GNU Artanis

Mu Lei known as NalaGinrut

Table of Contents

1 Introduction

This manual documents GNU Artanis, a fast, monolithithic web framework of Scheme.

A web application framework (WAF) (http://en.wikipedia.org/wiki/Web_application_framework) is a software framework that is designed to support the development of dynamic websites, web applications, web services and web resources. This framework aims to alleviate the overhead associated with common activities in web development. GNU Artanis provides several tools for web development: database access, templating frameworks, session management, URL-remapping (http://en.wikipedia.org/wiki/Rewrite_engine) for RESTful (http://en.wikipedia.org/wiki/Representational_state_transfer), page caching, and more.

Guile is the GNU Ubiquitous Intelligent Language for Extensions, the official extension language for the GNU operating system (http://www.gnu.org/). Guile is also an interpreter and compiler for other dynamic programming languages including Scheme.

Scheme (http://en.wikipedia.org/wiki/Scheme_%28programming_language%29) is a functional programming language and one of the two main dialects of the programming language Lisp (http://en.wikipedia.org/wiki/Lisp_(programming_language)). Scheme follows a minimalist design philosophy specifying a small standard core with powerful tools for language extension.

1.1 Conventions used in this manual

In this manual the following syntax is used to demonstrate the use of the API:

```
(api-name arg1 arg2 #:key0 val0 ... [optional-arg1 <- default-value1] ...)
```

If you are new to Scheme, it is recommended that you read the Chapter 6 [Basic in Scheme], page 14, chapter first.

1.2 No warranty

We distribute software in the hope that it will be useful, but without any warranty. No author or distributor of this software accepts responsibility to anyone for the consequences of using it or for whether it serves any particular purpose or works at all, unless they say so in writing. This is exactly the same warranty that proprietary software companies offer: none.

2 License

GNU Artanis is Free Software (`http://www.gnu.org/philosophy/free-sw.html`). GNU Artanis is under the terms of the GNU GPLv3+ and GNU LGPLv3+. See the files COPYING and COPYING.LESSER in toplevel of source code.

This manual is published under the terms of the GNU Free Documentation License (`http://www.gnu.org/copyleft/fdl.html`) 1.3 or later.

You must be aware there is no warranty whatsoever for GNU Artanis. This is described in full in the This is described in full in the licenses.

3 Installation

3.1 For users

Install GNU Guile-2.2.2 or higher version:

Since GNU Artanis-0.2, the GNU Guile-2.2+ is required because of the suspendable-ports (`https://www.gnu.org/software/guile/manual/html_node/Non_002dBlocking-I_002fO.html`), which is the key to implement asynchronous non-blocking server core in GNU Guile.

```
wget -c ftp://ftp.gnu.org/gnu/guile/guile-2.2.2.tar.xz
tar xvf guile-2.2.2.tar.xz
cd guile-2.2.2 && ./configure && make #(NOTE: this may take very long time even looks
sudo make install
```

I would NOT recommend you trying to compile/install Guile from Git repo, since it'll take too much time of you.

Install dependencies:

- guile-dbi-2.1.6 [**Optional**]

```
wget -c http://download.gna.org/guile-dbi/guile-dbi-2.1.6.tar.gz
# or mirror
wget -c https://github.com/yagelix/guile-dbi/archive/guile-dbi-2.1.6.tar.gz

tar xvzf guile-dbi-2.1.6.tar.gz
cd guile-dbi-2.1.6 && ./configure && make
sudo make install
```

- guile-dbd [**Optional**]. The dbd plugins connect to an actual database server.

```
wget -c http://download.gna.org/guile-dbi/guile-dbd-mysql-2.1.6.tar.gz
# or mirror
wget -c https://github.com/yagelix/guile-dbi/archive/guile-dbd-mysql-2.1.6.tar.gz

tar xvzf guile-dbd-mysql-2.1.6.tar.gz
cd guile-dbd-mysql-2.1.6 && ./configure && make
sudo make install
```

MySQL is used for the examples in this manual. You may find dbd plugins for other databases at here (`http://download.gna.org/guile-dbi`) or mirror (`https://github.com/yagelix/guile-dbi/releases`). The installation process is identical.

Install the latest GNU Artanis:

```
wget -c http://ftp.gnu.org/gnu/artanis/artanis-latest.tar.bz2
tar xvjf artanis-latest.tar.bz2
cd artanis-latest && ./configure && make
sudo make install
```

3.2 For contributors

First of all, thank you for contributing! You may clone the main git repository, or the mirror on GitLab:

```
git clone git://git.savannah.gnu.org/artanis.git

# mirror on GitLab
git clone https://gitlab.com/NalaGinrut/artanis.git
```

4 Configuration

A configuration file is required when Artanis is run for the first time.

- If you're using minimum mode, say, all code are in a script file without application folder. The configure file must be named `/etc/artanis/artanis.conf`.
- If you're using application folder, the configure file `conf/artanis.conf` will be generated automatically for you.

4.1 Database config

`db.enable = <boolean>`

- Whether to use database, if disabled, the database won't be initialized in the beginning, which saves memory and boot time.
  - Some users may want to use GNU Artanis without configuring any databases, so please set it to **false** to avoid error.

`db.dbd = mysql | postgresql | sqlite3`

- What database server should be used, depends on the database installed on your machine.
  - NOTE: If you use MariaDB then you should set it to mysql as well.

`db.proto = tcp | socketfile`

- The protocol for connecting the databse. If you use tcp then a socket port must be specified in the address, and if you choose socketfile, then you should specify the unix socket file which has been configured by the databases.

`db.addr = <string>`

- The address of the database server, for example, in default MariaDB, the address should be *localhost:3306*.

`db.socketfile = <string>`

- If you configured the database server to be connected with an unix socket file, then you should fill this field with the file name.

`db.username = <string>`

- User name of the database server.

`db.passwd = <string>`

- Password of the database server.

`db.name = <string>`

- The database name of the database server.

`db.engine = <string>`

- The engine of the database server.
  - *NOTE:* for sqlite3, you have to set it to nothing, say `db.engine =` . If you remove this item at all, it'll be *InnoDB* in default!

4.2 Server config

`server.info = <string>`

- Specify your own server info, it'll be /*Artanis-x.x.x/* in default, depends on the version.

`server.nginx = enable | disable`

- If you used Nginx as the reversed-proxy, please enable it.

`server.charset = <string>`

- Charset in server side. **utf-8** in default.
  - **Note:** Don't change it unless you know what you're doing!

`server.syspage.path = /etc/artanis/pages`

- The path of status page. You may customize your own status pages.

`server.backlog = <integer>`

- Backlog of the socket.
  - **Note:** Don't change it unless you really know what you're doing!

`server.wqlen = <integer>`

- The length of the work queue in Artanis server.
  - **Note:** Added since Artanis-0.2.

`server.trigger = edge | level`

- The trigger mode of epoll.
  - **Note:** Added since Artanis-0.2.

`server.engine = ragnarok | guile | fibers | <customized engine>`

- The server core which is used for holding high concurrent connections. Artanis has a strong server core named Chapter 23 [Ragnarok server core], page 48, which is based on delimited continuations (`https://en.wikipedia.org/wiki/Delimited_continuation`) to provide asynchronous non-blocking high concurrent serving.
  - You may choose guile inner server which is weak, but sometimes you may under an operating system lacking of key features to run Raganrok, for example, maybe there's no epoll in your Operating System, for example, GNU/Hurd.
  - You may choose fibers server implemented with threads and delimited continuations, which is preemptable by the timer you set. For more details please see Fiber Manual (`https://github.com/wingo/fibers/wiki/Manual`).
  - **Note:** Added since Artanis-0.2.
  - **Note:** Fibers is supported since Artanis-0.2.5.

`server.polltimeout = <integer>`

- The the timeout for each event polling round, in miliseconds.
  - The default value is 500 miliseconds.
  - **Note:** Added since Artanis-0.2.

`server.bufsize = <integer>`

- The buffer size of the connecting socket. In Chapter 23 [Ragnarok server core], page 48, the request handling will be scheduled when the socket buffer is full. This item effects

the performance of socket I/O largely. Usually, if you're handling massive small requests, it's better to set the buffer size small; but if you're providing kind of downloading or uploading service, it's better to set it larger. But the large buffer size will increase the latency of unserved requests. Please read Chapter 23 [Ragnarok server core], page 48, chapter to learn the design principle, which will be helpful for you to decide how to tweak.

  - The default value is 12288, say, 12KB.
  - **Note:** Added since Artanis-0.2.

`server.multi = <boolean>`

- This is the most significant feature of Chapter 23 [Ragnarok server core], page 48. Please remember that **there's no any thread in GNU Artanis**. All the tasks are based on delimited continuations (`https://en.wikipedia.org/wiki/Delimited_continuation`), this kind of design is the so-called Green Threads. (`https://en.wikipedia.org/wiki/Green_threads`) Then how to take advantage of multi-cores? Fortunately, GNU/Linux has introduced a feature named SO_REUSEPORT (`https://lwn.net/Articles/542629/`) since 3.9. This feature let us start multiple Artanis instances listenning on the same socket port. When requests come, the Linux kernel will do necessary lock and allocation work for us to dispatch requests to these Artanis instances. This makes GNU Artanis provide performance and stateless perfectly.
  - The default value is true.
  - **Note:** Added since Artanis-0.2, GNU/Linux-3.9+ is required.

4.3 Websocket config

`websocket.maxpayload = <integer>`

- The max payload size in bytes.

`websocket.minpayload = <integer>`

- The min payload size in bytes.

`websocket.minpayload = <integer>`

- The min payload size in bytes.

`websocket.fragment = <integer>`

- If *fragment* is larger than zero, then it's the size of websocket frame fragment.
- If *fragment* is zero, then the websocket frame will not be fragmented.

4.4 Host config

`host.name = enable | disable | <boolean>`

- If disabled, you have to set the address to IP, say, `host.addr = 127.0.0.1`.

`host.addr = <URL> | <IP>`

- The host address of the site.

`host.port = <integer>`

- The listenning port of your hosting site.

```
host.family = ipv4 | ipv6
```

- Specify the protocol family
  - Added since Artanis-0.2.

4.5 Session config

```
session.path = <PATH>
```

- Specify the session files path. It depends on the session engine.

```
session.engine = simple | db | file | <third-party-engine>
```

- Specify session engine.
  - **simple** uses hash table for memcache.
  - **db** uses RDBMS for storing sessions.
  - **file** stores session information into text files.

4.6 Upload config

```
upload.types = <item-list>
```

- Specify allowed upload file type, say, `upload.types = jpg,png,gif`.
  - **Note:** Added since Artanis-0.2.

```
upload.path = <PATH>
```

- The path to put the uploaded files.

```
upload.size = <interger>
```

- The size limitation of uploaded file in bytes.
  - **Note:** Added since Artanis-0.2

4.7 Cache config

```
cache.maxage = <integer>
```

- The maximum age of cached page in seconds.
  - This is the global maxage of any cache. If you want to specify maxage for certain page, please read Chapter 20 [Cache], page 41.

4.8 Debug config

```
debug.enable = <boolean>
```

- Wheather to enable debug mode. If you enable debug mode, Artanis will print debug information verbosely. The module you modified will be reloaded instantly, and the page view will be rendered either.
- *NOTE: This option will drag the performance of Artanis, so please use it for debug only.*

```
debug.monitor = <PATHs>
```

- The paths need to be monitored in debug-mode. This will take advantage of 'inotify' in GNU/Linux kernel.
  - **Note:** We may support GNU/Hurd as well, with its file monitor mechanism, in the future.

4.9 Config APIs

To change the default configurations:

```
(conf-set! key value)
;;e.g
(conf-set! 'debug-mode #t)
```

To get the current configre

```
(get-conf key)
;;e.g
(get-conf '(server charset))
```

To get current hostname in GNU Artanis environment.

```
(current-myhost)
```

5 Hello World

5.1 Use Guile REPL and verify GNU Artanis installation

If you are already familiar with Guile, you may skip this section.

Type 'guile' in your console to enter the Guile REPL. You should see the following text displayed on your screen:

```
GNU Guile 2.2.2
Copyright (C) 1995-2017 Free Software Foundation, Inc.

Guile comes with ABSOLUTELY NO WARRANTY; for details type ‘,show w’.
This program is free software, and you are welcome to redistribute it
under certain conditions; type ‘,show c’ for details.

Enter ‘,help’ for help.
scheme@(guile-user)>
```

Welcome to Guile world! We are now going to play with GNU Artanis. Before we start, we need to check that GNU Artanis is installed correctly:

(Just type them, you don't have to understand them at present)

```
,use (artanis artanis)
artanis-version
```

The expected output should be similar to this:

```
$1 = "GNU Artanis-x.x.x"
```

5.2 Simple HTTP server

Run this code in your console:

```
guile -c "(use-modules (artanis artanis))(init-server)(run)"
## You'll see this screen:
Anytime you want to quit just try Ctrl+C, thanks!
http://127.0.0.1:3000
```

Assuming there's a file named "index.html" in the current path. Now you may try `http://localhost:3000/index.html` in your browser. It's just simply fetching static file by the URL: `http://localhost:3000/path/filename`

5.3 Try simple URL remappinge

Type these code in Guile REPL:

```
(use-modules (artanis artanis))
(get "/hello" (lambda () "hello world"))
(run #:port 8080)
```

Now you can visit `http://localhost:8080/hello` with your browser, and (hopefully) see the result.

If you encounter "[EXCEPTION] /favicon.ico is abnormal request" , please just ignore that warning.

Let me explain the code:

- *line 1:* Load GNU Artanis module, (artanis artanis) is the name.
- *line 2:* The first argument *get* is GNU Artanis' API correspondence to the GET method of the HTTP protocol. The second argument "/hello" is the URL rule to register showing in the address line of e.g. firefox. The third argument is the handler which will be triggered if the registered URL rule is hit.
- *line 3:* Run the GNU Artanis web server, and listen on socket port 8080.

You may type Ctrl+C to quit and stop the server, see also the message printed on the screen accordingly.

5.4 More complex URL remapping

Try this code:

```
(use-modules (artanis artanis))
(init-server)
(get "/hello/:who"
  (lambda (rc)
    (format #f "<p>hello ~a</p> " (params rc "who"))))
(run #:port 8080)
```

Now you can try `http://localhost:8080/hello/artanis` in your browser.

There are two differences:

- 1. The special rule, `"/hello/:who"`, *:who* means you can use *params* to reference the value of this section of URL with the key "who". `(params rc "who")` is the way for that.
- 2. You may have noticed that the handler is being defined as an anonymous function with *lambda* has one argument *rc*. It means *route context* which preserves all the related context information. Many GNU Artanis APIs need it, e.g. *params*.

And *format* is a Scheme lib function. It is similar to *sprintf* in the C language, which outputs text with a formatted pattern. The second argument #f (means FALSE) indicates that returning the result as string type rather than printing out.

5.5 Regex in URL remapping

You can use regex in the URL rule.

```
(use-modules (artanis artanis))
(init-server)
(get "/.+\\.(png|gif|jpeg)" static-page-emitter)
(run #:port 8080)
```

static-page-emitter is an GNU Artanis API which emits a static file (images, data files) to the client.

5.6 Database operating

GNU Artanis supports mysql/postgresql/sqlite3, we use mysql as a example here.

Please ensure that your DB service was started before you try.

If you encounter any problems, please check your config of DB first.

You can use a DB (such as mysql) with GUI tools such as "adminer" prior and independent of running an web-server, e.g. artanis-based.

```
(use-modules (artanis artanis))
(init-server)
(define conn (connect-db 'mysql #:db-username "your_db_username"
                         #:db-name "your_db_name" #:db-passwd "your_passwd"))
(define mtable (map-table-from-DB conn))
((mtable 'create 'Persons '((name varchar 10)
                            (age integer)
                            (email varchar 20)))
 'valid?)
;; ==> #t
(mtable 'set 'Persons #:name "nala" #:age 99 #:email "nala@artanis.com")
(mtable 'get 'Persons #:columns '(name email))
;; ==> ((("name" . "nala") ("email" . "nala@artanis.com")))
```

- *map-table-from-DB* is GNU Artanis API handling tables in DB. Here, we define this mapping as the var *mtable*.
- And we can use *mtable* to handle tables, you can get values from table with 'get command.
- *mtable* is a function which accepts the first argument as a command, say 'create is a command to create a new table; 'set command is used to insert/update the table; 'get command for fetch the values of specified columns.
- The second argument of *mtable* is the name of the table as you guess. Please note that it is case sensitive. But the column name could be case insensitive.
- **'create** command returns a function too, which also accepts an argument as a command. Here, we use **'valid?** command to check if the table has been created successfully.

Here's just simple introduction. You may read the DB section in this manual for detail describing.

Of course, you can use DB in your web application.

```
(get "/dbtest" #:conn #t ; apply for a DB connection from pool
     (lambda (rc)
       (let ((mtable (map-table-from-DB (:conn rc))))
         (object->string
          (mtable 'get 'Persons #:columns '(name email))))))

(run #:use-db? #t #:dbd 'mysql #:db-username "your_db_username"
     #:db-name "your_db_name" #:db-passwd "your_passwd" #:port 8080)
```

Now, try loading `http://localhost:8080/dbtest` in your browser.

Here are some explanations:

- The keyword-value pair `#:conn #t` means applying for a DB connection from connection-pool. Then you can use `(:conn rc)` to get the allocated connection for DB operations.
- Finally, the handler needs to return a string as the HTTP response body, so we have to use Guile API *object->string* to convert the query result to string, for this naive example case.

Exercise: Return a beautiful table in HTML rather than using object->string.

6 Basic in Scheme

This chapter introduces some useful documents to help you understand Scheme language well. Feel free to come back here if you have any problem with Scheme syntax.

If expedient, read the section repeatedly.

Scheme was introduced in 1975 by Gerald J. Sussman and Guy L. Steele Jr. and was the first dialect of Lisp to fully support lexical scoping, first-class procedures, and continuations. In its earliest form it was a small language intended primarily for research and teaching, supporting only a handful of predefined syntactic forms and procedures. Scheme is now a complete general-purpose programming language, though it still derives its power from a small set of key concepts. Early implementations of the language were interpreter-based and slow, but Guile Scheme is trying to implement sophisticated compiler that generate better optimized code, and even a plan for AOT compiler generated native code in the future.

6.1 For newbies

If you're not familiar with Scheme and Guile in particular, here is a simple tutorial for you.

If you already know the basics of the Scheme language, please feel free to skip this section.

I would recommend newbies to type/paste the code in Guile REPL following the guide in tutorial: Learn Scheme in 15 minutes (`http://web-artanis.com/scheme.html`)

And here's a nice section in Guile manual for basics in Scheme: Hello Scheme (`https://www.gnu.org/software/guile/manual/guile.html#Hello-Scheme_0021`)

Please don't spend too much time on these tutorials, the purpose is to let newbies get a little familiar with the grammar of Scheme.

6.2 For Pythonistas

These are good pythonic articles for Pythoners:

1. Guile basics from the perspective of a Pythonista (`http://draketo.de/proj/guile-basics/`)
2. Going from Python to Guile Scheme (`http://draketo.de/proj/py2guile`)

Still, please don't spend too much time on them, the purpose is to let newbies get a little familiar with the grammar of Scheme.

6.3 For Rubyist

Here's a great article for Rubyist to learn Scheme:

1. Scheme for ruby programmers (`http://wiki.call-cc.org/chicken-for-ruby-programmers`)

6.4 For deep learners

These two books are very good for learning Scheme seriously:

1. The Scheme Programming Language (`http://www.scheme.com/tspl4/`)
2. Structure and Interpretation of Computer Programs(SICP) (`http://mitpress.mit.edu/sicp/`)

Please don't bother reading them if you simply want to use GNU Artanis to build your web application/site in few minutes.

And if you really want to try to work these books seriously, please ignore GNU Artanis before you are done with them.

But once you're done reading them **carefully**, you may want to write a new GNU Artanis all by yourself!

Hold your horses. ;-)

7 Basic in GNU Artanis

7.1 How to run a site with GNU Artanis

This is the simplest case to run a site:

```
#!/bin/env guile
!#
(use-modules (artanis artanis))
(init-server)
(get "/hello" (lambda () "hello world"))
(run)
```

7.2 Initialization

It's better to use (init-server) to init GNU Artanis.

```
(init-server #:statics '(png jpg jpeg ico html js css)
             #:cache-statics? #f #:exclude '())
```

`#:statics` specifies the static files with the extension file. GNU Artanis is based on URL remapping, so this keyword let you avoid to handle each static file types. In default, it covers the most static file types. So you may ignore it usually.

`#:cache-statics?` indicates if the static files should be cached.

`#:exclude` specifies the types should be excluded. This is useful when you want to generate image files dynamically. Even JavaScript/CSS could be generated dynamically, depends your design.

7.3 Registering handler of HTTP methods

Please read Section 9.2 [URL handling], page 20.

7.4 Emit Response

```
(response-emit body #:status 200 #:headers '() #:mtime (current-time))
```

body is the response body, it can be bytevector or literal string (in HTML).

`#:status` is HTTP status, 200 in default, which means OK.

`#:headers` let you specify customized HTTP headers. The headers must follow certain format, you have to read about the Response Headers (`http://www.gnu.org/software/guile/manual/html_node/HTTP-Headers.html#Response-Headers`).

`#:mtime` specifies the modify time in the response. GNU Artanis will generate it for you if you just ignore it.

```
(emit-response-with-file filename [headers <- '()])
```

filename is the filename to be sent as a response.

[headers] is the customized HTTP headers.

7.5 Running server

```
(run #:host #f #:port #f #:debug #f #:use-db? #f
     #:dbd #f #:db-username #f #:db-passwd #f #:db-name #f)
```

You may see all the keyword is #f in default, this means these items will be gotten from config file.

But you can specify them as will.

`#:host` specify the hostname.

`#:port` specify the socket port of the server.

`#:debug` set #t if you want to enable debug mode. Maybe verbose.

`#:use-db?` set #t if you want to use DB, and GNU Artanis will init DB config for you.

`#:dbd` choose dbd, there're three supported dbd: mysql, postgresql, and sqlite3.

`#:db-username` specify the username of your DB server.

`#:db-passwd` the DB password.

`#:db-name` specify DB name.

7.6 Working with Nginx

You may try GNU Artanis+Nginx with so-called reverse proxy.

Although GNU Artanis has good server core, I would recommend you use Nginx as the front server. In addition to the performance, GNU Artanis hasn't prepared for many security things. But if you use Ngxin with reverse-proxy, then it'll be easier to be safer.

For example, you may add these lines to your /etc/nginx/nginx.conf:

```
location / {
proxy_pass http://127.0.0.1:1234;
proxy_set_header Host $host;
proxy_set_header X-Real-IP $remote_addr;
proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
}
```

Then restart you Nginx:

```
sudo service nginx restart
```

And run GNU Artanis:

```
(run #:port 1234)
```

8 The Art command line

GNU Artanis provides **art** command line tool to save users' time.

8.1 art create

If you want to set up your site/app within an application folder, and take advantage of MVC, you have to use this command to create the application folder first.

```
art create proj_path
```

8.2 art draw

This command will generate the specified component:

```
Usage:
  art draw <component> NAME [options]

component list:
  model
  controller
  migration

Options:
  -h, [--help]     # Print this screen
  -d, [--dry]      # Dry run but do not make any changes
  -f, [--force]    # Overwrite files that already exist
  -s, [--skip]     # Skip files that already exist
                   # If -s and -f are both provided, -f will be enabled
  -q, [--quiet]    # Suppress status output

Example:
  art draw model myblog
```

Please see Chapter 11 [MVC], page 23, to learn more about how to use these components.

8.3 art migrate

Migrate is used for Database migration.

```
Usage:
  art migrate operator name [OPTIONS]

Operators:
  up
  down

OPTIONS:
  VERSION=version
```

Please see Section 14.2 [Migration], page 28, for more detail.

8.4 art work

This command is used to start the server to run your site in the application folder:

```
Usage:
  art work [options]

Options:
  -c, [--config=CONFIG]          # Specify config file
                                   Default: conf/artanis.conf
                                            if no, /etc/artanis/artanis.conf
  -h, [--host=HOST]              # Specify the network host
                                   Default: 0.0.0.0
  -d, [--usedb]                  # Whether to use Database
                                   Default: false
  -b, [--dbd=DBD]                # Specify DBD, mysql/postgresql/sqlit3
                                   Default: mysql
  -n, [--name=DATABASE_NAME]     # Database name
                                   Default: artanis
  -w, [--passwd=PASSWD]          # Database password
                                   Default: none
  -u, [--user=USER]              # Database user name
                                   Default: root
  -p, [--port=PORT]              # Specify listening port
                                   Default: 3000
  -g, [--debug]                  # Debug mode
                                   Default: disable
  -s, [--server=SERVER]          # Specify server core
                                   Default: Ragnarok (New server core since 0.2)
  --help                         # Show this screen
```

- For server core alternatives, please see Section 4.2 [Server config], page 6.
- For Database (DBD) alternatives, please see Section 4.1 [Database config], page 5.

9 URL remapping

9.1 Introduction to URL remapping

URL remapping is used to modify a web URL's appearance to provide short, pretty or fancy, search engine friendly URLs. It's largely used in modern WAF(web application framework) to provide RESTful web APIs.

9.2 URL handling

According to RFC2616, there are GET, POST, PUT, PATCH and DELETE methods. You may register handler for specified URL rule to these methods.

There would be a HEAD method, but in GNU Artanis, the HEAD method is handled by the server, so users can not use it.

The usage:

```
(method rule handler)
```

And the handler could be one of two types, depending on your needs:

```
(lambda ()
  ...
  ret)

(lambda (rc)
  ...
  ret)
```

ret also has two types:

- 1. literal string as the returned response body
- 2. See Section 7.4 [Emit Response], page 16,

```
(get "/hello" (lambda () "hello world"))
```

For POST method:

```
(post "/auth" (lambda (rc) ...))
```

9.3 Get parameters from URL

```
(params rc name)
;; e.g
(get "/hello/:who" (lambda (rc) (params rc "who")))
```

9.4 Redirect link

```
(redirect-to rc path #:status 301
             #:scheme 'http)
;; e.g
(get "/aaa" (lambda (rc) (redirect-to rc "/bbb")))
(get "/bbb" (lambda () "ok bbb"))
```

10 Route context

Route context is a struct which encapsulated server necessary information from the current request context. We named it *route* because it's related to the route of Chapter 9 [URL remapping], page 20. Usually it's passed to the page handler as the unique argument, it's expected to provide sufficient information in the current request status.

```
(HTTP-METHOD URL-rule (lambda (<route-context>) ...))
;; e.g:
(get "/hello" (lambda (rc) "world")) ; rc is <route-context> type
```

10.1 Route context APIs

`(rc-path <route-context>)`

- Get the requested path, that is to say, the actual URI visited by the client.

```
;; e.g
(get "/hello/world" (lambda (rc) (rc-path rc)))
;; visit localhost:3000/hello/world or from any port you specified
;; the result is "/hello/world".
(get "/hello/:who" (lambda (rc) (rc-path rc)))
;; visit localhost:3000/hello/world or from any port you specified
;; the result is "/hello/world".
```

`(rc-req <route-context>)`

- Get the current HTTP request wrapped in record-type. About HTTP request please see HTTP Request (`https://www.gnu.org/software/guile/manual/html_node/Requests.html`). It stores HTTP request of Guile.

`(rc-body <route-context>)`

- Get the current request body:
  - For a regular HTTP request, the body should be a bytevector;
  - For a Websocket request, the body should be Section 22.4 [Websocket frame], page 46, as a record-type.

`(rc-method <route-context>)`

- Get the current requested HTTP method.

`(rc-conn <route-context>)`

- Get the current DB connection if you've requested one, please checkout [BROKEN LINK: nil].

`(rc-qt <route-context>)`

- Get query table, which is a key-value list parsed from Chapter 12 [Query String], page 24.

`(rc-handler <route-context>)`

- Get the current request handler. The tricky part is that you can only get this handler within this handler unless you can go no where to run *rc-handler* correctly.
  - It's on your own risk to use this API. But now that we have powerful first class lambda, you may do some magic. Well, depends on you.

```
(rc-mtime <route-context>) ; getter
(rc-mtime! <route-context>) ; setter
```

- You may set it in the handler to return you customized modified time. For static pages, the mtime is set automatically. But sometimes people may want to set it in a dynamic generated page.

  ```
  (rc-cookie <route-context>)
  ```

- The cookies parsed from request header.

  ```
  (rc-set-cookie! <route-context>)
  ```

- Set response cookie from server side. If you want to return cookies to the client, please use it.

There're other APIs in *route-context*, but they're largely used for internals of Artanis, rarely useful for users. So we don't list them here.

11 MVC

MVC is Model-Views-Controller, the most classic architectural pattern for implementing user interfaces. It divides a given software application into three interconnected parts, so as to separate internal representations of information from the ways that information is presented to or accepted from the user.

11.1 Controllers/Views

When you run it to generate a controller named *article*:

```
art draw controller article show edit
```

show and *edit* are the name of methods for the controller named *article*.

And it'll generate both **controller** and **view** for *article*:

```
drawing    controller article
working    Controllers 'article.scm'
create     app/controllers/article.scm
working    Views 'article'
create     app/views/article/show.html.tpl
create     app/views/article/edit.html.tpl
```

As you may see, there're three files were generated:

```
app/controllers/article.scm
app/views/article/show.html.tpl
app/views/article/edit.html.tpl
```

This means the controller *article* has two methods mapped to URL rule named *show* and *edit*. And *view* component will generate HTML template for each method, say, **show.html.tpl**. For example, the controller *article* generate *show* method handler automatically:

```
(article-define show
                (lambda (rc)
                  "<h1>This is article#show</h1><p>Find me in app/views/article/show.h
                  ;; TODO: add controller method 'show'
                  ;; uncomment this line if you want to render view from template
                  ;; (view-render "show")
                  ))
```

Of course, it depends on you whether to use these template. If you want to use *view template*, just uncomment the last line `(view-render "show")`.

For more detail about template in Views, please see Chapter 13 [Layouts and Rendering in GNU Artanis], page 25.

11.2 Models

Models contains operations of database.

For modifying tables, you should read Section 14.2 [Migration], page 28.

For other DB operation, please read Section 14.5 [FPRM (experimental)], page 30.

(To be continue. . .)

12 Query String

Query string is a special form of URL:

```
http://example.com/over/there?name=ferret&color=purple
```

It's useful to pass parameters to the server side.

GNU Artanis provides convenient API to handle query strings.

12.1 Query string from GET

The query string would be encoded in URL if the method is GET.

```
http://example.com/over/there?name=ferret&color=purple
```

Please notice that URL-remapping support regex, so you should register URL rule like this:

```
(get "/there?"
  (lambda (rc)
    (get-from-qstr rc "name")))
```

Or it will throw 404 since URL-remapping failed to hit the rule with query string.

12.2 Query string from POST

The query string would be encoded in HTTP body if the method were POST.

There's only slitely difference when you pass query string by POST: you don't have to use regex, for example, "?" for matching the URL.

```
(post "/there"
  (lambda (rc)
    (get-from-qstr rc "name")))
```

GNU Artanis will detect the method type in *get-from-qstr*, if it's POST, then the router will parse query string from the HTTP body automatically.

13 Layouts and Rendering in GNU Artanis

13.1 Templating

Templating provides a way to mix programming code into HTML.

13.2 Templating for Pythoners

If you're familiar with Django, which implemented a DSL(Domain Specific Language) to express presentation rather than program logic. You may realize that the templating of GNU Artanis has different philosophy.

In templating of GNU Artanis, it's simply embedded Scheme code into HTML. Why? Because of the philosophy of FP(Functional Programming), everything could be a function. So obviously, `(filesizeformat size)` is enough for understanding, and it's just simple function calling in prefix-notation. There's no need to implement DSL like `size|filesizeformat` to increase the complexity of code. Let alone the syntax is very different from Python.

The syntax like `size | filesizeformat` is postfix-notation used in stack-based languages, say Forth. Such a language used to delegate another programming paradigm named concatenative programming. It's very different from the paradigm of Scheme (functional programming), and the paradigm of Python (imperative programming).

The philosophy of GNU Artanis templating is to bring it into correspondence with the paradigm of the language. And reduce the unnecessary complexities. KISS (`http://en.wikipedia.org/wiki/KISS_principle`).

13.3 Templating for Rubyists

Templating in GNU Artanis looks very similar to Rails.

The Rails code:

```
<% if( @fullscreen == 1 ) %>
<%= "<div class='full'><p>...</p></div>" %>
<% end %>
```

And the same function in GNU Artanis code:

```
<% (if (= fullscreen 1) %>
       <% "<div class='full'><p>...</p></div>" %>
       <% ) %>
```

13.4 Templating APIs

```
(tpl->response filename/sxml [environment <- (the-environment)] [escape? <- #f])

(tpl->html filename/sxm [environment <- (the-environment)] [escape? <- #f])
```

The difference is that tpl->html returns a string, but tpl->response will return HTTP response.

[environment] is the environment you want to pass in. We often ignore it. But if you want to ref some vars defined outside your template string, you should pass (the-environment).

[escape?] If you want to HTML char-escaping with the returned string, set it to #t.

There're two kinds of different templating:

13.5 Embedded Templating

Example: Write a tpl file named "my.tpl":

```
<html>
  <p> <%= "This is tpl test!" %> </p>
  <p> <% (format #t "And this is ~a" (getcwd)) %> </p>
  <p> <%= external-var %> </p>
</html>
```

The filename extension ".tpl" is NOT trivial, since the MVC will find the template by detecting controller name automatically. But if you don't use MVC, say, you just write a simple .scm file for loading GNU Artanis modules. then the extension filename ".tpl" is trivial.

```
(get "/test"
  (lambda (rc)
    (let ((external-var 123))
      (tpl->response "my.tpl" (the-environment)))))
(run #:port 8080)
```

In this case, make sure to put my.tpl to the same path with your GNU Artanis code.

Because **external-var** is defined outside the file "my.tpl", and it's bound in *let* with 123, you have to pass (the-environment). Or the template render will blame that it can't find variable named **external-var**.

If you don't have any external var needs to be referenced, just use `(tpl->response "file.tpl")` is fine.

Then see `http://localhost:3000/test` in your browser.

13.6 Template special commands

GNU Artanis provide special helper commands.

Please notice that GNU Artanis constrains the path of sources in application folder for security reasons. The resources files, CSS, JS etc, should be put int **pub** directory in the application folder, or the client can't access them.

Those special commands are useful to expand the path for you, and they should be added into the tamplate file, for example:

```
<html>
  <head>
    <@icon favicon.ico %>
      <@js functions.js %>
        <@css blog.css %>
  </head>

  <@include sidebar.html %>
```

```
    <body>
      ...
    </body>
  </html>
```

NOTE: The command name is prefixed with **@**, say, **@include**, **@css**, etc. Please do not seperate the **@**, or it will throw excepton.

For example, you may include html files with **include** command:

```
;; @include is the command name, not <@ include filename %>
<@include filename.html %>
```

This will be expanded like this:

```
/current_toplevel/pub/filename.html
```

NOTE: Please make sure the included file is put to **pub** directory in the application folder.

To reference CSS file:

```
<@css filename.css %>
```

This will be expanded like this:

```
<link rel="stylesheet" href="/pub/css/filename.css">
```

To reference JS file in the HTML head:

```
<@js filename.js %>
```

This will be expanded like this:

```
<script type="text/javascript" src="/pub/js/filename.js"> </script>
```

To specify the icon:

```
<@icon favicon.ico %>
```

This will be expanded like this:

```
<link rel="icon" href="/pub/img/favicon.ico" type="image/x-icon">
```

13.7 SXML Templating

SXML (`http://en.wikipedia.org/wiki/SXML`) is an alternative syntax for writing XML data, using the form of S-expressions.

SXML is to Scheme as JSON is to ECMAScript(the so-called Javascript). Maybe this explains clearer.

The benefit of SXML is to take advantage of quasiquote in Scheme. If you no little about it, then you may google "scheme quasiquote" for more details.

```
(tpl->response '(html (body (p (@ (id "content")) "hello world"))))
```

You would get a html string:

```
<html><body><p id="content">hello world</p></body></html>
```

Sometimes you may need quasiquote to reference a variable, for example:

```
(let ((content "hello world"))
  (tpl->response `(html (body (p (@ (id "content")) ,content)))))
```

14 Database

14.1 DB connection pool

TODO

14.2 Migration

Migration provides a way do complicated modification of tables in database automatically. Here's an example.

First, draw a migration:

```
# art draw migration person
drawing    migration person
working    Migration '20151107040209_person.scm'
```

You'll see something similar like above.

In this case, you may edit file db/migration/20151107040209_person.scm:

```
(migrate-up
 (create-table
  'person
  '(id auto (#:primary-key))
  '(name char-field (#:not-null #:maxlen 10))
  '(age tiny-integer (#:not-null))
  '(email char-field (#:maxlen 20))))

(migrate-down
 (drop-table 'person))
```

Now you may run **up** command of migration:

```
art migrate up person
```

Then migrate-up function will be called, and this will create a table named *person*:

```
+-------+---------------------+------+-----+---------+----------------+
| Field | Type                | Null | Key | Default | Extra          |
+-------+---------------------+------+-----+---------+----------------+
| id    | bigint(20) unsigned | NO   | PRI | NULL    | auto_increment |
| name  | varchar(10)         | NO   |     | NULL    |                |
| age   | tinyint(4)          | NO   |     | NULL    |                |
| email | varchar(20)         | YES  |     | NULL    |                |
+-------+---------------------+------+-----+---------+----------------+
```

If you run **down** command of migration:

```
art migrate down person
```

Obviously, the table *person* will be dropped.

14.3 ORM problem

ORM stands for Object Relational Mapping, which is a popular approach to handle relational DB nowadays, in OOP.

Of course, Guile has it's own Object System named GOOPS (`https://www.gnu.org/software/guile/manual/html_node/GOOPS.html#GOOPS`). Users may use OOP with it. And it's possible to implement ORM in GNU Artanis as well.

However, FP fans realized that they don't have to use OOP if they can use FP features reasonably.

Besides, there're some criticism pointing to ORM:

- ORM Hate (`http://martinfowler.com/bliki/OrmHate.html`)
- Vietnam of Computer Science (`http://blogs.tedneward.com/2006/06/26/The+Vietnam+Of+Computer+Science.aspx`)
- Object-Relational Mapping is the Vietnam of Computer Science (`http://blog.codinghorror.com/object-relational-mapping-is-the-vietnam-of-computer-science/`)

And here're some known ways for trying to solve the problems of ORM:

- 1. *Give up ORM*.
- 2. *Give up relational storage model*. Don't use relational DB, pick up others, say, No-SQL. Well, this way is not cool when you have to use relational DB.
- 3. *Manual mapping*. Write SQL code directly. It's fine sometimes. But the code increases when things get complicated. Refactoring and reusing would be worth to consider.
- 4. *Limited ORM*. Limited the utility of ORM. And use ORM to solve part of your work rather than whole, depends on you. This may avoid some problems.
- 5. *SQL related DSL*. Design a new language. LINQ from Microsoft is one of the cases.
- 6. *Integration of relational concepts into frameworks*. Well, harder than 5, but worth to try.
- 7. *Stateless*. This is the critical hit to counter complexity and unreliability.

Basically, GNU Artanis has no ORM yet, and maybe never will. GNU Artanis is trying to experiment new ways to solve the problems of ORM.

GNU Artanis provides three ways to complete this mission. All of them, are **experimental** at present.

- SSQL (1,3,5)
- FPRM (4,7)
- SQL Mapping (1,3,6)

14.4 SSQL (experimental)

The concept of SSQL is very easy. Write SQL in s-expression (`https://en.wikipedia.org/wiki/S-expression`).

Usage:

```
(->sql sql-statement)
```

```
(where #:key val ... [literal string])
(having #:key val ... [literal string])
(/or conds ...)
(/and conds ...)
```

For example:

```
(->sql select * from 'Persons (where #:city "Shenzhen"))
(->sql select '(age name) from 'Persons (where "age < 30"))
```

The SQL update maybe quite different to SQL grammar, it should like blow.

```
(->sql update 'table set (list (list phone_number "13666666666")) (where #:name "john"
```

14.5 FPRM (experimental)

FPRM stands for Functional Programming Relational Mapping. It's a new word I invented. But it's not new concept. FP here indicates **stateless**.

FPRM is still experimental and work-in-progress.

14.5.1 Connect to DB server

```
;; usage 1:
(connect-db dbd init-str)

;; usage 2:
(connect-db dbd #:db-name "artanis" #:db-username "root" #:db-passwd ""
            #:proto "tcp" #:host "localhost" #:port 3306)
```

- **dbd** is a string, could be "mysql", "postgresql", and "sqlite3".
- **init-str** is a string for DB init, for example:

  ```
  (connect-db "mysql" "root:123:artanis:tcp:localhost:3306")
  ```

- `#:db-name` specifies the DB name.
- `#:db-username` specifies the DB username.
- `#:proto` specifies the socket protocol, which is related to DB server you chosen.
- `#:host` specifies the host name.
- `#:port` specifies the socket port.

14.5.2 Map DB table

This step will generate an new instance (as a closure) mapped to database table or view. In ORM, it is often called Active Record (`http://www.martinfowler.com/eaaCatalog/activeRecord.html`) which maps the database view to an class object.

And there're two differences:

- FPRM doesn't create object for each table. It maps a whole database in concept, and generates SQL for each table as you choose. So it maybe lightweight compared to an ORM object.
- FPRM doesn't maintain any states at all, say, it keeps stateless in the object (Not in database).

These two points may decrease the power of FPRM, but our main philosophy in GNU Artanis is that

- *The best way to control DB is SQL, don't bother with other guile schemes.*

That means we're not going to develop a complicated ORM in GNU Artanis, but a promising way to interact with SQL easily. This is what Section 14.6 [SQL Mapping (experimental)], page 32, provided. FPRM aims to reduce states & complexity to provide reliability, and SQL-Mapping will provide a convenient way to handle complex SQL for better performance and security (from SQL-Injection).

```
(define m (map-table-from-DB rc/conn))
```

rc/conn can be route-context or connection of DB.

map-table-from-DB returns a function, we named it **m** here for explaining.

14.5.3 Create table

```
(m 'create table-name defs #:if-exists? #f #:primary-keys '() #:engine #f)
```

- **table-name** specifies the name of the table in DB.
- **defs** is a list to define the columns' types. For example:

  ```
  '((name varchar 10) (age integer) (email varchar 20))
  ```

- `#:if-exists?` has two kinds of possible options:
  - '**overwrite** or '**drop** means overwriting the existed table if possible.
  - '**ignore** means ignore the table when there's an existed one.
- `#:primary-keys` specifies the primary keys in the created table.
- `#:engine` specifies the engine, depends on the dbd you chosen.

14.5.4 Get columns from table

```
(m 'get table-name #:columns '(*) #:functions '() #:ret 'all
   #:group-by #f #:order-by #f)
```

- `#:column` is the columns list you wanted.
- `#:functions` is built-in functions calling, e.g:

  ```
  #:functions '((count Persons.Lastname))
  ```

- `#:ret` specifies how to return the result, there're three options:
  - 'all for returning all results
  - 'top for returning the first result
  - integer (larger than 0), you specify the number.
- `#:group-by` used in conjunction with the aggregate functions to group the result-set by one or more columns.
- `#:order-by` used to sort the result-set by one or more columns.

For example, to get Lastname and City column, and return the first result.

```
(m 'get 'Persons #:columns '(Lastname City) #:ret 'top)
```

14.5.5 Set values to table

```
(m 'set table-name . kargs)
```

kargs is a var-list to accept the key-value arguments.

For example:

```
(m 'set 'Persons #:name "nala" #:age 99 #:email "nala@artanis.com")
```

14.5.6 Drop a table

```
(m 'drop table-name)
```

14.5.7 Check existence of table

```
;; case sensitive
(m 'exists? table-name . columns)
;; or for case-insensitive
(m 'ci-exists? table-name . columns)
```

For example:

```
(m 'exists? 'Persons 'city 'lastname)
```

14.5.8 Get schema of a table

```
(m 'schema table-name)
```

NOTE: all the returned name of schema will be down-cased.

14.6 SQL Mapping (experimental)

To be continued . . .

15 MIME

`#:mime` method is used to return the proper MIME type in the HTTP response.

```
#:mime type ; for registering type
(:mime rc body) ; for emit the reponse with the proper MIME
```

15.1 JSON

GNU Artanis integrated the third-party module guile-json (`https://github.com/aconchillo/guile-json`). You may use #:mime method to handle JSON:

```
(get "/json" #:mime 'json
     (lambda (rc)
       (let ((j (json (object ("name" "nala") ("age" 15)))))
         (:mime rc j))))
```

For example:

```
(define my-json
  (json (object ("name" "nala") ("age" 15)
                ("read_list"
                 (object
                  ("book1" "The interpreter and structure of Artanis")
                  ("book2" "The art of Artanis programming"))))))
(scm->json my-json) ; scm->json will print json
;; ==> {"name" : "nala",
;;      "age" : 15,
;;      "read_list" : {"book2" : "The art of Artanis programming",
;;                     "book1" : "The interpreter and structure of Artanis"}}
```

`scm->json` will print the result directly.

If you need to format JSON as a string to return to clients, please use `scm->json-string`.

15.2 CSV

GNU Artanis integrated the third-party module guile-csv (`https://github.com/NalaGinrut/guile-csv`). You may use #:mime method to handle CSV:

```
(get "/csv" #:mime 'csv
     (lambda (rc)
       (:mime rc '(("a" "1") ("b" "2")))))
```

15.3 XML

In Scheme, XML is handled with SXML. Another way is to use strings appending method.

```
(get "/xml" #:mime 'xml
     (lambda (rc)
       (:mime rc '(*TOP* (WEIGHT (@ (unit "pound"))
                                 (NET (@ (certified "certified")) "67")
                                 (GROSS "95"))))))
```

The rendered result to the client will be:

```
<WEIGHT unit="pound">
  <NET certified="certified">67</NET>
  <GROSS>95</GROSS>
</WEIGHT>
```

15.4 SXML

You can use SXML to replace XML for exchanging data format. This way saves some bandwidth.

```
(get "/sxml" #:mime 'sxml
     (lambda (rc)
       (:mime rc '((a 1) (b 2)))))
```

16 Upload files

If you want to deal with uploading files, store-uploaded-files would be you friend.

16.1 Receive upload from client

```
(store-uploaded-files rc #:path (current-upload-path)
                      #:uid #f
                      #:gid #f
                      #:simple-ret? #t
                      #:mode #o664
                      #:path-mode #o775
                      #:sync #f)
```

rc is the route-context.

`#:path` is specified path to put uploaded files.

`#:uid` is new UID for uploaded files, #f means don't change the default UID.

`#:gid` specifies new GID.

`#:simple-ret?` specifies the mode of return:

- if #t, there're only two possible return value, 'success for success, 'none for nothing has been done.
- if #f, and while it's successful, it returns a list to show more details: (success size-list filename-list).

`#:mode` chmod files to mode.

`#:path-mode` chmod upload path to mode.

`#:sync` sync while storing files.

16.2 Send upload to Server

Although GNU Artanis is often used in server-side, we provide this function for users to upload files from client.

```
(upload-files-to uri pattern)
```

uri is standard HTTP URL:

```
scheme://[user:password@]domain:port/path?query_string#fragment_id
```

pattern should be: ((file filelist ...) (data datalist ...)), for example:

```
(upload-files-to "ftp://nala:123@myupload.com/"
                 '((data ("data1" "hello world"))
                   (file ("file1" "filename") ("file2" "filename2"))))
```

17 Sessions

You have to use `#:session mode` while you defining URL rule handler.

```
(post "/auth" #:session mode
      (lambda (rc) ...))
```

mode could be:

- #t or 'spawn, to spawn a new session, the name of SID is "sid" in default.
- '(spawn ,sid) specify a name of sid to spawn.
- '(spawn ,sid ,proc) specify a name of sid and a proc to **define your own session spawner**.

And the APIs of session is :session

```
(:session rc cmd)
```

cmd could be:

- 'check to check session with name "sid".
- '(check ,sid) to check session with a specified sid name.
- 'check-and-spawn check "sid" first, if no, then spawn it.
- '(check-and-spawn ,sid) the same with above, but specified name of sid.
- '(check-and-spawn-and-keep ,sid) check then spawn then keep it, with the name of sid.
- 'spawn spawn a session with the name "sid".
- 'spawn-and-keep spawn a session then keep with the name "sid".

18 Cookies

You have to use `#:cookies mode` while you defining URL rule handler.

```
(get "/certain-rule" #:cookies mode
     (lambda (rc) ...))
```

mode could be:

- ('names names ...) specifies the name list of the cookies.
- ('custom (names ...) maker setter getter modifier) specify a more complicated customized cookie handlers.

And the APIs:

```
(:cookies-set! rc cookie-name key val)

(:cookies-ref rc cookie-name key)

(:cookies-setattr! rc cookie-name #:expir #f #:domain #f
                   #:path #f #:secure #f #:http-only #f)

(:cookies-remove! rc key) ; remove cookie from client

(:cookies-update! rc) ; cookies operations won't work unless you update it
```

NOTE: You don't have to call `:cookies-update!` yourself, since it will be called automatically by the hook before the response.

For example:

```
(get "/cookie" #:cookies '(names cc)
     (lambda (rc)
       (:cookies-set! rc 'cc "sid" "123321")
       "ok"))

(get "/cookie/:expires" #:cookies '(names cc)
     (lambda (rc)
       (:cookies-set! rc 'cc "sid" "123321")
       (:cookies-setattr! rc 'cc #:expir (string->number (params rc "expires")))
       "ok"))
```

Now you may use this command in the console to see the result:

```
curl --head localhost:3000/cookie
# and
curl --head localhost:3000/cookie/120
```

19 Authentication

19.1 Init Authentication

GNU Artanis provides flexible mechanism for authentication.

You have to use `#:auth mode` while you defining URL rule handler.

```
(get "/certain-rule" #:auth mode
     (lambda (rc) ...))
```

mode could be:

- SQL as Section 24.1 [String Template], page 50. You may write your own customized SQL for fetching & checking username and password.
- ('basic (lambda (rc user passwd) ...)) init a Basic Authentication mode. *user* is submitted username, *passwd* is submitted password value.
- ('table table-name username-field passwd-field) init a common Authentication mode. **The passwd will be encrypted by default algorithm**.
- ('table table-name username-field passwd-field crypto-proc) similar to the above item, but encrypt passwd with crypto-proc.
- (table-name crypto-proc), so passwd field will be "passwd" and username will be "username" in default, and you may encrypt passwd with crypto-proc.

Available crypto-proc helper functions listed here:

- (string->md5 str)
- (string->sha-1 str)

19.2 Basic Authentication

HTTP Basic authentication (BA) implementation is the simplest technique for enforcing access controls to web resources because it doesn't require cookies, session identifier and login pages. Rather, HTTP Basic authentication uses static, standard HTTP headers which means that no handshakes have to be done in anticipation.

The BA mechanism provides no confidentiality protection for the transmitted credentials. They are merely encoded with Base64 in transit, but not encrypted or hashed in any way. Basic Authentication is, therefore, typically used over HTTPS.

GNU Artanis doesn't support HTTPS at present, it is planned to support it in the future.

Let's see a simple example:

```
(define (my-checker rc user passwd)
  (and (string=? user "jack") (string=? passwd "123")))

(post "/bauth" #:auth `(basic ,checker)
  (lambda (rc)
    (if (:auth rc)
        "auth ok"
    (throw-auth-needed))))
```

Another simple way is to compare the passsword stored in the database table:

```
(post "/bauth" #:auth `(basic Person username passwd)
  (lambda (rc) ... ))
```

NOTE: Assuming **username** and **passwd** is the fields of Person table.

You have to define your own checker with the anonymous function `(lambda (rc u p) ...)`. #t for succeed, #f for failed.

APIs:

- `(:auth rc)` will check if Basic Authentication succeeded, #f for failed.
- `(throw-auth-needed)` is a useful helper function to ask for auth in client side.

19.3 Common Authentication

Actually, there are multiple authentication methods that can be used by developers. Most of them are sort of tricky hacks. Here we only introduce the most common way.

The most common and relative safe way for authentication is to use POST method. And check username and passwd from a table in DB.

There're several syntax sugar for authentication.

The simplest case is for Section 24.1 [String Template], page 50:

```
#:auth "string-template"
```

If you put the account information in a database table, then you may use table mode:

```
#:auth `(table ,table-name [,username-field] [,passwd-field] [,salt-field] [,hmac])
```

NOTE: The square-bracked **[item]** is optional.

The default values of optional items are:

- username-field: username
- passwd-field: passwd
- salt-field: salt

GNU Artanis requires the salted password, it's not optional.

So please prepare a field in the table for salt string. It's your duty to generate a salt string, please see Section 24.2 [Random String Generator], page 50. When authenticate, please specify the salt field name in salt-field.

For hmac item, please see Section 19.5 [HMAC], page 40.

19.4 Login authentication

Usually, in login case, you need both `#:auth` and `#:session` for long time session. The first step is to authenticate, if it's successful, then spawn a new session for this request. And the

Here is a simple example:

```
(post "/auth" #:auth '(table user "user" "passwd") #:session #t
      (lambda (rc)
        (cond
         ((:session rc 'check) "auth ok (session)")
         ((:auth rc)
```

```
 (:session rc 'spawn)
 "auth ok")
(else (redirect-to rc "/login?login_failed=true")))))
```

NOTE: The passwd will be encrypted by default algorithm.

19.5 HMAC

HMAC (`https://en.wikipedia.org/wiki/HMAC`) is hash-based message authentication code. Because it's very dangerous to store the passwd directly, the safer way is to salt then hash with a strong cryptograpic hash function before store the passwd.

The default salt is a random string get from the operating system. And the default cryptographic hash function is SHA256. You may customize your own HMAC function:

```
(define (my-hmac passwd salt)
  (string->sha-512 (format #f "~a-~a-~a" passwd salt (current-time))))

(post "/auth" #:auth `(table user "user" "passwd" "salt" ,my-hmac)
 ..... )
```

The default HMAC function is:

```
(define (default-hmac passwd salt)
  (string->sha-256 (string-append passwd salt)))
```

For more hash functions, please see Section 24.3 [Cryptographic hash functions], page 50.

20 Cache

20.1 On web caching

Web caching is very important nowadays. This section discusses proper web caching. It is not a full product guide document, but may help to understand how to cache in GNU Artanis.

(to be continued...)

20.2 Cache APIs

You have to use `#:cache mode` while you defining URL rule handler.

```
(get "/certain-rule" #:cache mode
     (lambda (rc) ...))
```

NOTE: the default value of maxage is defined by `cache.maxage` in `/etc/artanis/artanis.conf`. The default value is 3600 seconds.

mode could be:

- `#t` for enabling caching the page.
- `#f` for disabling caching the page explicitly. It's default to not cache.
- `('static [maxage <- 3600])` This mode must be used for static files, which means the URL rule must be a real path to a static file.
- `(filename [maxage <- 3600])` Specify a static file to cache. This is useful when you don't want to reveal actual path of the static file, but use a fake URL for it.
- `('public filename [maxage <- 3600])` Allow proxies cache the content of specified static file. If HTTP authentication is required, responses are automatically set to private.
- `('private filename [maxage <- 3600])` Not-Allow proxies cache the content of specified static file.

Let's see the simplest cache test (for dynamic content):

```
(get "/new" #:cache #t
     (lambda (rc)
       (:cache rc "hello world")))
```

If you want to cache a static file, and permit proxies cache the content:

```
(get "/hide" #:cache '(public "pub/some.html")
     (lambda (rc)
       (:cache rc)))
```

But, if your current URL rule is used for authentication (once you use `#:auth`), the cache will be changed to **private** even if you specify **public**.

```
(get "/pauth"
  #:auth `(basic ,(lambda (rc u p) (and (string=? u "nala")
                                        (string=? p "123"))))
  #:cache '(public "pub/some.html") ; will be changed to 'private' forcely.
  (lambda (rc) (:cache rc)))
```

21 Shortcuts

21.1 What is shortcuts?

The *shortcuts* is a series of special functions. It's used to simplify the complex operations, according to the configuration specified by the related keyword, which is set by you after a URL-rule.

It was named *OHT* which stands for *Optional Handler Table*, which indicate the basic principle to be implemented, but too obscured to understand. So let's just call it *shortcut*.

Anyway, you may find them in the module (artanis oht) (`https://gitlab.com/NalaGinrut/artanis/blob/master/artanis/oht.scm`).

It's a good practice to use *shortcuts* as possible and avoid calling low-level APIs.

Each shortcuts consists of 2 parts: **config** and **apply**.

config means you need to configure certain service for the specific URL rule. This configuration will only be availble for this URL rule, and indenpendent to other registered URL rules.

apply is used for calling specific functions related to your configuration in **config** step. The first argument of **apply** method must be `route-context` and it is described in Chapter 10 [Route context], page 21.

21.2 Database connection

This is useful when you use database. The shortcut provides a way to interact with the raw connection. The connection is fetched from connection pool, which has been created when GNU Artanis is started.

```
;; config
#:conn #t

;; apply
(:conn <route-context> [sql])
```

- The second argument is optional, if it's missing, then `:conn` will return the raw connection after applying `(:conn rc)`.
  - NOTE: If you didn't set `#:conn #t`, and applied `(:conn rc)`, then `(rc-conn rc)` will return `#f`. This is why you shouldn't use low-level `(rc-conn rc)`.
- If the second argument exists, then it should be a valid SQL string for querying. The returne value is described in Section 14.1 [DB connection pool], page 28.
  - The SQL string could be generated from Section 14.4 [SSQL (experimental)], page 29.

21.3 Raw SQL

This shortcut is useful for simple oneshot query.

```
;; config
#:raw-sql sql
```

```
;; apply
(:raw-sql <route-context> mode)
```

Sql must be a valid SQL string.

Mode is listed below:

- 'all for getting all the results.
- 'top for getting the first results.
- A positive integer to indicate how many results should be returned.

21.4 String template

The shortcut for Section 24.1 [String Template], page 50. Sometimes it's useful when you just need a quick way to use string template. But it doesn't support multi templates. If you do need multi templates please use the traditional Section 24.1 [String Template], page 50.

```
;; config
#:str "string template"

;; apply
(:str <route-context> key-values ...)
```

Please checkout Section 24.1 [String Template], page 50, to find out how to use the *string-template* and *key-values*.

21.5 SQL-Mapping shortcut (unfinished)

This is related to Section 14.6 [SQL Mapping (experimental)], page 32, which is still experimental, maybe you should wait for the next version.

```
;; config
#:sql-mapping config-patterns

;; apply
(:sql-mapping <route-context> command ...)
```

Here're the **config-patterns**:

- `#t` enable the simple sql-mapping.
- `'(path ,path ,name)` Fetch the sql-mapping with *name* in specified *path*.
  - *name* should be in symbol type.
  - *path* should be in string type, and an existing path in your filesystem.
- `'(add ,name ,sql-template)` Fetch the sql-mapping with *name* rendered from *sql-template*.
  - *name* should be in symbol type.
  - *sql-template* is decribed in Section 14.6 [SQL Mapping (experimental)], page 32.

22 Websocket (Experimental)

22.1 Websocket introduction

Websocket is becoming more and more important for modern web development. GNU Artanis is trying to provide an industrial strenth and efficient Websocket implementation. Moreover, Websocket is important for the design of GNU Artanis, please see Section 23.2 [Principles], page 48, for more details.

22.2 Websocket basic usage

(The Websocket support is still experimental and unfinished, only demo works, please don't use it)

In GNU Artanis, the Websocket handling is triggered by certain URL registered by you. You should use `#:websocket` to configure the Websocket. For example:

```
(use-modules (artanis artanis))

(get "/echo" #:websocket '(proto echo)
     (lambda (rc)
       (:websocket rc 'payload)))

(run #:port 3000)
```

In this simple test, we choose the simplest **echo** protocol of Websocket that return the string sent from the client back. And we write a simple JS for web frontend:

```
function WebSocketTest()
{
    if ("WebSocket" in window)
    {
        document.write("<p>WebSocket is supported by your Browser!</p>");

        // Let us open a web socket
        var ws = new WebSocket("ws://localhost:3000/echo");

        ws.onopen = function()
        {
            // Web Socket is connected, send data using send()
            ws.send("hello world");
            document.write("<p>Message is sent...</p>");
        };

        ws.onmessage = function (evt)
        {
            var received_msg = evt.data;
            document.write("<p>hello welcome...</p>");
        };
```

```
            ws.onclose = function()
            {
                // websocket is closed.
                document.write("<p>Connection is closed...</p>");
            };

            window.onbeforeunload = function(event) {
                socket.close();
            };
        }
        else
        {
            // The browser doesn't support WebSocket
            document.write("<p>WebSocket NOT supported by your Browser!</p>");
        }
    }
```

22.3 Websocket APIs

NOTE: The Websocket is very preliminary that only support echo. So it's not usable yet.

22.3.1 Websocket configuration

```
#:websocket '(proto ,protocol_name)
```

The **protocol_name** could be:

- **'echo** for simply echo test. **NOTE:** More regular protocols will be added in the future.

  ```
  #:websocket simple_pattern
  ```

  The **simple_pattern** could be:

  - **#t** or **'raw** means this URL enables Websocket without specifed protocol. So you will get raw data of the decoded payload.

    ```
    #:websocket '(redirect ,ip/usk)
    ```

    This is used for redirecting Websocket stream to other address. ip/usk means ip or unix-socket, the pattern should be this:

    - `^ip://(?:[0-9]{1,3}\\.){3}[0-9]{1,3}(:[0-9]{1,5})?$`
    - `^unix://[a-zA-Z-_0-9]+\\.socket$`

      ```
      #:websocket '(proxy ,protocol)
      ```

      Setup a proxy with certain protocol handler. Different from the regular proxy design, the proxy in Artanis doesn't need a listen port, since it's always 80/443 or customized HTTP port. The client should support websocket, and visit the related URL for establishing a websocket channel. Then the rest is the same with regular proxy.

22.3.2 Websocket application

```
(:websocket <route-context> command)
```

The **command** could be:

- **'payload** to get the decoded data from the client. It's decoded from Websocket frame automatically. So you don't have to parse the frame.

22.4 Websocket frame

According to RFC6455 (`https://tools.ietf.org/html/rfc6455`), GNU Artanis provides Websocket frame data struct.

The frame will not be decoded or parsed into a record-type, on the contrary, it'll be kept as a binary frame read from client, and use bitwise operations for fetching the fields. This kind of 'lazy' design will save much time on parsing unused fields each time, and eaiser for redirecting without any serialization. If users want to get certain field, Artanis provides APIs for fetching them. Users can decide how to parse the frames for efficiency.

Here're the APIs:

```
(websocket-frame? <websocket-frame>)

;; parser: bytevector -> customized data frame
(websocket-frame-parser <websocket-frame>)
```

websocket-frame-parser is the registered reader for the protocol specified by `#:websocket` configuration. The protocol is customizable based on protobuf. *NOTE: The customized protocol hasn't been implemented yet.*

```
(websocket-frame-head <websocket-frame>)
(websocket-frame-final-fragment? <websocket-frame>)
(websocket-frame-opcode <websocket-frame>)
(websocket-frame-payload <websocket-frame>)
(websocket-frame-mask <websocket-frame>)
```

Get the Websocket frame information, see Data framing (`https://tools.ietf.org/html/rfc6455#page-27`).

- **head** is the first 2 bytes in the data frame.
- **final-fragment** means it's the last frame in a session.
- **opcode** is the opcode in the frame, see Section 22.5 [Websocket opcode], page 46.
- **payload** is the actual data which is encoded.
- **mask** is the mask of the frame.

22.5 Websocket opcode

Defines the interpretation of the "Payload data". If an unknown opcode is received, the receiving endpoint MUST fail the WebSocket connection.

```
;;  check if it's a continuation frame
(is-continue-frame? opcode)

;;  check if it's text frame
(is-text-frame? opcode)

;; check if it's binary frame
(is-binary-frame? opcode)

;; check if it's control frame
(is-control-frame? opcode)
(is-non-control-frame? opcode)
```

```
;;  websocket requires closing
(is-close-frame? opcode)

;;  check if it's a ping frame
(is-ping-frame? opcode)

;;  check if it's a pong frame
(is-pong-frame? opcode)

;;  %xB-F are reserved for further control frames
(is-reserved-frame? opcode)
```

23 Ragnarok server core

23.1 Introduction

Since 0.2, GNU Artanis has a strong server core for high concurrency. It is named Ragnarok. In the philosophy of the design of GNU Artanis, everything is meant to be flexible and customizable. So the server core is customizable, in case someone thought Ragnarok is not good enough.

Ragnarok doesn't use any popular library for handling events (libev/libuv etc . . .). It's a brand new server core based on epoll and delimited continuations (`https://en.wikipedia.org/wiki/Delimited_continuation`).

23.2 Principles

The basic principle of Ragnarok is co-routine. And these co-routines are implemented with delimited continuations (`https://en.wikipedia.org/wiki/Delimited_continuation`). Actually, there's no OS-kernel controlled threads (say, pthread) for scheduling *request-handler* in Ragnarok. All the tasks are scheduled by an userland scheduler, and the task is nothing but just a special continuation. The key difference between it and regular full-stack continuation (`https://en.wikipedia.org/wiki/Call-with-current-continuation#Criticism`) is that it could be delimited for fine granularity rather than capture the whole stack.

For reaserchers, there is a paper published on ICFP Scheme Workshop 2016 conference (`http://www.schemeworkshop.org/2016/`) to explain the principle and the design of GNU Artanis:

Multi-purpose web framework design based on websockets over HTTP Gateway (`https://github.com/NalaGinrut/artanis/raw/gh-pages/research/scheme16/art2016.pdf`).

(to be continued . . .)

23.3 Features

In Artanis, the request handling could be scheduled when the socket buffer is full (depends on `server.bufsize`). And let other request's handler run. Just like the scheduling of OS but it's in the userland.

So if it's the buffer issue when scheduling, then there's no way to flush before break since we can't tell if the scheduling caused by buffering or blocking.

Ragnarok takes advantage of `SO_REUSEPORT` introduced since GNU/Linux 3.9 to provde a feature named `server.multi` which could be enabled in config. This feature allows users to start several Artanis instances which are all listenning on the same port to take advantage of multi cores. And the events are dispatched by the Linux kernel.

(to be continued . . .)

23.4 Ragnarok APIs

You may use these APIs for customizing your own server core.

(to be continued . . .)

24 Utils

**The functions listed below requires to import (artanis utils) (`https://gitlab.com/NalaGinrut/artanis/blob`
module.**

24.1 String Template

GNU Artanis provides Python3-like template strings:

```
(make-string-template tpl . vals)
```

- **tpl** stands for template string.
- **vals** is varg-list specifying default value to certain key.

For an example:

```
(define st (make-string-template "hello ${name}"))
(st #:name "nala")
;; ==> "hello nala"

;; or you may specify a default value for ${name}
(define st (make-string-template "hello ${name}" #:name "unknown"))
(st)
;; ==> "hello unknown"
(st #:name "john")
;; ==> "hello john"
```

24.2 Random String Generator

Get random string from `/dev/urandom`.

```
(get-random-from-dev #:length 8 #:uppercase #f)
```

24.3 Cryptographic hash functions

```
;; hash a string with MD5
(string->md5 str)
;; hash a string with SHA-1
(string->sha-1 str)
```

Since Artanis-0.2.5, we have SHA-2 hash functions:

```
(string->sha-224 str)
(string->sha-384 str)
(string->sha-512 str)
```

24.4 Stack & Queue

GNU Artanis provides simple interfaces for stack & queue:

```
;; stack operations
(new-stack)
(stack-pop! stk)
(stack-push! stk elem)
```

```
(stack-top stk)
(stack-remove! stk key)
(stack-empty? stk)

;; queue operations
(new-queue)
(queue-out! q)
(queue-in! q elem)
(queue-head q)
(queue-tail q)
(queue-remove! q key)
(queue-empty? q)
```

24.5 Useful string operation

If you want to get all contents in string from a file, then don't use `get-string-all` imported from rnrs. Because it will not detect the correct charset from locale, and this may cause the length different from the actual length. Although GNU Artanis can handle the length issue properly, you should use `get-string-all-with-detected-charset` once you need to do the similar thing. If you don't care about the contents but just want to get the contents anyway, it's better to use `get-bytevector-all` imported from rnrs.

```
(get-string-all-with-detected-charset filename)
```

24.6 Time operation tool

TODO

25 Debug mode

GNU Artanis provides debug-mode for more convenient debug. You may enable it easy.

For the simplest way, pass `#:debug #t` when calling `run` function:

```
(run #:debug #t)
```

If you use MVC or created an app folder, just pass –debug or -g:

```
# In app folder
art work --debug
# Or
art work -g
```

When you enabled debug-mode, the Model and Controller modules written by you will be reloaded automatically on the fly.

If *not*, you have to press Ctrl+C to quit GNU Artanis server and start it again. This saves time.

And you may add paths to monitor certain files (for an instance, JSON as config file to be reloaded on the fly) if you want to be notified when they're changed. Just put the paths here:

```
debug.monitor = my/lib/json, my/lib/modules
```

26 Appendix A GNU Free Documentation License

 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 noncommercially. 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.

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.

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.

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.

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:

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. 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. State on the Title page the name of the publisher of the Modified Version, as the publisher. Preserve all the copyright notices of the Document. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices. 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. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice. Include an unaltered copy of this License.

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. 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. 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. 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. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section. 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.

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."

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.

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.

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.

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 explic-

itly 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.

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.

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.