

Using POSTMAN with REST API

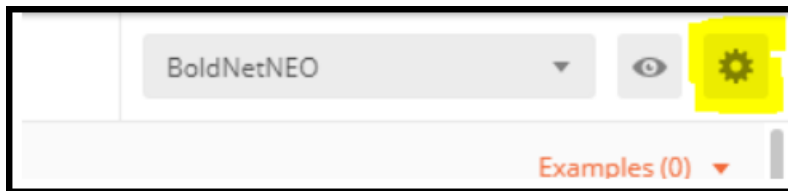
05/27/2025 1:26 pm EDT

Overview

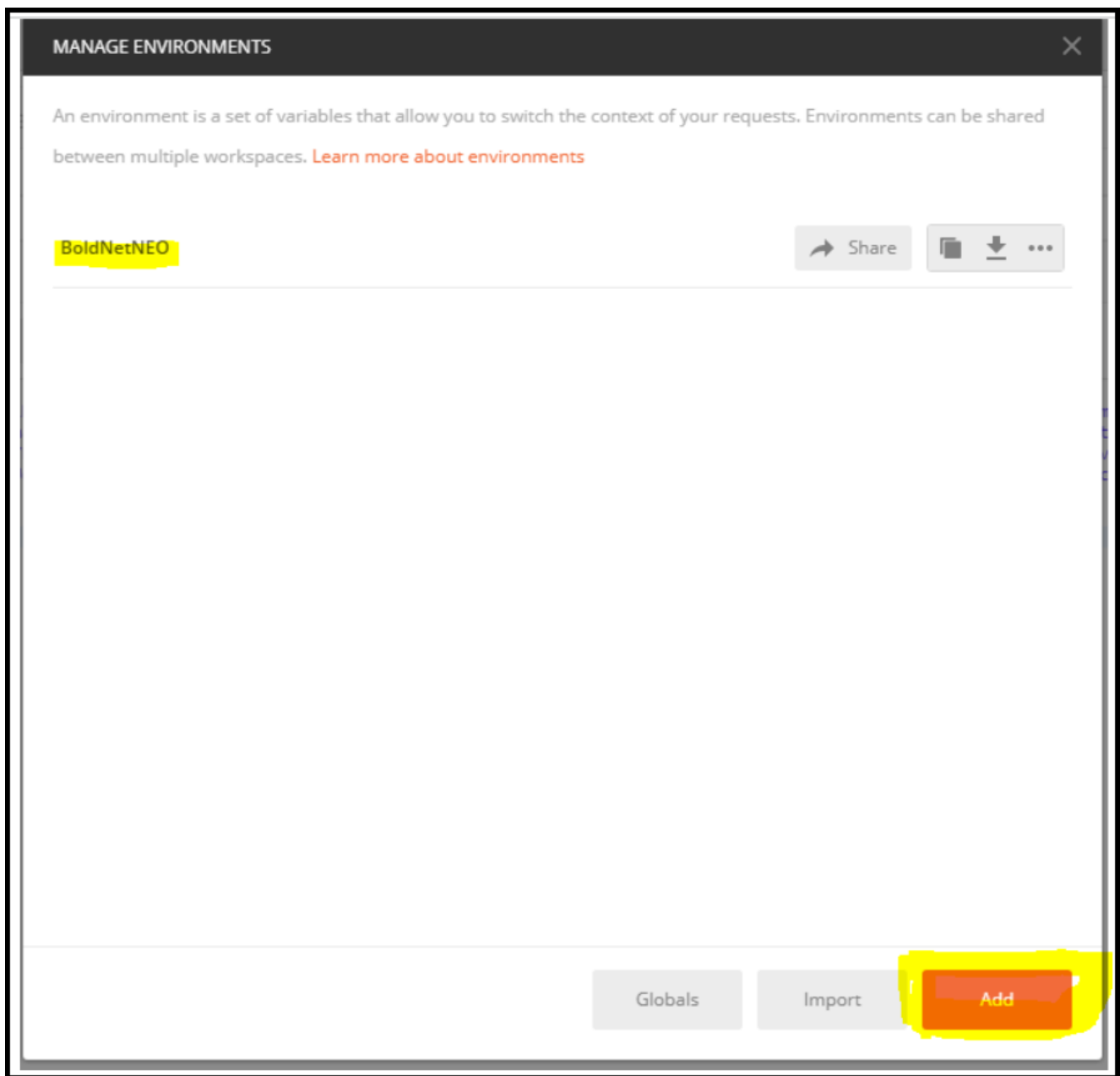
The Manitou REST API is used with BoldNet NEO and ManitouNEO. This provides a modern interface that is used by our web based Client UI but can also be used by third party integrations. Because it uses HTML calls to an IIS server, it allows very quick testing through several tools.

Setup Environment

Once you have downloaded and installed POSTMAN, the first thing that is needed is to set up an Environment and a variable to store the OAUTH token. This is found in the top right corner of the POSTMAN IDE.



Add a new Environment and call it BoldNetNEO.



Click on the new environment and Add a variable called "token". Don't worry about populating the values right now.

MANAGE ENVIRONMENTS

Environment Name

BoldNetNEO

	VARIABLE	INITIAL VALUE ⓘ	CURRENT VALUE ⓘ	...	Persist All	Reset All
☒	token					
	Add a new variable					

ⓘ Use variables to reuse values in different places. The current value is used while sending a request and is never synced to Postman's servers. The initial value is auto-updated to reflect the current value. [Change this](#) behaviour from Settings. [Learn more about variable values](#)

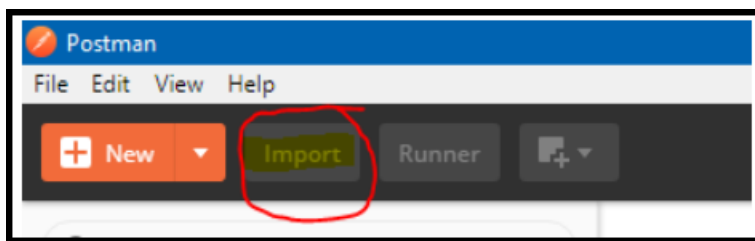
Cancel

Update

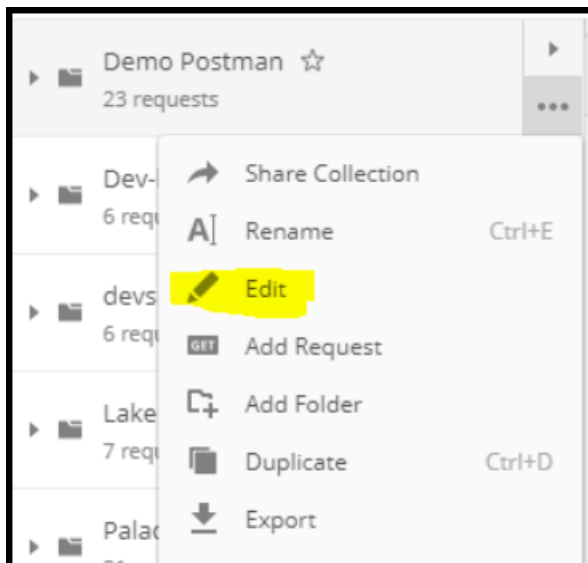
You can now use the X at the top right to return.

Importing Demo Collection

Click on the Import button on the top left part of the POSTMAN interface. Drag and drop the attached .JSON file for the demo collection.



Once this collection is imported, right click on the collection and select Edit.



Update the Variables settings for url, username, and password with your correct values for BoldNet NEO and click the update button at the bottom of the form. You may need to hit the Reset All button if there are current values already in place.

NOTE: The password cannot contain an ampersand '&' sign.

EDIT COLLECTION

Name
Demo Postman

Description
Authorization
Pre-request Scripts
Tests
Variables

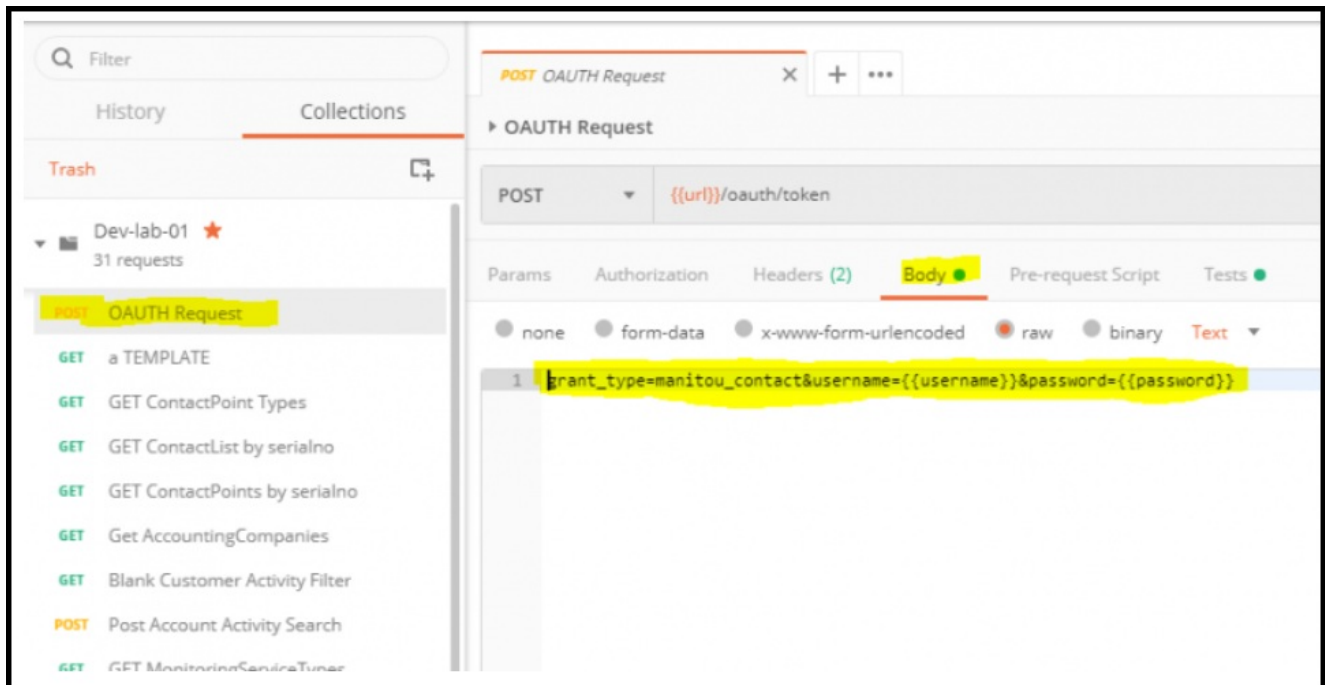
These variables are specific to this collection and its requests. [Learn more about collection variables.](#)

	VARIABLE	INITIAL VALUE ⓘ	CURRENT VALUE ⓘ	...	Persist All	Reset All
☒	url	https://addyourpath	https://addyourpath			
☒	username	username				
☒	password	password				X ...
	Add a new variable					

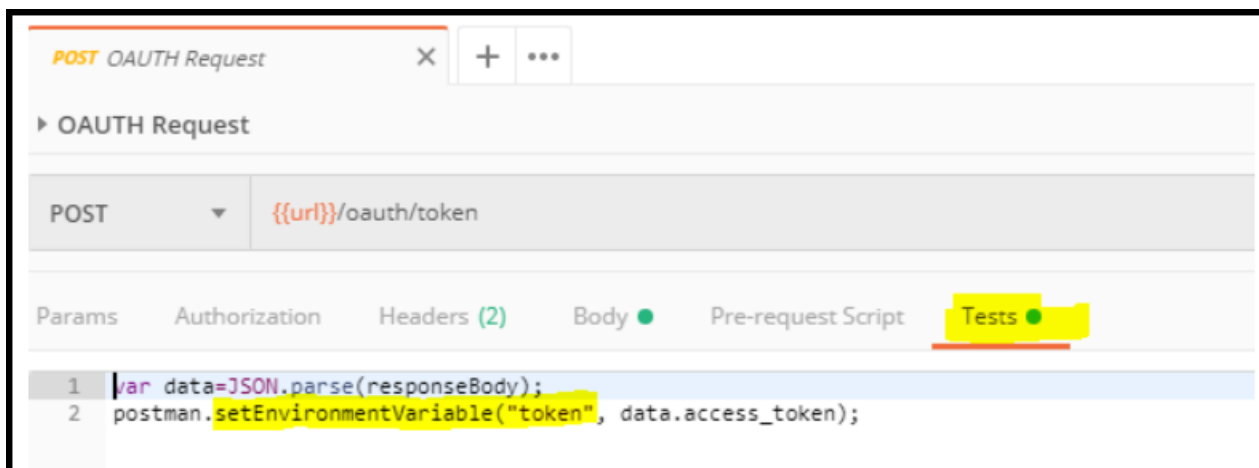
You are now set up and ready to begin working with API calls. You can rename the collection to better describe the Manitou system you are hitting.

Getting Authorized with OAUTH

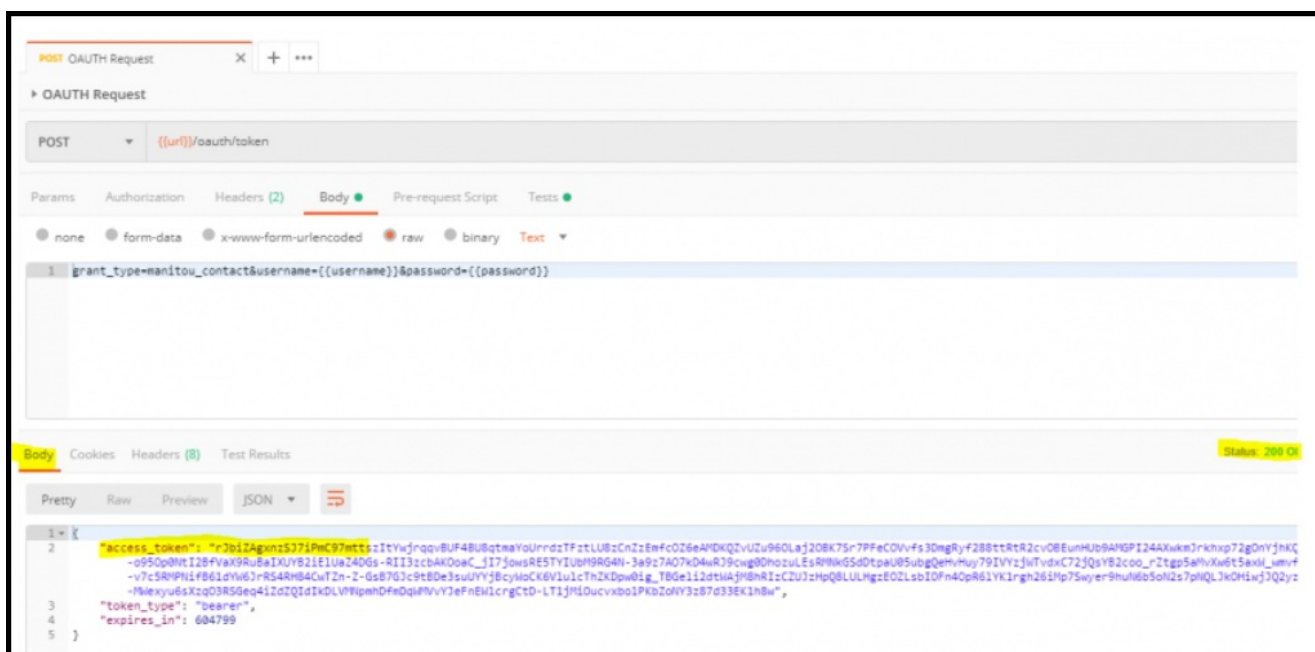
The first thing that must happen before any API calls can be made is to get an OAUTH token for your user. There is a pre-built script to do this under the collection. Expand the collection and click on the OAUTH Request POST link. Under the Body section, you can see that we pass in the username and password.



The Tests section has a script that will store the returned OAUTH token in the environment variable so that it can be easily used in subsequent calls.



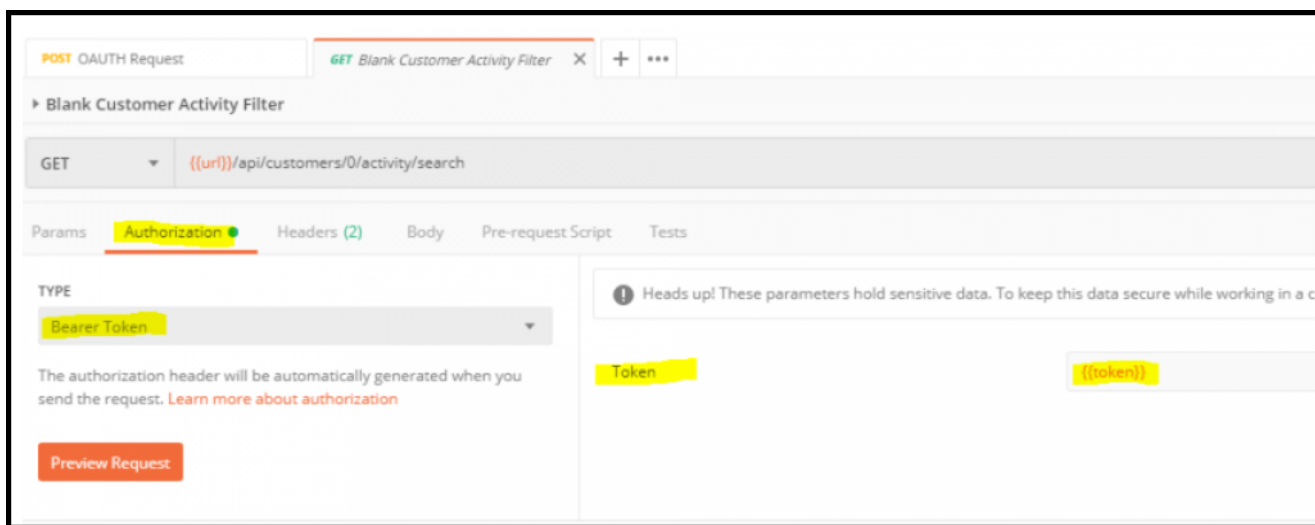
Hit the Send button and you should get an OAUTH token back. Once you have this working you can move on to the next steps.



Making API Calls

Now that you have a valid OAUTH token, we can hit the various API endpoints and make requests. An easy one to start with is requesting a blank Customer Activity Filter. This returns an object that you can use when making calls to get customer Activity. But I use it just to make sure the basic connection is working.

Open the Blank Customer Activity Filter script. This already has the OAUTH token setup to be used under the Authorization tab.



This request does not require a body message so you can just hit send. It should return a JSON array object showing that the connection is working to the API including sending in the OAUTH token.



The screenshot shows a web browser's developer tools interface. The 'Body' tab is selected, displaying a JSON response. The JSON is formatted in 'Pretty' mode. The response is a large object with many fields, including 'allowedLogTypes', 'area', 'countryId', 'dateFrom', 'dateTo', 'device', 'eventCategories', 'eventNo', 'eventSource', 'falseAlarmCode', 'format', 'includeFalseAlarms', 'includeTimeRange', 'invertSearch', 'lastSequence', 'logSequence', 'logTypes', 'pageCount', 'pages', 'policed', 'relatedObjectTypes', 'returnAllRows', 'reverseOrder', 'rldes', 'searchId', 'sector', 'sensor', 'serialNo', 'sessionId', 'standardCodes', 'systemNo', 'timeFrom', 'timeTo', 'transmitterNumber', 'txId', 'utcDateFrom', 'utcDateTo', 'utcTimeFrom', 'utcTimeTo', and 'zone'. The 'logTypes' field is highlighted with a mouse cursor. The line numbers 194 through 259 are visible on the left side of the JSON editor.

```
1 {
2   "allowedLogTypes": [
194   "area": "",
195   "countryId": 0,
196   "dateFrom": "0001-01-01T00:00:00",
197   "dateTo": "0001-01-01T00:00:00",
198   "device": "",
199   "eventCategories": [],
200   "eventNo": 0,
201   "eventSource": "",
202   "falseAlarmCode": "",
203   "format": 0,
204   "includeFalseAlarms": false,
205   "includeTimeRange": false,
206   "invertSearch": false,
207   "lastSequence": 0,
208   "logSequence": 0,
209   "logTypes": [
236   "pageCount": 1000,
237   "pages": 0,
238   "policed": false,
239   "relatedObjectTypes": [],
240   "returnAllRows": false,
241   "reverseOrder": false,
242   "rldes": "",
243   "searchId": 0,
244   "sector": "",
245   "sensor": "",
246   "serialNo": 0,
247   "sessionId": 0,
248   "standardCodes": [],
249   "systemNo": 0,
250   "timeFrom": "0001-01-01T00:00:00",
251   "timeTo": "0001-01-01T00:00:00",
252   "transmitterNumber": "",
253   "txId": "",
254   "utcDateFrom": "0001-01-01T00:00:00",
255   "utcDateTo": "0001-01-01T00:00:00",
256   "utcTimeFrom": "0001-01-01T00:00:00",
257   "utcTimeTo": "0001-01-01T00:00:00",
258   "zone": ""
259 }
```

You can now work through the other API calls.

Many of the API calls have variables in the URL Path as well as data that must be sent in the Body of the message. These vary by the endpoint so you must refer to the documentation on the structure. The demo collection has some of my values and must be updated accordingly. For example, the Get Customer by ID is calling my customer.

POST OAUTH Request

GET Blank Customer Activity Filter

GET Get Customer by ID

X

+

...

▶ Get Customer by ID

GET

▼

{{url}}/api/customers?id=TOBY

Params

Authorization

Headers (2)

Body

Pre-request Script

Tests

	KEY	VALUE
☒	id	TOBY
	Key	Value

Response