



# Using HTTP Methods for RESTful Services

[Quick-Tips](#)[Resource Naming](#)

The HTTP verbs comprise a major portion of our “uniform interface” constraint and provide us the action counterpart to the noun-based resource. The primary or most-commonly-used HTTP verbs (or methods, as they are properly called) are POST, GET, PUT, PATCH, and DELETE. These correspond to create, read, update, and delete (or CRUD) operations, respectively. There are a number of other verbs, too, but are utilized less frequently. Of those less-frequent methods, OPTIONS and HEAD are used more often than others.

Below is a table summarizing recommended return values of the primary HTTP methods in combination with the resource URIs:

HTTP Verb	CRUD	Entire Collection (e.g. /customers)	Specific Item (e.g. /customers/{id})
POST	Create	201 (Created), 'Location' header with link to /customers/{id} containing new ID.	404 (Not Found), 409 (Conflict) if resource already exists..
GET	Read	200 (OK), list of customers. Use pagination, sorting and filtering to navigate big lists.	200 (OK), single customer. 404 (Not Found), if ID not found or invalid.
PUT	Update/Replace	405 (Method Not Allowed), unless you want to update/replace every resource in the entire collection.	200 (OK) or 204 (No Content). 404 (Not Found), if ID not found or invalid.
PATCH	Update/Modify	405 (Method Not Allowed), unless you want to modify the collection itself.	200 (OK) or 204 (No Content). 404 (Not Found), if ID not found or invalid.
DELETE	Delete	405 (Method Not Allowed), unless you want to delete the whole collection—not often desirable.	200 (OK). 404 (Not Found), if ID not found or invalid.

Below is a more-detailed discussion of the main HTTP methods. Click on a tab for more information about the desired HTTP method.

[POST](#)[GET](#)[PUT](#)[PATCH](#)[DELETE](#)

PATCH is used for **modify** capabilities. The PATCH request only needs to contain the changes to the resource, not the complete resource.

This resembles PUT, but the body contains a set of instructions describing how a resource currently residing on the server should be modified to produce a new version. This means that the PATCH body should not just be a modified part of the resource, but in some kind of patch language like JSON Patch or XML Patch.

PATCH is neither safe nor idempotent. However, a PATCH request can be issued in such a way as to be idempotent, which also helps prevent bad outcomes from collisions between two PATCH requests on the same resource in a similar time frame. Collisions from multiple PATCH requests may be more dangerous than PUT collisions because some patch formats need to operate from a known base-point or else they will corrupt the resource. Clients using this kind of patch application should use a conditional request such that the request will fail if the resource has been updated since the client last accessed the resource. For example, the client can use a strong ETag in an If-Match header on the PATCH request.

## Examples:

- *PATCH* <http://www.example.com/customers/12345>
  - *PATCH* <http://www.example.com/customers/12345/orders/98765>
  - *PATCH* [http://www.example.com/buckets/secret\\_stuff](http://www.example.com/buckets/secret_stuff)
- 



This work by [RestApiTutorial.com](http://RestApiTutorial.com) is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-sa/4.0/).

