# HTTPmsg_CNL113312 Test Port for TTCN-3 Toolset with TITAN, Function Specification

István Óváry

Version 155 17-CNL 113 312, Rev. E, 2010-12-14

# Table of Contents

# How to Read This Document

This is the Function Specification for the HTTPmsg_CNL113312 (called HTTP from now on) test port. The HTTP test port is developed for the TTCN-3 Toolset with TITAN. This document is intended to be read together with User's Guide [3]

# Scope

The purpose of this document is to specify the functionality of the HTTP test port. The document is primarily addressed to the end users of the product. Basic knowledge of TTCN-3, TITAN TTCN-3 Test Executor and the HTTP protocol is valuable when reading this document (see [1] and [2]

This document is based on specifications of Hypertext Transfer Protocol (HTTP1.1) defined by RFC 2616.

# General

The HTTP Test Port makes possible to execute test suites towards an IUT. The test port allows sending and receiving HTTP messages between the test suite and IUT via a TCP/IP socket connection.

The HTTP Test Port can be used as a protocol module via encoding and decoding functions, see Encoding and Decoding Functions.

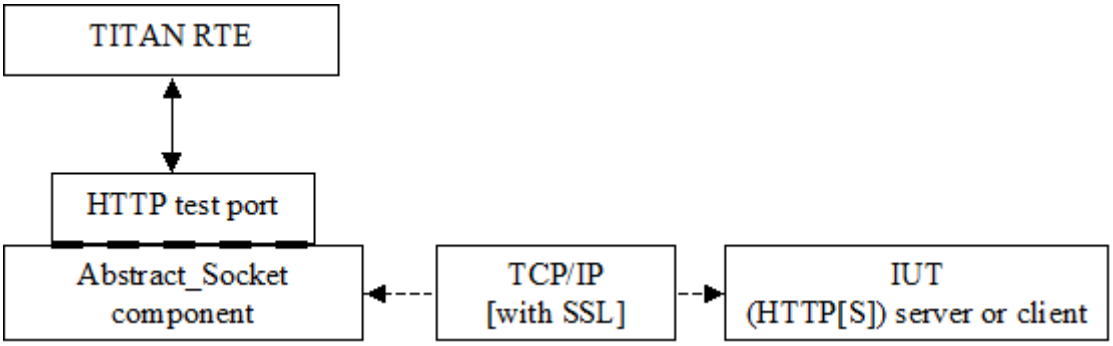
Both IPv4 and IPv6 are supported.

The test port can handle multiple connections. Every connection gets an 'id' when it is established. When sending HTTP messages, the `client_id` parameter selects the connection on which the message should be sent. If it is set to `omit`, the HTTP message will be sent on the first available connection.

The communication between the HTTP test port and the TITAN RTE is done by using the API functions described in [2]

The HTTP protocol messages are then transferred by the HTTP test port to the IUT through a network connection.

See the transfer of HTTP protocol messages by the HTTP test port to the IUT through a network connection below:




# Function Specification

# Implementation

## Environment

The HTTP test port makes use of the services provided by the UNIX socket interface. When connecting to an SSL enabled IUT, the connection is secured with the OpenSSL toolkit based on configuration data. Every test port is able to handle one listening (server) port and multiple TCP connections. Proxy is supported.

## Module Structure

The HTTP test port is implemented in the following TTCN-3 modules:

- *HTTPmsg_Types.ttcn*
- *HTTPmsg_PortType.ttcn*

The file *HTTPmsg_Types.ttcn* defines the HTTP message types and ASPs to control the TCP connection, furthermore contains the declaration of the declaration of the encoding and decoding external functions.

The port type is defined in *HTTPmsg_PortType.ttcn*.

The c++ implementation of the test port and the encoding-decoding functions are contained in the following files:

- *HTTPmsg_PT.hh*
- *HTTPmsg_PT.cc*

The encoding and decoding functions also have been implemented here.

The port is using the Abstract_Socket, a common component with the product number CNL 113 384, implementing the basic sending, receiving and socket handling routines. The following files should be included in the Makefile:

- *Abstract_Socket.hh*

- *Abstract_Socket.cc*

# Configuration

The configuration of the HTTP test port is done by the TITAN RTE configuration file. The description of the specific parameters can be found in the HTTP test port User's Guide [2]

## Notification ASPs

The test port is able to provide the result of the Connect, Listen operations, and inform the server test suite if a new client has connected or the remote end has disconnected a specific connection. This behavior is switched on by setting the `use_notification_ASPs` test port parameter to `"yes"`.

# Start Procedure

After the test port is mapped by TITAN RTE to the IUT system's interface port, it waits for a `Connect` or a `Listen` ASP.

The `Connect` ASP sets up a connection toward an HTTP server, on which instances of HTTP Messages can be sent and received.

The `Listen` ASP commands the test port to wait for incoming connections from HTTP clients by opening a listening port.

For detailed operation see the User Guide [3]

# Sending/Receiving HTTP Messages

## HTTP Messages Sent by the Test Port

The HTTP test port is able to send and receive `HTTPMessage` structures. The `HTTPMessage` can be one of the following types:

- `HTTPRequest`

  The Request message represents a single request to perform by the HTTP server, usually to access a resource on the server.

- `HTTPResponse`

  The Response message is sent by the HTTP server to the client. It includes the return status code of the request and the requested resource.

- `HTTPRequest_binary_body`

  The same as the `HTTPRequest` message. It is passed to TTCN when the body of the message contains non-ascii characters.

- `HTTPResponse_binary_body`

  The same as the `HTTPResponse` message. It is passed to TTCN when the body of the message contains non-ascii characters.

Apart from the `HTTPRequest` and `HTTPResponse` ASPs above, the `erronous_msg` is received by the test port and sent to the test suite:

- `HTTP_erronous_msg` If a message is received on the connection, which can not be decoded as a HTTP1.1 or HTTP1.0 message, the `Message` will contain an erroneous message with the `client_id`, and sent to the test suite.

For detailed operation see the User Guide [3]

# Encoding and Decoding Functions

If the test port is used as protocol module, the following encoder and decoder funcions are available:

| Name | Type of formal parameters | Type of return value |
|---|---|---|
| `enc_HTTPMessage` | HTTPMessage | octetstring |
| `dec_HTTPMessage` | in octetstring stream<br>inout HTTPMessage msg<br>in boolean socket debugging | integer |

The encoder function returns with an octetstring as the encoded form of the `HTTPMessage` structure.

The decoder function returns with the number of not processed octets of the input octetstring stream and the decoded message in its inout parameter.

If the return value is not zero, there are not processed octetets. Those octets can be gathered from the original octetstring by the user and can be processed by calling the decoding function once again with the modified stream. This process is necessary only if more http message can be found in the original stream.

# Message Length Function

The following function can be used to calculate the length of the received HTTP message. It returns the length of the received HTTP message in octets or -1 if the length can not be determined.

| Name | Type of formal parameters | Type of return value |
|---|---|---|
| `f_HTTPMessage_len` | in octetstring stream | integer |

# Closing Down

## Close

The `Close` shuts down the client connection between the test port and the IUT. The `client_id` parameter of the `Close` ASP identifies the connection to be closed. If it is set to omit, all current connections will be closed.

The `Half_close` ASP indicates that the remote end closed the connection.

## Shutdown

Instructs the test port to close the server listening port. The client connections will remain open. The server will not accept further client connections until a `Listen` ASP is sent again.

For detailed operation see the User Guide [3]

# Logging

The type of information that will be logged can be categorized into two groups. The first one consists of information that shows the flow of the internal execution of the test port, e.g. important events, which function that is currently executing etc. The second group deals with presenting valuable data, e.g. presenting the content of a PDU. The logging printouts will be directed to the RTE log file. The user is able to decide whether logging is to take place or not by setting appropriate configuration data, see [2]

# Error Handling

Erroneous behavior detected during runtime may be presented on the console and directed into the RTE log file. The following two types of messages are taken care of:

- Errors: information about errors detected is provided. If an error occurs the execution of the test case will stop immediately. The test ports will be unmapped.
- Warnings: information about warnings detected is provided. The execution continues after the warning is shown.

# SSL Functionality

The SSL implementation is based on the same OpenSSL as TITAN. Protocols SSLv2, SSLv3 and TLSv1 are supported.

## Compilation

The usage of SSL and even the compilation of the SSL related code parts are optional. This is because SSL related code parts cannot be compiled without the OpenSSL installed.

The compilation of SSL related code parts can be disabled by not defining the `AS_USE_SSL` macro in

the *Makefile* during the compilation. If the macro is defined in the *Makefile*, the SSL code parts are compiled to the executable test code. The usage of the SSL can be enabled/disabled by setting the `use_ssl` field of the `Connect`/`Listen` ASPs. For more information about the compilation see [3]

## Authentication

The test port provides both server side and client side authentication. When authenticating the other side, a certificate is requested and the own trusted certificate authorities' list is sent. The received certificate is verified whether it is a valid certificate or not (the public and private keys are matching). No further authentication is performed (e.g. whether hostname is present in the certificate). The verification can be enabled/disabled in the runtime configuration file, see [3].

In server mode the test port will always send its certificate and trusted certificate authorities' list to its clients. If verification is enabled in the runtime configuration file, the server will request for a client's certificate. If the client does not send a valid certificate, the connection will be refused. If verification is disabled, then the connection will be accepted even if the client does not send or send an invalid certificate.

In client mode the test port will send its certificate to the server on the server's request. If verification is enabled in the runtime configuration file, the client will send its own trusted certificate authorities' list to the server and will verify the server's certificate as well. If the server's certificate is not valid, the SSL connection will not be established. If verification is disabled, then the connection will be accepted even if the server does not send or send an invalid certificate.

The own certificate(s), the own private key file, the optional password protecting the own private key file and the trusted certificate authorities' list file can be specified in the runtime configuration file, see [3].

The test port will check the consistency between the own private key and the public key (based on the own certificate) automatically. If the check fails, a warning is issued and execution continues.

## Other Features

Both client and server support SSLv2, SSLv3 and TLSv1, however no restriction is possible to use only a subset of these. The used protocol will be selected during the SSL handshake automatically.

The usage of SSL session resumption can be enabled/disabled in the runtime configuration file, see [3].

The allowed ciphering suites can be restricted in the runtime configuration file, see [3].

The SSL re-handshaking requests are accepted and processed, however re-handshaking cannot be initiated from the test port.

# Limitations

- No restriction is possible on the used protocols (e.g. use only SSLv2); it is determined during SSL handshake between the peers.

- SSL re-handshaking cannot be initiated from the test port.
- The own certificate file(s), the own private key file and the trusted certificate authorities' list file must be in PEM format. Other formats are not supported.

# Terminology

**Sockets:**
The sockets is a method for communication between a client program and a server program in a network. A socket is defined as "the endpoint in a connection." Sockets are created and used with a set of programming requests or "function calls" sometimes called the sockets application programming interface (API). The most common sockets API is the Berkeley UNIX C language interface for sockets. Sockets can also be used for communication between processes within the same computer.

# Abbreviations

**API**

Application Program Interface

**ASP**

Abstract Service Primitive

**IUT**

Implementation Under Test

**RTE**

Run-Time Environment

**HTTP**

Hypertext Transfer Protocol

**SUT**

System Under Test

**SSL**

Secure Sockets Layer

**TTCN-3**

Testing and Test Control Notation version 3

# References

[1] ETSI ES 201 873-1 v3.1.1 (2005-06)The Testing and Test Control Notation version 3; Part 1: Core Language

[2] TITAN User Guide

[3] HTTPmsg_CNL113312 Test Port for TTCN-3 Toolset with TITAN, User Guide

[4] RFC 2616
Hypertext Transfer Protocol – HTTP/1.1

[5] OpenSSL toolkit
http://www.openssl.org