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

Contents

[1 About this Document 3](#_Toc479714166)

[1.1.1 How to Read this Document 3](#_Toc479714167)

[1.1.2 Presumed Knowledge 3](#_Toc479714168)

[2 Functionality 3](#_Toc479714169)

[2.1 Implemented protocols 3](#_Toc479714170)

[2.2 Non-standard wildcarded uri support. 3](#_Toc479714171)

[2.3 Routing Functionality 3](#_Toc479714172)

[2.4 Modified and non-implemented Protocol Elements 4](#_Toc479714173)

[2.5 Ericsson-specific changes 4](#_Toc479714174)

[2.6 Backward incompatibilities 4](#_Toc479714175)

[2.7 System Requirements 4](#_Toc479714176)

[3 Feature list 4](#_Toc479714177)

[3.1 Message handling 4](#_Toc479714178)

[3.1.1 Encoding messages 4](#_Toc479714179)

[3.1.2 Decoding messages 5](#_Toc479714180)

[3.2 Network handling 5](#_Toc479714181)

[3.2.1 Source port number of the sent messages 5](#_Toc479714182)

[3.2.2 Basic mode 5](#_Toc479714183)

1. [3.2.2.1 Start-up operation 6](#_Toc479714184)
2. [3.2.2.2 Accepting TCP connections 6](#_Toc479714185)
3. [3.2.2.3 Receive messages 6](#_Toc479714186)
4. [3.2.2.4 Sending messages 6](#_Toc479714187)

[3.2.3 Advanced mode 6](#_Toc479714188)

1. [3.2.3.1 Start-up operation 7](#_Toc479714189)
2. [3.2.3.2 Accepting TCP connections 7](#_Toc479714190)
3. [3.2.3.3 Receive messages 7](#_Toc479714191)
4. [3.2.3.4 Sending messages 7](#_Toc479714192)
5. [3.2.3.5 Transport error handling 7](#_Toc479714193)

[4 Protocol Modules 7](#_Toc479714194)

[4.1 Overview 7](#_Toc479714195)

[4.2 Installation 8](#_Toc479714196)

[4.2.1 Description of the files in the package 9](#_Toc479714197)

1. [4.2.1.1 SIPmsg\_Types.ttcn and SIPmsg\_PortType.ttcn 9](#_Toc479714198)
2. [4.2.1.2 SIPmsg\_PT.hh, SIP\_parse.h 9](#_Toc479714199)
3. [4.2.1.3 SIPmsg\_PT.cc, SIP\_parse\_.tab.c, SIP\_parse\_.tab.h, lex.SIP\_parse\_.c 9](#_Toc479714200)
4. [4.2.1.4 SIP\_parse.l, SIP\_parse.y 9](#_Toc479714201)

[4.3 Configuration 9](#_Toc479714202)

[4.3.1 SIP parameters in the configuration file 9](#_Toc479714203)

1. [4.3.1.1 local\_sip\_port 10](#_Toc479714204)
2. [4.3.1.2 default\_local\_adress 10](#_Toc479714205)
3. [4.3.1.3 default\_sip\_protocol 10](#_Toc479714206)
4. [4.3.1.4 default\_dest\_port 10](#_Toc479714207)
5. [4.3.1.5 default\_dest\_address 10](#_Toc479714208)
6. [4.3.1.6 length\_calculation 10](#_Toc479714209)
7. [4.3.1.7 listen\_enabled 10](#_Toc479714210)
8. [4.3.1.8 debug 11](#_Toc479714211)
9. [4.3.1.9 ASP\_or\_MSG 11](#_Toc479714212)
10. [4.3.1.10 error\_mode 11](#_Toc479714213)
11. [4.3.1.11 header\_format 11](#_Toc479714214)
12. [4.3.1.12 raw\_mode 11](#_Toc479714215)
13. [4.3.1.13 port\_mode 12](#_Toc479714216)
14. [4.3.1.14 multiple\_headers 12](#_Toc479714217)
15. [4.3.1.15 MTU\_size 12](#_Toc479714218)
16. [4.3.1.16 random\_udp\_sending\_port 12](#_Toc479714219)
17. [4.3.1.17 transport\_error\_reporting 12](#_Toc479714220)
18. [4.3.1.18 IPv6enabled 13](#_Toc479714221)
19. [4.3.1.19 wildcarded\_uri 13](#_Toc479714222)
20. [4.3.1.20 SIPmsg\_binary\_body\_mode It controls where the body is decoded when using the decoder function f\_SIP\_decode\_binary. 13](#_Toc479714223)

[4.3.2 SIPmsg Moduleparmeters 13](#_Toc479714224)

1. [4.3.2.1 tsp\_SIPmsg\_ipv6enabled 13](#_Toc479714225)

[4.4 Upgrading from previous versions 13](#_Toc479714226)

[5 ASP’s and messages 14](#_Toc479714227)

[5.1 Message based interface 14](#_Toc479714228)

[5.2 ASP interface 14](#_Toc479714229)

[6 Encoder decoder functions 16](#_Toc479714230)

[6.1 Encoder function 16](#_Toc479714231)

[6.2 Decoder function 17](#_Toc479714232)

[7 Error messages 18](#_Toc479714233)

[7.1.1 Structure of Error Messages 18](#_Toc479714234)

[7.1.2 Error Messages 18](#_Toc479714235)

[8 Warning messages 19](#_Toc479714236)

[9 Examples 21](#_Toc479714237)

[9.1 Script to modify Makefile (for development purposes) 21](#_Toc479714238)

[10 How to find the faulty header in case of parse error 21](#_Toc479714239)

[10.1 Identify the error 22](#_Toc479714240)

[10.2 Example 22](#_Toc479714241)

[11 Terminology 23](#_Toc479714242)

[11.1 Abbreviations 23](#_Toc479714243)

[12 References 24](#_Toc479714244)

# About this Document

### How to Read this Document

This is the Desciption for the SIPmsg test port. The SIPmsg test port is developed for the TTCN-3 Toolset with TITAN. This document should be read together with Product Revision Information [2].

### Presumed Knowledge

The knowledge of the TITAN TTCN-3 Test Executor [3] and the TTCN-3 language [1] is essential.

For configuration of the SIPmsg test port knowledge of SIP [4] is of importance.

# Functionality

The SIPmsg test port implements the message structure of the SIP protocol in a formalized way, using the standard specification language TTCN-3. It also implements the connection between test suite and SUT using TCP/IP or UDP/IP protocol.

## Implemented protocols

The test port implements all protocol messages and elements described in RFC3261, RFC2806, RFC2976, RFC3262, RFC3311, RFC3323, RFC3325, RFC3326, RFC3265, RFC3455, IETF Draft draft-ietf-dip-session-timer-15.txt, RFC3428, RFC3515, RFC3841, RFC3313, RFC3327, RFC3329, RFC3603, RFC3608, RFC3891, RFC3892, RFC3903, RFC3911, RFC 3420 [28], IETF Draft draft-levy-sip-diversion-08.txt, RFC4244, RFC4488, draft-ietf-sip-refer-with-norefersub-04, draft-allen-sipping-poc-p-headers-01, RFC5009, draft-kaplan-sip-session-id-02, RFC5502, RFC5002, RFC4457, RFC4412, draft-ott-sip-serv-indication-notification-00, RFC6442, Indication of features supported by proxy draft-holmberg-sipcore-proxy-feature-04, RFC4538, RFC6086, RFC6050,FEATURE CONCEPT STUDY ICBS and Flexible Charging Support for Japan

CPM Conversation Functions [47]

## Non-standard wildcarded uri support.

The decoder supports a wildcarded URI format described in. The support is controlled by a test port and function parameters.

## Routing Functionality

Routing functionality is not performed.

## Modified and non-implemented Protocol Elements

-

## Ericsson-specific changes

There is no Ericsson specific change in this product.

## Backward incompatibilities

-

## System Requirements

In order to operate the SIPmsg test port the following system requirements must be satisfied:

* TITAN TTCN-3 Test Executor R7A (1.7.pl0) or higher installed. For installation guide see [3]. Please note: This version of the test port is not compatible with TITAN releases earlier than R7A. The usage of TITAN releases earlier than R8A is not recommended because this version of the test port is prepared to handle the big integer numbers which feature is introduced in TITAN R8A. The usage of TITAN releases earlier than R8A can result a dynamic test case error.
* *gcc*, *make, makedepend* utilities installed
* Network interface

# Feature list

## Message handling

The test port can handle SIP request and SIP response messages and it can use both UDP and TCP connection to send and receive messages.

### Encoding messages

The built in encoder can encode SIP request and SIP response messages and it is possible to send raw and fragmented [28] messages through the test port.

The encoding consists of three steps:

1. Encoding the request or response line
2. Encoding headers
3. Adding message body.

The name of the header can be encoded in short or long format. Multiple header fields can be encoded as a comma separated list or several header rows. The behaviour of the test port is controlled by test port parameter.

### Decoding messages

After all headers are received the messages are parsed. The parser is implemented using Bison and Flex. The parser accepts all valid message formats.

The error behaviour of the test port is controlled by test port parameters. The test port can ignore any decoding errors and discard the message silently or pass the erroneous message to the test case as a RAW message or issue an error.

The parser can be deactivated by test port parameter. In that case the received messages are passed in raw format to the test case.

After the parsing is finished the message body is extracted from the buffer if it exists.

## Network handling

The test port has two different network handling method:

* Basic mode
* Advanced mode

The local host name and port number can be set in both modes.

### Source port number of the sent messages

When the SIP test port sends UDP packets the source port number can be either the listening port number or a random port number chosen by the operating system. The behaviour of the test port is controlled by the parameter 'random\_udp\_sending\_port'.   
  
When the SIP test port establishes a TCP connection the source port is always selected by the operating system because it is not possible to open more than one TCP connection from one TCP port.

### Basic mode

In basic mode the test port can handle only one TCP connection or one UDP socket. It is not possible to send and receive messages using both protocols at the same time, but the test port can switch between protocols and remote hosts.

#### Start-up operation

During the map operation the test port checks the default protocol and the default target host name. If the default target host name is specified in the config file the port establishes connection with the specified host. That means the port opens a TCP connection if the default protocol is TCP or the port opens a UDP socket if the default protocol is UDP.

If no default target host is specified the port opens a listening socket (UDP or TCP according to the default sip protocol) if listening is enabled.

#### Accepting TCP connections

The test port can accept only one incoming TCP connection if listening is enabled. After the port accepts the incoming connection request, the listening socket will not be closed but the port also will not accept new connections until the current TCP connection is closed.

#### Receive messages

The port can receive messages through the opened TCP connection or on the opened UDP socket.

In the case of TCP the test port uses a buffer to assemble the whole message if it is received in multiple TCP packets.

#### Sending messages

When sending of a message is requested the port first compares the parameters (target host name, port number and protocol) of the current connection with the requested parameters.

If there is any difference in the parameters the test port closes the current connection and opens a new one.

If the size of the message exceeds the MTU size and the protocol applied is UDP the messages will not be sent and a TTCN error will be generated. The MTU size check can be disabled. The default MTU length is 1300 octets according to [4] 18.1.1.

### Advanced mode

In advanced mode the test port can handle several TCP connections and listen on both UDP and TCP ports at the same time.

Each connection is distinguished by the protocol id, remote host name and the remote port number.

Any connection including the listening sockets can be opened and closed during run time using the ASP\_SIP\_open and ASP\_SIP\_close messages.

#### Start-up operation

During map operation the test port opens the listening socket according to the listen settings. The port opens a UDP socket or TCP listening socket or both.

If the target host name is specified and the default protocol is TCP the test port opens a TCP connection to the specified host.

#### Accepting TCP connections

The port can accept any number of connection requests on the listening port. After accepting a connection the port is ready to send and receive messages over it and it is able to accept a new one.

#### Receive messages

The port can receive messages through any opened TCP connections or on the opened UDP socket at the same time

In case of TCP the test port uses a buffer to assemble a full message if it is received in multiple TCP packets. Every TCP connection has its own buffer.

#### Sending messages

When sending of a message is requested the port first compares the parameters (target host name, port number and protocol) of the current connections with the requested parameters.

If there is no open connection towards the requested host and port the test port will open a new one.

If the size of the message exceeds the MTU size and the protocol applied is UDP the messages will not be sent and a TTCN error will be generated or an ASP\_SIP\_error will be sent to the test case. The MTU size check can be disabled. The default MTU length is 1300 octets according to [4] 18.1.1.

#### Transport error handling

The test port is able to generate TTCN error or send ASP\_SIP\_error message to the test case in the case of the transport layer error (eg. send or receive operation failed). The test port behaviour is configured via run time configuration file.

# Protocol Modules

## Overview

The SIPmsg test port provides a connection between the executable test suite and the system under test. The test port opens IPv4 sockets, closes the sockets, encodes and sends SIPmsg messages through the socket and decodes the received SIPmsg messages.

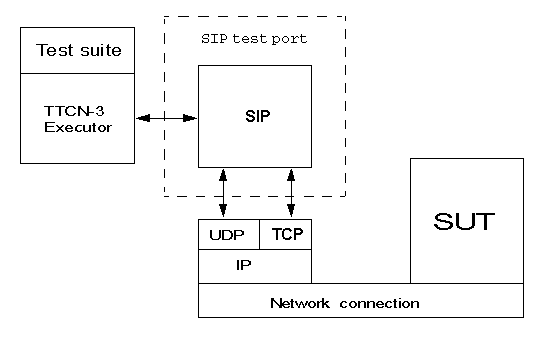


Figure 1: The overview of the test system using SIPmsg test port

## Installation

Since the SIPmsg test port is used as a part of the TTCN-3 test environment this requires TTCN-3 Test Executor to be installed before any operation of the SIPmsg test port. For more details on the installation of TTCN-3 Test Executor see the relevant section of [3].

The SIPmsg test port install package is available via ClearCase, from the following vob:

/vobs/ttcn/TCC\_Release/TestPorts/SIPmsg\_CNL113319

The package contains

* one or more TTCN-3 files with the messages and attributes sup­ported by the test port
* one or more C++ header and source files using the test port template as described in [3].

The packages include the following files:

SIPmsg\_Types.ttcn

SIPmsg\_PortTypes.ttcn

SIP\_parse.h

SIP\_parse.l

SIP\_parse.y

SIP\_parse\_.tab.c

SIP\_parse\_.tab.h

lex.SIP\_parse\_.c

SIPmsg\_PT.hh

SIPmsg\_PT.cc

Note: All files need to be added to the .prj project file in TITAN GUI, except the .l and .y files.

### Description of the files in the package

#### SIPmsg\_Types.ttcn and SIPmsg\_PortType.ttcn

Contains the TTCN-3 structure of the SIP messages, the defini­tions of message types, attributes, parameters, types and ports. The user should never change this file.

#### SIPmsg\_PT.hh, SIP\_parse.h

C++ header file of the SIPmsg test port. The user should never change this file.

#### SIPmsg\_PT.cc, SIP\_parse\_.tab.c, SIP\_parse\_.tab.h, lex.SIP\_parse\_.c

The files containing the implementation of the SIPmsg test port. The user should never change this files.

#### SIP\_parse.l, SIP\_parse.y

Parser and lexer source files. Not needed for the compilation of the test port. Used to generated SIP\_parse\_.tab.c, SIP\_parse\_.tab.h, lex.SIP\_parse\_.c files. The user should never change this files.

## Configuration

The executable test program behaviour is determined via the run-time configuration file. This is a simple text file, which contains various sections (e.g. [TESTPORT\_PARAMETERS]) after each other. The usual suffix of configuration files is .cfg. For further information on the configuration file see [3].

### SIP parameters in the configuration file

The test port parameters section is introduced by the keyword [TESTPORT\_PARAMETERS].

In this section you can specify parameters that are passed to the SIPmsg test port. Each parameter definition consists of a component name, a port name, a parameter name and a parameter value. The component name can be either an identifier or a component reference (integer) value. The port and parameter names are identifiers while the parameter value must be always a charstring (with quotation marks). Instead of component name or port name (or both of them) the asterisk ("\*") sign can be used, which means "all components" or "all ports of the component".

All parameters are optional. Because the listening is not enabled by default it should be enabled with parameter listen\_enabled.

The following parameters are allowed:

#### local\_sip\_port

The UDP/TCP port number used for incoming SIP messages. Default value is *“5061”*.

#### default\_local\_adress

The address of the local network interface on which the local UDP or TCP port is opened. If it is not specified the test port will use any of the interfaces.

#### default\_sip\_protocol

The default transport protocol used by SIP. Allowed values: *TCP, UDP*. It can be changed during the test. Default value is *UDP.*

#### default\_dest\_port

The default UDP/TCP port number of the remote host, used for outgoing SIP messages. It can be changed during the test. Default value is *“5060”*.

#### default\_dest\_address

The name or the IP address of the remote host. If it is supplied the test port automatically connects to the host when mapped.

#### length\_calculation

If enabled the test port automatically calculates the value of the Content-Length header if the original value of the header is zero. There is no automatic calculation of the Content-Length header if the sent value is differing from zero or the length calculation is disabled.

Allowed values:

* “disabled“
* “enabled“ Default value

#### listen\_enabled

If enabled the test port opens a listening socket and ready to accept incoming requests after mapped. The parameter has only effect during mapping. The listening ports can be opened and closed in advanced mode with ASP\_SIP\_open regardless of the value of the listen\_enabled parameter.

Allowed values in basic mode: *“Enabled“,“Disabled“*. Default value is *“Disabled“*.

Allowed values in advanced mode:

* “disabled“ Default value
* “enabled“ The listening is enabled on both UDP and TCP.
* “TCP\_only“ The listening is enabled only on TCP.
* “UDP\_only“ The listening is enabled only on UDP.
* Note: see clause 4.3.1.12.

#### debug

If enabled the test port will log some debug and miscellaneous information. Allowed values: *”enabled”, ”disabled”*. Default value is *”disabled”*.

#### ASP\_or\_MSG

Determines the usage of test port interface. If it is set to *“ASP”* than the test port will use the "ASP interface and if it is set to *“MSG”* the test port will use the MSG interface. The default value is *“MSG”.*

#### error\_mode

Determines the error behaviour of the test port. Possible values:

* “ignore” The test port ignore any erroneous messages and discard them without notice.
* “Warning” The test port will issue a warning if erroneous message received and pass the message as RAW message to the test case.
* “error” The test port will generate error if erroneous message received.

The default value is “*error”.*

#### header\_format

Determines if the test port use the long or the sort format of the SIP header during encoding. Possible values:

* “short” The test port will use the short format of the SIP header.
* “long” The test port will use the long format of the SIP header.

The default value is *“long”.*

#### raw\_mode

In raw mode the decoding of the message is disabled and all received SIP messages are passed to the test case as raw messages.

Allowed values: *“Enabled”, “Disabled”*. Default value is *“Disabled”*.

#### port\_mode

Determines the network handling of the port.

* “basic” In the basic mode the test port can handle only one TCP connection or one UDP socket. It is not possible to send and receive the messages using both protocols at the same time, but the test port can switch between protocols and remote hosts.
* “advanced” In advanced mode the test port can handle several TCP connections and listen on both UDP and TCP at the same time.

The default mode is the *“basic”* mode.

#### multiple\_headers

Multiple header fields can be encoded as a comma separated list or several header rows.

* “enabled” The multiple header fields encoded as several header rows.
* “disabled” The multiple header fields encoded as comma separated list.

The default value is *“disabled”*.

#### MTU\_size

Defines the used MTU size. The MTU size checking can be disabled if the MTU\_size is set to *“disabled”*.

The default value is *“1300”*.

#### random\_udp\_sending\_port

When the SIP test port sends UDP packets the source port number can be either the listening port number or a random port number chosen by the operating system.

* “enabled” The UDP source port is selected by the operating system.
* “disabled” The UDP source port is the listening port.

The default value is *“disabled”*.

#### transport\_error\_reporting

This parameter controls the transport error reporting behaviour of the test port.

* “enabled” *The test port use ASP\_SIP\_error ASP to report transport errors.*
* “disabled” *The test port will generate TTCN error in the case of the transport error*.

The default value is *“disabled”*.

#### IPv6enabled

When set to *“false”,* ip addresses are handled as is. Otherwise, IPv6 addresses are enclosed in [].  
The default value of the parameter is *“true”.*

#### wildcarded\_uri

Enables or disables the support of the wildcarded URI format.

* “enabled” The wilcarded URI support is enabled.
* “disabled” The wilcarded URI support is disabled.

The default value is *“disabled”*.

#### SIPmsg\_binary\_body\_mode It controls where the body is decoded when using the decoder function f\_SIP\_decode\_binary.

* “COMPATIBLE” The body is always returned in field messageBody regardless the presence of 8 bit binary octets.
* “AUTOMATIC” The messageBody is used if the body doesn't contain binary octets. The payload is used if the body contains any binary octets.
* “PAYLOAD” The body always returned in field payload regardless the presence of 8 bit binary octets.
* “BOTH” The body is copied into the both fields.

### SIPmsg Moduleparmeters

#### tsp\_SIPmsg\_ipv6enabled

The module parameter is used in the standalone encoding/decoding functions and controls the ipv6 support when transforming the messages. Default value is true.

## Upgrading from previous versions

Few changes are possibly needed on the existing test suites to upgrade to the new version of the test port if a new headers have been added to the new version.

In order to avoid the continuous update of the templates, the SIP test port provides a template (t\_SIP\_msgHeader\_any) and a constant (c\_SIP\_msgHeader\_empty) which can be used as a base of the template structure of the test suite.

The new header fields are listed in the Product Revision Infromation document [3].

# ASP’s and messages

The test port has a message based and an ASP based interface for sending and receiving SIP messages.

The test case can use both interfaces simultaneously to send messages and control connections on different test port instances, but a single test port can use message based interface or ASP interface to communicate with the test case. The test port behaviour is determined by the parameter ASP\_or\_MSG.

## Message based interface

The following messages can be used:

* PDU\_SIP\_Request Carries a SIP request message.
* PDU\_SIP\_Response Carries a SIP response message.
* PDU\_SIP\_Raw Raw messages. It contains the SIP message in encoded form.

## ASP interface

The following ASP’s can be used in both basic and advanced modes (see clause 4.3.1.13):

* ASP\_SIP\_Request
* ASP\_SIP\_Response
* ASP\_SIP\_Raw

All ASP’s contain an address field. The remote\_host, remote\_port and the protocol determines the target of the message. If any of it is omitted the specified default value will be used. If the address field completely omitted the messages will be sent on the last used connection.

The following ASP’s can be used in advanced mode only:

* ASP\_SIP\_open Open a new connection. The remote\_host and remote\_port fields determine the remote side and the protocol field specifies the transport protocol to be used.   
  To open a new connection towards the SUT the remote\_host must be specified. If the remote\_port or protocol are omitted default values will be used.  
  If the remote\_host is not specified the port will open a new listening socket. The listening socket can be opened regardless of the value of listen\_enabled parameter.
* ASP\_SIP\_close Close the connection. All parameter of the connection (remote\_host, remote\_port and protocol) should be specified to identify the connection to be closed.  
  If only the protocol field is defined the port will close the listening socket on the defined protocol.
* ASP\_SIP\_error This ASP used by the test port to report transport layer errors instead of generating TTCN error. The test port parameter ‘transport\_error\_reporting’ controls the usage of the ASP.  
  The test port fills the error\_code field with the error code of the operating system. The error\_text field holds the textual representation of the error code. The failed\_operation field contains the failed operation. The possible values:

|  |  |
| --- | --- |
| SIP operation | Description |
| SIP\_OP\_CREATE\_SOCKET | Socket creation failed |
| SIP\_OP\_SET\_SOCKET\_OPT | setsockopt system call failed |
| SIP\_OP\_SOCKET\_FCNTL | fcntl system call failed |
| SIP\_OP\_SOCKET\_BIND | bind system call failed |
| SIP\_OP\_SOCKET\_LISTEN | listen system call failed |
| SIP\_OP\_SOCKET\_CONNECT | connect system call failed |
| SIP\_OP\_ACCEPT | accept system call failed |
| SIP\_OP\_RECEIVE | receive system call failed |
| SIP\_OP\_LONG\_MESSAGE | The SIP message is too long to send over UDP. |
| SIP\_OP\_SEND | send system call failed |

The addr field contains the source and target address of the failed operation when the information is available. The SIP\_message field is filed with the encoded SIP message if the error is occurred during the sending of the message, after the encoding.

# Encoder decoder functions

The following functions are available to encode SIP messages into charstring/octetstring or decode charstring/octetstring that contains SIP message.

## Encoder function

external function f\_SIP\_encode(in PDU\_SIP pdu) return charstring;

Encodes a PDU\_SIP value into a charstring using long header names and multiple header fields are encoded as a comma separated list.

external function f\_SIP\_encode\_binary(in PDU\_SIP pdu) return octetstring;

Encodes a PDU\_SIP value into an octetstring using long header names and multiple header fields are encoded as a comma separated list. (To be used when binary message is carried in PDU\_SIP.)

external function f\_SIP\_encode\_formatted(  
 in PDU\_SIP pdu,   
 in boolean short\_headers,  
 in boolean multiple\_headers  
 in Boolean ipv6enabled) return charstring;

Encodes a PDU\_SIP value into a charstring. If the short\_headers parameter is “true” the header names encoded in short form and if it is “false” the header names encoded in long form.

If the multiple\_headers parameter is “true” the multiple headers encoded as several header rows and if it is “false” the multiple header fields encoded as comma separated list.

The ipv6enabled means the same as the Test Port parameter IPv6enabled, see 4.3.1.18.

external function f\_SIP\_encode\_formatted\_binary(  
 in PDU\_SIP pdu,   
 in boolean short\_headers,   
 in boolean multiple\_headers,  
 in boolean ipv6enabled) return octetstring;

The above function works as f\_SIP\_encode\_formatted but encodes PDU\_SIP into an octetstring. (To be used when binary message is carried in PDU\_SIP.)

The following external functions are capable of handling fragmented SIP messages (according to RFC 3420) and can be used the same way as the previous ones, with the difference that they use values of type PDU\_SIP\_Fragment.

external function f\_SIP\_encode\_fragment(in PDU\_SIP\_Fragment pdu) return charstring;

external function f\_SIP\_encode\_fragment\_formatted(

in PDU\_SIP\_Fragment pdu,

in boolean short\_headers,

in boolean multiple\_headers  
in Boolean ipv6enabled) return charstring;

## Decoder function

external function f\_SIP\_decode(in charstring pdu,   
in Boolean ipv6enabled, in boolean wildcarded\_uri) return PDU\_SIP;

external function f\_SIP\_decode\_backtrack(in charstring pdu, out PDU\_SIP msg  
in Boolean ipv6enabled, in boolean wildcarded\_uri) return integer;

Decodes encoded SIP messages. If the supplied message contains syntax errors the decoder function returns raw SIP messages.

external function f\_SIP\_decode\_binary  
 (in octetstring pdu,   
 in Boolean ipv6enabled,   
 in boolean wildcarded\_uri,   
 in SIPmsg\_body\_handling\_modes  
 body\_mode) return PDU\_SIP;

external function f\_SIP\_decode\_binary\_backtrack  
 (in octetstring pdu,  
 out PDU\_SIP msg,   
 in Boolean ipv6enabled,   
 in boolean wildcarded\_uri,   
 in SIPmsg\_body\_handling\_modes  
 body\_mode) return integer;

Decodes encoded SIP messages from octetstring format. (To be used when binary message is carried in PDU\_SIP.)

Similarly, the following external function works with PDU\_SIP\_Fragment:

external function f\_SIP\_decode\_fragment(in charstring pdu, in Boolean ipv6enabled) return PDU\_SIP\_Fragment;

# Error messages

### Structure of Error Messages

The structure of an error message is

<*Time stamp*> Dynamic test case error: <*error message*>

The error messages - listed below - are considered to be self-explan­atory. However an error explanation and/or some advice, how to try to solve the problem that caused the error, have been added after some error messages.

### Error Messages

**Listening socket creation failed.** or   
**Socket creation failed.**

This is an unexpected error returned by the UNIX operating system. A typical reason for such errors is that you have run out of some re­sources, like file descriptors in your machine.

**Listening socket bind failed.** or **Socket bind failed.**

The binding of the file descriptor to the IP address was unsuccessful. Check the local IP address.

**Fcntl error.**

This is an unexpected error returned by the UNIX operating system. A typical reason for such errors is that you have run out of some re­sources, like file descriptors in your machine.

**Listen failed.**

The test port failed to listen on the given port. A typical reason for such errors is that you have run out of some re­sources.

**-> unexpected character at character position: .**

The SIP test port found an error in the message. The error message contains the erroneous character and its position within the message. For more details see

**Error during accepting connection request.**

The test port failed to accept an incoming TCP connection request. A typical reason for such errors is that you have run out of some resources, like file descriptors in your machine.

**Message too long for UDP.**

The sip message too long for UDP packet. Redesign the testcase to use TCP connection.

**UDP/TCP recvfrom failed.**

There was an error during receiving data from the socket. This is an unexpected error returned by the UNIX operating system. A typical reason for such errors is that you have run out of some re­sources, like file descriptors in your machine.

**Getting of IP address of remote host failed.**

The test port cannot determine the IP address of remote host. Check the name or the IP address of the remote host.

**There is no valid destination address available. Message can not be sent!**

The destination address was not given. Specify the destination host.

**Send failed.**

This is an unexpected error returned by the UNIX operating system. A typical reason for such errors is that you have run out of some resources, like file descriptors in your machine.

# Warning messages

During the execution of the SIPmsg test suite the user is notified about useful information, discrepancies, non-critical errors that have no effect on the execution of the test suite but may be a result of some misconfiguration or other type of mistakes. They are self-explanatory so no description is given here.

Example warning messages:

**Incorrect default sip port in config file, default value <port\_num> is used.**

Invalid port number supplied in configuration file for *local\_sip\_port*. Default port number used.

**Incorrect default sip protocol in config file, default value <protocol> is used.**

Invalid protocol name supplied in configuration file for *default\_sip\_protocol*. Default protocol used.

**Incorrect destination sip port in config file, default value <port\_num> is used.**

Invalid protocol name supplied in configuration file for *default\_dest\_por*t. Default protocol used.

**TCP connection closed by peer.**

Remote host closed the TCP connection. The test port reopens the connection when needed.

**Source address differs from IUT address.**

The received SIPmsg message comes from an unexpected host.

**Error in SIPmsg start line. Message discarded.**

The SIPmsg start line contains unrecoverable error. The received message discarded.

**Missing mandatory headers. Message discarded.**

Some mandatory headers are missing from the SIPmsg message. The received message discarded.

# Examples

You can find some example files in the demo directory of the product.

## Script to modify Makefile (for development purposes)

The following script can be used to modify the generated Makefile in the TITAN GUI. The modified Makefile can be used to compile the Bison and Flax sources. Typically the compilation of Bison and Flex sources is needed during development and fault correction.

#!/bin/sh

editcmd='

/(PLATFORM)\_LIBS)/{

a\

a\

SIP\_parse\_.tab.c SIP\_parse\_.tab.h: SIP\_parse.y

a\

\ bison -dv -p SIP\_parse\_ -b SIP\_parse\_ $<

a\

a\

lex.SIP\_parse\_.c: SIP\_parse.l

a\

\ flex -Bvpp -PSIP\_parse\_ SIP\_parse.l

a\

}

/ -$(RM) ./ {

a\

SIP\_parse\_.output \\

}

/# Add your rules here if necessary.../ {

a\

SipPort.cc: SIP\_parse\_.tab.h

a\

lex.SIP\_parse\_.c: SIP\_parse\_.tab.h

a\

SDP\_parse\_.tab.c: SDP\_TypesAndConf.hh

}

'

sed -e "$editcmd" <$1 >$2

# How to find the faulty header in case of parse error

The following error or warning message indicates a faulty SIP message:

**-> unexpected character at character position:**

The behaviour of the test port in case of the faulty SIP message is controlled by the “error\_mode” test port parameter:

* “ignore” The test port ignore any erroneous messages and discard them without notice.
* “Warning” The test port will issue a warning if erroneous message received and pass the message as RAW message to the test case.
* “error” The test port will generate error if erroneous message received.

The default value is “*error”.*

In order to avoid the TTCN error during the test campaign it is recommended to set it to “warning”. The default value are kept for backward compatibility reason.

The decoder function, f\_SIP\_decode, returns the erroneous message as RAW message.

## Identify the error

The error message contains the faulty character and its position within the faulty SIP message. In order to find the fault the message should be logged.

The faulty message is logged if the “debug” is enabled or the test case should log the received RAW message.

The place of the fault is indicated in the error message, so SIP parser found the fault at the nth character.

## Example

Warning: SIP Test Port: syntax error "=" -> unexpected character at character position 170.

f\_EPTF\_SIP\_Message\_MsgHandlerUnhandled raw message: { raw := \"SIP/2.0 200 OK\r\nContent-Length: 0\r\nTo: <sip:46750000001@thule.lugv.ericsson.se>;tag=ft1copij-8p7\r\nContact: <sip:10.64.66.134:5060;fid=traffic\_instance\_PL\_2\_8\_1;bekey=sip=46750000001%40thule.lugv.ericsson.se>\r\nCseq: 2125564419 REGISTER\r\nVia: SIP/2.0/TCP 130.100.127.147:37000;branch=z9hG4bK1633698T000001\r\nCall-Id: TTCN3293710000000@130.100.127.147\r\nFrom: <sip:130.100.127.147:37000>;tag=168211000000\r\nServer: PGM5.0\_RLS\r\n\r\n" }"

The faulty character, which is the 170th character of the message, is indicated by red mark.

Please note that the \n and the \r represents only 1 character.

# Terminology

No specific terminology used.

## Abbreviations

SIP Session Initiation Protocol

ETSI European Telecommunication Standards Institute

IETF Internet Engineering Task Force

IP Internet Protocol

IUT Implementation Under Test

RFC Request For Comments

SCTP Stream Control Transmission Protocol

SUT System Under Test

TCP Transmission Control Protocol

TTCN-3 Testing and Test Control Notation version 3

UDP User Datagram Protocol

TP Test Port An adaptation between TTCN-3 Test Ex­ecutor and SUT

MTU Maximum Transmission Unit

# References

1. ETSI ES 201 873-1 v.3.1.1 (2005-06)  
   The Testing and Test Control Nota­tion version 3. Part 1: Core Language
2. 109 21-CNL 113 319-16  
   SIPmsg Test Port for TTCN-3 Toolset with TITAN, Product Revision Information 2/198 17-CRL 113 200 Uen
3. 1/198 17-CRL 113 200/6 Uen  
   User Guide for TITAN TTCN–3 Test Executor
4. RFC3261  
   SIP: Session Initiation Protocol
5. RFC2806  
   URLs for Telephone Calls
6. RFC2976  
   The SIP INFO Method
7. RFC3262  
   Reliability of Provisional Responses in the Session Initiation Protocol (SIP)
8. RFC3311  
   The Session Initiation Protocol (SIP) UPDATE Method
9. RFC3323  
   A Privacy Mechanism for the Session Initiation Protocol (SIP)
10. RFC3325  
    Private Extensions to the Session Initiation Protocol (SIP) for Asserted Identity within Trusted Networks
11. RFC3326  
    The Reason Header Field for the Session Initiation Protocol (SIP)
12. RFC3265  
    Session Initiation Protocol (SIP)-Specific Event Notification
13. RFC3455  
    Private Header (P-Header) Extensions to the Session Initiation Protocol (SIP) for the 3rd-Generation Partnership Project (3GPP)
14. IETF Draft draft-ietf-dip-session-timer-15.txt  
    Session Timers in the Session Initiation Protocol (SIP)  
    draft-ietf-sip-session-timer-15
15. RFC3428  
    Session Initiation Protocol (SIP) Extension for Instant Messaging
16. RFC3515  
    The Session Initiation Protocol (SIP) Refer Method
17. RFC3841  
    Caller Preferences for the Session Initiation Protocol (SIP)
18. RFC3313  
    Private Session Initiation Protocol (SIP) Extensions for Media Authorization
19. RFC3327  
    Session Initiation Protocol (SIP) Extension Header Field for Registering Non-Adjacent Contacts
20. RFC3329  
    Security Mechanism Agreement for the Session Initiation Protocol (SIP)
21. RFC3603  
    Private Session Initiation Protocol (SIP) Proxy-to-Proxy Extensions for Supporting the PacketCable Distributed Call Signalling Architecture
22. RFC3608  
    Session Initiation Protocol (SIP) Extension Header Field for Service Route Discovery During Registration
23. RFC3891  
    The Session Initiation Protocol (SIP) "Replaces" Header
24. RFC3892  
    The Session Initiation Protocol (SIP) Referred-By Mechanism
25. RFC3903  
    Session Initiation Protocol (SIP) Extension for Event State Publication
26. RFC3911  
    The Session Initiation Protocol (SIP) "Join" Header
27. RFC 3420 Internet Media Type message/sipfrag
28. IETF Draft draft-levy-sip-diversion-08.txt  
    Diversion Indication in SIP - draft-levy-sip-diversion-08
29. RFC 4244  
    An Extension to the Session Initiation Protocol (SIP) for Request History Information
30. RFC4488  
    Suppression of Session Initiation Protocol (SIP) REFER Method Implicit Subscription
31. draft-ietf-sip-refer-with-norefersub-04  
    Suppression of Session Initiation Protocol REFER Method Implicit Subscription
32. draft-allen-sipping-poc-p-headers-01  
    Private Header (P-Header) Extensions to the Session Initiation Protocol (SIP) for the Open Mobile Alliance (OMA) Push to talk over Cellular (PoC)
33. RFC 5009  
    Private Header (P-Header) Extension to the Session Initiation Protocol (SIP) for Authorization of Early Media
34. draft-kaplan-sip-session-id-02  
    A Session Identifier for the Session Initiation Protocol (SIP)
35. RFC5502  
    The SIP P-Served-User Private-Header (P-Header) for the 3GPP IP Multimedia (IM) Core Network (CN) Subsystem
36. RFC5002  
    The Session Initiation Protocol (SIP) P-Profile-Key Private Header (P-Header)
37. RFC4457  
    The Session Initiation Protocol (SIP) P-User-Database Private-Header (P-Header)
38. 66/1594-FCP 101 6059 Uen CSCF Support of wildcarded IMPU
39. RFC4412  
    Communications Resource Priority for the Session Initiation Protocol (SIP)
40. draft-ott-sip-serv-indication-notification-00  
    Private Header (P-Header) Extensions to the Session Initiation Protocol (SIP) for the support of the Services for the European Telecommunications Standards Institute
41. RFC6442  
    Location Conveyance for the Session Initiation Protocol
42. Indication of features supported by proxy   
    draft-holmberg-sipcore-proxy-feature-04
43. RFC4538  
    Request Authorization through Dialog Identification in the Session Initiation Protocol (SIP)
44. RFC6086  
    Session Initiation Protocol (SIP) INFO Method and Package Framework
45. RFC6086  
    Session Initiation Protocol (SIP) INFO Method and Package Framework
46. 531/0363-FCP 101 5091 Rev. PD2  
    FEATURE CONCEPT STUDY  
    ICBS and Flexible Charging Support for Japan
47. CPM Conversation Functions  
    Open Mobile Alliance  
    OMA-TS-CPM\_Conv\_Fnct-V1\_0-20101012-C  
    http://www.openmobilealliance.org