## Contents

1 Cheatsheet  
   1.1 Postman Cheatsheet .................................................. 1

2 Dynamic variables  
   2.1 Dynamic variables .................................................... 9

3 Simple solutions to common problems  
   3.1 Request creation .................................................... 15  
   3.2 Assertions ........................................................... 20  
   3.3 Schema validation ................................................... 25  
   3.4 JavaScript libraries .................................................. 26  
   3.5 Workflows ........................................................... 28  
   3.6 Newman ............................................................... 29

4 Online course  
   4.1 Contributions ....................................................... 31  
   4.2 Postman online course .............................................. 31
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**

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

**Getting**

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

Alternatively, depending on the scope:

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

**Removing**

Remove one variable

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

Remove ALL global variables (rather unusual)

```javascript
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**

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

**Getting**

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

**Removing**

```javascript
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**

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

```javascript
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 (localhost, production, ...) you can use the name property:

```javascript
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**

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

Depending on the closest scope:

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

**Removing**

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

**Local variables**

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

**When to use:**

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

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

Getting

```javascript
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:

```json
{"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.

```javascript
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:

```javascript
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:

```javascript
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:

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

Checking multiple status codes:

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

Response time below 100ms:

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

Headers

Header exists:

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

Header has value:

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

Cookies

Cookie exists:

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

Cookie has value:

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

Body

Any content type / HTML responses

Exact body match:

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

Partial body match / body contains:

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

JSON responses

Parse body (need for all assertions):

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

Simple value check:

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

Nested value check:

```javascript
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**

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

**Conditional skip**

```javascript
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:

```javascript
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:

```javascript
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:

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

```javascript
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

```javascript
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.

Additionally 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);`
2.1 Dynamic variables

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

<table>
<thead>
<tr>
<th>KEY</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td><code>{{guid}}</code></td>
</tr>
</tbody>
</table>

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

```javascript
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:

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>$guid</td>
<td>Generates a GUID (Globally Unique Identifier) in v4</td>
<td>15aacbb1-1615-47d8-b001-e5411a044761</td>
</tr>
<tr>
<td>Stimestamp</td>
<td>Returns the current timestamp</td>
<td>1561013396</td>
</tr>
<tr>
<td>SrandomInt</td>
<td>Generates random integer between 0 and 1000</td>
<td>764</td>
</tr>
</tbody>
</table>

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.
<table>
<thead>
<tr>
<th>Variable name</th>
<th>Description</th>
<th>Examples</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>$randomCity</td>
<td>City</td>
<td>East Ryanfurt, Jenkinsview</td>
<td></td>
</tr>
<tr>
<td>$randomStreetName</td>
<td>Street name</td>
<td>Mckenna Pines, Schiller Highway, Vandervort Pike</td>
<td></td>
</tr>
<tr>
<td>$randomStreetAddress</td>
<td>Street with number</td>
<td>98165 Tanya Passage, 0695 Monahan Squares</td>
<td></td>
</tr>
<tr>
<td>$randomCountry</td>
<td>Country</td>
<td>Belgium, Antarctica (the territory South of 60 deg S)</td>
<td></td>
</tr>
<tr>
<td>$randomCountryCode</td>
<td>Country code (2-letter)</td>
<td>CY, TK, BG</td>
<td></td>
</tr>
<tr>
<td>$randomLatitude</td>
<td>Latitude</td>
<td>-79.9881, 87.8072</td>
<td></td>
</tr>
<tr>
<td>$randomLongitude</td>
<td>Longitude</td>
<td>-41.5763, 10.4960</td>
<td></td>
</tr>
<tr>
<td>$randomColor</td>
<td>Color</td>
<td>lime, azure, maroon, gold, violet</td>
<td></td>
</tr>
<tr>
<td>$randomDepartment</td>
<td>Departments in a store</td>
<td>Garden, Clothing, Grocery, Kids</td>
<td></td>
</tr>
<tr>
<td>$randomProductName</td>
<td>Product name</td>
<td>Intelligent Steel Sausages, Awesome Rubber Cheese</td>
<td></td>
</tr>
<tr>
<td>$randomPrice</td>
<td>Price</td>
<td>244.00, 301.00</td>
<td></td>
</tr>
<tr>
<td>$randomProductAdjective</td>
<td>Product adjective</td>
<td>Refined, Handcrafted, Handmade, Sleek</td>
<td></td>
</tr>
<tr>
<td>$randomProductMaterial</td>
<td>Product material</td>
<td>Frozen, Cotton, Wooden, Soft</td>
<td></td>
</tr>
<tr>
<td>$randomProduct</td>
<td>Simple product name</td>
<td>Salad, Cheese, Bike, Soap</td>
<td></td>
</tr>
<tr>
<td>$randomCompanyName</td>
<td>Company name</td>
<td>Christiansen LLC, Corwin Inc, Fahey - Boyer</td>
<td></td>
</tr>
<tr>
<td>$randomCompanySuffix</td>
<td>Company suffix</td>
<td>LLC, Group, Inc, and Sons</td>
<td></td>
</tr>
<tr>
<td>$randomCatchPhrase</td>
<td>Catch phrase</td>
<td>Centralized upward-trending attitude</td>
<td></td>
</tr>
<tr>
<td>$randomBs</td>
<td>BS</td>
<td>one-to-one unleash communities</td>
<td></td>
</tr>
<tr>
<td>$randomCatchPhraseAdjective</td>
<td>Catch phrase adjective</td>
<td>Total, Diverse, Horizontal</td>
<td></td>
</tr>
<tr>
<td>$randomCatchPhraseDescriptor</td>
<td>Catch phrase descriptor</td>
<td>leading edge, dynamic, attitude-oriented</td>
<td></td>
</tr>
<tr>
<td>$randomCatchPhraseNoun</td>
<td>Catch phrase noun</td>
<td>Graphical User Interface, matrix, benchmark</td>
<td></td>
</tr>
<tr>
<td>$randomBsAdjective</td>
<td>BS adjective</td>
<td>compelling, vertical, revolutionary</td>
<td></td>
</tr>
<tr>
<td>$randomBsBuzz</td>
<td>BS buzz</td>
<td>strategize, redefine, streamline</td>
<td></td>
</tr>
<tr>
<td>$randomBsNoun</td>
<td>BS noun</td>
<td>systems, bandwidth, paradigms</td>
<td></td>
</tr>
<tr>
<td>$randomDatabaseColumn</td>
<td>Database column</td>
<td>status, title, name, password, createdAt</td>
<td></td>
</tr>
<tr>
<td>$randomDatabaseType</td>
<td>Database column type</td>
<td>enum, mediumint, double, timestamp</td>
<td></td>
</tr>
<tr>
<td>$randomDatabaseCollation</td>
<td>Database collation</td>
<td>utf8_general_ci, utf8_bin</td>
<td></td>
</tr>
<tr>
<td>$randomDatabaseEngine</td>
<td>Database engine</td>
<td>MEMORY, InnoDB, CSV, MyISAM</td>
<td></td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Variable name</th>
<th>Description</th>
<th>Examples</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>$randomDatePast</td>
<td>Date in the past</td>
<td>Wed Mar 06 2019 04:17:52 GMT+0800 (WITA)</td>
<td></td>
</tr>
<tr>
<td>$randomDateFuture</td>
<td>Date in the future</td>
<td>Wed Nov 20 2019 20:26:40 GMT+0800 (WITA)</td>
<td></td>
</tr>
<tr>
<td>$randomDateRecent</td>
<td>Recent date</td>
<td>Thu Jun 20 2019 13:29:11 GMT+0800 (WITA)</td>
<td></td>
</tr>
<tr>
<td>$randomMonth</td>
<td>Month</td>
<td>February, April</td>
<td></td>
</tr>
<tr>
<td>$randomWeekday</td>
<td>Weekday</td>
<td>Saturday, Monday</td>
<td></td>
</tr>
<tr>
<td>$randomBankAccount</td>
<td>Bank account (8-digit)</td>
<td>58484223, 18983115</td>
<td></td>
</tr>
<tr>
<td>$randomBankAccountName</td>
<td>Bank account name</td>
<td>Home Loan Account, Investment Account</td>
<td></td>
</tr>
<tr>
<td>$randomCreditCardMask</td>
<td>Masked credit card number (4-digit)</td>
<td>7333, 6202</td>
<td></td>
</tr>
<tr>
<td>$randomCurrencyAmount</td>
<td>Amount</td>
<td>297.80, 529.26</td>
<td></td>
</tr>
<tr>
<td>$randomTransactionType</td>
<td>Transaction type</td>
<td>invoice, deposit, withdrawal, payment</td>
<td></td>
</tr>
<tr>
<td>$randomCurrencyCode</td>
<td>Currency code</td>
<td>THB, HTG USD, AUD</td>
<td></td>
</tr>
<tr>
<td>$randomCurrencyName</td>
<td>Currency name</td>
<td>Pound Sterling, Bulgarian Lev</td>
<td></td>
</tr>
<tr>
<td>$randomCurrencySymbol</td>
<td>Currency symbol</td>
<td>$, , Kč</td>
<td></td>
</tr>
<tr>
<td>$randomBitcoin</td>
<td>???</td>
<td>1URwKsDptEkTYNGYfKBZUWcuavqKSL</td>
<td></td>
</tr>
<tr>
<td>$randomBankAccountIban</td>
<td>IBAN</td>
<td>PK46Y5057909541310025311</td>
<td></td>
</tr>
<tr>
<td>$randomBankAccountBic</td>
<td>BIC</td>
<td>YQCIFMA1762</td>
<td></td>
</tr>
<tr>
<td>$randomAbbreviation</td>
<td>Abbreviation</td>
<td>RSS, SQL, TCP, HTTP, SMS</td>
<td></td>
</tr>
<tr>
<td>$randomAdjective</td>
<td>Adjective</td>
<td>virtual, solid state, digital</td>
<td></td>
</tr>
<tr>
<td>$randomNoun</td>
<td>Noun</td>
<td>microchip, interface, system, firewall</td>
<td></td>
</tr>
<tr>
<td>$randomVerb</td>
<td>Verb</td>
<td>connect, parse, navigate, synthesize</td>
<td></td>
</tr>
<tr>
<td>$randomIngverb</td>
<td>Verb with -ing</td>
<td>bypassing, copying, programming</td>
<td></td>
</tr>
<tr>
<td>$randomPhrase</td>
<td>Phrase</td>
<td>We need to copy the online CSS microchip!</td>
<td></td>
</tr>
<tr>
<td>$randomImageUrl</td>
<td>Image URL</td>
<td><a href="http://lorempixel.com/640/480">http://lorempixel.com/640/480</a></td>
<td></td>
</tr>
<tr>
<td>$randomAbstractImage</td>
<td>Abstract image</td>
<td><a href="http://lorempixel.com/640/480/abstract">http://lorempixel.com/640/480/abstract</a></td>
<td></td>
</tr>
<tr>
<td>$randomAnimalsImage</td>
<td>Image with animals</td>
<td><a href="http://lorempixel.com/640/480/animals">http://lorempixel.com/640/480/animals</a></td>
<td></td>
</tr>
<tr>
<td>$randomCatImage</td>
<td>Image with cats</td>
<td><a href="http://lorempixel.com/640/480/cats">http://lorempixel.com/640/480/cats</a></td>
<td></td>
</tr>
<tr>
<td>Variable name</td>
<td>Description</td>
<td>Examples</td>
<td>Comment</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------</td>
<td>-----------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>$randomCityImage</td>
<td>Image with a city</td>
<td><a href="http://lorempixel.com/640/480/city">http://lorempixel.com/640/480/city</a></td>
<td></td>
</tr>
<tr>
<td>$randomFoodImage</td>
<td>Image with food</td>
<td><a href="http://lorempixel.com/640/480/food">http://lorempixel.com/640/480/food</a></td>
<td></td>
</tr>
<tr>
<td>$randomNightlifeImage</td>
<td>Image with nightlife</td>
<td><a href="http://lorempixel.com/640/480/nightlife">http://lorempixel.com/640/480/nightlife</a></td>
<td></td>
</tr>
<tr>
<td>$randomFashionImage</td>
<td>Image with fashion</td>
<td><a href="http://lorempixel.com/640/480/fashion">http://lorempixel.com/640/480/fashion</a></td>
<td></td>
</tr>
<tr>
<td>$randomPeopleImage</td>
<td>Image with people</td>
<td><a href="http://lorempixel.com/640/480/people">http://lorempixel.com/640/480/people</a></td>
<td></td>
</tr>
<tr>
<td>$randomNatureImage</td>
<td>Image with nature</td>
<td><a href="http://lorempixel.com/640/480/nature">http://lorempixel.com/640/480/nature</a></td>
<td></td>
</tr>
<tr>
<td>$randomSportsImage</td>
<td>Image with sport</td>
<td><a href="http://lorempixel.com/640/480/sports">http://lorempixel.com/640/480/sports</a></td>
<td></td>
</tr>
<tr>
<td>$randomTransportImage</td>
<td>Image with transportation</td>
<td><a href="http://lorempixel.com/640/480/transport">http://lorempixel.com/640/480/transport</a></td>
<td></td>
</tr>
<tr>
<td>$randomImageDataUri</td>
<td>Image as data URI</td>
<td>data:image/svg+xml;charset=UTF-8,%3Csvg%20...</td>
<td></td>
</tr>
<tr>
<td>$randomEmail</td>
<td>Email from popular email providers</td>
<td><a href="mailto:Mable_Crist@hotmail.com">Mable_Crist@hotmail.com</a>, <a href="mailto:Ervin47@gmail.com">Ervin47@gmail.com</a></td>
<td></td>
</tr>
<tr>
<td>$randomExampleEmail</td>
<td>Example email</td>
<td><a href="mailto:Ayla.Kozy27@example.net">Ayla.Kozy27@example.net</a>, <a href="mailto:Adrian.Hickle@example.com">Adrian.Hickle@example.com</a></td>
<td></td>
</tr>
<tr>
<td>$randomUserName</td>
<td>Username</td>
<td>Minerva42, Shania_Nitzsche</td>
<td></td>
</tr>
<tr>
<td>$randomProtocol</td>
<td>HTTP Protocol</td>
<td>http, https</td>
<td></td>
</tr>
<tr>
<td>$randomDomainName</td>
<td>Domain name</td>
<td>adaline.org, murray.name, abdul.biz</td>
<td></td>
</tr>
<tr>
<td>$randomDomainSuffix</td>
<td>Top Level Domain (TLD) extension</td>
<td>com, net, biz, name, org</td>
<td></td>
</tr>
<tr>
<td>$randomDomainWord</td>
<td>Word that can be used within a domain name</td>
<td>guadalupe, willa, jase</td>
<td></td>
</tr>
<tr>
<td>$randomIP</td>
<td>IP v4</td>
<td>147.236.215.88, 139.159.148.94</td>
<td></td>
</tr>
<tr>
<td>$randomIPV6</td>
<td>IP v6</td>
<td>64d7:f61e:d265:167f:3971:9ae3:6853:3c48</td>
<td></td>
</tr>
<tr>
<td>$randomUserAgent</td>
<td>Browser User-agent</td>
<td>Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 5.2; Trident/5.1)</td>
<td></td>
</tr>
<tr>
<td>$randomHexColor</td>
<td>Color in hex format</td>
<td>#010638, #010638</td>
<td></td>
</tr>
<tr>
<td>$randomPassword</td>
<td>Password</td>
<td>v_Ptr4aTaBONsM0, 8xQM6pKgBUndK_J</td>
<td></td>
</tr>
<tr>
<td>$randomLoremWord</td>
<td>Lorem ipsum word</td>
<td>ipsa, dolor, dicta</td>
<td></td>
</tr>
<tr>
<td>$randomLoremWords</td>
<td>Lorem ipsum words</td>
<td>debitis tenetur deserunt</td>
<td></td>
</tr>
<tr>
<td>$randomLoremSentence</td>
<td>Lorem ipsum phrase</td>
<td>Qui fugiat necessitatibus porro quasi ea modi.</td>
<td></td>
</tr>
<tr>
<td>$randomLoremSlug</td>
<td>Lorem ipsum slug</td>
<td>sint-dolorum-expedita, modi-quo-ut</td>
<td></td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Variable name</th>
<th>Description</th>
<th>Examples</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>$randomLoremSentences</td>
<td>Lorem ipsum sentence</td>
<td>Voluptatum quidem rerum occaecati ...</td>
<td></td>
</tr>
<tr>
<td>$randomLoremParagraph</td>
<td>Lorem ipsum paragraph</td>
<td>Asperiores dolor illo, Ex ...</td>
<td></td>
</tr>
<tr>
<td>$randomLoremParagraphs</td>
<td>Lorem ipsum paragraphs</td>
<td>Saepe unde qui rerum ...</td>
<td></td>
</tr>
<tr>
<td>$randomLoremText</td>
<td>Lorem ipsum text</td>
<td>Ipsum repellat qui aspernatur ...</td>
<td></td>
</tr>
<tr>
<td>$randomFirstName</td>
<td>First name</td>
<td>Dillan, Sedrick, Daniela</td>
<td></td>
</tr>
<tr>
<td>$randomLastName</td>
<td>Last name</td>
<td>Schamberger, McCullough, Becker</td>
<td></td>
</tr>
<tr>
<td>$randomFullName</td>
<td>Full name</td>
<td>Layne Adams, Bridget O’Reilly III</td>
<td></td>
</tr>
<tr>
<td>$randomJobTitle</td>
<td>Job title</td>
<td>Product Usability Consultant, Product Mobility Architect</td>
<td></td>
</tr>
<tr>
<td>$randomNamePrefix</td>
<td>Personal title (used as prefix)</td>
<td>Miss, Mrs., Mr., Ms</td>
<td></td>
</tr>
<tr>
<td>$randomNameSuffix</td>
<td>Title (used as suffix)</td>
<td>I, II, Sr., MD, PhD</td>
<td></td>
</tr>
<tr>
<td>$randomNameTitle</td>
<td>Job title</td>
<td>Product Markets Administrator, Internal Functionality Producer</td>
<td></td>
</tr>
<tr>
<td>$randomJobDescriptor</td>
<td>Job title descriptor</td>
<td>Corporate, Global, International, Chief, Lead</td>
<td></td>
</tr>
<tr>
<td>$randomJobArea</td>
<td>Job area</td>
<td>Creative, Markets, Tactics</td>
<td></td>
</tr>
<tr>
<td>$randomJobType</td>
<td>Job type</td>
<td>Administrator, Consultant, Supervisor</td>
<td></td>
</tr>
<tr>
<td>$randomPhoneNumber</td>
<td>Phone number</td>
<td>946.539.2542 x582, (681) 083-2162</td>
<td></td>
</tr>
<tr>
<td>$randomPhoneNumberFormat</td>
<td>Phone number format</td>
<td>840-883-9861, 353-461-5243</td>
<td></td>
</tr>
<tr>
<td>$randomArrayElement</td>
<td>Random element from array [a, b, c]</td>
<td>a, b, c</td>
<td></td>
</tr>
<tr>
<td>$randomObjectElement</td>
<td>Random object element</td>
<td>car, bar</td>
<td></td>
</tr>
<tr>
<td>$randomUUID</td>
<td>UUID</td>
<td>1f9a0bc0-582c-466f-ba78-67b82ebbd8a8</td>
<td></td>
</tr>
<tr>
<td>$randomBoolean</td>
<td>Boolean</td>
<td>true, false</td>
<td></td>
</tr>
<tr>
<td>$randomWord</td>
<td>Word or abbreviation</td>
<td>transmitting, PCI, West Virginia</td>
<td></td>
</tr>
<tr>
<td>$randomWords</td>
<td>Words</td>
<td>portal bypassing indigo, Cotton transmitting</td>
<td></td>
</tr>
<tr>
<td>$randomLocale</td>
<td>Locale</td>
<td>en</td>
<td></td>
</tr>
<tr>
<td>$randomAlphaNumeric</td>
<td>Alphanumeric character</td>
<td>4, a, h</td>
<td></td>
</tr>
<tr>
<td>$randomFileName</td>
<td>Filename</td>
<td>soft_smtp.wvx, calculate.grv</td>
<td></td>
</tr>
<tr>
<td>$randomCommonFileName</td>
<td>Common filename</td>
<td>mall.pdf, chair.mp4, facilitator.mp3</td>
<td></td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Variable name</th>
<th>Description</th>
<th>Examples</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>$randomMimeType</td>
<td>MIME type</td>
<td>application/x-font-bdf, application/omdoc+xml</td>
<td></td>
</tr>
<tr>
<td>$randomCommonFileType</td>
<td>Common filetype</td>
<td>image, application, audio</td>
<td></td>
</tr>
<tr>
<td>$randomCommonFileExt</td>
<td>Common file extension</td>
<td>png, mp3, mpeg, gif</td>
<td></td>
</tr>
<tr>
<td>$randomFileType</td>
<td>File type</td>
<td>x-shader, font, audio, message</td>
<td></td>
</tr>
<tr>
<td>$randomFileExt</td>
<td>File extension</td>
<td>xsm, zirz, xar</td>
<td></td>
</tr>
<tr>
<td>$randomDirectoryPath</td>
<td>Directory path</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$randomFilePath</td>
<td>File path</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$randomSemver</td>
<td>Version (using semantic version)</td>
<td>6.3.4, 2.8.0, 1.7.6</td>
<td></td>
</tr>
</tbody>
</table>

1 Limited usability as you cannot specify a country.
2 Warning: it may generate invalid data, with street numbers starting with 0. Limited usability as you cannot specify a country.
3 Not possible to specify a format. It seems that the price is never with a subdivision (cents). Alternative: $randomCurrencyAmount.
4 May not be a valid IBAN.
5 May not be a valid BIC.
6 Better use example emails.
7 Includes \n or \r characters (CR + LF).
8 Length is unpredictable. May include \n or \r characters (CR + LF).
9 Length is unpredictable. May include \n or \r characters (LF).
10 Seems to overlap with $randomJobTitle.
11 Random format. Cannot specify a format / country.
12 Fixed format. Cannot specify another format / country.
13 Warning: the output is still a string!
14 May return only one word.
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 create or modify the request body?

You can use `console.log(pm.request.body)` to understand the current data structure of the request body. With the method `pm.request.body.update` you can update the request.

**Create/replace JSON request body**

```javascript
const body = {
    mode: "raw",
    raw: JSON.stringify(
        {
            name: "jane",
            age: 33
        }
    ),
    options: {
        raw: {
            language: "json"
        }
    }
}

pm.request.body.update(body);
```

**Add property to JSON request body**

```javascript
const body = JSON.parse(pm.request.body.raw);
body.email = 'jane@example.com';
pm.request.body.raw = body;
```

**Encode request body as base64**

```javascript
pm.request.body.update(btoa(pm.request.body.toString()));
```

**Removing comments from JSON body**

```javascript
const jsonWithComments = pm.request.body.toString();
const jsonWithoutComments = jsonWithComments.replace(/\"|\(\?\:\|[^\"]\)+\|\(\//.\-
\*\|\//\*[\s\$]+?\//\*/g, (m, g) => g); (continues on next page)```
const body = {
    mode: "raw",
    raw: jsonWithoutComments,
    options: {
        raw: {
            language: "json"
        }
    }
};

pm.request.body.update(body);

### 3.1.4 How can I modify the request headers?

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

**Add header**

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

**Remove header**

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

**Update header**

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

### 3.1.5 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}}`:

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

This will generate something like:
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 example function that you can use to generate integer number between a specific interval:

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

You can call the function like this:

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

And the output will look similar to:

67

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

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

You can call the function like this:

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

And the output will look similar to:

5q04pes32yi

### 3.1.6 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:

```javascript
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:

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

pm.sendRequest(options, function (error, response) {
    (continues on next page)
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

3.1.7 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.

```javascript
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);
});
```

3.1.8 How to pass arrays and objects between requests?

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

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

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

```javascript
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.

### 3.1.9 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.

### 3.1.10 How to add a delay between Postman requests?

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

```javascript
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/](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
        },
        {
            "id": 102,
            "name": "COUNTRY",
            "isAllowed": true
        },
        {
            "id": 103,
            "name": "MANUFACTURER",
...
```

(continues on next page)
Assert that the property isAllowed is true for the COUNTRY filter.

```javascript
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:

```json
{
    "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:

```javascript
function findObjectContainsLists(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:

```javascript
pm.test("Check status", function () {
    // Parse JSON body
    var jsonData = pm.response.json();
    // Retrieve the lists object
    var lists = findObjectContainsLists(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.

```javascript
// 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:

```javascript
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:

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

(continues on next page)
You want to check that `TargetType` is *Subscriber* or *Customer*.

The assertion can look like:

```javascript
pm.test("Should be subscriber or customer", function () {
    var jsonData = pm.response.json();
    pm.expect(jsonData.ACL[0].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:

```html
<form name="loginForm" action="/login" method="POST">
    <input type="hidden" name="_csrf" value="a0e2d230-9d3e-4066-97ce-flc3dc37915" />
    <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 retrieve the value, we will use the cheerio JavaScript library:

```javascript
// 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:

```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);
```

3.2. Assertions
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)

   ```javascript
   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).

   ```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); // 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 response:

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

You want to assert that a part of the response 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.

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

- string
- number
- boolean
- null
- object
- array

3.3.3 Object has required property

Given this JSON response:
This is the JSON schema with a property named “code” of type String that is mandatory:

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

### 3.3.4 Nested objects

Given this JSON response:

```javascript
{
    "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.

```javascript
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 additional 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.

```javascript
responseHTML = cheerio(pm.response.text());
console.log(responseHTML.find('[name="firstName"]').val());
```
Example fetching a CSRF code form the meta tag:

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

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

**Moment.js**

Moment.js is already built into Postman and can make your life easier if you are working with dates.

Here is a basic usage example:

```javascript
const moment = require('moment');
moment('2021-08-15').add(1, 'days').format('DD.MM.YYYY')
```

This script will parse a date, will add one day and will reformat the date.

Read more: https://momentjs.com/docs

**crypto-js**

Library which implements different crypto functions.

Hash string using SHA256

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

HMAC-SHA1 encryption

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

AES Encryption

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

AES Decryption

```javascript
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:

```javascript
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:

```javascript
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
1,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:

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

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

```javascript
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:

```javascript
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):
3.5.2 How to read links from response and execute a request for each of them?

Given the following response:

```json
{
    "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.

```javascript
// 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 specify a delay in milliseconds.

```
newman run collection.json --delay 10000
```
3.6.2 Jenkins is showing weird 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}. 
4.1 Contributions

This is an open-source project. The code is available on Github: https://github.com/vdespa/postman-quick-reference-guide/

Thank you for all who have contributed to making this guide better.

- Christophe Gesché (@Moosh-be)
- Khanh Do (@khanhdodang)
- Bernard Vander Beken (@jawn)
- Joseph Klos (@joeklos)
- Florian Mayer (@flomayer)
- Zuba Zenovka (@EzikMarconi)
- sweetnoods (@sweetnoods)
- gassims (@gassims)

4.2 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, you can find more details (including a discount here):

https://vdespa.com/courses

Enjoy!