

Individual Submission  
Internet-Draft  
Intended status: Standards Track  
Expires: October 12, 2015

J. Reschke  
  
A. Malhotra  
  
J. Snell  
April 10, 2015

**HTTP SEARCH Method**  
**draft-snell-search-method-00**

Abstract

This specification updates the definition and semantics of the HTTP SEARCH request method previously defined by [[RFC5323](#)].

Status of This Memo

This Internet-Draft is submitted to IETF in full conformance with the provisions of [BCP 78](#) and [BCP 79](#).

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on October 12, 2015.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to [BCP 78](#) and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document.

## Table of Contents

<a href="#">1.</a>	<a href="#">Introduction</a>	<a href="#">2</a>
<a href="#">2.</a>	<a href="#">SEARCH</a>	<a href="#">4</a>
<a href="#">3.</a>	<a href="#">The "Accept-Search" Header Field</a>	<a href="#">5</a>
<a href="#">4.</a>	<a href="#">Examples</a>	<a href="#">5</a>
<a href="#">4.1.</a>	<a href="#">Simple SEARCH with a Direct Response</a>	<a href="#">5</a>
<a href="#">4.2.</a>	<a href="#">Simple SEARCH with Indirect Response (303 See Other)</a>	<a href="#">6</a>
<a href="#">5.</a>	<a href="#">Security Considerations</a>	<a href="#">6</a>
<a href="#">6.</a>	<a href="#">IANA Considerations</a>	<a href="#">7</a>
<a href="#">7.</a>	<a href="#">Normative References</a>	<a href="#">7</a>
	<a href="#">Authors' Addresses</a>	<a href="#">7</a>

## [1.](#) Introduction

This specification updates the HTTP SEARCH method originally defined in [[RFC5323](#)].

Many existing HTTP-based applications use the HTTP GET and POST methods in various ways to implement the functionality provided by SEARCH.

Using a GET request with some combination of query parameters included within the request URI (as illustrated in the example below) is arguably the most common mechanism for implementing search in web applications. With this approach, implementations are required to parse the request URI into distinct path (everything before the '?') and query elements (everything after the '?'). The path identifies the resource processing the query (in this case 'http://example.org/feed') while the query identifies the specific parameters of the search operation.

A typical use of HTTP GET for requesting a search

```
GET /feed?q=foo&limit=10&sort=-published HTTP/1.1
Host: example.org
```

While there are definite advantages to using GET requests in this manner, the disadvantages should not be overlooked. Specifically:

- o Without specific knowledge of the resource and server to which the GET request is being sent, there is no way for the client to know that a search operation is being requested. Identical requests sent to two different servers can implement entirely different semantics.
- o Encoding query parameters directly into the request URI effectively casts every possible combination of query inputs as



distinct resources. For instance, because mechanisms such as HTTP caching handle request URIs as opaque character sequences, queries such as 'http://example.org/?q=foo' and 'http://example.org/?q=Foo' will be treated as entirely separate resources even if they yield identical results.

- o While most modern browser and server implementations allow for long request URIs, there is no standardized minimum or maximum length for URIs in general. Many resource constrained devices enforce strict limits on the maximum number of characters that can be included in a URI. Such limits can prove impractical for large or complex query parameters.
- o Query expressions included within a request URI must either be restricted to relatively simple key value pairs or encoded such that the query can be safely represented in the limited character-set allowed by URL standards. Such encoding can add significant complexity, introduce bugs, or otherwise reduce the overall visibility of the query being requested.

As an alternative to using GET, many implementations make use of the HTTP POST method to perform queries, as illustrated in the example below. In this case, the input parameters to the search operation are passed along within the request payload as opposed to using the request URI.

A typical use of HTTP GET for requesting a search

```
POST /feed HTTP/1.1
Host: example.org
Content-Type: application/x-www-form-urlencoded

q=foo&limit=10&sort=-published
```

This variation, however, suffers from the same basic limitation as GET in that it is not readily apparent -- absent specific knowledge of the resource and server to which the request is being sent -- that a search operation is what is being requested. Web applications use the POST method for a wide variety of uses including the creation or modification of existing resources. Sending the request above to a different server, or even repeatedly sending the request to the same server could have dramatically different effects.

The SEARCH method provides a solution that spans the gap between the use of GET and POST. As with POST, the input to the query operation is passed along within the payload of the request rather than as part of the request URI. Unlike POST, however the semantics of the SEARCH method are specifically defined.



In this document, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in [[RFC2119](#)].

## 2. SEARCH

The SEARCH method is used to initiate a server-side search. Unlike the HTTP GET method, which requests that a server return a representation of the resource identified by the effective request URI (as defined by [[RFC7230](#)]), the SEARCH method is used by a client to ask the server to perform a query operation (described by the request payload) over some set of data scoped to the effective request URI. The payload returned in response to a SEARCH cannot be assumed to be a representation of the resource identified by the effective request URI.

The body of the request defines the query. Implementations MAY use a request body of any content type with the SEARCH method; however, for backwards compatibility with existing WebDAV implementations, SEARCH requests that use the text/xml or application/xml content types MUST be processed in exact accordance with the requirements established by [[RFC5323](#)].

SEARCH requests are both safe and idempotent with regards to the resource identified by the request URI. That is, the performance of a search is not expected to alter the state of the targeted resource. However, while processing a search request, a server can be expected to allocate computing and memory resources or even create additional HTTP resources through which the response to the search can be retrieved.

A successful response to a SEARCH request is expected to provide some indication as to the final disposition of the search operation. For instance, a successful search that yields no results can be represented by a 204 No Content response. If the response includes a body payload, the payload is expected to describe the results of the search operation. In some cases, the search arbiter might choose to respond indirectly to the SEARCH request by returning a 3xx Redirection with a Location header specifying an alternate Request URI from which the search results can be retrieved by using an HTTP GET request. Various non-normative examples of successful SEARCH responses are illustrated in [Section 4](#).

The response to a SEARCH request is not cacheable. It ought to be noted, however, that because SEARCH requests are safe and idempotent, responses to a SEARCH MUST NOT invalidate previously cached responses to other requests directed at the same effective request URI.



The semantics of the SEARCH method change to a "conditional SEARCH" if the request message includes an If-Modified-Since, If-Unmodified-Since, If-Match, If-None-Match, or If-Range header field ([\[RFC7232\]](#)). A conditional SEARCH requests that the query be performed only under the circumstances described by the conditional header field(s). It is important to note, however, that such conditions are evaluated against the state of the target resource itself as opposed to the collected results of the search operation.

### **3. The "Accept-Search" Header Field**

The "Accept-Search" response header field MAY be used by servers to directly signal support for the SEARCH method while identifying the specific query format Content-Type's that can be used.

Accept-Search = "Accept-Search" ":" 1#media-type

The Accept-Search header specifies a comma-separated listing of media types (with optional parameters) as defined by [\[RFC7231\]](#), [Section 3.1.1.1](#).

### **4. Examples**

The non-normative examples in this section make use of a simple, hypothetical plain-text based query syntax based on SQL with results returned as comma-separated values. This is done for illustration purposes only. Implementations are free to use any format they wish on both the request and response.

#### **4.1. Simple SEARCH with a Direct Response**

A simple SPAQRL query with a Direct Response:

```
SEARCH /contacts HTTP/1.1
Host: example.org
Content-Type: text/query
Accept: text/csv
```

```
select surname, givenname, email limit 10
```





Response:

```
HTTP/1.1 200 OK
Content-Type: text/csv

surname, givenname, email
Smith, John, john.smith@example.org
Jones, Sally, sally.jones@example.com
Dubois, Camille, camille.dubois@example.net
```

#### **[4.2.](#) Simple SEARCH with Indirect Response (303 See Other)**

A simple SPAQL query with an Indirect Response (303 See Other)

```
SEARCH /contacts HTTP/1.1
Host: example.org
Content-Type: text/query
Accept: text/csv

select surname, givenname, email limit 10
```

Response:

```
HTTP/1.1 303 See Other
Location: http://example.org/contacts/query123
```

Fetch Query Response:

```
GET /contacts/query123 HTTP/1.1
Host: example.org
```

Response:

```
HTTP/1.1 200 OK
Content-Type: text/csv

surname, givenname, email
Smith, John, john.smith@example.org
Jones, Sally, sally.jones@example.com
Dubois, Camille, camille.dubois@example.net
```

## **[5.](#) Security Considerations**

The SEARCH method is subject to the same general security considerations as all HTTP methods as described in [[RFC7231](#)].



## 6. IANA Considerations

IANA is requested to update the registration of the SEARCH method in the permanent registry at <<http://www.iana.org/assignments/http-methods>> (see [Section 8.1 of \[RFC7231\]](#)).

Method Name	Safe	Idempotent	Specification
SEARCH	Yes	Yes	<a href="#">Section 2</a>

## 7. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", [BCP 14](#), [RFC 2119](#), March 1997.
- [RFC4918] Dusseault, L., "HTTP Extensions for Web Distributed Authoring and Versioning (WebDAV)", [RFC 4918](#), June 2007.
- [RFC5323] Reschke, J., Reddy, S., Davis, J., and A. Babich, "Web Distributed Authoring and Versioning (WebDAV) SEARCH", [RFC 5323](#), November 2008.
- [RFC7230] Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", [RFC 7230](#), June 2014.
- [RFC7231] Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", [RFC 7231](#), June 2014.
- [RFC7232] Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests", [RFC 7232](#), June 2014.
- [RFC7234] Fielding, R., Nottingham, M., and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Caching", [RFC 7234](#), June 2014.

### Authors' Addresses

Julian Reschke

Email: [julian.reschke@greenbytes.de](mailto:julian.reschke@greenbytes.de)

Ashok Malhotra

Email: [ashok.malhotra@oracle.com](mailto:ashok.malhotra@oracle.com)



James M Snell

Email: [jasnell@gmail.com](mailto:jasnell@gmail.com)