An implementation of HTTP /HTTPS and SSL for Scheme 48

Harald Glab-Plhak <hglabplhak@icloud.com>

September 22, 2025

Contents

1 Abstract

What we like to do is to support ***SSL / HTTP / HTTPS*** in Scheme 48. The interface for that will be implemented nearly implemented as Domain Specific Language for Networking. I will use ***LibreSSL*** as external library to avoid errors and bugs in the implementation by doing that on my own with much less time than the developers and designers this components have due to much more manpower. The goal is to implement the ***HTTP /1.1*** and ***HTTP/2*** and ***HTTP/3*** as well. To get something like a pseudo standard, the interface is inspired by the ***HTTP-EASY*** library written in ***Racket Language*** and is a free style. port of it (we use the codebase and transfer it to real scheme and make it fit to S48) HTTP-EASY (Lib Site) which is implemented in racket which is also inspired by scheme and which also has an ***R6RS*** Scheme standard implemented. The implementation and interface is committed to the requirements defined by the following ***RFC Documents:***

– RFC-5246 (TLS V 1.2)

The Transport Layer Security (TLS) Protocol Version 1.2 Authors: T. Dierks Independent, E. Rescorla RTFM, Inc.

– RFC-8446 (TLS V 1.3)
The Transport Layer Security (TLS) Protocol Version 1.3 Author(s): E. Rescorla Mozilla

– RFC-6101 (SSL V 3.0)
The Secure Socket Layer (SSL) Protocol Version 3.0 Authors: A. Freier, P. Karlton Netscape Communications, P. Kocher Independent Consultant

– RFC-3986 (URI)
The Uniform Resource Identifier (URI) Authors: T. Berners-Lee W3C/MIT, R. Fielding Day. Software, L. Masinter Adobe Systems

– RFC-9112 (HTTP/1.1)
The specification of the Hyper Text Transfer Protocol / 1.1 (Authors: R. Fielding, Ed. , M. Nottingham, Ed., J. Reschke, Ed.)
– RFC-9110 (HTTP Semantics)
The semantics specification of the Hyper Text Transfer Protocol (Authors: R. Fielding,Â Ed., M. Nottingham,Â Ed., J. Reschke,Â Ed.)
– RFC-9113 (HTTP/2)
The specification of the Hyper Text Transfer Protocol / 2 (Authors: M. Thomson, Ed. , C. Benfield, Ed.)
– RFC-7541 (HPAC)
HPACK: Field Compression for HTTP/2 Authors: R. Peon Google Inc., H. Ruellan Canon CRF
– RFC-9114 (HTTP/3)
The specification of the Hyper Text Transfer Protocol / 3(Authors: M. Bishop, Ed.)
– RFC-9204 (QPAC)
QPACK: Field Compression for HTTP/3 Authors: C. Krasic, M. Bishop Akamai Technologies, A. Frindell, Ed.Facebook

In addition to the client library a full functional toolkit will be implemented for developing application containers with ***RESTful***[1] interfaces (e.g. using ***YAML*** definitions) or at last my be application containers and something like ***OCSP***[2] Services. This will be really enough for the first step.

2 Introduction

Scheme 48 is a SCHEME interpreter actually following the R5RS standard but the R6RS is nearly ready for delivery. In this implementation up to now there is a net component implementing socket communication. The first step to reach our goal will be to introduce a SSL capable implementation using the current implementation for port and channel handling to get an appropriate layer definition for implementing the things for HTTP / HTTPS.

Short explanation of the Plain Socket / SSL Layer: For the SSL Layer we built up a little syntax with define-syntax, which is a define for a new generic usable function definition which contains the protocol definition, the CTX the Certificate and the connection parameters in a record after that this command executes a handshake and open connection in a way that we get a port number which can be used either by LibreSSL or by the native C socket API to do HTTP / HTTPS. Short explanation of the HTTP / HTTPS Layer: The HTTPS layer is defined on top of the SSL Layer by the next `define-syntax`.

2.1 The Sockets implementations

First of all we have to design our interface and the syntax and attributes and functional syntactic layout of our tiny DSL. After that it is necessary to build the low level functionality in C so that we can then build our things in scheme. I prefer this way because it is good to see an early success so that the work is not too long before knowing that the concept works. The first part that we develop of course. also as a base for the DSL is a. procedural interface (function calls).

2.2 Issues for the project

Here are some issues which are important for our task:

- How read / write is planned
  - Using the new. Port / Channeel I/O written for R6RS
  - Make it stable for multi-threading via atomically

[1] ***REST Requests:***The ***R***epres***E***ntational abbrhighcolState abbrhighcolTransfer API is an architectural style and the RESTful services follows this specification to design a special form of HTTP requests.
[2] ***OCSP***: The ***O***pen ***C***ertificate ***S***ervice ***P***rotocol is used to implement servers being requested using OCSP Requests and defined OCSP Responses for checking the correctness and the validity of a certificate.

- We use some kind of pseudo code to describe the logic here
  - Using the My Lisp dialect / McCarthy Lisp dialect and formula
- Additional things which we will implement
  - Implement URL encoding
  - Basic and Certificate AUTH Headers
  - Implement base for RESTful services including yaml like definition and JSON support
  - Implement MULTIPart attachemenzs with MIME spec.

2.3 The procedural interface

The procedural interface is the interface where several functions are defined for each layer. This interface can be used in a very flexible way. This part is the positive thing about this part of implementation. The negative effect of this part of interface is that the function have to be used in the right way and order. In cause of that we define also a DSL which ensures the usage of the functions in the right way. This part will be described. later. For getting a clear architecture the implementations are divided in a low-level interface on which the high level interface (The recommended interface) is built as layer with more simple. and straight arguments not needing too much technical details for better under5standing by the people using the library.

2.4 Procedural Interface for Sockets

The interface for defining Sockets is build by using the C-Socket API and for the Secure Socket Layer and Transport Security Layer LibreSSL is used. Th functions we need are called in a Scheme48 FFI Module. The FFI is the Foreign Function Interface. With this interface the functions can be called and S48 values can be returned which we use after defining this layer in the pure scheme part.

2.5 Procedural Interface for HTTP/ HTTPS

In this part of the procedural interface the HTTP / HTTPS protocols are implemented V1.1 the older nearly outdated format although it is widely used up to now.The V 2.0 with HPAC[3] and V 3.0 with QPAC [4].

2.6 The DSL definition as interface

The DSL[5] for the SSL Layer itself will be designed by using the define-syntax and the other hygienic macro functionality.

A Domain specific language has to be defined in a way that the Function / Keywords can be used inside the definition of the syntax for the DSL

2.7 The SSL syntactic layer

At first we need a syntax to declare a Socket***Plain/ SSL***. Here we have to think about connection parameters and about parameters and attributes for ***SSL / TLS*** (e.g. ***CTX*** and certificate keys and a specific Algorithm for transport. The***CTX*** will be a structure/ record-type as well as the definition of the base connection parameters.

2.8 The HTTP / HTTPS syntactic layer

In this layer a grammar is defined to have a save way to create ***HTTP (GET, PUT, POST, DELETE,HEAD....)*** Requests and to define the request data in the same step. With the created 'object' a send can be done. The same is done for the server side receive the other way round. There is also a grammar defined for sending the HTTP / HTTPS request to the server. and to handle / give back the response.

3 Procedural interface definitions

First we design and implement the procedural interface. In this interface we define functions for the different tasks. The parameters and the returns are designed in a way that the functions are doing a create of the request, a connection

[3] ***HPAC:*** Header Compression for HTTP/2
[4] ***QPAC:*** Header Compression for HTTP/3
[5] ***DSL:*** A ***D***omain ***S***pecific ***L***anguage is a language normally built up in the host language. A DSL is designed for special tasks and areas e.g.: Windowing.

establishment and the send .... for each task there are one or more functions. Of course the functions differ if the use either a secure or a non secure connection and transmission. For using that interface a deeper technical know how as in the syntactic layer about all these things and the possibilities is neccessary. And now finally let's dive into it more detailed.

3.1 The SSL procedural interface in C / Scheme

3.2 The HTTP/HTTPS Common functionality

3.3 The HTTP/HTTPS procedural interface in Scheme

3.4 The transport layer

3.5 The HTTP Method definition and the resource definition

3.6 The QUIC protocol implementation

3.7 The header compression for HTTP/2 and HTTP/3 HPAC / QPAC

3.8 Multipart attachements

3.9 RESTful service layer over HTTP

4 APPENDIX A: Terminology Explained in short

4.1 The HTTP 1.1 / 2 / 3 Protocol

The HTTP[6] was developed for transport of data over TCP/IP in a well defined structured way The HTTP 1.1 protocol was and is widely used for a long time. After this the HTTP 2.0 protocol was developed containing many new features and optimizing especially in compressing header information via HPAC to lower the amount of data transported per each request. With the HTTP/3 now a new way is established with a protocol not based on TCP/IP but on a protocol called QUIC which is built on top of UDP.

4.2 The HTTP/2 and HTTP/3 protocols

The HTTP/2 protocol is. built on top of the TCP/IP. protocol. A. header values compression takes place here called HPACK. HPACK uses the Huffman algorithm to compress header values in an encode / decode process. Since TCP / IP is designed im the first priority to be reliable and to keep data consistent as well as not loosing data packages it is not optimal for transporting a high amount of data even also parallel in a short time. For that reason ***HTTP/3*** was developed based on the ***QUIC protocol*** [7]. ***TCP***[8] and ***UDP*** [9] are two different ways of handling data transfer and requests in the internet.They handle data and requests and responses. in another way. UDP is connection less unordered and faster. TCP has the advantage that data loss is correctly detected and handled and that the data are checked for consistence. The QUIC protocol is designed to have a way of using UDP with the additional advantage that the consistence of data is save as far as possible.

[6] ***HTTP:*** The ***H***yper ***T***ext ***T***ransport ***P***rotocol used for transport e.g. ***HTML*** or designing ***REST Requests*** . The protocol is implemented by all Web Browsers and is used all over the internet to transport / requesting and servicing data in a structured form with defined requests like ***GET, PUT, DELETE, HEAD ....*** All of these requests have a defined response format.

[7] ***QUIC protocol:*** ***Q***uick ***UDP I***nternet ***C***onnection

[8] ***TCP:*** ***T***ransmission ***C***ontrol ***P***rotocol

[9] ***UDP:*** ***U***ser ***D***atagram ***P***rotocol