PERSISTENT CLIENT STATE HTTP COOKIES

Preliminary Specification - Use with caution

INTRODUCTION

Cookies are a general mechanism which server side connections (such as CGI scripts) can use to both store and retrieve information on the client side of the connection. The addition of a simple, persistent, client-side state significantly extends the capabilities of Web-based client/server applications.

OVERVIEW

A server, when returning an HTTP object to a client, may also send a piece of state information which the client will store. Included in that state object is a description of the range of URLs for which that state is valid. Any future HTTP requests made by the client which fall in that range will include a transmittal of the current value of the state object from the client back to the server. The state object is called a **cookie**, for no compelling reason.

This simple mechanism provides a powerful new tool which enables a host of new types of applications to be written for web-based environments. Shopping applications can now store information about the currently selected items, for fee services can send back registration information and free the client from retyping a user-id on next connection, sites can store per-user preferences on the client, and have the client supply those preferences every time that site is connected to.

SPECIFICATION

A cookie is introduced to the client by including a **Set-Cookie** header as part of an HTTP response, typically this will be generated by a CGI script.

Syntax of the Set-Cookie HTTP Response Header

This is the format a CGI script would use to add to the HTTP headers a new piece of data which is to be stored by the client for later retrieval.

```
Set-Cookie: NAME=VALUE; expires=DATE;
path=PATH; domain=DOMAIN_NAME; secure
```

NAME=VALUE

This string is a sequence of characters excluding semi-colon, comma and white space. If there is a need to place such data in the name or value, some encoding method such as URL style %XX encoding is recommended, though no encoding is defined or required.

This is the only required attribute on the **Set-Cookie** header.

expires=*DATE*

The **expires** attribute specifies a date string that defines the valid life time of that cookie. Once the expiration date has been reached, the cookie will no longer be stored or given out.

The date string is formatted as:

```
Wdy, DD-Mon-YYYY HH:MM:SS GMT
```

This is based on RFC 822, RFC 850, RFC 1036, and RFC 1123, with the variations that the only legal time zone is **GMT** and the separators between the elements of the date must be dashes.

expires is an optional attribute. If not specified, the cookie will expire when the user's session ends.

Note: There is a bug in Netscape Navigator version 1.1 and earlier. Only cookies whose **path** attribute is set explicitly to "/" will be properly saved between sessions if they have an **expires** attribute.

domain=*DOMAIN_NAME*

When searching the cookie list for valid cookies, a comparison of the **domain** attributes of the cookie is made with the Internet domain name of the host from which the URL will be fetched. If there is a tail match, then the cookie will go through **path** matching to see if it should be sent. "Tail matching" means that **domain** attribute is matched against the tail of the fully qualified domain name of the host. A **domain** attribute of "acme.com" would match host names "anvil.acme.com" as well as "shipping.crate.acme.com".

Only hosts within the specified domain can set a cookie for a domain and domains must have at least two (2) or three (3) periods in them to prevent domains of the form: ".com", ".edu", and "va.us". Any domain that fails within one of the seven special top level domains listed below only require two periods. Any other domain requires at least three. The seven special top level domains are: "COM", "EDU", "NET", "ORG", "GOV", "MIL", and "INT".

The default value of **domain** is the host name of the server which generated the cookie response.

path=*PATH*

The **path** attribute is used to specify the subset of URLs in a domain for which the cookie is valid. If a cookie has already passed **domain** matching, then the pathname component of the URL is compared with the path attribute, and if there is a match, the cookie is considered valid and is sent along with the URL request. The path "/foo" would match "/foobar" and "/foo/bar.html". The path "/" is the most general path.

If the **path** is not specified, it as assumed to be the same path as the document being described by the header which contains the cookie.

secure

If a cookie is marked **secure**, it will only be transmitted if the communications channel with the host is a secure one. Currently this means that secure cookies will only be sent to HTTPS (HTTP over SSL) servers.

If **secure** is not specified, a cookie is considered safe to be sent in the clear over unsecured channels.

Syntax of the Cookie HTTP Request Header

When requesting a URL from an HTTP server, the browser will match the URL against all cookies and if any of them match, a line containing the name/value pairs of all matching cookies will be included in the HTTP request. Here is the format of that line:

```
Cookie: NAME1=OPAQUE_STRING1; NAME2=OPAQUE_STRING2 ...
```

Additional Notes

- Multiple **Set-Cookie** headers can be issued in a single server response.
- Instances of the same path and name will overwrite each other, with the latest instance taking precedence. Instances of the same path but different names will add additional mappings.

- Setting the path to a higher-level value does not override other more specific path mappings. If there are multiple matches for a given cookie name, but with separate paths, all the matching cookies will be sent. (See examples below.)

- The expires header lets the client know when it is safe to purge the mapping but the client is not required to do so. A client may also delete a cookie before it's expiration date arrives if the number of cookies exceeds its internal limits.

- When sending cookies to a server, all cookies with a more specific path mapping should be sent before cookies with less specific path mappings. For example, a cookie "name1=foo" with a path mapping of "/" should be sent after a cookie "name1=foo2" with a path mapping of "/bar" if they are both to be sent.

- There are limitations on the number of cookies that a client can store at any one time. This is a specification of the minimum number of cookies that a client should be prepared to receive and store.
    - 300 total cookies
    - 4 kilobytes per cookie, where the name and the OPAQUE_STRING combine to form the 4 kilobyte limit.
    - 20 cookies per server or domain. (note that completely specified hosts and domains are treated as separate entities and have a 20 cookie limitation for each, not combined)

  Servers should not expect clients to be able to exceed these limits. When the 300 cookie limit or the 20 cookie per server limit is exceeded, clients should delete the least recently used cookie. When a cookie larger than 4 kilobytes is encountered the cookie should be trimmed to fit, but the name should remain intact as long as it is less than 4 kilobytes.

- If a CGI script wishes to delete a cookie, it can do so by returning a cookie with the same name, and an **expires** time which is in the past. The path and name must match exactly in order for the expiring cookie to replace the valid cookie. This requirement makes it difficult for anyone but the originator of a cookie to delete a cookie.

- When caching HTTP, as a proxy server might do, the **Set-cookie** response header should never be cached.

- If a proxy server receives a response which contains a **Set-cookie** header, it should propagate the **Set-cookie** header to the client, regardless of whether the response was 304 (Not Modified) or 200 (OK).

  Similarly, if a client request contains a Cookie: header, it should be forwarded through a proxy, even if the conditional If-modified-since request is being made.

EXAMPLES

Here are some sample exchanges which are designed to illustrate the use of cookies.

First Example transaction sequence:

Client requests a document, and receives in the response:

```
Set-Cookie: CUSTOMER=WILE_E_COYOTE; path=/; expires=Wednesday, 09-Nov-99 23:12:40 GMT
```

When client requests a URL in path "/" on this server, it sends:

```
Cookie: CUSTOMER=WILE_E_COYOTE
```

Client requests a document, and receives in the response:

```
Set-Cookie: PART_NUMBER=ROCKET_LAUNCHER_0001; path=/
```

When client requests a URL in path "/" on this server, it sends:

```
Cookie: CUSTOMER=WILE_E_COYOTE; PART_NUMBER=ROCKET_LAUNCHER_0001
```

Client receives:

```
Set-Cookie: SHIPPING=FEDEX; path=/foo
```

When client requests a URL in path "/" on this server, it sends:

```
Cookie: CUSTOMER=WILE_E_COYOTE; PART_NUMBER=ROCKET_LAUNCHER_0001
```

When client requests a URL in path "/foo" on this server, it sends:

```
Cookie: CUSTOMER=WILE_E_COYOTE; PART_NUMBER=ROCKET_LAUNCHER_0001; SHIPPING=FEDEX
```

Second Example transaction sequence:

Assume all mappings from above have been cleared.

Client receives:

```
Set-Cookie: PART_NUMBER=ROCKET_LAUNCHER_0001; path=/
```

When client requests a URL in path "/" on this server, it sends:

```
Cookie: PART_NUMBER=ROCKET_LAUNCHER_0001
```

Client receives:

```
Set-Cookie: PART_NUMBER=RIDING_ROCKET_0023; path=/ammo
```

When client requests a URL in path "/ammo" on this server, it sends:

```
Cookie: PART_NUMBER=RIDING_ROCKET_0023; PART_NUMBER=ROCKET_LAUNCHER_0001
```

NOTE: There are two name/value pairs named "PART_NUMBER" due to the inheritance of the "/" mapping in addition to the "/ammo" mapping.
