

CHEATSHEET

for getting started using Postman for ArchivesSpace.

Postman terminology

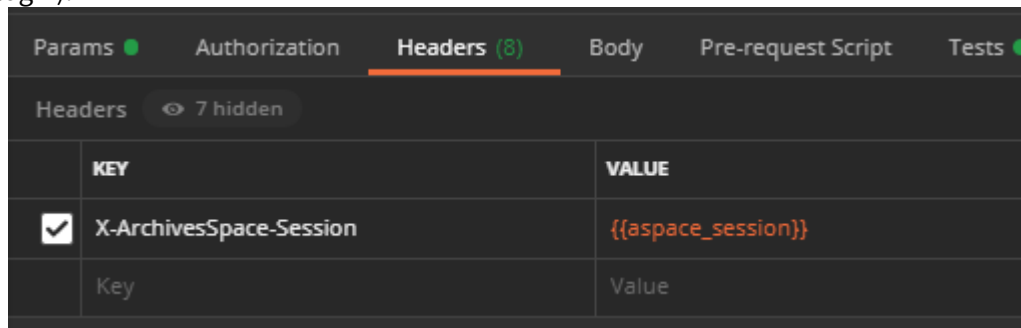
- **Pre-request script:** Something you want to have happen before the API call occurs.
- **Body:** Something you want to pass through the API in a POST request.
- **Tests:** Something you want to have happen after the API call occurs.
- **Collections:** A folder of API calls which can be run in sequence.
- **JSON:** A data structure that ArchivesSpace uses on the backend. Uses name/value pairs.

Variables to set before you begin

- Within an [empty environment](#), set the following values:
 - **aspace_base_url:** Your institution's API.
 - **aspace_user:** A username for ASpace. The ability to do things with the API will be based on the permissions assigned to the account.
 - **aspace_password:** The password for that account.
- Save your environment and this step can be skipped next time. Don't save credentials on a shared or public device.

How to start a session (authentication)

- Run the [ASpace Login collection](#) every time you start working with the ASpace API to set a session. Make sure you are passing the session variable in every ASpace API call (except the login):



Params ● Authorization Headers (8) Body Pre-request Script Tests ●		
Headers 7 hidden		
	KEY	VALUE
<input checked="" type="checkbox"/>	X-ArchivesSpace-Session	{{aspace_session}}
	Key	Value

Using the collection runner

- The Collection Runner allows you to run a collection multiple times with different data by passing a .csv with some variable data.
- The Collection Runner only cares about the column of data in the .csv that you told it to care about in the collection. There are two ways to get at the data in a .csv:

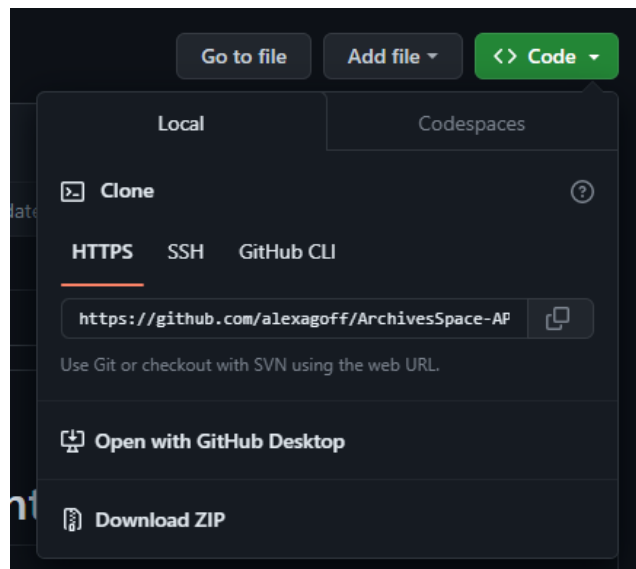
- In the parameters, put the column header inside double curly brackets
`{{column-header}}`
- In pre-request scripts and tests, call the data from the column with
`pm.iterationData.get("column-header")`
- Tips when using the Collection Runner:
 - Always preview your data before you begin running.
 - Don't forget to select an environment with the right variables set.

Using the console

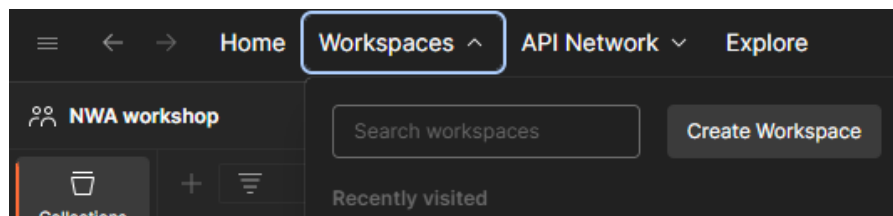
- The console records all your actions until you clear it. You can also filter it and copy and paste data out of it. This can be a useful way of generating data for a report.
- Use `console.log(something)` in a test or pre-request script to log something specific.

Download and import scripts into Postman

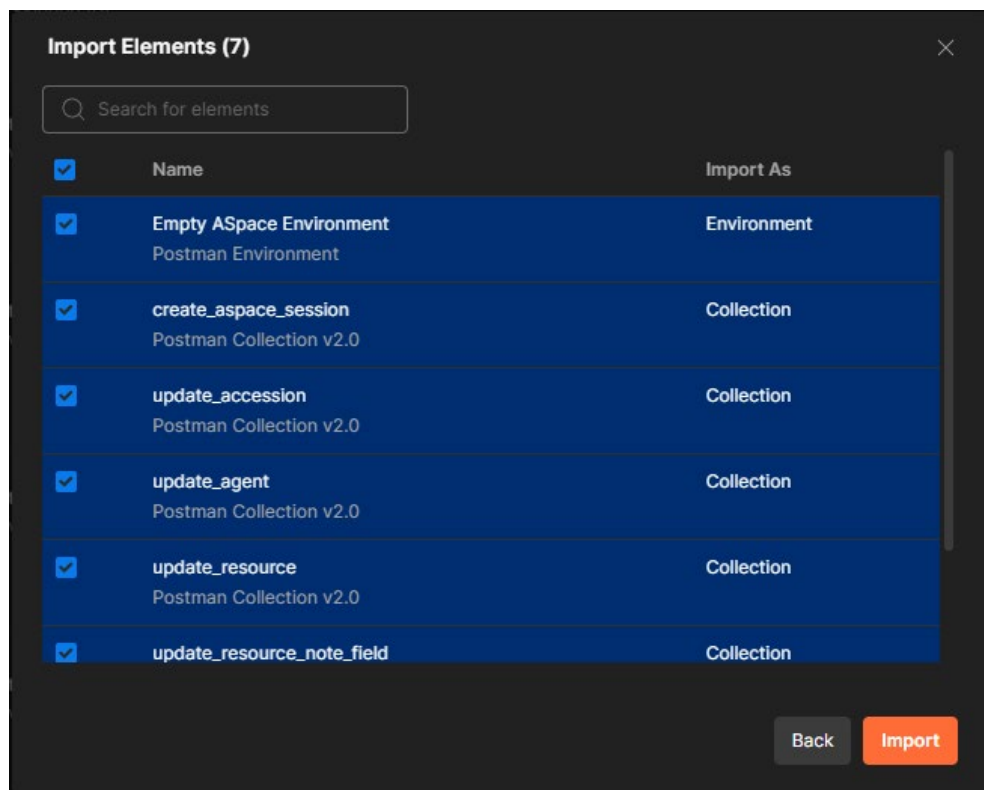
- Download or 'clone' GitHub repo to your computer. This is a folder containing the sample scripts we will work with during the workshop.
 - [Go to GitHub repository](#)
 - Click on green 'Code' button > 'Download ZIP'; then unzip the file



- Create a new workspace in Postman
 - Open Postman application
 - Click on 'Workspaces' > 'Create Workspace' - (



- Import folder into Postman
 - In your new workspace, click on 'Import' > 'Folders'
 - Navigate to the unzipped **folder** you downloaded from GitHub > 'Import'

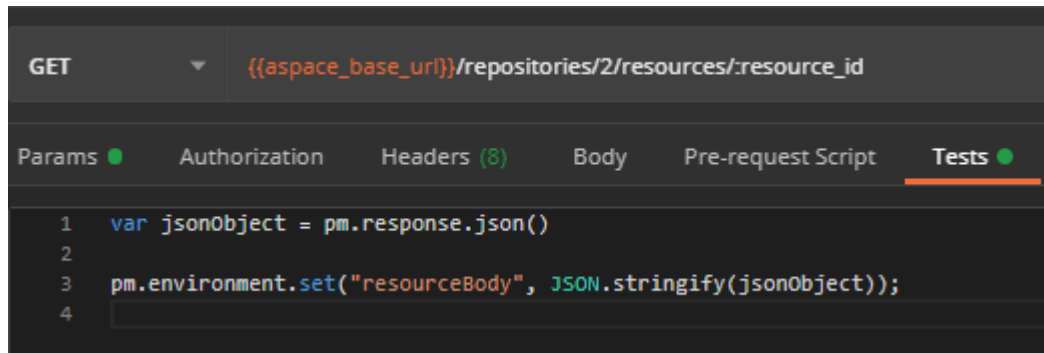


Some useful tricks

Changing something in a record

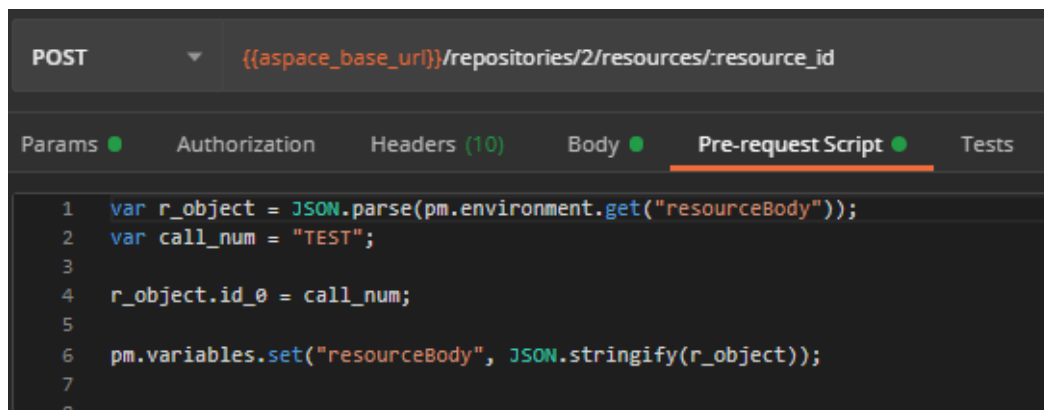
The basic approach to updating a record of any kind (resource, accession, top container, agent,...) is pretty much always the same. You GET the record you want to update by it's ID and store it to the Postman environment in the test. Then you POST the same record back into ASpace, changing or adding something in the pre-request script.

Example:

A screenshot of the Postman interface showing a GET request. The URL is {{aspace_base_url}}/repositories/2/resources/:resource_id. The 'Tests' tab is selected, showing a script that saves the response JSON to the environment.

```
1 var jsonObject = pm.response.json();
2
3 pm.environment.set("resourceBody", JSON.stringify(jsonObject));
4
```

- You have to turn the body into a string with JSON.stringify to save it as a variable

A screenshot of the Postman interface showing a POST request. The URL is {{aspace_base_url}}/repositories/2/resources/:resource_id. The 'Pre-request Script' tab is selected, showing a script that parses the saved JSON, updates the id_0 field to 'TEST', and saves it back to the environment.

```
1 var r_object = JSON.parse(pm.environment.get("resourceBody"));
2 var call_num = "TEST";
3
4 r_object.id_0 = call_num;
5
6 pm.variables.set("resourceBody", JSON.stringify(r_object));
7
8
```

- This pre-request script then takes that string, turns it back into JSON and changes the first field of the call number (id_0) to “Test”, which is also set as a variable.

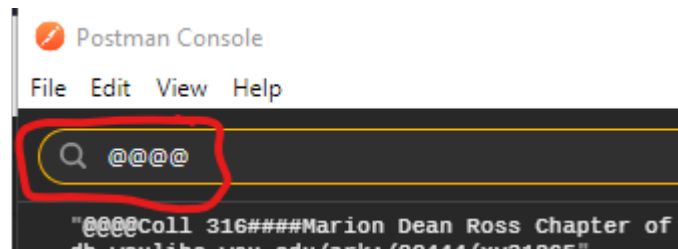
You can use the collection runner to do this in batch

Pulling data out of the console and turning it into a spreadsheet

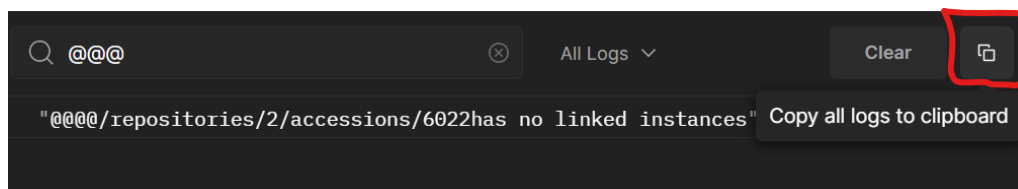
Postman isn't really set up for custom reporting (GETting data out and turning it into a spreadsheet, for example), but this kind of janky method will work in a pinch:

- Set up a console.log statement that begins with “@@@” and put “####” between any data you want in separate cells.
 - Example: console.log ("@@@" + jsonObject.id_0 + "####" + jsonObject.title + "####" + jsonObject.ead_location)

- Run a set of records through the collection runner.
- Filter the console by “@@@@”



- Copy the filtered contents of the console to the clipboard using the little Copy all logs button in the top right corner of the console.



- Run your API through the collection runner, then copy the filtered data into Notepad ++. You can use find and replace to get rid of the “@@@@” and turn the “####” into tabs. This is really slick if you create a macro.
- Select all the data from Notepad++ and paste it into a spreadsheet. The tabs should turn into new cells.

Working with resource notes and other arrays

Notes can be tricky to work with and change because they can repeat and they are nested in an array. A for loop combined with an if (see below) can help. In this case, the loop works through all the notes attached to a resource looking for one of the type “abstract”. If it finds it, it logs the contents of the note to the console.

```
for(var j = 0; j < notes.length; j++)
{
  if(notes[j].type == 'abstract')
  {
    console.log ("@@$ab$$ " + JSON.stringify(notes[j].content) + " %$");
  }
}
```