Store and reuse values using variables

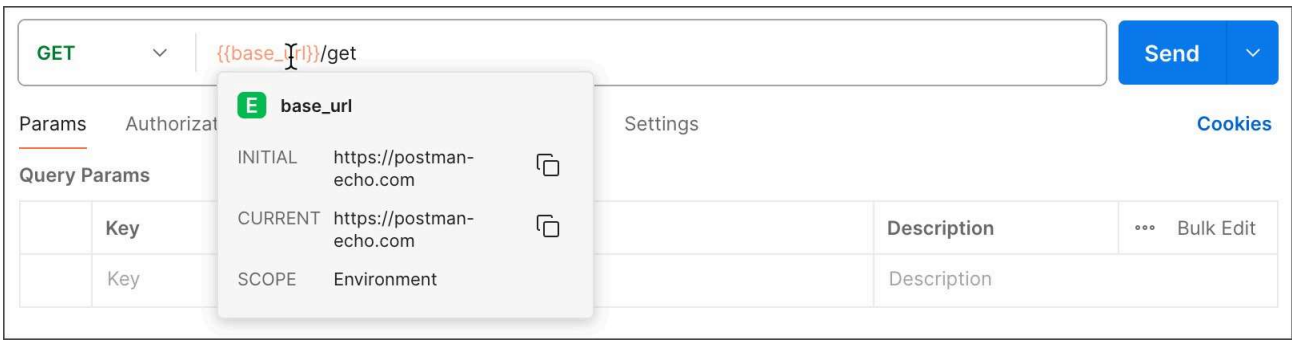
Variables enable you to store and reuse values in Postman. By storing a value as a variable, you can reference it throughout your collections, environments, requests, and scripts. Variables help you work efficiently, collaborate with teammates, and set up dynamic workflows.

Understanding variables

A variable is a symbolic representation of data that enables you to access a value without having to enter it manually wherever you need it. This can be useful if you are using the same values in multiple places. Variables make your requests more flexible and readable, by abstracting the detail away.

For example, if you have the same URL in more than one request, but the URL might change later, you can store the URL in a variable `base_url` and reference it in your requests using `{{base_url}}`. If the URL changes, you can change the variable value and it will be reflected throughout your collection, wherever you've used the variable name.

The same principle applies to any part of your request where data is repeated. Whatever value is stored in the variable will be included wherever you've referenced the variable when your requests run. If the base URL value is `https://postman-echo.com`, and is listed as part of the request URL using `{{base_url}}/get`, Postman will send the request to `https://postman-echo.com/get`.



Variables in Postman are key-value pairs. Each variable name represents its key, so referencing the variable name enables you to access its value.

You can use variables to pass data between requests and tests, for example if you are chaining requests <https://www.postman.com/postman/workspace/postman-team-collections/collection/1559645-81122f8b-5e07-4760-9504-f4387f45d2bc> in a collection.

Use your Postman Vault to store sensitive data as vault secrets, and reuse them in your local instance of Postman. Only you can access and use your vault secrets, and secrets aren't synced to the Postman cloud.

Use environments to group sets of variables together and share them with collaborators, for example if you use one set of config details for your production server and another for testing. See Group sets of variables in Postman using environments for more on how you can incorporate environments into your team workflows.

Variable scopes

Postman supports variables at different scopes, allowing you to tailor your processing to a variety of development, testing, and collaboration tasks. Scopes in Postman relate

to the different contexts that your requests run in, and different variable scopes are suited to different tasks.

In order from broadest to narrowest, these scopes are: *global*, *collection*, *environment*, *data*, and *local*.

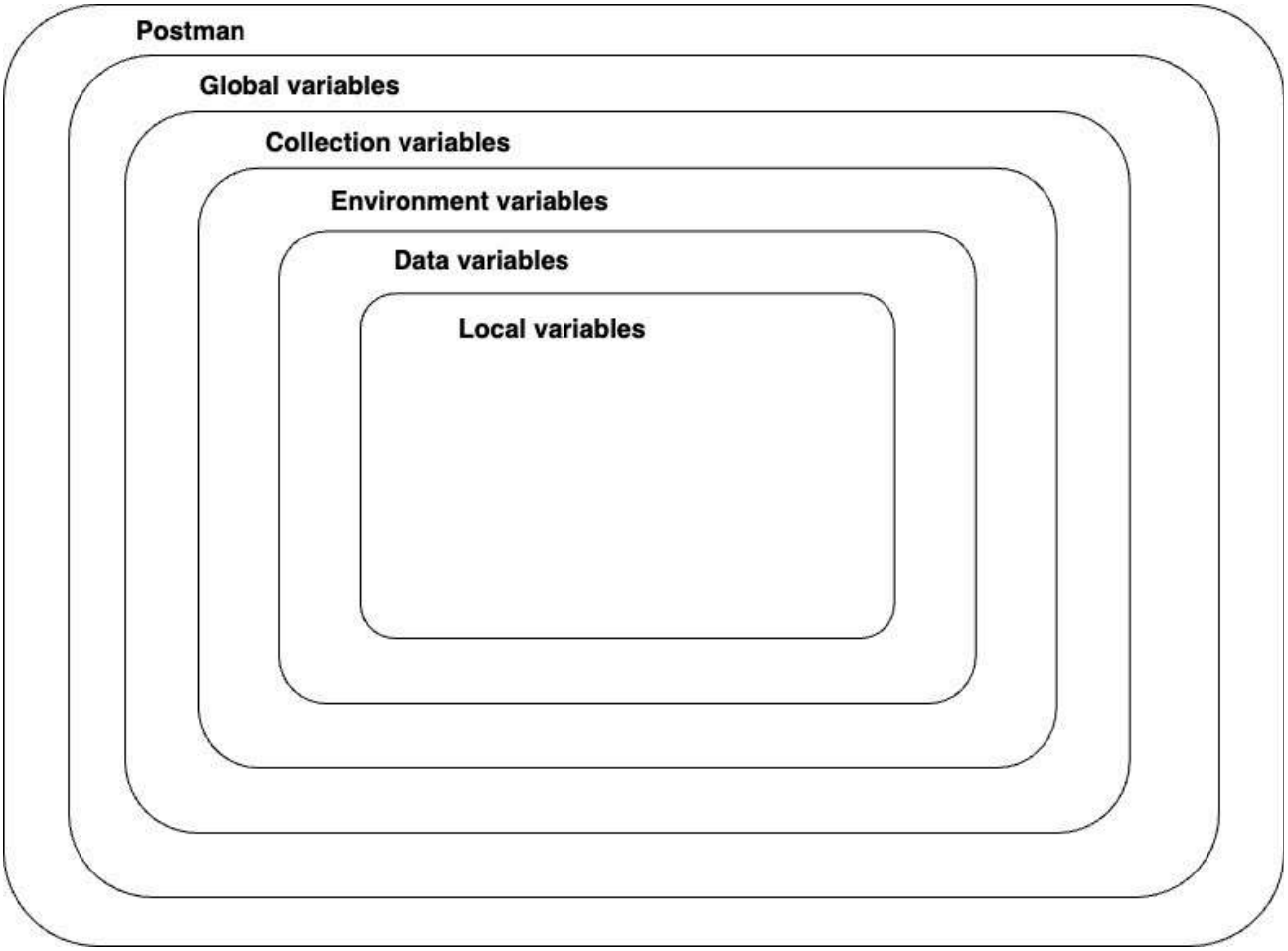
- **Global variables** enable you to access data between collections, requests, scripts, and environments. Global variables are available throughout a **workspace**. Since global variables have the broadest scope available in Postman, they're well-suited for testing and prototyping. In later development phases, use more specific scopes.

- **Collection variables** are available throughout the requests in a collection and are independent of environments. Collection variables don't change based on the selected environment. Collection variables are suitable if you're using a single environment, for example for auth or URL details.

- **Environment variables** enable you to scope your work to different environments, for example local development versus testing or production. One environment can be active at a time. If you have a single environment, using collection variables can be more efficient, but environments enable you to specify **role-based access levels**.

- **Data variables** come from external CSV and JSON files to define data sets you can use when running collections with **Newman** or the **Collection Runner**. Data variables have current values, which don't persist beyond request or collection runs.

- **Local variables** are temporary variables that are accessed in your request scripts. Local variable values are scoped to a single request or collection run, and are no longer available when the run is complete. Local variables are suitable if you need a value to override all other variable scopes but don't want the value to persist once execution has ended.



If a variable with the same name is declared in two different scopes, the value stored in the variable with narrowest scope will be used. For example, if there is a global variable named `username` and a local variable named `username`, the local value will be used when the request runs.

Postman stores variables as strings. If you store objects or arrays, remember to `JSON.stringify()` them before storing, and `JSON.parse()` them when you retrieve them.

Initial and current values

When you edit variables, each one has an *Initial value* and *Current value*:

- **Initial value** is a value that's set in the element (collection, environment, or globals) where the variable is defined. This value is synced to Postman's servers, and is shared with your team when you share that element. Setting

an initial value can be useful when sharing elements with teammates, but note that data in an initial value will also be shared with others, and potentially with the world.

If you need to store and reuse sensitive data, it's recommended to use Postman Vault to store it as vault secrets in your local instance of Postman. Only you can access and use your vault secrets, and vault secrets aren't synced to the Postman cloud. If you want to share sensitive data with collaborators or access it in scripts, you can store it in an environment as a secret type variable.

- **Current value** is used when sending a request. These are local values, and aren't synced to Postman's servers. If you change a current value, it won't be persisted in the original shared collection, environment, or globals.

You can persist or reset current values you have changed in variables. For more information, see sharing and persisting data.

> Learn more about the differences between vault secrets and variables.

Variable types

Beyond scope, global and environment variables can also be defined by type. The two variable types that you can configure for global and environment variables are:

- **Default type** is automatically assigned to variables. This type is shown as plain text and doesn't have extra properties.

- **Secret type** masks the initial and current values for all workspace members and can be used to prevent unintentional disclosure of sensitive data, including API secrets, passwords, tokens, and keys.

  It's recommended that you use your Postman Vault to store sensitive data, such as API keys, as vault secrets. Only you can access and use your vault secrets, and vault secrets aren't synced to the Postman cloud. If you want to share sensitive data with collaborators or access it in scripts, you can store it in an environment as a secret type variable.

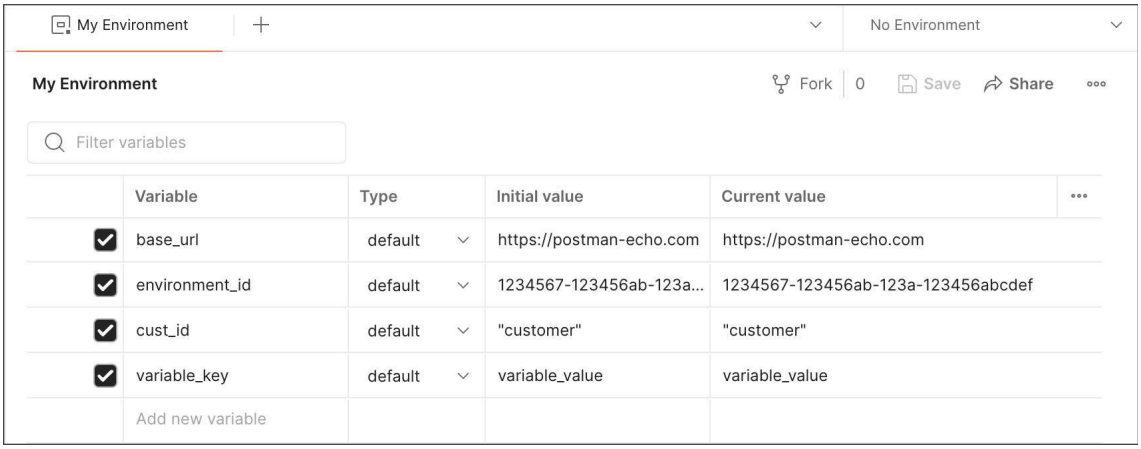
> Learn more about the differences between vault secrets and variables.

Users with Editor access on a workspace (for global variables) or environment (for environment variables) can opt to change these variables from default to secret type.

> Regardless of the type you configure for a variable, Postman stores variables as strings on its servers. To learn about how Postman keeps your data safe, see Security at Postman <https://www.postman.com/trust/security/> .

To set the variable type to secret, do the following:

- Select the environment quick look icon in the workbench.
- For environment or global variables, select **Edit** to open the editor.

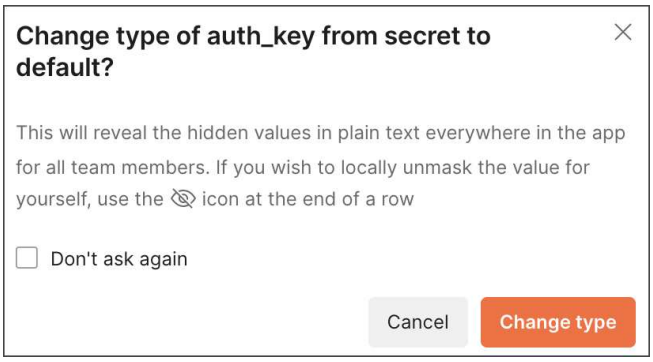


> You can also edit an environment by navigating to the workspace it resides in and selecting **Environments** from the sidebar, then selecting your environment.

- Select **default** next to the variable you want to change to open the dropdown, then select **secret** to update the variable type.
- Select **Save** to confirm your changes.

Changing from secret to default variable type

You must have **Editor** access on a workspace (for global variables) or environment (for environment variables) to control variable type. Editors can change the variable type from secret to default at any time, and vice versa. When you change a variable's type from secret back to default, you must confirm by selecting **Change type**.



Viewing and changing secret variable values

All workspace members can view a secret variable's initial and current values by selecting the eye icon next to the variable.

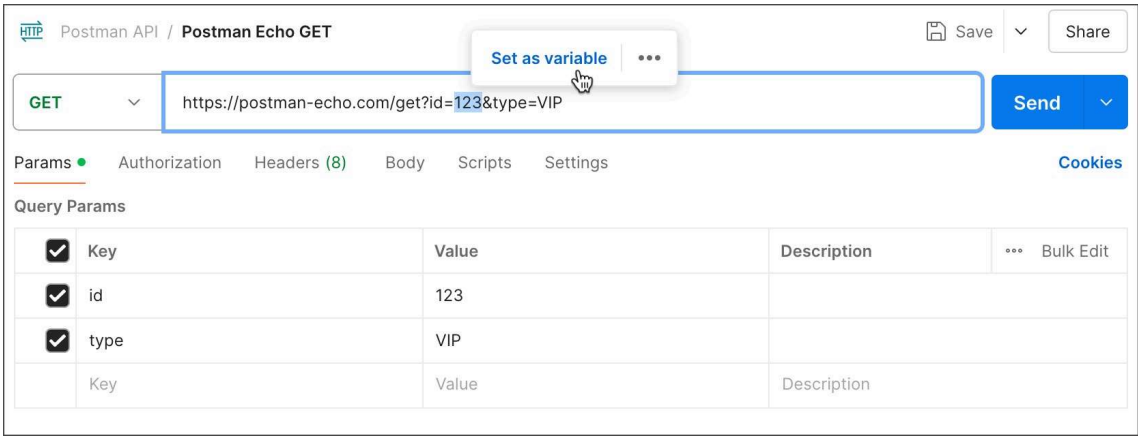
Editors can change a variable's initial values, which are shared with collaborators, by selecting the eye icon next to the variable, then selecting the initial value. All collaborators can change a variable's current values by selecting the eye icon, then selecting the current value.

Defining variables

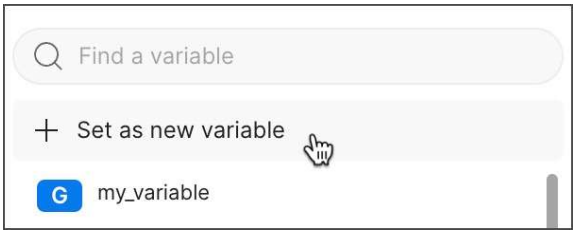
You can define variables in a variety of ways, depending on if you need **global**, **environment**, or **collection** scope.

To define variables at any scope in the request builder, do the following:

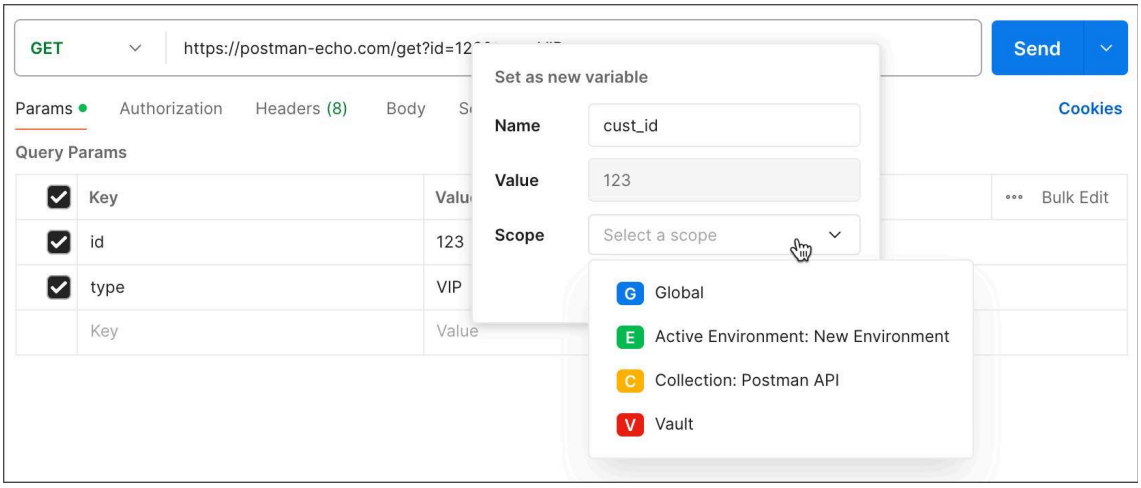
- Select the data you need, for example in the address, parameters, headers, or body. Select **Set as variable**.



- Select **Set as a new variable**.



- Enter a **Name**, confirm the **Value** is correct, and select a scope. Select **Set variable**.

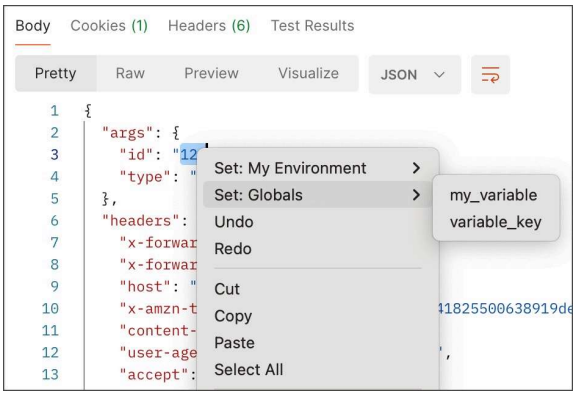


Remember to delete variables you are no longer using.

Setting response body values as variables

To set the values for existing variables to values from a request's response body, do the following:

- Select the text, then right-click or Control-click.
- Select the relevant scope (environment or global), then select the name of the variable.



Defining global variables

To view global variables, do the following:

- Select **Environments** in the sidebar.
- Select **Globals**.

> You can also view global variables by selecting the environment quick look icon in the workbench.
>
> The environment quick look shows the selected environment along with global variables in your workspace. You can edit the current value for an existing variable inline by selecting the value. To add a variable, select **Edit** next to the global section.

To add a new global variable, do the following:

- Select **Add a new variable**, and enter a name for the variable.

- Select a **Type** for the new variable.
- Add an **Initial Value**, and if you choose, a **Current Value**.
- Select **Save** to confirm your changes.

To edit an existing global variable, do the following:

- Change the desired variable value.
- Select **Save** to confirm your changes.

You can also define global variables in scripts.

Downloading global environments

To download global variables as JSON, do the following:

- Select **Environments** in the sidebar.
- Select **Globals**.
- Select **Export**.
- Choose where to save the file, then select **Save**.

Defining environment variables

To view environment variables, do the following:

- Select **Environments** in the sidebar.
- Select the environment you want to inspect variables for.

You can also inspect environment variables by selecting the environment quick look icon in the workbench.

The environment quick look shows the selected environment along with global variables in your workspace. You can edit the current value for an existing variable inline by selecting the value. To add a variable, select **Edit** next to the environment section.

To add a new environment variable, do the following:

- Select **Add a new variable**, and enter a name for the variable.
- Select a **Type** for the new variable.
- Add an **Initial Value**, and if you choose, a **Current Value**.
- Select **Save** to confirm your changes.

To edit an existing environment variable, do the following:

- Change the desired variable value.
- Select **Save** to confirm your changes.

- If you have Editor access to the environment, you can add and edit variables.
- If you have Viewer access to the environment, you are restricted to updating the *current value* of existing variables. Any variables you edit are accessible to you, but not to collaborators in your **workspace**.

See Managing environments for more on working with environments in your team.

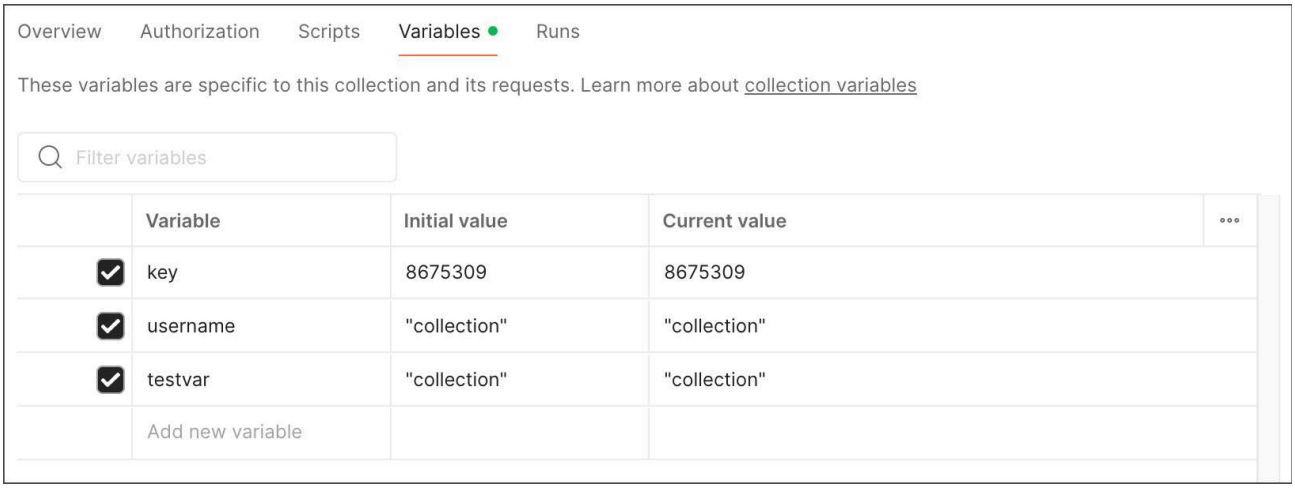
You can also define environment variables in scripts.

Defining collection variables

You can add collection variables when you create the collection or at any time after that.

To create or edit a variable for an existing collection, do the following:

- Select **Collections** in the sidebar.
- Select a collection, and then select the **Variables** tab.



> If you don't have Editor access to a collection, you can select **Request Access**. Without Editor access, you won't be able to add new collection variables, update initial values, or persist values. You can edit the current value for local use, override the collection variable by using an environment variable with the same name, or request Editor access to the collection.

You can also define collection variables in scripts.

Defining variables in scripts

You can set variables programmatically in your request scripts.

> You can't set or access vault secrets in scripts.

Method	Use-case	Example
`pm.globals`	Use to define a global variable.	`pm.globals.set("variable_key", "variable_value");`
`pm.collectionVariables`	Use to define a collection variable.	`pm.collectionVariables.set("variable_key", "variable_value");`
`pm.environment`	Use to define an environment variable in the currently selected environment.	`pm.environment.set("variable_key", "variable_value");`
`pm.variables`	Use to define a local variable.	`pm.variables.set("variable_key", "variable_value");`
`unset`	You can use `unset` to remove a variable.	`pm.environment.unset("variable_key");`

If you don't have Editor access to an environment, your script code will affect the current value but won't be synced or shared with your team.

For instructions on how to use variables in pre-request or post-response scripts, see Using variables in scripts.

Specifying variable detail

You can add and edit variables at any time. A variable requires a name. You can set an initial value immediately or later, including from scripts. Use the variable checkbox to turn it on or off.

Initial values are shared when you share a collection or environment. Current values are local and not synced or shared. See Initial and current values for more on local versus synced variables.

Using variables

You can use double curly braces to reference variables throughout Postman. For example, to reference a variable named `username` in your request authorization settings, you would use the following syntax with double curly braces around the name:

```
{{username}}
```
Copy

When you run a request, Postman will resolve the variable and replace it with its current value.

For example, you could have a request URL referencing a variable as follows:

```
https://postman-echo.com/get?customer_id={{cust_id}}
```
Copy

Postman will send whatever value you have stored for the `cust_id` variable when the request runs. If `cust_id` is `3`, the request will be sent to the following URL including the query parameter:

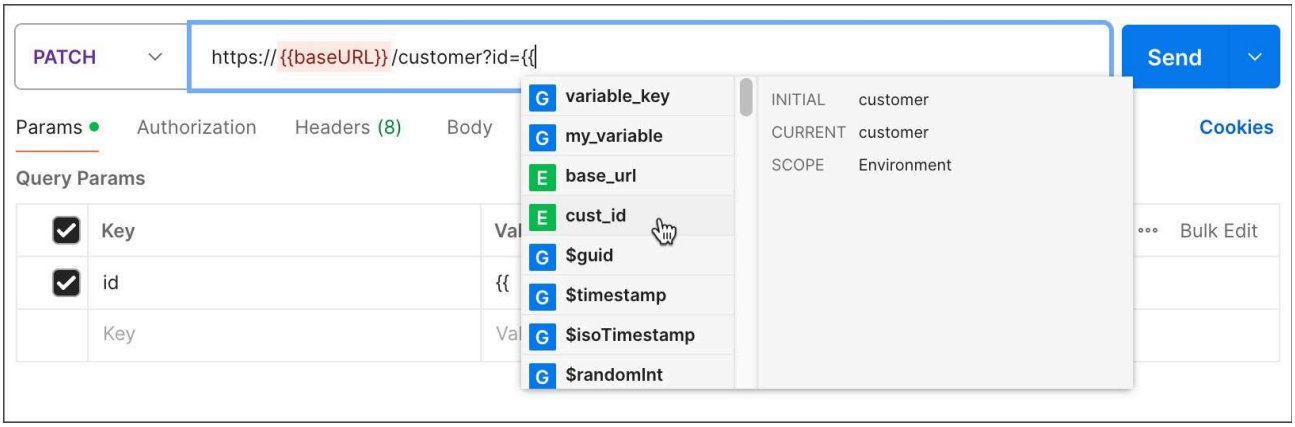
```
https://postman-echo.com/get?customer_id=3
```
Copy

If you are want to access a variable from within a request body, wrap its reference in double-quotes:

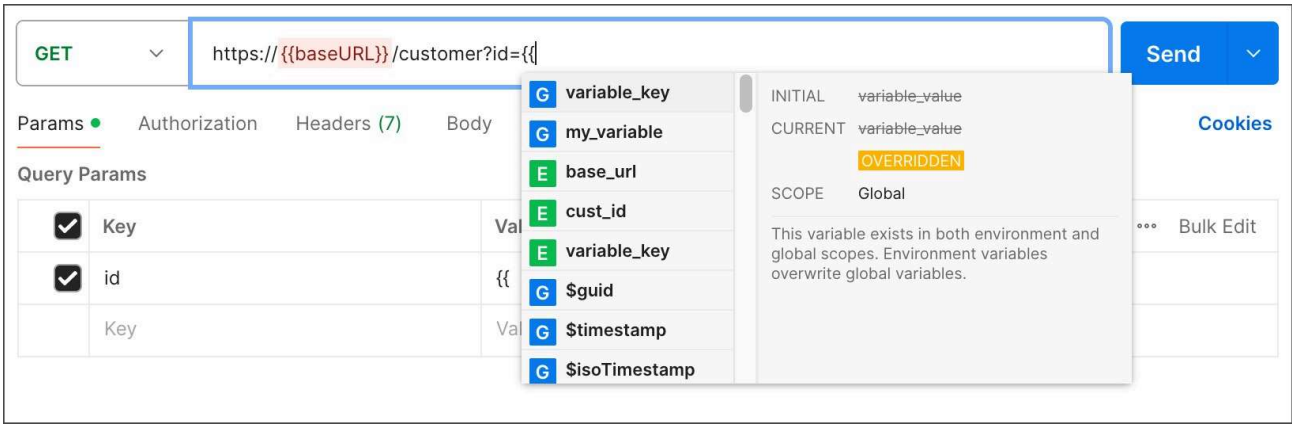
```
{ "customer_id" : "{{cust_id}}" }
```
Copy

You can use variables in request URLs, parameters, headers, authorization, body, and header presets.

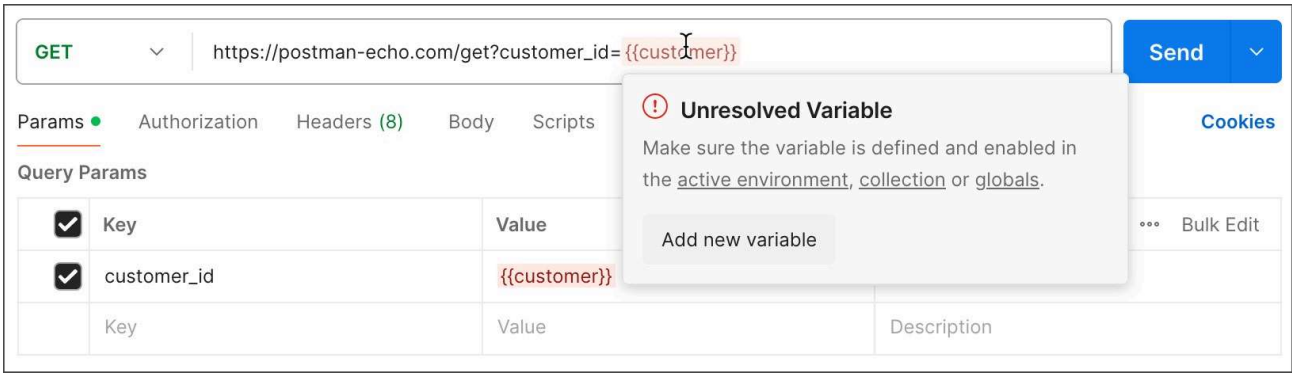
When you hover over a variable, Postman shows an overview of its current status. As you add variables to your requests, Postman prompts you with any already defined variables.



The prompt indicates the current value, scope (highlighted by color), and overridden status where relevant.



If a variable is unresolved, Postman highlights it in red. For information on how to fix an unresolved variable, see Fixing unresolved variables.



Using dynamic variables

Postman provides dynamic variables you can use in your requests.

Examples of dynamic variables include:

- `{{$guid}}` : A v4-style GUID
- `{{$timestamp}}`: The current Unix timestamp in seconds
- `{{$randomInt}}`: A random integer between 0 and 1000

See the Use dynamic variables to return randomly generated data section for a full list.

Using variables in scripts

You can retrieve the current value of a variable in your scripts using the object representing the scope level and the `.get` method:

```
//access a variable at any scope including local
pm.variables.get("variable_key");
//access a global variable
pm.globals.get("variable_key");
//access a collection variable
pm.collectionVariables.get("variable_key");
//access an environment variable
pm.environment.get("variable_key");
```

Copy

Using `pm.variables.get()` to access variables in your scripts gives you the option to change variable scope without affecting your script functionality. This method will return the variable that has the highest precedence (or narrowest scope).

You can't set or access vault secrets in scripts.

To use dynamic variables in pre-request or post-response scripts, use `pm.variables.replaceIn()`, for example `pm.variables.replaceIn('{{$randomFirstName}}')`.

See Process data and script workflows using Postman JavaScript objects for more details about scripting with variables.

Logging variables

You can log variable values to the Postman Console while your requests run.

Use the following syntax in your script to log the value of a variable:

```
console.log(pm.variables.get("variable_key"));
```

Copy

To view the results, select **Console** in the footer. You can also access the Console by selecting **View > Show Postman Console**.

Using data variables

The Collection Runner lets you import a CSV or a JSON file, and use the values from the data file inside requests and scripts. You can't set a data variable inside Postman because it's pulled from the data file, but you can access data variables inside scripts, for example using `pm.iterationData.get("variable_name")`.

For more details, see working with data files and the Postman JavaScript reference.

Sharing and persisting data

When you edit global, collection, and environment variables in Postman, there is a **Current value** that you can choose to **Persist** or **Reset** for individual variables. You can also select **Persist All** or **Reset All** to apply this setting to all variables. These enable you to control what happens within your local instance of Postman, independently of how the data is synced with anyone sharing your workspace, requests, collections, and environments.

When you create or edit a variable, you can enter both an initial and a current value. When you create a new variable in Postman, if you leave the current value empty, it will autofill with the initial value. If you specify a current value, it will be local to your instance. The **Persist** option pushes your current value to the shared data, updating the initial value to match the current value.

> If you don't have Editor access to an environment, you can't edit the initial value of an environment variable. You can edit the current value, and your edit won't be visible to anyone sharing your workspace.

Using **Persist** makes your current value sync with Postman's servers and be reflected for anyone sharing your collection or environment. To reset your current local values to reflect the initial shared values, use **Reset**.

To persist individual values, do the following:

- Hover over a variable's current value.
- Select the more actions icon ooo next to the value.
- Select **Persist**.

Your local session in Postman can use values that are transient and visible to you, but aren't synced or shared with your team. This lets you develop and test using private credentials or experimental values, without risk of exposing these details or affecting others on your team.

> For example, your team could have a shared API key and individual API keys. You could do experimental development work locally using your personal key, but use the shared key for team collaboration. Similarly, you could have a variable that represents exploratory work you're doing locally but aren't ready to share with the team. You can later choose to persist the local data so that others on your team can also access it.

You can edit a current value inline using the environment quick look icon  in the workbench.

For more information on working with variables as a team, see Work with environments as a team in Postman.

> Local and data variables have current values, which don't persist beyond request or collection runs.

Fixing unresolved variables

An *unresolved variable* is a variable that's not defined in an active scope (environment, collection, or globals) that's available for the request it's used in.

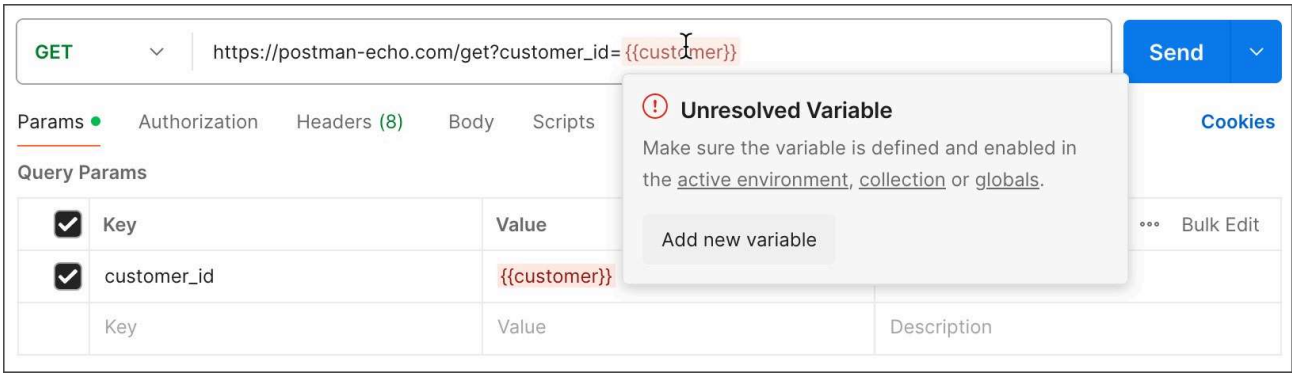
For example, for the request `https://postman-echo.com/get?customer_id={{cust_id}}`, Postman expects to be able to find a definition for `{{cust_id}}` in the environment the request uses, in the collection the request is saved in, or at the global level. If Postman doesn't find a definition for `{{cust_id}}` in one of those scopes, it flags the variable as unresolved. If you send a request that includes an unresolved variable, the request might fail.

A variable can be unresolved for the following reasons:

- The variable isn't present in an active scope for the request
- The variable was created but the changes weren't saved
- The environment in which the variable is present isn't active

- The variable is turned off in an active environment

When you are working on an API request, Postman highlights unresolved variables in the **URL builder**, the **Params** tab, the **Authorization** tab, and the **Headers** tab. Postman highlights unresolved variable text in red. For more details about the error and how to resolve it, hover over the unresolved variable.



To check if the variable is available and in scope for the request, do the following:

- Select one of the **collection** or **globals** links. To turn on an environment, use the **select an environment** link.
- Turn on or make the necessary changes to the value of the variable.
- Select **Save** to confirm your changes.

To set a variable that's unresolved because it doesn't exist, do the following:

- Select **Add new variable**.
- Enter a **Name**, set a **Value** for the variable, and select the appropriate **Scope** (global, collection, or environment) from the dropdown.
- Select **Set variable**.

Variables that are **defined programmatically in a script** are resolved differently depending on the variable scope. This means that unresolved variables will also be handled differently. Local variables that are set programmatically using `pm.variables.set` may appear to be unresolved since they're not stored and are only used at runtime, but if they're set and used correctly the request will still run successfully. Environment, global, and collection variables that are set programmatically are saved for later use, so they will resolve if they're set and used correctly. Depending on how an unresolved variable is used in a script, you may receive a `400 Bad Request` error response from the API, or Postman may be unable to send the request at all. Open the **Postman Console** to help identify unresolved variables in your scripts.

Last modified: 2023/02/14

← Overview **Create and use environments →**

Additional resources

Videos

How to Use Variables in Postman

Use Secret Variables | Postman Level Up

Intro to Postman | Chain Requests

Consuming REST APIs | Postman Enterprise

Blog posts

How to persist Postman variables

How to Securely Deploy Postman at Scale, Part 2: Information Management

Faster and easier variable management in Postman

Case Studies

Paylocity uses variables to automate workflows