POST, GET E REQUEST

```
<form action="/action_page.php" method="get">
  <label for="fname">First name:</label>
  <input type="text" id="fname" name="fname"><br><br>
  <label for="lname">Last name:</label>
  <input type="text" id="lname" name="lname"><br><br>
  <input type="submit" value="Submit">
</form>
```

HTTP Request Methods

What is HTTP?

The Hypertext Transfer Protocol (HTTP) is designed to enable communications between clients and servers.

HTTP works as a request-response protocol between a client and server.

Example: A client (browser) sends an HTTP request to the server; then the server returns a response to the client. The response contains status information about the request and may also contain the requested content.

HTTP Methods

- **GET**
- **POST**
- **PUT**
- **HEAD**
- **DELETE**
- **PATCH**
- **OPTIONS**

The two most common HTTP methods are: GET and POST.

The GET Method

GET is used to request data from a specified resource.

GET is one of the most common HTTP methods.

Note that the query string (name/value pairs) is sent in the URL of a GET request:

/test/demo_form.php?name1=value1&name2=value2

Some other notes on GET requests:

- GET requests can be cached
- GET requests remain in the browser history
- GET requests can be bookmarked
- GET requests should never be used when dealing with sensitive data
- GET requests have length restrictions
- GET requests are only used to request data (not modify)

The POST Method

POST is used to send data to a server to create/update a resource.

The data sent to the server with POST is stored in the request body of the HTTP request:

POST /test/demo_form.php HTTP/1.1
Host: w3schools.com
name1=value1&name2=value2

POST is one of the most common HTTP methods.

Some other notes on POST requests:

- POST requests are never cached
- POST requests do not remain in the browser history
- POST requests cannot be bookmarked
- POST requests have no restrictions on data length

The PUT Method

PUT is used to send data to a server to create/update a resource.

The difference between POST and PUT is that PUT requests are idempotent. That is, calling the same PUT request multiple times will always produce the same result. In contrast, calling a POST request repeatedly have side effects of creating the same resource multiple times.

The HEAD Method

HEAD is almost identical to GET, but without the response body.

In other words, if GET /users returns a list of users, then HEAD /users will make the same request but will not return the list of users.

HEAD requests are useful for checking what a GET request will return before actually making a GET request - like before downloading a large file or response body.

The DELETE Method

The DELETE method deletes the specified resource.

The OPTIONS Method

The OPTIONS method describes the communication options for the target resource.

Compare GET vs. POST

The following table compares the two HTTP methods: GET and POST.

	GET	POST
BACK button/Reload	Harmless	Data will be re-submitted (the browser should alert the user that the data are about to be re-submitted)
Bookmarked	Can be bookmarked	Cannot be bookmarked
Cached	Can be cached	Not cached
Encoding type	application/x-www-form-urlencoded	application/x-www-form-urlencoded or multipart/form-data. Use multipart encoding for binary data
History	Parameters remain in browser history	Parameters are not saved in browser history
Restrictions on data length	Yes, when sending data, the GET method adds the data to the URL; and the length of a URL is limited (maximum URL length is 2048 characters)	No restrictions
Restrictions on data type	Only ASCII characters allowed	No restrictions. Binary data is also allowed
Security	GET is less secure compared to POST because data sent is part of the URL Never use GET when sending passwords or other sensitive information!	POST is a little safer than GET because the parameters are not stored in browser history or in web server logs
Visibility	Data is visible to everyone in the URL	Data is not displayed in the URL

https://www.w3schools.com/tags/ref_httpmethods.asp

PHP - GET & POST Methods

There are two ways the browser client can send information to the web server.

- The GET Method
- The POST Method

Before the browser sends the information, it encodes it using a scheme called URL encoding. In this scheme, name/value pairs are joined with equal signs and different pairs are separated by the ampersand.

```
name1=value1&name2=value2&name3=value3
```

Spaces are removed and replaced with the + character and any other nonalphanumeric characters are replaced with a hexadecimal values. After the information is encoded it is sent to the server.

The GET Method

The GET method sends the encoded user information appended to the page request. The page and the encoded information are separated by the **?** character.

```
http://www.test.com/index.htm?name1=value1&name2=value2
```

- The GET method produces a long string that appears in your server logs, in the browser's Location: box.
- The GET method is restricted to send upto 1024 characters only.
- Never use GET method if you have password or other sensitive information to be sent to the server.
- GET can't be used to send binary data, like images or word documents, to the server.
- The data sent by GET method can be accessed using QUERY_STRING environment variable.
- The PHP provides **$_GET** associative array to access all the sent information using GET method.

Try out following example by putting the source code in test.php script.

```
<?php
   if( $_GET["name"] || $_GET["age"] ) {
      echo "Welcome ". $_GET['name']. "<br />";
      echo "You are ". $_GET['age']. " years old.";

      exit();
   }
?>
<html>
   <body>

      <form action = "<?php $_PHP_SELF ?>" method = "GET">
         Name: <input type = "text" name = "name" />
         Age: <input type = "text" name = "age" />
```

```
         <input type = "submit" />
      </form>

   </body>
</html>
```

It will produce the following result −

The POST Method

The POST method transfers information via HTTP headers. The information is encoded as described in case of GET method and put into a header called QUERY_STRING.

- The POST method does not have any restriction on data size to be sent.
- The POST method can be used to send ASCII as well as binary data.
- The data sent by POST method goes through HTTP header so security depends on HTTP protocol. By using Secure HTTP you can make sure that your information is secure.
- The PHP provides **$_POST** associative array to access all the sent information using POST method.

Try out following example by putting the source code in test.php script.

```
<?php
   if( $_POST["name"] || $_POST["age"] ) {
      if (preg_match("/[^A-Za-z'-]/",$_POST['name'] )) {
         die ("invalid name and name should be alpha");
      }
      echo "Welcome ". $_POST['name']. "<br />";
      echo "You are ". $_POST['age']. " years old.";

      exit();
   }
?>
<html>
   <body>

      <form action = "<?php $_PHP_SELF ?>" method = "POST">
         Name: <input type = "text" name = "name" />
         Age: <input type = "text" name = "age" />
         <input type = "submit" />
      </form>

   </body>
</html>
```

It will produce the following result −

The $_REQUEST variable

The PHP $_REQUEST variable contains the contents of both $_GET, $_POST, and $_COOKIE. We will discuss $_COOKIE variable when we will explain about cookies.

The PHP $_REQUEST variable can be used to get the result from form data sent with both the GET and POST methods.

Try out following example by putting the source code in test.php script.

```
<?php
   if( $_REQUEST["name"] || $_REQUEST["age"] ) {
      echo "Welcome ". $_REQUEST['name']. "<br />";
      echo "You are ". $_REQUEST['age']. " years old.";
      exit();
   }
?>
<html>
   <body>

      <form action = "<?php $_PHP_SELF ?>" method = "POST">
         Name: <input type = "text" name = "name" />
         Age: <input type = "text" name = "age" />
         <input type = "submit" />
      </form>

   </body>
</html>
```

Here $_PHP_SELF variable contains the name of self script in which it is being called.

It will produce the following result −

https://www.tutorialspoint.com/php/php_get_post.htm

PHP GET/POST request

last modified July 9, 2020

PHP GET/POST request tutorial shows how to generate and process GET and POST requests in PHP. We use plain PHP and Symfony, Slim, and Laravel frameworks.

HTTP

The Hypertext Transfer Protocol () is an application protocol for distributed, collaborative, hypermedia information systems. HTTP protocol is the foundation of data communication for the World Wide Web.

HTTP GET

The HTTP GET method requests a representation of the specified resource.

GET requests:

- should only be used to request a resource
- parameters are displayed in the URL
- can be cached
- remain in the browser history
- can be bookmarked
- should never be used when dealing with sensitive data
- have length limits

HTTP POST

The HTTP POST method sends data to the server. It is often used when uploading a file or when submitting a completed web form.

POST requests:

- should be used to create a resource
- parameters are not displayed in the URL
- are never cached
- do not remain in the browser history
- cannot be bookmarked
- can be used when dealing with sensitive data
- have no length limits

PHP $_GET and $_POST

PHP provides the `$_GET` and `$_POST` superglobals. The `$_GET` is an associative array of variables passed to the current script via the URL parameters (query string). The `$_POST` is an associative array of variables passed to the current script via the HTTP POST method when using `application/x-www-form-urlencoded` or `multipart/form-data` as the HTTP Content-Type in the request.

PHP GET request

In the following example, we generate a GET request with curl tool and process the request in plain PHP.

get_req.php

```
<?php

$name = $_GET['name'];

if ($name == null) {
    $name = 'guest';
}

$message = $_GET['message'];

if ($message == null) {
    $message = 'hello there';
}

echo "$name says: $message";
```

The example retrieves the `name` and `message` parameters from the `$_GET` variable.

```
$ php -S localhost:8000 get_req.php
```

We start the server.

```
$ curl 'localhost:8000/?name=Lucia&message=Cau'
Lucia says: Cau
$ curl 'localhost:8000/?name=Lucia'
Lucia says: hello there
```

We send two GET requests with curl.

PHP POST request

In the following example, we generate a POST request with curl tool and process the request in plain PHP.

post_req.php

```
<?php

$name = $_POST['name'];

if ($name == null) {
    $name = 'guest';
```

```
}

$message = $_POST['message'];

if ($message == null) {
    $message = 'hello there';
}


echo "$name says: $message";
```

The example retrieves the `name` and `message` parameters from the `$_POST` variable.

```
$ php -S localhost:8000 post_req.php
```

We start the server.

```
$ curl -d "name=Lucia&message=Cau" localhost:8000
Lucia says: Cau
```

We send a POST request with curl.

PHP send GET request with Symfony HttpClient

Symfony provides the `HttpClient` component which enables us to create HTTP requests in PHP.

```
$ composer req symfony/http-client
```

We install the `symfony/http-client` component.

send_get_req.php

```
<?php

require('vendor/autoload.php');

use Symfony\Component\HttpClient\HttpClient;

$httpClient = HttpClient::create();
$response = $httpClient->request('GET', 'http://localhost:8000', [
    'query' => [
        'name' => 'Lucia',
        'message' => 'Cau',
    ]
]);

$content = $response->getContent();
echo $content . "\n";
```

The example sends a GET request with two query parameters to `localhost:8000/get_request.php`.

```
$ php -S localhost:8000 get_req.php
```

We start the server.

```
$ php send_get_req.php
Lucia says: Cau
```

We run the `send_get_req.php` script.

PHP send POST request with Symfony HttpClient

In the following example, we send a POST request with Symfony HttpClient.

send_post_req.php

```
<?php

require('vendor/autoload.php');

use Symfony\Component\HttpClient\HttpClient;

$httpClient = HttpClient::create();
$response = $httpClient->request('POST', 'http://localhost:8000', [
    'body' => [
        'name' => 'Lucia',
        'message' => 'Cau',
    ]
]);

$content = $response->getContent();
echo $content . "\n";
```

The example sends a POST request with two parameters to `localhost:8000/post_req.php`.

```
$ php -S localhost:8000 post_req.php
```

We start the server.

```
$ php send_post_req.php
Lucia says: Cau
```

We run the `send_post_req.php` script.

PHP GET request in Symfony

In the following example, we process a GET request in a Symfony application.

```
$ symfony new symreq
$ cd symreq
```

A new application is created.

```
$ composer req annot
$ composer req maker --dev
```

We install the `annot` and `maker` components.

```
$ php bin/console make:controller HomeController
```

We create a new controller.

src/Controller/HomeController.php

```
<?php

namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\Routing\Annotation\Route;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\HttpFoundation\Request;

class HomeController extends AbstractController
{
    /**
     * @Route("/", name="home", methods={"GET"})
     */
    public function index(Request $request): Response
    {
        $name = $request->query->get('name', 'guest');
        $message = $request->query->get('message', 'hello there');

        $output = "$name says: $message";

        return new Response($output, Response::HTTP_OK,
            ['content-type' => 'text/plain']);
    }
}
```

Inside the `HomeController's index` method, we get the query parameters and create a response.

```
$name = $request->query->get('name', 'guest');
```

The GET parameter is retrieved with `$request->query->get`. The second parameter of the method is a default value which is used when no value was retrieved.

```
$ symfony serve
```

We start the server.

```
$ curl 'localhost:8000/?name=Lucia&message=Cau'
Lucia says: Cau
```

We generate a GET request with curl.

PHP POST request in Symfony

In the following example, we process a POST request in a Symfony application.

src/Controller/HomeController.php

```
<?php

namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\Routing\Annotation\Route;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\HttpFoundation\Request;
```

```
class HomeController extends AbstractController
{
    /**
     * @Route("/", name="home", methods={"POST"})
     */
    public function index(Request $request): Response
    {
        $name = $request->request->get('name', 'guest');
        $message = $request->request->get('message', 'hello there');

        $output = "$name says: $message";

        return new Response($output, Response::HTTP_OK,
            ['content-type' => 'text/plain']);
    }
}
```

We change the controller to process the POST request.

```
$name = $request->request->get('name', 'guest');
```

The POST parameter is retrieved with `$request->request->get`. The second parameter of the method is a default value which is used when no value was retrieved.

```
$ symfony serve
```

We start the server.

```
$ curl -d "name=Lucia" localhost:8000
Lucia says: hello there
```

We generate a POST request with curl.

PHP GET request in Slim

In the following example, we are going to process a GET request in the Slim framework.

```
$ composer req slim/slim
$ composer req slim/psr7
$ composer req slim/http
```

We install `slim/slim`, `slim/psr7`, and `slim/http` packages.

public/index.php

```
<?php
use Psr\Http\Message\ResponseInterface as Response;
use Psr\Http\Message\ServerRequestInterface as Request;
use Slim\Factory\AppFactory;

require __DIR__ . '/../vendor/autoload.php';

$app = AppFactory::create();

$app->get('/', function (Request $request, Response $response): Response {

    $name = $request->getQueryParam('name', 'guest');
    $message = $request->getQueryParam('message', 'hello there');
    $output = "$name says $message";

    $response->getBody()->write($output);
```

```
    return $response;
});

$app->run();
```

We get the parameters and return a response in Slim.

```
$name = $request->getQueryParam('name', 'guest');
```

The query parameter is retrieved with `getQueryParam`; the second parameter is the default value.

```
$response->getBody()->write($output);
```

We write the output to the response body with `write`.

```
$ php -S localhost:8000 -t public
```

We start the server.

```
$ curl 'localhost:8000/?name=Lucia&message=Cau'
Lucia says: Cau
```

We generate a GET request with curl.

PHP POST request in Slim

In the following example, we are going to process a POST request in the Slim framework.

public/index.php

```
<?php
use Psr\Http\Message\ResponseInterface as Response;
use Psr\Http\Message\ServerRequestInterface as Request;
use Slim\Factory\AppFactory;

require __DIR__ . '/../vendor/autoload.php';

$app = AppFactory::create();

$app->post('/', function (Request $request, Response $response): Response {

    $data = $request->getParsedBody();

    $name = $data['name'];
    $message = $data['message'];

    if ($name == null) {
        $name = 'guest';
    }

    if ($message == null) {
        $message = 'hello there';
    }

    $output = "$name says: $message";

    $response->getBody()->write($output);
    return $response;
});
```

```
$app->run();
```

We get the POST parameters and return a response in Slim.

```
$data = $request->getParsedBody();
```

The POST parameters are retrieved with `getParsedBody`.

```
$ php -S localhost:8000 -t public
```

We start the server.

```
$ curl -d "name=Lucia" localhost:8000
Lucia says: hello there
```

We generate a POST request with curl.

PHP GET request in Laravel

In the following example, we process a GET request in Laravel.

```
$ laravel new larareq
$ cd larareq
```

We create a new Laravel application.

routes/web.php

```
<?php

use Illuminate\Support\Facades\Route;
use Illuminate\Http\Request;

Route::get('/', function (Request $request) {

    $name = $request->query('name', 'guest');
    $message = $request->query('message', 'hello there');
    $output = "$name says $message";

    return $output;
});
```

We get the GET parameters and create a response.

```
$ php artisan serve
```

We start the server.

```
$ curl 'localhost:8000/?name=Lucia&message=Cau'
Lucia says Cau
```

We send a GET request with curl.

PHP POST request in Laravel

In the following example, we send a POST request from an HTML form.

resources/views/home.blade.php

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <title>Home page</title>
        <style>
            .alert { color: red}
        </style>
    </head>
    <body>
        @if ($errors->any())
        <div class="alert">
            <ul>
                @foreach ($errors->all() as $error)
                    <li>{{ $error }}</li>
                @endforeach
            </ul>
        </div>
        @endif
         <form action="process_form" method="post">
            @csrf
            <label for="name">Name</label> <input id="name"
                value="{{old('name')}}"type="text" name="name">
            <label for="message">Message</label> <input id="message"
                value="{{old('message')}}" type="text" name="message">
            <button type="submit">Submit</button>
        </form>
    </body>
</html>
```

We have a POST form in a Blade template. Laravel requires CSRF protection for POST requests. We enable CSRF protection with `@csrf`.

routes/web.php

```
<?php

use Illuminate\Support\Facades\Route;
use Illuminate\Http\Request;

Route::get('/', function () {
    return view('home');
});


Route::post('/process_form', function (Request $request) {

    $request->validate([
        'name'  =>  'required|min:2',
        'message' =>  'required|min:3'
    ]);

    $name = $request->input('name');
    $message = $request->input('message');

    $output = "$name says: $message";

    return $output;
});
```

We validate and retrieve the POST parameters and send them in the response. This example should be tested in a browser.

In this tutorial, we have worked with GET and POST requests in plain PHP, Symfony, Slim, and Laravel.

https://zetcode.com/php/getpostrequest/

$_REQUEST

$_REQUEST — Variáveis de requisição HTTP

Descrição ¶

Um array associativo que por padrão contém informações de *$_GET*, *$_POST* e *$_COOKIE*.

Changelog ¶

Versão	Descrição
5.3.0	Introduzida a request_order. Esta diretiva afeta o conteúdo de *$_REQUEST*.
4.3.0	Informação da *$_FILES* foi removida de *$_REQUEST*.
4.1.0	Introduzida a *$_REQUEST*.

Notas ¶

Nota:

Esta é uma 'superglobal', ou variável global automática. Isto significa que ela está disponível em todos escopos pelo script. Não há necessidade de fazer **global $variable;** para acessá-la dentro de uma função ou método.

Nota:

Quando executando em linha de comando , esta *não* incluirá as entradas argv e argc; estas estão presentes no array *$_SERVER*.

Nota:

As variáveis em *$_REQUEST* são providas para o script via mecanismos de entradas GET, POST, e COOKIE e portando poderia ser modificadas por um usuário remoto e não podem ser confiadas. A presença e ordem das variáveis listadas neste array é definido de acordo com a diretiva de configuração do PHP variables_order.

Veja Também ¶

- **import_request_variables()**
- Manuseamento de variáveis externas
- A extensão filter

☐ add a note

User Contributed Notes 6 notes

up

down

179

strata_ranger at hotmail dot com ¶

13 years ago

Don't forget, because $_REQUEST is a different variable than $_GET and $_POST, it is treated as such in PHP -- modifying $_GET or $_POST elements at runtime will not affect the ellements in $_REQUEST, nor vice versa.

e.g:

```
<?php

$_GET['foo'] = 'a';
$_POST['bar'] = 'b';
var_dump($_GET); // Element 'foo' is string(1) "a"
var_dump($_POST); // Element 'bar' is string(1) "b"
var_dump($_REQUEST); // Does not contain elements 'foo' or 'bar'

?>
```

If you want to evaluate $_GET and $_POST variables by a single token without including $_COOKIE in the mix, use $_SERVER['REQUEST_METHOD'] to identify the method used and set up a switch block accordingly, e.g:

```
<?php

switch($_SERVER['REQUEST_METHOD'])
{
case 'GET': $the_request = &$_GET; break;
case 'POST': $the_request = &$_POST; break;
.
. // Etc.
.
default:
}
?>
```

https://www.php.net/manual/pt_BR/reserved.variables.request.php

$_GET, $_POST, and $_REQUEST

$_GET Variable

What is it?

The $_GET variable is used to get data from a form that is written in HTML. Also in the url $_GET variable will display the data that was taken by the $_GET variable. For example using the second example on this page it will display in the url as ?name='it will equal to what text was entered in the text box'. $_GET has limits on the amount of information that can be sent.

How to use it

Before you can use the the $_GET variable you have to have a form in html that has the method equal to GET. Then in the php, you can use the $_GET variable to get the data that you wanted. The $_GET syntax is ($_GET['name of the form field goes here']).

Examples

The $_GET Syntax

```
<?php
//Displays the data that was received from the input box named name in the form
        ($_GET['form name goes here'])
?>
```

HTML form with $_GET

```
<?php
// It will display the data that it was received from the form called name
         echo ($_GET['name']);
 ?>

 //This is the html form that creates the input box and submit button
 //The method for the form is in the line below
 <form action="name of the php file that has the ($_GET[]) variable in it"
method="GET">
        Name:<input type="text" name="name">

        <input type="submit" value="Submit">
 </form>
```

$_POST

What is it?

The $_POST variable is also used to collect data from forms, but the $_POST is slightly different because in $_GET it displayed the data in the url and $_POST does not. The data that $_POST gets, is invisible to others and the amount that can be sent is not limited besides the 8MB max size.

How to use it?

Before you can use the the $_POST variable you have to have a form in html that has the method equal to POST. Then in the php, you can use the $_POST variable to get the data that you wanted. The $_POST syntax is ($_POST['name of the form field goes here']).

Examples

The $_POST syntax

```
<?php
//The $_POST gets the data from the form
        ($_POST['form name goes here'])
?>
```

The html form with $_POST

```
<?php
//Displays the data that was received from the input box named name in the form
 echo ($_POST['name']);
?>

//This is the html form that creates the input box and submit button
//The method for the form is in the line below
<form action="test.php" method=POST>
        Name:<br><input type="text" name="name"><br>
        <input type="submit" value="Submit">
 </form>
```

$_REQUEST

What is it?

The $_REQUEST variable is a variable with the contents of $_GET and $_POST and $_COOKIE variables.

How to use it?

Before you can use the the $_REQUEST variable you have to have a form in html that has the method equal to GET and POST. Then in the php, you can use the $_REQUEST variable to get the data that you wanted. Depending on what you wrote for the method in the form and using $_REQUEST in the php, $_REQUEST will use $_Get if GET is written for the method and

$_REQUEST will use $POST if POST is written in the method. The $_REQUEST syntax is ($_REQUEST['name of the form field goes here']).

Examples

The $_REQUEST syntax

```
<?php
//this will stay the same but the method in the form will change to you
preference from GET or POST
        ($_REQUEST['form name goes here'])
?>
```

Using GET for the method

```
<?php
//Displays the data that was received from the input box named name in the form
        echo ($_REQUEST['name']);
 ?>

 //This is the html form that creates the input box and submit button
 //The method for the form is in the line below
 <form action="test.php" method=GET>
        Name:<br><input type="text" name="name"><br>
        <input type="submit" value="Submit">
 </form>
```

Using POST for the method

```
<?php
//Displays the data that was received from the input box named name in the form
        echo ($_REQUEST['name']);
 ?>

 //This is the html form that creates the input box and submit button
 //The method for the form is in the line below
 <form action="test.php" method=POST>
        Name:<br><input type="text" name="name"><br>
        <input type="submit" value="Submit">
 </form>
```

http://www.shodor.org/~kevink/phpTutorial/nileshc_getreqpost.php