HTTP Response Message

**SAFE METHOD**

* Client does not request and does not expect any state change or the origin server as a result of applying the method to the target resource. Implementors should be aware that the software represents the user in their interactions over the Internet, and should be careful to allow the user to be aware of any actions they might take which may have an unexpected significance to themselves or others.
* In particular, the convention has been established that the GET and HEAD methods SHOULD NOT have the significance of taking an action other than retrieval. These methods ought to be considered "safe". This allows user agents to represent other methods, such as POST, PUT and DELETE, in a special way, so that the user is made aware of the fact that a possibly unsafe action is being requested.
* GET - The GET method means retrieve whatever information (in the form of an entity) is identified by the Request-URI. If the Request-URI refers to a data-producing process, it is the produced data which shall be returned as the entity in the response and not the source text of the process, unless that text happens to be the output of the process.
* HEAD - The HEAD method is identical to GET except that the server MUST NOT return a message-body in the response. The metainformation contained in the HTTP headers in response to a HEAD request SHOULD be identical to the information sent in response to a GET request. This method can be used for obtaining metainformation about the entity implied by the request without transferring the entity-body itself. This method is often used for testing hypertext links for validity, accessibility, and recent modification.
* TRACE - The TRACE method is used to invoke a remote, application-layer loop- back of the request message. The final recipient of the request SHOULD reflect the message received back to the client as the entity-body of a 200 (OK) response.
* OPTIONS - The OPTIONS method represents a request for information about the communication options available on the request/response chain identified by the Request-URI. This method allows the client to determine the options and/or requirements associated with a resource, or the capabilities of a server, without implying a resource action or initiating a resource retrieval.
* POST - The POST method is used to request that the origin server accept the entity enclosed in the request as a new subordinate of the resource identified by the Request-URI in the Request-Line. POST is designed to allow a uniform method to cover the following functions:

- Annotation of existing resources;

- Posting a message to a bulletin board, newsgroup, mailing list,

or similar group of articles;

* DELETE - The DELETE method requests that the origin server delete the resource identified by the Request-URI. This method MAY be overridden by human intervention (or other means) on the origin server.
* CONNECT - This specification reserves the method name CONNECT for use with a proxy that can dynamically switch to being a tunnel (e.g. SSL tunneling [[44]](https://www.w3.org/Protocols/rfc2616/rfc2616-sec17.html#bib44)).
* PUT - The PUT method requests that the enclosed entity be stored under the supplied Request-URI. If the Request-URI refers to an already existing resource, the enclosed entity SHOULD be considered as a modified version of the one residing on the origin server.

**INDEMPONENT METHOD**

* Methods can also have the property of "idempotence" in that (aside from error or expiration issues) the side-effects of N > 0 identical requests is the same as for a single request. The methods GET, HEAD, PUT and DELETE share this property. Also, the methods OPTIONS and TRACE SHOULD NOT have side effects, and so are inherently idempotent.
* Intended eefect of the server of multiple identical request with the method is the same as the effectof single such as request.

**CACHEABLE METHOD**

* Indicates that the request to amethod is allowed to be stored for future reuse.

GET , HEAD , POST

HTTP Extensions

-HTTP can be extended by defining new request methods , message headers and status codes and then implementing servers of clients that make use of new functionality

Example:

* WebDAV ( web distributed Authoring and Versioning)
* Request Method:
* PROFIND , PROPATCH ,MKCOL , COPY , MOVE ,LOCK
* Message Headers:
* DAV , Depth , Destinations , Overwrite , TimeOut
* Status Code:
* (207)MultiStacks , (422)Unprocessable Entity
* (423)Locked , (424) Failed Dependency