Postman Quick Reference Guide

SIMPLE SOLUTIONS
TO COMMON PROBLEMS

by Valentin Despa

Postman Quick Reference Guide Documentation
Release Preview - February 2018

Valentin Despa

Feb 15, 2018

Contents:

Cheatsheet

1.1 Postman Cheatsheet

Thank you for downloading this cheat sheet. This guide refers to the Postman App, not the Chrome extension. Please report any problems with it.

Postman Cheat Sheet is based on the official Postman documentation and own experience.

For a detailed documentation on each feature, check out https://www.getpostman.com/docs.

1.1.1 Variables

Code snippets for working with variables in scripts (pre-request, tests).

Setting variables

Global variable

```
pm.globals.set('myVariable', MY_VALUE);
```

Collection variable

> Can only be set from Postman (at this moment).

Environment variable

```
pm.environment.set('myVariable', MY_VALUE);
```

Data variable

> Can only be set from a CSV or a JSON file.

Local variable

```
var myVar = MY_VALUE;
```

Examples:

```
pm.globals.set('name', 'John Doe');
pm.environment.set('age', 29);
```

Getting variables

Request builder

Syntax: `{{myVariable}}`

Request URL: `http://{{domain}}/users/{{userId}}`

Headers (key:value): X-{{myHeaderName}}:foo

Body: `{"id":  "{{userId}}", "name":  "John Doe"}`

Depending on the matching scope

```
pm.variables.get('myVariable');
```

Global variable

```
pm.globals.get('myVariable');
```

Collection variable

Can only be used in the Request Builder (at this moment). Syntax: `{{myVariable}}`

Environment variable

```
pm.environment.get('myVariable');
```

Data variable

```
pm.iterationData.get('myVariable);
```

Local variable

```
myVar
```

Removing variables

Global variable

Remove one variable

```
pm.globals.unset('myVariable');
```

Remove ALL global variables

```
pm.globals.clear();
```

Environment variable

Remove one variable

```
pm.environment.unset("myVariable");
```

Remove ALL environment variables

```
pm.environment.clear();
```

DYNAMIC VARIABLES

Experimental feature. Can only be used in request builder. Only one value is generated per request.

All dynamic variables can be combined with strings, in order to generate dynamic / unique data.

Example JSON body:

```
{"name": "John Doe", "email": "john.doe.{{$timestamp}}@example.com"}
```

`{{$guid}}` - global unique identifier.

Example output: `d96d398a-b655-4638-a6e5-40c0dc282fb7`

`{{$timestamp}}` - current timestamp.

Example output: *1507370977*'

`{{$randomInt}}` - random integer between 0 and 1000.

Example output: `567`

LOGGING / DEBUGGING VARIABLES

Open Postman Console and use *console.log* in your test or pre-request script.

Example:

```
var myVar = pm.globals.get("myVar");
console.log(myVar);
```

1.1.2 Assertions

Note: You need to add any of the assertions inside a `pm.test` callback.

Example:

```
pm.test("Your test name", function () {
    var jsonData = pm.response.json();
    pm.expect(jsonData.value).to.eql(100);
});
```

Status code

Check if status code is 200:

```
pm.response.to.have.status(200);
```

Checking multiple status codes:

```
pm.expect(pm.response.code).to.be.oneOf([201,202]);
```

Response time

Response time below 100ms:

```
pm.expect(pm.response.responseTime).to.be.below(9);
```

Headers

Header exists:

```
pm.response.to.have.header(X-Cache');
```

Header has value:

```
pm.expect(pm.response.headers.get('X-Cache')).to.eql('HIT');
```

Cookies

Cookie exists:

```
pm.expect(pm.cookies.has('sessionId')).to.be.true;
```

Cookie has value:

```
pm.expect(pm.cookies.get('sessionId')).to.eql('ad3se3ss8sg7sg3');
```

Body

Any content type / HTML responses

Exact body match:

```
pm.response.to.have.body("OK");
pm.response.to.have.body('{"success"=true});
```

Partial body match / body contains:

```
pm.expect(pm.response.text()).to.include('Order placed.);
```

JSON responses

Parse body (need for all assertions):

```
var jsonData = pm.response.json();
```

Simple value check:

```
pm.expect(jsonData.age).to.eql(30);
pm.expect(jsonData.name).to.eql('John);
```

Nested value check:

```
pm.expect(jsonData.products.0.category).to.eql('Detergent');
```

XML responses

Convert XML body to JSON:

```
var jsonData = xml2Json(responseBody);
```

Note: see assertions for JSON responses.

1.1.3 Postman Sandbox

pm

this is the object containing the script that is running, can access variables and has access to a read-only copy of the request or response.

pm.sendRequest

Allows to send simple HTTP(S) GET requests from tests and pre-request scripts. Example:

```
pm.sendRequest('http://example.com', function (err, res) {
    console.log(err ? err : res.json());
});
```

Full-option HTTP(S) request:

```
const postRequest = {
    url: 'http://example.com', method: 'POST',
    header: 'X-Foo:foo',
    body: {
        mode: 'raw',
        raw: JSON.stringify({ name: 'John' })
    }
};
pm.sendRequest(postRequest, function (err, res) {
    console.log(err ? err : res.json());
});
```

1.1.4 Postman Echo

Helper API for testing requests. Read more at: https://docs.postman-echo.com.

Get Current UTC time in pre-request script

```
pm.sendRequest('https://postman-echo.com/time/now', function (err, res) {
    if (err) { console.log(err); }
    else {
        var currentTime = res.stream.toString();
        console.log(currentTime);
        pm.environment.set("currentTime", currentTime);
    }
});
```

1.1.5 Workflows

Only work with automated collection runs such as with the Collection Runner or Newman.

Set which will be the next request to be executed

```
postman.setNextRequest("Request name");
```

Stop executing requests / stop the collection run

```
postman.setNextRequest(null);
```

Simple solutions to common problems

2.1 Request creation

2.1.1 I have an environment variable as {{url}}. Can I use it inside a script (like pm.sendRequest)?

The following syntax will not work while writing scripts:

```
pm.sendRequest({{url}}/mediaitem/)
```

You are inside a script, so you need to use the pm.* API to get to that variable. The syntax {{url}} works only inside the request builder, not in scripts.

Example:

```
var requestUrl = pm.environment.get("url") + "/mediaitem/");

pm.sendRequest(requestUrl, function (err, res) {
    // do stuff
});
```

2.1.2 How to use pre-request script to pass dynamic data in the request body?

In the pre-request script you can simply create a JavaScript object, set the desired values and save it as a variable ()

For example if you want to send a request body that looks like:

```
{
    "firstName": "First Name",
    "lastName": "Last Name",
    "email": "test@example.com"
}
```

You can do the following in the pre-request script:

```
// Define a new object
var user = {
    "firstName": "First Name",
    "lastName": "Last Name",
    "email": "test@example.com"
}


// Save the object as a variable.
// JSON.stringify will serialize the object so that Postman can save it
pm.globals.set('user', JSON.stringify(user));
```

In the request body you can simply use `{{user}}`. This also works just as well for nested objects:

```
{
    "user": {{user}}
    "address": {
        "street": "Foo"
        "number": "2"
        "city": "Bar"
    }
}
```

2.1.3 How to generate random data?

Option 1 Using existing Postman random generators

If you need to create an unique string (with every request) and pass it in the request body, in the example below there will be generated an unique GroupName everytime the request is executed.

You can use the variable `{{$guid}}` - this is automatically generated by Postman. Or you can use the current timestamp, `{{$timestamp}}`:

```
{
    "GroupName":"GroupName_{{$guid}}",
    "Description": "Example_API_Admin-Group_Description"
}
```

This will generate something like:

```
{
    "GroupName":"GroupName_0542bd53-f030-4e3b-b7bc-d496e71d16a0",
    "Description": "Example_API_Admin-Group_Description"
}
```

The disadvantage of this method is that you cannot use these special variables in a pre-request script or test. Additionally they will be only generated once per request, so using `{{$guid}}` more than once will generate the same data in a request.

Option 2 Using existing JavaScript random generators

TODO

2.1.4 How to trigger another another request from pre-request script?

Option 1 You can trigger another request in the collection from the pre-request script using `postman.setNextRequest`.

That can be done with:

```
postman.setNextRequest('Your request name as saved in Postman');
```

The difficulty is returning to the request that initiated the call. Additionally you need to make sure you do not create endless loops.

Option 2 Another possibility is making an HTTP call from the pre-request script to fetch any data you might need.

Below I am fetching a name from a remote API and setting it as a variable for use in the actual request that will execute right after the pre-request script completed:

```
var options = { method: 'GET',
  url: 'http://www.mocky.io/v2/5a849eee300000580069b022'
};

pm.sendRequest(options, function (error, response) {
    if (error) throw new Error(error);
    var jsonData = response.json();
    pm.globals.set('name', 'jsonData.name');
});
```

Tip You can generate such requests by using the "Code" generator button right below the Save button, once you have a request that works. There you can Select NodeJS > Request and the syntax generated is very similar to what Postman expects.

You can import this example in Postman by using this link: https://www.getpostman.com/collections/5a61c265d4a7bbd8b303

2.1.5 How to pass arrays and objects between requests?

Assuming your response is in JSON format, You can extract data from the response by using

```
var jsonData = pm.response.json();
```

After this you can set the whole response (or just a subset like this):

```
pm.environment.set('myData', JSON.stringify(jsonData));
```

You need to use JSON.stringify() before saving objects / arrays to a Postman variable. Otherwise it will not work.

In the next request where you want to retrieve the data, just use:

- `{myData}}` if you are inside the request builder
- `var myData = JSON.parse(pm.environment.get('myData'));`

Using JSON.stringify and JSON.parse methods is not needed if the values are strings or integers or booleans.

JSON.stringify() converts a value to a JSON string while JSON.parse() method parses a JSON string, creating the value described by the string.

2.1.6 How to read external files?

If you have some information saved on a file locally on your computer, you might want to access this information with Postman.

Unfortunately this is not really possible. There is a way to read a data file in JSON or CSV format, which allows you to make some variables dynamic. These variables are called data variables and are mostly used for testing different iterations on a specific request or collection.

The alternative would be to start a local server to serve that file and to get it in Postman with a GET request.

2.2 Assertions

Assertions in Postman are based on the capabilities of the Chai Assertion Library. You can read an extensive documentation on Chai by visiting http://chaijs.com/api/bdd/

2.2.1 How find object in array by property value?

Given the following response:

```
{
    "companyId": 10101,
    "regionId": 36554,
    "filters": [
        {
            "id": 101,
            "name": "VENDOR",
            "isAllowed": false
        },
        {
            "id": 102,
            "name": "COUNTRY",
            "isAllowed": true
        },
        {
            "id": 103,
            "name": "MANUFACTURER",
            "isAllowed": false
        }
    ]
}
```

Assert that the property isAllowed is true for the COUNTRY filter.

```
pm.test("Check the country filter is allowed", function () {
    // Parse response body
    var jsonData = pm.response.json();

    // Find the array index for the COUNTRY
    var countryFilterIndex = jsonData.filters.map(
            function(filter) {
                return filter.name; // <-- HERE is the name of the property
            }
        ).indexOf('COUNTRY'); // <-- HERE is the value we are searching for
```

```
    // Get the country filter object by using the index calculated above
    var countryFilter = jsonData.filters[countryFilterIndex];

    // Check that the country filter exists
    pm.expect(countryFilter).to.exist;

    // Check that the country filter is allowed
    pm.expect(countryFilter.isAllowed).to.be.true;
});
```

2.2.2 How to compare value of a response with an already defined variable?

Lets presume you have a value from a previous response (or other source) that is saved to a variable.

```
// Getting values from response
var jsonData = pm.response.json();
var username = jsonData.userName;

// Saving the value for later use
pm.globals.set("username", username);
```

How do you compare that variable with values from another API response?

In order to access the variable in the script, you need to use a special method, basically the companion of setting a variable. Curly brackets will not work in this case:

```
pm.test("Your test name", function () {
    var jsonData = pm.response.json();
    pm.expect(jsonData.value).to.eql(pm.globals.get("username"));
});
```

2.2.3 How to compare value of a response against multiple valid values?

Given the following API response:

```
{
    "SiteId": "aaa-ccc-xxx",
    "ACL": [
        {
            "GroupId": "xxx-xxx-xx-xxx-xx",
            "TargetType": "Subscriber"
        }
    ]
}
```

You want to check that `TargetType` is *Subscriber* or *Customer*.

The assertion can look like:

```
pm.test("Should be subscriber or customer", function () {
    var jsonData = pm.response.json();
    pm.expect(.TargetType).to.be.oneOf(["Subscriber", "Customer"]);
});
```

where:

- jsonData.ACL[0] is the first element of the ACL array
- to.be.oneOf allows an array of possible valid values

2.2.4 How to parse a HTML response to extract a specific value?

Presumed you want to get the _csrf hidden field value for assertions or later use from the response below:

```html
<form name="loginForm" action="/login" method="POST">
        <input type="hidden" name="_csrf" value="a0e2d230-9d3e-4066-97ce-f1c3cdc37915
→" />
        <ul>
            <li>
                <label for="username">Username:</label>
                <input required type="text" id="username" name="username" />
            </li>
            <li>
                <label for="password">Password:</label>
                <input required type="password" id="password" name="password" />
            </li>
            <li>
                <input name="submit" type="submit" value="Login" />
            </li>
        </ul>
</form>
```

To parse and retrive the value, we will use the cherrio JavaScript library:

```javascript
// Parse HTML and get the CSRF token
responseHTML = cheerio(pm.response.text());
console.log(responseHTML.find('[name="_csrf"]').val());
```

Cheerio is desinged for non-browser use and implements a subset of the jQuery functionality. Read more about it at https://github.com/cheeriojs/cheerio

2.2.5 How to fix the error "ReferenceError: jsonData is not defined"?

If you have a script like this:

```javascript
pm.test("Name should be John", function () {
    var jsonData = pm.response.json();
    pm.expect(jsonData.name).to.eql('John');
});

pm.globals.set('name', jsonData.name);
```

You will get the error `ReferenceError:  jsonData is not defined` while setting the global variable.

The reason is that `jsonData` is only defined inside the scope of the anonymous function (the part with `function()` `{...}` inside `pm.test`). Where you are trying to set the global variables is outside the function, so `jsonData` is not defined. `jsonData` can only live within the scope where it was defined.

So you have multiple ways to deal with this:

1. define `jsonData` outside the function, for example before your pm.test function (preferred)

```
var jsonData = pm.response.json(); <-- defined outside callback

pm.test("Name should be John", function () {
    pm.expect(jsonData.name).to.eql('John');
});

pm.globals.set('name', jsonData.name);
```

2. set the environment or global variable inside the anonymous function (I would personally avoid mixing test / assertions with setting variables but it would work).

```
pm.test("Name should be John", function () {
    var jsonData = pm.response.json();
    pm.expect(jsonData.name).to.eql('John');
    pm.globals.set('name', jsonData.name); // <-- usage inside callback
});
```

Hope this helps and clarifies a bit the error.

2.3 Workflows

2.3.1 How to extract value of an authentication token from a login response body and pass in subsequent request as 'Bearer Token'?

Given the response from the authentication server:

```
{
    "accessToken": "foo",
    "refreshToken": "bar"
    "expires": "1234"
}
```

Extract the value of the token from the response in the **Tests** tab:

```
var jsonData = pm.response.json();
var token = jsonData.accessToken;
```

Set the token as a variable (global, environment, etc) so that it can used in later requests:

```
pm.globals.set('token', token);
```

To use the token in the next request, in the headers part the following needs to be added (key:value example below):

```
Authorization:Bearer {{token}}
```

2.3.2 How to read links from response and execute a request for each of them?

Given the following response:

```
{
    "links": [
        "http://example.com/1",
        "http://example.com/2",
```

```
        "http://example.com/3",
        "http://example.com/4",
        "http://example.com/5"
    ]
}
```

With the following code we will read the response, iterate over the links array and for each link will submit a request, using `pm.sendRequest`. For each response we will assert the status code.

```
// Parse the response
var jsonData = pm.response.json();

// Check the response
pm.test("Response contains links", function () {
    pm.response.to.have.status(200);
    pm.expect(jsonData.links).to.be.an('array').that.is.not.empty;
});


// Iterate over the response
var links = jsonData.links;

links.forEach(function(link) {
    pm.test("Status code is 404", function () {
        pm.sendRequest(link, function (err, res) {
            pm.expect(res).to.have.property('code', 404);
        });
    });
});
```

2.3.3 How to create request parameterization from Excel or JSON file?

TODO

2.4 Newman

2.4.1 How to set delay while running a collection?

You have a collection and have a requirement to insert a delay of 10 secs after every request.

In order to do that you can use the `--delay` parameter and specifiy a delay in miliseconds.

```
newman run collection.json --delay 10000
```

2.4.2 How to pass machine name and port number dynamically when running the tests?

Suppose, the URL to the server under the test may be different every time you get a new environment for testing, which is common with cloud environments. i.e. the part machine_name:port_number may be different.

There can be multiple way to do it, so below is one possible solution:

You can set global variables using newman from the CLI.

```
newman run my-collection.json --global-var "machineName=mymachine1234" --global-var
↪"machinePort=8080"
```

In your request builder, just use them as `https://{machineName{}}:{{machinePort}}`.

Online course

3.1 Postman online course

This document is part of the online course "Postman: The Complete Guide".

If you are not already a student of this course you are missing on a lot of training on Postman, including:

- Introduction to Postman
- Creating requests and workflows
- Writing tests
- Continuous Integration / Delivery with Jenkins or other CI/CI tools
- Practical assignments with personal feedback
- Q&A Forum where you can get answers to your Postman problems
- and much more

If you want to register for this course, make sure you use the link below as it will give you a **50% DISCOUNT** from the regular price:

https://www.udemy.com/postman-the-complete-guide/?couponCode=GUIDE50