An implementation of HTTP /HTTPS and SSL for Scheme 48

Harald Glab-Plhak <hglabplhak@icloud.com>

September 16, 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 and (TODO Fill in) as external Libraries to avoid errors and bugs in the implementation by doing that on my own with much less time than the developers and designers of 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 slightly by the HTTP-EASY Library 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-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-9114 (HTTP/3)
The specification of the Hyper Text Transfer Protocol / 3(Authors: M. Bishop, Ed.)

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

[1] **REST Requests:** The **R**epres**E**ntational **S**tate **T**ransfer 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.

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 the HTTPS layer is defined on top of the SSL Layer by the next `define-syntax`.

The Plain Sockets and SSL 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 DSL definitions

The DSL[3] 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

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.

3 APPENDIX A: Terminology Explained in short

3.1 The HTTP 1.1 / 2 / 3 Protocol

The HTTP[4] protocol has its origin in

[3] **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.

[4] **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 requestshave a defined response format.