

Postman Test Automation Using Newman

- **Newman** is an open-source command-line tool (built on Node.js) for running and testing Postman Collections outside the Postman GUI
- It supports the same pre-request and test scripts as Postman, making it ideal for automation and CI pipelines.

Overview

This guide provides step-by-step instructions for:

- Writing Pre-request and Post-response scripts in Postman.
- Running the API test collections using Newman, the CLI companion tool for Postman.
- Generating HTML test reports.

Prerequisites

- Postman installed: <https://www.postman.com/downloads/>
- Node.js installed: <https://nodejs.org/>
- Newman installed via npm:

```
npm install -g newman
```

1. Creating Pre-request and Test Scripts in Postman

1.1 Create a Collection and Add Request

- Open Postman.
- Click "Collections" → "New Collection" → Name it.
- Add a request (e.g., a GET or POST request) to your collection.

1.2 Pre-request Script

A Pre-request Script is JavaScript code that executes *before* an API request is sent. It enables you to set up variables, generate dynamic data, and configure request parameters such as headers or authentication tokens.

Steps:

1. Go to your request → Click on the "Pre-request Script" tab.
2. Add your script.

	Params	Authorization	Headers (11)	Body	Scripts ●	Settings
Pre-request					<pre> 1 // parse the headers 2 const headers = pm.request.headers.reduce((acc, obj) => { 3 acc[obj.key] = obj.value; 4 return acc; 5 }, {}); 6 7 // check if headers are present and have all required properties 8 if(headers && Object.keys(headers).length > 0) { 9 pm.test("Request header includes all the required properties: source, client_id, client_secret", function() 10 pm.expect(headers).to.include.all.keys('source', 'client_id', 'client_secret'); 11 }); 12 13 } else { 14 pm.test("Request does not contain certain headers or headers are empty", function() { 15 pm.expect.fail("Expected request to contain headers array with elements"); 16 }); 17 } </pre>	
Post-response						

1.3 Post-response Script

Also known as a Test Script, this JavaScript runs *after* a response is received. It's used to verify outcomes, assert conditions on status codes, headers, response time, and body content.

Steps:

1. Go to your request → Click on the "Post-response Script" tab.
2. Add your script.

	Params	Authorization	Headers (11)	Body	Scripts ●	Settings
Pre-request						
Post-response					<pre> 1 // Test to check if the response status code is one of the expected values 2 pm.test("Status code is " + pm.response.code, function() { 3 pm.expect(pm.response.code).to.be.oneOf([200, 201, 202]); 4 }); 5 6 // Parse the response JSON 7 const responseData = pm.response.json(); 8 9 if(responseData && Object.keys(responseData).length > 0) { 10 11 // Test to check if response body contain 'headers' and 'records' properties 12 pm.test("Response body contains all required properties", function () { 13 pm.expect(responseData).to.have.all.keys('headers', 'records'); 14 }); 15 16 // Test to check if response field: 'header' has all required properties: 'batchRunRecordId', 'batchId', 'recordType 17 pm.test("Response field: 'headers' has all required properties: 'batchRunRecordId', 'batchId', 'recordType', 'batchR 18 pm.expect(responseData.headers).to.have.all.keys('batchRunRecordId', 'batchId', 'recordType', 'batchRunStatus'); 19 }); 20 21 // Test to check if recordType is MasterCustomer 22 pm.test("Record type is 'MasterCustomer'", function () { 23 pm.expect(responseData.headers.recordType).to.equal('MasterCustomer'); 24 }); 25 </pre>	

2. Exporting Collection

Before using Newman, export your Postman collection:

- Go to Collections → Hover over your collection → Click "..." → "Export"
- Select Collection v2.1 (recommended) → Save as collection.json

Also export environment (if needed):

- Go to Environments → Select the environment → Click "..." → "Export"

3. Running Tests with Newman

3.1 Basic Run :

```
newman run collection.json
```

3.2 With Environment :

```
newman run collection.json -e environment.json
```

3.3 Generate HTML Report :

```
npm install -g newman-reporter-html
```

3.4 Run with HTML report :

```
newman run collection.json -e environment.json -r html
```

Newman Report

Collection	NetSuite SYS API		
Description	- This is the NetSuite System API that has one endpoint "records" that works with the POST method only. - It is protected with Client ID enforcement, the following headers are supposed to be passed. - client_id: The unique identifier for the client application, issued by the authorization. - client_secret: The confidential key associated with the client_id that will be used to authenticate the application. - One mandatory header is being passed, specifying the NetSuite object we are dealing with. - object_type: Specifies object type. - LogOnSuccessEnableFlag is a non-mandatory header, for which there can be 2 values: - true: if the value is true, then the records will be upserted to the recordLog object in NetSuite. - false: if the value is false, then the records will be upserted into the recordLog object only when NetSuite upsertion errors.		
Time	Tue Jun 24 2025 13:02:05 GMT+0530 (India Standard Time)		
Exported with	Newman v6.2.1		
	Total		Failed
Iterations	1		0
Requests	10		0
Prerequisite Scripts	8		0
Test Scripts	10		0
Assertions	40		0
Total run duration		18.1s	
Total data received		2.89KB (approx)	
Average response time		1726ms	
Total Failures	0		
Requests			
customer			

Upsert Records customer

Method POST
URL <https://dev-sys-netsuite-app-k6owae.6euqcy.usa-e1.cloudhub.io/api/v1/franchise/customer>

Mean time per request 2.2s
Mean size per request 272B

Total passed tests 6
Total failed tests 0

Status code 200

Tests

Name	Pass count	Fail count
Header includes all the required properties: client_id, client_secret, record_log_on_success_enable_flag	1	0
Record type is Customer	1	0
All records have recordType, batchRunStatus, recordId, customerId, parentCompany properties	1	0
Status code is 200	1	0
Upsert status is successful	1	0
All Customer records are upserted successfully	1	0