Guile-WWW Modules Reference

edition 2.36, released 22 November 2012

Thien-Thi Nguyen

This reference manual is for Guile-WWW 2.36.

Table of Contents

1 (www http)

The `(www http)` module includes module-configuration fluids, procedures for high-level HTTP operation, low-level HTTP message object access, and common messages.

1.1 Dynamic Configuration

`protocol-version` [Fluid]

A pair of integers representing the major and minor portions of the protocol version this module should support. The default value is `(1 . 0)`. Users:

```
http:request
http:head        ; via http:request
http:get         ; likewise
http:post-form   ; likewise
```

1.2 High-Level HTTP Operation

`http:connect` *proto addrfam address* [*address-rest...*] [Procedure]

Return a TCP stream socket connected to the location specified by protocol *proto*, *addrfam* and *address*. *proto* is `PF_INET` or `PF_UNIX`, and the other args take corresponding forms:

`PF_INET` `(AF_INET` *ipaddr portno*`)`, where *ipaddr* is an integer. Use `(car (hostent:addr-list (gethost` *host*`)))` to compute the ipaddr of *host* (a string).

`PF_UNIX` `(AF_UNIX` *filename*`)`, made, for example, by `(list AF_UNIX "/tmp/foo-control")`.

Note that `PF_foo` and `AF_foo` are names of variables that have constant values, not symbols.

`http:open` *host* [*port*] [Procedure]

Return an HTTP connection (a socket) to *host* (a string) on TCP port *port* (default 80 if unspecified).

`send-request` *sock method url* [`keyword value...`] [Procedure]

Keywords: `headers`, `body`, `flags`, `protocol-version`

Submit to socket *sock* an HTTP request using *method* (a symbol) and *url*, an object returned by `url:parse`, forming the message with additional *headers*, a list of strings, each of which should have one of the forms:

```
NAME ": " VALUE
NAME ": " VALUE CRLF
```

and *body*, which may be `#f` (which means no body), a string or list of them, a `u8` vector or list of them, or a procedure *m* which manages the transmission of the body data by supporting the *body transmission protocol*. This means *m* takes one arg, *command* (symbol):

`content-length`
: This is called if the transfer is not "chunked" (see below). *m* returns the total length (in bytes) of the data.

`next-chunk`
: Return two values: the length (in bytes) of the next chunk of data to send, and either a string, or a procedure *w* that does the actual writing to its port arg (which will be *sock*). If there is no more data, both values should be `#f`, i.e., *m* should return `(values #f #f)`.

If `flags` contains the symbol `chunked`, send the body with "chunked" `Transfer-Encoding`. Otherwise, compute and add to the headers its total `Content-Length`.

If `flags` contains the symbol `close`, add `Connection: close` to the headers.

The *protocol-version* is a pair specifying the major and minor version numbers of HTTP to send as. It defaults to `(1 . 1)`. For HTTP 1.x (x >= 1), automatically add `chunked` to *flags* as well as the headers:

```
TE: trailers
Connection: TE
```

Return an unspecified object *pending* that can be passed to `receive-response`.

`receive-response` *pending* [`keyword value...`] [Procedure]
: Keywords: `s2s`, `intervene`, `flags`

Receive the *pending* (from `send-request`) response. Return an HTTP message object. The header names are symbols made by `(string->symbol (s2s` *orig*`))`, where *s2s* defaults to `string-titlecase`. The status code is a 3-digit integer. The body of the message may be `#f` if there is no body possible (per HTTP). Otherwise, its value depends on *intervene* and *flags*.

- If *intervene* is specified, it should be a procedure that takes two args, *hget* and *flags* and returns two values, *new-headers* and *new-flags*. It is called after the headers are parsed but before the body is received so that its returned values may influence the body processing.

  hget is a procedure that takes one arg *sel*.

  `#f` Return the headers (alist).

  `#t` Return the name normalization procedure (involving *s2s*, described above).

  string Normalize it; return the associated header value.

  symbol Return the associated header value.

  A `#f` value for *new-headers* means "don't change the headers". Likewise, for *new-flags*. Otherwise, the respective items are replaced (**NB**: not just added!).
- If *flags* is null (the default), the body is a string.
- If *flags* contains the symbol `u8`, the body is a `u8` vector.
- If *flags* contains the symbol `custom`, the following item in *flags* should be a thunk that returns four values (all procedures) that support the *chunk transfer protocol*. These are:

`(mkx len)` Create and return a container capable of holding *len* bytes.

`(r! x sock)`
Fill *x*, reading from *sock*. Return the number of bytes read (positive integer), or zero on EOF.

`(cat-r list)`
Return a new container formed by reversing *list* and concatenating its elements.

`(subseq x len)`
Return a new container that holds the first *len* bytes of container *x*.

The message body is a single container, either constructed from multiple exact chunks ("chunked" `Transfer-Encoding`), or read in one swoop (if `Content-Length` is given), or from multiple inexact chunks (the default).

For backward compatability, instead of a thunk returning four values, you can also specify the four values directly. **NB: Support for specifying four values directly will NO LONGER BE AVAILABLE after 2013-05-15.**[1]

- If *flags* contains the symbol `no-cat`, then all multi-chunk transfers are not "concatenated"; instead, the message body is the list of chunk data (string, `u8` or `custom`), in order of reception.

Here is an example that uses `receive-response` argument *intervene* to arrange for the message body to be a `u8` vector if the `Content-Type` is not "text/*".

```
(use-modules
 (srfi srfi-13)   ; string-prefix?
 (www url))       ; url:parse

(define (text? type)
  (string-prefix? "text/" type))

(define (u8-maybe hget flags)
  (cond ((hget 'Content-Type)
         => (lambda (type)
              (values #f (and (not (text? type))
                              (cons 'u8 flags)))))
        (else
         (values #f #f))))

(define SOCK (http:open ...))

(define (gimme string)
```

[1] Out of an explicit `call-with-values` context, Guile 2.x will silently discard all values following the "expected number" (one, in this case):

```
(list 'custom (values P1 P2 P3 P4))
⇒ (custom P1)
```

This is apparently allowed under R5RS and R6RS.

```
  (send-request SOCK 'GET (url:parse string)))

(define (ok pending)
  (receive-response pending #:intervene u8-maybe))

(define ICO (ok (gimme "http://localhost/favicon.ico")))
(define IDX (ok (gimme "http://localhost/index.html")))

(http:message-body ICO)
⇒ #u8(0 0 1 0 1 0 46 ...)

(http:message-body IDX)
⇒ "<?xml version=\"1.0\" ..."
```

Note that to find the content type in `u8-maybe`, we rely on the default header-name normalization of `string-titlecase`, since we know `ok` does not specify `#:s2s` *`s2s`* in its call to `receive-response`. To enable `u8-maybe` to work with any pending response, you can instead use `(hget "Content-Type")` (i.e., a string name).

`http:request` *method url* [*headers* [*body*]] [Procedure]

Submit an HTTP request using *method* and *url*, wait for a response, and return the response as an HTTP message object. The field types and values of this message object are as described in `receive-response`, with two exceptions (for backward compatability): the status code is a string; the header names are symbols, all lowercase.

method is the symbolic name of some HTTP method, e.g., `GET` or `POST`. It may also be a string. *url* is a url object returned by `url:parse`. Optional args *headers* and *body* are lists of strings that comprise the lines of an HTTP message. The header strings should not end with '`CR`' or '`LF`' or '`CRLF`'; `http:request` handles that. Also, the Content-Length header and Host header are calculated automatically and should not be supplied. Here are two examples:

```
(http:request 'GET parsed-url
  (list "User-Agent: Anonymous/0.1"
        "Content-Type: text/plain"))

(http:request 'POST parsed-url
  (list "User-Agent: Fred/0.1"
        "Content-Type: application/x-www-form-urlencoded")
  (list "search=Gosper"
        "&case=no"
        "&max_hits=50"))
```

In the second example, the `Content-Length` header is computed to have value 33 (the sum of 13, 8 and 12).

1.3 Low-Level HTTP Message Object Access

`http:message-version` *msg* [Procedure]

Return the HTTP version in use in HTTP message *msg*.

`http:message-status-code` *msg* [Procedure]
Return the status code returned in HTTP message *msg*.

`http:message-status-text` *msg* [Procedure]
Return the text of the status line from HTTP message *msg*.

`http:message-status-ok?` *msg* [Procedure]
Return `#t` iff status code of *msg* indicates a successful request.

`http:status-ok?` *status* [Procedure]
Return `#t` iff *status* (string or integer) is `2xx`.

`http:message-body` *msg* [Procedure]
Return the body of the HTTP message *msg*.

An HTTP message header is represented by a pair. The CAR is a symbol representing the header name, and the CDR is a string containing the header text. E.g.:

```
'((date . "Thu, 29 May 1997 23:48:27 GMT")
  (server . "NCSA/1.5.1")
  (last-modified . "Tue, 06 May 1997 18:32:03 GMT")
  (content-type . "text/html")
  (content-length . "8097"))
```

Note: these symbols are all lowercase, although the original headers may be mixed-case. Clients using this library should keep this in mind, since Guile symbols are case-sensitive.

`http:message-headers` *msg* [Procedure]
Return a list of the headers from HTTP message *msg*.

`http:message-header` *header msg* [Procedure]
Return the header field named *header* from HTTP message *msg*, or `#f` if no such header is present in the message.

1.4 Common Messages

`http:post-form` *url extra-headers fields* [Procedure]
Submit an http request using the `POST` method on the *url*. *extra-headers* is a list of extra headers, each a string of form "*name*: *value* ...".

The "Content-Type" and "Host" headers are sent automatically and do not need to be specified. *fields* is a list of elements of the form `(fkey . fvalue)`, where *fkey* is a symbol and *fvalue* is normally a string.

fvalue can also be a list of file-upload specifications, each of which has the form `(source name mime-type transfer-encoding)`. *source* can be a string or a thunk that returns a string.

The rest of the elements are strings or symbols: *name* is the filename (only the non-directory part is used); *mime-type* is a type/subtype pair such as "image/jpeg", or `#f` to mean "text/plain". *transfer-encoding* is one of the tokens specified by RFC 1521, or `#f` to mean "binary". File-upload spec elements with invalid types result in a "bad upload spec" error prior to the http request.

Note that *source* is used directly without further processing; it is the caller's responsibility to ensure that the MIME type and transfer encoding specified describe *source* accurately.

If there are no file-upload specifications in *fields*, the `Content-Type` is `application/x-www-form-urlencoded`, and furthermore all the *fkey* and *fvalue* are transformed by `url-coding:encode` (see Chapter 5 [url-coding], page 13) with the additional reserved characters `#\&` (ampersand) and `#\=` (equal sign).

Otherwise, the `Content-Type` is `multipart/form-data`, with each field in *fields* formatted as a MIME sub-part.

NB: The following two procedures will NO LONGER BE AVAILABLE after 2013-02-28. Using `send-request` and `receive-reply` directly is more flexible and featureful.

`http:head` *url* [Procedure]

Submit an http request using the `HEAD` method on the *url*. The `Host` header is automatically included.

`http:get` *url* [Procedure]

Submit an http request using the `GET` method on the *url*. The `Host` header is automatically included.

2 (www url)

The `(www url)` module provides procedures for high-level url object conversion, low-level url object construction and access, and character decoding/encoding.

2.1 High-Level URL Object Conversion

`url:parse` *string* [Procedure]
Parse *string* and return a url object, with one of the following "schemes": HTTP, FTP, mailto, unknown.

`url:unparse` *url* [Procedure]
Return the *url* object formatted as a string. Note: The username portion is not included!

2.2 Low-Level URL Object Construction

`url:make` *scheme* [*args...*] [Procedure]
Construct a url object with specific *scheme* and other *args*. The number and meaning of *args* depends on the *scheme*.

`url:make-http` *host port path* [Procedure]
Construct a HTTP-specific url object with *host*, *port* and *path* portions.

`url:make-ftp` *user host port path* [Procedure]
Construct a FTP-specific url object with *user*, *host*, *port* and *path* portions.

`url:make-mailto` *address* [Procedure]
Construct a mailto-specific url object with an *address* portion.

2.3 Low-Level URL Object Access

`url:scheme` *url* [Procedure]
Extract and return the "scheme" portion of a *url* object. `url:scheme` is an unfortunate term, but it is the technical name for that portion of the URL according to RFC 1738. Sigh.

`url:user` *url* [Procedure]
Extract and return the "user" portion of the *url* object.

`url:host` *url* [Procedure]
Extract and return the "host" portion of the *url* object.

`url:port` *url* [Procedure]
Extract and return the "port" portion of the *url* object.

`url:path` *url* [Procedure]
Extract and return the "path" portion of the *url* object.

NB: The following two procedures will NO LONGER BE AVAILABLE after 2012-12-31. They are misguided attempts at abstraction better left behind.

`url:address` *url* [Procedure]
Extract and return the "address" portion of the *url* object.

`url:unknown` *url* [Procedure]
Extract and return the "unknown" portion of the *url* object.

2.4 Character Decoding/Encoding

`url:decode` *str* [Procedure]
Re-export `url-coding:decode`. See Chapter 5 [url-coding], page 13.

`url:encode` *str reserved-chars* [Procedure]
Re-export `url-coding:encode`. See Chapter 5 [url-coding], page 13.

3 (www cgi)

The `(www cgi)` module provides procedures to support painlessly writing Common Gateway Interface scripts to process interactive forms. These scripts typically follow the following steps: initialization and discovery, data transfer in, data transfer out.

3.1 Initialization and Discovery

`cgi:init` [*opts...*] [Procedure]

(Re-)initialize internal data structures. This must be called before calling any other '`cgi:foo`' procedure. For FastCGI, call this "inside the loop" (that is, for each CGI invocation).

opts are zero or more symbols that configure the module.

`uploads-lazy`
: This controls how uploaded files, as per `cgi:uploads` and `cgi:upload`, are represented.

`cookies-split-on-semicolon`
: This causes cookies parsing to use `#\;` (semicolon), instead of the default `#\,` (comma), for splitting multiple cookies. This is necessary, for example, if the server is configured to provide "Netscape style" (i.e., old and deprecated) cookies.

Unrecognized options are ignored.

`cgi:form-data?` [Procedure]

Return `#t` iff there is form data available.

`cgi:names` [Procedure]

Return a list of variable names in the form. The order of the list is the same as that found in the form for the first occurance of each variable and each variable appears at most once. For example, if the form has variables ordered `a b a c d b e`, then the returned list would have order `a b c d e`.

`cgi:cookie-names` [Procedure]

Return a list of cookie names.

3.2 Data Transfer In

`cgi:getenv` *key* [Procedure]

Return the value of the environment variable associated with *key*, a symbol. Unless otherwise specified below, the return value is a (possibly massaged, possibly empty) string. The following keys are recognized:

```
server-software
server-software-type     ; part of server-software before "/"
server-software-version  ; part of server-software after "/"
server-name
server-hostname          ; alias for server-name
```

```
gateway-interface
server-protocol
server-protocol-name     ; part of server-protocol before "/"
server-protocol-version  ; part of server-protocol after "/"
server-port (integer)
request-method
path-info
path-translated
script-name
query-string
remote-host
remote-addr
auth-type
authentication-type      ; alias for auth-type
remote-user
remote-ident
content-type
content-length (integer, possibly 0)
http-accept (list, possibly empty, of strings)
http-accept-types        ; alias for http-accept-types
http-user-agent
http-cookie
```

Keys not listed above result in an "unrecognized key" error.

`cgi:values` *name* [Procedure]

Fetch any values associated with *name* found in the form data. Return a list, even if it contains only one element. A value is either a string, or `#f`. When there are multiple values, the order is the same as that found in the form.

`cgi:value` *name* [Procedure]

Fetch only the CAR from `(cgi:values name)`. Convenient for when you are certain that *name* is associated with only one value.

`cgi:uploads` *name* [Procedure]

Return a list of file contents associated with *name*, or `#f` if no files are available.

Uploaded files are parsed by `parse-form` (see Chapter 9 [form-2-form], page 20). If the `uploads-lazy` option is specified to `cgi:init`, then the file contents are those directly returned by `form-2-form`. If unspecified, the file contents are strings with the object property `#:guile-www-cgi` whose value is an alist with the following keys:

`#:name` identical to *name* (sanity check)

`#:filename`
: original/suggested filename for this bunch of bits

`#:mime-type`
: something like "image/jpeg"

`#:raw-mime-headers`
: the MIME headers before parsing

Note that the string's object property and the keys are all keywords. The associated values are strings.

Unless `uploads-lazy` is specified (to `cgi:init`), `cgi:uploads` can only be called once per particular *name*. Subsequent calls return `#f`. Caller had better hang onto the information, lest the garbage man whisk it away for good. This is done to minimize the amount of time the file is resident in memory.

`cgi:upload` *name* [Procedure]
Fetch the first file associated with form var *name*. Can only be called once per *name*, so the caller had better be sure that there is only one file associated with *name*. Use `cgi:uploads` if you are unsure.

`cgi:cookies` *name* [Procedure]
Fetch any cookie values associated with *name*. Return a list of values in the order they were found in the HTTP header, which should be the order of most specific to least specific path associated with the cookie. If no cookies are associated with *name*, return `#f`.

`cgi:cookie` *name* [Procedure]
Fetch the first cookie value associated with *name*.

3.3 Uncollated Form Data

With `cgi:values`, when a name occurs more than once, its associated values are collated, thus losing information about the relative order of different and intermingled names. For this, you can use `cgi:nv-pairs` to access the uncollated (albeit ordered) form data.

`cgi:nv-pairs` [Procedure]
Fetch the list of `(name . value)`, in the same order as found in the form data. A name may appear more than once. A value is either a string, or `#f`.

4 (www main)

NB: This module will NO LONGER BE AVAILABLE after 2012-12-31. Using `send-request` and `receive-reply` directly is more flexible and featureful (see Chapter 1 [http], page 1).

The `(www main)` module provides a generic interface useful for retrieving data named by any URL. The URL scheme `http` is pre-registered.

`www:set-protocol-handler!` *proto handler* [Procedure]
: Associate for scheme *proto* the procedure *handler*. *proto* is a symbol, while *handler* is a procedure that takes three strings: the host, port and path portions, respectively of a url object. Its return value is the return value of `www:get` (for *proto*), and need not be a string.

`www:get` *url-string* [Procedure]
: Parse *url-string* into portions. For HTTP, open a connection, retrieve and return the specified document. Otherwise, consult the handler procedure registered for the particular scheme and apply it to the host, port and path portions of *url-string*. If no such handler exists, signal `"unknown URL scheme"` error.

There is also the convenience proc `www:http-head-get`.

`www:http-head-get` *url-string* [*alist?*] [Procedure]
: Parse *url-string* into portions; issue an `"HTTP HEAD"` request. Signal error if the scheme for *url-string* is not `http`. Optional second arg `alist?` non-`#f` means return only the alist portion of the HTTP response object.

5 (www url-coding)

The `(www url-coding)` module provides two procedures for decoding and encoding URL strings for safe transmission according to RFC 1738.

`url-coding:decode` *str* [Procedure]
: Return a new string made from url-decoding *str*. Specifically, turn `+` into space, and hex-encoded `%XX` strings into their eight-bit characters.

`url-coding:encode` *str reserved-chars* [Procedure]
: Return a new string made from url-encoding *str*, unconditionally transforming those in *reserved-chars*, a list of characters, in addition to those in the standard (internal) set.

6 (www utcsec)

The `(www utcsec)` module provides procedures to work with the *utc-seconds* of an object, that is, the number of seconds after epoch, in the GMT time zone (also known as UTC).

`format-utcsec` *port format utc-seconds* [Procedure]
Write to output port *port* the *utc-seconds* formatted according to *format* (a string). If *port* is `#f`, return the output string, instead. This uses `strftime`, q.v.

`rfc1123-date<-` *port utc-seconds* [Procedure]
Write to output port *port* the *utc-seconds* formatted according to RFC1123. If *port* is `#f`, return the output string, instead.

For example:

```
(rfc1123-date<- #f 1167791441)
⇒ "Wed, 03 Jan 2007 02:30:41 GMT"
```

`<-rfc1123-date` *s* [Procedure]
Parse the RFC1123-compliant date string *s*, and return the utc-seconds it represents.

For example:

```
(<-rfc1123-date "Wed, 03 Jan 2007 02:30:41 GMT")
⇒ 1167791441
```

`<-mtime` *filespec* [Procedure]
Return the utc-seconds of the modification time of *filespec*. *filespec* can be a filename (string), a port opened on a `stat`able file, or the object resulting from a `stat` on one of these.

For example:

```
(= (<-mtime "COPYING")
   (<-mtime (open-input-file "COPYING"))
   (<-mtime (stat "COPYING")))
⇒ #t
```

`<-ctime` *filespec* [Procedure]
Return the utc-seconds of the creation time of *filespec*. *filespec* can be a filename (string), a port opened on a `stat`able file, or the object resulting from a `stat` on one of these.

`rfc1123-now` [Procedure]
The "current time" formatted according to RFC1123.

7 (www server-utils big-dishing-loop)

The `(www server-utils big-dishing-loop)` module provides procedures that facilitate generation of a customized listener/dispatch proc.

`named-socket` *family name* [`keyword value...`] [Procedure]

Keywords: `socket-setup`

Return a new socket in protocol *family* with address *name*.

First, evaluate `(socket` *`family`* `SOCK_STREAM 0)` to create a new socket *sock*. Next, handle `#:socket-setup`, with value *setup*, like so:

`#f` Do nothing. This is the default.

procedure Call *procedure* on *sock*.

`((`*`opt`* `.` *`val`*`) ...)`
For each pair in this alist, call `setsockopt` on *sock* with the pair's *opt* and *val*.

Lastly, `bind` *sock* to *name*, which should be in a form that is appopriate for *family*. Two common cases are:

`PF_INET` `(AF_INET` *`ipaddr portno`*`)`, made, for example, by `(list AF_INET INADDR_ANY 4242)`.

`PF_UNIX` `(AF_UNIX` *`filename`*`)`, made, for example, by `(list AF_UNIX "/tmp/foo-control")`.

Note that `PF_foo`, `AF_foo`, and `INADDR_foo` are names of variables that have constant values, not symbols.

`echo-upath` *M upath* [*extra-args...*] [Procedure]
Use mouthpiece *M* (see Chapter 13 [answer], page 31) to compose and send a `"text/plain"` response which has the given *upath* (a string) and any *extra-args* as its content. Shut down the socket for both transmission and reception, then return `#t`.

This proc can be used to ensure basic network connectivity (i.e., aliveness testing).

`make-big-dishing-loop` [`keyword value...`] [Procedure]
Return a proc *dish* that loops serving http requests from a socket. *dish* takes one arg *ear*, which may be a pre-configured socket, a TCP port number, or a list of the form: `(`*`family address ...`*`)`. When *ear* is a TCP port number, it is taken to be the list `(PF_INET AF_INET INADDR_ANY` *`ear`*`)`.

In the latter two cases, the socket is realized by calling `named-socket` with parameters *family* and *name* taken from the CAR and CDR, respectively, of the list, with the `#:socket-setup` paramater (see below) passed along unchanged.

dish behavior is controlled by the keyword arguments given to `make-big-dishing-loop`. The following table is presented roughly in order of the steps involved in processing a request, with default values shown next to the keyword.

`#:socket-setup #f`
: This may be a proc that takes a socket, or a list of opt/val pairs which are passed to `setsockopt`. Socket setup is done for newly created sockets (when *dish* is passed a TCP port number), prior to the `bind` call.

`#:queue-length 0`
: The number of clients to queue, as set by the `listen` system call. Setting the queue length is done for both new and pre-configured sockets.

`#:concurrency #:new-process`
: The type of concurrency (or none if the value is not recognized). Here are the recognized values:

  `#:new-process`
  `#:new-process/nowait`
  : Fork a new process for each request. The latter does not wait for the child process to terminate before continuing the listen loop.

  `#f`
  : Handle everything in the current in process (no concurrency). Unrecognized values are treated the same as `#f`.

`#:bad-request-handler #f`
: If the first line of an HTTP message is not in the proper form, this specifies a proc that takes a mouthpiece *m*. Its return value should be the opposite boolean value of the `#:loop-break-bool` value, below. See Chapter 13 [answer], page 31.

`#:method-handlers ()`
: This alist describes how to handle the (valid) HTTP methods. Each element has the form `(`*`method . handler`*`)`. *method* is a symbol, such as `GET`; and *handler* is a procedure that handles the request for *method*.

  handler normally takes two arguments, the mouthpiece *m* and the *upath* (string), composes and sends a response, and returns non-`#f` to indicate that the big dishing loop should continue.

  The proc's argument list is configured by `#:need-headers`, `#:need-input-port` and `#:explicit-return`. Interpretation of the proc's return value is configured by `#:explicit-return` and `#:loop-break-bool`. See below.

`#:need-headers #f`
`#:need-input-port #f`
: If non-`#f`, these cause additional arguments to be supplied to the handler proc. If present, the headers arg precedes the input port arg. See Chapter 8 [parse-request], page 18. The input port is always positioned at the beginning of the HTTP message body.

  If `#:need-input-port` is `#f`, after the handler proc returns, the port is `shutdown` in both (r/w) directions. When operating concurrently, this is done on the child side of the split. See Section "Network Sockets and Communication" in *The Guile Reference Manual*.

`#:explicit-return #f`
: If non-`#f`, this arranges for a continuation to be passed (as the last argument) to the handler proc, and ignores that proc's normal return value in favor of one explicitly passed through the continuation. If the continuation is not used, the *effective return value* is computed as `(not #:loop-break-bool)`.

`#:loop-break-bool #f`
: Looping stops if the effective return value of the handler is `eq?` to this value.

`#:unknown-http-method-handler #f`
: If `#f`, silently ignore unknown HTTP methods, i.e., those not specified in `#:method-handlers`. The value may also be a procedure that takes three arguments: a mouthpiece *m*, the *method* (symbol) and the *upath* (string). Its return value should be the opposite boolean value of the `#:loop-break-bool` value, below. See Chapter 13 [answer], page 31.

`#:parent-finish close-port`
: When operating concurrently (`#:concurrency` non-`#f`), the "parent" applies this proc to the port after the split.

`#:log #f`
: This proc is called after the handler proc returns. Note that if *ear* is a unix-domain socket, the *client* parameter will be simply `"localhost"`. See Chapter 14 [log], page 36.

`#:status-box-size #f`
: This may be a non-negative integer, typically 0, 1 or 2. It is used by `#:log` (has no meaning if `#:log` is `#f`). See Chapter 14 [log], page 36.

`#:style #f`
: An object specifying the syntax of the first-line and headers. The default specifies a normal HTTP message (see Chapter 1 [http], page 1).

The combination of `#:need-headers`, `#:need-input-port` and `#:explicit-return` mean that the `#:GET-upath` proc can receive anywhere from two to five arguments. Here is a table of all the possible combinations (1 means non-`#f` and 0 means `#f`):

```
+----- #:explicit-return
| +--- #:need-input-port
| | +- #:need-headers
| | |
| | |  args to #:GET-upath proc
=====  ============================
0 0 0  M upath
0 0 1  M upath headers
0 1 0  M upath in-port
0 1 1  M upath headers in-port
1 0 0  M upath return
1 0 1  M upath headers return
1 1 0  M upath in-port return
1 1 1  M upath headers in-port return
```

8 (www server-utils parse-request)

The `(www server-utils parse-request)` module provides procedures to read the first line, the headers and the body, of an HTTP message on the input port.

`receive-request` *port* [`keyword value...`] [Procedure]

Keywords: `s2s`, `style`

Return a request object read from *port*. Use *s2s* (defaults to `string-titlecase`) to normalize the header names. With `#:s2s string-downcase`, for instance, you would see `(host . "example.com")` in the `headers` field of the request object.

Keyword arg *style* is an object specifying the syntax of the initial (non-body) portion. By default, *parse* expects a normal HTTP 1.1 request message as per RFC 2616.

`request` [Type]

A request object has five fields.

`method` A symbol, such as `GET`.

`upath` A string. You can use `hqf<-upath` and `alist<-query` to break this down further.

`protocol-version`
A pair of integers indicating the protocol version. For example, `(1 . 1)` corresponds to HTTP 1.1.

`headers` A list of pairs `(`*`name`*` . `*`value`*`)`, aka alist, where *name* is a symbol and *value* is a string. How *name* is normalized depends on which *s2s* was specified to `receive-request`.

`body` Either `#f` or a procedure *get-body*. This should be called with one arg, *flags*, to retrieve the request body. See Chapter 1 [http], page 1, procedure `receive-response`, for *flags* documentation.

`request?` *obj* [Procedure]

Return `#t` if *obj* is a request object.

`request-method` *req* [Procedure]
`request-upath` *req* [Procedure]
`request-protocol-version` *req* [Procedure]
`request-headers` *req* [Procedure]
`request-body` *req* [Procedure]

Return the respective field of request object *req*.

`hqf<-upath` *upath* [Procedure]

Parse string *upath* and return three values representing its hierarchy, query and fragment components. If a component is missing, its value is `#f`.

```
(hqf<-upath "/aa/bb/cc?def=xyz&hmm#frag")
⇒ "/aa/bb/cc"
⇒ "def=xyz&hmm"
⇒ "frag"
```

```
(hqf<-upath "/aa/bb/cc#fr?ag")
⇒ "/aa/bb/cc"
⇒ #f
⇒ "fr?ag"
```

`alist<-query` *query-string* [Procedure]
Parse urlencoded *query-string* and return an alist. For each element `(`*name* `.` *value*`)` of the alist, *name* is a string and *value* is either `#f` or a string.

NB: The following four procedures will NO LONGER BE AVAILABLE after 2013-02-28. Better to use `receive-request`.

`read-first-line` *port* [Procedure]
Parse the first line of the HTTP message from input *port* and return a list of the method, URL path and HTTP version indicator, or `#f` if the line ends prematurely or is otherwise malformed. A successful parse consumes the trailing '`CRLF`' of the line as well. The method is a symbol with its constituent characters upcased, such as `GET`; the other elements are strings. If the first line is missing the HTTP version, `parse-first-line` returns the default `"HTTP/1.0"`.

`read-headers` *port* [Procedure]
Parse the headers of the HTTP message from input *port* and return a list of key/value pairs, or `#f` if the message ends prematurely or is otherwise malformed. Both keys and values are strings. Values are trimmed of leading and trailing whitespace and may be empty. Values that span more than one line have their `"continuation whitespace"` reduced to a single space. A successful parse consumes the trailing '`CRLF`' of the header block as well.

Sometimes you are interested in the body of the message but not the headers. In this case, you can use `skip-headers` to quickly position the port.

`skip-headers` *port* [Procedure]
Scan without parsing the headers of the HTTP message from input *port*, and return the empty list, or `#f` if the message ends prematurely. A successful scan consumes the trailing '`CRLF`' of the header block as well.

`read-body` *len port* [Procedure]
Return a new string of *len* bytes with contents read from input *port*.

9 (www server-utils form-2-form)

The `(www server-utils form-2-form)` module provides a procedure to parse a string in '`multipart/form-data`' format.

`parse-form` *content-type-more raw-data* [Procedure]

Parse *raw-data* as raw form response data of enctype '`multipart/form-data`' and return an alist.

content-type-more is a string that should include the `boundary="..."` information. (This parameter name reflects the typical source of such a string, the Content-Type header value, after the '`multipart/form-data`'.)

Each element of the alist has the form `(name . value)`, where *name* is a string and *value* is either a string or four values (extractable by `call-with-values`):

filename A string, or `#f`.

type A string representing the MIME type of the uploaded file.

raw-headers
A string, including all eol CRLF chars. Incidentally, the *type* should be (redundantly) visible in one of the headers.

squeeze A procedure that takes one arg *abr* (standing for access byte range). If *abr* is `#f`, then internal references to the uploaded file's data are dropped. Otherwise, *abr* should be a procedure that takes three arguments: a string, a beginning index (integer, inclusive), and an ending index (integer, exclusive).

If there is no type information, *value* is a simple non-empty string, and no associated information (filename, raw-headers, squeeze) is kept.

`parse-form` ignores *degenerate uploads*, that is those parts of `raw-data` where the part header specifies no filename and the part content-length is zero or unspecified.

why squeeze?

The *squeeze* interface can help reduce data motion. Consider a common upload scenario: client uploads file(s) for local (server-side) storage.

```
classic  squeeze
   *        *       0. (current-input-port)
   *        *       1. Guile-WWW string (for parsing purposes)
   *                2. your substring (image/jpeg)
   *        *       3. filesystem
```

You can achieve the same effect as the "classic" approach by specifying `substring` (or something like it) as the access-byte-range proc, but **you don't have to**. You could, instead, call *squeeze* with a procedure that writes the byte range directly to the filesystem.

10 (www server-utils filesystem)

The `(www server-utils filesystem)` module provides procedures for cleaning filenames, checking filesystem access, and mapping from a URL path to a filename.

`cleanup-filename` *name* [Procedure]

Return a new filename made from cleaning up filename *name*. Cleaning up is a transform that collapses each of these, in order:

- '`//`'
- '`/./`'
- '`/foo/../`'

into a single slash ('`/`'), everywhere in *name*, plus some fixups. The transform normally preserves the trailing slash (if any) in *name*, and does not change any leading '`..`' components if *name* is relative, i.e., does not begin with slash. Due to proper '`/foo/../`' cancellation for relative *name*, however, the result may be the empty string. (Here, *proper* means that *foo* is not '`..`', but a normal filename component.)

Following is a fairly comprehensive list of the `cleanup-filename` edge cases, paired by *name* and result. The numbers represent string lengths.

```
0                       ;; empty string
0                       ;; result is empty string

1  /
1  /

2  ok
2  ok

3  ok/
3  ok/

3  /ok
3  /ok

4  /ok/
4  /ok/

1  .                    ;; relative name
0                       ;; result is empty string

2  ./                   ;; likewise
0                       ;; note, end-slash not preserved

2  /.
1  /
```

```
3  /./
1  /

2  ..                   ;; relative, with leading double-dot
2  ..                   ;; unchanged

3  ../                  ;; likewise
3  ../

3  /..                  ;; absolute
1  /                    ;; can't go higher than root

4  /../
1  /

4  ./..                 ;; next 8 are like the previous 4;
2  ..                   ;;  they show that  .  makes no difference

5  ./../
3  ../

5  /./..
1  /

6  /./../
1  /

4  ../.
2  ..

5  .././
3  ../

5  /../.
1  /

6  /.././
1  /

5  ../..               ;; relative
5  ../..               ;; leading  ..  sequences unchanged

6  ../../
6  ../../

6  /../..              ;; absolute
1  /                    ;; can't go higher than root
```

```
 7  /../../
 1  /

 4  z/..                ;; relative
 0                      ;; only dir cancelled ⇒ empty string

 5  z/../               ;; likewise
 0

 5  /z/..               ;; absolute
 1  /

 6  /z/../
 1  /

 6  z/../o             ;; next 4 like previous 4, with trailing component
 1  o

 7  z/../o/
 2  o/

 7  /z/../o
 2  /o

 8  /z/../o/
 3  /o/

 8  z/./../o           ;; next 4 like previous 4;
 1  o                   ;;  they show that  .  makes no difference

 9  z/./../o/
 2  o/

 9  /z/./../o
 2  /o

10  /z/./../o/
 3  /o/

 9  z/../../o          ;; relative, more double-dot than parents
 4  ../o                ;; leftover double-dot preserved

10  z/../../o/
 5  ../o/

10  /z/../../o         ;; absolute, more double-dot than parents
```

```
 2  /o                    ;; all cancelled

11  /z/../../o/
 3  /o/

43  ../../abc/./bye0/./../def/bye1/bye2/../..        ;; bye bye-bye
14  ../../abc/def/

44  ../../abc/./bye0/./../def/bye1/bye2/../../
14  ../../abc/def/

44  /../../abc/./bye0/./../def/bye1/bye2/../..
 9  /abc/def/

45  /../../abc/./bye0/./../def/bye1/bye2/../../
 9  /abc/def/
```

`access-forbidden?-proc` *docroot forbid-rx* [Procedure]

Create and return a filesystem-access procedure based on *docroot* and *forbid-rx*. The returned procedure *p* takes a *filename* and returns `#t` if access to that file should be denied for any of the following reasons:

- *filename* does not begin with *docroot*
- *filename* matches regular expression *forbid-rx*

If *forbid-rx* is `#f`, the regular expression check is skipped. *p* returns `#f` if access should be granted.

`upath->filename-proc` *docroot* [*dir-indexes*] [Procedure]

Create and return a url-path-to-filename mapping procedure based on *docroot*. The returned procedure *p* takes a (string) *upath* and returns a valid local filename path for the requested resource, or `#f` if that file cannot be found. Optional arg *dir-indexes* specifies an ordered list of filenames to try if the resolved filename path turns out to be a directory.

If no such files exist, return the directory name. As a special case, when *p* encounters a value of `#f` during iteration over *dir-indexes*, it returns `#f` immediately.

For example, presuming files `/a/b/c.txt` and `/a/b/index.html` both exist and are readable:

```
(define resolve (upath->filename-proc
                  "/a/b/"
                  '("index.shtml" "index.html")))

(resolve "/random") ⇒ #f
(resolve "/c.txt") ⇒ "/a/b/c.txt"
(resolve "/") ⇒ "/a/b/index.html"
```

Directory names are always returned with a trailing slash.

`filename->content-type` *filename* [*default*] [Procedure]

Return a valid Content-Type string which matches *filename* best. Matching is done by comparing the extension (part of *filename* after the last `"."` if available) against a table. If none match, return `"application/octet-stream"`. Optional arg *default* specifies another value to use instead of `"application/octet-stream"`.

If there are multiple MIME types associated with the extension, return the first one.

See Chapter 17 [mime-types], page 39, proc `put-mime-types!`, for more info.

11 (www server-utils cgi-prep)

Often the server cannot do everything by itself, and makes use of external programs invoked in a *common gateway interface* environment. These programs are also known as *CGI scripts*.

The `(www server-utils cgi-prep)` module provide a procedure to set up such an environment. Actually invoking the CGI script is not covered.

`cgi-environment-manager` *initial-bindings* [Procedure]

Return a closure encapsulating *initial-bindings*, a list of pairs `(name . value)`, where *name* is a symbol listed in the following table, and *value* is a string unless otherwise noted.

- `server-hostname`
- `gateway-interface`
- `server-port` (integer)
- `request-method`
- `path-info`
- `path-translated`
- `script-name`
- `query-string`
- `remote-host`
- `remote-addr`
- `authentication-type`
- `remote-user`
- `remote-ident`
- `content-type`
- `content-length` (integer, or `#f`)
- `http-user-agent`
- `http-cookie`
- `server-software`
- `server-protocol`
- `http-accept-types` (list of strings)

If *name* is not recognized, signal `"unrecognized key"` error. Encapsulation includes *`name=value`* formatting.

The closure accepts these commands:

`name value`
: Encapsulate an additional binding. *name* and *value* are as above.

`#:clear!`
: Drop the additional bindings. Note that initial bindings can never be dropped (you can always create a new closure).

`#:environ-list`
: Return a list of strings suitable for passing to `environ` or as the second argument to `execle`.

Any other command results in a `"bad command"` error.

example

Following is a simple example of how to use `cgi-environment-manager`. A more realistic example would include port and connection management, input validation, error handling, logging, etc. First, we set up the manager with more-or-less constant bindings.

```
(define M (cgi-environment-manager
            '((server-software . "FooServe/24")
              (server-protocol . "HTTP/1.0")
              (server-port . 80))))
```

Later, we add connection-specific bindings. We use `read-first-line` from the Chapter 8 [parse-request], page 18 module.

```
(define PORT ...)
(define UPATH (list-ref (read-first-line PORT) 1))
(define QMARK (string-index UPATH #\?))
(define CGI (substring UPATH 0 QMARK))

(M 'script-name CGI)
(M 'query-string (substring UPATH (1+ QMARK)))
```

Lastly, we spawn the child process, passing the constructed environment as the second arg to `execle`, and drop the connection-specific bindings afterwards.

```
(let ((pid (primitive-fork)))
  (if (zero? pid)
      (execle CGI (M #:environ-list) (list CGI)) ; child
      (waitpid pid)))                            ; parent

(M #:clear!)
```

Now we can re-use `M` for another connection.

12 (www server-utils cookies)

Cookies are bits of client-side state the server can maintain through designated HTTP response headers. At this time (2009), there are two specifications, RFC2109[1] and RFC2965[2], the latter obsoleting the former.

This chapter describes the `(www server-utils cookies)` module, which provides facilities for creating such headers, and parsing those sent by the client. Procedures that return trees are meant to be used with the `mouthpiece` command `#:add-header` (see Chapter 13 [answer], page 31).

`simple-parse-cookies` *string* [*sep*] [Procedure]

Parse *string* for "cookie-like fragments", that is, zero or more substrings of the form *`name=value`*, separated by character *sep* and optionally trailing whitespace. By default, *sep* is `#\,` (comma).

Return a list of elements `(`*`name . value`*`)`, where both *name* and *value* are strings. For example:

```
(define COOKIE "abc=def; z=z, ans=\"42\", abc=xyz")

(simple-parse-cookies COOKIE)
⇒ (("abc" . "def; z=z") ("ans" . "\"42\"") ("abc" . "xyz"))

(simple-parse-cookies COOKIE #\;)
⇒ (("abc" . "def") ("z" . "z, ans=\"42\", abc=xyz"))
```

`rfc2109-set-cookie-string` *name value* [`keyword value`...] [Procedure]

Keywords: `path`, `domain`, `expires`, `secure`

Return a string suitable for inclusion into an HTTP response header as a cookie with *name* and *value*. Both args may be strings, symbols or keywords. Also, recognize and format appropriately the optional keyword parameters `#:path`, `#:domain`, `#:expires` (strings); and `#:secure` (boolean).

`rfc2965-set-cookie2-tree` *M* [*cookie-specs*...] [Procedure]

Compute a list suitable for inclusion in an HTTP response header, composed by formatting *cookie-specs*, each a list of the form `(`*`name value a1 v1...`*`)`. Each *name* may be a string, symbol or keyword. Each *value* may be a string or symbol. Each *a* must be a keyword, precisely one of:

```
#:Comment  #:CommentURL  #:Discard  #:Domain
#:Max-Age  #:Path  #:Port  #:Secure
```

The #:Version attribute is automatically included as the last one; it cannot be specified (or de-specified).

Possible values for *v* depend on *a*. If *a* is `#:Discard` or `#:Secure`, then there is no *v* (it must be omitted). If *a* is `#:Port`, then *v* must be either a number; a list of numbers, for instance `(8001 8002 8003)`; or omitted entirely. If *a* is `#:Max-Age`, then *v* must be a number. For all other *a*, *v* can be a string or symbol.

[1] RFC2109

[2] RFC2965

If *M* is `#f`, return a list. The CAR of the list is the keyword `#:Set-Cookie2`, and the CDR is a tree of strings. Otherwise *M* should be a `mouthpiece` (see Chapter 13 [answer], page 31) in which case it is applied with the `#:add-header` command to the list.

example

Here is an example that demonstates both RFC2109 and RFC2965 formatting. Notable differences: the keyword to specify the path is now capitalized; the representation of the cookie's value is now double-quoted.

```
;; RFC2109
(rfc2109-set-cookie-string 'war 'lose #:path "/ignorance/suffering")
⇒ "Set-Cookie: war=lose; path=/ignorance/suffering"

;; RFC2965
(use-modules ((www server-utils answer) #:select (walk-tree)))

(define TREE (rfc2965-set-cookie2-tree
              '(war lose #:Path "/ignorance/suffering" #:Discard)))

(car TREE)
⇒ #:Set-Cookie2

(walk-tree display (cdr TREE))
⊣ war="lose";Path="/ignorance/suffering";Discard;Version=1
```

To generate a cookie spec from the `Cookie` http response header sent by a client, you can use `rfc2965-parse-cookie-header-value`.

`rfc2965-parse-cookie-header-value` *s* [*flags...*] [Procedure]

Parse the `Cookie` HTTP response header string *s*. Return a list of the form `(`*`vers n`* `[`*`cookie-spec...`*`])`, where *vers* is the version number of the cookie specification, 0 (zero) for RFC2109 compliance and 1 (one) for RFC2965 compliance; and *n* is the number of cookie-specs the CDR of the form.

Each *cookie-spec* has the form: `(`*`name value a1 v1...`*`)`. *name*, *value* are strings. Each *a* is a keyword, one of `#:Path`, `#:Domain` or `#:Port`. Each *v* is a string, except for that associated with `#:Port`, which is can be either a single number or a list of numbers.

Optional *flags* configure the parsing and/or return value.

`#:keep-attribute-dollarsign-prefix`
: Prevent conversion of, for example, `#:$Port` to `#:Port`.

`#:strict-comma-separator`
: Disable support for older clients that use a semicolon to separate cookies instead of a comma. Normally, parsing copes (heuristically) with this by reparsing an unrecognized attribute as the beginning of a new cookie. With this flag, an unrecognized attribute signals an error.

`#:canonicalize-NAME-as-keyword`
: Convert the *name* in each cookie-spec into a keyword whose first character and characters following a hyphen are upcased. For example, `"session-id-no"` would become `#:Session-Id-No`.

Parsing may signal an error and display an error message in the form: "*situation* while *context*", where *situation* is one of "unexpected end", "missing equal-sign", "bad attribute", or "missing semicolon"; and *context* is one of: "reading string", "reading token", "reading pair", "reading one cookie" or "parsing". The error message also displays string *s* on a line by itself and on the next line a caret by itself indented to be at (or near) the site of the error.

RFC2965 also specifies some other small algorithms, some of which are codified as procedures available in this module.

`reach` *h* [Procedure]

Return the *reach* (a string) of host name *h*. Quoting from RFC2965 section 1 (Terminology):

The reach R of a host name H is defined as follows:
If
- H is the host domain name of a host; and,
- H has the form A.B; and
- A has no embedded (that is, interior) dots; and
- B has at least one embedded dot, or B is the string `"local"`.

then the reach of H is .B.
Otherwise, the reach of H is H.

Note that comparison with `"local"` uses `string=?`, i.e., case-sensitively.

13 (www server-utils answer)

The `(www server-utils answer)` module provides a simple wrapper around the formatting/accounting requirements of a standard HTTP response. Additionally, the `#:rechunk-content` facility allows some degree of performance tuning; a server may be able to achieve better throughput with certain chunk sizes than with others.

The output from `compose-response`, `mouthpiece` and `string<-headers` is formatted according to their optional *style* argument. By default, headers have the form:

```
NAME ": " VALUE CR LF
```

Additionally, for `compose-response` and `mouthpiece`, the first line, preceding all the headers, has the form:

```
"HTTP/" MAJOR "." MINOR SP NNN SP MSG
```

and a single `CRLF` pair separates the headers from the body. (Actually, *mouthpiece* hardcodes the protocol version to '`1.0`', which is one reason why new code should use `compose-response`.) See Chapter 15 [modlisp], page 37, for another way to format this information.

`compose-response` *host* [`keyword value...`] [Procedure]

Keywords: `style`, `protocol-version`

Return a command-delegating closure capable of writing a properly formatted HTTP 1.1 response with `Host` header set to *host*. The actual status and header format is controlled by *style*, an opaque object. The actual protocol version is controlled by *protocol-version*, a pair of integers, such as `(1 . 0)` to indicate HTTP 1.0.

The returned closure *r* accepts commands and args:

`#:set-protocol-version` *pair*
: Set the major and minor version protocol-version numbers.

`#:set-reply-status` *number message*
: Set the reply status. *message* is a short string.

`#:add-header` *name value*
: *name* may be `#f`, `#t`, a string, symbol or keyword. *value* is a string. If *name* is `#f` or `#t`, *value* is taken to be a pre-formatted string, `"A: B"` or `"A: B\r\n"`, respectively. If *name* is not a boolean, *value* may also be a tree of strings or a number.

`#:add-content` [*tree* ...]
: *tree* may be a string, a nested list of strings, or a series of such. Subsequent calls to `#:add-content` append their trees to the collected content tree thus far.

`#:add-formatted` *format-string* [*args* ...]
: *format-string* may be `#f` to mean `~S`, `#t` to mean `~A`, or a normal format string. It is used to format *args*, and the result passed to `#:add-content`.

`#:add-direct-writer` *len write*
: *len* is the number of bytes that procedure *write* will output to its arg, *out-port* (passed back), when called during `#:send-reply`. This is to allow sendfile(2) and related hackery.

`#:entity-length`
: Return the total number of bytes in the content added thus far.

`#:rechunk-content` *chunk*
: *chunk* may be `#f`, in which case a list of the string lengths collected thus far is returned; `#t` which means to use the content length as the chunk size (effectively producing one chunk); or a number specifying the maximum size of a chunk. The return value is a list of the chunk sizes.

  It is an error to use `#:rechunk-content` with a non-`#f` *chunk* in the presence of a previous `#:add-direct-writer`.

`#:inhibit-content!` *bool*
: Non-`#f` *bool* arranges for `#:send-reply` (below) to compute content length and add the appropriate header, as usual, but no content is actually sent. This is useful, e.g., when answering a `HEAD` request. If *bool* is `#f`, `#:send-reply` acts normally (i.e., sends both headers and content).

`#:send!` *sock* [*flags*]
: Send the properly formatted response to file-port *sock*. It is an error to invoke `#:send-reply` without having first set the reply status.

  Optional arg *flags* are the same as for `send-request`. See Chapter 1 [http], page 1.

`mouthpiece` *out-port* [*status-box* [*style*]] [Procedure]

Return a command-delegating closure capable of writing a properly formatted HTTP 1.0 response to *out-port*. Optional arg *status-box* is a list whose CAR is set to the numeric status code given to a `#:set-reply-status` command. If *status-box* has length of two or more, its CADR is set to the content-length on `#:send-reply`. A content-length value of `#f` means there have been no calls to `#:add-content`. The commands and their args are:

`#:reset-protocol!`
: Reset internal state, including reply status, headers and content. This is called automatically by `#:send-reply`.

`#:set-reply-status` *number message*
: Set the reply status. *message* is a short string.

`#:set-reply-status:success`
: This is equivalent to `#:set-reply-status 200 "OK"`.

`#:add-header` *name value*
: *name* may be `#f`, `#t`, a string, symbol or keyword. *value* is a string. If *name* is `#f` or `#t`, *value* is taken to be a pre-formatted string, `"A: B"` or `"A: B\r\n"`, respectively. If *name* is not a boolean, *value* may also be a tree of strings or a number.

`#:add-content` [*tree* `...`]
: *tree* may be a string, a nested list of strings, or a series of such. Subsequent calls to `#:add-content` append their trees to the collected content tree thus far.

`#:add-formatted` *`format-string`* `[`*`args ...`*`]`
: *format-string* may be `#f` to mean `~S`, `#t` to mean `~A`, or a normal format string. It is used to format *args*, and the result passed to `#:add-content`.

`#:add-direct-writer` *`len write`*
: *len* is the number of bytes that procedure *write* will output to its arg, *out-port* (passed back), when called during `#:send-reply`. This is to allow sendfile(2) and related hackery.

`#:content-length`
: Return the total number of bytes in the content added thus far.

`#:rechunk-content` *`chunk`*
: *chunk* may be `#f`, in which case a list of the string lengths collected thus far is returned; `#t` which means to use the content length as the chunk size (effectively producing one chunk); or a number specifying the maximum size of a chunk. The return value is a list of the chunk sizes.

  It is an error to use `#:rechunk-content` with a non-`#f` *chunk* in the presence of a previous `#:add-direct-writer`.

`#:inhibit-content!` *`bool`*
: Non-`#f` *bool* arranges for `#:send-reply` (below) to compute content length and add the appropriate header, as usual, but no content is actually sent. This is useful, e.g., when answering a `HEAD` request. If *bool* is `#f`, `#:send-reply` acts normally (i.e., sends both headers and content).

`#:send-reply [close]`
: Send the properly formatted response to *out-port*, and reset all internal state (status reset, content discarded, etc). It is an error to invoke `#:send-reply` without having first set the reply status.

  Optional arg *close* means do a `shutdown` on *out-port* using *close* — directly, if an integer, or called with no arguments, if a thunk — as the shutdown `how` argument. (Note: If *out-port* is not a socket, this does nothing silently.) See Section "Network Sockets and Communication" in guile.

  If *close* is specified, the closure forgets about *out-port* internally; it is an error to call other mouthpiece commands, subsequently.

example

Here is an example that uses most of the `mouthpiece` commands:

```
(use-modules (www server-utils filesystem) (scripts slurp))

(define SERVER-NAME "Guile-WWW-example-server")
(define SERVER-VERSION "1.0")
(define STATUS (list #f #f))
(define M (mouthpiece (open-output-file "fake") STATUS))

(define (transmit-file filename)
```

```
  (M #:set-reply-status:success)
  (M #:add-header 'Server (string-append SERVER-NAME "/"
                                         SERVER-VERSION))
  (M #:add-header 'Connection "close")
  (M #:add-header 'Content-Type (filename->content-type
                                filename "text/plain"))
  (M #:add-content (slurp filename))
  (simple-format #t "rechunked: ~A~%"
                 (M #:rechunk-content (* 8 1024)))
  ;; We don't shutdown because this is a file port;
  ;; if it were a socket, we might specify 2 to
  ;; stop both reception and transmission.
  (M #:send-reply))

(transmit-file "COPYING")
⊣ rechunked: (8192 8192 1605)
STATUS
⇒ (200 17989)
```

For higher performance, you can preformat parts of the response, using `CRLF`, and some lower-level convenience procedures. If preformatting is not possible (or desirable), you can still declare a nested list of strings (aka *tree*) to have a *flat length*, i.e., the size in bytes a tree would occupy once flattened, thus enabling internal optimizations. (The flat length of a string is its `string-length`.)

`CRLF` [Constant String]
: The string "\r\n".

`flat-length` *object* [Object Property]
: Return the flat length of *object*, or `#f` if not yet computed.

`fs` *s* [*args*...] [Procedure]
: Return a new string made by using format string *s* on *args*. As in `simple-format` (which this procedure uses), `~A` expands as with `display`, while `~S` expands as with `write`.

`walk-tree` *proc tree* [Procedure]
: Call *proc* for each recursively-visited leaf in *tree*, excluding empty lists. It is an error for *tree* to contain improper lists.

`tree-flat-length!` *tree* [Procedure]
: If *tree* is a string, return its `string-length`. If *tree* already has a `flat-length`, return that. Otherwise, recursively compute, set, and return the `flat-length` of *tree*.

`string<-tree` *tree* [Procedure]
: Return a new string made from flattening *tree*. Set the `flat-length` (using `tree-flat-length!`) of *tree* by side effect.

`string<-headers` *alist* [*style*] [Procedure]

Return a string made from formatting name/value pairs in *alist*, according to the optional `style` argument. If unspecified or specified as `#f`, the default is to format headers like so:

```
NAME #\: #\space VALUE #\cr #\lf
```

Each name may be a string, symbol or keyword. Each value may be a string, number, symbol, or a tree.

example

Here is `transmit-file` from the above example, slightly modified to use preformatted headers and `fs`:

```
(define CONSTANT-HEADERS
  (string<-headers
   `((#:Server     . ,(fs "~A ~A" SERVER-NAME SERVER-VERSION))
     (#:Connection . "close"))))

(define (transmit-file filename)
  (M #:set-reply-status:success)
  (M #:add-header #t CONSTANT-HEADERS)
  (M #:add-header 'Content-Type (filename->content-type
                                 filename "text/plain"))
  (M #:add-content (slurp filename))
  (display (fs "rechunked: ~A~%" (M #:rechunk-content (* 8 1024))))
  (M #:send-reply))
```

Note that `mouthpiece` accepts trees for both `#:add-header` and `#:add-content` commands. Thus, the following two fragments give the same result, although the latter is both more elegant and more efficient:

```
;; Doing things "manually".
(walk-tree (lambda (string)
             (M #:add-content string))
           tree)

;; Letting the mouthpiece handle things.
(M #:add-content tree)
```

14 (www server-utils log)

The `(www server-utils log)` module provides procedure generators for writing log information to an output port. Each generator is conventionally named `log-SOMETHING-proc`.

`string<-sockaddr` *saddr* [Procedure]

Return a string describing the `AF_INET` or `AF_UNIX` socket address object *saddr*. This is typically found as the CDR of the *accept* return value.

For `AF_UNIX`, return `"localhost"` unless (somehow) the expression `(sockaddr:path saddr)` has non-`#f` and non-empty-string value.

For `AF_INET` the format is *`hostname`*`:`*`port`*, where *hostname* is from `inet-ntoa` and *port* is an integer.

For any other family, return what `object->string` returns.

`log-http-response-proc` *port* [*gmtime?* [*stamp-format* [*method-pair?*]]] [Procedure]

Return a procedure that writes an HTTP response log entry to *port*. The procedure is called with args *client*, *method*, *upath* (strings or symbols) and *status* (either an atom or a list), and writes a one-line entry of the form:

```
CLIENT - - [YYYY-MM-DD:HH:MM:SS TZ] "METHOD UPATH" ST1 ST2...
```

where the '`YYYY..TZ`' are the year, month, day, hour, minute, second and timezone components, respectively, of the `localtime` representation of the current time; and '`STn`' are the space-separated elements of *status*.

Optional second arg *gmtime?* non-`#f` means use `gmtime` instead of `localtime`. Optional third arg *stamp-format* specifies a format string passed to `strftime` to use for the timestamp portion that appears between the square braces (default: `"%Y-%m-%d:%H:%M:%S %Z"`).

Optional fourth arg *method-pair?* non-`#f` means that *method* is expected to be a pair `(`*`meth . vers`*`)`, in which case the portion between the double quotes becomes "*meth upath vers*". This is to support excruciating conformity to Apache for the benefit of downstream programs that might fall over less than gracefully otherwise. Please enjoy the slack.

The buffering mode for *port* is set to line-buffered.

15 (www server-utils modlisp)

The `(www server-utils modlisp)` module provides support for the implementing the Lisp side of the Apache mod_lisp protocol, in the form of a header-grokking protocol object for the big dishing loop, and a style elements object for the mouthpiece. When these objects are specified, the headers are read from (written to) the Apache front end in the form:

```
name #\lf value #\lf
```

with a lone '`end\n`' to separate the headers from the body. Furthermore, on input, the headers must include `method`, `url` and `server-protocol`. On output, the status information (always output first) has the form:

```
"Status" #\lf nnn #\space msg #\lf
```

Note that this is in essense the same format as used for the headers, with *name* being '`Status`' and *value* being '*`nnn msg`*'.

`modlisp-hgrok` [Object]
: An object suitable for the `#:style` argument to both `make-big-dishing-loop` (see Chapter 7 [big-dishing-loop], page 15) and `receive-request` (see Chapter 8 [parse-request], page 18).

`modlisp-ish` [Object]
: An object suitable as the optional `style` argument for `string<-headers`, `compose-response` and `mouthpiece`. See Chapter 13 [answer], page 31.

Although these are separate objects, you should probably use or not use them in conjunction, lest the front-end (Apache) server become confused.

16 (www data http-status)

The `(www data http-status)` module exports a single procedure:

`http-status-string` *number* [Procedure]
Return the string associated with HTTP status *number*.

example

Here is a simple example using this module:

```
(use-modules ((www data http-status)
              #:select (http-status-string)))

(define (h2 n)
  (format #f "<H2>~A ~A</H2>"
          n (http-status-string n)))

(h2 404) ⇒ "<H2>404 Not Found</H2>"
(h2 307) ⇒ "<H2>307 Temporary Redirect</H2>"
```

17 (www data mime-types)

The `(www data mime-types)` module maintains an internal hash table mapping filename extensions to one or more *mime-types*.

The exported procedures provide convenience abstractions over the underlying hash-table manipulation operations, including extension and mime-type validation, init from a file in a "standard" format (i.e., that of `/etc/mime.types` or `~/.mime.types`), and support for straightforward incremental init (aka *merging*). There are two predefined entries in the hash table:

```
text => text/plain
html => text/html
```

To support merging, the `put-FOO` procedures both take a symbol *resolve* as the first arg, which specifies how *conflicts* should be handled. This happens when the hash table already contains an entry for *extension* and *new-mime-type* differs from *old-mime-type*.

`error` Throw an error with key `mime-type-conflict`, displaying a message describing the *extension*, *old-mime-type* and *new-mime-type*.

`prefix` Make the mime-type of *extension* a list (unless already one), with *new-mime-type* at the beginning.

`suffix` Make the mime-type of *extension* a list (unless already one), with *new-mime-type* at the end.

`stomp` Use *new-mime-type* directly, discarding *old-mime-type*.

`quail` Discard *new-mime-type*, keeping *old-mime-type*.

For any other method, the operation throws an error, with key `invalid-resolve`.

Validation happens on all "put" operations. The extension must be a symbol, such as `txt`. The mime-type must be a symbol with exactly one '`/`' (slash) in its name, such as `text/plain`, or a proper list of such symbols. The mime-type may also be `#f`, which means to remove *extension* from the hash table.

If an entry does not validate, the operation throws an error, with key `invalid-extension` or `invalid-mime-type`.

`reset-mime-types!` *size* [Procedure]
Clear all entries from the mime-types hash table, and prepare it for *size* (approximately) entries. This procedure must be called before any others in this module.

`put-mime-types-from-file!` *resolve filename* [Procedure]
Open *filename* and parse its contents as "mime-types" format. This line-oriented file format is briefly described as follows:

- Blank lines and lines beginning with '`#`' are ignored.
- Lines of the format *mime-type* (only one symbol) are ignored.
- Otherwise, the line is expected to be in the format *`mime-type extension extension...`*, that is, at least one *extension* must be present. Each *extension* results in an entry in the hash table.

Put those those entries that specify an extension into the hash table, validating both extension and mime-type first. *resolve* specifies how to resolve extension conflicts.

`put-mime-types!` *resolve* [*extension1 mime-type1 ...*] [Procedure]
Put *extension1*/*mime-type1*... into the hash table, validating both extension and mime-type first. *resolve* specifies how to resolve extension conflicts.

If an extension is given but there is no mime-type (i.e., the list has an odd length), throw an error with key `missing-mime-type`.

`mime-types<-extension` *ext* [Procedure]
Return the mime-type(s) associated with *ext* (a symbol or string), or `#f` if none are found. Note that generally the value may be a single mime-type or a list of them.

`select-extensions` *sel* [Procedure]
Return a list of extensions in the hash table that match the *sel* criteria (a symbol). If *sel* is `#t`, return all the extensions; if `single`, only those who have a single mime-type associated; if `multiple`, only those who have more than one mime-type associated.

why `select-extensions`?

The last procedure is intended to ease non-generalizable merging, without providing too much exposure to implementation internals. Suppose you want to maintain a local policy of having only one mime-type associated per extension (to keep things simple). In that case, after populating the hash, you can fix up those entries, like so:

```
(reset-mime-types! 491)
(put-mime-types-from-file! 'prefix "/etc/mime.types")
(define AMBIGUOUS (select-extensions 'multiple))

(use-modules (ice-9 format))
(define (display-ext ext)
  (format #t "~7,@A  ~A~%" ext (mime-types<-extension ext)))

(for-each display-ext AMBIGUOUS)
    ent  (chemical/x-ncbi-asn1-ascii chemical/x-pdb)
    sdf  (application/vnd.stardivision.math chemical/x-mdl-sdfile)
     sh  (application/x-sh text/x-sh)
    csh  (application/x-csh text/x-csh)
    cpt  (application/mac-compactpro image/x-corelphotopaint)
    asn  (chemical/x-ncbi-asn1 chemical/x-ncbi-asn1-spec)
    wrl  (model/vrml x-world/x-vrml)
    tcl  (application/x-tcl text/x-tcl)
     ra  (audio/x-pn-realaudio audio/x-realaudio)
    spl  (application/futuresplash application/x-futuresplash)
    m3u  (audio/mpegurl audio/x-mpegurl)

;; Local policy: For foo.wrl, we want the last variant,
;; but everything else we'll settle for the first.
(define ((keep! yes) ext)
  (put-mime-types!
   'stomp ext
   (yes (mime-types<-extension ext))))
```

```
((keep! reverse) 'wrl)
(for-each (keep! car) AMBIGUOUS)

(for-each display-ext AMBIGUOUS)
    asn  chemical/x-ncbi-asn1
    wrl  x-world/x-vrml
    tcl  application/x-tcl
     ra  audio/x-pn-realaudio
    spl  application/futuresplash
    m3u  audio/mpegurl
    ent  chemical/x-ncbi-asn1-ascii
    sdf  application/vnd.stardivision.math
     sh  application/x-sh
    csh  application/x-csh
    cpt  application/mac-compactpro
```

Seasoned schemers will note that the same result could have been achieved if *resolve* were allowed to be a general resolution procedure instead of simply a method specifier. Perhaps that feature will be added in the future, and `select-extensions` replaced by `map-mime-types`. We'll see. . .

Appendix A GNU Free Documentation License

Version 1.3, 3 November 2008

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or non-commercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released

under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

   If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

   If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

   If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

   It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

   You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

   A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any,

be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their

titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

   You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

   The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

   In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

   You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

   You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

   A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

   If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

   Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

   If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

   You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

   However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

   Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

   Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

    The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

    Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

    "Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

    "CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

    "Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

    An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

    The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C)  year  your name.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.3
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover
Texts.  A copy of the license is included in the section entitled ``GNU
Free Documentation License''.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with. . . Texts." line with this:

```
with the Invariant Sections being list their titles, with
the Front-Cover Texts being list, and with the Back-Cover Texts
being list.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Index

#

<

A

B

C

E

F

H

I

L

M

N

P

Q

R

S

T

U

W