



Tableau 10.0 REST API & Postman

This document is provided for information purposes only.



Summary

Tableau enables programmatic management of server resources through a [REST](#) API. Postman is a developer that enables rapid test and development of API's. This document will outline how to use these two products to explore usage of the Tableau REST API's.

Intended usage

Rapid prototyping of Tableau Server management through a programmatic interface

Demo the capabilities of the API

Training yourself or others to use the API

Training and Resources

1. Tableau Server API Documentation - https://onlinehelp.tableau.com/current/api/rest_api/en-us/help.htm
2. Tableau Training Video - <http://www.tableau.com/learn/tutorials/on-demand/rest-api>

Requirements

Access to Tableau Server 10.0 or higher. This samples are developed on the Tableau 10.0 REST API. For Postman API on ver 9.0:

https://github.com/TableauExamples/Tableau_Postman.

Tableau Server (on premise or installed in a virtual machine) can utilize the full API's. Tableau Online has a limited set of API's available.

http://onlinehelp.tableau.com/current/api/rest_api/en-us/REST/rest_api_ref.htm

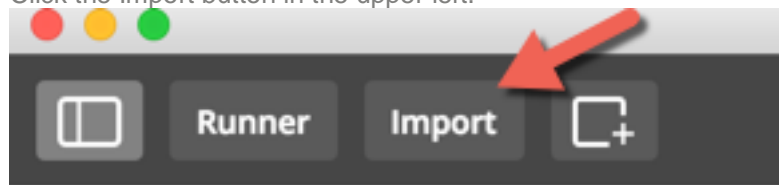
- The Tableau API is enabled. It is enabled by default.
<http://onlinehelp.tableau.com/current/server/en-us/tabadmin.htm>
- A Tableau user with the appropriate security level for the actions you want to implement. Some actions can only be taken by a Server Administrator.

Installation Instructions

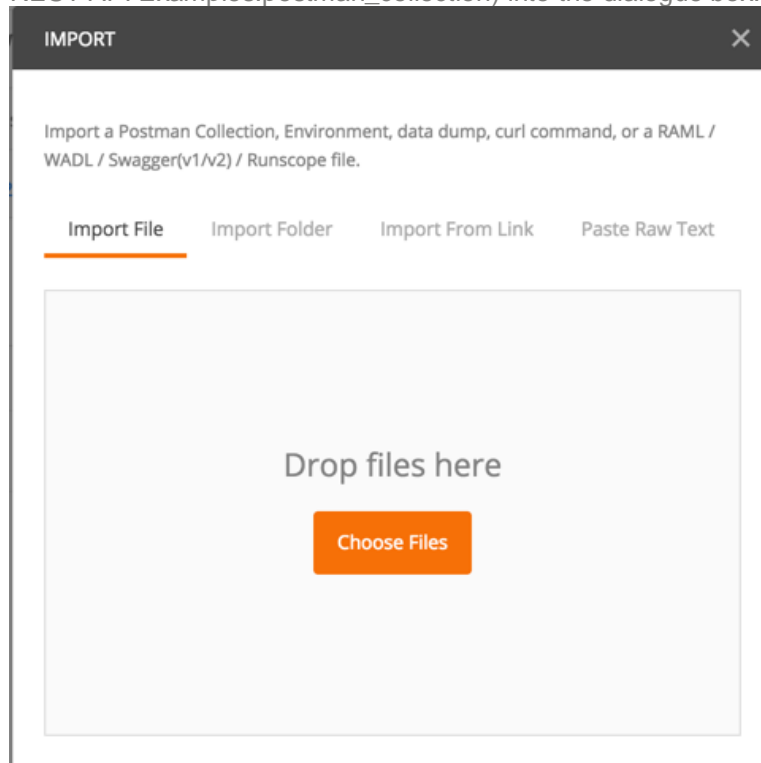
1. Download and install Postman (<http://getpostman.com>). For the purposes of this document, you can install the browser based version or the standalone version. The browser based version allows additional debugging and discovery features not covered in this document.



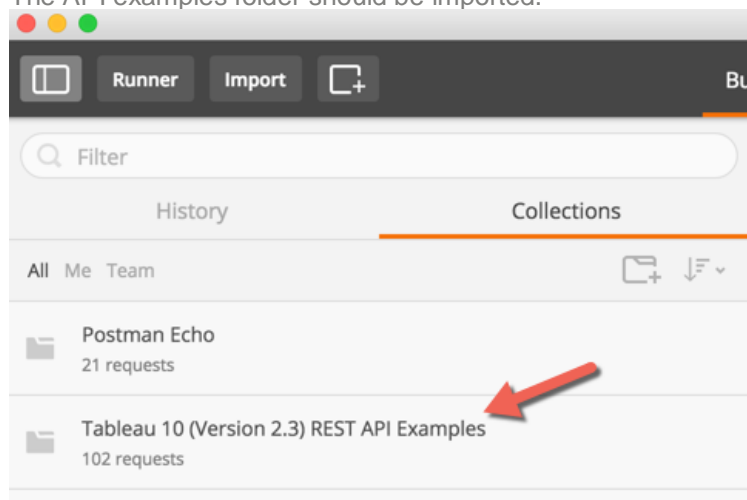
2. Click the Import button in the upper left.



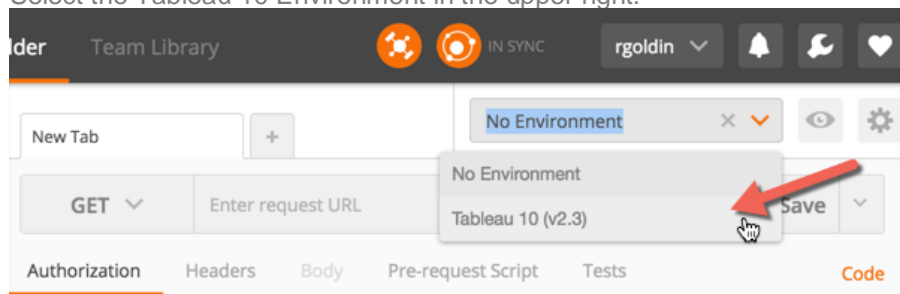
- Drop the two file (Tableau 10 (v2.3).postman_environment and Tableau 10 (Version 2.3) REST API Examples.postman_collection) into the dialogue box.



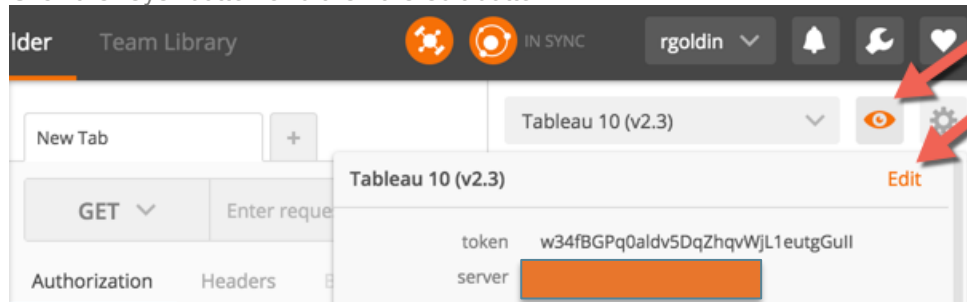
- The API examples folder should be imported.



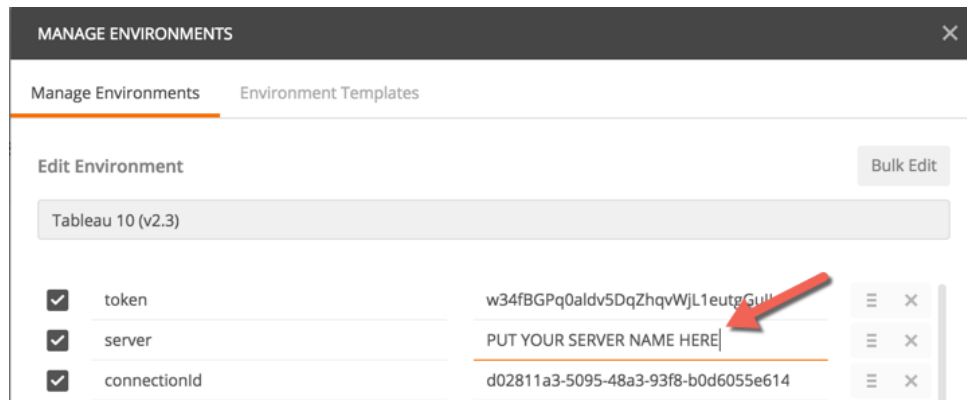
- Select the Tableau 10 Environment in the upper right.



- Click the “eye” button and then the edit button.



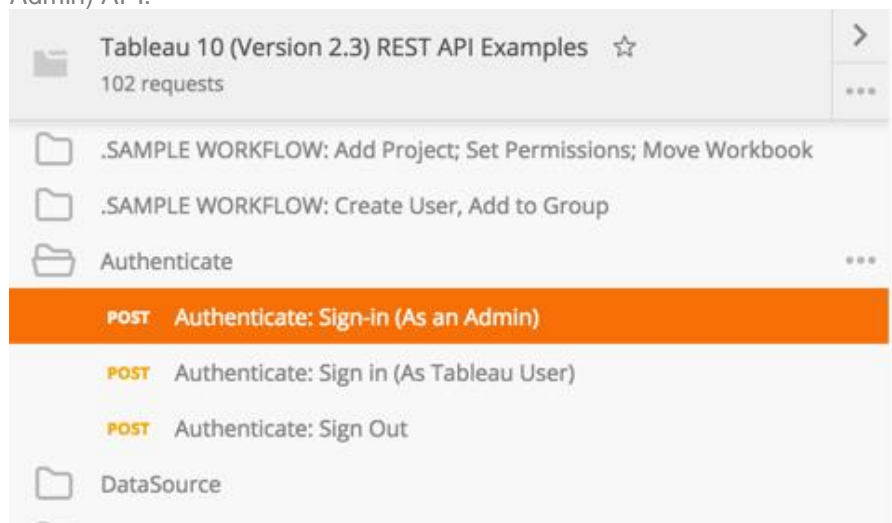
- Replace the current server name with your Tableau Server name (as called from a web browser).



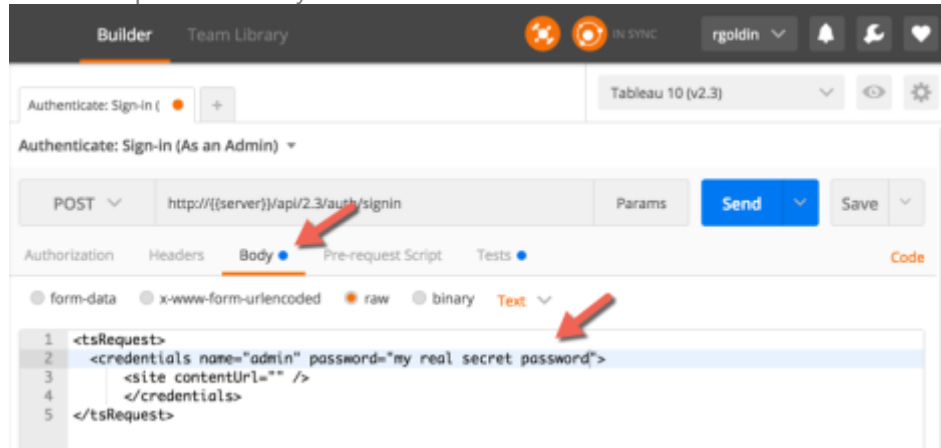
Test the login API

To ensure that Postman can connect to your server, test the Login API.

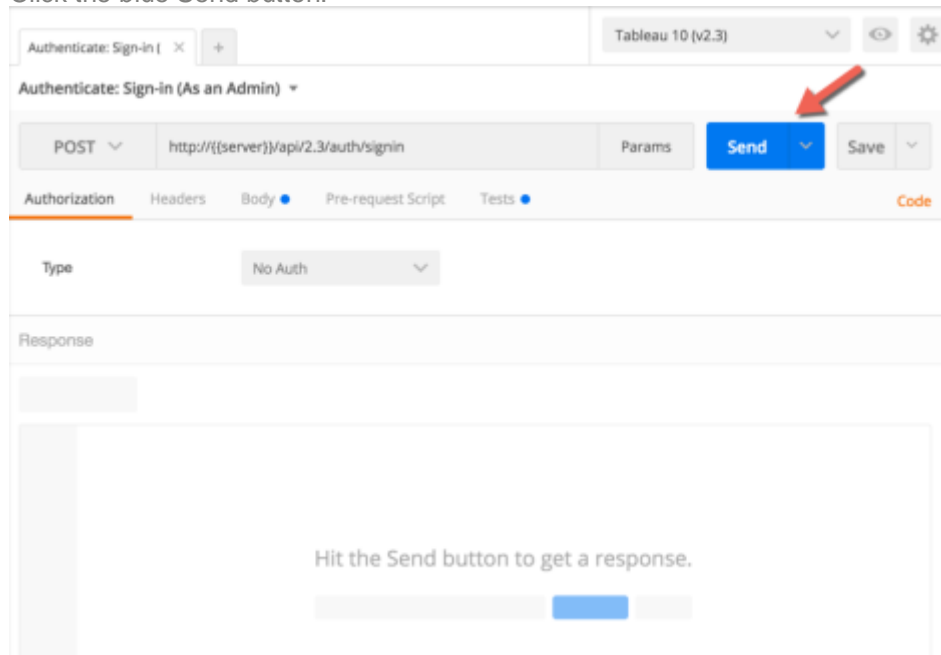
- Expand the Tableau 10 folder, and the Authenticate folder, and select the “Sign-in (As an Admin) API.



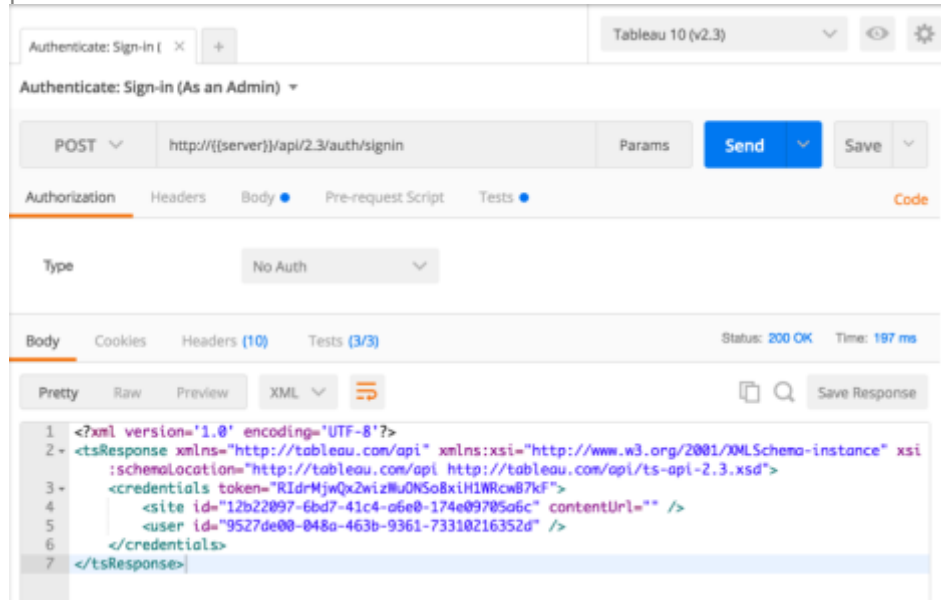
2. The login API will open in a Tab in the main pane. Click on the Body tab and edit your username/password for your Site Administrator.



3. Click the blue Send button.



4. If the call is successful, you will see the body of the response message in the Response pane at the bottom.



Usage

The right-hand pane has folders that hold all of the API calls. These loosely match the REST API documentation, but the folders are more granular. To make any call, simply click on it and it will open in the right pane and then click Run.

Variables

Postman allows environment variables to be set manually and programmatically. These are represented by two open curly braces, the variable name and two close curly braces like: `{{variable}}`

This enables many workflows to be executed with only the need to click the API and then click send.

Manually vs programmatically setting the variables

The best way to see what is set is to check the Tests tab in the Response pane.

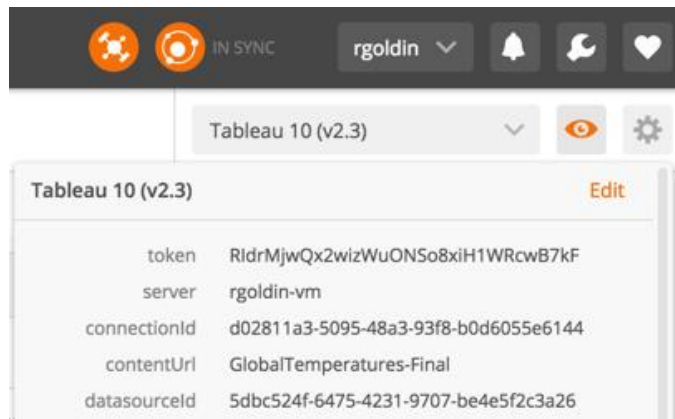
The screenshot shows the Postman interface for a POST request to `http://{{server}}/api/2.3/auth/signin`. The request body is raw text containing XML credentials. The Tests tab is selected, showing three passed tests that set environment variables for token, siteId, and userId. A red arrow points to the Tests tab.

```
1 <tsRequest>
2   <credentials name="admin" password="my real secret password">
3     <site contentUrl="" />
4   </credentials>
5 </tsRequest>
```

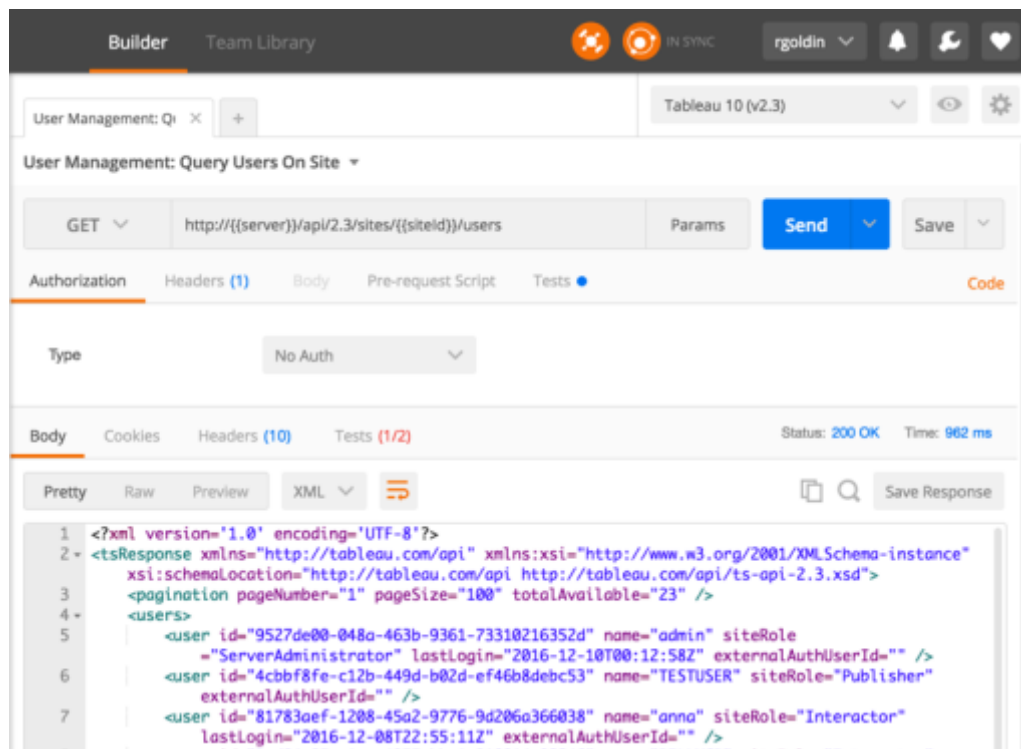
Tests (3/3)

- PASS token env variable set = RldrMjwQx2wizWuONSo8xiH1WRcwB7kF
- PASS siteId env variable set = 12b22097-6bd7-41c4-a6e0-174e09705a6c
- PASS userId env variable set = 9527de00-048a-463b-9361-73310216352d

When the test says “... env variable set = ...” then variable will be automatically set. The “eye” button will allow you to verify what was set.



Typically, where the call returns multiple results such as 'Query Users on Site' or 'Query Workbooks on Site' the Test will remind you to manually pick which variables you want to set.



Builder Team Library IN SYNC rgoldin

User Management: Query Users On Site

GET http://{{server}}/api/2.3/sites/{{siteid}}/users Params Send Save

Authorization Headers (1) Body Pre-request Script Tests Code

Type No Auth

Body Cookies Headers (10) Tests (1/2) Status: 200 OK Time: 962 ms

All Passed Failed

PASS User list received

FAIL Please select userId and userName for subsequent calls

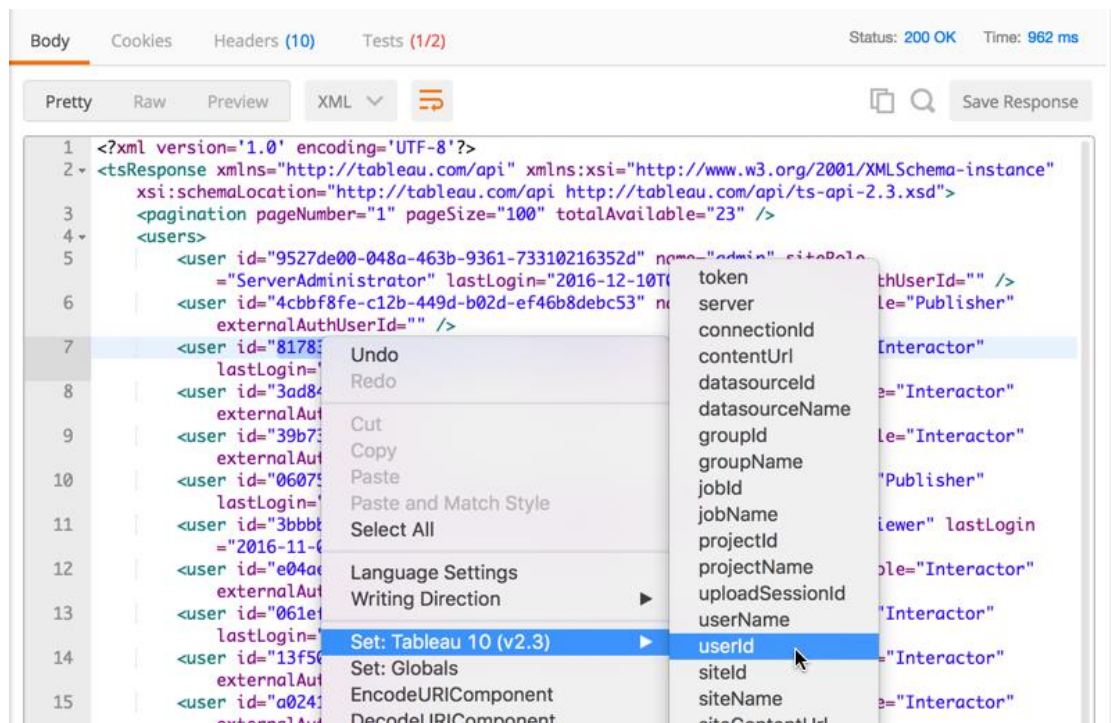
To manually select a user (or workbook, view, site, project, etc), click on the Body tab.

Body Cookies Headers (10) Tests (1/2) Status: 200 OK Time: 962 ms

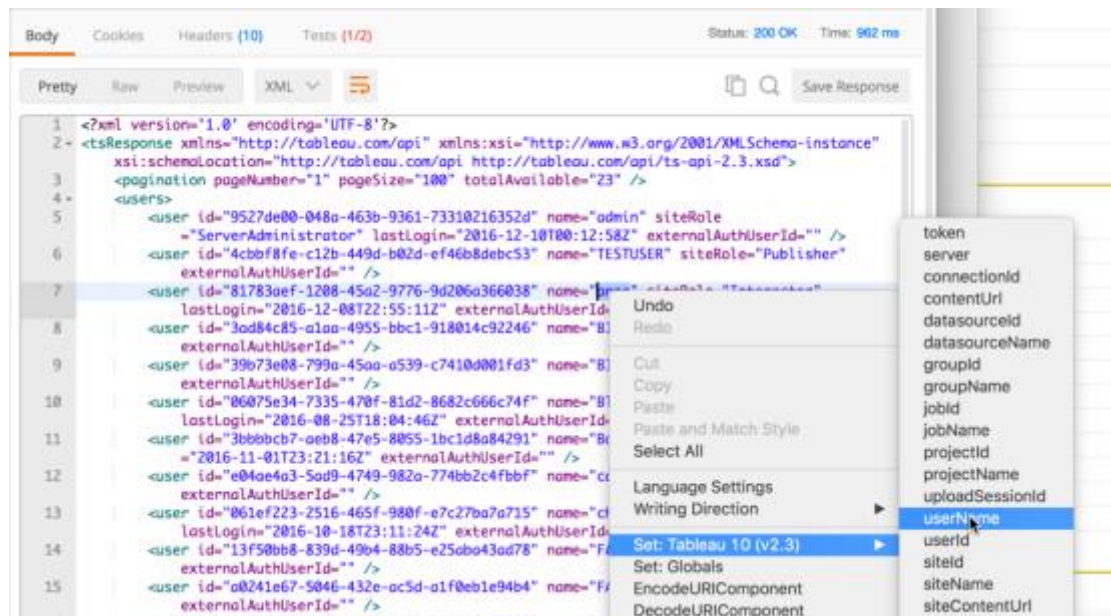
Pretty Raw Preview XML Save Response

```
<?xml version='1.0' encoding='UTF-8'?>
<tsResponse xmlns="http://tableau.com/api" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://tableau.com/api http://tableau.com/api/ts-api-2.3.xsd">
  <pagination pageNumber="1" pageSize="100" totalAvailable="23" />
  <users>
    <user id="9527de00-048a-463b-9361-73310216352d" name="admin" siteRole="ServerAdministrator" lastLogin="2016-12-10T00:12:58Z" externalAuthUserId="" />
    <user id="4cbbf8fe-c12b-449d-b02d-ef46b8debc53" name="TESTUSER" siteRole="Publisher" externalAuthUserId="" />
    <user id="81783aef-1208-45a2-9776-9d206a366038" name="anna" siteRole="Interactor" lastLogin="2016-12-08T22:55:11Z" externalAuthUserId="" />
    <user id="3ad84c85-a1aa-4955-bbc1-918014c92246" name="BILLMGR" siteRole="Interactor" externalAuthUserId="" />
    <user id="39b73e08-799a-45aa-a539-c7410d001fd3" name="BILLUSER" siteRole="Interactor" externalAuthUserId="" />
    <user id="06075e34-7335-470f-81d2-8682c666c74f" name="Bloom" siteRole="Publisher" lastLogin="2016-08-25T18:04:46Z" externalAuthUserId="" />
  </users>
</tsResponse>
```

For this example, we want to see more details about Anna. Highlight her entire ID string and right click to bring up the context menu. Select the Environment and then the matching variable.



Every time you set an ID environment variable, also set the name variable to the matching name of the ID you are acting on.



This is not necessary for the Tableau API calls. However, it should be a best practice so you can easily identify what pairs of IDs/Names you are working on from the environment variables "eye".

BuilderTeam Library

rgoldin

User Management: Query Users On Site

Tableau 10 (v2.3)

GET

http://{{server}}/api/2.3/sites/{{siteid}}/users

Authorization

Headers (1)

Body

Pre-request Script

Tests

☒

X-Tableau-Auth

{{token}}

☐

key

value

☐

key

value

☐

key

value

☐

key

value

Body

Cookies

Headers (10)

Tests (1/2)

Pretty

Raw

Preview

XML

1

<?xml version="1.0" encoding="UTF-8"?>

2

<tsResponse xmlns="http://tableau.com/api" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://tableau.com/api http://tableau.com/api/tableau.xsd">

3

<pagination pageNumber="1" pageSize="100" totalAvailable="100">

4

<users>

5

<user id="9527de00-048a-463b-9361-73310216352d" lastLogin="2016-12-10T00:12:58Z" externalAuthUser="false">

6

<user id="4cbbf8fe-c12b-449d-b02d-ef46b8debc53" name="TESTUSER" siteRole="Publisher">

7

<user id="81783aef-1208-45a2-9776-9d206a366038" name="anna" siteRole="Interactor" lastLogin="2016-12-10T00:12:58Z">

8

<user id="3ad84c85-a1aa-4955-bbc1-918014c92246" name="BILLMGR" siteRole="Interactor">

token

RldrMjwQx2wizWuONS08xiH1WRcwB7kF

server

rgoldin-wm

connectionid

d02811a3-5095-48a3-93f8-b0d6055e6144

contentUrl

GlobalTemperatures-Final

datasourceid

5dbc524f-6475-4231-9707-be4e5f2c3a26

datasourceName

CoffeeChainCoffee

groupid

35073d99-241e-4f97-a5a5-fa86bf54ba6b

groupName

myNewGroup

jobid

notSet

jobName

notSet

projectid

a7dd5364-1d56-4835-ae73-9ddfc9ba6744

projectName

RestAPI Project

uploadSessionId

15973:B1E348668D194C8EA8C12284C3A57F24-0:0

username

anna

userid

81783aef-1208-45a2-9776-9d206a366038

siteid

12b22097-6bd7-41c4-a6e0-174e0970536c

siteName

anna

siteContentUrl

tag

notSet

moduleid

4547c808-d038-459d-bafb-17793ba13ef0

Good to know...

1. The API calls have a subset of the information that is available in the Online documentation. To see it, click the collapse/expand triangle next to the API call name.

The screenshot shows the Tableau API Builder interface. At the top, there's a header with 'Builder', 'Team Library', and user information 'rgoldin'. Below the header, there's a search bar with 'User Management: Qi' and a '+' button. To the right, there's a dropdown for 'Tableau 10 (v2.3)'. The main content area shows the API call 'User Management: Query Users On Site' with a collapse/expand triangle. Below the call name, there's a description 'Get all users in site' and 'URIs'. A list of four GET URIs is provided: 1. GET /api/api-version/sites/site-id/users, 2. GET /api/api-version/sites/site-id/users?filter=filter-expression, 3. GET /api/api-version/sites/site-id/users?sort=sort-expression, 4. GET /api/api-version/sites/site-id/users?pageSize=page-size&pageNumber=page-number. Below the URIs, there's a 'Parameters' section with a list of parameters: api-version, site-id, filter-expression, sort-expression, page-size, and page-number. At the bottom, there's a 'GET' dropdown, a URL field with 'http://{{server}}/api/2.3/sites/{{siteid}}/users', a 'Params' button, a 'Send' button, and a 'Save' button.

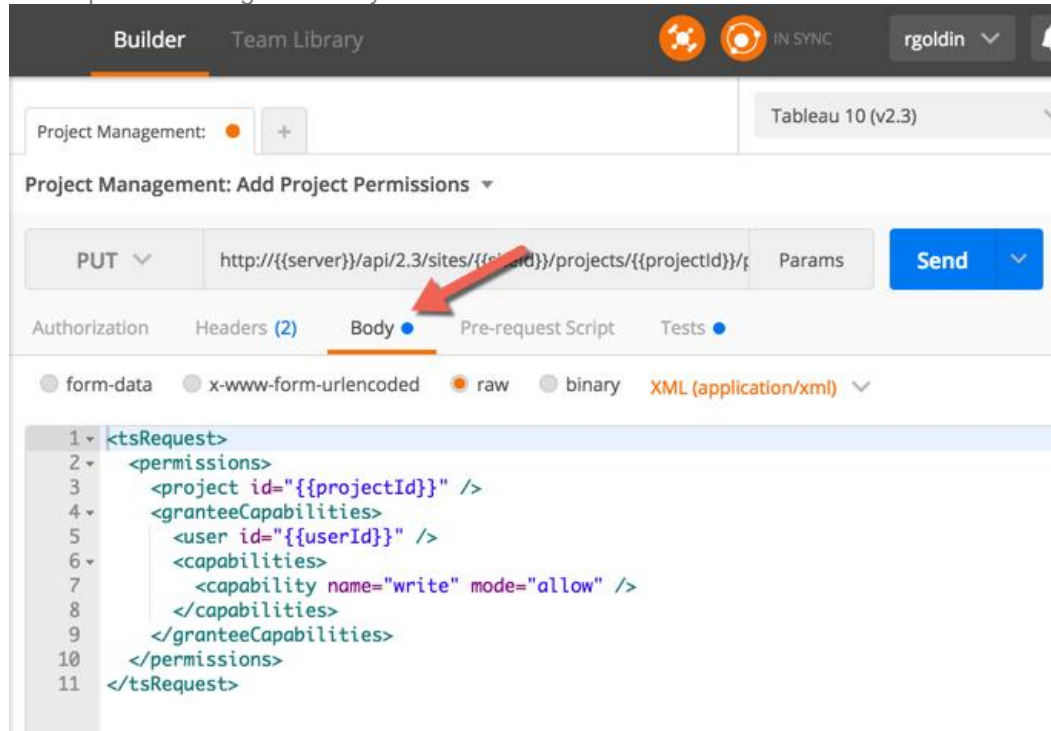
This call has 4 variations that can be used.

2. The type of call is listed in the folder.

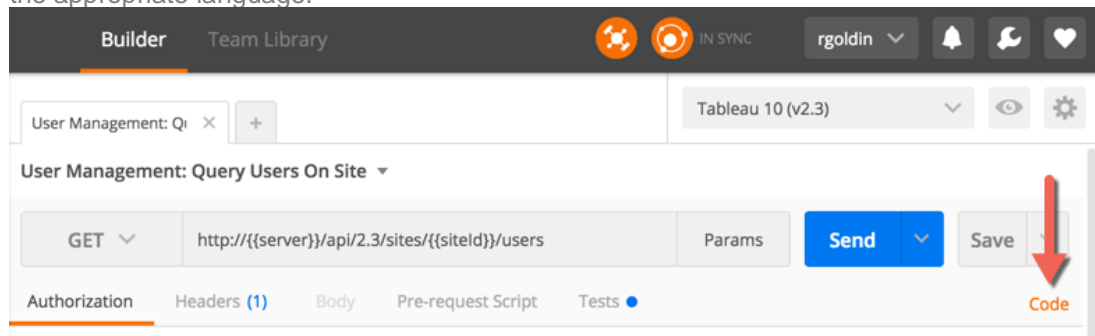
The screenshot shows a folder named 'Users' in the Tableau API Builder interface. The folder contains a list of API calls, each with a color-coded HTTP method icon and the call name. The calls are: GET User Management: Query Users On Site (highlighted in orange), GET User Management: Query User on Site, POST User Management: Add user to site, DEL User Management: Remove User from Site, GET User Management: Query Workbooks for a User, PUT User Management: Update User, POST User Management: Add User to Group, GET User Management: Get Users in Group, and DEL User Management: Remove User from Group.

GET = Ask the server for information
POST = Add information
PUT = Update information
DEL = Delete something

3. When calling a PUT or POST you will be providing information to the server. This is accomplished through the Body tab in the form of an XML document.



4. Now that you see how the API(s) works, you'll want to see how to implement it in your favorite programming language. Click the Code button and then select the drop-down for the appropriate language.




Python http.client (Python 3) ▾

Copy to Clipboard

```
1 import http.client
2
3 conn = http.client.HTTPConnection("rgoldin-vm")
4
5 headers = {
6     'x-tableau-auth': "RIIdrMjwQx2wizWuONSo8xiH1WRcwB7kF",
7     'cache-control': "no-cache",
8     'postman-token': "ab350850-c80b-c6fe-9cb1-651850bff0c3"
9 }
10
11 conn.request("GET", "/api/2.3/sites/12b22097-6bd7-41c4-a6e0-174e09705a6c/users",
12             headers=headers)
13
14 res = conn.getresponse()
15 data = res.read()
16 print(data.decode("utf-8"))
```

Python http.client (Python 3) ▾

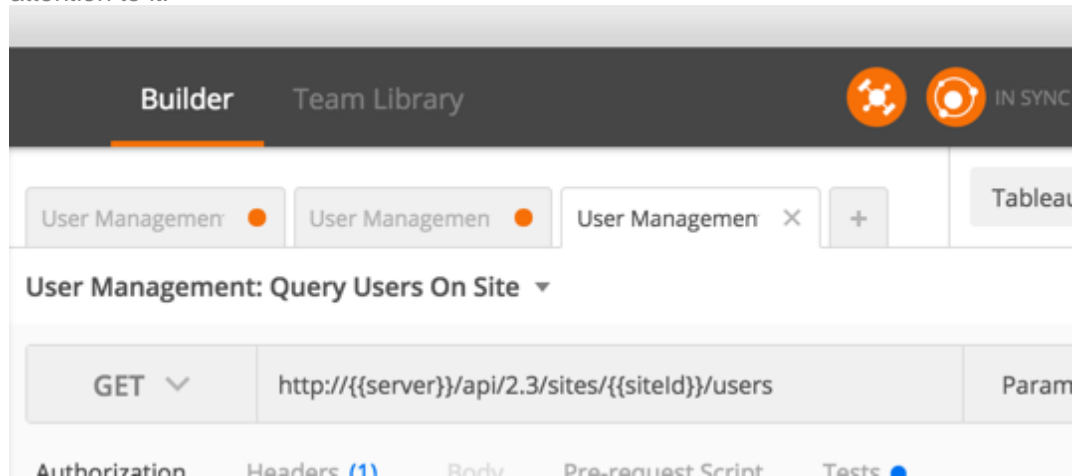
Copy to Clipboard



HTTP	client
C (LibCurl)	client.HTTPConnection("rgoldin-vm")
cURL	
C# (RestSharp)	u-auth': "RIIdrMjwQx2wizWuONSo8xiH1WRcwB7kF", ontrol': "no-cache", oken': "ab350850-c80b-c6fe-9cb1-651850bff0c3"
Go	
Java	▶ "GET", "/api/2.3/sites/12b22097-6bd7-41c4-a6e0-174e09705a6c/users",
JavaScript	▶ headers)
NodeJS	▶ response()
Objective-C (NSURL)	ad()
OCaml (Cohttp)	code("utf-8"))
PHP	▶
Python	▶
Ruby (NET::Http)	
Shell	▶
Swift (NSURL)	

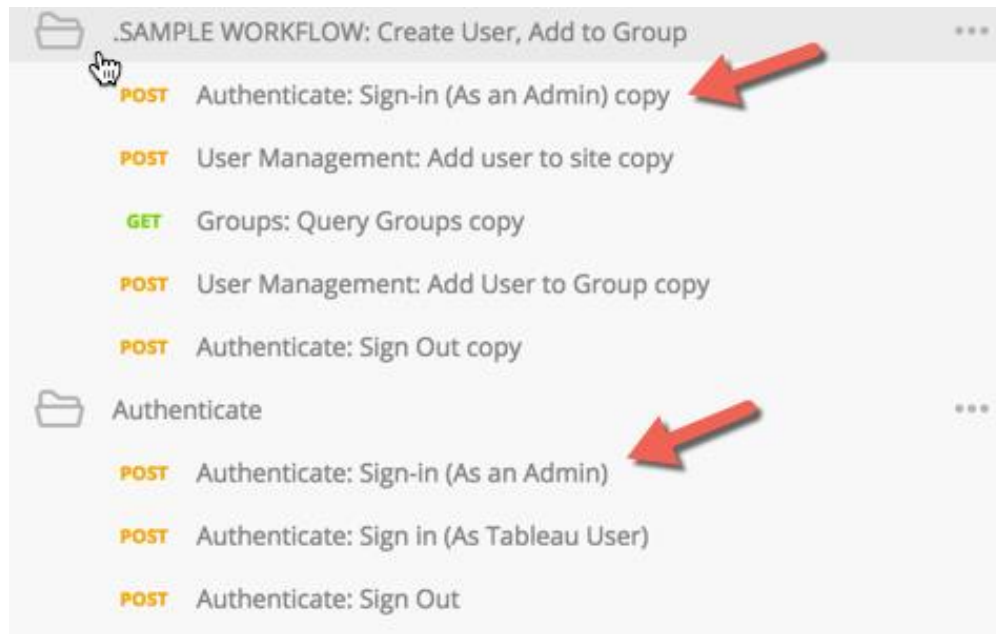
Limitations

1. It is possible to have the same API call open numerous times. If you make any changes to the saved copy, and then re-open the call, this will happen. The orange dot means that it has not been saved. This can easily become confusing if you aren't paying attention to it.



2. You cannot have nested folders.
3. Each API call can only exist in one place. If you want to create Sample Workflows (as provided) then you will need to make a copy ("duplicate") of the original and put it in the right place. Remember to make changes to all copies and not just one, if needed.





4. Content upload/download does not work.
 - * If you try to download content, Postman will show you the raw data. You will need another app (or production code) to use any of the download calls (workbook, datasource, etc). You can show that *something* is being downloaded, but it won't be useable
 - * Postman does not support multi-part forms, which are needed for any types of uploads/publishing.

