Methods GET and POST in HTML forms - what's the difference?

In [HTML](http://www.cs.tut.fi/~jkorpela/html-primer.html), one can specify two different submission methods for a [form](http://www.cs.tut.fi/~jkorpela/forms/index.html). The method is specified inside a [FORM element](http://www.htmlhelp.com/reference/html40/forms/form.html), using the METHOD attribute. The difference betweenMETHOD="GET" (the default) and METHOD="POST" is primarily defined in terms of form data encoding. The official recommendations say that "GET" should be used if and only if the form processing is idempotent, which typically means a pure query form. Generally it is advisable to do so. There are, however, problems related to long URLs and non-ASCII character repertoires which can make it necessary to use "POST" even for idempotent processing.

## **The fundamental differences between "GET" and "POST"**

The HTML specifications technically define the difference between "GET" and "POST" so that former means that form data is to be encoded (by a browser) into a [URL](http://www.cs.tut.fi/~jkorpela/rfc/2396/toc.html) while the latter means that the form data is to appear within a message body. But the specifications also give the usage recommendation that the "GET" method should be used when the form processing is "idempotent", and in those cases only. As a simplification, we might say that **"GET" is basically for just getting (retrieving) data** whereas "POST" may involve anything, like storing or updating data, or ordering a product, or sending E-mail.

The [HTML 2.0 specification](http://www.w3.org/MarkUp/html-spec/html-spec_toc.html) says, in section [*Form Submission*](http://www.w3.org/MarkUp/html-spec/html-spec_8.html#SEC8.2) (and the [HTML 4.0 specification](http://www.w3.org/TR/REC-html40/)repeats this with minor stylistic changes):

If the processing of a form is idempotent (i.e. it has no lasting observable effect on the state of the world), then the form method should be GET. Many database searches have no visible side-effects and make ideal applications of query forms.

- -

If the service associated with the processing of a form has side effects (for example, modification of a database or subscription to a service), the method should be POST.

In the [HTTP specifications](http://www.w3.org/Protocols/) (specifically [RFC 2616](http://www.w3.org/Protocols/rfc2616/rfc2616.html)) the word **idempotent** is defined as follows:

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 word *idempotent*, as used in this context in the specifications, is (pseudo)mathematical jargon (see [definition of "idempotent" in FOLDOC](http://foldoc.doc.ic.ac.uk/foldoc/foldoc.cgi?query=idempotent&action=Search)) and should not be taken too seriously or literally here. The phrase "no lasting observable effect on the state of the world" isn't of course very exact either, and isn't really the same thing. Idempotent processing, as defined above, does not exclude fundamental changes, only that processing the same data twice has the same effect as processing it once. But here, in fact, idempotent processing means that a form submission causesno changes anywhere except on the user's screen (or, more generally speaking, in the user agent's state). Thus, it is basically for retrieving data. If such a form is resubmitted, it might get different data (if the data had been changed meanwhile), but the submission would not cause any update of data or other events. The concept of changes should not be taken too pedantically; for instance, it can hardly be regarded as a change that a form submission is logged into the server's log file. On the other hand, sending E-mail should normally be regarded as "an effect on the state of the world".

The HTTP specifications aren't crystal clear on this, and section [*Safe Methods*](http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html#sec9.1.1) in the HTTP/1.1 specification describes the principles in yet another way. It opens a different perspective by saying that users "cannot be held accountable" for side effects, which presumably means any effect than mere retrieval:

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.

Naturally, it is not possible to ensure that the server does not generate side-effects as a result of performing a GET request; in fact, some dynamic resources consider that a feature. The important distinction here is that the user did not request the side-effects, so therefore cannot be held accountable for them.

The concept and its background is explained in section [*Allowing input*](http://www.w3.org/Provider/Style/Input.html) in [Tim Berners-Lee](http://www.w3.org/People/Berners-Lee/)'s [*Style Guide for online hypertext*](http://www.w3.org/Provider/Style/). It refers, for more information, to [*User agent watch points*](http://www.w3.org/DesignIssues/UserAgent.html), which emphatically says that GET should be used if and only if there are no side effects. But this line of thought, however logical, is not always practical at present, as we shall see.

## **Why the distinction matters**

In the same context where the pragmatic difference is stated, the HTML 2.0 specification describes the corresponding submission methods technically. However, it does not explain why those methods are recommended for idempotent vs. non-idempotent submissions. Neither does it explain what practical difference there might be between the methods.

[Alan Flavell](http://www.alanflavell.org.uk/) has explained, in [an article](http://www.dejanews.com/getdoc.xp?AN=407045343) in a thread titled Limit on URL length in [the newsgroupcomp.infosystems.www.authoring.html](http://www.cs.tut.fi/~jkorpela/ciwah.html), that "the distinction [between GET and POST] is a real one, and has been observed by some browser makers, which can result in undesirable consequences if the inappropriate one is used". He gives the following reason for conforming to the advice:

When users revisit a page that resulted from a form submission, they might be presented with the page from their history stack (which they had probably intended), or they might be told that the page has now expired. Typical user response to the latter is to hit Reload.

This is harmless if the request is idempotent, which the form author signals to the browser by specifying the GET method.

Browsers typically will (indeed "should") caution their users if they are about to resubmit aPOST request, in the belief that this is going to cause a further "permanent change in the state of the universe", e.g. ordering another Mercedes-Benz against their credit card or whatever. If users get so accustomed to this happening when they try to reload a harmless idempotent request, then sooner or later it's going to bite them when they casually [OK] the request and do, indeed, order a second pizza, or invalidate their previous competition entry by apparently trying to enter twice, or whatever.

Thus, some browsers can act more cleverly if the author uses "GET" or "POST" consistently, i.e. using "GET" for pure queries and "POST" for other form submissions. It needs to be noted, though, that using "GET" gives no protection against causing changes. A script which processes a form submission sent with the "GET" could cause a pizza ordering. It's just that authors are expected to take care that such things don't happen.

Moreover, the use of "POST" cannot guarantee that the user does not inadvertantly submit the same form data twice; the browser might not give a warning, or the user might fail to understand the warning. Users are known to become impatient when it seems that "nothing happens" when they click on a button, so they might click on it again and again. Thus, robust processing of forms should take [precautions against unintended duplicate actions](http://www.webthing.com/tutorials/cgifaq.3.html#19). (As a simple example, a submission might be processed first by a script which sends back a page containing a confirmation request, echoing back the data submitted and asking the user to verify it and then submit the confirmation.)

A "GET" request is often cacheable, whereas a "POST" request can hardly be. For query systems this may have a considerable efficiency impact, especially if the query strings are simple, since caches might serve the most frequent queries.

## **Differences in form submission**

For both METHOD="GET" and METHOD="POST", the processing of a user's submit request (such as clicking on a submit button) in a browser begins with a construction of the [form data set](http://www.w3.org/TR/REC-html40/interact/forms.html#form-data-set), which is then encoded in a manner which depends on the [ENCTYPE attribute](http://www.w3.org/TR/REC-html40/interact/forms.html#adef-enctype). That attribute has two possible values mentioned in the specifications, but multipart/form-data is for "POST"submissions only, whereas application/x-www-form-urlencoded (the default) can be used both for "POST" and for "GET".

Then the form data set is transmitted as follows (quotation from the HTML 4.0 specification):

* If the method is "get" - -, the user agent takes the value of action, appends a ? to it, then appends the form data set, encoded using the application/x-www-form-urlencodedcontent type. The user agent then traverses the link to this URI. In this scenario, form data are restricted to ASCII codes.
* If the method is "post" --, the user agent conducts an HTTP post transaction using the value of the action attribute and a message created according to the content type specified by the enctype attribute.

Thus, for METHOD="GET" the form data is encoded into a URL (or, speaking more generally, into a [URI](http://www.cs.tut.fi/~jkorpela/rfc/2396/toc.html)). This means that an equivalent to a form submission can be achieved by following a normallink referring to a suitable URL; see the document [Choices in HTML forms](http://www.cs.tut.fi/~jkorpela/forms/choices.html) for details and examples. On a typical browser, the user sees the URL of a document somewhere (e.g. on Locationline), and if he is viewing the results of a query sent using METHOD="GET", he will see what the actual query was (i.e. the part of the URL that follows the ? sign). The user could then bookmark it or cut&paste it for later use (e.g. to be E-mailed or put into one's own HTML document after some editing).

Although the HTML specifications don't say it very explicitly, the fundamental difference between the methods is really that they correspond to **different HTTP requests**, as defined in the [HTTP specifications](http://www.w3.org/Protocols/). See especially [Method Definitions](http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html#sec9) in [RFC 2616](http://www.w3.org/Protocols/rfc2616/rfc2616.html). For form submission withMETHOD="GET", the browser constructs a URL as described above, then processes it as if following a link (or as if the user had typed the URL directly). The browser divides the URL into parts and recognizes a host, then sends to that host a [GET request](http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html#sec9.3) with the rest of the URL as argument. The server takes it from there. Submission of a form with METHOD="POST" causes a[POST request](http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html#sec9.5) to be sent.

## **Differences in server-side processing**

In principle, processing of a submitted form data depends on whether it is sent withMETHOD="GET" or METHOD="POST". Since the data is encoded in different ways, different decoding mechanisms are needed. Thus, generally speaking, changing the METHOD may necessitate a change in the script which processes the submission. For example, [when using the CGI interface](http://hoohoo.ncsa.uiuc.edu/cgi/forms.html), the script receives the data in an environment variable when METHOD="GET" is used but in the standard input stream (stdin) when METHOD="POST" is used.

It is, however, possible to construct libraries of subroutines (e.g. Perl functions) which allow one to write scripts in a manner which works both for METHOD="GET" and METHOD="POST". This would be based on distinguishing between the cases within the subroutine code and returning the data to the caller in a uniform manner.

HTML forms (up to HTML version 4 and XHTML 1) only support **GET** and **POST** as HTTP request methods. A workaround for this is to tunnel other methods through POST by using a hidden form field which is read by the server and the request dispatched accordingly.

However, **GET**, **POST**, **PUT** and **DELETE** *are* supported by the implementations of XMLHttpRequest (i.e. AJAX calls) in all the major web browsers (IE, Firefox, Safari, Chrome, Opera).

<http://stackoverflow.com/a/166501/3317808>