openssl egg

Bindings to the OpenSSL SSL/TLS library
Extension for Chicken Scheme
Version 1.1.1

Thomas Chust

Table of Contents

1 About this egg

1.1 Version history

`1.1.1`	Output that would block properly suspends threads now
`1.1.0`	`##sys#tcp-port->fileno` and `tcp-addresses` are now supported on SSL ports
`1.0.0`	Corrections, tests against `openssl s_server`, `openssl s_client` and comparison with the PLT module
`0.4.0`	Server functionality added
`0.3.1`	Client-only with certificate functions
`0.2.0`	Client-only prerelease

1.2 Usage

Load this egg like so:

```
(require-extension openssl)
```

2 Documentation

This reference is basically a copy of the documentation of PLT Scheme's openssl module. The API provided here is largely compatible with that one. The exceptions are the missing `.../enable-break` and `ssl-available?` procedures and the missing `reuse?` argument to `ssl-listen`.

Please note that all the procedures described here may fail and raise a non-continuable exception of the composite type `(exn i/o net openssl)`. The `openssl` property condition contains a property called `status` which will be bound to a symbol corresponding to the OpenSSL error code that was encountered. It may have the following values:

Symbol	**Meaning**
`'zero-return`	The SSL/TLS connection was shut down unexpectedly but in a controlled way
`'want-read`	The operation didn't finish because data must be read from a nonblocking socket. This error condition only occurs though, when it could not be handled automatically because there is actually no socket involved or some other strange thing happended in the OpenSSL library.
`'want-write`	The operation didn't finish because data must be read from a nonblocking socket. The same comment as for `'want-read` applies.
`'want-connect`	The operation didn't finish because a nonblocking socket must first be connected. The same comment as for `'want-read` applies.
`'want-accept`	The operation didn't finish because a nonblocking socket must first be acepted. The same comment as for `'want-read` applies.
`'want-X509-lookup`	The operation failed because an application callback that could not even have been registered through this API was apparently registered anyway and has asked to be called again.
`'syscall`	Some low-level I/O error occurred.
`'ssl`	Something went wrong in the OpenSSL library itself.
`#f`	The error is not classified

Of course the exception that is thrown also has an appropriate message set.

If you feel that this documentation lacks some information, please also consider the manual pages of OpenSSL.

2.1 Client procedures

`ssl-connect` [procedure]

```
(ssl-connect (hostname <string>) #!optional (port <exact>) ((ctx <ssl-client-cont
```

Connect to the given `host` on the given `port` (a number from 1 to 65535). This connection will be encrypted using SSL. The return values are as tcp-connect; an input port and an output port.

The optional `ctx` argument determines which encryption protocol is used, whether the server's certificate is checked, etc. The argument can be either a client context created by `ssl-make-client-context` (see below), or one of the following symbols: `'sslv2-or-v3` (the default), `'sslv2`, `'sslv3`, or `'tls`. See `ssl-make-client-context` for further details, including the meanings of the protocol symbols.

`ssl-make-client-context` [procedure]

```
(ssl-make-client-context #!optional ((protocol <symbol>) 'sslv2-or-v3)) => <ssl-c
```

Creates a context to be supplied to `ssl-connect`. The context identifies a communication protocol (as selected by `protocol`), and also holds certificate information (i.e., the client's identity, its trusted certificate authorities, etc.). See the "Certificate procedures" section below for more information on certificates.

The `protocol` must be one of the following:

Symbol	**Meaning**
`'sslv2-or-v3`	SSL protocol versions 2 or 3, as appropriate
`'sslv2`	SSL protocol version 2
`'sslv3`	SSL protocol version 3
`'tls`	the TLS protocol version 1

By default, the context returned by `ssl-make-client-context` does not request verification of a server's certificate. Use `ssl-set-verify!` to enable such verification.

`ssl-client-context?` [procedure]

```
(ssl-client-context? (obj <top>)) => <bool>
```

Returns `#t` if `obj` is a value produced by `ssl-make-client-context`, `#f` otherwise.

2.2 Server procedures

`ssl-listen` [procedure]

```
(ssl-listen (port <exact>) #!optional ((backlog <exact>) 4) ((hostname <string>)
```

Like `tcp-listen`, but the result is an SSL listener. The extra optional `ctx` argument is as for `ssl-connect`.

Call `ssl-load-certificate-chain!` and `ssl-load-private-key!` to avoid a `"no shared cipher"` error on accepting connections.

`ssl-close` [procedure]
`ssl-listener?` [procedure]
`ssl-listener-port` [procedure]
`ssl-listener-fileno` [procedure]
`ssl-listener-accept-ready?` [procedure]
`ssl-accept` [procedure]

```
(ssl-close (listener <ssl-listener>)) => <void>
(ssl-listener? (obj <top>)) => <bool>
(ssl-listener-port (listener <ssl-listener>)) => <exact>
(ssl-listener-fileno (listener <ssl-listener>)) => <exact>
(ssl-listener-accept-ready? (listener <ssl-listener>)) => <bool>
(ssl-accept (listener <ssl-listener>)) => <input-port>, <output-port>
```

Analogous to `tcp-close`, `tcp-listener?`, `tcp-listener-port`, `tcp-listener-fileno`, `tcp-accept-ready?` and `tcp-accept`.

2.3 Certificate procedures

`ssl-load-certificate-chain!` [procedure]

```
(ssl-load-certificate-chain! (obj <ssl-client-context|ssl-listener>) (pathname <s
```

Loads a PEM-format certification chain file for connections to be made with the given context (created by `ssl-make-context`) or listener (created by `ssl-listener`).

This chain is used to identify the client or server when it connects or accepts connections. Loading a chain overwrites the old chain. Also call `ssl-load-private-key!` to load the certificate's corresponding key.

`ssl-load-private-key!` [procedure]

```
(ssl-load-private-key! (obj <ssl-client-context|ssl-listener>) (pathname <string>
```

Loads the first private key from `pathname` for the given client context or listener. The key goes with the certificate that identifies the client or server.

If `rsa?` is `#t`, the first RSA key is read (i.e., non-RSA keys are skipped). If `asn1?` is `#t`, the file is parsed as ASN1 format instead of PEM.

`ssl-set-verify!` [procedure]

```
(ssl-set-verify! (obj <ssl-client-context|ssl-listener>) (v <bool>)) => <void>
```

Enables or disables verification of a connection peer's certificates. By default, verification is disabled.

Enabling verification also requires, at a minimum, designating trusted certificate authorities with `ssl-load-verify-root-certificates!`.

`ssl-load-verify-root-certificates!` [procedure]

```
(ssl-load-verify-root-certificates! (obj <ssl-client-context|ssl-listener>) (path
```

Loads a PEM-format file containing trusted certificates that are used to verify the certificates of a connection peer. Call this procedure multiple times to load multiple sets of trusted certificates.

The optional second argument specifies a directory in which certificates are automatically looked up. You may also only pass a path in this argument and pass `#f` as the first argument to this procedure. See the OpenSSL documentation on `SSL_CTX_load_verify_locations` for more details.

`ssl-load-suggested-certificate-authorities!` [procedure]

`(ssl-load-suggested-certificate-authorities! (obj <ssl-client-context|ssl-listene`

Loads a PEM-format file containing certificates that are used by a server. The certificate list is sent to a client when the server requests a certificate as an indication of which certificates the server trusts.

Loading the suggested certificates does not imply trust, however; any certificate presented by the client will be checked using the trusted roots loaded by `ssl-load-verify-root-certificates!`.

3 License

Index