|  |  |
| --- | --- |
|  |  |

EPTF Applib HTTP for TitanSim, User Guide

Contents

[1 Introduction 2](#_Toc334797199)

[1.1 Revision history 2](#_Toc334797200)

[1.2 About this Document 2](#_Toc334797201)

[1.2.1 How to Read this Document 2](#_Toc334797202)

[1.2.2 Presumed Knowledge 2](#_Toc334797203)

[1.2.3 References 2](#_Toc334797204)

[1.2.4 Abbreviations 3](#_Toc334797205)

[1.2.5 Terminology 3](#_Toc334797206)

[1.3 System Requirements 4](#_Toc334797207)

[2 The HTTP Application Library 4](#_Toc334797208)

[2.1 Overview 4](#_Toc334797209)

[2.1.1 Protocol handling 4](#_Toc334797210)

[2.1.2 Architecture: 7](#_Toc334797211)

[2.1.3 Transport 8](#_Toc334797212)

[2.2 Description of files in this feature 13](#_Toc334797213)

[2.2.1 EPTF\_HTTP\_Functions 13](#_Toc334797214)

[2.2.2 EPTF\_HTTP\_Definitions 13](#_Toc334797215)

[2.2.3 EPTF\_HTTP\_Logger\_Functions 14](#_Toc334797216)

[2.2.4 EPTF\_HTTP\_Logger\_Definitions 14](#_Toc334797217)

[2.2.5 EPTF\_HTTP\_Transport\_Functions 14](#_Toc334797218)

[2.2.6 EPTF\_HTTP\_Transport\_Definitions 15](#_Toc334797219)

[2.3 Description of required files from other features 15](#_Toc334797220)

[2.4 Description of test steps 15](#_Toc334797221)

[2.4.1 EPTF\_HTTP\_Functions.ttcn 15](#_Toc334797222)

[2.5 Installation 16](#_Toc334797223)

[2.6 Configuration 16](#_Toc334797224)

[2.6.1 Binding layers 16](#_Toc334797225)

[2.6.2 EPTF related parameters 18](#_Toc334797226)

[2.6.3 EPTF related statistics 19](#_Toc334797227)

[3 Error messages 20](#_Toc334797228)

[3.1 EPTF\_HTTP\_Functions.ttcn 20](#_Toc334797229)

[3.2 EPTF\_HTTP\_Logger\_Functions.ttcn 20](#_Toc334797230)

[3.3 EPTF\_HTTP\_Transport\_Functions.ttcn 20](#_Toc334797231)

[4 Warning messages 21](#_Toc334797232)

[4.1 EPTF\_HTTP\_Functions.ttcn 21](#_Toc334797233)

[4.2 EPTF\_HTTP\_Logger\_Functions.ttcn 21](#_Toc334797234)

[4.3 EPTF\_HTTP\_Transport\_Functions.ttcn 21](#_Toc334797235)

# Introduction

## Revision history

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Rev | Characteristics | Prepared |
| 2009-01-08 | PA1 | First draft version | EAKOPER |
| 2009-01-19 | PA2 | Updated after review | EAKOPER |
| 2009-03-10 | PA3 | External template support added | EPTEDIM |
| 2009-04-28 | PA4 | Updated after bug fixing | EBENMOL |
| 2009-08-10 | PB1 | Added server functionality;  Added server side Digest authentication | EGERGFT |
| 2011-10-25 | PC1 | Functionality in the remote transport to get the target LGen component based on the URI in the request has been added. | ETHECS |
| 2012-07-24 | PD1 | Added MessageBuffer | EENDKIS |

## About this Document

### How to Read this Document

This is the User Guide for the HTTP application library. HTTP application library is developed for the TTCN-3 Toolset with TITAN. This document should be read together with Product Revision Information [5] and Function Specification [6].

### Presumed Knowledge

To use this protocol module the knowledge of the TTCN-3 language [4] is essential.

### References

1. [RFC 1945](http://www.ietf.org/rfc/rfc1945.txt)  
   Hypertext Transfer Protocol – HTTP/1.0

1. [RFC 2616](http://www.ietf.org/rfc/rfc2616.txt)  
   Hypertext Transfer Protocol – HTTP/1.1
2. [RFC 2617](http://tools.ietf.org/html/rfc2617)  
   HTTP Authentication: Basic and Digest Access Authentication
3. ETSI ES 201 873-1 v.3.1.1  
   The Testing and Test Control Notation version 3. Part 1: Core Language, July 2006
4. 109 21-CNL 113 618  
   EPTF AppLib HTTP for TitanSim, Product Revision Information
5. 155 17-CNL 113 618  
   EPTF AppLib HTTP for TitanSim, Function Specification
6. 1/1553-CRL 113 200 Uen  
   User Documentation for the TITAN TTCN-3 Test Executor
7. 155 17-CNL 113 512 Uen  
   EPTF Core Library for TTCN-3 toolset with TITAN, Function Specification.
8. Naturaldocs document for HTTP application library (\vobs\ttcn\TCC\_Releases\Libraries\EPTF\_Applib\_HTTP\_CNL113618\doc\apidoc)
9. 155 17-CNL 113 472 Uen  
   TCC Useful Functions for TTCN-3 toolset with TITAN, Function Specification.

### Abbreviations

EPTF Ericsson Performance Test Framework

FSM Finite State Machine

HTTP Domain Name System

SUT System Under Test

TTCN-3 Testing and Test Control Notation version 3

### Terminology

*Port*

A port represents connections (incoming/outgoing) or listening sockets.

*Port group*

A port group is a set of ports with a unique group identifier. A port group contains one or more ports.

*TitanSim Core (Load) Library(CLL)*

The part of the TitanSim software that is totally project independent. (I.e., which is not protocol-, or application-dependent). The TitanSim CLL is to be supplied and supported by the TCC organization. Any TitanSim CLL development is to be funded centrally by Ericsson.

*TitanSim Appliaction Library*

is the application-specific part of the TitanSim software.It provides load generation functionalities belonging to specific protocols or products.

## System Requirements

Application library modules are a set of TTCN-3 source code files that can be used as part of TTCN-3 test suites only. Hence, the modules alone do not put specific requirements on the system used. However in order to compile and execute a TTCN-3 test suite using the set of implemented modules the following system requirements must be satisfied:

* TITAN TTCN-3 Test Executor with version 1.8.pl0 or later is installed. For installation guide see [7].

# The HTTP Application Library

## Overview

The HTTP application library is part of TitanSim. Using the modules found in the application library, an HTTP message traffic generator can be implemented in the TTCN-3 language.

### Protocol handling

By means of the HTTP application library either a HTTP client or a HTTP server can be implemented.

The HTTP application library provides a container called HTTP application context for each HTTP user. An HTTP context element is going to be associated with an HTTP user (entity in TitanSim [8] terminology) after the creation of the user. The HTTP application context element stores all the necessary information to the HTTP user to create, send, receive and handle messages, maintain connections.

The information stored in an HTTP context element associated to an HTTP user is the following:

* method: HTTP method (e.g.: POST, GET etc)
* statusCode: response status code
* statusText: response status text
* URI: HTTP request URI
* Major version number
* Minor version number
* header lines pointer: an index pointing into the header lines database and uniquely identifying a sequence of HTTP header name and value pairs in the header lines database
* connection id: the unique id of an HTTP connection object
* port id: the id of the port to send the response
* authentication details: authentication details of the user
* message body: HTTP message body (it can be binary body)

After a set of HTTP users are created, their HTTP context elements will be set to default values. The HTTP application library provides getter/setter API functions for getting and setting the HTTP contexts with arbitrary data. These API functions are the following:

* f\_EPTF\_HTTP\_setEntityContext (set HTTP client context)
* f\_EPTF\_HTTP\_setEntityContextServer (set HTTP server context)
* f\_EPTF\_HTTP\_setEntityConnection
* f\_EPTF\_HTTP\_setAuthDetails4Entity
* f\_EPTF\_HTTP\_setCharstringBodyHandlerFunction4Entity
* f\_EPTF\_HTTP\_setBinaryBodyHandlerFunction4Entity
* f\_EPTF\_HTTP\_getEntityContextMethod
* f\_EPTF\_HTTP\_setEntityContextMethod
* f\_EPTF\_HTTP\_getEntityContextURI
* f\_EPTF\_HTTP\_setEntityContextURI
* f\_EPTF\_HTTP\_getEntityContextVersionMajor
* f\_EPTF\_HTTP\_setEntityContextVersionMajor
* f\_EPTF\_HTTP\_getEntityContextVersionMinor
* f\_EPTF\_HTTP\_setEntityContextVersionMinor
* f\_EPTF\_HTTP\_getEntityContextMessageHeader
* f\_EPTF\_HTTP\_setEntityContextAddCustomHeader
* f\_EPTF\_HTTP\_getEntityContextConnectionId
* f\_EPTF\_HTTP\_setEntityContextConnectionId
* f\_EPTF\_HTTP\_getHeaderContentForEntity
* f\_EPTF\_HTTP\_getBodyContentForEntity
* f\_EPTF\_HTTP\_getHeaderForEntity
* f\_EPTF\_HTTP\_setEntityContextMessageBody
* f\_EPTF\_HTTP\_getEntityContextStatusCode
* f\_EPTF\_HTTP\_setEntityContextStatusCode
* f\_EPTF\_HTTP\_getEntityContextStatusText
* f\_EPTF\_HTTP\_setEntityContextStatusText

For detailed specification of above functions, read NaturalDocs documentation [9] of HTTP application library.

For the server, authentication data is also stored. Server authentication data can be set by means of the f\_EPTF\_HTTP\_setAuthData API function.

Note that above functions require the index of the HTTP user (entity) whose HTTP context is subject to a modification. The suggested way to acquire these indices is to use the built-in API of LGenBase feature of TitanSim [8].

* f\_EPTF\_LGenBase\_entityGrpNameIndex
* f\_EPTF\_LGenBase\_getEGrpBaseOffset

The created HTTP users with properly set HTTP context elements can be used to implement an HTTP based load generator. The control API of the HTTP application library is the number of provided test steps (see section 2.4). FSMs and test steps can be applied together to realize the user specific behavior of the application library.

The application library also provides a number of statistics. See detailed introduction of these statistics in section 2.6.3.

The event dispatch mechanism can be controlled of the library by setting the tsp\_EPTF\_HTTP\_dispatchLevel test suite parameter. Valid values are OnlyReceived, Class and Detailed. If the test suite parameter is set to OnlyReceived then only message reception events will be reported to the FSMs of the HTTP based application (reception of request, response message with charstring or binary body). If the dispatch level is set to Class, then status code based HTTP response class events will be reported as well (e.g.: class 100 which means a message reception with status code in range [100..200)). The highest dispatch level is Detailed. If it is set, then message reception, HTTP class and single status code events will be dispatched also.

### Architecture:

**SUT**

TCP 1

EPTF HTTP Transport Local

**HTTP Application**

**EPTF Transport Layer**

TCP n

TCP 2

**…**

EPTF HTTP Mapper

EPTF HTTP Transport Remote

EPTF CLL Routing

HTTP AppLib

**EPTF HTTP AppLib**

HTTP Transport Local

HTTP Transport Remote

EPTF HTTP AppLib

HTTP Mapper

The HTTP application library has the above defined architecture. In the figure dotted lines denote TTCN internal or external data communication while arrows denote component hierarchy.

Section EPTF HTTP AppLib shows the location of the HTTP application library in the module hierarchy.

Section EPTF HTTP Transport explains the transport layer of the HTTP application library. The transport layer can handle multiple outgoing/incoming TCP connections (TCP i). These connections are based on the Transport CommPortIPL4 feature of TitanSim [8].

The transport layer supports both Remote and Local operation mode. The necessary components for Remote operation mode can also be seen on the picture: EPTF HTTP Transport Remote and EPTF HTTP Mapper. Note that the EPTF HTTP Mapper component depends on the Transport Routing feature of TitanSim [8].

Local mode:  
External TCP communication is located on the same component where the HTTP load generation is deployed.

Remote mode:   
External TCP communication ports are located on the EPTF HTTP Mapper component. Multiple EPTF HTTP Transport Remote load generators can connect to the EPTF HTTP Mapper trough local connection. Performance demanding operations like encoding/decoding are performed on the EPTF HTTP Transport Remote load generators and their messages are multiplexed trough the EPTF HTTP Mapper component to the SUT and in the opposite direction (with routing). This configuration makes it possible to perform load balancing by using multiple load generator PTCs in parallel. The other advantage of Remote configuration is that the EPTF HTTP Mapper component can have only one local TCP port used by multiple load generators, so the number of load generators look like a uniform system, with increased performance.

Finally, section HTTP Application in the figure above shows a possible configuration of an Application based on the HTTP application library.

### Transport

The transport layer of the HTTP application library has multiple built-in features.

The transport layer is able to maintain a number of HTTP connection objects. These connection objects are the ports.

A port group is a set of ports with a unique id. The user can perform an operation on the port group using its unique id.

**Port group**

* name
* local host info
* remote host info
* management data
* registerable functions

TCP1

TCP2

TCP3

**…**

TCPn

A port group can either operate in client or in server mode. In client mode operation, by means of the port group TCP connections can be opened towards the SUT, HTTP requests can be sent and the response can be processed. In server mode operation a listening socket can be opened, incoming connections can be accepted, requests can be handled and responses can be sent.

The following API functions can be used to manage connection objects:

* f\_EPTF\_HTTP\_LocalTransport\_newPortGroup
* f\_EPTF\_HTTP\_LocalTransport\_deletePortGroup
* f\_EPTF\_HTTP\_LocalTransport\_newPort (port group with one port)
* f\_EPTF\_HTTP\_LocalTransport\_newServerPort
* f\_EPTF\_HTTP\_LocalTransport\_deletePort
* f\_EPTF\_HTTP\_LocalTransport\_closePortOfUser
* f\_EPTF\_HTTP\_LocalTransport\_freePortOfUser
* f\_EPTF\_HTTP\_LocalTransport\_addPortToGroup\_default
* f\_EPTF\_HTTP\_LocalTransport\_removePortFromGroup\_default
* f\_EPTF\_HTTP\_LocalTransport\_closePort
* f\_EPTF\_HTTP\_RemoteTransport\_newPortGroup
* f\_EPTF\_HTTP\_RemoteTransport\_deletePortGroup
* f\_EPTF\_HTTP\_RemoteTransport\_newPort
* f\_EPTF\_HTTP\_RemoteTransport\_deletePort
* f\_EPTF\_HTTP\_RemoteTransport\_closePortOfUser
* f\_EPTF\_HTTP\_RemoteTransport\_freePortOfUser
* f\_EPTF\_HTTP\_RemoteTransport\_addPortToGroup\_default
* f\_EPTF\_HTTP\_RemoteTransport\_removePortFromGroup\_default
* f\_EPTF\_HTTP\_RemoteTransport\_closePort

For details, read NaturalDocs documentation [9] of HTTP application library.

**Client mode operation**

Client port groups have instant connection open, instant connection closed and use SSL properties.

Use SSL is reserved for future use to implement security.

Instant connection open means that before sending an HTTP message on a on a port of the port group, a connection open operation will be attempted on the given port.

Instant connection close means that after receiving a message on a port of the port group, a connection close operation will be attempted on the given port.

**Server mode operation**

Server port groups have buffering property.

In buffering mode, each server port in the group maintains a buffer for response sequencing.

#### Ports

In this section, a port is a port group that contains one port.

First, a port has to be created with the provided API. If a port is successfully created, a non-negative and unique identifier will be provided which later can be used to access the port object.

**Client port**

If a client port is created with disabled instant connection open property, then not only the port object is instanced, but the connection is opened as well (if possible). If socket error occurs during connection open a post event will be sent.

With the unique port identifier acquired during the creation of the port, the HTTP user can send a HTTP on a certain port object. If the port has been created with enabled instant connection open property, then before sending a message, a connection open operation is attempted to be applied to the port. Otherwise an explicit connection open operation must be applied to the port before sending the message. After sending on the port, it will go to Busy for sending state which means that the port cannot be used again until a response message has not arrived on it.

Connection open, connection close and connection half close operations can be applied to a port again with the acquired unique port identifier. Connection open operation opens the port connection with the parameters supplied during its creation. Connection close closes the connection of the port. Connection half close means that the connection of the port is not closed, but will not send any more messages. So in half closed state, a port can handle incoming messages but it is forced not to send any messages towards the SUT.

If an HTTP user is sending on a port and the answer message has not arrived yet, then the port is associated to the user. In this case the port of the user can be closed with the close port of user API as well or can be made free for sending with the free port of user API. In the latter case the user’s port won’t be closed but it reuse is enabled. Incoming messages are dropped until the port is reused. If socket error occurs during sending a post event will be sent to notify user.

If the port is created with enabled instant connection close property, then receiving a message on a port will automatically close the connection on the port.

The default behavior of a port object can be overdefined if different default functions are registered to the port during creation. These functions should be provided to the API of port creation. For details, read the NaturalDocs documentation [9] of HTTP application library

Finally, the port object can be removed. Removal of a port object includes closing its connection (if open). If the HTTP application library is stopped via Base feature of TitanSim [9], then all port objects will be removed automatically from the system.

**Server port**

If a server port is created with the buffering property, the responses are sent back on a given connection in the same order that the requests were received. The size of the buffer can be set with a module parameter.

If the server port is created with the instant connection close property, the incoming connection is closed after the response has been sent.

With the unique port identifier acquired during the creation of the port, a listening socket can be opened and closed.

The transport layer handles incoming connections. In case of an incoming connection, a unique port identifier for the new connection is provided by transport layer. By means of the group identifier and the port identifier the incoming connection can be closed.

#### Port groups

First, a port group has to be created with the provided API. If a port group is successfully created, a non-negative and unique identifier will be provided which later can be used to access the port group object. The difference between the creation of a port and a port group is that for a port group a number of ports, a local port step and a remote port step parameter has to be provided as well. As a result a port group will be created with a number of ports. Its local and remote port properties will be calculated according to the provided data. E.g.: suppose that an HTTP application is located on a host machine with IP address 160.100.10.100 and it intends to use 5 ports with local port numbers 2000, 2002, 2004, 2006, 2008 to send and receive messages. The server node is located on the host machine with IP address 160.100.10.101 and listens on the port 4000. In this case, during creation of the port group a local host information: { “160.100.10.100”, 2000 } and a remote host information: { “160.100.10.101”, 4000 } should be provided with the port number, local port step and remote port step parameters set to 5, 2 and 0. This configuration will automatically create the necessary communication objects to the above example.

If the port group is created with disabled instant connection open property, then not only the port group object is instanced, but all the connections are opened as well (if possible).

If the creation of a port fails during creation of the port group, then the port group will be deleted, all related data will be gracefully rolled back.

With the unique port identifier acquired during the creation of the port group, the HTTP user can send on a certain port group object. The port group will automatically select one of its ports for sending. If the port has been created with enabled instant connection open property, then before sending the message a connection open operation is attempted to be applied to the selected port of the port group. Otherwise an explicit connection open operation must be applied to the port group before sending. After sending on a port of a port group, the given port will go to Busy for sending state which means that the port cannot be used again until a response message arrives on it. Port groups are advantageous of this because sending on a port group will automatically provide the next Free port for sending and automatically administrate the state of ports.

Connection open, connection close and connection half close operations can be applied to a port group again with the acquired unique port group identifier. Connection open operation opens all the connections of ports of the port group with the parameters supplied during their creation. Connection close closes all the connections of the ports of the port group. Connection half close means that all the connections of the ports of a port group are not closed, but no port will send any more messages. So in half closed state, the ports of a port group can handle incoming messages but it is forced not to send any messages.

If an HTTP user is sending on a port of a port group and the answer message has not arrived yet, then the port is associated to the user. In this case the port of the user within a port group can be closed with the close port of user API or can be made free for sending with the free port of user API. In the latter case the user’s port won’t be closed but its reuse is enabled. Incoming messages are dropped until the port of the port group is reused.

If the port is created with enabled instant connection close property, then receiving a message on a port of a port group will automatically close the connection on the port.

Single ports can be added to or remove from port groups. The properties of these ports are calculated according to base properties of the port group. If a port is added to the port group in the example at the beginning of this section, then its local and remote connection properties will be: { “160.100.10.100”, 2010 } and { “160.100.10.101”, 4000 }. If a port is removed from the group, the one which is removed is the one that is added the most earlier to the port group. In the previous example this port is the one with the connection properties: { “160.100.10.100”, 2000 } and { “160.100.10.101”, 4000 }. This behavior can be overdefined during port group creation.

The default behavior of a port object can be overdefined if different default functions are registered to the port during creation. These functions should be provided to the API of port group creation. For details, read the NaturalDocs documentation [9] of HTTP application library

Finally, the port group object can be removed. Removal of a port group object includes closing its connections (if open). If the HTTP application library is stopped via Base feature of TitanSim [9], then all port group objects will be removed automatically from the system.

## Description of files in this feature

The EPTF Applib HTTP API includes the following files:

### EPTF\_HTTP\_Functions

#### Overview

The functions implemented in the EPTF\_HTTP\_Functions.ttcnTTCN-3 module provide the following functionality:

* + - * prepare HTTP messages
      * send encoded HTTP messages
      * receive encoded HTTP messages
      * maintain database for HTTP users
      * calculate statistics of sent/received messages

#### Dependencies

For module dependencies, read NaturalDocs documentation [9] of HTTP application library.

#### Function

For detailed function and parameter description, read NaturalDocs documentation [9] of HTTP application library.

### EPTF\_HTTP\_Definitions

The EPTF\_HTTP\_Definitions.ttcn TTCN-3 module contains only type definitions and constants used by 2.2.1.

#### Dependencies

For module dependencies, read NaturalDocs documentation [9] of HTTP application library.

#### Definitions

For detailed description of types and constants, read NaturalDocs documentation [9] of HTTP application library

### EPTF\_HTTP\_Logger\_Functions

#### Overview

The functions implemented in the EPTF\_HTTP\_Logger\_Functions.ttcn TTCN-3 module should provide logging/decoding functionality on separate component PTC.

#### Dependencies

For module dependencies, read NaturalDocs documentation [9] of HTTP application library.

#### Function

For detailed function and parameter description, read NaturalDocs documentation [9] of HTTP application library.

### EPTF\_HTTP\_Logger\_Definitions

The EPTF\_HTTP\_Logger\_Definitions.ttcn TTCN-3 module contains only type definitions and constants used by 2.2.3.

#### Dependencies

For module dependencies, read NaturalDocs documentation [9] of HTTP application library.

#### Definitions

For detailed description of types and constants, read NaturalDocs documentation [9] of HTTP application library

### EPTF\_HTTP\_Transport\_Functions

#### Overview

The module EPTF\_HTTP\_Transport\_Functions.ttcn provides the functionality of a transparent, TCP based transport layer. Layers above should work without any prior knowledge on the state of the transport layer.

#### Dependencies

For module dependencies, read NaturalDocs documentation [9] of HTTP application library.

#### Function

For detailed function and parameter description, read NaturalDocs documentation [9] of HTTP application library.

### EPTF\_HTTP\_Transport\_Definitions

The EPTF\_HTTP\_Transport\_Definitions.ttcn TTCN-3 module contains only type definitions and constants used for 2.2.5.

#### Dependencies

For module dependencies, read NaturalDocs documentation [9] of HTTP application library.

#### Definitions

For detailed description of types and constants, read NaturalDocs documentation [9] of HTTP application library

## Description of required files from other features

For product dependencies see [5].

## Description of test steps

### EPTF\_HTTP\_Functions.ttcn

This module defines the test steps of the AppLib.

The following test steps are implemented:

* f\_EPTF\_HTTP\_step\_handleTimeout
* f\_EPTF\_HTTP\_step\_sendRequest
* f\_EPTF\_HTTP\_step\_closePortOfUser
* f\_EPTF\_HTTP\_step\_freePortOfUser
* f\_EPTF\_HTTP\_step\_processResponse
* f\_EPTF\_HTTP\_step\_sendConnect
* f\_EPTF\_HTTP\_step\_sendClose
* f\_EPTF\_HTTP\_step\_sendHalfClose
* f\_EPTF\_HTTP\_step\_setEntityContextHTTPMethodCONNECT
* f\_EPTF\_HTTP\_step\_setEntityContextHTTPMethodDELETE
* f\_EPTF\_HTTP\_step\_setEntityContextHTTPMethodGET
* f\_EPTF\_HTTP\_step\_setEntityContextHTTPMethodHEAD
* f\_EPTF\_HTTP\_step\_setEntityContextHTTPMethodOPTIONS
* f\_EPTF\_HTTP\_step\_setEntityContextHTTPMethodPOST
* f\_EPTF\_HTTP\_step\_setEntityContextHTTPMethodPUT
* f\_EPTF\_HTTP\_step\_setEntityContextHTTPMethodTRACE
* f\_EPTF\_HTTP\_step\_handleRequest
* f\_EPTF\_HTTP\_step\_sendResponse
* f\_EPTF\_HTTP\_step\_closePort
* f\_EPTF\_HTTP\_step\_cleanUp

For detailed description of test steps, read NaturalDocs documentation [9] of HTTP Application Library or Functional specification.

## Installation

Since the implemented modules are used as a part of a TTCN-3 test suite, this requires TTCN-3 Test Executor be installed before the module can be compiled and executed together with other parts of the test suite. For more details on the installation of TTCN-3 Test Executor see the relevant section of [7].

If not otherwise noted in the respective sections, the followings are needed to use the HTTP application library:

* Copy the files listed in section [2.2, 2.3] to the directory of the test suite or create symbolic links to them.
* Write your own application using DNS application library.
* Implement the transport layer (or use the one delivered with the AppLib)
* Create Makefile
* Edit the configuration file according to your needs, see the following section 2.6.

## Configuration

The executable test program behavior is determined via the run-time configuration file. This is a simple text file, which contains various sections. The usual suffix of configuration files are .cfg. For further information on the configuration file see [7].

HTTP application library defines TTCN-3 module parameters as defined in [7] clause 4. Actual values of these parameters – when no default value or a different from the default actual value wished to be used – shall be given in the [MODULE\_PARAMETERS] section of the configuration file.

Parameters are defined via test suite parameters in the configuration file.

### Binding layers

Before the actual usage the HTTP Applib has to be bound to the upper and lower layers. Upper layers consist of protocols that use the HTTP Applib as a transport layer such as SOAP. The lower layer is the HTTP transport layer.

Binding the HTTP Applib to the HTTP transport layer in case of a client:

f\_EPTF\_HTTP\_LocalTransport\_init\_CT("HTTP",  
 refers(f\_EPTF\_HTTP\_messageReceived),  
 refers(f\_EPTF\_HTTP\_eventReceived),  
 refers(f\_EPTF\_HTTP\_socketErrorReceived);

f\_EPTF\_HTTP\_setSendRequestFunction(refers( f\_EPTF\_HTTP\_LocalTransport\_sendMessage));

f\_EPTF\_HTTP\_setConnectionCloseFunction(refers( f\_EPTF\_HTTP\_LocalTransport\_connectionClose));

f\_EPTF\_HTTP\_setConnectionOpenFunction(refers( f\_EPTF\_HTTP\_LocalTransport\_connectionOpen));

f\_EPTF\_HTTP\_setConnectionHalfCloseFunction(refers( f\_EPTF\_HTTP\_LocalTransport\_connectionHalfClose));

f\_EPTF\_HTTP\_setClosePortOfUserFunction(refers( f\_EPTF\_HTTP\_LocalTransport\_closePortOfUser));

f\_EPTF\_HTTP\_setFreePortOfUserFunction(refers( f\_EPTF\_HTTP\_LocalTransport\_freePortOfUser));

Binding the HTTP Applib to the HTTP transport layer in case of a server:

f\_EPTF\_HTTP\_LocalTransport\_init\_CT("HTTP",  
 refers(f\_EPTF\_HTTP\_messageReceivedServer));

f\_EPTF\_HTTP\_setSendResponseFunction(refers(f\_EPTF\_HTTP\_LocalTransport\_sendResponse));

f\_EPTF\_HTTP\_setListenFunction(refers(f\_EPTF\_HTTP\_LocalTransport\_listen));

f\_EPTF\_HTTP\_setClosePortFunction(refers(f\_EPTF\_HTTP\_LocalTransport\_closePort));

f\_EPTF\_HTTP\_setSearchContextFunction(refers(f\_HTTP\_searchServerContext));

Upper layers set the header and body content getter functions with the following setter functions:

* f\_EPTF\_HTTP\_setHeaderContentGetterFunction4Entity
* f\_EPTF\_HTTP\_setHeaderContentGetterFunction4Entities
* f\_EPTF\_HTTP\_setRawHeaderContentGetterFunction4Entity
* f\_EPTF\_HTTP\_setRawHeaderContentGetterFunction4Entities
* f\_EPTF\_HTTP\_setContentGetterFunction4Entity
* f\_EPTF\_HTTP\_setContentGetterFunction4Entities

### EPTF related parameters

Load generator parameters:

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Description** |
| tsp\_EPTF\_HTTP\_loggingEnable | Boolean | Enable/disable logging of HTTP load generator component |
| tsp\_EPTF\_HTTP\_ maxNumberOfEntities | Integer | Number of maximum entities on a single component |
| tsp\_EPTF\_HTTP\_ loggingComponentMask | Charstring | Logging component mask name |
| tsp\_EPTF\_HTTP\_dispatchLevel | EPTF\_HTTP\_ EventDispatchLevel | Event dispatch level of component (OnlyReceived, Class, Detailed) |
| tsp\_EPTF\_HTTP\_nofNonceValues | Integer | Number of nonce values used by the server |

Logger parameters:

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Description** |
| tsp\_EPTF\_HTTP\_Logging\_Decode | Boolean | Enable/disable decoding of messages received by the logger component |

Transport parameters:

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Description** |
| tsp\_EPTF\_HTTP\_Transport\_ loggingEnable | Boolean | Enable/disable logging of HTTP transport |
| tsp\_EPTF\_HTTP\_Transport\_bufferLogEnable | Boolean | Enable/disable buffer logging of HTTP transport |
| tsp\_EPTF\_HTTP\_Transport loggingComponentMask | Boolean | Logging component mask name of EPTF HTTP Transport |
| tsp\_EPTF\_HTTP\_bufferSize | Integer | The maximum size of the message buffer for storing responses |
| tsp\_EPTF\_HTTP\_Transport\_ messageBufferEnabled | Boolean | Enable/disable the HTTP message buffer |

### EPTF related statistics

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Description** |
| c\_EPTF\_HTTP\_Stat\_nofIncOptionMessages | Charstring | Number of incoming OPTIONS messages |
| c\_EPTF\_HTTP\_Stat\_nofOutOptionMessages | Charstring | Number of outgoing OPTIONS messages |
| c\_EPTF\_HTTP\_Stat\_nofIncGetMessages | Charstring | Number of incoming GET messages |
| c\_EPTF\_HTTP\_Stat\_nofOutGetMessages | Charstring | Number of outgoing GET messages |
| c\_EPTF\_HTTP\_Stat\_nofIncHeadMessages | Charstring | Number of incoming HEAD messages |
| c\_EPTF\_HTTP\_Stat\_nofOutHeadMessages | Charstring | Number of outgoing HEAD messages |
| c\_EPTF\_HTTP\_Stat\_nofIncPostMessages | Charstring | Number of incoming POST messages |
| c\_EPTF\_HTTP\_Stat\_nofOutPostMessages | Charstring | Number of outgoing POST messages |
| c\_EPTF\_HTTP\_Stat\_nofIncPutMessages | Charstring | Number of incoming PUT messages |
| c\_EPTF\_HTTP\_Stat\_nofOutPutMessages | Charstring | Number of outgoing PUT messages |
| c\_EPTF\_HTTP\_Stat\_nofIncDeleteMessages | Charstring | Number of incoming DELETE messages |
| c\_EPTF\_HTTP\_Stat\_nofOutDeleteMessages | Charstring | Number of outgoing DELETE messages |
| c\_EPTF\_HTTP\_Stat\_nofIncTraceMessages | Charstring | Number of incoming TRACE messages |
| c\_EPTF\_HTTP\_Stat\_nofOutTraceMessages | Charstring | Number of outgoing TRACE messages |
| c\_EPTF\_HTTP\_Stat\_nofIncConnectMessages | Charstring | Number of incoming CONNECT messages |
| c\_EPTF\_HTTP\_Stat\_nofOutConnectMessages | Charstring | Number of outgoing CONNECT messages |
| c\_EPTF\_HTTP\_Stat\_nofIncXXXStatusCode | Charstring | Number of incoming messages with status code of XXX |
| c\_EPTF\_HTTP\_Stat\_nofOutXXXStatusCode | Charstring | Number of outgoing messages with status code of XXX |

# Error messages

## EPTF\_HTTP\_Functions.ttcn

* Unhandled HTTP method: <METHOD>
  + The HTTP user’s context element refers to an invalid HTTP method

## EPTF\_HTTP\_Logger\_Functions.ttcn

None exists.

## EPTF\_HTTP\_Transport\_Functions.ttcn

* Group name: <NAME> is not unique
  + The provided port / port group name is not unique. If name is not necessary, then omitting it during creation of the port or port group will automatically generate an unique name.

# Warning messages

## EPTF\_HTTP\_Functions.ttcn

* Status code of incoming message is invalid: <Status code>
  + The status code of the message is outside of valid range [100 .. 600)
* Entity cannot be determined to message on session: <SessionId>
  + A message has been received to which no HTTP user belongs. The most possible cause is that the port related to the message reception is not assigned to an HTTP user yet.
* Entity cannot be determined to event: <EventId> on session: <SessionId>
  + An event has been received to which no HTTP user belongs. The most possible cause is that the port related to the event reception is not assigned to an HTTP user yet. E.g. the port is opened but it will be assigned to an entity only after one sends on it and the connection closed event from the remote side occurs before sending.
* Entity cannot be determined to socket error on session: <SessionId>
  + A socket error message has been received to which no HTTP user belongs. The most possible cause is that the port related to the message reception is not assigned to an HTTP user yet (it occurs mainly on instant connection opens).
* You should set your sender function with f\_EPTF\_HTTP\_ setSendRequestFunction
  + The send request function is not yet set
* Setting invalid template parameter
  + The user can add optional template parameters to the pre-defined template type. Before getting the template content, values must be assigned to each optional parameter. This warning indicates that optional parameter could not be found in the indexed database.
* Unhandled message type received <message>
  + The registered message handler could not handle the incoming message

## EPTF\_HTTP\_Logger\_Functions.ttcn

None exists.

## EPTF\_HTTP\_Transport\_Functions.ttcn

* Cannot add/remove ports to port groups created in Port mode
  + Addition of ports to a port object is disabled
* Port selection is unsuccessful for group: <GroupId>
  + If a port is occupied for sending or a port group occupies all its ports for sending, then no more ports can be selected on the object to send.
* Group already deleted: <GroupId> - <GroupName>
  + The port or port group subject to the previous operation is already deleted
* Already sending on group: <GroupId> - <GroupName>
  + The port object is already occupied for sending. New send is not possible till previous send operation is not finished.
* Connection id was not found in Connection id to port id hash map. Message dropped!
  + The message arrives on a port or on a port of a port group to which no group and port id is associated. This should be an error.
* No more free ports available
  + It is not possible to send on the given port or port group object because all ports are Busy for sending.
* Connection is not in Idle state, state: <State>
  + Instant connection open is attempted on port but it is not in Idle state (so it is not closed, cannot be reopened).
* Instant connection opening is unsuccessful on port at: <Group and Port info>
  + Instant connection opening was attempted to a port or to a port of a port group, but the operation failed.
* Sending on port which has already been removed! <GroupId> / <PortId>
  + Trying to send on a port which existed once but it has already been removed. E.g. a port is removed on-the-fly from a port group.
* Connection is not Opened state, state: <State>
  + Trying to send on a port or on a port of a port group, but its state is not Opened.
* Port already closed: <Group and Port info>
  + A message is received with activated instant connection close mode, but the port cannot be closed as it is already closed.
* Connection opening is unsuccessful for port (<Result>) at <Group and port info>
  + Open connection operation failed for a port or a port in a port group
* Connection already opened: <Group and Port info>
  + The open connection operation was applied to a port which is already opened.
* Port 0 is already deleted from group: <GroupId>
  + The port to be removed is already deleted.
* Connection (<ConnectionId>) is already closed for port at <Group and port info>
  + The port in the port group is already closed
* Connection already closed
  + The port is already closed.
* Connection already half closed
  + The port is already in half closed state
* Group does not exist.
  + The port or port group to which the previous operation referred does not exist
* Invalid group index
  + The port or port group index used by the previous operation is invalid
* Creation of port group failed for group: <Group info>
  + The creation of port group failed. This message is received in case of remote port group creation.
* Creation of port failed for port: <Port info>
  + The creation of port failed. This message is received in case of remote port creation.
* Wrong port group index: <GroupId>
  + The port or port group id used for addition/removal/close/halfclose of a port is invalid. This message is received in case of remote port creation.
* Port addition to group failed: <GroupInfo>
  + Addition of a port to a port group failed. This message is received in case of remote port creation.
* Open connection is unsuccessful for port group: <GroupId>
  + Open connection operation on group failed. This message is received in case of remote port creation.
* A component reference for session id (<SessionId>) is already stored! Message not sent.
  + The session id associated for the message is already used. Without a valid session id the message cannot be sent.
* Session id (<SessionId>) is not found in the hash map! Possibly connection closed remotely!
  + The session id is not found in the routing database hash map. This possibly indicates that the connection has been closed remotely to ports to which a session id is not yet associated.
* Could not find port identifier!
  + In case of sending a response, a unique port identifier must be provided by the HTT user that identifies the connection to send the response. The warning probably means that the connection has been closed.
* Error while sending buffered responses.
  + When the client pipelines the requests, responses are buffered in the server so they can be sent in the order that the requests were received. This warning indicates that the buffered responses could not be sent out properly.
* Send message unsuccessful! <GroupInfo>
  + Socket error, message could not be sent properly.
* Listening unsuccessful for port (<error>) at <GroupInfo>
  + Listening operation unsuccessful.
* Error removing incoming connection: port(<portId>) is not busy : <GroupInfo>
  + Internal error.

Buffer warnings:

* Re-using slot! Replacing old message with new.
  + Internal error. The application library tries to store a response with a busy sequence number. A sequence number is busy if the response associated with the sequence number is still in the buffer. The new message will overwrite the old one.
* Buffer is full, dropping message: <HTTPmessage>
  + Response could not be stored since the message buffer is full. The message will not be processed.
* Error while sending buffered responses, clearing buffer.
  + Sending error. It can mean either a socket error or that the socket has already been closed. All of the buffered messages will be deleted.
* Inconsistency in message buffer!
  + The ring buffer is not consistent with the message buffer. Either a sequence number could not be found in the message buffer or the given message buffer slot is not busy. All of the buffered messages will be deleted.