

GNU Artanis

Mu Lei known as NalaGinrut

Table of Contents

1	Introduction	1
1.1	Conventions used in this manual	1
1.2	No warranty	1
2	License	2
3	Installation	3
3.1	For users	3
3.2	For contributors	3
4	Configuration	4
4.1	Config APIs	4
5	Hello World	5
5.1	Use Guile REPL and verify GNU Artanis installation	5
5.2	Simple HTTP server	5
5.3	Try simple URL remapping	5
5.4	More complex URL remapping	6
5.5	Regex in URL remapping	6
5.6	Database operating	7
6	Scheme Basics	9
6.1	For newbies	9
6.2	For Pythonistas	9
6.3	For Rubyist	9
6.4	For deep learners	9
7	GNU Artanis Basics	11
7.1	How to run a site with GNU Artanis	11
7.2	Initialization	11
7.3	Registering handlers for HTTP methods	11
7.4	Emit a Response	11
7.5	Running the server	12
7.6	Working with Nginx	12
8	The Art command line	13
8.1	art create	13
8.2	art draw	13
8.3	art migrate	13
8.4	art work	14

9	URL remapping	15
9.1	Introduction to URL remapping	15
9.2	URL handling	15
9.3	Get parameters from a URL	15
9.4	Redirecting	15
10	Route context	16
10.1	Route context APIs	16
11	MVC	18
11.1	Controllers/Views	18
11.2	Models	18
12	Query String	19
12.1	Query string from GET	19
12.2	Query string from POST	19
13	Layouts and Rendering in GNU Artanis	21
13.1	Templating	21
13.2	Templating for Pythoners	21
13.3	Templating for Rubyists	21
13.4	Templating APIs	21
13.5	Embedded Templates	22
13.5.1	Template special commands	22
13.6	SXML Templates	23
14	Databases	24
14.1	DB init hooks	24
14.2	DB connection pool	24
14.3	Migration	24
14.4	ORM problem	25
14.5	SSQL	26
14.6	FPRM (experimental)	26
14.6.1	Connect to DB server	26
14.6.2	Map DB table	27
14.6.3	Create table	27
14.6.4	Get columns from table	27
14.6.5	Set values to table	28
14.6.6	Drop a table	28
14.6.7	Check existence of table	28
14.6.8	Get schema of a table	28
14.7	SQL Mapping (experimental)	28

15	MIME	29
15.1	JSON	29
15.2	CSV	29
15.3	XML	29
15.4	SXML	30
16	Upload files	31
16.1	Receive an upload from the client	31
16.2	Send an upload to a Server	31
17	Sessions	32
17.1	Session backend	32
18	Cookies	33
19	Authentication	34
19.1	Init Authentication	34
19.2	Basic Authentication	34
19.3	Common Authentication	35
19.4	Login authentication	36
19.5	HMAC	36
20	Cache	37
20.1	On web caching	37
20.2	Cache APIs	37
21	Shortcuts	38
21.1	What is shortcuts?	38
21.2	Database connection	38
21.3	Raw SQL	38
21.4	String template	39
21.5	SQL-Mapping shortcut (unfinished)	39
22	Websocket (Experimental)	40
22.1	Websocket introduction	40
22.2	Websocket basic usage	40
22.3	Websocket named-pipe	41
22.4	Websocket APIs	42
22.4.1	Websocket configuration	42
22.4.2	Websocket application	42
22.5	Websocket frame	42
22.6	Websocket opcode	43

23	Ragnarok server core	45
23.1	Introduction	45
23.2	Principles	45
23.3	Features	45
23.4	Ragnarok APIs	46
24	Key-Value Database	47
24.1	LPC	47
25	Utils	48
25.1	String Template	48
25.2	Random String Generator	48
25.3	Cryptographic hash functions	48
25.4	Stack & Queue	48
25.5	Useful string operation	49
25.6	Time operation tool	49
26	Debug mode	50
27	Appendix A GNU Free Documentation License	51

1 Introduction

Copyright (C) 2018 Mu Lei known as NalaGinrut.
 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'.

This manual documents GNU Artanis, a fast, monolithic 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_\(programming_language\)](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 [BROKEN LINK: nil] 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_002f0.html), which is the key to implement asynchronous non-blocking server core in GNU Artanis.

```
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-2.1.7 (The tarball included guile-dbd) [**Optional**]

```
wget -c https://github.com/opencog/guile-dbi/archive/guile-dbi-2.1.7.tar.gz
```

```
tar xvzf guile-dbi-2.1.7.tar.gz
cd guile-dbi-guile-dbi-2.1.7/guile-dbi && ./configure && make
sudo make install
```

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

```
tar xvzf guile-dbd-mysql-2.1.6.tar.gz
cd guile-dbi-guile-dbi-2.1.7/guile-dbd-mysql && ./configure && make
sudo make install
```

MySQL is used for the examples in this manual. You may find dbd plugins for other databases here (<http://download.gna.org/guile-dbi>) or at this 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 directory. The configure file must be named `/etc/artanis/artanis.conf`.
- If you're using application directory, the configure file `conf/artanis.conf` will be generated automatically for you.

4.1 Config APIs

To change the default configurations:

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

To get the current configuration:

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

To get the 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 remapping

Type these code in Guile REPL:

```
(use-modules (artanis artanis))
(init-server)
(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 compared to the simpler example:

- 1. The special rule, `"/hello/:who"`, `:who` means you can use *params* to refer to the value of the par of the URL with the key "who". Like: `(params rc "who")`.
- 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 the formatted output should be returned in a string rather than printed out.

5.5 Regex in URL remapping

You can use regular expressions as a URL rule argument.

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

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

5.6 Database operating

GNU Artanis supports mysql/postgresql/sqlite3. We use mysql as an example here.

Please ensure that your DB service was started before you run this code.

If you encounter any problems, it's very likely it's with your DB config.

You can use a DB (such as mysql) with GUI tools such as "adminer", independently of the running 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 the GNU Artanis API handling tables in a DB. Here, we define this mapping as the var *mtable*.
- We can use *mtable* to handle tables, you can get the values from a table with the 'get command.
- *mtable* is a function which accepts the first argument as a command, say 'create, which is a command to create a new table. 'set is used to insert/update values in the table. And the 'get command to fetch the values of specific columns.
- The second argument of *mtable* is the name of the table as you can guess. Please note that it is case sensitive, while the column name isn't.
- The 'create command returns a function too, which also accepts an argument as a command. Here, we use the 'valid? command to check if the table has been created successfully.

This was just a simple introduction. You may read the DB section in this manual for details.

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 is quick explanation:

- 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 in this example, we have to use the Guile API `object->string` to convert the query result into a string.

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

6 Scheme Basics

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

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 the 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 GNU Artanis Basics

7.1 How to run a site with GNU Artanis

This is the simplest way 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 file name extensions. GNU Artanis is based on URL remapping, so the requested URL will have to end in the requested file name, matching the string defined, and returning the file without any extra definitions per file type. By default, it covers the most common static file types.

`#:cache-statics?` controls whether the static files should be cached.

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

7.3 Registering handlers for HTTP methods

Please read Section 9.2 [URL handling], page 15.

7.4 Emit a Response

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

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

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

#:headers lets you specify custom HTTP headers. The headers must follow a certain format. Please read Response Headers (http://www.gnu.org/software/guile/manual/html_node/HTTP-Headers.html#Response-Headers) for details.

#:mtime specifies the modified time in the response. GNU Artanis will generate it for you when not defined.

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

filename is the filename to be sent as a response.

[headers] is the custom HTTP headers.

7.5 Running the server

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

keywords with the value #f, as default, will get the values from the config file.

But you can define them as well.

`#:host` the hostname.

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

`#:debug` set `#t` if you want to enable debug mode. Logs will be more verbose.

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

`#:dbd` choose a dbd. These are the supported three: mysql, postgresql, and sqlite3.

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

`#:db-passwd` the DB password for the user above.

`#:db-name` the DB name to use.

7.6 Working with Nginx

You may try GNU Artanis+Nginx with a reverse proxy.

Although GNU Artanis has good server core, I would recommend you use Nginx as the front server. In addition to the enhanced performance, it'll also be less vulnerable to attacks.

These are some sample lines for `/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;
}
```

To make it work, restart Nginx after editing the file:

```
sudo service nginx restart
```

And run GNU Artanis:

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

8 The Art command line

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

8.1 art create

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

```
art create proj_path
```

8.2 art draw

This command will generate the specified components:

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 18, 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 14.3 [Migration], page 24, for more detail.

8.4 art work

This command is used to start the server when run in a project directory:

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 [BROKEN LINK: nil].
- For Database (DBD) alternatives, please see [BROKEN LINK: nil].

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 WAFs(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 write a handler for any type.

In GNU Artanis the HEAD method is handled by the server, so you can't define specific handlers directly for it within GNU Artanis.

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 [BROKEN LINK: nil]

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

For a POST method:

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

9.3 Get parameters from a URL

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

9.4 Redirecting

```
(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 type object which encapsulates the necessary information for the server from the current request context. We named it *route* because it's related to the route of Chapter 9 [URL remapping], page 15. Usually it's passed to the page handler as a unique argument. It's supposed to provide sufficient data about the current request.

```
(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 22.5 [WebSocket frame], page 42, 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 19.

```
(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-View-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

Let's make a controller named *article*:

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

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

This will generate both a **controller** and a **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
```

These three files are generated:

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

Based on this, the controller *article* will have two methods mapped to the URL rule, *show* and *edit*. As part of the *view* component. An HTML template is generated for each method. For the *show* method for example, the view file **show.html.tpl** is created. For the *controller* component, you get a *show* method handler, as:

```
(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, you're free to use or not use these templates. If you want to use the *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 21.

11.2 Models

Models contains operations of database.

For modifying tables, you should read Section 14.3 [Migration], page 24.

For other DB operation, please read Section 14.6 [FPRM (experimental)], page 26.

(To be continued. . .)

12 Query String

A query string is a special part of a URL:

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

In this example, "name" and "color" are query strings with the values "ferret" and "purple" respectively. It's useful to pass parameters to the server side like this.

GNU Artanis provides a convenient API to handle query strings.

12.1 Query string from GET

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

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

Please notice that URL-remapping supports regex. So you could register a 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 the query string.

12.2 Query string from POST

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

There's only a slight difference when you pass query string by POST instead of with GET: you don't need a URL rule using a regex. So, the "?" above would be gone in this example.

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

Please notice that `#:from-post 'qstr` is necessary when you're trying to get data from POST. And you should use `:from-post` to get related data from query-string.

```
#:from-post <mode>
```

The mode includes:

- `#t` or `'qstr`: handle query-string for you.
- `'json`: returns a parsed json as hashtable.
- `'qstr-safe`: similar to `'qstr`, but try to eliminate evil HTML entities first.
- `'bv`: returns the body as bytevector.
- `'store rest ...`: It's for Chapter 16 [Upload files], page 31.

The reason to design `:from-post` is for the efficient purpose. Artanis will not try to auto parse POST body as query-string for at least 2 reasons:

- 1. It may not be query-string, maybe json, or uploaded file
- 2. It may be long query-string, and could be delayed to parse. This is

useful to avoid redundant parsing. However, each time you call `:from-post`, it will parse the query-string again, the correct way to fetch multiple values is:

```
(:from-post rc 'get-vals "key1" "key2" "key3")
```

BTW, you may get the parsed query-string as an assoc-list as well:

```
(let ((qstr (:from-post rc 'get)))  
  (assoc-ref qstr "key-1"))
```

13 Layouts and Rendering in GNU Artanis

13.1 Templating

Templating provides a way to mix programmatic 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. Templating in GNU Artanis follows a different philosophy.

Templating in GNU Artanis, is just writing Scheme code in the HTML document. Why? Because of the philosophy of FP(Functional Programming), everything is a function. So, (`filesizeformat size`) is easy enough to grasp for anyone with scheme experience. It's just a 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 `size | filesizeformat` follows 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 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` returns an HTTP object response.

[environment] is the environment you want to use. We often ignore it. If you want to ref some vars defined outside your template string, you need to pass this variable.

[escape?] If you want to char-escape the HTML with the returned string, set it to `#t`. There are two main ways of writing templates:

13.5 Embedded Templates

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 mandatory when using the MVC. Since the MVC will find the template by detecting controller name automatically.

If you don't use the MVC, and are rather writing all in one script file loading GNU Artanis modules. Then you don't need to follow this rule.

```
(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` in the same path as your GNU Artanis code.

Since `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 complain about not being able to find the variable `external-var`.

If you don't need to refer to any external vars, just use `(tpl->response "file.tpl")`.

To test, access `http://localhost:3000/test` in your browser.

13.5.1 Template special commands

GNU Artanis provides special helper commands.

Please notice that GNU Artanis constrains the path of sources in the application directory for security reasons. The resources files, CSS, JS etc, should be put into `pub` directory in the application directory, or the client won't be able to access them.

These special commands are useful to expand the path for you, and they should be added into the template 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 @, as, **@include**, **@css**, etc. Please do not separate the @, or it will throw exception.

You can include html files with the **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 in the **pub** directory in the application directory.

To refer to a CSS file:

```

<@css filename.css %>

```

This will be expanded like this:

```

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

```

To refer to a JS (javascript) 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 an icon for the domain:

```

<@icon favicon.ico %>

```

This will be expanded like this:

```

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

```

13.6 SXML Templates

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 it clearer.

One benefit of SXML is that it takes advantage of the quasiquote in Scheme. Please search in the internet "scheme quasiquote" for more details.

This is an SXML expression:

```

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

```

The above would result the following HTML code:

```

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

```

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

```

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

```

Here, the "html" block is being quoted with the backtick, which in combination with a "," character before the variable name, makes the variable be referred to instead of just passing a string.

14 Databases

14.1 DB init hooks

Sometimes you need to do some configurations before using DB, Artanis provide an API for that, you should put it to ENTRY file before run the server.

For example, assuming you're using MySQL/MariaDB, and you need to configure it to UTF-8, you should add these lines to you ENTRY file. Or if you're using minimal mode without application folder, just put it before running the server.

```
(run-when-DB-init!
 (lambda (conn)
  (DB-query conn "set names utf8;")))
```

NOTE: Don't forget '!' here, it implies the side-effects in Scheme!

14.2 DB connection pool

TODO

14.3 Migration

Migrations provide a way to do complicated modification of tables in a database by GNU Artanis. 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.

Then you'd edit the 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))
```

Then you run the **up** command for 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	auto		primary-key		
name	char-field	not-null		maxlen 10	
age	tiny-integer	not-null			
email	char-field			maxlen 20	

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 the **down** command of migration, as:

```
art migrate down person
```

The table *person* will be dropped.

14.4 ORM problem

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

Of course, Guile has its own Object System named GOOPS (https://www.gnu.org/software/guile/manual/html_node/GOOPS.html#GOOPS). Users can 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 critics about 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/>)

Here are some known ways to solve ORM related problems:

- 1. *Give up ORM*.
- 2. *Give up relational storage model*. Don't use a relational DB. Use another DB style, such as No-SQL. Well, this way is not cool when you must use a 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*. Limiting 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. Microsoft's LINQ is such a case.
- 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)

14.5 SSQL

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 command is quite different to SQL grammar. Example:

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

14.6 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.

14.6.1 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. It must match any of these: "mysql", "postgresql", or "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 the DB server of your choice.
- **#:host** specifies the host name.
- **#:port** specifies the socket port.

14.6.2 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 an Active Record. (<http://www.martinfowler.com/eaCatalog/activeRecord.html>) It maps the database view to a class object.

There are two main differences to SSQL:

- FPRM doesn't create an object for each table. It maps the whole database, and generates an SQL for each table as you use it. So it might be lighter compared to an ORM object.
- FPRM doesn't maintain any states at all. It's stateless as an object (Not in the 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 14.7 [SQL Mapping (experimental)], page 28, provided. FPRM aims to reduce states & complexity to provide reliability. And SQL-Mapping will provide a convenient way to handle complex SQL queries for better performance and security (SQL-Injection, etc).

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

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

`map-table-from-DB` returns a function, named `m` here for simplicity.

14.6.3 Create table

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

- `table-name` specifies the name of the DB table.
- `defs` is a list to define the type of columns. For example:

```
'((name varchar 10) (age integer) (email varchar 20))
```
- `#:if-exists?` has two kinds of possible options:
 - `'overwrite` and `'drop`, which will overwrite the existing table.
 - `'ignore` means to ignore the table when it already exists.
- `#:primary-keys` specifies the primary keys in the created table.
- `#:engine` specifies the DB engine. It depends on what dbd you chose.

14.6.4 Get columns from table

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

- `#:column` is the list of columns to get.
- `#:functions` is a function to call, e.g:

```
#:functions '((count Persons.Lastname))
```
- `#:ret` specifies how to return the result, there are three options:
 - `'all` for returning all results

- `'top` for returning the first result
- integer (larger than 0), to give it the end of a range from 0 to this number to give as result.
- `#: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)
```

14.6.5 Set values to table

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

`kargs` is a var-list that takes key-value arguments.

For example:

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

14.6.6 Drop a table

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

14.6.7 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)
```

14.6.8 Get schema of a table

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

NOTE: all the returned names of the schema will be in lowercase.

14.7 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) ; to emit the reponse with the proper MIME
```

15.1 JSON

GNU Artanis integrates the third-party module `guile-json` (<https://github.com/aconchillo/guile-json>) to parse json. You can use the `#: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 a json object
;; ==> {"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 the client, please use `scm->json-string`.

15.2 CSV

GNU Artanis integrates the third-party module `guile-csv` (<https://github.com/NalaGinrut/guile-csv>) to parse csv. You can use the `#: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 would be to append the text to a common string.

```
(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 be able to upload files, `store-uploaded-files` is your friend.

16.1 Receive an upload from the client

The typical configuration of an uploading WebAPI looks like this:

```
(post "/upload" #:from-post '(store #:path "upload" #:sync #f)
  (lambda (rc)
    (case (:from-post rc 'store)
      ((success) (response-emit "upload succeeded!"))
      ((none) (response-emit "No uploaded files!"))
      (else (response-emit "Impossible! please report bug!")))))
```

However, you may use the low-level API for more configurations as well:

```
(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 the specified path to put uploaded files.

`#:uid` is new UID for the uploaded files, `#f` uses the default UID.

`#:gid` specifies the 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 an upload to a Server

Although GNU Artanis is often used in server-side, we provide this function for users to upload files from the 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 can use `#:session mode` to define a URL rule handler.

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

mode can be:

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

And the APIs of the session is `:session`

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

cmd can be:

- `'check` to check the session with name "sid".
- `'(check ,sid)` to check the session with a specified sid name.
- `'check-and-spawn` to check "sid" first, if it doesn't exist, then spawn new.
- `'(check-and-spawn ,sid)` to do the same as above, but with specifying the sid name.
- `'(check-and-spawn-and-keep ,sid)` to check, then spawn, then keep, specifying the sid name.
- `'spawn` to spawn a session with the name "sid".
- `'spawn-and-keep` to spawn a session then keep with the name "sid".

17.1 Session backend

Artanis provide several backends for implementing sessions in the lower-level. Please take a look at the description of `session.backend` in your `artanis.conf`.

18 Cookies

You can use `#:cookies` mode to define a URL rule handler.

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

mode can be:

- (`'names names ...`) to specify the list name for the cookies.
- (`'custom (names ...) maker setter getter modifier`) to specify a more complicated customized cookie handler.

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) ; cookie 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"))
```

You can use these commands in your console to see the results:

```
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 can use `#:auth mode` to define a URL rule handler.

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

mode can be:

- SQL as Section 25.1 [String Template], page 48. You can write your own custom SQL string to fetch & check your username and password.
- `('basic (lambda (rc user passwd) ...))` init Basic Authentication mode. *user* is the username, and *passwd* is the password.
- `('table table-name username-field passwd-field)` init a common Authentication mode. **The passwd will be encrypted by the 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 the passwd field will be "passwd" and username will be "username" by default. You can encrypt the passwd with `crypto-proc`.

Available `crypto-proc` helper functions listed here:

```
(string->md5 <string>)
(string->sha-1 <string>)
(string->sha-224 <string>)
(string->sha-256 <string>)
(string->sha-384 <string>)
(string->sha-512 <string>)
```

NOTE: Please make sure that the `username-field` and `passwd-field` must be the same with the field name specified in the submit form of you web page code.

For example, if there is a form on you page:

```
<input type="password" name="passwd">
```

Please notice that name of password input was specified to "passwd".

Then you should write authentication like this:

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

Please notice that the "passwd" here is the same with what you specified in the form.

19.2 Basic Authentication

The HTTP Basic authentication (BA) implementation is the simplest technique for enforcing access control to web resources, as it doesn't require cookies, session identifiers, or login pages. But rather uses static, standard HTTP headers, which means that no extra handshakes are necessary for the connection.

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

GNU Artanis doesn't support HTTPS at present. There are plans 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 to provide authentication is to compare the password stored in a database table:

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

NOTE: Assuming **username** and **passwd** are columns of the Person table.

You have to define your own checker with the anonymous function `(lambda (rc u p) ...)`. `#t` to show success, and `#f` to fail.

APIs:

- `(:auth rc)` will check if Basic Authentication succeeded, `#f` if not.
- `(throw-auth-needed)` is a useful helper function to ask for auth in client side.

19.3 Common Authentication

There are multiple authentication methods that can be used by developers. Most of them are sort of tricky hacks. Here are the most common.

The most common, and relatively safe way to authenticate, is to use the POST method, and check the username and passwd from a table in the DB.

There are several ways to provide authentication.

The simplest case is for Section 25.1 [String Template], page 48:

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

If you save the account data in a database table, then you can use the table mode:

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

NOTE: The square-bracketed **args** above are optional.

The default values for the optional items are:

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

TODO: remove the brackets for salt-field, as it says below that it's not optional. And specify whether "optional" means that can be skipped or set to #f to use the default value.

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

So please prepare a field in a table for the salt string. It's your duty to generate a salt string, please see Section 25.2 [Random String Generator], page 48. When authenticating, please specify the salt field name in the salt-field argument.

For hmac item, please see Section 19.5 [HMAC], page 36.

19.4 Login authentication

Usually, when doing a login, you will need both #:auth and #:session options for a long time session. The first step is to authenticate, if it's successful, then spawn a new session for this request.

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 the default algorithm.

19.5 HMAC

HMAC (<https://en.wikipedia.org/wiki/HMAC>) is a hash-based message authentication code. It's dangerous to store the passwd in a raw string. A safer way is to salt then hash with a strong cryptographic hash function when storing the passwd.

The default salt is a random string got from the operating system. And the default cryptographic hash function is SHA256. You can set your own HMAC function, as in this example:

```
(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 on hash functions, please refer to Section 25.3 [Cryptographic hash functions], page 48.

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 it may help to understand how to cache data in GNU Artanis.

(to be continued...)

20.2 Cache APIs

You can use `#:cache` mode to define a URL rule handler.

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

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

mode can be:

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

Let's set a simple cache setting 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?

shortcuts are a series of special functions. They're used to simplify complex operations, according to the configuration specified by the related keyword, set after a URL-rule.

It was named *OHT* which stands for *Optional Handler Table*, which indicates the basic principle to be implemented. But it was too hard to remember. 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 good practice to use *shortcuts* as much as possible and avoid calling low-level APIs.

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

config is to configure a certain service for the specific URL rule. This configuration will only be available to this URL rule, and independent to other registered URL rules.

apply is used to call specific functions related to your configuration in the **config** step. The first argument of the **apply** method must be a **route-context** (Chapter 10 [Route context], page 16).

21.2 Database connection

This is a useful feature to use when you connect to a database. The shortcut provides a way to interact with the raw connection. The connection is fetched from the connection pool, which is created at GNU Artanis's start up.

```
;; 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 haven'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 you pass a second argument, it should be a valid SQL query string. The returned value is described in Section 14.2 [DB connection pool], page 24.
 - You can create the SQL query string with Section 14.5 [SSQL], page 26.

21.3 Raw SQL

This shortcut is useful for a simple one-shot query.

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

;; apply
```

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

Sql must be a valid SQL query string.

Mode is one of:

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

21.4 String template

This is a shortcut for Section 25.1 [String Template], page 48. Sometimes it's useful when you just need a quick way to use a string template. It doesn't support multi templates, so if you do need to, please use the traditional Section 25.1 [String Template], page 48.

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

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

Please checkout Section 25.1 [String Template], page 48, to find out how to use the *string-template* and *key-values*.

21.5 SQL-Mapping shortcut (unfinished)

This is related to Section 14.7 [SQL Mapping (experimental)], page 28, which is still experimental, maybe you should wait for the next version if you wish to use it.

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

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

config-patterns can be any of:

- **#t** enable the simple sql-mapping.
- '(path ,path ,name) Fetch the sql-mapping with *name* in the specified *path*.
 - *name* must be an object of the symbol type.
 - *path* must be an object of the 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 an object of the symbol type.
 - *sql-template* is described in more detail in Section 14.7 [SQL Mapping (experimental)], page 28.

22 Websocket (Experimental)

22.1 Websocket introduction

Websockets are becoming more and more important for modern web development. GNU Artanis is trying to provide an industrial strength and efficient Websocket implementation. Websockets are important for GNU Artanis's design. Please see Section 23.2 [Principles], page 45, for more details.

22.2 Websocket basic usage

In GNU Artanis, a Websocket handling is triggered by setting it on a specific URL. 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 the Websocket. This will return back the string sent from the client. Let's also write a simple javascript function for the 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
  {
    // Your browser doesn't support WebSockets
    document.write("<p>WebSocket NOT supported by your Browser!</p>");
  }
}

```

22.3 Websocket named-pipe

Artanis provides the named-pipe based on Websocket, which is very useful to interact between the client and server. *NOTE: The Websocket named-pipe is very useful to implement server-push messaging.*

Here is the critical API:

```
(named-pipe-subscribe <route-context>)
```

Let's try a simple example:

```
(get "/robot" #:websocket '(proto echo) named-pipe-subscribe)
```

Here we register a WebAPI named `"/robot"`, and configure it a Websocket API by `#:websocket '(proto echo)`. As you may see, we specify its protocol to a simple echo protocol, it is to say, it will redirect what it receives. Each time it receives a message, it will redirect the message to the specified named-pipe, and the subscriber of that named-pipe will get the message instantly. The name of the pipe is specified by the client like this:

```
var ws = new WebSocket("ws://localhost:3000/robot?artanis_named_pipe=robot-1");
```

The query-string `"artanis_named_pipe"` is a special key in Artanis, you **MUST** use this key to specify the pipe name. Here we specify the pipe name as `"robot-1"`.

Now we setup the named-pipe, however, we still need to setup another WebAPI to let the client send message:

```
(get "/welcome/:whom/:what" #:websocket 'send-only
  (lambda (rc)
    (:websocket rc 'send (params rc "whom") (params rc "what"))
    "ok"))
```

The configure `#:websocket 'send-only` means this API is only for sending, so that it's half-duplex. And another critical Websocket API here:

```
(:websocket <route-context> 'send <pipe-name> <data>)
```

Each time you send data from the WebAPI, for example:

```
curl localhost:3000/welcome/robot-1/nala
```

In our simple example of Websocket named-pipe test (<https://gitlab.com/NalaGinrut/artanis/blob/master/examples/websocket-named-pipe.html>), and you should run The named-pipe will receive the message "nala", and the page on the browser will print it instantly.

22.4 Websocket APIs

NOTE: The Websocket is in a very preliminary stage. It only support echo.

22.4.1 Websocket configuration

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

protocol_name can be:

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

```
#:websocket simple_pattern
```

simple_pattern can be:

- **#t** or **'raw** will enable a WebSocket on this URL without specifying the protocol. So you will get the raw data from the decoded payload.

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

This is used for redirecting a WebSocket stream to another address. ip/usk is an ip or a unix-socket. The string regexp pattern has to match:

- `^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 a specific protocol handler. Unlike the regular proxy approach, the proxy in Artanis doesn't need a listening port. Since it's always 80/443 or a custom port. The client has to have websocket support. And be able to access the specified URL to establish a websocket connection. The rest is the same as with a regular proxy.

22.4.2 Websocket application

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

command can 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.

22.5 Websocket frame

GNU Artanis provides a WebSocket data frame struct, as defined in RFC6455 (<https://tools.ietf.org/html/rfc6455>).

The frame will not be decoded or parsed into a record-type, but will be kept as the binary frame read from the client, and use bitwise operations to fetch fields. This kind of 'lazy' design saves much time as it doesn't parse unused fields, and makes it easier to redirect without any serialization. If users want to get a certain field, Artanis provides APIs for fetching them. Users can decide how to parse the frames by themselves, which we think is more efficient.

Here are the APIs you can use:

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

websocket-frame-parser is the registered reader for the protocol specified by the #:websocket configuration. The protocol is customizable based on protobuf. *NOTE: Custom protocols support 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>)
```

To get the WebSocket frame information. See Data framing (<https://tools.ietf.org/html/rfc6455#page-27>) for details.

- **head** returns the first 2 bytes in the data frame.
- **final-fragment** returns true if it's the last frame in a session.
- **opcode** returns the opcode in the frame. Refer to Section 22.6 [WebSocket opcode], page 43.
- **payload** returns the actual encoded data.
- **mask** returns the frame mask.

22.6 WebSocket opcode

Opcode defines the interpretation of "Payload data". If an unknown opcode is received, the receiving endpoint's WebSocket connection will fail.

```
;; 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 version 0.2, GNU Artanis has started to use a strong server core for high concurrency. Its name is Ragnarok. In the philosophy of the design of GNU Artanis, everything is meant to be flexible and customizable. The server core is no exception. In case Ragnarok doesn't suit your needs, you're free to use something else.

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

A basic characteristic of Ragnarok is the use of co-routines. These co-routines are implemented with delimited continuations (https://en.wikipedia.org/wiki/Delimited_continuation). There are no OS/kernel controlled threads, like `pthread`, for scheduling *request-handlers* in Ragnarok. All the tasks are scheduled by a userland scheduler. And the task is nothing but just a special continuation. The key difference between this and a regular full-stack continuation (<https://en.wikipedia.org/wiki/Call-with-current-continuation#Criticism>), is that you can set limits with precision, instead of having to capture the whole stack.

For researchers, 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 can be scheduled even when the socket buffer is full (depends on `server.bufsize`). And let other handlers deal with the requests. Just like the scheduling of an OS, but in userland.

If you have issues with the buffer when scheduling, there's no way to flush before it breaks, since we can't tell if the scheduling is caused by the buffering or the blocking.

Ragnarok takes advantage of `SO_REUSEPORT` introduced in GNU/Linux 3.9 to provide a feature named `server.multi` which can be enabled in the config. This feature allows users to start several Artanis instances listening to the same port, to take advantage of multi core CPUs, and the Linux Kernel managing the events.

(to be continued ...)

23.4 Ragnarok APIs

You can use these APIs to customize your own server core.

(to be continued . . .)

24 Key-Value Database

24.1 LPC

LPC stands for Lightweight Persistent Cache. It's the easiest way to use key-value DB in Artanis.

For example:

```
(get "/certain_rule" #:lpc <backend> (lambda (rc) ...))
```

The backend includes:

- #t or 'redis
- 'json

After configured, it's easy to use it:

```
;; Setter  
(:lpc rc 'set <key> <val>)
```

```
;; Getter  
(:lpc rc 'get <key>)  
;; or  
(:lpc rc 'ref <key>)
```

In default, the key will be prefixed automatically:

```
(string-append "__artanis_lpc_" (get-conf '(db name)) "_" key)
```

25 Utils

The functions listed below require the (artanis utils) (<https://gitlab.com/NalaGinrut/artanis/blob/master>) module.

25.1 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 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"
```

25.2 Random String Generator

Get random string from `/dev/urandom`.

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

25.3 Cryptographic hash functions

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

SHA-2 hash functions are also supported from Artanis-0.2.5.

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

25.4 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)
```

25.5 Useful string operation

If you want to get all the contents from a file into a string, then don't use `get-string-all` imported from `rnr`. Because it will not detect the correct charset from the locale, and this may cause the length differ from the actual value. Although GNU Artanis can handle this length issue properly, you should use `get-string-all-with-detected-charset` when you need to do something like this. If you don't care about the contents encoding but just want to get the them, it's better to use `get-bytevector-all` imported from `rnr`.

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

25.6 Time operation tool

TODO

26 Debug mode

GNU Artanis provides a debug-mode for a more convenient way to debug. It's very easy to use.

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

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

If you are using the MVC system, or created a project directory, just pass `-debug` or `-g` to `art`:

```
# In the project directory
art work --debug
# Or
art work -g
```

When you enable debug-mode, the Model and Controller modules in the directory will be reloaded automatically every time they're called.

When *not* in debug mode, you have to press Ctrl+C to quit GNU Artanis server and start it again to test changed modules. Debug mode saves time when testing.

You can add paths to monitor certain files (for example, a JSON file 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
```

27 Appendix A GNU Free Documentation License

Version 1.3, 3 November 2008 Copyright © 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc. <http://fsf.org/>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed. 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. Secondly, 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.