URIs, Addressability, and the use of HTTP GET and POST

TAG Finding 21 March 2004

This version:
http://www.w3.org/2001/tag/doc/whenToUseGet-20040321
Latest version:
http://www.w3.org/2001/tag/doc/whenToUseGet (XML)
Previous versions:
22 Sep 2003 Draft, 19 Sep 2003 Draft
Editor:
Ian Jacobs, W3C

Abstract

An important principle of Web architecture is that all important resources be identifiable by URI. The finding discusses the relationship between the URI addressability of a resource and the choice between HTTP GET and POST methods with HTTP URIs. HTTP GET promotes URI addressability so, designers should adopt it for safe operations such as simple queries. POST is appropriate for other types of applications where a user request has the potential to change the state of the resource (or of related resources). The finding explains how to choose between HTTP GET and POST for an application taking into account architectural, security, and practical considerations.

This finding does not discuss URI schemes other than "http" or protocols other than HTTP/1.1 [RFC2616].

Status of this Document

This document has been produced by the W3C Technical Architecture Group (TAG). This finding addresses TAG issue whenToUseGet-7.

This is the 21 March 2004 revision of this finding. It has been approved by the TAG. At their 2 March 2004 face-to-face meeting, the TAG resolved to approve this version of the finding with one change from the previous version, the addition of a link to ongoing work in the W3C Web Services Description Working Group, in section 6. The 21 March 2004 revision supersedes the 22 Sep 2003 version of this finding, which the TAG had previously approved.

Additional TAG findings, both approved and in draft state, may also be available. The TAG expects to incorporate this and other findings into a Web Architecture Document that will be published according to the process of the W3C Recommendation Track.

The terms MUST, SHOULD, and SHOULD NOT are used in this document in accordance with [RFC2119].

Please send comments on this finding to the publicly archived TAG mailing list www-tag@w3.org (archive).

Table of Contents

---

1 Introduction

In this finding, we discuss when it is appropriate to use HTTP GET and when it is appropriate to use HTTP POST, and the architectural principles behind the separation of these two methods.

In the introduction, we present scenarios that illustrate some of the topics covered, summarize the relevant principles of Web Architecture, and provide a quick checklist for choosing GET or POST. Following that, we discuss the benefits of URI addressability, which is the primary reason for distinguishing GET from POST. We then characterize safe interactions, when the use of GET is encouraged. However, when considering GET or POST, designers should also remain aware of considerations for sensitive data such as passwords or credit card information, and other practical considerations.

1.1 Scenarios

Scenario 1: Dan purchases a new wireless telephone and wishes to share information about it with his colleagues. Following the clue "CAT. NO. 43-893" on the bottom of the phone, Dan visits the Web site of the phone manufacturer. Typing the catalog number in the site's search box, Dan pulls up the relevant information. However, since the site designers created a POST Web form instead of a GET Web form for this read-only operation, Dan cannot bookmark the query, or send the query to his colleagues (who are potential customers) as a URI, or gain any of the other benefits made possible by the proper use of HTTP GET. The Web site designer should have provided GET access to the catalog information; see the benefits of URI addressability.

Scenario 2: My bank allows me to read current information about my checking account over the Web. The URI they provide me for this access includes my account number. This is sensitive information I do not want to appear in the URI since, for example, when I follow a link from the bank Web site to the site of their advertiser, my account information may appear in the advertiser's server logs. Rather than moving my account number from the URI into the body of a POST request, my bank allows me to establish a secure HTTP connection. Not only do I feel more confident retrieving information over a secure connection, my browser knows not to expose sensitive information to the next site I visit after the secure connection. As a result, I can still bookmark my account page without revealing sensitive information. For more information, see the considerations for sensitive data.

1.2 Relevant Principles of Web Architecture

1. All important resources SHOULD be identifiable by URI.
2. HTTP GET is safe; i.e., agents do not incur obligations by following links.

1.3 Quick Checklist for Choosing HTTP GET or POST

- Use GET if:
  - The interaction is more like a question (i.e., it is a safe operation such as a query, read operation, or lookup).
- Use POST if:
  - The interaction is more like an order, or
  - The interaction changes the state of the resource in a way that the user would perceive (e.g., a subscription to a service), or
  - The user be held accountable for the results of the interaction.

However, before the final decision to use HTTP GET or POST, please also consider considerations for sensitive data and practical considerations.

2 The Benefits of URI Addressability

Web architecture starts with Uniform Resource Identifiers (URI), whose generic syntax is defined by [RFC2396]. The Web relies on global agreement to follow the rules of URIs so that we can refer to things on the Web, access them, describe them, and share them. Providing a URI for a resource affords many advantages, including:

- linking
- bookmarking
- caching

As described in [TBL50K], "Great multiplicative power of reuse derives from the fact that all languages use URIs as identifiers: This allows things written in one language to refer to things defined in another language. The use of URIs allows a language to leverage the many forms of persistence, identity, and various forms of equivalence."

In this finding, the term "URI addressability" means that a URI alone is sufficient for an agent to carry out a particular type of interaction.

In some cases, it may be desirable to interact in different ways with the resource identified by a URI. For example, the W3C link checking service illustrates the usefulness of separating URIs from methods. The link checker works as follows (ignoring recursive link checking for this discussion): for each HTTP URI that is part of a link in an HTML or XHTML document, the link checker uses the HEAD method to gather information about the resource identified by the URI. As the HTTP/1.1 specification states, "The HEAD method is identical to GET except that the server MUST NOT return a message-body in the response." By using HEAD instead of GET, the link checker does not ask for more bits than it requires to do its job.

The Web architecture thus allows for the separation of URIs and methods. A URI's scheme determines a set of interaction methods. For instance, the HTTP URI scheme is defined in HTTP/1.1 [RFC2616], which allows "an open-ended set of methods and headers that indicate the purpose of a request. It builds on the discipline of reference provided by the Uniform Resource Identifier (URI) ... for indicating the resource to which a method is to be applied."; refer to section 1.1.

There are costs to separating URIs and methods, however. The separation means that some metadata required for the interaction may not be part of the URI, and as a result, the URI alone is no longer sufficient for an agent to carry out the interaction. URI addressability is lost.

In the following section we look more closely at when to use HTTP GET (since it promotes URI accessibility) and when to use HTTP POST.

3 Safe Interactions

Section 9.1 of HTTP/1.1 discusses *safe interactions*. A safe interaction is one where the user is not to be held accountable for the result of the interaction. Safe interactions are important because these are interactions where users can browse with confidence and where software programs (e.g., search engines and browsers that pre-cache data for the user) can follow links safely. Users (or software agents acting on their behalf) do not commit themselves to anything by querying a resource or following a link. For example, Web sites that say "by following the link to ABC, you agree to the following terms and conditions" do not account for the fact that anyone (in particular, a search service) can make another link to ABC, and anyone who follows this other link to ABC may never have seen the terms and conditions. See the related TAG finding "Deep Linking in the World Wide Web" [DEEPLINK].

By distinguishing safe from "unsafe" interactions (i.e., those where the user should be accountable for a particular interaction) at the protocol level:

- Clients can inform users when a particular action will have consequences (e.g., subscription to a list, payment of a fee, cancellation of a contract, etc.) In section 9.1.1, the HTTP/1.1 specification states, "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.
- Intermediaries (e.g., caches) can safely process on behalf of the user.

In the same section, HTTP/1.1 states, the convention is that GET is used for safe interactions and SHOULD NOT have the significance of taking an action other than retrieval. Indeed, if you use GET for interactions with side-effects, your make your system insecure. For example, a malicious Web page publisher outside a firewall might put a URI in an HTML page so that, when someone inside the firewall unwittingly follows the link, that person activates a function on another system within the firewall.

Users accept obligations through other mechanisms than requests to follow a link. Per the HTTP/1.1 specification, designers should use HTTP POST for those interactions.

HTTP GET is designed so that all information necessary for the interaction is part of the URI, thus promoting URI addressability. With HTTP POST, some information intended to affect change to the resource state may be part of the protocol headers, not in the URI. With this approach, the resulting URI for identifying the resource may be shorter, but the advantages of URI addressability are lost.

Note that it is possible to use POST even without supplying data in an HTTP message body. In this case, the resource is URI addressable, but the POST method indicates to clients that the interaction is unsafe or may have side-effects.

By convention, when GET method is used, all information required to identify the resource is encoded in the URI. There is no convention in HTTP/1.1 for a safe interaction (e.g., retrieval) where the client supplies data to the server in an HTTP entity body rather than in the query part of a URI. This means that for safe operations, URIs may be long. The case of large parameters to a safe operation is not directly addressed by HTTP as it is presently deployed. A QUERY or "safe POST" or "GET with BODY" method has been discussed (e.g., at the December 1996 IETF meeting) but no consensus has emerged.

WebDAV [RFC2518] uses a different HTTP method, PROPFIND (section 8.1 PROPFIND), for querying properties of resources; unfortunately, this provides no URI for the results of these queries.

Other work on identification of safe interactions includes the experimental RFC "The Safe Response Header Field" [RFC2310].

3.1 Examples of using GET and POST

In section 9.5, the HTTP/1.1 specification lists some example applications where POST should be used due to side-effects, including annotation of existing resources; posting a message to a bulletin board, newsgroup, mailing list, or similar group of articles; and extending a database through an append operation.

Whether and how one chooses between GET and POST depends on the format specification and the application context. When HTTP URIS are used for hyperlinks in HTML, SMIL, and SVG, for example, the application determines which method will be used (generally GET). However, for both HTML forms and XForms, the author can choose between GET and POST. One implication is that HTML application designers should implement "unsafe" operations (or others requiring POST) with a form, even if the application does not call for user input.

3.1.1 Mailing List Subscription

Consider the following two designs for mailing list subscription confirmation.

Design 1:

1. The user sends a subscribe message to an administrative mailbox (mylist-request@example.org).
2. The list processing software sends an email response to the user, requesting that the user confirm the subscription request, and including a link to a confirmation page.
3. The user follows this link to the confirmation page, and finds a form with a button to "[Confirm] your subscription". The form is submitted with method="POST".
4. The user activates the [Confirm] form control.
5. The list processing software confirms the subscription.

Design 2 (incorrect):

1. same as 1 above
2. same as 2 above
3. The user follows the link to the confirmation page and and is informed "your subscription is confirmed".

The latter design performed an unsafe operation (list subscription) in response to a request with a safe method (following the link from the mail message with GET). If the user's mail agent pre-fetched pages to speed up browsing, the subscription would be confirmed without the knowledge and consent of the user; the HTTP/1.1

specification makes it clear that the fault is with the server in this case; the user's mail agent is free to follow links without incurring obligations.

3.1.2 Steps for Establishing an Obligation

To establish obligations of confidentiality (or payment, or licensing terms, etc.), designers should follow this protocol:

1. The client requests access to materials that can only be viewed by people who agree to certain confidentiality terms.
2. The server declines, with an "authorization required" notice and a link to an account application form.
3. The client follows the link to the form, and applies for an account, agreeing to the terms and conditions in a POST request (or by fax or postal mail, for that matter).
4. The server provides credentials in response.
5. The client re-requests the materials, providing credentials.

Similarly, an individual might agree to terms and conditions by signing a contract with an employer or other organization, after which the person might legitimately incur an obligation through some interaction (e.g., following a link) covered by the terms of the agreement.

3.1.3 Side-effects do not Imply Unsafe Interactions

Some user interactions cause side effects (i.e., they change the state of the server) but are safe interactions. When a server is configured to count and display the number of visitors to a site, each user interaction increments a counter. Users do not commit themselves to anything through this sort of interaction, so it is safe.

4 Considerations for Sensitive Data

Some Web interactions involve sensitive data, such as passwords, credit card numbers, social security numbers, and bank account numbers (as in scenario 2).

To protect information carried by either GET or POST operations, it is often appropriate to use an underlying secure protocol such as the Secure Socket Layer [SSL3]. By using GET over SSL for safe operations, designers retain some of the benefits of URI addressability, even if they lose others (e.g., caching). Designers do need to consider the costs of using SSL, such as:

- Time required to establish an SSL connection.
- Memory required on the server to create a novel copy of (encrypted) data for each request.

In situations where the use of such protocols for security is inappropriate, designers MAY use POST to carry credentials or other information needed to authenticate an otherwise safe operation. For instance, a designer may require security beyond the protocol layer into the application layer (e.g., because software or data queues within a server site are not trusted, or because the application requires credentials not supported at a lower layer).

When the use of SSL is impractical, it may still be possible to keep some sensitive data out of a URI. For instance, designers can communicate authentication information in HTTP headers rather than in the query part of a URI.

Broad issues of data protection such as preventing unauthorized access to an individuals' bookmarks (or cookies, for that matter) lie beyond the scope of Web architecture.

5 Practical Considerations

Web application design should be informed by the above principles, but also by the relevant limitations. For instance, when designing a service that operates on a resource via URI, consider whether there are use cases where such operation would be inappropriate.

For example, the W3C HTML validation service allows for two scenarios (both using HTML forms). With the "safe" option, validation requests are done via a URI; the form uses GET, which gives the results a URI for bookmarks, links, etc. The second option allows clients to upload a document for validation. The form uses POST for that option, since:

- the document to be validated might be confidential; any link to the results of validating it (at the public W3C Web site) would divulge its contents.
- a URI that encoded the entire document would be at least as large as the document, and there's little or no use in linking to it, since the results will always be the same.

Whether or not GET with HTTP is used for the initial access, supplying a URI for subsequent access to the *same* information, e.g., using Content-Location, is useful.

5.1 Internationalization

Designers of HTML forms that accept non-ASCII characters have been challenged by some implementation limitations and gaps in specifications. Implementation limitations are length-related. Section section 17.13.4 of HTML 4.01 [HTML401] on multipart/form-data says, "The content type "application/x-www-form-urlencoded" is inefficient for sending large quantities of binary data or text containing non-ASCII characters."

This inefficiency is due to the octet-to-`%hh` escape conversion, combined with the fact that many characters need more than one octet to be encoded. But while somewhat inefficient, this is not a real obstacle to using GET for non-ASCII characters.

A more serious problem is that the mapping between characters and octets is not clearly specified beyond US-ASCII; refer to section 2.1 of [RFC2396]. For query parts resulting from filling in an HTML form, the default is to use the character encoding of the form. The definition of the accept-charset attribute on the form element in HTML 4.01 says, "The default value for this attribute is the reserved string 'UNKNOWN'. User agents may interpret this value as the character encoding that was used to transmit the document containing this FORM element."

The general direction to address this limitation is to converge to using UTF-8 for the mapping between characters and octets. The use of UTF-8 is already defined in various specifications, and we expect it to be adopted in future specifications and further deployed in due course. For instance, we expect XForms to specify that the encoding to be used in query parts is always UTF-8.

5.2 Ephemeral Limitations

While Web application design must take into account the limitations of technology that is widely deployed at present, it should not treat these as architectural invariants. The following is a list of limitations have already been largely resolved, or are likely to fade away as bugs are fixed and the scope of interoperable specifications expands.

URIs cannot be longer than 256 characters
: This was a limitation in some server implementations, and while servers continue to have limitations to prevent denial-of-service attacks, they are generally at least 4000 characters, and they evolve as the legitimate uses of application developers evolve.

GET requests are re-executed when the user uses the back button.
: This is not by design. Section 13.13 of [RFC2616] states: "History mechanisms and caches are different. In particular history mechanisms SHOULD NOT try to show a semantically transparent view of the current state of a resource. Rather, a history mechanism is meant to show exactly what the user saw at the time when the resource was retrieved."

If I visit a page via a secure protocol, and then follow a link to another page, the second site may have access to sensitive data in a URI.
: This is not by design. Section 15.1.3 of [RFC2616] states: "Because the source of a link might be private information or might reveal an otherwise private information source, it is strongly recommended that the user be able to select whether or not the "Referer" [sic] field is sent. For example, a browser client could have a toggle switch for browsing openly/anonymously, which would respectively enable/disable the sending of Referer and From information. Clients SHOULD NOT include a Referer header field in a (non-secure) HTTP request if the referring page was transferred with a secure protocol."

Search services will not index anything with a "?" in the URI.
: This was a heuristic to avoid infinite loops in some search service crawlers, but it was not an architectural constraint, and modern search services use more sophisticated heuristics to avoid loops.

6 Ongoing Work on GET in Web Services

Since the first publication of this finding, W3C's XML Protocol Working Group has enhanced SOAP Version 1.2 to support use of GET for safe operations (cf. section 4.1.2 of [SOAPADJUNCTS].) Specific conventions are also now suggested for use of GET in conjunction with SOAP Remote Procedure Calls (also in section 4.1.2) The SOAP HTTP binding (cf. section 7 of [SOAPADJUNCTS]) has been modified accordingly, and thus supports appropriate use of GET and POST in conjunction with the SOAP RPC Representation (cf. section 4 of [SOAPADJUNCTS]), as well as for non-RPC SOAP Request/Response messages (cf. section 6.2 and section 6.3 of [SOAPADJUNCTS]). Indeed, non-normative conventions are suggested which allow traditional Web servers (i.e., those not specifically enabled for SOAP support) to interoperate with SOAP clients using GET and resource representations in media type application/soap+xml (cf. section 7.1.3 of [SOAPADJUNCTS]).

Section 3 WSDL 1.2 Bindings [WSDL] provides a binding to HTTP GET, which makes it possible to respect the principle of using GET for safe operations. However, to represent safety in a more straightforward manner, it should be a property of operations themselves, not just a feature of bindings. As of publication of this document, the W3C Web Services Description Working Group has plans to specify a "safe" attribute for this purpose; see their issue 117 for more information.

7 Future Work

1. Reviewers of this finding asked about its applicability to HTTP PUT. Rather than address this question in this version of the finding, the TAG expects to address PUT semantics as part of issue putMediaType-38: "Relation of HTTP PUT to GET, and whether client headers to server are authoritative."

8 References

RFC2119
S. Bradner *Key words for use in RFCs to Indicate Requirement Levels,* RFC2119, March 1997. (See http://www.ietf.org/rfc/rfc2119.txt.)

RFC2396
T. Berners-Lee, R. Fielding, L. Masinter. *Uniform Resource Identifiers (URI): Generic Syntax*, RFC2396, August 1998. (See http://www.ietf.org /rfc/rfc2396.txt.)

RFC2518
Y. Goland, E. Whitehead, A. Faizi, S. Carter, D. Jensen *HTTP Extensions for Distributed Authoring -- WEBDAV*, RFC2518, February 1999. (See http://www.ietf.org/rfc/rfc2518.txt.)

RFC2616
R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee *Hypertext Transfer Protocol -- HTTP/1.1*, RFC2616, June 1999. (See http://www.ietf.org/rfc/rfc2616.txt.)

SOAPADJUNCTS
M. Gudgin, M. Hadley, N. Mendelsohn, J. J. Moreau, H. Frystyk Nielsen *SOAP Version 1.2 Part 2: Adjuncts*, W3C Proposed Recommendation, 7 May 2003. (See http://www.w3.org/TR/2003/PR-soap12-part2-20030507/.)

WSDL
J. J. Moreau, J. Schlimmer *Web Services Description Language (WSDL)*

Version 1.2: Bindings, W3C Working Draft, 24 January 2003. (See http://www.w3.org/TR/2003/WD-wsdl12-bindings-20030124/.)

TBL50K
T. Berners-Lee *Web Architecture from 50,000 feet.* (See http://www.w3.org /DesignIssues/Architecture.html.)

DEEPLINK
T. Bray *"Deep Linking" in the World Wide Web*, TAG finding, 17 Feb 2003. (See http://www.w3.org/2001/tag/doc/deeplinking-20030217.)

HTML401
D. Raggett, A. Le Hors, I. Jacobs *HTML 4.01 Specification*, W3C Recommendation, 24 Dec 1999. (See http://www.w3.org/TR/1999 /REC-html401-19991224/.)

SSL3
A. Frier, P. Karlton, and P. Kocher, *The SSL 3.0 Protocol*, Netscape Communications Corp., Nov 18, 1996. (See http://wp.netscape.com/eng/ssl3 /draft302.txt.)

RFC2310
K. Holtman, *The Safe Response Header* TUE, April 1998. (See http://www.ietf.org/rfc/rfc2310.txt.)

9 Acknowledgments

Dan Connolly edited the 10 June 2002 version of this finding. Some of the content from that revision is reused in this version, and Dan has been instrumental in subsequent discussions about this version.

Thanks to David Orchard, Larry Masinter, Paul Prescod, Roy Fielding, Martin Dürst, and others for their feedback in response to the 15 April 2002 call for review. Thanks also to Roger Cutler, Joseph Reagle, Stuart Williams, and Noah Mendelsohn for their suggestions. Many thanks to all reviewers for their contributions to this finding.