authorize any necessary security
- The `multipleresources.html` file presents a navigational driven, informative interface. Each link loads a different resource into the interface. This example is used to help learners understand that a single representation may be built from multiple resources.



9. The navigational links on the left-hand side make asynchronous, on demand calls to different types of resources to build the representation interface.
 - 1960s calls 60s.xml
 - 1970s calls 70s.js
 - 1980s calls 80s.json
 - 1990s calls 1990s.html
 - 2000s calls 2000s.html
10. Feel free to peruse the source code for the support resources and the `multipleresources.html` file

Task 4: Service Calls With node.js

The demo files for these demonstrations are in the `demos/nodedemo` directory. These demonstrations are used to demonstrate service to service calls and require node.js. If you haven't installed node.js yet, please do so before proceeding. You will be using 2 terminal windows to complete the demo.

This example is comprised of 2 files:

- `getlistener.js`
- `getsender.js`

The `getlistener.js` file is a simple HTTP server designed to respond to HTTP get requests. The file also logs data payloads it receives in the console window. The purpose of logging to the console window is so learners may see, real time, the logging that is occurring, without having to open another file.

The `getsender.js` file is used to send a JSON payload to the HTTP server via an HTTP GET request. The `getlistener.js` file is run as a console based application. Its purpose is to help the learner understand that any application that is aware of the TCP/IP stack, may make an HTTP call, and may transport JSON or any other agreed upon data exchange format.

The `getlistener.js` server, is not restricted to listening to just `getsender.js`. Any software may connect to the server and pass it a JSON payload. However, `getlistener.js` is not designed for

multi-dimensional processing. Ergo, any custom JSON data payloads that are passed should be single dimensional JSON or single dimensional JavaScript Objects.

NOTE: fiddler will not recognize the service to service calls. Use Internet Explorer to make a request to `getlistener.js` to view the packet exchange through fiddler.

Contents of `getlistener.js`

```
//getlistener.js
var http = require('http');
var querystring = require('querystring');
var server = http.createServer().listen(50000);

server.on('request', function(request,response) {
  if (request.method == 'GET') {
    var body = '';
    //append data chunk to body
    request.on('data', function (data) {
      body += data;
    });

    //data transmitted

    request.on('end', function () {
      var post = querystring.parse(body);
      console.log(post);
      response.writeHead(200, {'Content-Type' : 'text/plain'});
      response.end('Hello World\n');
    });
  }
});
console.log('server listening on 50000');
```

Contents of getsender.js

```
//getsender.js
var http = require('http');
var querystring = require('querystring');
var getData = querystring.stringify({
  "type": "contact", "id": "8675309",
  "fname": "Jenny", "lname": "Tutone",
  "created": "2015-05-22T14:56:29.000Z",
  "updated": "2015-05-22T14:56:28.000Z"
});
var options = {
  hostname: 'localhost', port: 50000, method: 'GET',
  headers: {
    'Content-Type': 'application/x-www-form-urlencoded',
    'Content-Length': getData.length
  }
};
var req = http.request(options, function(res) {
  console.log('STATUS: ' + res.statusCode);
  console.log('HEADERS: ' + JSON.stringify(res.headers));
  res.setEncoding('utf8');
  // get data as chunks
  res.on('data', function (chunk) {
    console.log('BODY: ' + chunk);
  });
  // end response
  res.on('end', function() {
    console.log('No more data in response.')
  })
});
req.on('error', function(e) {
  console.log('problem with request: ' + e.message);
});
// write data to request body
req.write(getData);
req.end();
```

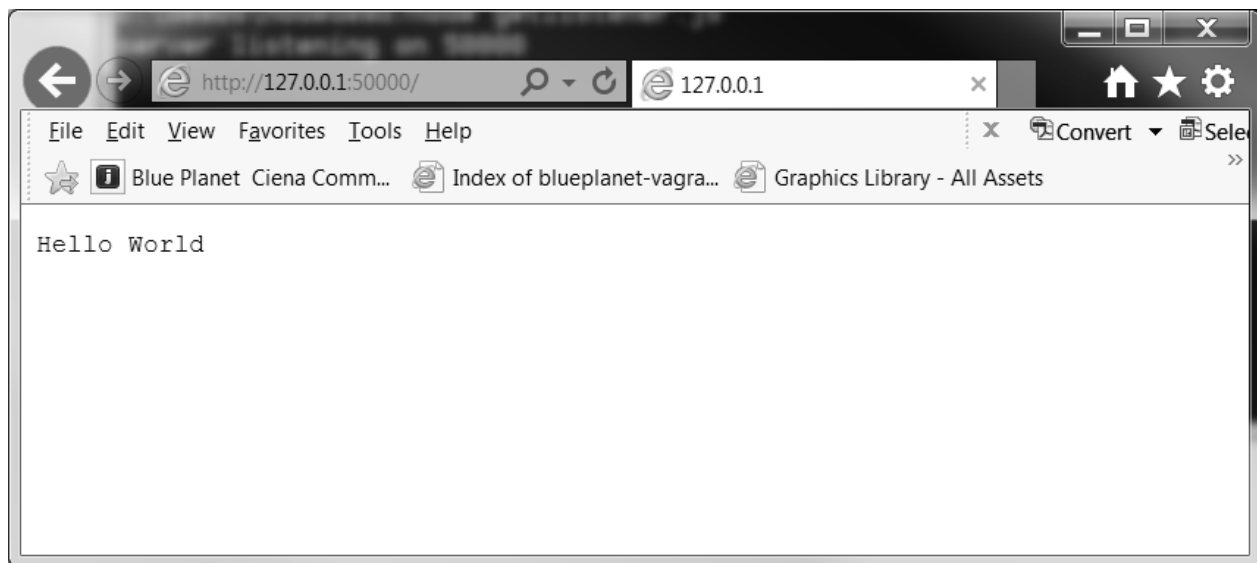
1. Open the first console window (cmd, term, etc.) and navigate to the `demos/nodedemo` directory. This console window should stay open. It will be used to run the web server `getlistener.js`.
2. You will now stand up an HTTP server with node.js, listening on port 50000. Execute `getlistener.js` with `node`. `node getlistener.js`



```
Administrator: C:\Windows\system32\cmd.exe - node getlistener.js

c:\demos\nodedemo>node getlistener.js
server listening on 50000
```

3. To test that your server is running: Open Internet Explorer (or any browser since this is a server) > and contact your local server at <http://127.0.0.1:50000> (or <http://localhost:50000> , or port 50000 using whatever name or address is attached to your system



4. Hello world is the HTTP response from the server. Check the console window running the server. You should see empty JSON payloads being processed. This is because no QueryString data was passed to the server.




```
Administrator: C:\Windows\system32\cmd.exe - node getlistener.js

c:\demos\nodedemo>node getlistener.js
server listening on 50000
{}
{}

```

5. You will now use node.js to make a request of your server. The `getsender.js` requester will send a JSON payload to the server, and will handle the response from the server. The server will also log the JSON payload in its console window. Feel free to run `getsender.js` more than once and monitor the responses. Execute `getsender.js` with `node. node getsender.js`



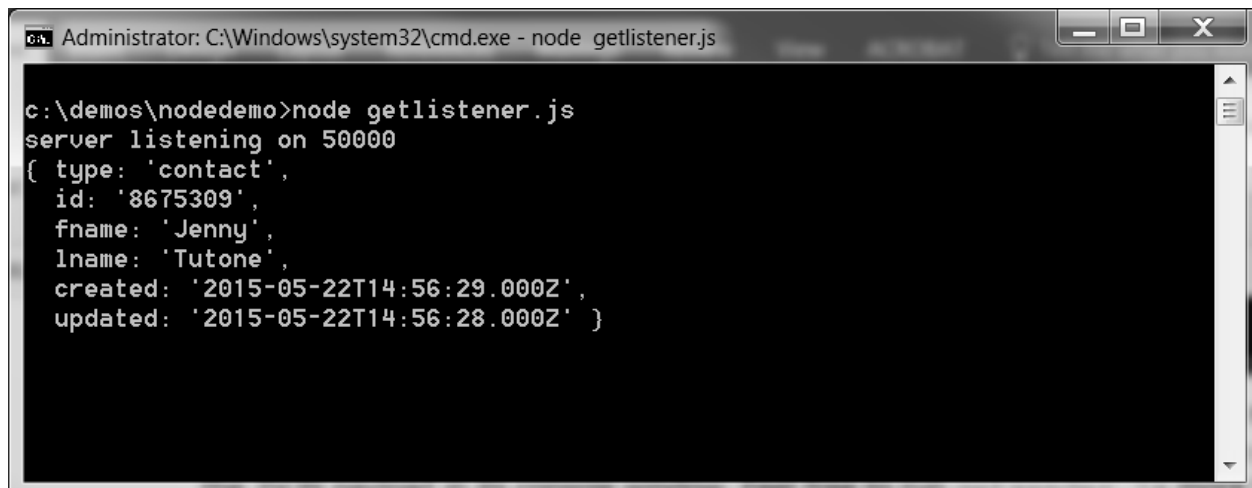
```
Administrator: C:\Windows\system32\cmd.exe

c:\demos\nodedemo>node getsender.js
STATUS: 200
HEADERS: {"content-type":"text/plain","date":"Mon, 12 Jun 2017 20:05:00 GMT","co
nnection":"close","transfer-encoding":"chunked"}
BODY: Hello World

No more data in response.

c:\demos\nodedemo>

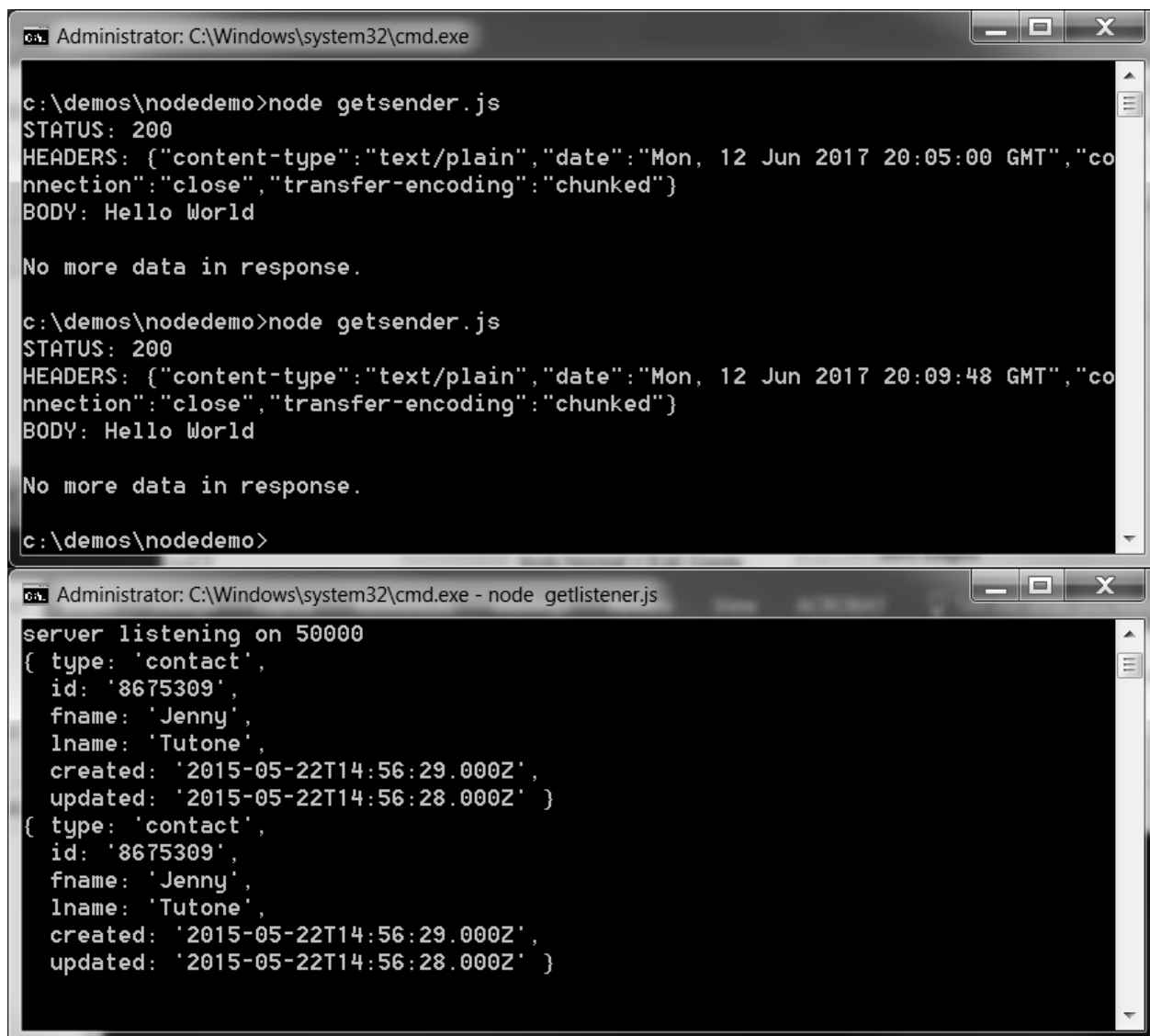
```

```
Administrator: C:\Windows\system32\cmd.exe - node getlistener.js

c:\demos\nodedemo>node getlistener.js
server listening on 50000
{ type: 'contact',
  id: '8675309',
  fname: 'Jenny',
  lname: 'Tutone',
  created: '2015-05-22T14:56:29.000Z',
  updated: '2015-05-22T14:56:28.000Z' }
```

6. Screen shots of multiple calls to the server



```
Administrator: C:\Windows\system32\cmd.exe

c:\demos\nodedemo>node getsender.js
STATUS: 200
HEADERS: {"content-type":"text/plain","date":"Mon, 12 Jun 2017 20:05:00 GMT","connection":"close","transfer-encoding":"chunked"}
BODY: Hello World

No more data in response.

c:\demos\nodedemo>node getsender.js
STATUS: 200
HEADERS: {"content-type":"text/plain","date":"Mon, 12 Jun 2017 20:09:48 GMT","connection":"close","transfer-encoding":"chunked"}
BODY: Hello World

No more data in response.

c:\demos\nodedemo>

Administrator: C:\Windows\system32\cmd.exe - node getlistener.js

server listening on 50000
{ type: 'contact',
  id: '8675309',
  fname: 'Jenny',
  lname: 'Tutone',
  created: '2015-05-22T14:56:29.000Z',
  updated: '2015-05-22T14:56:28.000Z' }
{ type: 'contact',
  id: '8675309',
  fname: 'Jenny',
  lname: 'Tutone',
  created: '2015-05-22T14:56:29.000Z',
  updated: '2015-05-22T14:56:28.000Z' }
```