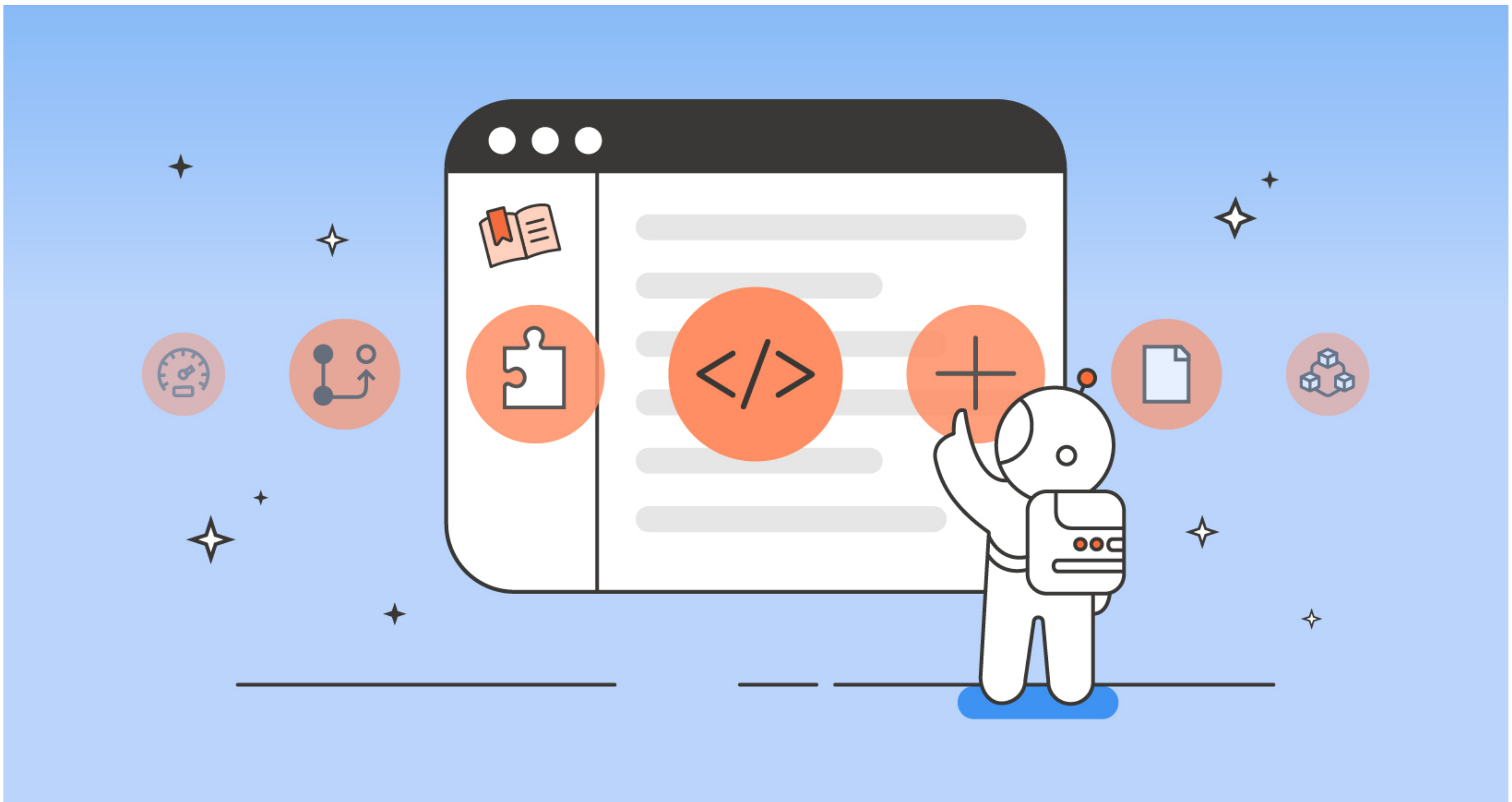


Adding External Libraries in Postman

Sean Keegan
April 27, 2022 · 5 mins

The Postman sandbox is an environment provided to execute Javascript code in the pre-request and tests scripts for a request. The sandbox has built-in JavaScript APIs via the `pm` and `postman` objects for testing your request and response data. While these objects help a ton with testing your API and workflow control, you may need additional functionality.

This is where external libraries come in. There are several helpful libraries built into the sandbox whose methods and functionality are available with very little extra work needed. However, you also have the ability to leverage additional external libraries. We'll first look at how to use the libraries that are available out of the box, then review some methods for adding your own desired libraries.

Get started right away

If you're a hands-on learner and want to see working examples, fork the Adding External Libraries collection to see the various methods of adding external libraries.

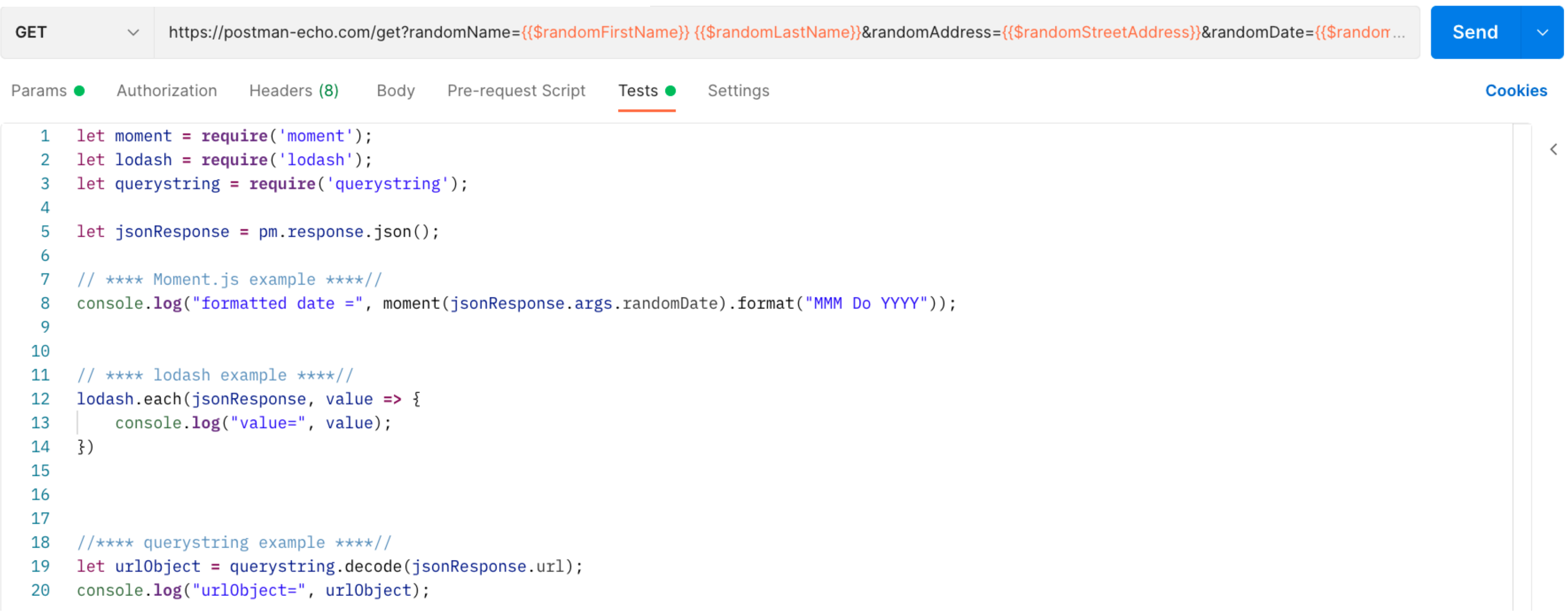
Using built-in library modules

Some popular JavaScript and NodeJS modules (such as moment, lodash, and querystring) are included in the Postman sandbox and can be imported by calling the `require` method. To do this, use the `require` method by passing the module name as a parameter and assigning the return object from the method to a variable.

Here's a basic example of using the `format` method from the moment library to format a date:

```
let moment = require('moment');
let jsonResponse = pm.response.json();
console.log("formatted date =", moment(jsonResponse.args.randomDate).format("MMM Do YYYY"));
```

To see the full list of available libraries with links to the documentation for each, check out the Postman Learning Center.



```
https://postman-echo.com/get?randomName={{$randomFirstName}} {{$randomLastName}}&randomAddress={{$randomStreetAddress}}&randomDate={{$randon...
```

```javascript
let moment = require('moment');
let lodash = require('lodash');
let querystring = require('querystring');

let jsonResponse = pm.response.json();

// **** Moment.js example ****//
console.log("formatted date =", moment(jsonResponse.args.randomDate).format("MMM Do YYYY"));


// **** lodash example ****//
lodash.each(jsonResponse, value => {
    console.log("value=", value);
})



//**** querystring example ****//
let urlObject = querystring.decode(jsonResponse.url);
console.log("urlObject=", urlObject);
```

Use built-in libraries, like lodash, by importing them with the `require` statement

Importing additional external libraries

For libraries that are not included in the Postman sandbox, you may still be able to make use of their functionality within Postman. If a CDN version of a library exists, or you have the pure JavaScript code for that library, we'll look at two methods of importing a library. For these examples, we'll import Day.js, a JavaScript library to help parse and format dates, and see how to use some of the Day.js methods.

Pull a library from a CDN

If a CDN version of a library exists, one option is to make a request to the CDN page hosting the library in the pre-request or tests tabs. Follow these steps:

1. Once you receive a response, convert the response to a string and save it as a collection variable.

2. With the code saved as a string, use the `eval` function to evaluate the JavaScript code.

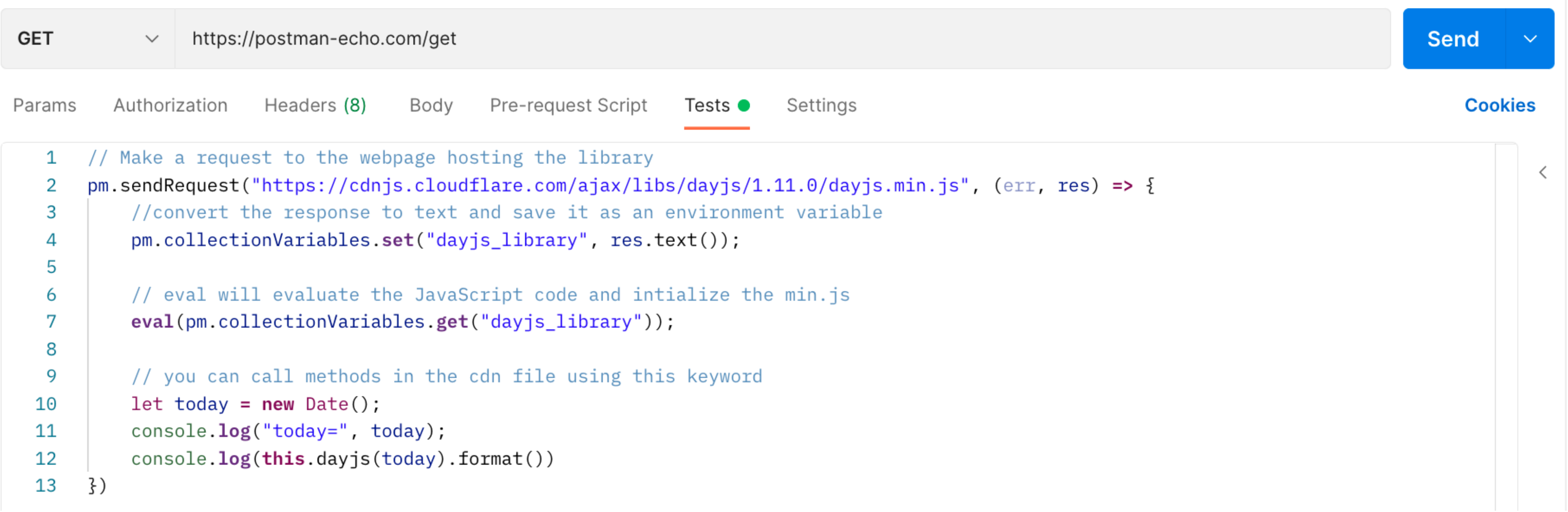
3. After running the `eval` function, the methods for the desired library are attached to the `this` keyword and can be used in the sandbox.

```
pm.sendRequest("https://cdnjs.cloudflare.com/ajax/libs/dayjs/1.11.0/dayjs.min.js", (err, res) => {
   //convert the response to text and save it as an environment variable
   pm.collectionVariables.set("dayjs_library", res.text());

   // eval will evaluate the JavaScript code and initialize the min.js
   eval(pm.collectionVariables.get("dayjs_library"));

   // you can call methods in the cdn file using this keyword
   let today = new Date();
   console.log("today=", today);
   console.log(this.dayjs(today).format())
})
```

Security note: Make sure you feel comfortable with the library you're importing this way. The `eval` function is a powerful and helpful tool, but since it just evaluates JavaScript code represented as a string, it can be a security risk to run random code with the `eval` function.



GET https://postman-echo.com/get

```
// Make a request to the webpage hosting the library
pm.sendRequest("https://cdnjs.cloudflare.com/ajax/libs/dayjs/1.11.0/dayjs.min.js", (err, res) => {
    //convert the response to text and save it as an environment variable
    pm.collectionVariables.set("dayjs_library", res.text());

    // eval will evaluate the JavaScript code and intialize the min.js
    eval(pm.collectionVariables.get("dayjs_library"));

    // you can call methods in the cdn file using this keyword
    let today = new Date();
    console.log("today=", today);
    console.log(this.dayjs(today).format())
})
```

Importing the Day.js library by sending a request to the CDN and saving it as a collection variable

Load a library from a variable

If you'd rather not make an additional API request every time you make your initial request, you can avoid it by loading in the library code ahead of time. Follow these steps:

1. Copy and paste the code of the library into a collection variable. Once saved, you can access the code in the Postman sandbox by creating and setting a variable equal to the collection variable.

2. Next, run an IIFE (Immediately Invoked Function Expression) which will give you access to the dayjs methods in the sandbox.

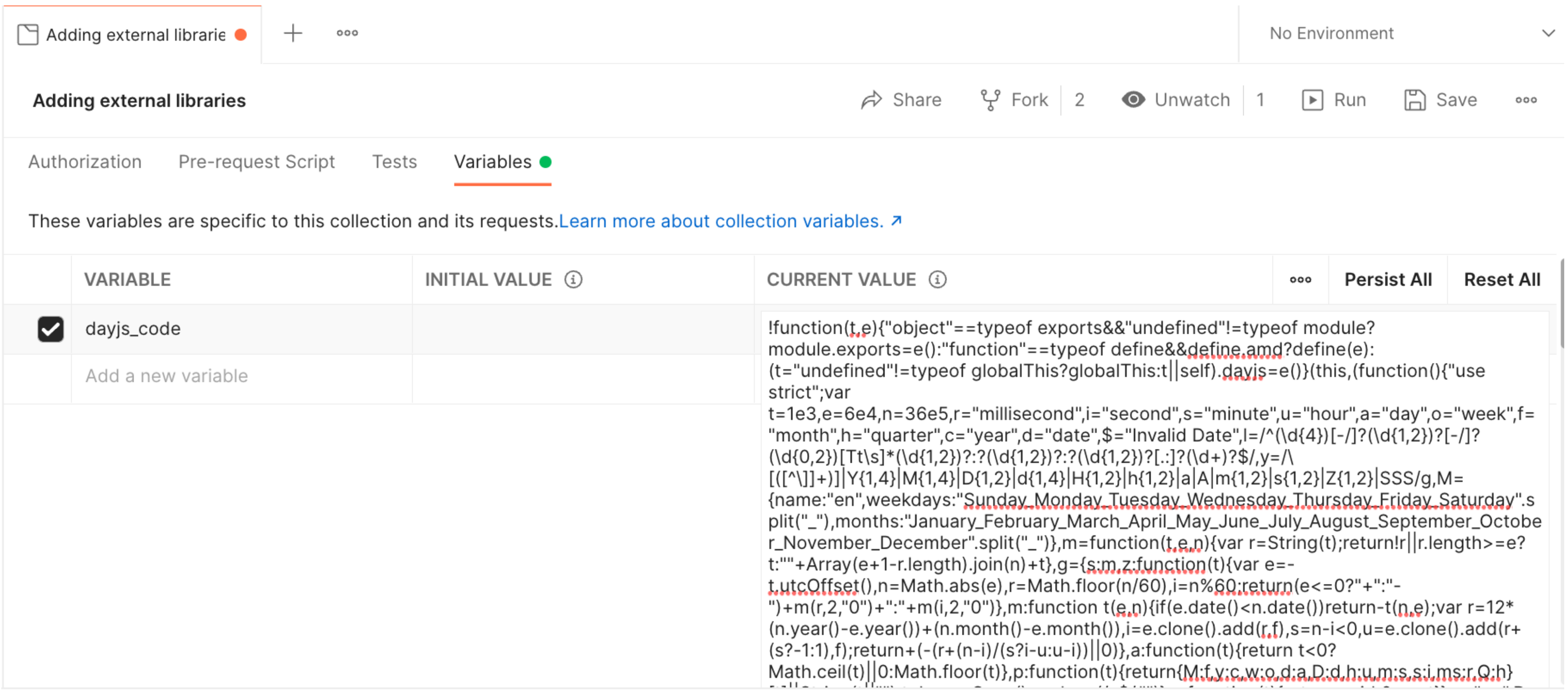
3. You'll now have access to the dayjs methods directly and can call them without using the `this` keyword.

```
const dayjs_code = pm.collectionVariables.get('dayjs_code');

// Invoke an anonymous function to get access to the dayjs library methods
(new Function(dayjs_code))();

let today = new Date();
console.log(dayjs(today).format())
```

Here is an example of pasting the minified code as a collection variable to access it in the **Tests** tab. It may look scary, but it's just condensed to limit the number of characters:



Adding external libraries

These variables are specific to this collection and its requests. Learn more about collection variables. ↗

	VARIABLE	INITIAL VALUE	CURRENT VALUE
☑	dayjs_code		
	Add a new variable		

```
!function(t,e){"object"==typeof exports&&"undefined"!=typeof module?
module.exports=e():"function"==typeof define&&define.amd?define(e):
(t="undefined"!=typeof globalThis?globalThis:t||self).dayjs=e()}(this,(function(){"use
strict";var
t=1e3,e=6e4,n=36e5,r="millisecond",i="second",s="minute",u="hour",a="day",o="week",f=
"month",h="quarter",c="year",d="date",$="Invalid Date",l=/^(\d{4})[-/]?(\d{1,2})?[-/]?
(\d{0,2})[Tt\s]*(\d{1,2})?:?(\d{1,2})?:?(\d{1,2})?[.:]?(\d+)?$/,y=/\
[([^\]]+)]|Y{1,4}|M{1,4}|D{1,2}|d{1,4}|H{1,2}|h{1,2}|a|A|m{1,2}|s{1,2}|Z{1,2}|SSS/g,M=
{name:"en",weekdays:"Sunday_Monday_Tuesday_Wednesday_Thursday_Friday_Saturday".s
plit("_"),months:"January_February_March_April_May_June_July_August_September_Octobe
r_November_December".split("_")},m=function(t,e,n){var r=String(t);return!r||r.length>=e?
t:""+Array(e+1-r.length).join(n)+t},g={s:m,z:function(t){var e=-
t.utcOffset(),n=Math.abs(e),r=Math.floor(n/60),i=n%60;return(e<=0?"+":"-
")+m(r,2,"0")+":"+m(i,2,"0")},m:function t(e,n){if(e.date()<n.date())return-t(n,e);var r=12*
(n.year()-e.year())+(n.month()-e.month()),i=e.clone().add(r,f),s=n-i<0,u=e.clone().add(r+
(s?-1:1),f);return+(-(r+(n-i)/(s?i-u:u-i))||0)},a:function(t){return t<0?
Math.ceil(t)||0:Math.floor(t)},p:function(t){return{M:f,y:c,w:o,d:a,D:d,h:u,m:s,s:i,ms:r,Q:h}
```

Try it out yourself

If you're a hands-on learner, don't forget to check out and fork the Adding External Libraries collection in our Postman Answer public workspace. You'll see code samples for all three methods of importing libraries discussed in this post and be able to play around with adding new libraries of your choosing. Happy testing!

Technical review by Arlemi Turpault

+20

Tags: Code Generator Scripts Tutorials

Sean Keegan

Sean Keegan is a developer advocate at Postman. View all posts by Sean Keegan.

What do you think about this topic? Tell us in a comment below.

Comment

Your name

Your email

Write a public comment

Post Comment

□ Receive replies to your comment via email.

4 thoughts on “Adding External Libraries in Postman”

Maximilian Liebsch
July 28, 2022

Dear Sean,
thank you for this article.
Since node-forge for example is using native browser capabilities and is somewhat in need of the window and document, it cannot be used. Do you have any suggestions?

‹ +2

Danys
June 14, 2023

Hello. I need a lib from the npmjs. Can i somehow import it into pre-request scripts? In particular https://www.npmjs.com/package/wsse

‹ 0

The Postman Team
June 14, 2023

Hi, Please contact our support team at https://www.postman.com/support and they’ll be glad to help you.

‹ 0

Rob
August 9, 2023

Does this approach still work in the new workspaces feature? I have it functioning in scratchpad to leverage jsrsasign but when i switch to workspace the exact same code does not work.

‹ +1

You might also like

Differences between Postman Collections and the collection format

Gbadebo Bello · 3 mins

The collection format specification defines the structure of Postman Collections that we know and love—providing all the information we need about a...

Read more →

The Postman CLI vs. Newman: choose the right tool for you

Ian Douglas · 9 mins

API testing is a big part of Postman’s history, and it continues to be a primary use of our platform. To extend...

Read more →

Create reusable tests and scripts with OpenAPI reusable components

Joyce · 6 mins

In programming, “hacking” has historically meant making something do what it wasn’t originally intended to do, like using a whistle from a...

Read more →

Postman named Best API Platform

Postman is the #1 place where developers come to work with APIs. See why we're top-ranked in G2's first-ever evaluation of API Platforms.

Read more →

Product

What is Postman?

API Repository

Tools

Governance

Workspaces

Integrations

Enterprise

Plans and pricing

Company

About

Careers and culture

Press and media

Contact us

Partner program

Legal and Security

Terms of Service

API Categories

App Security

Payments

Financial Services

DevOps

Developer Productivity

Data Analytics

Communication

Artifical Intelligence

Social

_Twitter

_LinkedIn

_GitHub

_YouTube

_Twitch