

# Httper

## Web HTTP Client in Scheme

Version 0.3  
22 August 2004

Neil W. Van Dyke  
neil@neilvandyke.org

<http://www.neilvandyke.org/httper/>

Copyright © 2003-2004 Neil W. Van Dyke. This program is Free Software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version. This program is distributed in the hope that it will be useful, but without any warranty; without even the implied warranty of merchantability or fitness for a particular purpose. See the GNU Lesser General Public License [LGPL] for details.

## 1 Introduction

*Note: No further work is planned on Httper, since the author plans to soon obsolete it with a more portable HTTP client library. This version has been released to migrate from `UriFrame` to `uri.scm`, the latter of which will be used by the new HTTP library. New development is encouraged to use `net/url.ss` rather than `Httper`.*

Httper is an HTTP 1.1 [RFC2616] client library for PLT Scheme. It provides multiple levels of abstraction over parts of the HTTP protocol, with the intent that it be convenient for programs with both simple and involved HTTP needs.

Httper requires the packages [Uri.scm] and [SRFI-19-PLT].

## 2 Option Parameters

Several option parameters affect behavior. The majority of applications will either accept the default values or set the parameters globally.

**current-http-user-agent** [Parameter]

String for value to send for the HTTP `User-Agent` header, or `#f` if no header is to be sent. The default is "Mozilla/5.0 (compatible;)", which seems to be accepted by most servers that improperly refuse to work with browsers that do not identify themselves as being particular versions of particular vendor's browsers.

**current-max-http-redirects** [Parameter]

The maximum count of HTTP redirections [RFC2616 sec. 10.3] to follow automatically, for the `execute/follow` method of `http-request%` and for convenience procedures like `http-get`. This is a non-negative integer, defaulting to 5.

**current-record-http-times?** [Parameter]  
Boolean value for whether or not to record the times at which specific states in an HTTP protocol conversation were entered. If **#f**, these times can be accessed by methods of the `http-request%` object. The default is **#f**, to not record times.

### 3 Exceptions

The exceptions hierarchy is expected to change somewhat in future versions of `Httper`, so this part of the documentation is intentionally vague.

**exn:httpper** [Exception]  
**exn:httpper:proto** *uri* [Exception]  
**exn:httpper:proto:http** *detail* [Exception]  
**exn:httpper:proto:transport** *sub-exn* [Exception]  
**exn:httpper:redirects** [Exception]  
**exn:httpper:status** *code reason* [Exception]  
**exn:httpper:time** [Exception]  
**exn:httpper:uri** *uri* [Exception]

`Httper`-specific exceptions types, with the subtype hierarchy implied by the colons in the name.

### 4 HTTP Date/Time Stamps

The `http-datetime->date` and `http-datetime->date-or-false` procedures parse an HTTP full date/time stamp [RFC2616 sec. 3.3.1] into a `struct:date` (or `struct:srfi19-date`) object. The standard three formats, and some variations on them, are supported. Normally, the procedures will not be called directly from a program — methods of `http-request%` such as `response-last-modified-header/date` can usually be used instead, and may be implemented more efficiently.

**http-datetime->date** *str* [Procedure]  
Parses the HTTP date/time string *str* into a `struct:date` (or `struct:srfi19-date`) object. The exception `exn:httpper:time` is raised on an error. For example:

```
(map (lambda (str) (date->string (http-datetime->date str)))
      '("Sun, 06 Nov 1994 08:49:37 GMT"
        "Sunday, 06-Nov-94 08:49:37 GMT"
        "Sun Nov 6 08:49:37 1994"))
⇒
("Sun Nov 06 08:49:37Z 1994"
 "Sun Nov 06 08:49:37Z 1994"
 "Sun Nov 06 08:49:37Z 1994")
```

**http-datetime->date-or-false** *str* [Procedure]  
Like `http-datetime->date`, except yields **#f** if the string cannot be parsed, rather than raising an exception.

## 5 Header Parsing

Internal to `Httper`, HTTP response headers are parsed from an input port using the `parse-http-response-headers` procedure. The exact implementation and representation used might change in a future version of `Httper`. Application programs will normally access individual headers through methods of `http-request%`.

**parse-http-response-headers** *in* [Procedure]

Read HTTP headers from input port *in* and emit an association list of name-value pairs. The `car` of the pair is the header name with all alphabetic characters forced to lowercase. The `cdr` of the pair is the header value, which is either a string with leading and trailing whitespace removed, or, in the unusual case of multi-line values, a list of such strings, one string per line. (The rationale for this representation has to do with time and space optimization for the usual cases.) For example:

```
(parse-http-response-headers
 (open-input-string (string-append "Alpha:  1 1 1 \r\n"
                                   "Bravo:   222  \r\n"
                                   "        333  \r\n"
                                   "Charlie: 444  \r\n"
                                   "        555  \r\n"
                                   "        666  \r\n"
                                   "Delta:  7 7 7 \r\n"
                                   "\r\n")))
⇒
```

```
((("alpha" . "1 1 1")
 ("bravo" . ("222" "333"))
 ("charlie" . ("444" "555" "666"))
 ("delta" . "7 7 7")))
```

## 6 Base Request Class

Each HTTP request *method* [RFC2616 sec. 5.1.1] supported by `Httper` is implemented as a subclass of `http-request%`.

**http-request%** [Class]

Base class for HTTP request. It represents information about the request and response, and has behavior for executing the request. It has various methods for querying response information. It has the optional initialization fields:

`uri` HTTP URL of the request, as an `http-uri<%>` object. Default is `#f`.

`prior-request`

If this request is in response to an HTTP redirection, the preceding request that resulted in the redirection, as an `http-request%` object; otherwise `#f`. Default is `#f`.

**request-referrer**

The referrer URL string to be given as the value of the `Referer` (sic) HTTP header in this request. Note that this is a string, not a `uri<%>` object. The default is `#f`, for no referrer.

**uri** [Method on `http-request%`]

**prior-request** [Method on `http-request%`]

**request-referrer** [Method on `http-request%`]

**set-uri!** *v* [Method on `http-request%`]

**set-prior-request!** *v* [Method on `http-request%`]

**set-request-referrer!** *v* [Method on `http-request%`]

Methods to get and set the initialization fields. These fields should not be set after the HTTP request has been initiated.

**initiate-time-seconds** [Method on `http-request%`]

**connect-time-seconds** [Method on `http-request%`]

**status-time-seconds** [Method on `http-request%`]

**headers-time-seconds** [Method on `http-request%`]

Methods to fetch the times at which the HTTP protocol entered certain states, if the value of the `current-record-http-times?` parameter was true at the time the state transition occurred. These time values can be used as input to procedures like `seconds->date-utc` and `seconds->date`. If the states have not yet been reached, or the time was not recorded, then the value yielded by the method is `#f`.

**response-version-major** [Method on `http-request%`]

**response-version-minor** [Method on `http-request%`]

**response-status-code** [Method on `http-request%`]

**response-status-class** [Method on `http-request%`]

**response-status-reason** [Method on `http-request%`]

Once the request execution has received and parsed the response status line, these methods yield information from the status line. `response-version-major` and `response-version-minor` yield the major and minor HTTP versions of the server's response, as integers. `response-status-code` yields the response status as an integer (e.g., 200, 404). `response-status-class` yields the class of the response code one of the symbols: `informational`, `success`, `redirection`, `client-error`, `server-error`. `response-status-reason` is a string with the human-readable reason phrase sent by the server.

**find-response-header/string-or-list** *name* [Method on `http-request%`]

**find-response-header/string** *name* [Method on `http-request%`]

**find-response-header/date** *name* [Method on `http-request%`]

These methods yield a response header value for a given header name in a requested data type, where *name* is the string name of the header, in all lowercase (e.g., "last-modified"). These methods can be used when there is no convenience method for the desired header. If the desired header cannot be found, or its value cannot be converted to the desired type, then `#f` is yielded. If multiple headers named *name* occurred in the response, the value of only one of them is used, selected in an unspecified way.

The suffix following / in the method name denotes the desired type: `find-response-header/string-or-list` returns the value as a string or, if the header was multi-line, as a list of strings. `find-response-header/string` returns the value as a single string. `find-response-header/date` returns the value as a `struct:date` or `struct:srfi19-date`.

<b>response-accept-ranges-header</b>	[Method on <code>http-request%</code> ]
<b>response-age-header</b>	[Method on <code>http-request%</code> ]
<b>response-date-header</b>	[Method on <code>http-request%</code> ]
<b>response-etag-header</b>	[Method on <code>http-request%</code> ]
<b>response-last-modified-header</b>	[Method on <code>http-request%</code> ]
<b>response-location-header</b>	[Method on <code>http-request%</code> ]
<b>response-proxy-authenticate-header</b>	[Method on <code>http-request%</code> ]
<b>response-retry-after-header</b>	[Method on <code>http-request%</code> ]
<b>response-server-header</b>	[Method on <code>http-request%</code> ]
<b>response-vary-header</b>	[Method on <code>http-request%</code> ]
<b>response-www-authenticate-header</b>	[Method on <code>http-request%</code> ]

Yields a response header value as a string, or `#f` if the header did not occur. The method names are of the format `response-name-header`, where *name* is the name of the corresponding HTTP header in lowercase. For example, the value of the `WWW-Authenticate` response header is yielded by the `response-www-authenticate-header` method.

<b>response-date-header/date</b>	[Method on <code>http-request%</code> ]
<b>response-last-modified-header/date</b>	[Method on <code>http-request%</code> ]

For response headers with date/time values, yields the value as a `struct:date` (or `struct:srfi19-date`), or `#f` if the header did not occur or the value could not be parsed.

<b>execute/nocheck</b>	[Method on <code>http-request%</code> ]
<b>execute/nofollow</b>	[Method on <code>http-request%</code> ]
<b>execute/follow</b> <i>follow</i>	[Method on <code>http-request%</code> ]

Execute the HTTP request. `execute/nocheck` is the most basic of the three methods, opening the connection, sending the request, and receiving the status line and headers. Yields `void`.

`execute/nofollow` performs all the behavior of `execute/nocheck`, then checks the status code and raises the `exn:httpper:status` exception if the status class is `client-error` or `server-error`. Yields `void`.

`execute/follow` performs all the behavior of `execute/nofollow`, then, if the response was a redirection, can make additional requests to follow the redirection. The maximum count of redirections to follow this way is given by *follow*, which is either a non-negative integer, or `#t`, meaning the value of the `current-max-http-redirects`. For each redirection followed, the ports for the current request are closed, a new `http-request%` object is constructed with its `prior-request` field referencing the prior `http-request%`, and the new request is executed. If a redirection response is received after the limit of *follow* has been encountered, then exception `exn:http:redirects`

is raised. This method yields the `http-request%` object of the lastmost request. Programs will want to capture this value, such as by:

```
(let ((ultimate-request (send request execute/follow #t)))
  ...)
```

**input-port** [Method on `http-request%`  
**output-port** [Method on `http-request%`

These methods yield an input port and an output port, respectively, to the server, or `#f` if the port is known by the object to be closed. The execution methods of the `http-request%` subclasses will close one or both of these ports under some conditions, usually leaving them open only if the program is expected to need them. For example, the input port will be left open when executing the HTTP GET method, so that the program can read the response content from the server.

**close-input** [Method on `http-request%`  
**close-output** [Method on `http-request%`  
**close-ports** [Method on `http-request%`

Closes the response input port, output port, or both, respectively.

**make-redirect-request** [Method on `http-request%`

When execution of the request has resulted in a redirection response, invoking this method will yield a new `http-request%` object that, when executed, will follow the redirection. An error is signaled if the new request object cannot be created. Most programs will not invoke this method directly, but rather will use the `execute/follow` method or one of the higher-level convenience procedures.

Note that, should the server erroneously supply a relative for the redirection, this method will first resolve that URL relative to the original request URL, and then attempt to force the URL absolute.

## 7 Request Method Classes

This part of the documentation can be very brief until more HTTP method-specific behavior is added.

**http-get-request%** [Class]  
**http-head-request%** [Class]

HTTP method-specific classes. After execution of an `http-get-request%` request, the output port is closed. After execution of an `http-head-request%` request, both input and output ports are closed.

## 8 Convenience Procedures

A few high-level convenience procedures are provided for instantiating and executing HTTP request objects.

**http-get** *url* [*follow* [*referrer*]] [Procedure]

The `http-get` procedure creates an `http-get-request%` object for a specified URL, executes it, and returns the resulting request object. *url* is either an `http-uri<%>` or a string. *follow* is either a the maximum count of redirections to follow or `#t` (the default), meaning to use the value of the `current-max-http-redirects` parameter. *referrer* is a string with the URL to give as `Referer` (sic) header in requests, or `#f` (the default) meaning no referrer. Exceptions may be thrown. Example:

```
(http-get "http://www.schemers.org/")
⇒ #<struct:object:http-get-request%>
```

**http-get-content** *url* [*follow* [*referrer*]] [Procedure]

The `http-get-content` procedure is like `http-get`, except instead of yielding a `http-request%` object, it yields an input port. For example, quick scripts using `Httper` and `[HtmlPrag]` might do something like:

```
(html->sxml (http-get-content "http://www.plt-scheme.org/") #f)
⇒
(*top*
 (html (head (title "PLT Scheme")
             (meta (@ (name "generator") (content "PLT Scheme")))
             (style (@ (type "text/css")) ...)
             ...))
       (body (@ (bgcolor "white"))
             ...))
 "\n")
```

**call-with-http-get** *url* *proc* [*follow*] [Procedure]

Similar to the `http-get-content` procedure, except instead of returning the input port, it instead applies arity-1 procedure *proc* with the input port as its argument. `call-with-http-get` returns the value of applying *proc*.

## History

Version 0.3 — 22 August 2004

Replaced uses of `[UriFrame]` with `[Uri.scm]`. `current-force-http-uri-absolute?` removed. Packaging changes.

Version 0.2 — 30 June 2003

`initiate` method closes ports on exception. `uri` field added to `exn:http:proto` exception.

Version 0.1 — 1 May 2003

Initial release.

## References

- [HtmlPrag] <http://www.neilvandyke.org/htmlprag/>
- [LGPL] Free Software Foundation, “GNU Lesser General Public License,” Version 2.1, February 1999, 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA.  
<http://www.gnu.org/copyleft/lesser.html>
- [RFC1945] T. Berners-Lee, R. Fielding, H. Frystyk, “Hypertext Transfer Protocol – HTTP/1.0,” IETF RFC 1945, May 1996.  
<http://www.ietf.org/rfc/rfc1945.txt>
- [RFC2145] J. C. Mogul, R. Fielding, J. Gettys, H. Frystyk, “Use and Interpretation of HTTP Version Numbers,” IETF RFC 2145, May 1997.  
<http://www.ietf.org/rfc/rfc2145.txt>
- [RFC2616] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, “Hypertext Transfer Protocol – HTTP/1.1,” IETF RFC 2616, June 1999.  
<http://www.ietf.org/rfc/rfc2616.txt>
- [SRFI-19-PLT] <http://www.neilvandyke.org/srfi19-plt/>
- [Uri.scm] Neil W. Van Dyke, “uri.scm: Web Resource Identifiers (URI) in Scheme.”  
<http://www.neilvandyke.org/uri-scm/>
- [UriFrame] Neil W. Van Dyke, “UriFrame: Web Uniform Resource Identifier Framework.”  
<http://www.neilvandyke.org/uriframe/>