NAME

CGI::Cookie - Interface to HTTP Cookies

SYNOPSIS

```
use CGI qw/:standard/;
use CGI::Cookie;

# Create new cookies and send them
$cookie1 = CGI::Cookie->new(-name=>'ID',-value=>123456);
$cookie2 = CGI::Cookie->new(-name=>'preferences',
                            -value=>{ font => Helvetica,
                                      size => 12 }
                            );
print header(-cookie=>[$cookie1,$cookie2]);

# fetch existing cookies
%cookies = CGI::Cookie->fetch;
$id = $cookies{'ID'}->value;

# create cookies returned from an external source
%cookies = CGI::Cookie->parse($ENV{COOKIE});
```

DESCRIPTION

CGI::Cookie is an interface to HTTP/1.1 cookies, an innovation that allows Web servers to store persistent information on the browser's side of the connection. Although CGI::Cookie is intended to be used in conjunction with CGI.pm (and is in fact used by it internally), you can use this module independently.

For full information on cookies see

```
http://tools.ietf.org/html/rfc2109
http://tools.ietf.org/html/rfc2965
http://tools.ietf.org/html/draft-ietf-httpstate-cookie
```

USING CGI::Cookie

CGI::Cookie is object oriented. Each cookie object has a name and a value. The name is any scalar value. The value is any scalar or array value (associative arrays are also allowed). Cookies also have several optional attributes, including:

1. expiration date

The expiration date tells the browser how long to hang on to the cookie. If the cookie specifies an expiration date in the future, the browser will store the cookie information in a disk file and return it to the server every time the user reconnects (until the expiration date is reached). If the cookie species an expiration date in the past, the browser will remove the cookie from the disk file. If the expiration date is not specified, the cookie will persist only until the user quits the browser.

2. domain

This is a partial or complete domain name for which the cookie is valid. The browser will return the cookie to any host that matches the partial domain name. For example, if you specify a domain name of ".capricorn.com", then the browser will return the cookie to Web servers running on any of the machines "www.capricorn.com", "ftp.capricorn.com", "feckless.capricorn.com", etc. Domain names must contain at least two periods to prevent attempts to match on top level domains like ".edu". If no domain is specified, then the browser

will only return the cookie to servers on the host the cookie originated from.

3. path

If you provide a cookie path attribute, the browser will check it against your script's URL before returning the cookie. For example, if you specify the path "/cgi-bin", then the cookie will be returned to each of the scripts "/cgi-bin/tally.pl", "/cgi-bin/order.pl", and "/cgi-bin/customer_service/complain.pl", but not to the script "/cgi-private/site_admin.pl". By default, the path is set to "/", so that all scripts at your site will receive the cookie.

4. secure flag

If the "secure" attribute is set, the cookie will only be sent to your script if the CGI request is occurring on a secure channel, such as SSL.

5. httponly flag

If the "httponly" attribute is set, the cookie will only be accessible through HTTP Requests. This cookie will be inaccessible via JavaScript (to prevent XSS attacks).

This feature is only supported by recent browsers like Internet Explorer 6 Service Pack 1, Firefox 3.0 and Opera 9.5 (and later of course).

See these URLs for more information:

```
http://msdn.microsoft.com/en-us/library/ms533046.aspx
http://www.owasp.org/index.php/HTTPOnly#Browsers_Supporting_HTTPOnly
```

Creating New Cookies

```
my $c = CGI::Cookie->new(-name    =>  'foo',
                         -value   =>  'bar',
                         -expires =>  '+3M',
                         -domain  =>  '.capricorn.com',
                         -path    =>  '/cgi-bin/database',
                         -secure  =>  1
                       );
```

Create cookies from scratch with the **new** method. The **-name** and **-value** parameters are required. The name must be a scalar value. The value can be a scalar, an array reference, or a hash reference. (At some point in the future cookies will support one of the Perl object serialization protocols for full generality).

-expires accepts any of the relative or absolute date formats recognized by CGI.pm, for example "+3M" for three months in the future. See CGI.pm's documentation for details.

-max-age accepts the same data formats as **-expires**, but sets a relative value instead of an absolute like **-expires**. This is intended to be more secure since a clock could be changed to fake an absolute time. In practice, as of 2011, `-max-age` still does not enjoy the widespread support that `-expires` has. You can set both, and browsers that support `-max-age` should ignore the `Expires` header. The drawback to this approach is the bit of bandwidth for sending an extra header on each cookie.

-domain points to a domain name or to a fully qualified host name. If not specified, the cookie will be returned only to the Web server that created it.

-path points to a partial URL on the current server. The cookie will be returned to all URLs beginning with the specified path. If not specified, it defaults to '/', which returns the cookie to all pages at your site.

-secure if set to a true value instructs the browser to return the cookie only when a cryptographic protocol is in use.

-httponly if set to a true value, the cookie will not be accessible via JavaScript.

For compatibility with Apache::Cookie, you may optionally pass in a mod_perl request object as the first argument to `new()`. It will simply be ignored:

```
my $c = CGI::Cookie->new($r,
                         -name    =>  'foo',
                         -value   =>  ['bar','baz']);
```

Sending the Cookie to the Browser

The simplest way to send a cookie to the browser is by calling the bake() method:

```
$c->bake;
```

This will print the Set-Cookie HTTP header to STDOUT using CGI.pm. CGI.pm will be loaded for this purpose if it is not already. Otherwise CGI.pm is not required or used by this module.

Under mod_perl, pass in an Apache request object:

```
$c->bake($r);
```

If you want to set the cookie yourself, Within a CGI script you can send a cookie to the browser by creating one or more Set-Cookie: fields in the HTTP header. Here is a typical sequence:

```
my $c = CGI::Cookie->new(-name    =>  'foo',
                         -value   =>  ['bar','baz'],
                         -expires =>  '+3M');


print "Set-Cookie: $c\n";
print "Content-Type: text/html\n\n";
```

To send more than one cookie, create several Set-Cookie: fields.

If you are using CGI.pm, you send cookies by providing a -cookie argument to the header() method:

```
print header(-cookie=>$c);
```

Mod_perl users can set cookies using the request object's header_out() method:

```
$r->headers_out->set('Set-Cookie' => $c);
```

Internally, Cookie overloads the "" operator to call its as_string() method when incorporated into the HTTP header. as_string() turns the Cookie's internal representation into an RFC-compliant text representation. You may call as_string() yourself if you prefer:

```
print "Set-Cookie: ",$c->as_string,"\n";
```

Recovering Previous Cookies

```
%cookies = CGI::Cookie->fetch;
```

fetch returns an associative array consisting of all cookies returned by the browser. The keys of the array are the cookie names. You can iterate through the cookies this way:

```
%cookies = CGI::Cookie->fetch;
for (keys %cookies) {
   do_something($cookies{$_});
       }
```

In a scalar context, fetch() returns a hash reference, which may be more efficient if you are manipulating multiple cookies.

CGI.pm uses the URL escaping methods to save and restore reserved characters in its cookies. If you are trying to retrieve a cookie set by a foreign server, this escaping method may trip you up. Use raw_fetch() instead, which has the same semantics as fetch(), but performs no unescaping.

You may also retrieve cookies that were stored in some external form using the parse() class method:

```
$COOKIES = `cat /usr/tmp/Cookie_stash`;
%cookies = CGI::Cookie->parse($COOKIES);
```

If you are in a mod_perl environment, you can save some overhead by passing the request object to fetch() like this:

```
CGI::Cookie->fetch($r);
```

If the value passed to parse() is undefined, an empty array will returned in list context, and an empty hashref will be returned in scalar context.

Manipulating Cookies

Cookie objects have a series of accessor methods to get and set cookie attributes. Each accessor has a similar syntax. Called without arguments, the accessor returns the current value of the attribute. Called with an argument, the accessor changes the attribute and returns its new value.

name()

Get or set the cookie's name. Example:

```
$name = $c->name;
$new_name = $c->name('fred');
```

value()

Get or set the cookie's value. Example:

```
$value = $c->value;
@new_value = $c->value(['a','b','c','d']);
```

value() is context sensitive. In a list context it will return the current value of the cookie as an array. In a scalar context it will return the **first** value of a multivalued cookie.

domain()

Get or set the cookie's domain.

path()

Get or set the cookie's path.

expires()

Get or set the cookie's expiration time.

AUTHOR INFORMATION

Address bug reports and comments to: lstein@cshl.org

BUGS

This section intentionally left blank.

SEE ALSO

CGI::Carp, *CGI*

RFC 2109, *RFC 2695*