Test examples

getpostman.com/docs/v6/postman/scripts/test_examples

Test scripts are run after a request is sent and a response has been received from the server.

Let’s look at some examples of Postman tests. Most of these are available as snippets inside Postman. You can have as many tests as you want for a request.

Setting an environment variable

```
pm.environment.set("variable_key", "variable_value");
```

Setting a nested object as an environment variable

```
var array = [1, 2, 3, 4];
pm.environment.set("array", JSON.stringify(array, null, 2));

var obj = { a: [1, 2, 3, 4], b: { c: 'val' } };
pm.environment.set("obj", JSON.stringify(obj));
```

Getting an environment variable

```
pm.environment.get("variable_key");
```

Getting an environment variable (whose value is a stringified object)

```
// These statements should be wrapped in a try-catch block if the data is coming from
an unknown source.

var array = JSON.parse(pm.environment.get("array"));
var obj = JSON.parse(pm.environment.get("obj"));
```

Clear an environment variable

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

Set a global variable

```
pm.globals.set("variable_key", "variable_value");
```

Get a global variable

```
pm.globals.get("variable_key");
```

Clear a global variable

```
pm.globals.unset("variable_key");
```

Get a variable

This function searches for the variable across globals and the active environment.

```
pm.variables.get("variable_key");
```

Check if response body contains a string

```
pm.test("Body matches string", function () {
    pm.expect(pm.response.text()).to.include("string_you_want_to_search");
});
```

Check if response body is equal to a string

```
pm.test("Body is correct", function () {
    pm.response.to.have.body("response_body_string");
});
```

Check for a JSON value

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

Content-Type is present

```
pm.test("Content-Type is present", function () {
    pm.response.to.have.header("Content-Type");
});
```

Response time is less than 200ms

```
pm.test("Response time is less than 200ms", function () {
    pm.expect(pm.response.responseTime).to.be.below(200);
});
```

Status code is 200

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

Code name contains a string

```
pm.test("Status code name has string", function () {
    pm.response.to.have.status("Created");
});
```

Successful POST request status code

```
pm.test("Successful POST request", function () {
    pm.expect(pm.response.code).to.be.oneOf([201,202]);
});
```

Use TinyValidator for JSON data

```
var schema = {
 "items": {
 "type": "boolean"
 }
};
var data1 = [true, false];
var data2 = [true, 123];

pm.test('Schema is valid', function() {
  pm.expect(tv4.validate(data1, schema)).to.be.true;
  pm.expect(tv4.validate(data2, schema)).to.be.true;
});
```

Decode base64 encoded data

```
var intermediate,
 base64Content, // assume this has a base64 encoded value
 rawContent = base64Content.slice('data:application/octet-stream;base64,'.length);

intermediate = CryptoJS.enc.Base64.parse(base64content); // CryptoJS is an inbuilt
object, documented here: https://www.npmjs.com/package/crypto-js
pm.test('Contents are valid', function() {
  pm.expect(CryptoJS.enc.Utf8.stringify(intermediate)).to.be.true; // a check for non-
emptiness
});
```

Send an asynchronous request

This function is available as both a pre-request and test script.

```
pm.sendRequest("https://postman-echo.com/get", function (err, response) {
    console.log(response.json());
});
```

Convert XML body to a JSON object

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

Sample data files

JSON files are composed of key/value pairs.

Download JSON file

For CSV files, the top row needs to contain variable names.

Download CSV file

Older style of writing Postman tests

The older style of writing Postman tests relies on setting values for the special `tests` object. You can set a descriptive key for an element in the object and then say if it's true or false. For example, `tests["Body contains user_id"] = responsebody.has("user_id");` will check whether the response body contains the `user_id` string.

You can add as many keys as needed, depending on how many things you want to test for. Under the **Tests** tab under the response viewer, you can view your test results. The tab header shows how many tests passed, and the keys that you set in the tests variable are listed here. If the value evaluates to true, the test passed.

Setting an environment variable

```
postman.setEnvironmentVariable("key", "value");
```

Setting a nested object as an environment variable

```
var array = [1, 2, 3, 4];
postman.setEnvironmentVariable("array", JSON.stringify(array, null, 2));

var obj = { a: [1, 2, 3, 4], b: { c: 'val' } };
postman.setEnvironmentVariable("obj", JSON.stringify(obj));
```

Getting an environment variable

```
postman.getEnvironmentVariable("key");
```

Getting an environment variable (whose value is a stringified object)

```
// These statements should be wrapped in a try-catch block if the data is coming from
an unknown source.

var array = JSON.parse(postman.getEnvironmentVariable("array"));
var obj = JSON.parse(postman.getEnvironmentVariable("obj"));
```

Clear an environment variable

```
postman.clearEnvironmentVariable("key");
```

Set a global variable

```
postman.setGlobalVariable("key", "value");
```

Get a global variable

```
postman.getGlobalVariable("key");
```

Clear a global variable

```
postman.clearGlobalVariable("key");
```

Check if response body contains a string

```
tests["Body matches string"] = responseBody.has("string_you_want_to_search");
```

Convert XML body to a JSON object

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

Check if response body is equal to a string

```
tests["Body is correct"] = responseBody === "response_body_string";
```

Check for a JSON value

```
var data = JSON.parse(responseBody);
tests["Your test name"] = data.value === 100;
```

Content-Type is present (Case-insensitive checking)

```
tests["Content-Type is present"] = postman.getResponseHeader("Content-Type"); //Note:
the getResponseHeader() method returns the header value, if it exists.
```

Content-Type is present (Case-sensitive)

```
tests["Content-Type is present"] = responseHeaders.hasOwnProperty("Content-Type");
```

Response time is less than 200ms

```
tests["Response time is less than 200ms"] = responseTime < 200;
```

Response time is within a specific range (lower bound inclusive, upper bound exclusive)

```
tests["Response time is acceptable"] = _.inRange(responseTime, 100, 1001); // _ is the
inbuilt Lodash v3.10.1 object, documented at https://lodash.com/docs/3.10.1
```

Status code is 200

```
tests["Status code is 200"] = responseCode.code === 200;
```

Code name contains a string

```
tests["Status code name has string"] = responseCode.name.has("Created");
```

Successful POST request status code

```
tests["Successful POST request"] = responseCode.code === 201 || responseCode.code ===
202;
```

Use TinyValidator for JSON data

```
var schema = {
 "items": {
 "type": "boolean"
 }
};
var data1 = [true, false];
var data2 = [true, 123];

tests["Valid Data1"] = tv4.validate(data1, schema);
tests["Valid Data2"] = tv4.validate(data2, schema);
console.log("Validation failed: ", tv4.error);
```

Decode base64 encoded data

```
var intermediate,
 base64Content, // assume this has a base64 encoded value
 rawContent = base64Content.slice('data:application/octet-stream;base64,'.length);

intermediate = CryptoJS.enc.Base64.parse(base64content); // CryptoJS is an inbuilt
object, documented here: https://www.npmjs.com/package/crypto-js
tests["Contents are valid"] = CryptoJS.enc.Utf8.stringify(intermediate); // a check for
non-emptiness
```