WEBDAV Working Group
INTERNET-DRAFT
<draft-goland-state-token-00.txt>

Y.Y. Goland
Microsoft Corporation
E.J. Whitehead, Jr.
U.C. Irvine

May 21, 1997

Expires November 21, 1997

State Tokens and Preconditions for HTTP

Status of this Memo

Abstract

There are times when a principal will want to predicate successful execution of a method on the current state of a resource. While HTTP/1.1 provides a mechanism for conditional execution of methods using entity tags via the "If-Match" and "If-None-Match" headers, the mechanism is not sufficiently extensible to express conditional statements involving more generic state indicators, such as lock tokens.

This draft defines the term "state token" as an identifier for a state of a resource. This draft defines requirements for state tokens and provides a state token syntax, along with two new headers which are used to express method preconditions using state tokens.

1 Overview

1.1 Problem Description

There are times when a principal will want to predicate successful execution of a method on the current state of a resource. While HTTP/1.1 [HTTP11] provides a mechanism for conditional execution of methods using entity tags via the "If-Match" and "If-None-Match" headers, the mechanism is not sufficiently extensible to express conditional statements involving more generic state indicators, such as lock tokens.

The fundamental issue with entity tags is that they can only be generated by a resource. However there are times when a client will want to be able to share state tokens between resources, potentially on different servers, as well as be able to generate certain types of lock tokens without first having to communicate with a server.

For example, a principal may wish to require that resource B have a certain state in order for a method to successfully execute on resource A. If the client submits an e-tag from resource B to resource A then A has no way of knowing that the e-tag is meant to describe resource B.

Another example occurs when a principal wishes to predicate the successful completion of a method on the absence of any locks on a resource. It is not sufficient to submit an "If-None-Match: *" as this refers to the existence of an entity, not of a lock.

This draft defines the term "state token" as an identifier for a state of a resource. The sections below define requirements for state tokens and provide a state token syntax, along with two new headers which can accept the new state token syntax.

1.2 Solution Requirements

1.2.1 Syntax

Self-Describing. A state token must be self describing such that upon inspecting a state token it is possible to determine what sort of state token it is, what resource(s) it applies to, and what state it represents.

This self-describing nature allows servers to accept tokens from other servers and potentially be able to coordinate state information cross resource and cross site through standardized protocols. For example, the execution of a request on resource A can be predicated on the state of resource B, where A and B are potentially on different servers.

Client Generable. The state token syntax must allow, when appropriate, for clients to generate a state token without having first communicated with a server.

One drawback of entity tags is that they are set by the server and there is no interoperable algorithm for calculating an entity tag. Consequently, a client cannot generate an entity tag from a particular state of a resource. However, a state token which encodes an MD5 state hash could be calculated

by a client based on a client-held state of a resource, and then submitted to a server in a conditional method invocation.

Another potential use for client generable state tokens is for a client to generate lock tokens with wild card fields, and hence be able to express conditionals such as: "only execute this GET if there are no write locks on this resource."

1.2.2 Conditonals

Universal. A solution must be applicable to all requests.

Positive and Negative. Conditional expressions must allow for the expression of both positive and negative state requirements.

2 State Token Syntax

State tokens are URLs [URL] employing the following syntax:

```
State-Token = "StateToken:" Type ":" Resources ":" State-Info
Type = "Type" "=" Caret-encoded-URL
Resources = "Res" "=" Caret-encoded-URL
Caret-encoded-URL = "^" Resource "^"
Resource = <A URI where all "^" characters are escaped>
State-Info = *(uchar | reserved) ; uchar, reserved defined section 3.2.1 of RFC 2068
```

This proposal has created a new URL scheme for state tokens because a state token names a network resource using its normal name, which is typically state-invariant, along with additional information that specifies a particular state of the resource. Encoding the state information into the native URL scheme of the network resource was not felt to be safe, since freedom from name space collisions could not be guaranteed. If this proposal is accepted, the StateToken URL scheme will need to be defined and registered with IANA.

State Token URLs begin with the URL scheme name "StateToken" rather than the name of the particular state token type they represent in order to make the URL self describing. Thus it is possible to examine the URL and know, at a minimum, that it is a state token.

Labeled name/value pairs are used within the token to allow new fields to be added. Processors of state tokens MUST be prepared to accept the fields in whatever order they are present and MUST ignore any fields they do not understand.

The "Type" field specifies the type of the state information encoded in the state token. A URL is used in order to avoid namespace collisions.

The "Res" field identifies the resource for which the state token specifies a particular state. Since commas and spaces are acceptable URL characters, a caret is used to delimit a URL. Since a caret is an acceptable URL

character, any instances of it must be escaped using the % escape convention.

The State-Info production is expanded upon in descriptions of specific state token types, and is intended to contain the state description information for a particular state token.

3 State Token Conditional Headers

3.1 If-State-Match

If-State-Match = "If-State-Match" ":" ("AND" | "OR") 1#("<" State-Token ">")

The If-State-Match header is intended to have similar functionality to the If-Match header defined in section 14.25 of RFC 2068.

If the AND keyword is used and all of the state tokens identify the state of the resource, then the server MAY perform the requested method. If the OR keyword is used and any of the state tokens identifies the current state of the resource, then server MAY perform the requested method. If neither of the keyword requirements is met, the server MUST NOT perform the requested method, and MUST return a 412 (Precondition Failed) response.

3.2 If-None-State-Match

If-None-State-Match = "If-None-State-Match" 1#("<" State-Token ">")

The If-None-State-Match header is intended to have similar functionality to the If-None-Match header defined in section 14.26 of RFC 2068.

If any of the state tokens identifies the current state of the resource, the server MUST NOT perform the requested method. Instead, if the request method was GET, HEAD, INDEX, or GETMETA, the server SHOULD respond with a 304 (Not Modified) response, including the cache-related entity-header fields (particularly ETag) of the current state of the resource. For all other request methods, the server MUST respond with a status of 412 (Precondition Failed).

If none of the state tokens identifies the current state of the resource, the server MAY perform the requested method.

Note that the "AND" and "OR" keywords specified with the If-State-Match header are intentionally not defined for If-None-State-Match, because this functionality is not required.

4 E-Tags

E-tags have already been deployed using the If-Match and If-None-Match headers. Introducing two mechanisms to express e-tags would only confuse matters, therefore e-tags should continue to be expressed using quoted strings and the If-Match and If-None-Match headers.

5 References

[URL] T. Berners-Lee, L. Masinter, M. McCahill. "Uniform Resource Locators (URL)", RFC 1738, CERN, Xerox PARC, University of Minnesota, December 1994.

[HTTP11] R. Fielding, J. Gettys, J. C. Mogul, H. Frystyk, and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2068, U.C. Irvine, DEC, MIT/LCS, January 1997.

6 Authors' Addresses

Yaron Y. Goland
Microsoft Corporation
One Microsoft Way
Bldg. 27N/3445
Redmond, WA 98052-6399

Fax: (206) 936 7329
EMail: yarong@microsoft.com

E. James Whitehead, Jr.
Department of Information and Computer Science
University of California
Irvine, CA 92697-3425

Fax: (714) 824-4056
EMail: ejw@ics.uci.edu