A Beginner's Guide to HTTP and REST

by Ludovico Fischer 9 Jan 2013

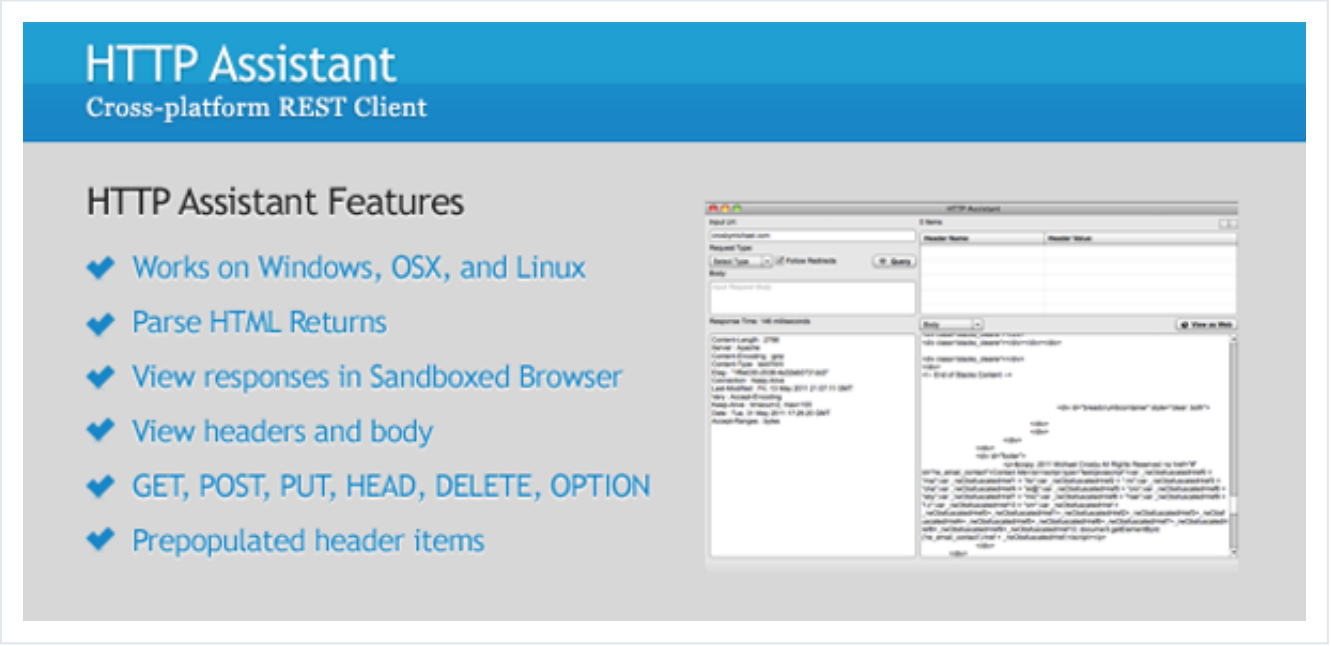
Difficulty: Intermediate Length: Quick Languages: English

Web Development | REST | HTTP | JavaScript | Ajax | Web Servers

Hypertext Transfer Protocol (HTTP) is the life of the web. It's used every time you transfer a document, or make an `AJAX` request. But HTTP is surprisingly a relative unknown among some web developers.

This introduction will demonstrate how the set of design principles, known as REST, underpin HTTP, and allow you to embrace its fullest power by building interfaces, which can be used from nearly any device or operating system.

Envato Market also has thousands of useful code scripts, plugins and apps to help you with web development, such as HTTP Assistant, an app that helps you test online web services.



The HTTP Assistant app on Envato Market

Republished Tutorial

Every few weeks, we revisit some of our readers' favorite posts from throughout the history of the site. This tutorial was first published in November, 2010.

Why REST?

> *REST is a simple way to organize interactions between independent systems.*

REST is a simple way to organize interactions between independent systems. It's been growing in popularity since 2005, and inspires the design of services, such as the Twitter API. This is due to the fact that REST allows you to interact with minimal overhead with clients as diverse as mobile phones and other websites. In theory, REST is not tied to the web, but it's almost always implemented as such, and was inspired by HTTP. As a result, REST can be used wherever HTTP can.

The alternative is building relatively complex conventions on top of HTTP. Often, this takes the shape of entire new XML-based languages. The most illustrious example is SOAP. You have to learn a completely new set of conventions, but you never use HTTP to its fullest power. Because REST has been inspired by HTTP and plays to its strengths, it is the best way to learn how HTTP works.

After an initial overview, we'll examine each of the HTTP building blocks: URLs, HTTP verbs and response codes. We'll also review how to use them in a RESTful way. Along the way, we'll illustrate the theory with an example application, which simulates the process of keeping track of data related to a company's clients through a web interface.

HTTP

> *HTTP is the protocol that allows for sending documents back and forth on the web.*

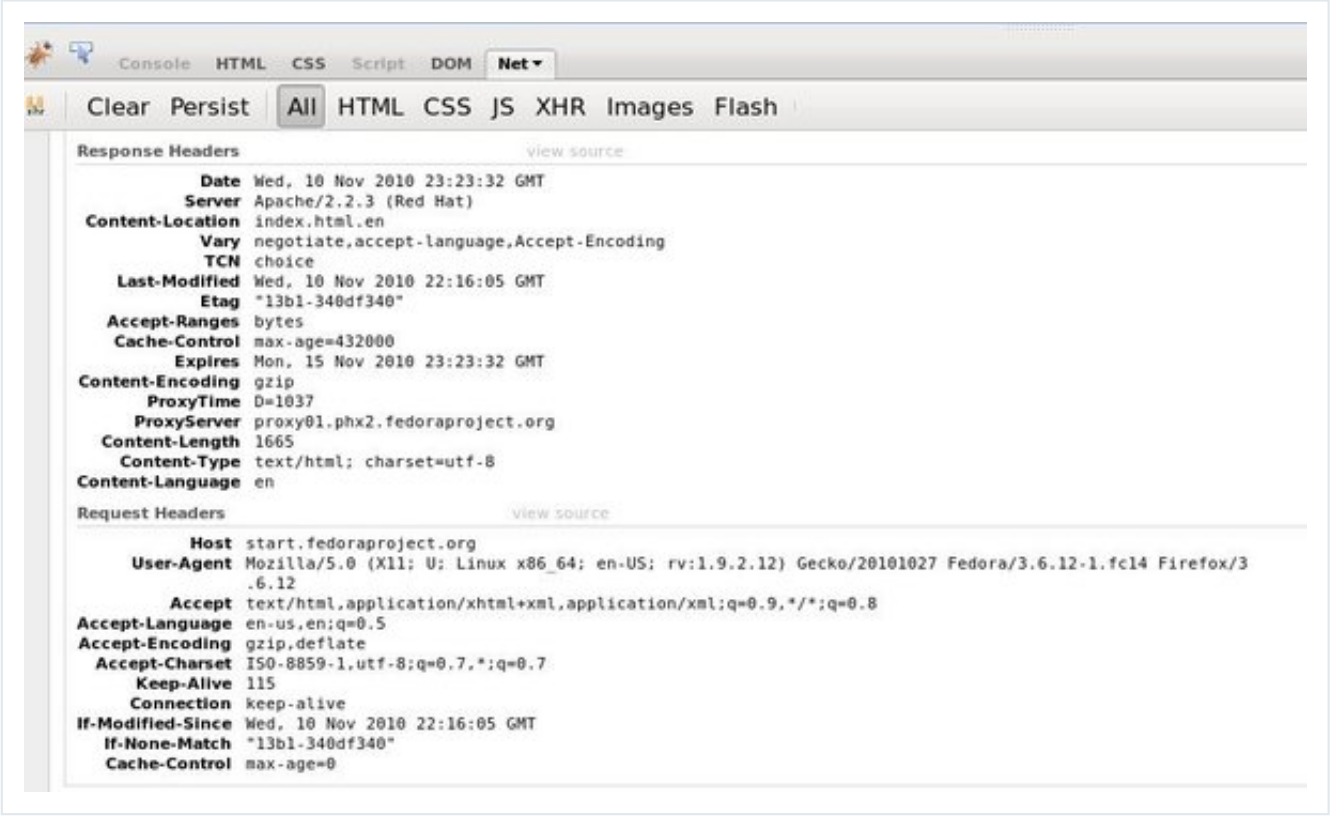
HTTP is the protocol that allows for sending documents back and forth on the web. A protocol is a set of rules that determines which messages can be exchanged, and which messages are appropriate replies to others. Another common protocol is POP3, which you might use to fetch email on your hard disk.

In HTTP, there are two different roles: server and client. In general, the client always initiates the conversation; the server replies. HTTP is text based; that is, messages are essentially bits of text, although the message body can also contain other media. Text usage makes it easy to monitor an HTTP exchange.

HTTP messages are made of a header and a body. The body can often remain empty; it contains data that you want to transmit over the network, in order to use it according to the instructions in the header. The header contains metadata, such as encoding information; but, in the case of a request, it also contains the important HTTP methods. In the REST style, you will find that header data is often more significant than the body.

Spying HTTP at Work

If you use Chrome Developer Tools, or Firefox with the Firebug extension installed, click on the `Net` panel, and set it to `enabled`. You will then have the ability to view the details of the HTTP requests as you surf. For example:



Another helpful way to familiarize yourself with HTTP is to use a dedicated client, such as cURL.

> *cURL is a command line tool that is available on all major operating systems.*

Once you have cURL installed, type:

```
1 curl -v google.com
```

This will display the complete HTTP conversation. Requests are preceded by `>`, while responses are preceded by `<`.

URLS

URLs are how you identify the things that you want to operate on. We say that each URL identifies a resource. These are exactly the same URLs which are assigned to web pages. In fact, a web page is a type of resource. Let's take a more exotic example, and consider our sample application, which manages the list of a company's clients:

```
/clients
```

will identify all clients, while

```
/clients/jim
```

will identify the client, named 'Jim', assuming that he is the only one with that name.

In these examples, we do not generally include the hostname in the URL, as it is irrelevant from the standpoint of how the interface is organized. Nevertheless, the hostname is important to ensure that the resource identifier is unique all over the web. We often say you send the request *for* a resource *to* a host. The host is included in the header separately from the resource path, which comes right on top of the request header:

```
GET /clients/jim HTTP/1.1

Host: example.com
```

Resources are best thought of as nouns. For example, the following is not RESTful:

```
/clients/add
```

This is because it uses a URL to describe an action. This is a fairly fundamental point in distinguishing RESTful from non-RESTful systems.

Finally, URLs should be as precise as needed; everything needed to uniquely identify a resource should be in the URL. You should not need to include data identifying the resource in the request. This way, URLs act as a complete map of all the data your application handles.

But how do you specify an action? For example, how do you tell that you want a new client record created instead of retrieved? That is where HTTP verbs come into play.

HTTP Verbs

Each request specifies a certain HTTP verb, or method, in the request header. This is the first all caps word in the request header. For instance,

```
1 GET / HTTP/1.1
```

means the GET method is being used, while

```
1 DELETE /clients/anne HTTP/1.1
```

means the `DELETE` method is being used.

> *HTTP verbs tell the server what to do with the data identified by the URL.*

HTTP verbs tell the server what to do with the data identified by the URL. The request can optionally contain additional information in its body, which might be required to perform the operation - for instance, data you want to store with the resource. You can supply this data in cURL with the `-d` option.

If you've ever created HTML forms, you'll be familiar with two of the most important HTTP verbs: `GET` and `POST`. But there are far more HTTP verbs available. The most important ones for building RESTful API are `GET`, `POST`, `PUT` and `DELETE`. Other methods are available, such as `HEAD` and `OPTIONS`, but they are more rare (if you want to know about all other HTTP methods, the official source is IETF).

GET

`GET` is the simplest type of HTTP request method; the one that browsers use each time you click a link or type a URL into the address bar. It instructs the server to transmit the data identified by the URL to the client. Data should never be modified on the server side as a result of a `GET` request. In this sense, a `GET` request is read-only, but of course, once the client receives the data, it is free to do

any operation with it on its own side - for instance, format it for display.

PUT

A `PUT` request is used when you wish to create or update the resource identified by the URL. For example,

```
PUT /clients/robin
```

might create a client, called `Robin` on the server. You will notice that `REST` is completely backend agnostic; there is nothing in the request that informs the server how the data should be created - just that it should. This allows you to easily swap the backend technology if the need should arise. `PUT` requests contain the data to use in updating or creating the resource in the body. In cURL, you can add data to the request with the `-d` switch.

```
curl -v -X PUT -d "some text"
```

DELETE

`DELETE` should perform the contrary of `PUT`; it should be used when you want to delete the resource identified by the URL of the request.

```
curl -v -X DELETE /clients/anne
```

This will delete all data associated with the resource, identified by `/clients/anne`.

POST

`POST` is used when the processing you wish to happen on the server should be repeated, if the `POST` request is repeated (that is, they are not *idempotent*; more on that below). In addition, `POST` requests should cause processing of the request body as a subordinate of the URL you are posting to.

In plain words:

```
POST /clients/
```

should not cause the resource at `/clients/`, itself, to be modified, but a resource whose URL *starts with* `/clients/`. For instance, it could append a new client to the list, with an `id` generated by the server.

```
1 /clients/some-unique-id
```

`PUT` requests are used easily instead of `POST` requests, and vice versa. Some systems use only one, some use `POST` for create operations, and `PUT` for update operations (since with a `PUT` request you always supply the complete URL), some even use `POST` for updates and `PUT` for creates.

Often, `POST` requests are used to trigger operations on the server, which do not fit into the `Create/Update/Delete` paradigm; but this, however, is beyond the scope of `REST`. In our example, we'll stick with `PUT` all the way.

Classifying HTTP Methods

Safe and unsafe methods:
safe methods are those that never modify resources. The only safe methods, from the four listed above, is `GET`. The others are unsafe, because they may result in a modification of the resources.

Idempotent methods:
These methods achieve the same result, no matter how many times the request is repeated: they are `GET`, `PUT`, and `DELETE`. The only non idempotent method is `POST`. `PUT` and `DELETE` being considered idempotent might be surprising, though, it, in fact, is quite easy to explain: repeating a `PUT` method with exactly the same body should modify a resource in a way that it remains identical to the one described in the previous `PUT` request: nothing will change! Similarly, it makes no sense to delete a resource twice. It follows that no matter how many times a `PUT` or `DELETE` request is repeated, the result should be the same as if it had been done only once.
Remember: it's you, the programmer, who ultimately decides what happens when

a certain HTTP method is used. There is nothing inherent to HTTP implementations that will automatically cause resources to be created, listed, deleted, or updated. You must be careful to apply the HTTP protocol correctly and enforce these semantics yourself.

Representations

> *The HTTP client and HTTP server exchange information about resources identified by URLs.*

We can sum up what we have learned so far in the following way: the HTTP client and HTTP server exchange information about resources identified by URLs.

We say that the request and response contain a representation of the resource. By representation, we mean information, in a certain format, about the state of the resource or how that state should be in the future. Both the header and the body are pieces of the representation.

The HTTP headers, which contain metadata, are tightly defined by the HTTP spec; they can only contain plain text, and must be formatted in a certain manner.

The body can contain data in any format, and this is where the power of HTTP truly shines. You know that you can send plain text, pictures, HTML, and XML in any human language. Through request metadata or different URLs, you can choose between different representations for the same resource. For example, you might send a webpage to browsers and JSON to applications.

The HTTP response should specify the content type of the body. This is done in the header, in the *Content-Type* field; for instance:

```
1 Content/Type: application/json
```

For simplicity, our example application only sends JSON back and forth, but the application should be architectured in such a way that you can easily change the

format of the data, to tailor for different clients or user preferences.

HTTP Client Libraries

> *cURL is, more often than not, the HTTP client solution of choice for PHP developers.*

To experiment with the different request methods, you need a client, which allows you to specify which method to use. Unfortunately, HTML forms do not fit the bill, as they only allow you to make GET and POST requests. In real life, APIs are accessed programmatically through a separate client application, or through JavaScript in the browser.

This is the reason why, in addition to the server, it is essential to have good HTTP client capabilities available in your programming language of choice.

A very popular HTTP client library is, again, cURL. You've already been familiarized with the cURL command from earlier in this tutorial. cURL includes both a standalone command line program, and a library that can be used by various programming languages. In particular, cURL is, more often than not, the HTTP client solution of choice for PHP developers. Other languages, such as Python, offer more native HTTP client libraries.

Setting up the Example Application

> *I want to expose the low-level functionality as much as possible.*

Our example PHP application is extremely barebones. I want to expose the low-level functionality as much as possible, without any framework magic. I also did not want to use a real API, such as Twitter's, because they are subject to change unexpectedly, you need to setup authentication, which can be a hassle, and, obviously, you cannot study the implementation.

To run the example application, you will need to install PHP5 and a web server, with some mechanism to run PHP. The current version must be at least version 5.2 to have access to the `json_encode()` and `json_decode()` functions.

As for servers, the most common choice is still Apache with *mod_php*, but you're free to use any alternatives that you're comfortable with. There is a sample Apache configuration, which contains rewrite rules to help you setup the application quickly. All requests to any URL, starting with /clients/, must be routed to our *server.php* file.

In Apache, you need to enable *mod_rewrite* and put the supplied *mod_rewrite* configuration somewhere in your Apache configuration, or your *.htacess* file. This way, *server.php* will respond to all requests coming from the server. The same must be achieved with Nginx, or whichever alternative server you decide to use.

How the Example Applications Works

There are two keys to processing requests the REST way. The first key is to initiate different processing, depending on the HTTP method - even when the URLs are the same. In PHP, there is a variable in the *$_SERVER* global array, which determines which method has been used to make the request:

```
1 $_SERVER['REQUEST_METHOD']
```

This variable contains the method name as a string, for instance '`GET`', '`PUT`', and so on.

The other key is to know which URL has been requested. To do this, we use another standard PHP variable:

```
1 $_SERVER['REQUEST_URI']
```

This variable contains the URL starting from the first forward slash. For instance, if the host name is '`example.com`', '`http://example.com/`' would return '`/`', while

'`http://example.com/test/`' would return '`/test/`'.

Let's first attempt to determine which URL has been called. We only consider URLs starting with '`clients`'. All other are invalid.

```
$resource = array_shift($paths);

if ($resource == 'clients') {
    $name = array_shift($paths);

    if (empty($name)) {
        $this->handle_base($method);
    } else {
        $this->handle_name($method, $name);
    }

} else {
    // We only handle resources under 'clients'
    header('HTTP/1.1 404 Not Found');
}
```

We have two possible outcomes:

- The resource is the clients, in which case, we return a complete listing
- There is a further identifier

If there is a further identifier, we assume it is the client's name, and, again, forward it to a different function, depending on the `method`. We use a `switch` statement, which should be avoided in a real application:

```
switch($method) {
  case 'PUT':
      $this->create_contact($name);
      break;

  case 'DELETE':
      $this->delete_contact($name);
      break;

  case 'GET':
      $this->display_contact($name);
      break;

  default:
      header('HTTP/1.1 405 Method Not Allowed');
      header('Allow: GET, PUT, DELETE');
      break;
```

```
18  }
```

Response Codes

> *HTTP response codes standardize a way of informing the client about the result of its request.*

You might have noticed that the example application uses the PHP `header()`, passing some strange looking strings as arguments. The *header()* function prints the HTTP `headers` and ensures that they are formatted appropriately. Headers should be the first thing on the response, so you shouldn't output anything else before you are done with the headers. Sometimes, your HTTP server may be configured to add other headers, in addition to those you specify in your code.

Headers contain all sort of meta information; for example, the text encoding used in the message body or the MIME type of the body's content. In this case, we are explicitly specifying the HTTP response codes. HTTP response codes standardize a way of informing the client about the result of its request. By default, PHP returns a `200` response code, which means that the response is successful.

The server should return the most appropriate HTTP response code; this way, the client can attempt to repair its errors, assuming there are any. Most people are familiar with the common `404 Not Found` response code, however, there are a lot more available to fit a wide variety of situations.

Keep in mind that the meaning of a HTTP response code is not extremely precise; this is a consequence of HTTP itself being rather generic. You should attempt to use the response code which most closely matches the situation at hand. That being said, do not worry too much if you cannot find an exact fit.

Here are some HTTP response codes, which are often used with REST:

200 OK

This response code indicates that the request was successful.

201 Created

This indicates the request was successful and a resource was created. It is used to confirm success of a `PUT` or `POST` request.

400 Bad Request

The request was malformed. This happens especially with `POST` and `PUT` requests, when the data does not pass validation, or is in the wrong format.

404 Not Found

This response indicates that the required resource could not be found. This is generally returned to all requests which point to a URL with no corresponding resource.

401 Unauthorized

This error indicates that you need to perform authentication before accessing the resource.

405 Method Not Allowed

The HTTP method used is not supported for this resource.

409 Conflict

This indicates a conflict. For instance, you are using a `PUT` request to create the same resource twice.

500 Internal Server Error

When all else fails; generally, a 500 response is used when processing fails due to unanticipated circumstances on the server side, which causes the server to error out.

Exercising the Example Application

Let's begin by simply fetching information from the application. We want the details of the client, '`jim`', so let's send a simple `GET` request to the URL for this resource:

```
curl -v http://localhost:80/clients/jim
```

This will display the complete message headers. The last line in the response will be the message body; in this case, it will be JSON containing Jim's address (remember that omitting a method name will result in a `GET` request; also replace `localhost:80` with the server name and port you are using).

Next, we can obtain the information for all clients at once:

```
curl -v http://localhost:80/clients/
```

To create a new client, named Paul...

```
curl -v -X PUT http://localhost:80/clients/paul -d '{"address":"Sunset Boulevar
```

and you will receive the list of all clients now containing Paul as a confirmation.

Finally, to delete a client:

```
curl -v -X DELETE http://localhost:80/clients/anne
```

You will find that the returned JSON no longer contains any data about Anne.

If you try to retrieve a non-existing client, for example:

```
curl -v http://localhost:80/clients/jerry
```

You will obtain a 404 error, while, if you attempt to create a client which already exists:

curl -v -X PUT http://localhost:80/clients/anne

You will instead receive a 409 error.

Conclusion

> *In general, the less assumptions beyond HTTP you make, the better.*

It's important to remember that HTTP was conceived to communicate between systems, which share nothing but an understanding of the protocol. In general, the less assumptions beyond HTTP you make, the better: this allows the widest range of programs and devices to access your API.

I used PHP in this tutorial, because it is most likely the language most familiar to Nettuts+ readers. That said, PHP, although designed for the web, is probably not the best language to use when working in a REST way, as it handles `PUT` requests in a completely different fashion than `GET` and `POST`.

Beyond PHP, you might consider the following:

- The various Ruby frameworks (Rails and Sinatra)
- There's excellent REST support in Python. Plain Django and WebOb, or Werkzeug should work
- node.js has excellent support for REST

Among the applications which attempt to adhere to REST principles, the classic example is the Atom Publishing Protocol, though it's honestly not used too often in practice. For a modern application, which is built on the philosophy of using HTTP to the fullest, refer to Apache CouchDB.

And don't forget to check out the selection of code scripts, plugins and apps on Envato Market.

Have fun!

Ludovico Fischer

I am software developer. I have started my carrier by building websites. Now I focus on complex enterprise systems. Currently taking some time off studying a for Master's degree in applied mathematics.

Download Attachment

Translations

Envato Tuts+ tutorials are translated into other languages by our community members—you can be involved too!

Translate this post

116 Comments Nettuts+ LeDanielH

Recommend 11 Share Sort by Best

Join the discussion…

Samuel • 3 years ago
Why should one not use switch statements in production as stated in the tutorial?
23 • Reply • Share ›

x Samuel • 2 years ago
http://google.com
5 • Reply • Share ›

John Slegers Samuel • 2 years ago
I wasn't aware of this myself, but it appears that "switch / case" in PHP is slower than "if / else if / else". Especially compared with "if / else if / else" statements using "===", "switch / case" is considerable slower. See http://www.phpbench.com/ .
• Reply • Share ›

Samuel John Slegers • 2 years ago
Yes, but performance is not always (but often is) a good reason for not using something in production. Using switch/case might have something to do with readability etc and the performance difference could be negligible. Sometimes it makes more sense to use switches.
4 • Reply • Share ›

Monie • 5 years ago
I am lost! It's too complicated man!
26 • Reply • Share ›

MPinteractiv Monie • 3 years ago
web dev 101 . HTTP is the single most important thing a web developper should know imho. I wish PHP api was more restfull and less "magic".
8 • Reply • Share ›

Brian Moeller • 3 years ago
Fantastic! I've built RESTful services before, but sometimes it's nice to read articles on the basics, again. Some knowledge that you take for granted or you dismiss over time and experience come back to light.
6 • Reply • Share ›

Manor Minors Brian Moeller • 10 months ago

Hi Brian, I'm kind of lost setting up this example application. looks like you have successfully tested it.

This tutorial says
"There is a sample Apache configuration, which contains rewrite rules to help you setup the application quickly. All requests to any URL, starting with /clients/, must be routed to our server.php file"
Where is that sample code which needs to be added in Apache Configuration?
Any help is greatly appreciated.

 • Reply • Share ›

Brian Moeller ➚ Manor Minors • 10 months ago

He doesn't supply any HTACCESS configurations. Here is a quick and simple one.

#.htaccess file configuration
#=======================
RewriteEngine On # Turn on the rewriting engine
RewriteRule ^clients/([a-z]+)/?$ server.php?name=$1 [NC,L] # Handle client requests

example.... http://www.test.com/clients/jo...
rewrite to.... http://www.test.com/server.php...
#=======================
In this example I'm passing part of the URI ([a-z]+) to the server.php file in the form of a querystring. This is not necessary and you can just remove the querystring at the end of the sever.php. Just threw it in there as reference. The code that author does provide will handle the method and name lookup from the request URI without the querystring.

Some good resources if you are not very familiar with htaccess rewrite rules...

https://www.addedbytes.com/art...

LOVE THIS!!! HTACCESS Tester....Beats bashing your brains out in development.
http://htaccess.madewithlove.b...

 • Reply • Share ›

Manor Minors ➚ Brian Moeller • 10 months ago

Hi Brian.
Thank you so much!
I'm going through the link you sent me. Will let you know if I'm able to set it up!
Thanks again!

1 • Reply • Share ›

Josh Thomas • 3 years ago

Shouldn't the 409 Conflict text read - "you are using a POST request to create the same resource twice."

6 • Reply • Share ›

Йордан Иванов • 3 years ago

There are also some very usefull resting and debugging browse extensions like Rest Console for Chome (https://chrome.google.com/webs...) and Rest Clinet for FireFox(https://addons.mozilla.org/bg/...)

5 • Reply • Share ›

Tanmay • 5 years ago

Nice tutorial.

But for a beginner to understand the code, if you can define more clearly how you got the $paths variable. I know its from $_SERVER['REQUEST_URI'], but again you are using array_shift, which means the $_SERVER['REQUEST_URI'] should be splited into an array.

But really great article to understand all the pieces.

Thanks
Tanmay

4 • Reply • Share ›

Torkild Dyvik Olsen ➦ Tanmay • 5 years ago

In most (RESTful) situations, you would only have to split the uri into segments by using $path = explode('/', $_SERVER['REQUEST_URI']);
Don't have to make it harder than that :)

1 • Reply • Share ›

Dale Hurley ➦ Torkild Dyvik Olsen • 5 years ago

Be careful with explode. It will split the URL into an array where the first value is blank because of the leading slash.

ie /client/jim/ becomes
[0] = ''
[1] = 'client'
[2] = 'jim'
[4] = ''

1 • Reply • Share ›

Igor • 3 years ago

Has anyone tested the 'PUT' method? I downloaded the code and

```
file_get_contents('php://input')
```

returns null every time. Here's the cURL command I use :

```
curl -i -X PUT -d '{"address":"Sunset Boulevard"}' http://localhost/clients/ryan
```

```
curl -i -X PUT -d '{ address : Sunset Boulevard }' http://localhost/clients/ryan
```

The other methods work fine. Does anyone know why this is happening?

3 • Reply • Share ›

chai ➚ Igor • a year ago

It is because of the 'single "double" quotes' not working on windows. Below should work (use escape '\' character for quotes in quotes)
curl -i -X PUT -d "{\"address\":\"Sunset Boulevard\"}" http://localhost/clients/ryan

• Reply • Share ›

Amaan Rizvi ➚ Igor • 2 years ago

curl -v -X PUT http://localhost/clients/ryan -d '{"address":"Sunset Boulevard"}'
try this.

• Reply • Share ›

regie • 3 years ago

i have learnt a lot regarding the concept and idea of REST but i was a bit lost on "Exercising the Example Application". i was wondering on how to simulate the data transfer and requests and where on my localhost can i generate or how to create the resources on the example. hope to hear from you. thanks.

3 • Reply • Share ›

gweiss27 • 2 years ago

AUTHOR!!! UPDATE YOUR POST SO OTHERS DON'T HAVE TO GO THROUGH THE
SAME FRUSTRATIONS YOUR READERS, WHO TOOK THE TIME TO READ AND APPRECIATE YOUR WORK, WENT THROUGH!!!!! #### READERS: read the comments to find the updates you need to even get this example to work before you waste your time!

Very nice tutorial. However, I MUST express my frustration in the fact it took like 2 hours to get this very simple application to work on my already very complete Apache Macbook setup. Further, I had to go through the long list of comments just to find that others were experiencing the same problems and follow their suggestions. Clearly, they were valid. Why did this author not take the time to update the article and at least put a LITTLE more time in helping the audience get this setup correctly on their machines to actually use and benefit from the knowledge in this article. This happens so often in tutorials and even when you try to move to more and more basic tutorials to start with the even the SIMPLEST of things, it seems there's always some wild goose chase to get it setup. AND, nearly ever time, these problems are ONLY RESOLVED by some very smart commenters who have bothered to resolve these problems. It's just very frustrating.

4 • Reply • Share ›

Erutan409 ➚ gweiss27 • a year ago

So...you're just going to complain about how long it took to figure out how to get the code to work and NOT post your resolution(s) on how you fixed it?

• Reply • Share ›

gweiss27 ➚ Erutan409 • a year ago

Is this a joke? I didn't write the article. If I write a post one day and its shoddy like this, you are more than welcome to rip me a new one for poor writing, but I owe you and the people on this thread nothing on this particular point. And this was like 5 months ago. If you really want my solution, I can probably dig it up for you, but I don't even remember what it was at this point.

^ | v • Reply • Share ›

Erutan409 ➚ gweiss27 • a year ago

No, not a joke. I'm merely suggesting that instead of complaining about how long it took for you to find the solution to a problem (that you don't want other people to suffer through)...maybe make your rant a little more constructive and offer your findings as well.

5 months or not - it's the Internet. It's [almost] practically timeless when it comes to sharing knowledge. If it's too much of a hassle to dredge up your working code, don't bother. I was trying to help you be more constructive in your future "reviews" of other peoples' tutorials.

2 ^ | v • Reply • Share ›

gweiss27 ➚ Erutan409 • a year ago

Well it certainly did not come across as constructive or suggestive. It came across as attacking, judgemental and harsh. Reword YOUR comments and maybe next time you'll get a much positive response. No problems with your point, but your delivery needs a lot of work.

^ | v • Reply • Share ›

Erutan409 ➚ gweiss27 • a year ago

"It came across as attacking, judgemental and harsh."

Perhaps you should re-read your initial post; objectively.

3 ^ | v • Reply • Share ›

gweiss27 ➚ Erutan409 • a year ago

I wrote my original post out of frustration. I stand by what I wrote, and I expressed my feelings directly to the author who put a tutorial out that was meant to instruct people, not some 3rd party nobody who is chiming in with unwarranted comments and putting his 2 cents in just to make himself feel better. I addressed this issue in my first post. If I was writing a tutorial for others to use, it would have been written better and not had these problems, and when I do that, you are more than welcome to comment on my post's effectiveness at that time. Until then, no one cares what you think, so buzz off.

1 ^ | v • Reply • Share ›

Erutan409 ↗ gweiss27 • a year ago

So constructive.

• Reply • Share ›

gweiss27 ↗ Erutan409 • a year ago

Where's your working solution? Talk about constructive. I dont see you with any solutions either, just another web users randomly annoying people without purpose. Point the finger at yourself first buddy.

1 • Reply • Share ›

Erutan409 ↗ gweiss27 • a year ago

If I remember, after I'm finished writing up my own solution for what this tutorial outlines, I'll be sure to upload it to my gist account and share it.

Quit being so defensive and hostile when people give you [constructive] feedback. It doesn't bode well for networking.

• Reply • Share ›

gweiss27 ↗ Erutan409 • a year ago

1.) You didn't give me feedback. You rudely jumped into a comment thread that had nothing to do with you. And your feedback wasn't voiced at all constructively.
2.) You only criticized someone else instead of addressing the problem which was to add a working solution, which I pointed out so your unsolicited comments only made things worse.
3.) I dont network on Disqus and I certainly dont care about networking with someone who I am offering criticism too. If I was trying to network with this person, my overtures would have been much more open and friendly. There are about a million other ways to network professionally and you don't know a single thing about me or how I interact with people face to face or through other virtual mediums, so your comments are baseless and pointless.

Anything else?

1 • Reply • Share ›

Erutan409 ↗ gweiss27 • a year ago

Sure. I admit that this back and forth has gone on longer than necessary. However, I will leave you with this:

You're on the right track with me not knowing you. However, the first impression I now have with you says a lot about your character. I know - you don't care what I think about you and that's fine. Your post mostly seemed to merely be a vehicle for you to chastise the author for giving you a hard time while trying to figure out some code.

Either offer your findings with your rant or go somewhere else.

I pointed out your complaining (you take liberty associating a negative connotation with) that could have also been balanced out by you actually posting your corrections.

Either way, it doesn't matter and I regret waking a sleeping giant. If not only for the unnecessary confrontatonal attitude. Have a wonderful day, sir.

"Buzzing off," now.

 • Reply • Share ›

gweiss27 ➚ Erutan409 • a year ago

Man, you just really do not get it. For some reason you keep pointing your finger back at me when in reality, you are the cause of the issue. Again, I have right to post whatever I want, if the author wants to shoot back, that's fine. If YOU had addressed the issue correctly, this would have been completely solved. Here, I will do it for you:

Hey gweiss27, Wow. Sounds like you are super frustrated! Believe me, I've been there! Nothing worse than spending hours going round and round in coding circles on some stupid minute point, but coders all know it happens. I thought the article was pretty good, but maybe you could share your solution so that others don't have to endure the same frustration. I'm sure the author didn't intend to omit or mislead anyone with his post. I mean, he took the time to post a free article to help people. But people are human, what can you do? Anyway, I'm sure your solution would be valuable to other readers on this thread. Happy Coding!

I rest my case, look in the mirror first before you start jumping in with unsolicited feedback just to bust on other people when it isn't even your place in the first place. This is your fault, NOT mine.

1 • Reply • Share ›

Erutan409 ➚ gweiss27 • a year ago

Actually, that was pretty good. Good job, man. Much more constructive!

 • Reply • Share ›

gweiss27 ➚ Erutan409 • a year ago

ok fair enough! We both made good points! Truce! :)

1 • Reply • Share ›

Erutan409 ➚ gweiss27 • a year ago

Truce.

 • Reply • Share ›

Paul Bakker ↗ Erutan409 • a year ago

gweiss27, you get some free information, can't get it to work on your mac and start to throw your frustrations to the author instead of being constructive and helpful. You don't help him or the rest of the people who come across this article but start whining away about how bad this all has been for you. You're behaving like a spoiled child. You could have contacted the author first and ask him a question instead of hitting on him straight away.

^ | v • Reply • Share ›

Manor Minors ↗ Paul Bakker • 10 months ago

I have big list of question? How do I contact Author? Anybody?

^ | v • Reply • Share ›

Guest ↗ Erutan409 • a year ago

ok fair enough! We both made good points! Truce! :)

^ | v • Reply • Share ›

Malcolm Mathis • 4 years ago

PHP is most likely the terminology most acquainted to Nettuts+ visitors. That being said, PHP, although developed for the web, is probably not the best language to use when operating in a REST way, as it manages PUT demands in a absolutely different fashion.

2 ^ | v • Reply • Share ›

mattsah • 5 years ago

Perhaps I'm missing something, but the creation of a extensively RESTful API is in many ways additional work to the creation of a web interface since HTML Forms do not support the PUT/DELETE methods, or anything aside from GET or POST, hence why we tend to get one or the other with parameters like ?action=delete

It's nice for creating a web API, but it only relates to the architecture of served web pages to a certain degree for this reason.

2 ^ | v • Reply • Share ›

jjonesdesign ↗ mattsah • a year ago

As somewhat mentioned in the article, standard web forms are still acceptable in application. One must use ajax to process the request of the form, which can be

Follow Envato Tuts+