Web Server Manual

Mike Burns (netgeek@speakeasy.net)
Greg Pettyjohn (gregp@ccs.neu.edu)
Jay McCarthy (jay.mccarthy@gmail.com)

January 12, 2006

Copyright notice

Copyright ©1996-2005 PLT

Permission to make digital/hard copies and/or distribute this documentation for any purpose is hereby granted without fee, provided that the above copyright notice, author, and this permission notice appear in all copies of this documentation.

Contents

1 Quick Start **1**

1.1 Quick Start on Unix 1

I Administration **2**

2 Directory Structure **3**

3 Configuration **4**

3.1 Named Virtual Hosts 4

3.2 The Configuration Tool 4

3.2.1 Managing Virtual Hosts 4

3.3 Configuration File Syntax 4

3.3.1 Host Table Syntax 6

4 Passwords **9**

5 Starting the Server **10**

6 Monitoring the Server **11**

II Programming **12**

7 Before You Begin **13**

7.1 Reloading the Cache 13

7.2 Directories 13

8 Static Content **14**

9 Writing Servlets **15**

9.1 Module-Based Servlets 15
9.2 Unit-Based Servlets 16

10 Servlet Library **17**

10.1 Data Definitions 17
10.1.1 Environment 17
10.1.2 Request 17
10.1.3 Response 18
10.2 Core Procedures 19
10.3 Helpful Servlet Procedures 20
10.4 Example Multiplication Servlet 21
10.5 Example Math Test Servlet 23
10.6 Servlet Development Environment 23

11 Semi-Internal Functions **25**

11.1 Miscellaneous 25
11.2 Starting the Server from a Program 26
11.3 Constructing Configurations 26
11.4 Monitoring the Server 27
11.4.1 Example 27

Index **28**

1. Quick Start

To just use a servlet without caring about administrative or programming details:

1.1 Quick Start on Unix

1. If you have root access, just run `plt/bin/web-server`; otherwise, run `plt/bin/web-server -p 8080`.
2. Example servlets are located in `collects/web-server/default-web-root/servlets/examples/`.
3. To run the example servlet **add.ss**, use the URL `http://localhost/servlets/examples/add.ss`, or `http://localhost:8080/servlets/examples/add.ss` if the server was started with `-p 8080`.

Part I

Administration

2. Directory Structure

By default, the configuration tool (see section 3.2) organizes files containing the Web server's configurations, documents, and servlets in one directory tree per virtual host. This tool can be used to manage all trees for the Web server. A default tree exists in `collection/web-server/`, which looks like:

```
configuration-table
default-web-root
  conf
    servlet-error.html
    forbidden.html
    servlet-refresh.html
    passwords-refresh.html
    not-found.html
    protocol-error.html
  htdocs
    Defaults
      index.html
      documentation
        ...
  log
  passwords
  servlets
    configure.ss
my-other-host
  conf ...
  htdocs ...
  log
  passwords
  servlets
still-another-host ...
```

Files may be relocated or shared between hosts by editing the details for that host using the configuration tool. For more on virtual hosts, see section 3.1.

3. Configuration

3.1 Named Virtual Hosts

Many Web sites can be served from one Web server process through the use of virtual hosting. When a client (Web browser) sends a request to the server, the domain name is embedded in the request. The Web server can then service the request in different ways based on that requested domain name.

Zero or more virtual hosts may be defined, in addition to the default host. If a request is sent to the server for a host which is not in the virtual host table, then the default host is used. For example, if the client requests `http://www.plt-scheme.org/`, the server can send back information in the directory `/var/www/plt/www.plt-scheme.org/`; if the client requests, from the save server, `http://www.drscheme.org/`, the server can send back information from the directory `/home/mburns/www.drscheme.org/`.

To configure virtual hosting using the configuration tool, see section 3.2.1. To configure virtual hosting by editing the file directly, see `virtual-host-table`.

3.2 The Configuration Tool

The configuration tool is a normal servlet, run by the Web server, that can manage the configuration files. This is provided as an alternative to editing the file directly.

For security reasons, the configuration servlet is only accessible from the local machine. Thus, one gets to the servlet via `http://localhost/servlets/configure.ss`.

Once there, select the filename for the configuration file (the default should work). As part of the basic configuration, you can change the port to listen on, the maximum number of waiting connections, and the initial timeout. Individual hosts can be added and edited below the basic configuration.

3.2.1 Managing Virtual Hosts

Adding a virtual host is as simple as selecting “Add Host.” The page will reload with the information for the new host; simply change the name to be the domain name for the new host, and change the directory to be the correct directory for the virtual host. The Web server must be restarted for these settings to take effect.

In addition to the virtual hosts’s domain and Web root, the details of all hosts (including the default host) can be changed by selecting “Edit Minor Details.” From there, the log file, static files directory, password file, timeout values, and message files can all be modified. For details, see section 3.3.

3.3 Configuration File Syntax

Instead of using the configuration tool, as described in section 3.2, the configuration file can be edited “by hand”. The provided, default configuration file is `collects/web-server/configuration-table`. The file has the following syntax.

```
((port natural-number)
 (max-waiting natural-number)
 (initial-connection-timeout number)
 (default-host-table host-table)
 (virtual-host-table (string host-table) ...))
```

`port : natural-number` Usually `80`, indicating which port to listen on by default.

`max-waiting : natural-number` Usually `40`, indicating the maximum number of clients that can wait for a TCP/IP connection to the Web server. This limit usually does not cause problems because the Web server quickly accepts connections as requested and there is no limit on the number of simultaneous connections. On some platforms, choosing a large number may consume too many resources.

`initial-connection-timeout : natural-number` Usually `30`. After a client connects to the Web server, the server only waits for an HTTP request for this many seconds before closing the connection. Browsers tend to send requests immediately, so a small number is best . This number should be kept small to prevent denial-of-service attacks.

`default-host-table : host-table` Described in section 3.3.1.

`virtual-host-table : (listof (cons string host-table))` A list of `(string host-table)`, where the `host-table` is defined in section 3.3.1 and the `string` represents the named virtual host's domain name.

For example, let the virtual host table be

```
(virtual-host-table
 ("www.plt-scheme.org"
  (host-table
    (default-indices "index.shtml" "index.html")
    (log-format parenthesized-default)
    (messages
      (servlet-message "servlet-error.html")
      (authentication-message "forbidden.html")
      (servlets-refreshed "servlet-refresh.html")
      (passwords-refreshed "passwords-refresh.html")
      (file-not-found-message "not-found.html")
      (protocol-message "protocol-error.html")
      (collect-garbage "collect-garbage.html"))
    (timeouts
      (default-servlet-timeout 120)
      (password-connection-timeout 300)
      (servlet-connection-timeout 86400)
      (file-per-byte-connection-timeout 1/20)
      (file-base-connection-timeout 30))
    (paths
      (configuration-root "errors")
      (host-root "/var/www/www.plt-scheme.org")
      (log-file-path "/var/log/www.plt-scheme.org")
      (file-root "htdocs")
      (servlet-root ".")
      (password-authentication "passwd")))))
```

Requesting the URL `http://localhost/` will use the `default-host-table`, but requesting the URL `http://www.plt-scheme.org/`, assuming DNS is set up correctly, will return the first of

1. **/var/www/www.plt-scheme.org/index.shtml**

2. **/var/www/www.plt-scheme.org/index.html**

For port-based virtual hosting, run separate instances of the Web server.

3.3.1 Host Table Syntax

The host table is a part of the configuration file with the following syntax

```
(host-table
 (default-indices string ...)
 (log-format parenthesized-default)
 (messages
  (servlet-message string)
  (authentication-message string)
  (servlets-refreshed string)
  (passwords-refreshed string)
  (file-not-found-message string)
  (protocol-message string)
  (collect-garbage string))
 (timeouts
  (default-servlet-timeout number)
  (password-connection-timeout number)
  (servlet-connection-timeout number)
  (file-per-byte-connection-timeout number)
  (file-base-connection-timeout number))
 (paths
  (configuration-root string)
  (host-root string)
  (log-file-path string)
  (file-root string)
  (servlet-root string)
  (password-authentication string)))
```

`default-indices : (listof string)` This is a list of index files for all directories. When a directory is specified as the URL, such as `http://www.plt-scheme.org/`, the Web server searches for a file to display, choosing from this list in the specified order. A good default is

```
(default-indices "index.html" "index.htm")
```

For example, if `default-indices` is set to

```
(default-indices "index.ss" "index.html" "index.htm" "last-resort")
```

and the user requests `http://www.plt-scheme.org/`, the Web server will attempt to serve the following URLS, in order, until a file is found:

1. `http://www.plt-scheme.org/index.ss`
2. `http://www.plt-scheme.org/index.html`
3. `http://www.plt-scheme.org/index.htm`
4. `http://www.plt-scheme.org/last-resort`

`log-format : symbol` The format to use for the log file. Currently only the symbol `parenthesized-default` is supported.

3.3.1.1 MESSAGES

`servlet-message : string` A filename, usually `"servlet-error.html"`, of the page to display when there is an error with the servlet. This file is located relative to the `configuration-root` directory.

`authentication-message : string` A filename, usually `"forbidden.html"`, of the page to display when there user fails authentication. This file is located relative to the `configuration-root` directory.

`servlets-refreshed : string` A filename, usually `"servlet-refresh.html"`, of the page to display when the servlets are refreshed. This file is located relative to the `configuration-root` directory.

`passwords-refreshed : string` A filename, usually `"passwords-refresh.html"`, of the page to display when the password file is refreshed. This file is relative to the `configuration-root` directory.

`file-not-found-message : string` A filename, usually `"not-found.html"`, of the page to display when the requested file was not found (HTTP error number 404). This file is located relative to the `configuration-root` directory.

`protocol-message : string` A filename, usually `"protocol-error.html"`, of the page to display when the Web browser violates the HTTP standard. This should never appear when interacting with the server through a correct Web browser. The server may close the connection instead of sending this error, depending on the severity of the problem.

`collect-garbage : string` A filename, usually `"collect-garbage.html"`, of the page to display when the garbage collected is run. This file is relative to the `configuration-root` directory.

3.3.1.2 TIMEOUTS

`default-servlet-timeout : number` This value, usually `120`, determines how long, in seconds, a servlet can wait in between requests before shutting down, unless the servlet calls `adjust-timeout!` to alter this value. Large values consume more memory, while smaller values annoy end users who must restart servlets.

`password-connection-timeout : number` Usually `300`, this is the number of seconds a user has to enter a username and password.

`servlet-connection-timeout : number` Usually `86400` (one day), this is the number of seconds a servlet has to respond to a single request. Some servlets may perform substantial computation on a loaded server before generating a page. On the other hand, most users will not wait very long for a response.

`file-per-byte-connection-timeout : number` Usually `1/20`. Let *`bytes`* by the bytes in a file. The total number of seconds the server has to send a file is `file-base-connection-timeout` + `file-per-byte-connection-timeout` $*$ *bytes*

`file-base-connection-timeout : number` Usually `30`. Let *`bytes`* by the bytes in a file. The total number of seconds the server has to send a file is `file-base-connection-timeout` + `file-per-byte-connection-timeout` $*$ *bytes*

3.3.1.3 PATHS

`configuration-root : string` The directory under which the files in section 3.3.1.1 may be found. Usually `"conf"`.

`host-root : string` The root directory for Web files. Under this directory is usually the `servlet-root`, `configuration-root`, `log-file-path`, `file-root`, and `password-authentication`. Usually `"default-web-root"`.

`log-file-path : string` The name of the log file. Usually `"log"`.

`file-root : string` The directory from which static (HTML) files are served. Usually `"htdocs"`.

`servlet-root : string` The directory from which dynamic (servlet) files are run and served. Causes odd behavior when set to anything besides `"."`. The servlets directory is always `servlets/`, under the `host-root`.

`password-authentication : string` The file from which passwords are read. For information on its format, see section 4. Usually `"passwords"`.

4. Passwords

The PLT Web server provides *basic* authentication following RFC 2617. These passwords are stored in **passwords**. The Web server caches passwords for performance reasons. Requesting the URL `http://localhost/conf/refresh-passwords` reloads the password file. The format is:

```
'((realm path-regexp (name password) ...) ...)
```

`realm : string` A unique identifier for this set of passwords.

`path-regexp : string` A regular expression applied to the resource. If it matches, then this set of passwords is used for this resource.

`name : symbol` The login name for a user. Note that case-sensitivity depends on the current settings in MzScheme.

`password : string` The password, *unencrypted*, for the user.

For example, to hide anything in the `/secret/` directory under the `host-root`, but allow access to `bubba` with the password `bbq` and `Billy` with the password `BoB`, use

```
'(("secret stuff" "/secret(/.*)?" (bubba "bbq") (|Billy| "BoB")))
```

5. Starting the Server

The Web server collection provides two programs for starting the Web server. Both accept the same options.

web-server Servlets can use the full MrEd functionality.

web-server-text Servlets can only use MzScheme functionality. This server cannot load servlets written using DrScheme's graphical XML boxes, uses less memory, and works with the MzScheme-only distribution of PLT.

The command line

```
web-server[-text] [-p <port>] [-f <configuration-table-file>] [-a <ip-address>]
```

starts the server on port 80 or *port* and uses the configuration options from the **configuration-table** file of the **web-server** collection or from the specified *configuration-table-file*. If *ip-address* is provided, the server binds to *ip-address*.

6. Monitoring the Server

When launched via

```
web-server-monitor [-p <port>]
                   [-f <frequency>]
                   [-t <timeout>]
                   <alert-email> <host-name>
```

the **web-server-monitor** polls any Web server running on host *host-name* at port *port* (or port 80) every *frequency* seconds (or 1 hour). If the server does not respond to a HEAD HTTP request for the homepage within *timeout* (or 75) seconds or sends an error response, the monitor will notify *alert-email* of the problem.

Part II

Programming

7. Before You Begin

7.1 Reloading the Cache

For efficiency, servlets are cached on their initial invocation. When editing a servlet, before changes can be observed, it is necessary to clear the cache by loading the URL `http://localhost/conf/refresh-servlets`.

7.2 Directories

Within the Web server's root directory, two important directories exist. Note that these names may change. Administrators should see section 2 and section 3.3.1.3.

`htdocs/` Under this directory exists static files which are just presented to the Web browser with an appropriate MIME type. See section 8 for the details.

`servlets/` Under this directory exists dynamic files which are processed by the Web server as servlets. See section 9 for the details.

8. Static Content

The Web server serves content statically or dynamically, depending on the request URL.

On static requests (those that do not match the Web server's servlet filter), the Web server serves files out of the content directory determined from `file-root` for the specified virtual host.

The Web server guarantees that for such static responses, it will serve only those files accessible in subdirectories of the content directory. In the absence of filesystem links, this means that only files which live in the content directory will be made available to the outside world. The server ignores the relative path specifiers `..` and `.` in URLs.

9. Writing Servlets

When a request URL matches the servlet filter, the Web server generates its response dynamically, (see section 7.2 for more explanation, and section 2 and section 3.3.1.3 for administrative details). Instead of serving the files in this directory verbatim, the server evaluates the file as a scheme program to produce output. Servlets are (by default) loaded in a case-sensitive manner. (Search in help-desk for `read-case-sensitive`.)

The path part of the URL supplies the file path to the servlet relative to the `servlets` directory. However, paths may also contain extra path components that servlets may use as additional input. For example all of the following URLs refer to the same servlet:

- `http://www.plt-scheme.org/servlets/my-servlet`
- `http://www.plt-scheme.org/servlets/my-servlet/extra`
- `http://www.plt-scheme.org/servlets/my-servlet/extra/directories`

The server supports two forms of servlets. For useful procedures to handle Web data, see section 10.

9.1 Module-Based Servlets

A module-based servlet is a module that provides three values: an `interface-version`, a `timeout`, and a `start` procedure.

```
(module a-module-servlet mzscheme
  (provide interface-version timeout start)

  (define interface-version 'v1)

  (define timeout +inf.0)

  ; start : request → response
  (define (start initial-request)
    `(html (head (title "A Test Page"))
           (body ([bgcolor "white"])
                 (p "This is a simple module servlet.")))))
```

`interface-version` : `symbol`

The `interface-version` is a symbol indicating how the server should interact with the servlet. The only supported value is `'v1` at this time.

`timeout : number`

The `timeout` is the number of seconds the server will allow the servlet to run before shutting it down. Large values consume more memory, while smaller values annoy users by forcing them to restart their session. The value can be adjusted dynamically by calling the `adjust-timeout!` procedure. For more information, see section 3.3.1.2 and section 10.2.

`start`

```
request -> response
(define (start request) ···)
```

The `start` function consumes a `request` and produces a `response`. Each time a client visits the URL associated with the beginning of the servlet, the server calls the `start` function with the `request` sent from the browser. The server then sends the `response` produced by the `start` function back to the browser.

9.2 Unit-Based Servlets

A unit-based servlet is a `unit/sig` that imports the `servlet^` signature and exports nothing. (See the manual for `unit/sig` and `signatures`.) To construct a signed unit with the appropriate imports, the servlet must require the two modules providing `unit/sig`s and the `servlet` signature:

```
(require (lib "unitsig.ss")
         (lib "servlet-sig.ss" "web-server"))
(unit/sig ()
  (import servlet^)
   ;;; ...insert servlet code here...
)
```

Evaluating

```
(require (lib "servlet-sig.ss" "web-server"))
```

loads the `servlet^` signature which contains the import `initial-request` of type `request`.

10. Servlet Library

To ease the development of interactive servlets, the **web-server** collection also provides the following data types and procedures:

10.1 Data Definitions

10.1.1 Environment

An `environment` is a `(listof (cons symbol string))`. That is, an `environment` is a `list` of improper `cons`-pairs of a `symbol` and a `string`.

A `byte-environment` is a `(listof (cons symbol bytes))`. That is, a `byte-environment` is a `list` of improper `cons`-pairs of a `symbol` and a `bytes`.

10.1.2 Request

A `request` is

```
(make-request symbol url environment environment string string)
(define-struct request (method uri headers bindings host-ip client-ip))
```

`method` One of

- `'get`
- `'post`

`uri` URL, see the **net** collection in help-desk for details.

`headers` An `environment` containing optional HTTP headers for this request.

`bindings` An `environment` containing optional name-value pairs from either the form submitted or the query part of the URL.

`bindings/raw` Either a string or a `byte-environment`. This is primarily used for file uploads.

`host-ip` A string representing what IP address of the server the request came to.

`client-ip` A string representing what IP address is associated with the client.

The bindings contain the information the user supplied when filling out a Web form and hence are usually the most useful part. This makes the following idiom common. For a full example, see Figure 10.2.

```
(extract-bindings/single
 'sym
 (request-bindings
  (send/suspend
   ; ...
   )))
```

10.1.3 Response

A `response` is one of the following:

an X-expression representing HTML Search for `XML` in help-desk. For example

```
'(html
  (head
    (title "Fruits")
    (link ((rev "made")
           (href "mailto:mburns@example.com")
           (title "Webmaster"))))
  (body
    (h1 "Fruits!")
    (ul
      (li "Banana")
      (li "Orange")
      (li "Grapefruit")
      (li "Shirley"))))
```

`(listof (union string bytes))` The first string is the MIME type (often `"text/html"`, but see RFC 2822 for other options). The rest of the strings provide the document's content.

`make-response/full`

`make-response/incremental`

`make-response/full`

```
natural-number string natural-number string environment environment(listof
(union string bytes)) -> response
(define (make-response/full code message seconds mime extras body) ...)
```

`code` A natural number indicating the HTTP response code.

`message` A string describing the code to a human.

`seconds` A natural number indicating the time the resource was created. Use `(current-seconds)` for dynamically created responses.

`mime` A string indicating the response type.

`extras` An environment containing extra headers for redirects, authentication, or cookies.

`body (listof (union string bytes))`

`make-response/incremental`

```
natural-number string natural-number string environment(((union string bytes)
-> void) -> void) -> response
(define (make-response/incremental code message seconds mime extras gen) ...)
```

`code`, `message`, `seconds`, `mime`, `extras` All the same as for `make-response/full`

`(gen` ***`output`***`):((union string bytes) -> void) -> void` The function `gen` consumes an `output` function. The `output` function consumes a string and sends it to the client. For HTTP/1.1 clients, the server uses the chunked encoding, which is reliable. HTTP/1.0 clients, however, can not distinguish between the end of the document and a lost connection. These facts have two implications. First, it is more efficient to send fewer, larger strings. Second, this response should not be used for data that must arrive reliably.

See also `make-html-response/incremental` in section 10.3.

10.2 Core Procedures

When writing module servlets, loading **servlet.ss** via

```
(require (lib "servlet.ss" "web-server"))
```

provides the following procedures:

`send/suspend`

```
(string -> response) -> request
(define (send/suspend page) ...)
```

The argument, a function that consumes a string, is given a URL that can be used in the document. The argument function must produce a response corresponding to the document's body. Requests to the given URL resume the computation at the point `send/suspend` was invoked. Thus, the argument function normally produces an HTML `form` with the `action` attribute set to the provided URL. The result of `send/suspend` represents the next request. This procedure is often used for interaction with the user. For an example, see section 10.4.

`send/forward`

```
(string -> response) -> request
(define (send/forward page) ...)
```

Acts like `send/suspend`, but clears the continuation table first. For example, if the servlet is an on-line quiz where the quiz-taker is not allowed to return to a previous page, use `send/forward`; this is how it is used in the example section 10.5.

`send/finish`

```
response -> void
(define (send/finish response) ...)
```

This provides a convenient way to report an error or otherwise produce a final response. Once called, all URLs generated by `send/suspend` become invalid. Calling `send/finish` allows the system to reclaim continuations used by the servlet. Often used as the final result of a servlet. For an example, see section 10.4.

`send/back`

```
response -> void
(define (send/back response) ...)
```

Acts like `send/finish`, but does not clear the continuation table. Useful for allowing the user to correct erroneous data.

To summarize the differences in the `send/` procedures, see Figure 10.1.

Type	Refreshes the continuation table	Does not refresh the continuation table
`response → void`	`send/finish`	`send/back`
`(string → response) → request`	`send/forward`	`send/suspend`

Figure 10.1: Differences in the send/ procedures

`adjust-timeout!`

```
natural-number -> void
(define (adjust-timeout! number) ...)
```

The server will stop an instance of a servlet after it has been idle for a certain amount of time, as described in the section 3.3.1.2. Calling `adjust-timeout!` allows the programmer to change the number of seconds before the servlet times out. Larger numbers consume more resources while smaller numbers force the user to restart computations more often.

Unit servlets receive these interaction procedures as `import`s through the `servlet^` signature.

10.3 Helpful Servlet Procedures

`extract-binding/single`

```
symbol environment -> string
(define (extract-binding/single sym environment) ...)
```

This extracts a single value associated with `sym` in the form bindings. If multiple or zero values are associated with the name, it raises an exception. For an example, see section 10.4.

`extract-bindings`

```
symbol environment -> (listof str)
(define (extract-bindings sym environment) ...)
```

Returns the list of values associated with the name `sym`.

`exists-binding?`

```
symbol environment -> bool
(define (exists-binding? sym environment) ...)
```

Returns true if the name `sym` is bound in the `environment`. This is useful for, e.g., checkboxes.

`extract-user-pass`

```
environment -> (union #f (cons str str))
(define (extract-user-pass environment) ...)
```

Servlets may implement password-based authentication by extracting password information from the HTTP headers. The return value is either a pair consisting of the username and password from the headers or `#f` if no password was provided. This only extracts the provided username and password; the servlet must perform any desired checking.

For more information on passwords, see section 4.

`report-errors-to-browser`

```
(response -> void) -> void
(define (report-errors-to-browser where) ···)
```

Calling this procedure at the beginning of a servlet causes otherwise uncaught exceptions to send an error page displaying the exception to the client. The argument should be based on `send/finish` if errors are fatal or `send/back` if the user may correct the failed form submission via the back button. For some applications, revealing the contents of an exception to a client may cause security problems by leaking sensitive information. For such applications, consider setting the `current-exception-handler` to email the error to the servlet author or store the exception in a log file.

`redirect-to`

```
string [symbol] -> response
(define (redirect-to url . redirection-status) ···)
```

Constructs a response that redirects to the given `url`. The optional argument specifies which kind of redirection to perform:

`permanently` Browsers should send future requests directly to this URL.

`temporarily` Browsers should send future requests to the original URL.

`see-other` The redirection is not replacing the current URL.

See the HTTP 1.1 specification for details on each kind of redirection. The default redirection type is `permanently`.

`make-html-response/incremental`

```
(((union string bytes) -> void) -> void) -> response/incremental
(define (make-html-response/incremental chunk-maker) ···)
```

This fills in default values for `make-response/incremental` to be appropriate for HTML.

Note that the function passed to this will be called in the connection thread, not the servlet thread. This means that e.g. `current-directory` will be the server's current directory, not the servlet's.

10.4 Example Multiplication Servlet

As an example of `send/suspend`, `send/finish`, and `request-bindings`, and `extract-bindings/single`, consider Figure 10.2. This servlet first prompts the user for the first number, then for the second number, and finally returns the product of the two numbers. Note that the final page is abstracted into a simple function, as is the common page to request a number.

For more example servlets, look in the `collects/web-server/default-web-root/sevlets/examples` directory.

```
(require (lib "unitsig.ss")
         (lib "servlet-sig.ss" "web-server"))

(unit/sig ()
  (import servlet^)

  ;; answer-page : Number → Xexpr
  (define (answer-page n)
    `(html
       (head
         (link ((rev "made") (href "mburns@example.com") (title "Webmaster"))))
       (body (h1 "Answer")
             (p ,(string-append "The answer is " (number→string n))))))

  ;; get-number : String → Number
  (define (get-number which)
    (string→number
     (extract-binding/single
      'num
      (request-bindings
       (send/suspend
        (get-number-page which))))))

  ;; get-number-page : String → String → Xexpr
  (define (get-number-page which)
    (lambda (k-url)
    `(html
       (head
         (link ((rev "made") (href "mburns@example.com") (title "Webmaster"))))
       (body (h1 "Enter a number")
         (form ((action ,k-url))
           (p ,(string-append "Please enter the " which " number"))
           (input ((type "text") (name "num") (id "num")))
           (input ((type "submit"))))))))

  (send/finish (answer-page (* (get-number "first") (get-number "second")))))
```

Figure 10.2: Example servlet using send/suspend, send/finish, request-bindings, and extract-binding/single

10.5 Example Math Test Servlet

The servlet in Figure 10.3 administers a simple math quiz to the user, keeping track of how many problems were answered correctly and incorrectly. As such, we don't want the user to simply hit the back button each time she enters a wrong answer; thus, we use `send/forward` to prevent against just that.

10.6 Servlet Development Environment

Choose "Add TeachPack..." from DrScheme's "Language" menu and select the **plt/teachpack/htdp/servlet.ss** teachpack. This provides functions for writing servlets including `send/suspend` and `send/finish`. All the extra servlet helper functions for extracting information from Web requests and building Web responses also become available through the TeachPack.

For information on **plt/teachpack/htdp/servlet2.ss**, see Extended Exercises.

The TeachPacks add the following functionality to the DrScheme environment:

- A Web browser is automatically started when needed.
- All the needed functions, described in section 10.2, as available.
- `send/suspend`, and the other needed functions, are available in the interactions window.

Currently, minor edits will need to be made to your existing servlets to use them with the TeachPacks.

```
(require (lib "unitsig.ss")
         (lib "servlet-sig.ss" "web-server"))

(unit/sig ()
  (import servlet^)

  ;; build-page : String Number Number Xexpr → Xexpr
  (define (build-page title num-wrong num-right body)
    `(html
      (head (title ,title)
            (link ((rev "made")
                   (href "mailto:netgeek@speakeasy.net")
                   (title "Webmaster"))))
      (body (h1 ,title)
            (p ,(string-append "Wrong: " (number→string num-wrong)))
            (p ,(string-append "Right: " (number→string num-right)))
            ,body)))

  ;; get-the-answer : Number Number Number Number → Number
  (define (get-the-answer num-wrong num-right a b)
    (string→number
     (extract-binding/single
      'answer
      (request-bindings
       (send/forward
        (get-the-answer-page num-wrong num-right a b))))))

  ;; get-the-answer-page : Number Number Number Number → String → Xexpr
  (define (get-the-answer-page num-wrong num-right a b)
    (lambda (k-url)
      (build-page
       "What's the answer?"
       num-wrong
       num-right
       `(form ((action ,k-url))
              (p ,(string-append (number→string a)
                                 " * "
                                 (number→string b)))
              (p (input ((type "text")
                         (id "answer")
                         (name "answer"))))
              (p (input ((type "submit"))))))))

  (let loop ((num-wrong 0)
             (num-right 0)
             (a 5)
             (b 7))
    (if (= (* a b) (get-the-answer num-wrong num-right a b))
        (loop num-wrong (+ num-right 1) (+ a 1) (+ b 1))
        (loop (+ num-wrong 1) num-right (+ a 1) (+ b 1)))))
```

Figure 10.3: Simple math test servlet, showing send/forward

11. Semi-Internal Functions

The following functions expose more of the Web server for use by the development environment. They are not intended for general use. They may change at anytime or disappear entirely. These are just for the developers.

11.1 Miscellaneous

`server-loop`

```
custodian (-> iport oport) (-> void) -> void
(define (server-loop custodian tcp-listener config initial-timeout) ···)
```

`custodian` The parent custodian for servlets.

`tcp-listener` Where requests arrive.

`config` Encapsulates most of the state of the server.

`initial-timeout` The number of seconds before timing out connections.

`make-config`

```
(string -> host) (hash-table-of sym script) (unit servlet-sig -> response)
(hash-table-of sym servlet-instance) -> config
(define (make-config hosts scripts instances access) ···)
```

host-table `string → host`

Maps host names to hosts

script-table `(hash-table-of sym script)`

Maps servlet names to servlet units

script `(unit servlet^ → response)`

Represents a servlet that is invoked on each request

instance-table `(hash-table-of sym servlet-instance)`

Maps the path part of a URL to the running servlet.

access-table `(hash-table-of sym (string sym string → (union \#f str)))`

Maps host names to functions that accept a protection domain, a user name, and a password and either return `#f` if access is not denied (i.e. is accessible) or a string prompting for a particular password (i.e. “Course Grades”).

servlet-instance `(make-servlet-instance nat channel (hash-table-of sym (continuation request)))`

The `natural-number` counts the continuations suspended by this servlet. The *channel* communicates HTTP requests from the connection thread to the suspended servlet. The *hash-table* maps parameter parts of URLs to suspended continuations.

11.2 Starting the Server from a Program

Requiring the library **web-server.ss**, via

```
(require (lib "web-server.ss" "web-server"))
```

provides the `serve` function, which starts the server with more configuration options.

`serve`

```
configuration [natural-number (union string #f)] -> (-> void)
(define (serve configuration . port ip-address) ...)
```

The `serve` function starts the Web server just like the launcher does, but the `configuration` argument supplies the server's settings. The optional `port` argument overrides the port supplied by the configuration. The optional `ip-address` binds the server to that address.

The result of invoking `serve` is a function of no arguments that shuts down the server.

`serve`

```
(opt-> (configuration?) ((and/f number? integer? exact? positive?) (union string? false?) (make-mixin-contract frame%)) ((-> void?) (any? . -> . (union false? string?)) (any? . -> . (union false? string?)) (any? . -> . string?) (-> (union false? (is-a?/c frame%))) (-> (is-a?/c frame%))))
(define (serve configuration . frame) ...)
```

In addition, this `serve` function accepts another optional argument. This argument is mixed into the `frame%` class created by the internal browser. The frame is then instantiated with one argument: the URL to visit.

11.3 Constructing Configurations

Constructing configurations requires another library.

```
(require (lib "configuration.ss" "web-server"))
```

`load-configuration`

```
string -> configuration
(define (load-configuration path) ...)
```

This function accepts a `path` to a configuration file and returns a configuration that `serve` accepts. The configuration tool can create configuration files, as explained in section 3.2. Configuration files can also be created by hand, as described in section 3.3.

11.4 Monitoring the Server

Requiring the library **monitor-web-server.ss**, via

```
(require (lib "monitor-web-server.ss" "web-server"))
```

provides the functions:

`poke-web-server`

```
channel string nat nat -> result
(define (poke-web-server channel server port timeout-seconds) ...)
```

The `poke-web-server` procedure takes a channel on which the results will come in, a server name to test, a port on that server to test, and a timeout (in seconds). The procedure returns immediately. The channel passed to the procedure will receive at least one result within (`timeout-seconds`$+\varepsilon$), for some reasonable ε.

A result is one of:

- `'(fail ,server-name ,server-port ,msg)`
- `'(timeout ,server-name ,server-port ,timeout-seconds)`
- `'(exn ,server-name ,server-port ,exn)`
- `'(ok)`

where `server-name` and `msg` are strings, `server-port` and `timeout-seconds` are numbers, and `exn` is an exception.

`result-message`

```
result -> string
(define (result-message result) ...)
```

The `result-message` procedure takes a result and produces a multi-line string suitable for inclusion in an error log or an email message.

11.4.1 Example

Here's a simple piece of code which checks the PLT download server:

```
(require (lib "monitor-web-server.ss" "web-server"))
(let ([result-channel (make-channel)]
      [server-name "download.plt-scheme.org"]
      [server-port 80])
  (poke-web-server result-channel server-name server-port 10)
  (let ([result (channel-get result-channel)])
    (match result
      ['(ok) (void)]
      [else (printf (result-message result))])))
```

To have this done automatically, see section 6 or Monit.

Index

.., 14

adjust-timeout!, 20
authentication-message, 7

`byte-environment`, 17

collect-garbage, 7
configuration-root, 7
configuration-table, 10

default-host-table, 5
default-indices, 6
default-servlet-timeout, 7

`environment`, 17
exists-binding?, 20
extract-binding/single, 20
extract-bindings, 20
extract-user-pass, 20

file-base-connection-timeout, 7
file-not-found-message, 7
file-per-byte-connection-timeout, 7
file-root, 8

gen, 19

host-root, 7

initial-connection-timeout, 5
interface-version, 15

load-configuration, 26
log-file-path, 7
log-format, 6

make-config, 25
make-html-response/incremental, 21
make-response/full, 18
make-response/incremental, 18
max-waiting, 5

name, 9

password, 9
password-authentication, 8
password-connection-timeout, 7
passwords, 9
passwords-refreshed, 7
path-regexp, 9

poke-web-server, 27
port, 5
protocol-message, 7

realm, 9
redirect-to, 21
refresh-passwords, 9
refresh-servlets, 13
report-errors-to-browser, 21
`request`, 17
request-bindings, 17
`response`, 18
result-message, 27

send/back, 19
send/finish, 19
send/forward, 19
send/suspend, 19
serve, 26
server-loop, 25
servlet-connection-timeout, 7
servlet-message, 7
servlet-root, 8
`servlet^`, 16
servlets-refreshed, 7
start, 16

timeout, 16

virtual host, 5
virtual-host-table, 5

web-server-monitor, 11