

Network Working Group
Internet-Draft
Expires: September 17, 2005

M. Nottingham
March 19, 2005

POST Once Exactly (POE)
draft-nottingham-http-poe-00

Status of this Memo

By submitting this Internet-Draft, I certify that any applicable patent or other IPR claims of which I am aware have been disclosed, and any of which I become aware will be disclosed, in accordance with [RFC 3668](#).

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at <http://www.ietf.org/ietf/lid-abstracts.txt>.

The list of Internet-Draft Shadow Directories can be accessed at <http://www.ietf.org/shadow.html>.

This Internet-Draft will expire on September 17, 2005.

Copyright Notice

Copyright (C) The Internet Society (2005). All Rights Reserved.

Abstract

This specification describes a pattern of use that allows HTTP clients to automatically retry POST requests in a manner that assures no unintended side effects will take place, and defines mechanisms to allow implementations to automatically determine when such patterns are supported.

Internet-Draft

POE

March 2005

Table of Contents

1.	Introduction	3
1.1	Notational Conventions	4
1.2	Relationships to Other Specifications	4
2.	POE Resources	4
3.	The POE-Links HTTP Response Header	5
4.	The POE HTTP Request Header	6
5.	Example: Hyperlinking to a POE Resource	6
6.	Example: Manual Retry	7
7.	IANA Considerations	8
8.	Security Considerations	8
9.	Normative References	8
	Author's Address	8
A.	Implementation Notes	8
B.	Acknowledgements	9
	Intellectual Property and Copyright Statements	10

Internet-Draft

POE

March 2005

1. Introduction

A very common sight on the Web is an admonishment to only "click submit once." This is because there are some cases where an HTTP request's status on the server is not known to the client; in some instances, a delay in a response might be due to a temporary load on the server that doesn't interfere with the request's processing, if given enough time. In other cases, it could be caused by a server failing, or a network problem; in these cases, the request may need to be retried.

As a result, users (as well as automated agents) are placed in a quandary; without a response, they don't know whether their request has been successfully processed or not.

This isn't a problem for those HTTP requests that use idempotent methods (e.g., GET and PUT); by definition, if the same request is repeated (either by an impatient user pressing "reload" or an automated agent timing out), there will be no additional side effects.

However, non-idempotent HTTP methods, namely POST, may have additional side effects when the same request is sent more than once. To give a common example, POST is used by Web shopping baskets; when the user wishes to finalise their purchase, a POST request is sent to the server which processes the credit card transaction and fills the order.

If the POST is sent twice, there is the danger of a duplicate order being made.

[RFC 2616, Section 8.1.4](#), states:

Non-idempotent methods or sequences MUST NOT be automatically retried, although user agents MAY offer a human operator the

choice of retrying the request(s). Confirmation by user-agent software with semantic understanding of the application MAY substitute for user confirmation.

This specification provides applications with one means for gaining such semantic understanding. Specifically, it describes a pattern of use that allows clients to automatically retry POST requests in a manner that assures no unintended side effects will take place.

It also defines HTTP header-fields to allow implementations to automatically determine when such patterns are supported. These are not an exclusive mechanism; other means for identifying POE resources (e.g., format-specific markup on links in the entity body) may be

used.

Note that the pattern described here can also be used in situations where user intervention is required.

[1.1](#) Notational Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [BCP 14](#), [[RFC2119](#)], as scoped to those conformance targets.

This specification defines extensions to the HTTP/1.1 [[RFC2616](#)] specification. It uses the augmented BNF of that document, and relies on both the non-terminals defined in that document and other aspects of the HTTP/1.1 specification.

[1.2](#) Relationships to Other Specifications

There are a number of "reliable" application protocols layered on top of HTTP. Generally, these operate by treating HTTP as a transport protocol, rather than a transfer protocol; i.e., they seek to make any message delivered by HTTP reliable.

This specification is more focused; it only attempts to provide reliable POST semantics, because most other HTTP methods, being idempotent, do not require reliable delivery of either the request or the response, in an application-to-application sense.

This specification does not address methods other than POST; however, similar techniques might be applied to other non-idempotent methods. Likewise, it does not address non-idempotent sequences of requests, or guaranteed ordered delivery of requests.

2. POE Resources

A POE resource is an HTTP [[RFC2616](#)] resource with particular properties that clients can take advantage of to assure that POST requests are made once exactly, without requiring user intervention.

A POE resource will respond to a POST request successfully exactly once; that is, a server **MUST NOT** respond to a POST with a 2xx-series status code if it has been successfully POSTed to at some point in the past.

If a POE resource is subsequently POSTed to, it **SHOULD** respond with a 405 Method Not Allowed status code. Such a response **MUST NOT** include POST in the content of the Allow header.

If the response to a POST to a POE resource contains an entity body, that entity body **SHOULD** be available by GETting it from the resource, to allow the client to retrieve the state of the response in the case that a POST was successful, but the client did not receive the response.

For example, if the POST response is cacheable, the same entity body should be available through a subsequent GET; or, if the response has a 303 See Other status code, subsequent GETs should redirect to the indicated resource.

Note that a POE resource **MAY** require HTTP authentication, employ some forms of HTTP redirection, etc.

When a client sends a POST request to a POE resource, it **MAY** automatically (i.e., without user intervention) retry the request if the response is indeterminate (e.g., the connection is lost or times out). However, clients **MUST NOT** retry requests indefinitely.

User-agents **SHOULD** inform users that they are retrying a request automatically, and **MUST** inform users if they stop retrying.

[3.](#) The POE-Links HTTP Response Header

The POE-Links HTTP header is an entity-header field whose field-value is a comma-separated list of quoted URI-references [[RFC3986](#)] (without fragment identifiers) that the origin server asserts to be POE resources.

```
POE-Links = "POE-Links" ":" 1#( <"> POE-URI <"> )
POE-URI = absolute-URI | ( relative-part [ "?" query ] )
```

The contents of the POE-Links response header SHOULD correspond to links found in the content of the response body.

[[Some reviewers have suggested using the "Link" HTTP header with a "rel" attribute instead of a special-purpose HTTP header, thereby allowing link mechanisms in HTML to be used without further specification.

Whilst this is attractive, it would require the Link header to be standardized (it never was) and would introduce the potential need for implementations to scan body content, whereas this is not necessary in the current design (which does not preclude in-body identification of POE resources, but does not encourage it). Feedback on this design decision is sought.]]

[4.](#) The POE HTTP Request Header

The POE HTTP header is a request-header field whose field-value indicates the version of POE that a client supports.

```
POE = "POE" ":" POE-version
POE-version = 1*DIGIT
```

The POE-version that identifies this specification is "1".

Clients SHOULD send the POE request header on all requests to inform the decisions of the server. Servers MAY vary their content based upon the presence of the POE header (e.g., changing the operation of submit buttons, changing user instructions), and MAY use the POE

header to determine when to send a POE-Links response header.

5. Example: Hyperlinking to a POE Resource

A typical use case for POE is assuring that an order to a Web shopping basket is only POSTed once, so that only one order is made. Here, the client retrieves the state of the shopping basket to allow the user to review it prior to checking out;

```
C: GET /accounts/bob/shopping_basket HTTP/1.1
  Host: www.example.com
  POE: 1
  ...
```

The server returns a summary of the shopping basket in HTML, along with a form that allows the user to check out.

```
S: 200 OK HTTP/1.1
  POE-Links: "/accounts/bob/orders/12345"
  ...
  <form method="POST" action="/accounts/bob/orders/12345">
    <input type="submit" value="Submit Order 12345"/>
  </form>
  ...
```

When the user activates 'Submit Order 12345', it sends a POST request to what has been identified as a POE resource;

```
C: POST /accounts/bob/orders/12345 HTTP/1.1
  Host: www.example.com
  POE: 1
  ...
```

If the request is processed successfully, the server will respond

with

```
S: 200 OK HTTP/1.1
  ...
```

If the client does not receive that response, it can retry the POST request automatically;

```
C: POST /accounts/bob/orders/12345
Host: www.example.com
POE: 1
```

If the server had received and accepted the first request, it will respond with

```
S: 405 Method Not Allowed HTTP/1.1
Allow: GET
...
```

If the response status is "405 Method Not Allowed" the client can infer that the earlier POST succeeded. A 2xx response indicates that earlier POST did not succeed, but that this one has. When the client receives either of these responses, it knows that the request has been accepted, and it can stop retrying.

6. Example: Manual Retry

As noted, it is not necessary to use POE-specific HTTP headers in conjunction with this pattern. As such, it is possible to assure exactly one POST without special accommodation on the User-Agent, because the user can retry requests as needed.

Operation is the same as in the previous example, with the caveat that POE-specific HTTP headers are optional. If the user is unsure whether the POST has succeeded (e.g., the browser "hangs"), they can be instructed to do one of two things;

1. submit a GET to the POE resource to determine its state
2. re-submit their POST to the resource

Many implementations support one or both of these actions through "reloading" the page, or re-requesting the page manually (through an "address bar" UI widget or similar).

In either case, the server can return an unambiguous response as to the success of the POST, along with a user-friendly message in the entity body.

7. IANA Considerations

[[TBD: header registration templates]]

8. Security Considerations

Clients that indiscriminately retry requests may be used to manufacture a denial-of-service attack on third party Web sites; i.e., if a particular implementation retries aggressively or indefinitely, an attack can be engineered whereby a number of clients send a large number of requests to a particular site. This risk can be mitigated by reasonable retry periods, backoff algorithms, a reasonable maximum count for retries before manual intervention, and selective treatment of links across administrative domains (e.g., those with different authorities).

9 Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", [BCP 14](#), [RFC 2119](#), March 1997.
- [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", [RFC 2616](#), June 1999.
- [RFC3986] Berners-Lee, T., Fielding, R. and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, [RFC 3986](#), January 2005.

Author's Address

Mark Nottingham

EMail: mnot@pobox.com

URI: <http://www.mnot.net/>

[Appendix A.](#) Implementation Notes

Implementations of POE resources need to carefully consider how to guarantee that only one POST request is allowed, whilst still assuring that the desired side effects take place. In practice, this means that the information needed to enact the side effects needs to be stored durably and that the state of the resource needs to be updated, all in an atomic fashion.

Careful consideration needs to be given to the generation of URIs for POE resources, since they should never be reused. Long-lived, unique

Internet-Draft

POE

March 2005

identifiers can be generated by combining a date with one or more other sources of context. For example,

`http://www.example.com/users/bob/orders/2005/03/15/no2`

encodes the date ("2005/03/15") along with a system-unique userid ("bob") and an order number for that day (no2"), allowing the system to generate unique identifiers without storing more information than the userid and how many orders have been made in the current day.

Note that this is an example only; the structure of POE URIs, like other URIs, is opaque to consumers in the absence of authoritative information.

[Appendix B](#). Acknowledgements

The author would like to thank Yves Lafon, Roy Fielding and Rohit Khare for their feedback. None of the flaws in this specification are theirs.

Internet-Draft

POE

March 2005

Intellectual Property Statement

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in [BCP 78](#) and [BCP 79](#).

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at <http://www.ietf.org/ipr>.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.

Disclaimer of Validity

This document and the information contained herein are provided on an "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Copyright Statement

Copyright (C) The Internet Society (2005). This document is subject to the rights, licenses and restrictions contained in [BCP 78](#), and except as set forth therein, the authors retain all their rights.

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.