Postman Quick Reference Guide Documentation

Release Version 1.6.3 - May 2020

Valentin Despa

May 15, 2020

Contents:

CHAPTER 1

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

All variables can be manually set using the Postman GUI and are scoped.

The code snippets can be used for working with variables in scripts (pre-request, tests).

Learn more about the different variables scopes in this tutorial.

Getting variables in the Request Builder

Depending on the closest scope:

Syntax: `{{myVariable}}`

Examples:

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

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

Request body: `{"id": "{{userId}}", "name": "John Doe"}`

Global variables

General purpose variables, ideal for quick results and prototyping.

Please consider using one of the more specialized variables below. Delete variables once they are no longer needed.

When to use:

- passing data to other requests

Setting

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

Getting

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

Alternatively, depending on the scope:

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

Removing

Remove one variable

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

Remove ALL global variables (rather unusual)

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

Collection variables

When to use:

- good alternative to global variables or environment variables
- for URLs / authentication credentials if only one environment exists

Setting

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

Getting

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

Removing

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

Environment variables

Environment variables are tied to the selected environment. Good alternative to global variables as they have a narrower scope.

When to use:

- storing environment specific information
- URLs, authentication credentials
- passing data to other requests

Setting

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

Getting

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

Depending on the closest scope:

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

Removing

Remove one variable

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

Remove ALL environment variables

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

Examples:

```
pm.environment.set('name', 'John Doe');
console.log(pm.environment.get('name'));
console.log(pm.variables.get('name'));
```

** Detecting the environment name **

If you need to know inside scripts which environment is currently active (locahost, production, ...) you can use the name property:

```
pm.environment.name
```

Data variables

Exist only during the execution of an iteration (created by the Collection Runner or Newman).

When to use:

- when multiple data-sets are needed

Setting

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

Getting

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

Depending on the closest scope:

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

Removing

Can only be removed from within the CSV or JSON file.

Local variables

Local variables are only available withing the request that has set them or when using Newman / Collection runner during the entire exection.

When to use:

- whenever you would like to override all other variable scopes—for whatever reason. Not sure though then this is needed.

Setting

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

Getting

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

Removing

Local variables are automatically removed once the tests have been executed.

Dynamic variables

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"}
```

If you want to use dynamic variables in scripts, you can use the *replaceIn* starting with Postman v7.6.0.

```
pm.variables.replaceIn('{{$randomFirstName}}'); // returns a String
```

For more details please see the section dedicated to *Dynamic variables*

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):

```
const response = pm.response.json();
```

Simple value check:

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

Nested value check:

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

XML responses

Convert XML body to JSON:

```
const response = xml2Json(responseBody);
```

Note: see assertions for JSON responses.

Skipping tests

You can use *pm.test.skip* to skip a test. Skipped tests will be displayed in reports.

Simple example

```
pm.test.skip("Status code is 200", () => {
    pm.response.to.have.status(200);
});
```

Conditional skip

```
const shouldBeSkipped = true; // some condition

(shouldBeSkipped ? pm.test.skip : pm.test)("Status code is 200", () => {
    pm.response.to.have.status(200);
});
```

Failing tests

You can fail a test from the scripts without writing an assertion:

```
pm.expect.fail('This failed because ...');
```

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('https://httpbin.org/get', (error, response) => {
    if (error) throw new Error(error);
    console.log(response.json());
});
```

Full-option **HTTP POST request with JSON body**:

```
const payload = { name: 'John', age: 29};

const options = {
    method: 'POST',
    url: 'https://httpbin.org/post',
    header: 'X-Foo:foo',
    body: {
        mode: 'raw',
        raw: JSON.stringify(payload)
    }
```

(continues on next page)

(continued from previous page)

```
};
pm.sendRequest(options, (error, response) => {
    if (error) throw new Error(error);
    console.log(response.json());
});
```

Form-data POST request (Postman will add the multipart/form-data header):

```
const options = {
    'method': 'POST',
    'url': 'https://httpbin.org/post',
    'body': {
            'mode': 'formdata',
            'formdata': [
                {'key':'foo', 'value':'bar'},
                {'key':'bar', 'value':'foo'}
            ]
    }
};
pm.sendRequest(options, (error, response) => {
    if (error) throw new Error(error);
    console.log(response.json());
});
```

Sending a file with form-data POST request

Due to security precautions, it is not possible to upload a file from a script using pm.sendRequest. You cannot read or write files from scripts.

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. It will NOT have any effect when using inside the Postman App.

Additionaly it is important to note that this will only affect the next request being executed. Even if you put this inside the pre-request script, it will NOT skip the current request.

Set which will be the next request to be executed

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

Stop executing requests / stop the collection run

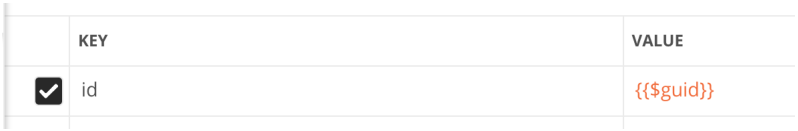
```
postman.setNextRequest(null);
```

CHAPTER 2

Dynamic variables

2.1 Dynamic variables

Dynamic variables can be used in the request builder like this:



If you want to use dynamic variables in scripts, you can use the *replaceIn* starting with Postman v7.6.0.

```
pm.variables.replaceIn('{{$randomFirstName}}');

pm.variables.replaceIn('{{$randomFirstName}} {{$randomLastName}}');
```

The *replaceIn* method will return a String with the resolved variables.

Before Postman 7.2, only the following dynamic variables were available:

Variable name	Description	Example
$guid	Generates a GUID (Globally Unique Identifier) in v4	15aacbb1-1615-47d8-b001-e5411a044761
$timestamp	Returns the current timestamp	1561013396
$randomInt	Generates random integer between 0 and 1000	764

Starting with version 7.2, Postman is using the faker.js library and added more variables. If used multiple times, they can return different values per request. Note: the autocomplete support in the Request Builder might be missing.

Variable name	Description	Examples	Comment
$randomZipCode	ZIP Code	83932, 40260-4447	[1]
$randomCity	City	East Ryanfurt, Jenkinsview	
$randomCityPrefix	City prefix	Port, West, East, Lake, New	
$randomCitySuffix	City suffix	mouth, borough, town, berg	
$randomStreetName	Street name	Mckenna Pines, Schiller Highway, Vandervort Pike	[2]
$randomStreetAddress	Street with number	98165 Tanya Passage, 0695 Monahan Squares	[3]
$randomStreetSuffix	Street suffix	Field, Bridge, Keys, Greens, Route	
$randomStreetPrefix	Street prefix	a, b, c	[4]
$randomSecondaryAddress	Additional address information	Suite 760, Apt. 636, Suite 043	[5]
$randomCounty	County	Bedfordshire, Cambridgeshire	[6]
$randomCountry	Country	Belgium, Antarctica (the territory South of 60 deg S)	
$randomCountryCode	Country code (2-letter)	GY, TK, BG	
$randomState	Random state	Arizona, South Dakota, Delaware	[7]
$randomStateAbbr	Random state code (2-letter)	GA, LA, AZ	[8]
$randomLatitude	Latitude	-79.9881, 87.8072	
$randomLongitude	Longitude	-41.5763, 10.4960	
$randomColor	Color	lime, azure, maroon, gold, violet	
$randomDepartment	Departments in a store	Garden, Clothing, Grocery, Kids	
$randomProductName	Product name	Intelligent Steel Sausages, Awesome Rubber Cheese	
$randomPrice	Price	244.00, 301.00	[9]
$randomProductAdjective	Product adjective	Refined, Handcrafted, Handmade, Sleek	
$randomProductMaterial	Product material	Frozen, Cotton, Wooden, Soft	
$randomProduct	Simple product name	Salad, Cheese, Bike, Soap	
$randomCompanyName	Company name	Christiansen LLC, Corwin Inc, Fahey - Boyer	
$randomCompanySuffix	Company suffix	LLC, Group, Inc, and Sons	
$randomCatchPhrase	Catch phrase	Centralized upward-trending attitude	
$randomBs	BS	one-to-one unleash communities	
$randomCatchPhraseAdjective	Catch phrase adjective	Total, Diverse, Horizontal	
$randomCatchPhraseDescriptor	Catch phrase descriptor	leading edge, dynamic, attitude-oriented	

Continued on next page

Table 1 – continued from previous page

Variable name	Description	Examples	Comment
$randomCatchPhraseNoun	Catch phrase noun	Graphical User Interface, matrix, benchmark	
$randomBsAdjective	BS adjective	compelling, vertical, revolutionary	
$randomBsBuzz	BS buzz	strategize, redefine, streamline	
$randomBsNoun	BS noun	systems, bandwidth, paradigms	
$randomDatabaseColumn	Database column	status, title, name, password, createdAt	
$randomDatabaseType	Database column type	enum, mediumint, double, timestamp	
$randomDatabaseCollation	Database collation	utf8_general_ci, utf8_bin	
$randomDatabaseEngine	Database engine	MEMORY, InnoDB, CSV, MyISAM	
$randomDatePast	Date in the past	Wed Mar 06 2019 04:17:52 GMT+0800 (WITA)	
$randomDateFuture	Date in the future	Wed Nov 20 2019 20:26:40 GMT+0800 (WITA)	
$randomDateBetween	???	Invalid Date	[10]
$randomDateRecent	Recent date	Thu Jun 20 2019 13:29:11 GMT+0800 (WITA)	
$randomMonth	Month	February, April	
$randomWeekday	Weekdat	Saturday, Monday	
$randomBankAccount	Bank account (8-digit)	58484223, 18983115	
$randomBankAccountName	Bank account name	Home Loan Account, Investment Account	
$randomCreditCardMask	Masked credit card number (4-digit)	7333, 6202	
$randomCurrencyAmount	Amount	297.80, 529.26	
$randomTransactionType	Transaction type	invoice, deposit, withdrawal, payment	
$randomCurrencyCode	Currency code	THB, HTG USD, AUD	
$randomCurrencyName	Currency name	Pound Sterling, Bulgarian Lev	
$randomCurrencySymbol	Currency symbol	$, , Kč	
$randomBitcoin	???	1XEW2WNQXFLUPQJU8F3D6OCJHV9UR	[11]
$randomBankAccountIban	IBAN	PK46Y5057900541310025311	[12]
$randomBankAccountBic	BIC	YQCIFMA1762	[13]
$randomAbbreviation	Abbreviation	RSS, SQL, TCP, HTTP, SMS	
$randomAdjective	Adjective	virtual, solid state, digital	
$randomNoun	Noun	microchip, interface, system, firewall	

Continued on next page

Table 1 – continued from previous page

Variable name	Description	Examples	Comment
$randomVerb	Verb	connect, parse, navigate, synthesize	
$randomIngverb	Verb with -ing	bypassing, copying, programming	
$randomPhrase	Phrase	We need to copy the online CSS microchip!	
$randomImage	Image URL	http://lorempixel.com/640/480/people	
$randomAvatarImage	Avatar image URL	https://s3.amazonaws.com/uifaces/faces/twitter/jacksonlatka/128.jpg	
$randomImageUrl	Image URL	http://lorempixel.com/640/480	
$randomAbstractImage	Abstract image	http://lorempixel.com/640/480/abstract	
$randomAnimalsImage	Image with animals	http://lorempixel.com/640/480/animals	
$randomBusinessImage	Business-related image	http://lorempixel.com/640/480/business	
$randomCatsImage	Image with cats	http://lorempixel.com/640/480/cats	
$randomCityImage	Image with a city	http://lorempixel.com/640/480/city	
$randomFoodImage	Image with food	http://lorempixel.com/640/480/food	
$randomNightlifeImage	Image with nightlife	http://lorempixel.com/640/480/nightlife	
$randomFashionImage	Image with fashion	http://lorempixel.com/640/480/fashion	
$randomPeopleImage	Image with people	http://lorempixel.com/640/480/people	
$randomNatureImage	Image with nature	http://lorempixel.com/640/480/nature	
$randomSportsImage	Image with sport	http://lorempixel.com/640/480/sports	
$randomTechnicsImage	Image with tech	http://lorempixel.com/640/480/technics	
$randomTransportImage	Image with transportation	http://lorempixel.com/640/480/transport	
$randomImageDataUri	Image as data URI	data:image/svg+xml;charset=UTF-8,%3Csvg%20 …	
$randomEmail	Email from popular email providers	Mable_Crist@hotmail.com, Ervin47@gmail.com	[14]
$randomExampleEmail	Example email	Ayla.Kozey27@example.net, Adrian.Hickle@example.com	
$randomUserName	Username	Minerva42, Shania_Nitzsche	
$randomProtocol	HTTP Protocol	http, https	
$randomUrl	URL	http://daphney.biz, https://ansley.com	
$randomDomainName	Domain name	adaline.org, murray.name, abdul.biz	
$randomDomainSuffix	Top Level Domain (TLD) extension	com, net, biz, name, org	
$randomDomainWord	Word that can be used within a domain name	guadalupe, willa, jose	

Continued on next page

Table 1 – continued from previous page

Variable name	Description	Examples	Comment
$randomIP	IP v4	147.236.215.88, 139.159.148.94	
$randomIPV6	IP v6	64d7:f61e:d265:167f:3971:9ae3:6853:3c48	
$randomUserAgent	Browser User-agent	Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 5.2; Trident/5.1)	
$randomHexColor	Color in hex format	#010638, #010638	
$randomMACAddress	MAC address	15:12:78:1e:96:fe, 99:f4:aa:40:49:59	
$randomPassword	Password	v_Ptr4aTaBONsM0, 8xQM6pKgBUndK_J	
$randomLoremWord	Lorem ipsum word	ipsa, dolor, dicta	
$randomLoremWords	Lorem ipsum words	debitis tenetur deserunt	
$randomLoremSentence	Lorem ipsum phrase	Qui fugiat necessitatibus porro quasi ea modi.	
$randomLoremSlug	Lorem ipsum slug	sint-dolorum-expedita, modi-quo-ut	
$randomLoremSentences	Lorem ipsum sentance	Voluptatum quidem rerum occaecati …	
$randomLoremParagraph	Lorem ipsum paragraph	Asperiores dolor illo. Ex …	
$randomLoremParagraphs	Lorem ipsum paragraphs	Saepe unde qui rerum …	[15]
$randomLoremText	Lorem ipsum text	Ipsam repellat qui aspernatur …	[16]
$randomLoremLines	Lorem ipsum text	aliquid enim reiciendis …	[17]
$randomFirstName	First name	Dillan, Sedrick, Daniela	
$randomLastName	Last name	Schamberger, McCullough, Becker	
$randomFullName	Full name	Layne Adams, Bridget O'Reilly III	
$randomJobTitle	Job title	Product Usability Consultant, Product Mobility Architect	
$randomNamePrefix	Personal title (used as prefix)	Miss, Mrs., Mr., Ms	
$randomNameSuffix	Title (used as suffix)	I, II, Sr., MD, PhD	
$randomNameTitle	Job title	Product Markets Administrator, Internal Functionality Producer	[18]
$randomJobDescriptor	Job title descriptor	Corporate, Global, International, Chief, Lead	
$randomJobArea	Job area	Creative, Markets, Tactics	
$randomJobType	Job type	Administrator, Consultant, Supervisor	
$randomPhoneNumber	Phone number	946.539.2542 x582, (681) 083-2162	[19]
$randomPhoneNumberFormat	Phone number	840-883-9861, 353-461-5243	[20]

Continued on next page

Table 1 – continued from previous page

Variable name	Description	Examples	Comment
$randomPhoneFormats	Phone number format	###.###.####, 1-###-###-#### x###, (###) ###-####	
$randomArrayElement	Random element from array [a,b, c]	a, b, c	
$randomObjectElement	Random object element	car, bar	
$randomUUID	UUID	1f9a0bc0-582c-466f-ba78-67b82ebbd8a8	
$randomBoolean	Boolean	true, false	[21]
$randomWord	Word or abbreviation	transmitting, PCI, West Virginia	
$randomWords	Words	portal bypassing indigo, Cotton transmitting	[22]
$randomLocale	Locale	en	[23]
$randomAlphaNumeric	Alphanumeric character	4, a, h	
$randomFileName	Filename	soft_smtp.wvx, calculate.grv	
$randomCommonFileName	Common filename	mall.pdf, chair.mp4, facilitator.mp3	
$randomMimeType	MIME type	application/x-font-bdf, application/omdoc+xml	
$randomCommonFileType	Common filetype	image, application, audio	
$randomCommonFileExt	Common file extension	png, mp3, mpeg, gif	
$randomFileType	File type	x-shader, font, audio, message	
$randomFileExt	File extension	xsm, zirz, xar	
$randomDirectoryPath	Directory path		[24]
$randomFilePath	File path		[25]
$randomSemver	Version (using semantic version)	6.3.4, 2.8.0, 1.7.6	

[1] Not really useful as you cannot specify a country.
[2] Limited usability as you cannot specify a country.
[3] Warning: it may generate invalid data, with street numbers starting with 0. Limited usability as you cannot specify a country.
[4] Not sure what a street prefix is. Unknown usage.
[5] Warning: it may generate invalid data, with numbers starting with 0. Limited usability as you cannot specify a country.
[6] Limited usability as you cannot specify a country.
[7] Limited to US states.
[8] Limited to US states.
[9] Not possible to specify a format. It seems that the price is never with a subdivision (cents). Alternative: $randomCurrencyAmount.
[10] Seems to be broken.
[11] Does not look like a Bitcoin address.
[12] May not be a valid IBAN.
[13] May not be a valid BIC.
[14] Better use example emails.
[15] Includes \n \r characters (CR + LF).
[16] Length is unpredictable. May include \n \r characters (CR + LF).
[17] Length is unpredictable. May include \n characters (LF).
[18] Seems to overlap with $$randomJobTitle.
[19] Random format. Cannot specify a format / country.
[20] Fixed format. Cannot specify another format / country
[21] Warning: the output is still a string!
[22] May return only one word.
[23] Seems broken as it returns only "en".
[24] Seems broken.
[25] Seems broken.

CHAPTER 3

Simple solutions to common problems

3.1 Request creation

3.1.1 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
});
```

3.1.2 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"
    }
}
```

3.1.3 How can I modify the request headers?

You can modify the request headers from the Pre-request script as follows.

Add header

```
pm.request.headers.add({
    key: 'X-Foo',
    value: 'Postman'
});
```

Remove header

```
pm.request.headers.remove('User-Agent'); // may not always work
```

Update header

```
pm.request.headers.upsert(
    {
        key: "User-Agent",
        value: "Not Postman"

    }
);
```

3.1.4 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}}",
```

(continues on next page)

(continued from previous page)

```
    "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

Below you will find an exemple function that you can use to generate integer number between a specific interval:

```
function getRandomNumber(minValue, maxValue) {
    return Math.floor(Math.random() * (maxValue - minValue +1)) + minValue;
}
```

You can call the function like this:

```
var myRandomNumber = getRandomNumber(0, 100);
```

And the output will look similar to:

```
67
```

Below you will find an exemple function that you can use to generate random strings:

```
function getRandomString() {
    return Math.random().toString(36).substring(2);
}
```

You can call the function like this:

```
var myRandomNumber = getRandomString();
```

And the output will look similar to:

```
5q04pes32yi
```

3.1.5 How to trigger 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

3.1.6 How to send request with XML body from a script?

You can use the following template to send a XML request from a script. Notice that *price* is a Postman variable that will be replaced.

```
const xmlBody = `<?xml version="1.0"?>
<catalog>
<book id="bk101">
    <author>Gambardella, Matthew</author>
    <title>XML Developer's Guide</title>
    <genre>Computer</genre>
    <price>{{price}}</price>
    <publish_date>2000-10-01</publish_date>
    <description>An in-depth look at creating applications
    with XML.</description>
</book>
</catalog>`;

const options = {
    'method': 'POST',
    'url': 'httpbin.org/post',
    'header': {
        'Content-Type': 'application/xml'
    },
    body: pm.variables.replaceIn(xmlBody) // replace any Postman variables
}


pm.sendRequest(options, function (error, response) {
    if (error) throw new Error(error);
    console.log(response.body);
});
```

3.1.7 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 may not work (depending on your Postman or Newman version).

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.

3.1.8 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.

Possible options:

- start a local server to serve that file and to get it in Postman with a GET request.
- use Newman as a custom Node.js script and read the file using the filesystem.

3.1.9 How to add a delay between Postman requests?

To add a delay after a request, add the following in your Tests:

```
setTimeout(() => {}, 10000);
```

The example above will add a delay of 10000 milliseconds or 10 seconds.

3.2 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/

3.2.1 How find object in array by property value?

Given the following response:

```
{
    "companyId": 10101,
    "regionId": 36554,
    "filters": [
        {
            "id": 101,
            "name": "VENDOR",
            "isAllowed": false
        },
        {
```

(continues on next page)

(continued from previous page)

```
            "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;
});
```

3.2.2 How find nested object by object name?

Given the following response:

```
{
    "id": "5a866bd667ff15546b84ab78",
    "limits": {
        "59974328d59230f9a3f946fe": {
            "lists": {
                "openPerBoard": {
                    "count": 13,
                    "status": "ok", <-- CHECK ME
                    "disableAt": 950,
                    "warnAt": 900
                },
                "totalPerBoard": {
                    "count": 20,
                    "status": "ok",  <-- CHECK ME
                    "disableAt": 950,
                    "warnAt": 900
                }
            }
        }
    }
}
```

You want to check the value of the *status* in both objects (openPerBoard, totalPerBoard). The problem is that in order to reach both objects you need first to reach the lists object, which itself is a property of a randomly named object (59974328d59230f9a3f946fe).

So we could write the whole path `limits.59974328d59230f9a3f946fe.lists.openPerBoard.status` but this will probably work only once.

For that reason it is first needed to search inside the `limits` object for the `lists` object. In order to make the code more readable, we will create a function for that:

```
function findObjectContaininsLists(limits) {
    // Iterate over the properties (keys) in the object
    for (var key in limits) {
        // console.log(key, limits[key]);
        // If the property is lists, return the lists object
        if (limits[key].hasOwnProperty('lists')) {
            // console.log(limits[key].lists);
            return limits[key].lists;
        }
    }
}
```

The function will iterate over the limits array to see if any object contains a `lists` object.

Next all we need to do is to call the function and the assertions will be trivial:

```
pm.test("Check status", function () {
    // Parse JSON body
    var jsonData = pm.response.json();

    // Retrieve the lists object
    var lists = findObjectContaininsLists(jsonData.limits);
    pm.expect(lists.openPerBoard.status).to.eql('ok');
    pm.expect(lists.totalPerBoard.status).to.eql('ok');
});
```

3.2.3 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"));
});
```

3.2.4 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

3.2.5 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:

```
<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:

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

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

3.2.6 How to fix the error “ReferenceError: jsonData is not defined”?

If you have a script like this:

```
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.

3.2.7 How to do a partial object match assertion?

Given the reponse:

```
{
    "uid": "12344",
    "pid": "8896",
    "firstName": "Jane",
    "lastName": "Doe",
    "companyName": "ACME"
}
```

You want to assert that a part of the reponse has a specific value. For example you are not interested in the dynamic value of uid and pid but you want to assert firstName, lastName and companyName.

You can do a partial match of the response by using the `to.include` expression. Optionally you can check the existence of the additional properties without checking the value.

```
pm.test("Should include object", function () {
    var jsonData = pm.response.json();
    var expectedObject = {
        "firstName": "Jane",
        "lastName": "Doe",
        "companyName": "ACME"
```

(continues on next page)

(continued from previous page)

```
    }
    pm.expect(jsonData).to.include(expectedObject);

    // Optional check if properties actually exist
    pm.expect(jsonData).to.have.property('uid');
    pm.expect(jsonData).to.have.property('pid');
});
```

3.3 Schema validation

This section contains different examples of validating JSON responses using the Ajv schema validator. I do not recommend using the tv4 (Tiny Validator for JSON Schema v4).

3.3.1 Response is an object

This is the JSON response:

```
{}
```

This is the JSON schema:

```
const schema = {
    "type": "object",
};
```

And here is the test:

```
pm.test("Validate schema", () => {
    pm.response.to.have.jsonSchema(schema);
});
```

3.3.2 Object has optional property

This is the JSON response:

```
{
    "code": "FX002"
}
```

This is the JSON schema with a property named code of type String:

```
const schema = {
    "type": "object",
    "properties": {
        "code": { "type": "string" }
    }
};
```

Possible types are:

- integer
- number
- boolean
- null

- Object
- array

3.3.3 Object has required property

Given this JSON response:

```
{
    "code": "FX002"
}
```

This is the JSON schema with a property named "code" of type String that is mandatory:

```
const schema = {
    "type": "object",
    "properties": {
        "code": { "type": "string" }
    },
    "required": ["code"]
};
```

3.3.4 Nested objects

Given this JSON response:

```
{
    "code": "2",
    "error": {
        "message": "Not permitted."
    }
}
```

This is the JSON schema with the an nested object named "error" that has a property named "message" that is a string.

```
const schema = {
    "type": "object",
    "properties": {
        "code": { "type": "string" },
        "error": {
            "type": "object",
            "properties": {
                "message": { type: "string" }
            },
            "required": ["message"]
        }
    },
    "required": ["code", "error"]
};
```

3.4 JavaScript libraries

Postman comes with a few built-in libraries. If you prefer to add additioal JavaScript libraries, please take a look at the Custom libraries section.

3.4.1 Built-in JavaScript libraries

cheerio

Simple library for working with the DOM model. Useful if you are getting back HTML.

```
responseHTML = cheerio(pm.response.text());
console.log(responseHTML.find('[name="firstName"]').val());
```

Example fetching a CSRF code form the meta tag:

```
const $ = cheerio.load('<meta name="csrf" Content="the code">');
console.log($("meta[name='csrf']").attr("content"));
```

Read more: https://github.com/cheeriojs/cheerio

crypto-js

Library which implements different crypto functions.

Hash string using SHA256

```
CryptoJS.SHA256("some string").toString()
```

HMAC-SHA1 encryption

```
CryptoJS.HmacSHA1("Message", "Key").toString()
```

AES Encryption

```
const encryptedText = CryptoJS.AES.encrypt('message', 'secret').toString();
```

AES Decryption

```
const plainText = CryptoJS.AES.decrypt(encryptedText, 'secret').toString(CryptoJS.
↪enc.Utf8);
```

Read more: https://www.npmjs.com/package/crypto-js

3.4.2 Node.js libraries

NodeJS modules that are available inside Postman:

- path
- assert
- buffer
- util
- url
- punycode
- querystring
- string_decoder
- stream
- timers
- events

3.4.3 Custom libraries

There is no standard way of including 3rd party JavaScript libraries.

Currently the only way is to fetch (and optionally store) the content of the JavaScript library and to use the JavaScript *eval* function to execute the code.

Template:

```
pm.sendRequest("https://example.com/your-script.js", (error, response) => {
    if (error || response.code !== 200) {
        pm.expect.fail('Could not load external library');
    }

    eval(response.text());

    // YOUR CODE HERE
});
```

Example loading a library using unpkg as a CDN:

```
pm.sendRequest("https://unpkg.com/papaparse@5.1.0/papaparse.min.js", (error,␣
↪response) => {
    if (error || response.code !== 200) {
        pm.expect.fail('Could not load external library');
    }

    eval(response.text());

    const csv = `id,name\n1,John`;
    const data = this.Papa.parse(csv); // notice the this
    console.log(data);
});
```

Notice: in order to load a library and use it in Postman, the JavaScript code needs to be "compiled" and ready for distribution. Usually the code will be available as a *.min.js file or within the dist or umd folder.

3.5 Workflows

3.5.1 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}}
```

3.5.2 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);
        });
    });
});
```

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

TODO

3.6 Newman

3.6.1 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
```

3.6.2 Jenkins is showing wierd characters in the console. What to do?

If the Newman output in your CI server is not properly displayed, try adding following flags: `--disable-unicode` or / and `--color off`

Example:

```
newman run collection.json --disable-unicode
```

3.6.3 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}}`.

CHAPTER 4

Online course

4.1 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 (Gitlab, TeamCity)
- 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 **75% DISCOUNT** from the regular price:

https://www.udemy.com/postman-the-complete-guide/?couponCode=PQRG10

Enjoy!