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

The purpose of this document is to specify the content of the TCC Useful Functions modules.

Contents

[1 About this Document 4](#_Toc509407997)

[1.1 How to Read this Document 4](#_Toc509407998)

[1.2 Presumed Knowledge 4](#_Toc509407999)

[2 Functionality 4](#_Toc509408000)

[2.1 Implemented protocols 4](#_Toc509408001)

[2.2 Supported Standards 4](#_Toc509408002)

[2.3 Routing Functionality 4](#_Toc509408003)

[2.4 Modified and non-implemented Protocol Elements 4](#_Toc509408004)

[2.5 Backward incompatibilities 4](#_Toc509408005)

[2.5.1 R35A 4](#_Toc509408006)

[2.6 System Requirements 5](#_Toc509408007)

[2.7 Installation 6](#_Toc509408008)

[3 Functions 6](#_Toc509408009)

[3.1 Message Encoding Functions 6](#_Toc509408010)

[3.1.1 Overview 6](#_Toc509408011)

[3.1.2 Function definitions 6](#_Toc509408012)

[3.1.3 Error messages 8](#_Toc509408013)

[3.1.4 Warning messages 8](#_Toc509408014)

[3.1.5 Examples 9](#_Toc509408015)

[3.2 Test Environment Handling Functions 10](#_Toc509408016)

[3.2.1 Overview 10](#_Toc509408017)

[3.2.2 Function definitions 10](#_Toc509408018)

[3.2.3 Error messages 10](#_Toc509408019)

[3.2.4 Warning messages 10](#_Toc509408020)

[3.2.5 Examples 11](#_Toc509408021)

[3.3 Network Interface Functions 11](#_Toc509408022)

[3.3.1 Overview 11](#_Toc509408023)

[3.3.2 Function definitions 11](#_Toc509408024)

[3.3.3 Error messages 14](#_Toc509408025)

[3.3.4 Warning messages 15](#_Toc509408026)

[3.3.5 Examples 16](#_Toc509408027)

[3.3.6 Makefile 16](#_Toc509408028)

[3.4 File I/O Functions 17](#_Toc509408029)

[3.4.1 Overview 17](#_Toc509408030)

[3.4.2 Function definitions 17](#_Toc509408031)

[3.4.3 Error messages 20](#_Toc509408032)

[3.4.4 Warning messages 21](#_Toc509408033)

[3.4.5 Examples 21](#_Toc509408034)

[3.5 Assertion Functions 22](#_Toc509408035)

[3.5.1 Overview 22](#_Toc509408036)

[3.5.2 Function definitions 22](#_Toc509408037)

[3.5.3 Error messages 22](#_Toc509408038)

[3.5.4 Warning messages 22](#_Toc509408039)

[3.5.5 Examples 22](#_Toc509408040)

[3.6 Conversion Functions 23](#_Toc509408041)

[3.6.1 Overview 23](#_Toc509408042)

[3.6.2 Function definitions 23](#_Toc509408043)

[3.6.3 Error messages 26](#_Toc509408044)

[3.6.4 Warning messages 26](#_Toc509408045)

[3.6.5 Examples 26](#_Toc509408046)

[3.6.6 Makefile 27](#_Toc509408047)

[3.7 Date and Time Functions 27](#_Toc509408048)

[3.7.1 Overview 27](#_Toc509408049)

[3.7.2 Function definitions 27](#_Toc509408050)

[3.7.3 Error messages 29](#_Toc509408051)

[3.7.4 Warning messages 29](#_Toc509408052)

[3.7.5 Examples 29](#_Toc509408053)

[3.8 Maths Functions 31](#_Toc509408054)

[3.8.1 Overview 31](#_Toc509408055)

[3.8.2 Function definitions 31](#_Toc509408056)

[3.8.3 Error messages 35](#_Toc509408057)

[3.8.4 Warning messages 35](#_Toc509408058)

[3.8.5 Examples 35](#_Toc509408059)

[3.9 Security Functions 37](#_Toc509408060)

[3.9.1 Overview 37](#_Toc509408061)

[3.9.2 Function definitions 37](#_Toc509408062)

[3.9.3 Error messages 40](#_Toc509408063)

[3.9.4 Warning messages 40](#_Toc509408064)

[3.9.5 Examples 40](#_Toc509408065)

[3.10 Message Handling Functions 43](#_Toc509408066)

[3.10.1 Overview 43](#_Toc509408067)

[3.10.2 Function definitions 43](#_Toc509408068)

[3.10.3 Error messages 44](#_Toc509408069)

[3.10.4 Warning messages 44](#_Toc509408070)

[3.10.5 Examples 44](#_Toc509408071)

[3.11 Permutated Parameters Handling 45](#_Toc509408072)

[3.11.1 Overview 45](#_Toc509408073)

[3.11.2 Function definitions 46](#_Toc509408074)

[3.11.3 Error messages 47](#_Toc509408075)

[3.11.4 Warning messages 47](#_Toc509408076)

[3.11.5 Examples 47](#_Toc509408077)

[3.12 Titan Metadata Functions 47](#_Toc509408078)

[3.12.1 Overview 47](#_Toc509408079)

[3.12.2 Function definitions 47](#_Toc509408080)

[3.12.3 Error messages 47](#_Toc509408081)

[3.12.4 Warning messages 47](#_Toc509408082)

[3.12.5 Examples 47](#_Toc509408083)

[3.13 Text Template Substitution Functions 48](#_Toc509408084)

[3.13.1 Overview 48](#_Toc509408085)

[3.13.2 Function definitions 48](#_Toc509408086)

[3.13.3 Error messages 49](#_Toc509408087)

[3.13.4 Warning messages 49](#_Toc509408088)

[3.13.5 Examples 49](#_Toc509408089)

[3.14 IPsec Functions 49](#_Toc509408090)

[3.14.1 Overview 49](#_Toc509408091)

[3.14.1.1 PF\_KEY API 49](#_Toc509408092)

[3.14.1.2 Netlink XFRM API 50](#_Toc509408093)

[3.14.2 Function definitions 50](#_Toc509408094)

[3.14.2.1 Security Parameter Index query function 50](#_Toc509408095)

[3.14.2.2 Security Association Database handling functions 51](#_Toc509408096)

[3.14.2.3 Security Policy Database handling functions 52](#_Toc509408097)

[3.14.3 Error messages 54](#_Toc509408098)

[3.14.4 Warning messages 54](#_Toc509408099)

[3.14.5 Examples 54](#_Toc509408100)

[3.15 XPath Support Functions 55](#_Toc509408101)

[3.15.1 Overview 55](#_Toc509408102)

[3.15.2 Function definitions 55](#_Toc509408103)

[3.15.3 Building the TinyXPath external library 56](#_Toc509408104)

[3.15.4 Preparing your TTCN-3 environment for using TinyXPath 57](#_Toc509408105)

[3.15.4.1 Modification of the TTCN-3 files in your project 57](#_Toc509408106)

[3.15.4.2 Modification of the Makefile of your project 58](#_Toc509408107)

[3.15.5 Error messages 58](#_Toc509408108)

[3.15.6 Warning messages 58](#_Toc509408109)

[3.15.7 Examples 58](#_Toc509408110)

[3.16 System functions 59](#_Toc509408111)

[3.16.1 Overview 59](#_Toc509408112)

[3.17 Function definitions 59](#_Toc509408113)

[3.17.1 Error messages 59](#_Toc509408114)

[3.17.2 Warning messages 59](#_Toc509408115)

[3.17.3 Examples 59](#_Toc509408116)

[3.18 FileSystem functions 59](#_Toc509408117)

[3.18.1 Overview 59](#_Toc509408118)

[3.19 Function definitions 60](#_Toc509408119)

[3.19.1 Error messages 60](#_Toc509408120)

[3.19.2 Warning messages 60](#_Toc509408121)

[3.19.3 Examples 60](#_Toc509408122)

[3.20 Regexp functions 61](#_Toc509408123)

[3.20.1 Overview 61](#_Toc509408124)

[3.20.2 Function definitions 61](#_Toc509408125)

[3.20.3 Error messages 62](#_Toc509408126)

[3.20.4 Warning messages 62](#_Toc509408127)

[3.20.5 Examples 62](#_Toc509408128)

[4 Terminology 63](#_Toc509408129)

[4.1 Abbreviations 63](#_Toc509408130)

[5 References 63](#_Toc509408131)

# About this Document

## How to Read this Document

This is the Function Specification for the TCC Useful Functions. The modules are developed for the TTCN-3 Toolset with TITAN.

## Presumed Knowledge

The knowledge of the TITAN TTCN-3 Test Executor [3] and the TTCN-3 language [1] is essential. Other documents that the reader should be familiar with are referenced in the consecutive section of each function.

# Functionality

The functions described in this document are TTCN-3 or external TTCN-3 functions. TTCN-3 functions are implemented in TTCN-3 language; external TTCN-3 functions are implemented in C++.

## Implemented protocols

-

## Supported Standards

-

## Routing Functionality

Routing functionality is not performed.

## Modified and non-implemented Protocol Elements

-

## Backward incompatibilities

### R35A

The non-open source TCCSecurity\_Functions are moved to a separate product.

The TCC Security Functions CNL 113 874 should be added to the project if the moved functions are used.

If the non-moved functions are used in the project the TCCOpenSecurity\_Functions should be imported instead of TCCSecurity\_Functions

Moved files:

TCCSecurity.cc

TCCSecurity\_Functions.ttcn

aka\_algorythm\_set.c

aka\_algorythm\_set.h

snow3g.c

snow3g.h

zuc.c

zuc.h

Moved functions:

f\_TCCSecurity\_hex2char

f\_IMSAKA\_f1

f\_IMSAKA\_f1\_opc

f\_IMSAKA\_f2

f\_IMSAKA\_f2\_opc

f\_IMSAKA\_f3

f\_IMSAKA\_f3\_opc

f\_IMSAKA\_f4

f\_IMSAKA\_f4\_opc

f\_IMSAKA\_f2345

f\_IMSAKA\_f2345\_opc

f\_IMSAKA\_f5

f\_IMSAKA\_f5\_opc

f\_IMSAKA\_f1star

f\_IMSAKA\_f1star\_opc

f\_IMSAKA\_f5star

f\_IMSAKA\_f5star\_opc

f\_EAPSIM\_A3A8

f\_IMSAKA\_calculateAUTN

f\_calculate128\_EEA3

f\_calculate128\_EIA3

f\_calculate128\_EEA1

f\_calculate128\_EIA1

f\_calculate128\_EIA2

f\_calculate128\_EEA2

## System Requirements

In order to operate any of the Useful Functions the following system requirements must be satisfied:

* TITAN TTCN-3 Test Executor version 6.3.pl0 (CRL 113 200/6 R3A) or higher installed. For installation guide see [2].

Further system requirements, if any, are listed in the consecutive section of each function.

## Installation

Since the Useful Functions are used as a part of the TTCN-3 test environment this requires TTCN-3 Test Executor to be installed before any operation of these functions. For more details on the installation of TTCN-3 Test Executor see the relevant section of [2].

If not otherwise noted in the respective sections, the following are needed to use the Useful Functions:

Copy the files listed in the section related to the specific functions to the directory of the test suite or create symbolic links to them.

* Import the TTCN-3 module containing the declaration of the functions to the modules that will use them.
* Create Makefile or modify the existing one. For more detail see the relevant section of[3].

# Functions

## Message Encoding Functions

### Overview

Encoding functions can be used to encode data to a specific format or decode encoded data.

Encoding functions are implemented in the following files:

* TCCEncoding\_Functions.ttcn: external TTCN-3 function declarations
* TCCEncoding.cc: source file of the functions

### Function definitions

The following functions are defined in Encoding Functions.

external function enc\_MIME\_Base64(in octetstring p\_msg) return charstring;

Encodes the octetstring parameter “msg” to MIME Base64 format according to [3].

external function dec\_MIME\_Base64(in charstring p\_b64) return octetstring;

Decodes a MIME Base64 encoded character string to octetstring.

external function enc\_LDIF\_Base64(in octetstring p\_msg) return charstring;

Encodes the parameter “msg” to LDIF Base64 format according to [5].

external function dec\_LDIF\_Base64(in charstring p\_b64) return octetstring;

Decodes an LDIF Base64 encoded character string to octetstring.

**function f\_encGSM7bit(in universal charstring pl\_str) return octetstring**

Encodes the parameter “pl\_str” (containing characters of the default alphabet) to octetstring containing USSD packing [12]. Note that the Greek capital letters and the euro sign have to be represented in the “quadruple” notation (for example the Greek capital letter delta is char(0, 0, 3, 148) )

**function f\_decGSM7bit(in octetstring pl\_gsm7bit) return universal charstring**

Decodes a USSD packed encoded octetstring to characters of the default alphabet [12]. Note that non ASCII characters will be decoded into the quadruple notation. For example the Japanese yen symbol will be decoded to char(0, 0, 0, 165).

**function f\_is\_TBCD(in charstring pl\_number) return boolean;**

Returns true if the pl\_number contains only valid TBCD characters, otherwise returns false. The valid TBCD characters are 0-9, \*,#,a,b,c.

**external function f\_enc\_TBCD(in charstring pl\_char) return octetstring;**

Encodes the parameter “pl\_char” (charstring containing valid TBCD characters) to octetstring using TBCD encoding [13]. Valid TBCD characters are 0-9, \*,#,a,b,c. If the encoding of an invalid character is attempted, then an empty octetstring is returned with a warning.

**external function f\_dec\_TBCD(in octetstring pl\_oct) return charstring;**

Decodes a valid TBCD encoded octetstring to charstring format [13]. If the encoding is incorrect then an empty charstring is returned with a warning.

**function f\_encode\_ISUP\_Called\_Party\_Number  
(in ISUP\_Called\_Party\_Number pl\_ISUP\_Called\_Party\_Number) return octetstring**

Encodes <ISUP\_Called\_Party\_Number> type to octetstring format. (Encoding defined in ISUP protocol [14] section 3.9)

**function f\_encode\_ISUP\_Calling\_Party\_Number  
(in ISUP\_Calling\_Party\_Number pl\_ISUP\_Calling\_Party\_Number) return octetstring**

Encodes <ISUP\_Calling\_Party\_Number> type to octetstring format. (Encoding defined in ISUP protocol [14] section 3.10)

**function f\_encode\_MobileL3\_CalledPartyBCDNumber  
(in MobileL3\_CalledPartyBCDNumber pl\_MobileL3\_CalledPartyBCDNumber) return octetstring**

Encodes <MobileL3\_CalledPartyBCDNumber> type to octetstring format. (Encoding defined in Mobile L3 protocol [15] section 10.5.4.7)

**public function f\_enc\_TBCD\_hex(in hexstring pl\_hex) return octetstring**

The function converts a hexstring into TBCD-String format.

### Error messages

None.

### Warning messages

**Warning: Invalid character in Base64 encoded data: ...**

This warning message is only printed by the function dec\_LDIF\_Base64, if the Base64 encoded data passed as a parameter contains one or more characters that are not defined in Base64 (i.e. a character other than [A-Z][a-z][0-9]/+). The decoded data may still be valid.

**Warning : Invalid character <character> will be encoded as space!**

Only characters contained in Section 6.2.1 of [12] can be correctly encoded with **f\_encGSM7bit**. Other characters will be encoded as space.

**Warning : Unknown extension character <extension code> will be decoded as space!**

Only extensions contained in section 6.2.1.1 of [12] can be correctly decoded with **f\_decGSM7bit**. Other extension characters will be decoded as space.

**Warning : Filler digit at invalid location!**

f\_dec\_TBCD returns an empty charstring and gives this warning if there is a filler digit at a location other than the msb side of the last octet.

**Warning : Invalid TBCD digit!**

f\_enc\_TBCD returns an empty octetstring and gives this warning if the encoding of an invalid TBCD character is attempted.

### Examples

module TCCEncoding\_Example {  
  
import from TCCEncoding\_Functions all;  
  
type component test\_CT {  
 var charstring base64;  
 var octetstring msg, dec;  
}  
  
testcase TC() runs on test\_CT  
{  
 msg := char2oct(“Test message.”);  
 log(“Message is: ”, msg);  
 base64 := enc\_MIME\_Base64(msg);  
 log(“MIME Base64 encoded message is: ”, base64);  
 dec := dec\_MIME\_Base64(base64);  
 log(“Decoded message is: ”, dec);  
 if(dec != msg) {  
 setverdict(fail);  
 } else {  
 setverdict(pass);  
 }  
}

testcase tc\_2() runs on MyComp  
{  
 var octetstring v\_bit := f\_encGSM7bit("The last transaction cost $ 0.25. Your account balance is $ 39.50. To refill your account go to www.att.com/GoPhone.")

log(v\_bit)

var universal charstring v\_char := f\_decGSM7bit(v\_bit);

log(v\_char)  
}

testcase tc\_3() runs on MyComp  
{ //Encoding and decoding the Greek capital letter delta  
 var octetstring v\_bit := f\_encGSM7bit(char(0, 0, 3, 148))

log(v\_bit)

var universal charstring v\_char := f\_decGSM7bit(v\_bit);

log(v\_char)  
}

testcase tc\_4() runs on MyComp  
{  
log(f\_dec\_TBCD('01234F6789ABCDFE'O))

log(f\_enc\_TBCD("0123456789\*#abc"))  
}

control {  
 execute(TC());  
 execute(tc\_2());  
 execute(tc\_3());  
 execute(tc\_4());  
}  
  
}

## Test Environment Handling Functions

### Overview

Environment Handling Functions can be used to get or set environment variables.

Environment Handling Functions are implemented in the following files:

* TCCEnv\_Functions.ttcn: external TTCN-3 function declarations
* TCCEnv.cc: source code of the functions

### Function definitions

The following Environment Handling Functions are defined.

external function f\_GetEnv(in charstring p\_env\_name) return charstring;

Returns the environment variable “p\_env\_name”.

external function f\_PutEnv(in charstring p\_env\_name, in charstring p\_env\_value) return boolean;

Sets the environment variable “p\_env\_name” to “p\_env\_value”. Returns false on error, true on success.

### Error messages

None.

### Warning messages

**putenv failed with error code ...**

An error happened while setting the environment variable.

### Examples

module TCCEnv\_Example {  
  
import from TCCEnv\_Functions all;  
  
type component test\_CT { }  
  
testcase TC() runs on test\_CT  
{  
 log(“PATH=”, f\_GetEnv(“PATH”));  
 f\_PutEnv(“ENV”, “foobar”);  
 log(“ENV=”, f\_GetEnv(“ENV”));  
}  
  
control {  
 execute(TC());  
}  
  
}

## Network Interface Functions

### Overview

Network Interface Functions can be used to modify interface related settings. Eg. get/set IP address, set up/down interface.

Interface Functions are supported only on Linux and Solaris. For the setter functions root privilege is necessary. Otherwise warning messages will appear.

Network Interface Functions are implemented in the following files:

* TCCInterface\_Functions.ttcn
* TCCInterface.cc
* TCCInterface\_ip.h

### Function definitions

The following Network Interface Functions are defined.

**external function f\_getPortAvailabilityStatus(in charstring ipAddress, in integer portNumber, in TCCInterface\_ProtocolType protocolType) return TCCInterface\_PortStatus;**

The function returns what was the status of the IP/port/Protocol some time ago. The returned information was outdated before the function returned, so use it only as a hint.

**external function f\_setIP(in charstring interface, in charstring ip, in charstring netmask, in charstring broadcast, in integer number := 1);**

Set the IP address, subnet mask and broadcast address of the given interface. Parameter “number” is optional (and supported on Linux only).

If parameter “number” is given, a range of virtual interfaces are set up with continuous IP addresses, starting from the given IP address. If parameter “interface” is a real network interface, the first virtual interface is “<interface>:0”. If parameter “interface” is virtual, the next sequence number is used.

Note: no subnet mask and broadcast checking is done by the function.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Linux** | **Solaris** | **Cygwin** |
| **IPv4** | **X** | **X** |  |
| **IPv6** |  |  |  |

**external function f\_deleteIP(in charstring interface);**

Delete the IP address of the given interface. If interface is virtual, it is set down.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Linux** | **Solaris** | **Cygwin** |
| **IPv4** | **X** | **X** |  |
| **IPv6** |  |  |  |

**external function f\_setIP\_ip(in charstring interface, in charstring ipaddress, in integer prefix := 32, in integer v\_set := 1) return boolean;**

This function is also used for setting up IPs on interfaces, but uses the NETLINK interface of the Linux kernel, what is far more fast than the original interface used by f\_setIP. Thus this function only works in Linux. Works with both IPv4 and IPv6 addresses.

On the specified interface the given new IP address will be registered with the addressprefix also given in the parameter list. The last parameter tells the behaviour, other than 1 will cause deleting the ip from the interface. Thus f\_setIP\_ip with 0 = v\_set is equal to call f\_delIP\_ip. On success returns true, otherwise false and print some information into logs what was the problem.

Can only be used if the LINUX flag is specified when compiling the test, otherwise a warning message will be printed. When the USE\_IPROUTE flag is specified, the f\_setIP and f\_deleteIP functions will be redefined to be used with f\_setIP\_ip with the following parameter list mapping:  
**interface = interface,  
ipaddress = ip,  
prefix = 32,  
v\_set = 1**

and

**interface = interface,  
ipaddress = “”,  
prefix = 32,  
v\_set = 0.**

**external function f\_delIP\_ip(in charstring interface, in charstring ipaddress, in integer prefix := 32) return boolean;**

Deletes the given IP address/prefix from an interface.

On success returns true, otherwise false and print some information into logs what was the problem.

**external function f\_getIP(in charstring interface, out charstring ip, out charstring netmask, out charstring broadcast, in TCCInterface\_IPAddressType addressType:=IPv4);**

Returns the IP address, subnet mask and broadcast address of the given interface, in the standard Internet dotted format (e.g. “10.0.0.2”). The type of address can be IPv4 or IPv6.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Linux** | **Solaris** | **Cygwin** |
| **IPv4** | **X** | **X** | **X** |
| **IPv6** | **X** |  |  |

**external function f\_setInterfaceUp(in charstring interface, in TCCInterface\_IPAddressType addressType:=IPv4);**

Set up the given interface. The type of address can be IPv4 or IPv6.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Linux** | **Solaris** | **Cygwin** |
| **IPv4** | **X** | **X** |  |
| **IPv6** | **X** | **X** |  |

**external function f\_setInterfaceDown(in charstring interface, in TCCInterface\_IPAddressType addressType:=IPv4);**

Set down the given interface. The type of address can be IPv4 or IPv6.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Linux** | **Solaris** | **Cygwin** |
| **IPv4** | **X** | **X** |  |
| **IPv6** | **X** | **X** |  |

**external function f\_getHostName() return charstring;**

Get name of the current host machine.

|  |  |  |
| --- | --- | --- |
| **Linux** | **Solaris** | **Cygwin** |
| **X** | **X** | **X** |

**external function f\_getIpAddr(in charstring hostname, in TCCInterface\_IPAddressType addressType:=IPv4) return charstring;**

Get IP address of the current host machine called with parameter **hostname**. The type of address can be IPv4 or IPv6.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Linux** | **Solaris** | **Cygwin** |
| **IPv4** | **X** | **X** | **X** |
| **IPv6** | **X** | **X** |  |

**external function f\_getIpAddresses(in charstring hostname return IPAddresses;**

Is a successor of **f\_getIpAddr** Get IP address of the machine called with parameter **hostname**. It will return two lists of **charstring**, one for all the ipv4 and one for all the ipv6 addresses of **hostname.**

Note, that the returned lists are not compatible with any other record of **charstring** in TITAN's load run time library, which is the default, and thus it is not acceptable to directly copy this lists for example to **EPTF**\_**CharstringList**.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Linux** | **Solaris** | **Cygwin** |
| **IPv4** | **X** | **X** | **X** |
| **IPv6** | **X** | **X** |  |

**external function** **f\_verifyIpAddr(in charstring pl\_host, in TCCInterface\_IPAddressType pl\_ipType := UNKNOWN) return boolean;**

Validates the supplied pl\_host and returns true if it is a valid IP address. The validation can be restricted to IPv4 or IPv6 with the pl\_ipType parameter.

### Error messages

**Unbound argument `<parameter>'.**

The given parameter is missing.

**Argument `<parameter>' is an empty string.**

The value of the given parameter is empty.

**Argument `interface' is too long (expected: at most %, given: % characters).**

The given interface name is not valid (too long).

**Invalid length of argument `<parameter>' (expected %, given: % octets).**

The length of the given parameter is not correct.

**Could not create socket.**

Some kernel error occurred when opening the socket.

**IP address range limit.**

No more virtual interfaces can be set up.

**Setting the IP address is supported on Linux and Solaris only.**

**Deleting the IP address is supported on Linux and Solaris only.**

**Getting the IP address is supported on Linux and Solaris only.**

**Setting up the interface is supported on Linux and Solaris only.**

**Setting down the interface is supported on Linux and Solaris only.**

Network Interface Functions are supported on Linux and Solaris only.

### Warning messages

Warning messages appear if the kernel could not execute the requested command.

**Could not set IP address of interface `<interface>'.**

**Could not set subnet mask of interface `<interface>'.**

**Could not set broadcast address of interface `<interface>'.**

**Could not get the flags of interface `<interface>'.**

**Could not set the flags of interface `<interface>'.**

**Could not delete IP address on interface `<interface>'.**

**Could not get address of interface '<interface>'.**

**Could not get flags of interface '<interface>'.**

**Could not set up interface '<interface>'.**

**Could not get flags of interface '<interface>'.**

**Could not set down interface '<interface>'.**

### Examples

module TCCInterface\_Example {

import from TCCInterface\_Functions all;

type component test\_CT { };

testcase TC() runs on test\_CT

{

var charstring ip, subnet, broadcast;

f\_getIP("eth2",ip,subnet,broadcast);

log("IP: " & ip);

log("Subnetmask: " & subnet);

log("Broadcast: " & broadcast);

setverdict(pass);

}

testcase f\_test\_gethostname() runs on test\_CT

{

log(" ----------- f\_getHostName ----------- ");

log("Hostname: ", f\_getHostName());

setverdict(pass);

}

testcase f\_test\_getipaddr() runs on test\_CT

{

log(" ----------- f\_getIpAddr ----------- ");

log("IP address: ", f\_getIpAddr(f\_getHostName()));

setverdict(pass);

}

control {

execute(TC());

execute(f\_test\_gethostname());

execute(f\_test\_getipaddr());

}

}

### Makefile

The –lresolv has to be added to SOLARIS8\_LIBS.

## File I/O Functions

### Overview

This bunch of external functions enables you to handle files from TTCN-3. These functions use the POSIX file I/O functions underneath, and they behave similarly. For examples see TCCFileIO\_Example.ttcn.

These File I/O Functions were tested under Linux/Solaris/Cygwin platforms.

File I/O Functions are implemented in the following files:

* TCCFileIO\_Functions.ttcn
* TCCFileIO.cc

### Function definitions

The following File I/O Functions are defined. For the exact API check the generated NaturalDocs documentation.

**external function f\_FIO\_get\_error\_code ();**

The last error code. It simply returns the value of errno. Errno is a global variable defined by the ISO C standard. Its value is set by the system calls. It always holds the error code for the last error.

**external function f\_FIO\_get\_error\_string ();**

The last error message in textual format. It simply returns the string associated with the errno value returned by f\_FIO\_get\_error\_code.

**external function f\_FIO\_open\_\* (in charstring name);**

These functions open the files in different ways using the open system call. Files can be opened in appending or truncating mode for reading only/writing only/reading and writing. These functions always return an integer, the file descriptor for the file name that is now opened or -1 if an error occurred. The error string/code will be available through f\_FIO\_get\_error\_string/f\_FIO\_get\_error\_code. The returned integer can be stored in TTCN-3 and can be used to do I/O operations on a specific file. It is possible to open files in exclusive mode (using the f\_FIO\_open\_\*\_excl functions), but it works differently on Linux/Unix and Cygwin platforms. If a file is opened in exclusive mode in Cygwin, it cannot be reopened from any other process or from the current one in any mode until the file is closed by the lock-holder. Closing the file implicitly releases the lock. Linux/Unix platforms are not that restrictive. It is possible to have a process, which opened the file in exclusive mode and another opened it with e.g. f\_FIO\_open\_append\_wronly. In this case the result is unpredictable. On these platforms it’s advisable to always use the functions for exclusive opening to prevent these situations.

**external function f\_FIO\_close (in integer fd);**

With this function a file associated with a given file descriptor can be closed.

**external function f\_FIO\_seek\_\* (in integer fd);**

These functions can seek in files. They can seek to the beginning or end of the file or they can move the file pointer with given bytes forward or backward from its actual position. They simply return what the POSIX function lseek returns.

**external function f\_FIO\_write\_\* (in integer fd, in octetstring/charstring data/text);**

With this function you're able to write binary or textual data to a file. The output buffer is a CHARSTRING in text mode or an OCTETSTRING in binary mode. The buffer is written out from the actual position of the file pointer. To make sure that all data is written out to the disk, call f\_flush after each write or use the appropriate write functions with flush support (with \*\_flush suffix).

**external function f\_FIO\_read\_\* (in integer fd, inout octetstring/charstring data/text, in integer bytes);**

With these functions you can read a given number of bytes into a TTCN-3 CHARSTRING/OCTETSTRING buffer. CHARSTRING is used for text mode OCTETSTRING is used for binary mode. The given number of bytes are read from the actual position of the file pointer.

**external function f\_FIO\_read\_\*\_until (in integer fd, inout octetstring/charstring data/text, in octetstring/charstring separator);**

With these functions you can read textual or binary data from a given file token by token. Each token must be separated with the same characters (CHARSTRING) or octets (OCTETSTRING). If the given separator is first matched the appropriate CHARSTRING/OCTETSTRING is returned. Calling these functions next time, they will find the next token. If EOF is reached before the separator pattern is matched -1 is returned.

**external function f\_FIO\_read\_data\_TLV (in integer pl\_fd, inout octetstring pl\_data) return integer;**

With this function you can read binary data -containing a full ASN.1 TLV structure- from a given file. If the full ASN.1 TLV structure is matched the appropriate OCTETSTRING stored in pl\_data. Calling these functions next time, they will find the next ASN.1 TLV structure. This function works only files containing nothing else but full TLV structures. If no full TLV structure is found -1 is returned, in case of success the length of the TLV structure is returned

**external function f\_FIO\_set\_filedescriptor\_previousline (in integer fd) return integer;**

With this function you can set the given file descriptor to the previous line. Calling this function next time, the functions will find the previous previous line. If there was no problem, then the return value will be 1.

**external function f\_FIO\_chdir (in charstring pl\_name) return boolean;**

With this function you can change the current directory. If directory change is successful, the true is returned, otherwise false is returned.

**external function f\_FIO\_mkdir (in charstring p\_dir\_name) return boolean;**

With this function you can create a directory. If directory creation is successful, the true is returned, otherwise false is returned.

**external function f\_FIO\_rmdir (in charstring p\_dir\_name) return boolean;**

With this function you can delete a directory. If directory is successfully deleted, the true is returned, otherwise false is returned.

**external function f\_FIO\_fileOrDirExists (in charstring p\_ \_name) return boolean;**

With this function you can rename an existing file or directory. If the file or directory not exists, then the return value is false. The target path has to exist as well, otherwise it returns false.

**external function f\_FIO\_rename (in charstring p\_old\_name, in charstring p\_new\_name) return boolean;**

With this function you can check the existence of a directory or a file. If the directory or file exists, the true is returned, otherwise false is returned.

**external function f\_FIO\_stat (in charstring p\_name, out FIO\_permissions p\_permissions) return boolean;**

With this function you can query the permission of a directory or a file. If the operation is successful, the true is returned and the **p\_permissions** contains the permissions, otherwise false is returned.

**external function f\_FIO\_chmod (in charstring p\_name, in FIO\_permissions p\_permissions) return boolean;**

With this function you can set the permissions of the file or directory according to the p\_permissions. If the value of the field is:

* true: set the permission
* false: clear the permission
* omit: doesn't change the permission

If the operation is successful, the true is returned, otherwise false is returned.

**external function f\_FIO\_remove (in charstring pl\_file\_name) return boolean;**

With this function you can delete a directory or file. If directory or file is successfully deleted, the true is returned, otherwise false is returned.

**external function f\_FIO\_getFileInfo (in charstring p\_name) return FIO\_FileInfo;**

This function returns a FIO\_FileInfo, which contains several information about the given file, such as:

* **fileType**: the type of the file
* **nodeNumber**: inode number
* **mode**: file type and mode
* **linkCount**: number of hard links
* **ownership**: user ID of owner
* **groupId**: group ID of owner
* **blockSize**: block size for file system I/O
* **fileSize**: total size, in bytes
* **blocksAllocated**: number of 512B blocks allocated
* **lastStatusChange**: time of last access in seconds since 1970
* **lastFileAccess**: time of last modification in seconds since 1970
* **lastFileModification**: time of last status change in seconds since 1970

### Error messages

These functions always return -1 if an error occurs during the execution. The return values must be checked to detect these situations. The corresponding error message/code is available through calling f\_FIO\_get\_error\_string/f\_FIO\_get\_error\_code. By default the functions don’t write anything to the standard output/error. If you want to get error reports on the screen, the functions must be compiled with “-DVERBOSE\_DEBUG” or define a macro “VERBOSE\_DEBUG”.

**Cannot open file**

The file cannot be opened.

**The file is already opened**

The file is already opened.

**The file is not opened**

The file is not yet opened.

**Cannot lock file**

The file cannot be opened in exclusive mode.

**Cannot allocate memory**

There’s not enough memory to increase the size of the input buffer.

**Cannot close file**

**Read error**

**Cannot write to file**

**End of file**

EOF is reached unexpectedly.

### Warning messages

None.

### Examples

module TCCFileIO\_Example

{

import from TCCFileIO\_Functions all;

type component empty\_CT { };

testcase TC\_test () runs on empty\_CT

{

var integer vl\_fd;

var charstring vl\_text\_out := "First lineabcdefSecond lineabcdefThird lineabcdefFourth lineabcdef";

var charstring vl\_text\_tmp := "abcdef";

var charstring vl\_text\_in;

vl\_fd := f\_FIO\_open\_trunc\_rdwr\_excl ("test1.txt");

if (vl\_fd < 0)

{

log (f\_FIO\_get\_error\_string ());

setverdict (fail);

}

f\_FIO\_write\_text (vl\_fd, vl\_text\_out);

f\_FIO\_flush (vl\_fd);

f\_FIO\_seek\_home (vl\_fd);

for (var integer vl\_i := 0; vl\_i < 4; vl\_i := vl\_i + 1)

{

var integer n := f\_FIO\_read\_text\_until (vl\_fd, vl\_text\_in, vl\_text\_tmp);

if (n < 0)

{

log (f\_FIO\_get\_error\_string ());

setverdict (fail);

}

log (vl\_text\_in);

}

f\_FIO\_seek\_end (vl\_fd);

f\_FIO\_write\_text (vl\_fd, vl\_text\_out);

f\_FIO\_close (vl\_fd);

/\* Try to close it again. \*/

if (f\_FIO\_close (vl\_fd) < 0)

{

setverdict (pass);

}

}

control

{

execute (TC\_test ());

}

}

## Assertion Functions

### Overview

Assertion functions enable you to apply assertion from TTCN-3.

These Assertion Functions were tested under Linux/Solaris/Cygwin platforms.

Assertion Functions are implemented in the following files:

* TCCAssertion\_Functions.ttcn
* TCCAssertion.cc

### Function definitions

The following Assertion Functions are defined. For the exact API check the generated NaturalDocs documentation.

**external function f\_assert(const CHARSTRING& pl\_assertMessage, boolean pl\_predicate);**

If assertion is activated, f\_assert assures that the pl\_predicate parameter is true at call of the function. Else it fails with dynamic test case error and displays assertion message (pl\_assertMessage) as well.  
To use assertion, optimized build has to be turned on with the –O2 switch and NDEBUG shall not be defined.  
By defining NDEBUG, the main block of f\_assert will be empty, therefore in case of enabled optimized build (-O2) the function won’t be compiled with the generated code. This results higher debug capability with higher performance.

### Error messages

None.

### Warning messages

None.

### Examples

None

## Conversion Functions

### Overview

This set of functions enables you to apply conversions from TTCN-3 that have been not supported by TTCN-3 before e.g.: universal charstring conversions.

These Assertion Functions were tested under Linux/Solaris/Cygwin platforms.

Conversion Functions are implemented in the following files:

* TCCConversion\_Functions.ttcn
* TCCConversion.cc

### Function definitions

The following Conversion Functions are defined. For the exact API check the generated NaturalDocs documentation.

**external function f\_putInLowercase(charstring pl\_string)   
return charstring;**

Convert a charstring value to lowercase.

**external function f\_putInUppercase(charstring pl\_string)   
return charstring;**

Convert a charstring value to uppercase.

**function f\_unichar2charstr (in universal charstring p\_unichar) return charstring**

Convert a universal charstring to charstring.

**function f\_charstr2unichar (in universal charstring p\_unichar) return charstring**

Convert a charstring to universal charstring.

**function f\_replaceFirstOccurenceOfSubstring(  
 in charstring parInStr,  
 in charstring parSubStrA,  
 in charstring parSubStrB) return charstring**

Replace the first occurance of parSubStrA with parSubStrB in parInStr.

**function f\_replaceEveryOccurenceOfSubstring(  
 in charstring parInStr,  
 in charstring parSubStrA,  
 in charstring parSubStrB) return charstring**

Replace every occurance of parSubStrA with parSubStrB in parInStr.

**function f\_replaceFirstOccurenceOfPattern(  
 in charstring parInStr,  
 in charstring parSubStrA,  
 in charstring parSubStrB) return charstring**

Replace the first occurrence of pattern parSubStrA with parSubStrB in parInStr.

**function f\_replaceEveryOccurenceOfPattern(  
 in charstring parInStr,  
 in charstring parSubStrA,  
 in charstring parSubStrB) return charstring**

Replace every occurrence of pattern parSubStrA with parSubStrB in parInStr.

**external function f\_addOctetstring(in octetstring par1, in octetstring par2) return octetstring;**

Add two integer values represented in OCTETSTRING.   
The function can be used in case when an integer representation is needed with larger range then it is available in TTCN-3.   
Note that negative values are not handled.

**external function f\_subOctetstring(in octetstring par1,   
in octetstring par2)   
return octetstring;**

Subtract two integer values represented in OCTETSTRING.   
The function can be used in case when an integer representation is needed with larger range then it is available in TTCN-3.   
Note that negative values are not handled.

**external function f\_compOctetstring(in octetstring par1,   
in octetstring par2)   
return integer;**

Compares two integer values represented in OCTETSTRING.   
The function can be used in case when an integer representation is needed with larger range then it is available in TTCN-3.   
Note that negative values are not handled.  
Return 0 in case of identical input arguments.  
Return 1 if the first argument is larger.   
Return 2 if the second argument is larger.

**external function f\_substr\_token(in charstring str,  
in charstring begin\_token, in charstring end\_token)  
return charstring;**

**external function f\_substr\_token\_oct(in octetstring str,  
in octetstring begin\_token, in octetstring end\_token)  
return octetstring;**

The function returns a substring from a value. The starting and the ending points are defined by the begin and end tokens. If one of the tokens is not found it returns an empty string. If end\_token is an empty string, the function returns the part of the value after the begin\_token. If begin\_token is an empty string, the function returns the part of the value until the end\_token. If both of them empty string, the function returns the part whole value.

**function f\_substr\_all\_tokens(in charstring str, in charstring begin\_token, in charstring end\_token) return TCC\_Conversion\_ch\_list**

**function f\_substr\_all\_tokens\_oct(in octetstring str, in octetstring begin\_token, in octetstring end\_token) return TCC\_Conversion\_ch\_list**

The function returns a list of substring from a value. The starting and the ending points are defined by the begin and end tokens. If one of the tokens is not found it returns an empty string. If end\_token is an empty string, the function returns the part of the value after the begin\_token. If begin\_token is an empty string, the function returns the part of the value until the end\_token. If both of them empty string, the function returns the part whole value.

**external function f\_strstr(in charstring s1,  
in charstring s2, in integer offset:=0) return integer;**

**external function f\_strstr\_oct(in octetstring s1,  
in octetstring s2, in integer offset:=0) return integer;**

The f\_strstr function locates the first occurrence of the string s2 in string s1 and returns an index of starting pont of the located string, or -1 if the string is not found. If s2 is an empty, the function returns 0. The offset determines the starting point of the search. Any occurance of the s2 before the offset is ignored. The offset is optional.

**function f\_OctetIpv4(in charstring pl\_ip) return octetstring;**

Converts a IPv4 dot noted address to its octetstring representation. On error, ‘’O is returned.

**function f\_oct2str\_safe(in octetstring par1, out charstring par2) return boolean;**

Works as the built-in function oct2str of Titan, but in a fault tolerant way. In case of any fault, false is returned, otherwise true. The second parameter will contain the decoded string as long as it could be decoded.

**function f\_IPv6CreateLiteral(in charstring pl\_ip) return charstring**

Encloses parameter pl\_ip in square brackets. (Used to generate IPv6 literal address.)

**external function f\_isNumber(charstring pl\_string, integer pl\_number)   
return charstring;**

Checks, if the input string starts with digits. If yes, then it returns the integer value in pl\_number and the remaining string as return value. If the string does not start with a conversable number, it will return 0 in pl\_number.

**function f\_isWhiteSpace(in charstring pl\_str)   
return boolean**

Checks, if the input string consists of only white space characters. If the string only contains white space characters then it returns true, else it returns false. For an empty string it returns false.

**function f\_prePadString(charstring pl\_string, charstring pl\_pad, integer pl\_length, inout charstring pl\_result) return Boolean**

Pads a string to fit in a specified length (pl\_length). The pl\_string will be prepended by the first character of pl\_pad to reach the length specified by pl\_length.

### Error messages

None.

### Warning messages

None.

### Examples

The following code part shows an example for using replace pattern functions. The input string (replacestr) is "AaACCCAaABbB" and the pattern that every occurrence shall be replaced is "\*([Aa]#(3,3))\*". The substitution string is “<3A>”. So the aim of this short test is to change three-character long A or a sequences to <3A>. Notice that pattern shall match to the whole string and pattern to swap shall be grouped with opening and closing brackets (). Awaited response (rStr4) is <3A>CCC<3A>BbB.

testcase f\_test\_replaceEveryOccuranceOfPattern() runs on Conversion\_CT

{

log(" -------- f\_replaceEveryOccurenceOfPattern ------- ");

log("Original string: ", replacestr);

tmpStr :=

f\_replaceEveryOccurenceOfPattern(

replacestr, "\*([Aa]#(3,3))\*", "<3A>");

log("Replace every occurence of \*([Aa]#(3,3))\* to <3A>: ",

tmpStr);

if (tmpStr == rStr4) { setverdict(pass); }

else { setverdict(fail); }

}

### Makefile

External functions, f\_substr\_token\_oct and f\_strstr\_oct use memmem, so D\_GNU\_SOURCE flag should be added to the CPPFLAGS in case of Cygwin environment.

## Date and Time Functions

### Overview

Date and Time Functions enable you to handle date and time from TTCN-3.

These Date and Time Functions were tested under Linux/Solaris/Cygwin platforms.

Date and Time Functions are implemented in the following files:

* TCCDateTime\_Functions.ttcn
* TCCDateTime.cc

### Function definitions

The following Date and Time Functions are defined. For the exact API check the generated NaturalDocs documentation.

**external function f\_time() return integer;**

Provides the current calendar time of the system in seconds.

**external function f\_time\_ms() return integer;**

Provides the current calendar time of the system in milliseconds.

**external function f\_ctime(integer pl\_sec) return charstring;**

Convert a time value in seconds to human readable string in local system time.

**external function f\_ctime\_ms(integer pl\_msec) return charstring;**

Convert a time value in milliseconds to human readable string in local system time.

**external function f\_ctime\_UTC(integer pl\_sec) return charstring;**

Convert a time value in seconds to human readable string in UTC.

**external function f\_ctime\_ms\_UTC(integer pl\_msec) return charstring;**

Convert a time value in milliseconds to human readable string in UTC.

**external function f\_getTimeFormatted(integer pl\_sec, charstring pl\_format) return charstring;**

Provide the current calendar time in a formatted way.   
Format string:

|  |  |  |
| --- | --- | --- |
| **specifier** | **Replaced by** | **Example** |
| %a | Abbreviated weekday name \* | Thu |
| %A | Full weekday name \* | Thursday |
| %b | Abbreviated month name \* | Aug |
| %B | Full month name \* | August |
| %c | Date and time representation \* | Thu Aug 23 14:55:02 2001 |
| %d | Day of the month (01-31) | 23 |
| %H | Hour in 24h format (00-23) | 14 |
| %I | Hour in 12h format (01-12) | 02 |
| %j | Day of the year (001-366) | 235 |
| %m | Month as a decimal number (01-12) | 08 |
| %M | Minute (00-59) | 55 |
| %p | AM or PM designation | PM |
| %S | Second (00-61) | 02 |
| %U | Week number with the first Sunday as the first day of week one (00-53) | 33 |
| %w | Weekday as a decimal number with Sunday as 0 (0-6) | 4 |
| %W | Week number with the first Monday as the first day of week one (00-53) | 34 |
| %x | Date representation \* | 08/23/01 |
| %X | Time representation \* | 14:55:02 |
| %y | Year, last two digits (00-99) | 01 |
| %Y | Year | 2001 |
| %Z | Timezone name or abbreviation | CDT |
| %% | A % sign | % |

The specifiers whose description is marked with an asterisk (\*) are locale-dependent.

**external function f\_time2sec(integer pl\_year, integer pl\_mon, integer pl\_day, integer pl\_hour, integer pl\_min, integer pl\_sec) return integer;**

Convert date format to seconds since January 1, 1970. The date treated as local time.

**external function f\_time2sec\_UTC(integer pl\_year, integer pl\_mon, integer pl\_day, integer pl\_hour, integer pl\_min, integer pl\_sec) return integer;**

Convert date format to seconds since January 1, 1970. The date treated as UTC time.

**external function f\_getCurrentDateWithOffset(integer pl\_sec) return charstring;**

Generate a date from the actual date and time plus the parameter in seconds e.g. getSdate(30) will return a charstring containing the date and time of 30 seconds later

**external function f\_getCurrentGMTDate() return charstring;**

Return the current GMT date in format RFC 1123-Date ex: Sat Nov 13 23:30:02 2010

**external function f\_getCurrentGMTDate\_ms() return charstring;**

Return the current GMT date in format Www Mmm dd hh:mm:ss.SSS yyyy ex: Sat Nov 13 23:30:02.347 2010

**external function f\_tic() return integer;**

Return the number of clock ticks used by the application since the program was launched.  
Warning! Functionality depends on the used library. Man page shall be read for clock() function.

**external function f\_toc() return charstring;**

Return the elapsed seconds since time t.   
Warning! Functionality depends on the used library. Man page shall be read for clock() function. f\_toc() depends on f\_tic().

**external function f\_timeDiff(integer t\_stop, integer t\_start) return integer;**

Return the difference between two time values.

### Error messages

None.

### Warning messages

None.

### Examples

The following code part shows some examples for date and time functions.

The first test case returns the current time in seconds (e.g.: 1195459285):

testcase f\_test\_time() runs on DateTime\_CT

{

log(" ----------- f\_time ----------- ");

log("Time (f\_time): ", f\_time());

setverdict(pass);

}

The second test case converts time in seconds into time string. Note that time is in format Www Mmm dd hh:mm:ss yyyy and awaited result is current local time minus an hour because of subtraction of 3600 seconds from input.

testcase f\_test\_ctime() runs on DateTime\_CT

{

log(" ----------- f\_ctime ----------- ");

log("CTime (f\_ctime) - an hour before: ",

f\_ctime(f\_time() - 3600));

setverdict(pass);

}

The third test case returns time as specified in format string

(e.g.: 1195459285 + “%m %I:%M%p.” -> 11 **09:01**AM.)

testcase f\_test\_getTimeFormatted() runs on DateTime\_CT

{

log(" ----------- f\_getTimeFormatted ----------- ");

log("TimeFormatted (f\_getTimeFormatted): ",

f\_getTimeFormatted(f\_time(),"Now is %m %I:%M%p."));

setverdict(pass);

}

The fourth test case returns current time with offset seconds defined by input. (E.g.: with no input offset: Mon Nov 19 **09:01**:25 2007)

testcase f\_test\_getCurrentDateWithOffset() runs on DateTime\_CT

{

log(" -------- f\_getCurrentDateWithOffset -------- ");

log("Date (f\_ getCurrentDateWithOffset): ",

f\_getCurrentDateWithOffset(0));

setverdict(pass);

}

The fifth test case returns GMT time. Notice bolded time value compared to f\_getTimeFormatted() and f\_getCurrentDateWithOffset().

(e.g: 1195459285 -> Mon Nov 19 **08:01**:25 2007)

testcase f\_test\_getCurrentGMTDate() runs on DateTime\_CT

{

log(" ----------- f\_getCurrentGMTDate ----------- ");

log("GMT Date (f\_getCurrentGMTDate): ",

f\_getCurrentGMTDate());

setverdict(pass);

}

## Maths Functions

### Overview

Maths functions enable you to use mathematical functions straightforward from TTCN-3.

These Maths Functions were tested under Linux/Solaris/Cygwin platforms.

Maths Functions are implemented in the following files:

* TCCMaths\_Functions.ttcn
* TCCMaths\_GenericTypes.ttcn
* TCCMaths.cc

### Function definitions

The following Maths Functions are defined.

**external function f\_maxIL(in IntegerList ilist) return IntegerList;**

Return an IntegerList with the highest number found at index 0 and the index of ilist where it's found at index 1.

**external function f\_maxFL(in FloatList flist) return FloatList;**

Return a FloatList with the highest number found at index 0 and the index of flist where it's found at index 1.

**external function f\_minIL(in IntegerList ilist) return IntegerList;**

Return an IntegerList with the lowest number found at index 0 and the index of ilist where it's found at index 1.

**external function f\_minFL(in FloatList flist) return FloatList;**

Return a FloatList with the lowest number found at index 0 and the index of flist where it's found at index 1.

**external function f\_averageFL(in FloatList flist) return float;**

Return the average of flist.

**external function f\_averageIL(in IntegerList ilist) return float;**

Return the average of ilist.

**external function f\_updateFL(inout FloatList head, in FloatList tail);**

Append tail to the end of head (head return as inout).

**external function f\_updateIL(inout IntegerList head, in IntegerList tail);**

Append tail to the end of head (head return as inout).

**external function f\_stdFL(in FloatList flist) return float;**

Return the normalized standard deviation of flist (so the average square distance from the centre of points).

For example, let there be a list of list = {2.0, 4.0} and thereafter the average of elements is u = (2.0 + 4.0) / 2.0. The result of the function is afterwards the square root of the average square distance, so ( (2.0 - u)^2 + (4.0 - u)^2 ) / len, where len means the length of the list.  
  
**external function f\_stdFLL(in FloatList flist, in float u) return float;**

Return the normalized standard deviation of flist from a user defined center value (so the average square distance from a user defined centre value).  
  
For example let there be a list of list = {2.0, 4.0} and thereafter the user specifies a value. This value is named as u. The result of the function is afterwards the square root of the average square distance from u, so ( (2.0 - u)^2 + (4.0 - u)^2 ) / len, where len means the length of the list.

**external function f\_stdIL(in IntegerList ilist) return float;**

Return the normalized standard deviation of ilist (so the average square distance from the centre of points).  
  
For example let there be a list of list = {2, 4} and thereafter the average of elements is u = (2 + 4) / 2. The result of the function is afterwards the square root of the average square distance, so ( (2 - u)^2 + (4 - u)^2 ) / len, where len means the length of the list.

**external function f\_stdILL(in IntegerList ilist, in float u) return float;**

Return the normalized standard deviation of ilist (so the average square distance from the center of points).  
Note: u is the average value of flist and has to be calculated before a call to this function.  
  
For example let there be a list of list = {2, 4} and thereafter the user specifies a value. This value is named as u. The result of the function is afterwards the square root of the average square distance from u, so ( (2 - u)^2 + (4 - u)^2 ) / len, where len means the length of the list.

**external function f\_sinVL(  
 in float freq,   
 in float altitude,   
 in float start\_val,   
 in integer len,   
 in float step) return FloatList;**

Return the values of the sine function. The computation is the following: altitude \* sin(2\*π\*freq\*start\_val) then start value is increased by step in every iteration till a number of len sine values are achieved.

**external function f\_cosVL(  
 in float freq,   
 in float altitude,   
 in float start\_val,   
 in integer len,   
 in float step) return FloatList;**

Return the values of the cosine function. The computation is the following: altitude \* cos(2\*π\*freq\*start\_val) then start value is increased by step in every iteration till a number of len cosine values are achieved.

.

**external function f\_sin(in float angle) return float;**

Return the sine of angle radians.

**external function f\_cos(in float angle) return float;**

Return the cosine of angle radians.

**external function f\_asin(in float val) return float;**

Return the arc sine of value in [-π/2, + π/2].

**external function f\_acos(in float val) return float;**

Return the arc cosine of value in [0, π].

**external function f\_powFF(in float base, in float expo) return float;**

Raise to power (float to float power).

**external function f\_powII(in integer base, in integer expo) return integer;**

Raise to power (integer to integer power).

**external function f\_powIF(in integer base, in float expo) return float;**

Raise to power (integer to float power).

**external function f\_powFI(in float base, in integer expo) return float;**

Raise to power (float to integer power).

**external function f\_sqrF(in float base) return float;**

Raise a float value to square.

**external function f\_sqrI(in integer base) return integer;**

Raise an integer value to square.

**external function f\_sqrtF(in float base) return float;**

Square root of a float value.

**external function f\_sqrtI(in integer base) return float;**

Square root of an integer value.

**external function f\_ceil(in float val) return integer;**

Return the smallest integer value that is not less then value.

**external function f\_floor(in float val) return integer;**

Return the largest integer value that is not greater then value.

**external function f\_exp(in float val) return float;**

Return the exponential value of the argument.

**external function f\_log(in float val) return float;**

Return the natural logarithm of the argument.

**function** f\_generate\_poissonTable(**in** **float** p\_lambda, **out** t\_Poisson\_Table p\_pTable) **return** **boolean**;

Generate the Poisson cumulative probability distribution list for a given lambda and returns true if the generation was success. If the lambda is below 0, the function returns false and the Poisson table will contain false values. With the implemented method we can only generate correct values for lambda-s smaller than about 100-110. It is because the elements above about 300 in the table will be 1.0 due to rounding errors. The table will contain lambda \* 2 elements.

**function** f\_getNext\_poissonValue(**in** **float** p\_lambda, **inout** t\_Poisson\_Table p\_pTable, **out** **boolean** p\_sucess) **return** **integer**

Return a random Poisson value by the given Poisson table and lambda. Regenerate the table if it is necessary.

**function** f\_gen\_Exponential\_Distribution(**in** **float** p\_lambda) **return** **float**

Return the exponential random variable for a given lambda.

### Error messages

None.

### Warning messages

None.

### Examples

The first example shows usage of f\_sinVL function. Let the following definitions be valid:

var float freq := 10.0; // frequency

var float alti := 5.0; // altitude

var float startval := 0.0; // start value

var float step := 0.37; // steplength

var integer len := 5; // length of returned value list

The function computes sine values as altitude \* sin(2 \* pi \* freq \* start\_val) and then start value is increased with step parameter. The returned value list will consist the values computed this way and a number of len values will be stored in that list.

The awaited result in this case is:

{0.0, -4.755283, 2.938926, 2.938926, -4.755283}

testcase f\_test\_sinVL() runs on Maths\_CT

{

log(" ----------- f\_sinVL ----------- ");

log("SINUS (Frequency: ", freq,

", altitude: ", alti,

", startval: ", startval,

", step: ", step,

", return length: ", len, "):");

fres := f\_sinVL(freq, alti, startval, len, step);

log("Sinus values returned: ", fres);

// check

sum := 0.0;

for (var integer i := 0; i < len; i := i + 1)

{

tmpFloat := fres[i] - sin1[i];

sum := sum + tmpFloat \* tmpFloat;

}

if (sum < LIMIT) { setverdict(pass); }

else { setverdict(fail); }

}

The second test case computes the normalized standard deviation from a user defined value of a float list. This case the user defined value is the average of values in the input list, so the awaited result is the same as f\_stdFL().

testcase f\_test\_stdFLL() runs on Maths\_CT

{

log(" ----------- f\_stdFLL ----------- ");

log("Original float list: ", flist1);

tmpFloat := f\_stdFLL(flist1, f\_averageFL(flist1));

log("Normalized, standard derivation (FLL): ", tmpFloat);

if ((tmpFloat - stdFLL1) \* (tmpFloat - stdFLL1) < LIMIT) {

setverdict(pass);

}

else {

setverdict(fail);

}

}

The third test case returns the value and the position of the maximal element in an integer list.

Return value for ilist1 = {1, 2, 3, 4, 5} is maxIL1 = {5, 4} as the maximal element is 5 and its index in the list is 4.

testcase f\_test\_maxIL() runs on Maths\_CT

{

log(" ----------- f\_maxIL ----------- ");

log("Original integer list: ", ilist1);

ires := f\_maxIL(ilist1);

log("MaxIL (maxVal, maxIdx): ", ires);

if (ires == maxIL1) { setverdict(pass); }

else { setverdict(fail); }

}

## Security Functions

### Overview

Security Functions enable you to use some security related functions from TTCN-3.

These Security Functions were tested under Linux/Solaris/Cygwin platforms.

Security Functions are implemented in the following files:

* TCCSecurity\_Functions.ttcn
* TCCSecurity.cc

### Function definitions

The pl\_OP input of the following functions is the Operator Variant OP that configures the AKA Algorithms to generate the proper keys for the Operator who will verify the generated keys.

The following Security Functions are defined. For the exact API check the generated NaturalDocs documentation.

**external function f\_calculateDigestResponse(  
 charstring nonce,  
 charstring cnonce,  
 charstring user,  
 charstring realm,  
 charstring passwd,  
 charstring alg,  
 charstring nonceCount,  
 charstring method,  
 charstring qop,  
 charstring URI,  
 charstring HEntity) return charstring;**

Calculate digest response.  
Support HTTP authentication (detailed description in RFC 2617) using uses one-way hash (md5) specified in RFC 1321. When a request arrives to server for an access-protected object, it responds an "401 Unauthorized" status code and a WWW-Authenticate header (encapsulate nonce and other necessary parameters). The client is expected to retry the request, passing an Authorization header with response field calculated with f\_calculateDigestResponse().   
Overview: <http://en.wikipedia.org/wiki/Digest_access_authentication>

**external function f\_calculateDigestHA1(  
 charstring nonce,  
 charstring cnonce,  
 charstring user,  
 charstring realm,  
 charstring passwd,  
 charstring alg) return charstring;**

Calculate digest H(A1) hash (detailed description in RFC 2617).  
  
**external function f\_calculateMD5(in charstring pszHashInput) return charstring;**

Calculate the md5 hash value of pszHashInput. The result is the hash converted to hexa string.

**external function f\_calculateMD5\_oct(in octetstring pszHashInput) return octetstring;**

Calculate the md5 hash value of pszHashInput.

**external function f\_calculateHMACMD5(in octetstring msg, in OCT\_64 key) return octetstring;**

Calculate the HMAC MD5 hash value of msg with the encryption key. The result is the hash octetstring which is 16 octet long.

**external function f\_calculate\_HMAC\_MD5(in octetstring pl\_key, in octetstring pl\_input, in integer pl\_length) return octetstring;**

Calculate the HMAC MD5 value of a message with specified key.

**external function f\_calculate\_HMAC\_SHA1(in octetstring pl\_key, in octetstring pl\_input, in integer pl\_length) return octetstring;**

Calculate the HMAC SHA1 value of a message with specified key.

**external function f\_calculate\_HMAC\_SHA256(in octetstring pl\_key, in octetstring pl\_input, in integer pl\_length) return octetstring;**

Calculate the HMAC SHA256 value of a message with specified key.

**external function f\_calculateRAND\_oct(in integer pl\_length) return octetstring;**

Generate a random octetstring via OpenSSL

**external function f\_calculateSHA1\_oct(in octetstring pszHashInput) return octetstring;**

Compute SHA1 hash value via OpenSSL in octetstring format

**external function f\_calculateSHA1(in charstring pszHashInput) return charstring;**

Compute SHA1 hash value via OpenSSL

**external function f\_AES\_CBC\_Encrypt\_OpenSSL(octetstring p\_key, octetstring p\_iv, octetstring p\_data) return octetstring;**

Calculate via OpenSSL the AES CBC encrypted value with arbitrary key length up to 128 bits

**external function f\_AES\_CBC\_Decrypt\_OpenSSL(octetstring p\_key, octetstring p\_iv, octetstring p\_data) return octetstring;**

Calculate via OpenSSL the AES CBC decrypted value with arbitrary key length up to 128 bits

The IMSAKA related functions have two forms:

* One which takes OP as input parameter
* One which works with OPc insytead of OP

The later ones are post fixed with “\_opc”

**external function ef\_3DES\_ECB\_Decrypt(in octetstring pl\_data, in octetstring pl\_key, in boolean pl\_use\_padding:=true) return octetstring;**

It encrypts the data using the triple DES algorithm in ECB (Electronic Codebook) mode taking key pl\_key. The pl\_use\_padding parameter controls the usage of standard PKCS padding.

**external function ef\_3DES\_ECB\_Decrypt(in octetstring pl\_data, in octetstring pl\_key, in boolean pl\_use\_padding:=true) return octetstring;**

Decrypts triple DES ECB encrypted data using pl\_key. The pl\_use\_padding parameter controls the usage of standard PKCS padding.

**external function ef\_3DES\_CBC\_Encrypt(in octetstring pl\_data, in octetstring pl\_key, in octetstring pl\_iv, in boolean pl\_use\_padding:=true) return octetstring;**

It encrypts the data using the triple DES algorithm in CBC (Cipher-block chaining) mode taking key pl\_key. The pl\_use\_padding parameter controls the usage of standard PKCS padding.

**f\_3DES\_CBC\_Encrypt(octetstring p\_key, octetstring p\_iv, octetstring p\_data) return octetstring**

Wrapper function for above **ef\_3DES\_CBC\_Encrypt**

**external function ef\_3DES\_CBC\_Decrypt(in octetstring pl\_data, in octetstring pl\_key, in octetstring pl\_iv, in boolean pl\_use\_padding:=true) return octetstring;**

Decrypts triple DES CBC encrypted data using pl\_key. The pl\_use\_padding parameter controls the usage of standard PKCS padding.

**function f\_3DES\_CBC\_Decrypt(octetstring p\_key, octetstring p\_iv, octetstring p\_data) return octetstring**

Wrapper function for above **ef\_3DES\_CBC\_Decrypt**

**external function ef\_Calculate\_AES\_XCBC\_128(in octetstring pl\_data, in octetstring pl\_key, in integer pl\_out\_length) return octetstring;**

Calculates the AES XCBC value from pl\_data and pl\_key. AES XCBC generates a value that is 16 bytes long and this can be truncated to the length given in pl\_out\_length.

**external function ef\_DH\_shared\_secret(in octetstring pl\_pubkey, in octetstring pl\_privkey) return octetstring;**

Computes the shared secret for the Diffie-Hellman exchange given the private key of the originating side and the public key of the responding side. The keys must be 96 or 128 or 256 bytes long. (DH MODP group 768 and 1024 and 2048)

**external function ef\_DH\_generate\_private\_public\_keys (in integer pl\_keyLength, inout octetstring pl\_pubkey, inout octetstring pl\_privkey) return integer;**

Computes the shared secret from the originating side's private key and the public key of the responding side as described in DH group 2 and 14. Keys must be either 96, 128 or 256 bytes long.

### Error messages

None.

### Warning messages

None.

### Examples

The following example shows a basic usage of digest response calculation. The provided data values are acquired from the server response and client data. Note that qop is equal to auth and therefore HEntity is the md5 hash value of empty string.

There is a sample test case for AKA key generation functions and EEA3 and EIA3 calculation.

module TCCSecurity\_Example

{

import from TCCSecurity\_Functions all;

type component Security\_CT {

var DigestData dg;

var AKAInput aka\_input;

var SIMOutput sim\_output;

};

//////////////////////////////////////////////////////////////

// Security functions

//////////////////////////////////////////////////////////////

type record DigestData {

charstring nonce,

charstring cnonce,

charstring user,

charstring realm,

charstring passwd,

charstring alg,

charstring nonceCount,

charstring method,

charstring qop,

charstring URI,

charstring HEntity

}

type record AKAInput {

octetstring k,

octetstring rand,

octetstring sqn,

octetstring amf

}

type record SIMOutput {

octetstring sres,

octetstring kc

}

// test f\_calculateDigestResponse function

testcase f\_test\_digestresponse() runs on Security\_CT

{

log(" --------- f\_calculateDigestResponse --------- ");

dg := {

nonce := "dcd98b7102dd2f0e8b11d0f600bfb0c093",

cnonce := "0a4f113b",

user := "Mufasa",

realm := "testrealm@host.com",

passwd := "password",

alg := "MD5",

nonceCount := "00000001",

method := "GET",

qop := "auth",

URI := "/dir/index.html",

// MD5 hash of entity body

HEntity := "d41d8cd98f00b204e9800998ecf8427e"

}

log("Data to compute digest from: ", dg);

log("Digest response: ",

f\_calculateDigestResponse(

dg.nonce,

dg.cnonce,

dg.user,

dg.realm,

dg.passwd,

dg.alg,

dg.nonceCount,

dg.method,

dg.qop,

dg.URI,

dg.HEntity));

// no check

setverdict(pass);

}

3DES ECB encrypt decrypt

var octetstring vl\_key := '000102030405060708090A0B0C0D0E0F0011223344556677'O

var octetstring vl\_data := '61A7D3BEAA4C7DBD1FA3B2237A8CC92AE6575BADF894A34643D74AFF502BF523973DF1637453388CC4C06EC02D8AB44E3EDC866555BC0FDC56D0B2029110D7A1A1F585B65FB84D3674EC87B30BF5E7E8F747330549C77A9CCD348B7898825302408CE53D25ED62D56CA0FFACC0DFECDC9CFCBD03279047'O

var octetstring vl\_encrypted\_data := ef\_3DES\_ECB\_Encrypt(vl\_data, vl\_key);

var octetstring vl\_decrypted\_data := ef\_3DES\_ECB\_Decrypt(vl\_encrypted\_data, vl\_key);

3DES CBC encrypt decrypt

var octetstring vl\_data := '61A7D3BEAA4C7DBD1FA3B2237A8CC92AE6575BADF894A34643D74AFF502BF523973DF1637453388CC4C06EC02D8AB44E3EDC866555BC0FDC56D0B2029110D7A1A1F585B65FB84D3674EC87B30BF5E7E8F747330549C77A9CCD348B7898825302408CE53D25ED62D56CA0FFACC0DFECDC9CFCBD03279047'O

var octetstring vl\_key := '000102030405060708090A0B0C0D0E0F0011223344556677'O

var octetstring vl\_iv := '0001020304050607'O

var octetstring vl\_encrypted :=   
ef\_3DES\_CBC\_Encrypt(vl\_data, vl\_key, vl\_iv);

var octetstring vl\_decrypted := ef\_3DES\_CBC\_Decrypt(vl\_encrypted, vl\_key, vl\_iv);

AES XCBC 128 with 20 bytes of data

var octetstring vl\_data := '000102030405060708090a0b0c0d0e0f10111213'O

var octetstring vl\_key := '000102030405060708090a0b0c0d0e0f'O

var octetstring vl\_computed\_value := ef\_Calculate\_AES\_XCBC\_128(vl\_data, vl\_key, 16);

Diffie-Hellman secret key

var octetstring vl\_key\_other := '73DF1637453388CC4C06EC02D8AB44E3EDC866555BC0FDC56D0B2029110D7A1A1F585B65FB84D3674EC87B30BF5E7E8F747330549C77A9CCD348B7898825302408CE53D25ED62D56CA0FFACC0DFECDC9CFCBD03279047E90E4E5013F173AE9E61A7D3BEAA4C7DBD1FA3B2237A8CC92AE6575BADF894A34643D74AFF502BF5239'O

var octetstring vl\_priv\_key := '9FCF731129397EF7DC51562D71DD819DE35891B739BD6BE7A5177F676F1A06775A0E915E3758130FE5493B95C7A67F11F45E4C541DDD2834E4A9248F18EE2597936499C97D25227C4A4B78BABD4F33BBC3E3A2C591369A3A4FAF3D851839249E23E90D15051268986E562D76D756F7FAF942FFBC4669199A3C04E31335E2BD70'O

var octetstring vl\_shared\_secret := ef\_DH\_shared\_secret(vl\_key\_other, vl\_priv\_key);

## Message Handling Functions

### Overview

Message handling functions enable you to get the length of the received message.

Message Handling Functions are implemented in the following files:

* TCCMessageHandling\_Functions.ttcn
* TCCMessageHandling.cc

### Function definitions

The following Message Handling Functions shall be defined:

**external function f\_** **TCCMessageHandling\_getMessageLength (in octetstring) return integer;**

Return the length of the given message. This function is searching the Content-Length filed in the given character based message and returns with the value of this field. This function can be used for example by the HTTP and SIP message transport.

**external function f\_** **TCCMessageHandling\_getMessageLength4Diameter (in octetstring) return integer;**

Return the length of the given message. This function can be used for Diameter message transport.

**external function f\_** **TCCMessageHandling\_getMessageLength4Radius (in octetstring) return integer;**

Return the length of the given message. This function can be used for Radius message transport.

**external function f\_** **TCCMessageHandling\_getMessageLength4BER (in octetstring) return integer;**

Return the length of the given BER encoded message.  
For definite length format the length is determined by parsing the Length field.  
For indefinite length format the whole TLV message is parsed.  
The function returns -1 if the Length cannot be determined (For definite length format the message segment does not contain the whole length field. For indefinite length format the message is not a complete TLV). Note that the input can be a message segment or multiple concatenated messages as long as the beginning of the input is the beginning of a TLV encoded BER message.   
This function is useful for TCP transport in the IPL4 test port.

### Error messages

None.

### Warning messages

None.

### Examples

module TCCMessageHandling\_Example {

import from TCCMessageHandling\_Functions all;

control

{

//the call of the function

f\_TCCMessageHandling\_getMessageLength(''O);

}

}

## Permutated Parameters Handling

### Overview

Permutated parameters functions enable you to determine permutations of multiple parameters. Parameters are assigned with their types and values in assignment lists. Permutation of the parameters are calculated with the available functions.

Permutation example:

Types and values:

• OPN (other party number): OPN\_1, OPN\_2

• TOD (Time Of Day): TOD\_1 (Monday morning), TOD\_2 (Saturday noon)

• CGI (Cell Global Identifier): CGI\_1, CGI\_2

• DUR (Call Duration): DUR\_1, DUR\_2

Generated permutations:



Permutated parameters handling Functions are implemented in the following files:

* TCCPermutatedParameters\_Definitions.ttcn
* TCCPermutatedParameters\_Functions.ttcn

### Function definitions

**function** f\_PP\_getPermutatedParams(  
 **in** **integer** pl\_permutationIdx,  
 **in** EPTF\_CharstringList pl\_paramTypes,  
 **in** PP\_ParamSet pl\_assignmentList  
) **return** PP\_ParamSet

Returns a parameter set calculated from the index of the permutation. The values of the parameters are composed into the prefix field.

**function** f\_PP\_countPermutations(  
 **in** EPTF\_CharstringList pl\_paramTypes,  
 **in** PP\_ParamSet pl\_assignmentList  
) **return** **integer**

Calculates the permutations of the parameters, using the parameter types as filters. Permutations are calculated by multiplying the number of parameters belonging to different types.

**function** f\_PP\_divideValuesOfTypes(  
 **in** EPTF\_CharstringList pl\_paramTypes,  
 **in** PP\_ParamSet pl\_assignmentList,  
 **in** **integer** pl\_LGenIdx,  
 **in** **integer** pl\_noOfLGens) **return** PP\_ParamSet

Function returns a portion of parameters. The portion is calculated from the portion (ie LGen) index and number of portions (ie number of LGens).

**function** f\_PP\_countValuesOfTypes(

**in** EPTF\_CharstringList pl\_paramTypes,

**in** PP\_ParamSet pl\_assignmentList

) **return** **integer** {

Returns the number of values present in the assignment list, filtered for the types requested.

**public** **function** f\_PP\_normalizeParams(  
**in** PP\_ParameterAssignmentList pl\_assignmentList,  
**out** PP\_ParamSet pl\_normalizedParams)

Function to be used on assignment list to group the values belonging to the same type together. The functions below can only work on normalized parameters.

**public** **function** f\_PP\_getParamValueFromSet(  
 **in** PP\_ParamSet pl\_paramSet,  
 **in** EPTF\_CharstringList pl\_typeList  
) **return** EPTF\_CharstringList

Function returns the single parameter value if its parameter types matches one of the given types.

### Error messages

None.

### Warning messages

None.

### Examples

None. Please contact ‘ttcn3 (ETH)’ mailing list to get advices on usage.

## Titan Metadata Functions

### Overview

Titan Metadata functions enable you to get meta information provided by the Titan compiler directly from TTCN-3.

These Titan Metadata Functions were tested under Linux/Solaris/Cygwin platforms.

Titan Metadata Functions are implemented in the following files:

* TCCTitanMetadata\_Functions.ttcn
* TCCTitanMetadata.cc

### Function definitions

The following Metadata Functions are defined. For the exact API check the generated NaturalDocs documentation.

**external function f\_assert(boolean pl\_predicate);**

Return the compilation time of module

### Error messages

None.

### Warning messages

None.

### Examples

The following example shows the compilation time of module e.g.: "Nov 17 2007 17:14:09.

module TCCTitanMetadata\_Example

{

import from TCCTitanMetadata\_Functions all;

type component Metadata\_CT

{

}

//////////////////////////////////////////////////////////////

// TitanMetadata functions

//////////////////////////////////////////////////////////////

// test f\_time function

function f\_test\_compilationTime() runs on Metadata\_CT

{

log(" ----------- f\_compilationTime ----------- ");

log("Compilation time: ", f\_compilationTime());

}

// test all TitanMetadata functions

testcase tc\_TitanMetadata\_execAll() runs on Metadata\_CT

{

log(" ################# TITAN MetaData ################# ");

f\_test\_compilationTime();

setverdict(pass);

}

}

## Text Template Substitution Functions

### Overview

This module enables to do string substitutions on a text template accord to a given dictionary. The dictionary tells what to substitute to a new string, but not only strings can be in the dictionary, but simple patterns too, and the may contain function references to calculate the new string.

Message Handling Functions are implemented in the following files:

* TCCTemplate\_Functions.ttcn

Patterns can be used as dictionary keys. Every character will match itself, but the “\w” meta string will match one or more characters in the text template.  
The substrings matched with “\w” will be passed to the function referenced in the dictionary as a list of charstrings.

### Function definitions

The following Template Functions shall be defined:

**function f\_Template\_substitutetemplate(in TCCSubstitutionList pl\_dict, in charstring pl\_string) return charstring**

This function applies the dictionary to its parameter pl\_string, and gives the result back.

**function f\_Template\_subsfiletemplate(in TCCSubstitutionList pl\_dict, in charstring pl\_file) return charstring**

The function reads in the given files content, applies the dictionary to it, and gives the result back.

### Error messages

**"Error while closing file!"**May happen when **f\_Template\_subsfiletemplate** is used, and the file can not be closed. “” returned in this case.

### Warning messages

None.

### Examples

function nlist(in charstringList pl\_params) return charstring{

// log("params: ",pl\_params,"\n");

if (sizeof(pl\_params)==3 and pl\_params[2]=="R"){

return int2str(6453);

} else { return int2str(9);}

}

log(f\_Template\_subsfiletemplate({{"$(caiSs)","CAISS",omit},{"$(caiSq)","CAISQ", omit},{"$(<\\w..\\w>[\\w])","", refers(nlist)}},"cai3g\_delete\_PoCGroupsXDMSUser.body"));

## IPsec Functions

### Overview

The IPsec Functions provides a TTCN interface to handle the IPsec parameters stored in two databases: the Security Associations Database (SAD) and the Security Policy Database (SPD). Both the PF\_KEY API and the Netlink XFRM API is supported with TTCN interfaces via different functions.

#### PF\_KEY API

IPsec Handling Functions are implemented in the following files:

* TCCIPsec\_Definitions.ttcn
* TCCIPsec\_Functions.ttcn
* TCCIPsec.cc

These functions can only be used if the kernel supports IPsec. Moreover, SPD handling functions can only be used if the kernel contains the KAME IPsec implementation.  
To use all functions the “-DUSE\_KAME\_IPSEC” compiler switch should be added to the CPPFLAGS variable in the Makefile.  
To use only the portable part (SAD handling) the “-DUSE\_KAME\_IPSEC” compiler switch should be added to the CPPFLAGS variable in the Makefile.

For using any of these functions root privileges are needed.

#### Netlink XFRM API

IPsec Handling Functions are implemented in the following files:

* TCCIPses\_XFRM\_Definitions.ttc
* TCCIPsec\_XFRM.hh
* TCCIPsec\_XFRM.cc
* TCCIPsec\_XFRM\_SA.cc
* TCCIPsec\_XFRM\_SP.cc

As only very old kernel versions do not support the XFRM interface, no extra build flags are required.

For using any of these functions root privileges are needed.

### Function definitions

#### Security Parameter Index query function

The Security Parameter Index (SPI) is an identification tag added to the header while using IPsec for tunneling the IP traffic. This tag helps the kernel discern between two traffic streams where different encryption rules and algorithms may be in use.

The kernel keeps track of the SPIs in use, and it does not allow creating multiple Security Associations with the same SPI. For this reason, it is recommended to query a unique SPI from the kernel before creating a new Security Association.

##### PF\_KEY API

**external function f\_IPsec\_SPI\_get (  
 in charstring srcAddress,  
 in charstring dstAddress,  
 in TCCIPsec\_Protocol protocol,  
 out integer spi  
 ) return TCCIPsec\_Result;**

Queries a unique SPI from the kernel and returns it in the ‘spi’ out parameter.

##### Netlink XFRM API

**external function f\_XFRM\_allocate\_SPI(**

**in AllocSPI\_Info pl\_AllocSPI\_info,**

**inout integer pl\_spi**

**) return XFRM\_Result;**

Queries a unique SPI value from the kernel that is from the default SPI range or from a defined range in the pl\_AllocSPI\_info parameter. The execution result returned in by the function and the reserved SPI will be in the pl\_spi parameter.

#### Security Association Database handling functions

Security Associations are identified by the source address (srcAddress), security protocol (protocol) and the Security Parameter Index (SPI).

##### PF\_KEY API

Security Associations may contain various extensions (extensionList) beside the algorithms (alg). The algorithm identifies the security or integrity protocol and the keys. The extensions include the soft and hard lifetime and an id to link the SAs to SPs.

IPSec supports two modes: transport, tunnel. It can be specified in the last IPSecMode parameter.

The useNatt parameter can be used to enable NAT traversal support specified in UDP Encapsulation of IPsec ESP Packets RFC 3948.

The functions return an error code to indicate success or the type of the error.

**external function f\_IPsec\_SADB\_add (  
 in charstring srcAddress,  
 in charstring dstAddress,  
 in TCCIPsec\_Protocol protocol,  
 in integer spi,  
 in TCCIPsec\_ExtensionList extensionList := {},  
 in TCCIPsec\_Algorithm alg  
 in boolean useNatt := false,  
 in TCCIPsec\_IPsecMode ipSecMode := anyMode  
) return TCCIPsec\_Result;**

Add an SA (Security Association) into the SAD (Security Association Database).

**external function f\_IPsec\_SADB\_delete (  
 in charstring srcAddress,  
 in charstring dstAddress,  
 in TCCIPsec\_Protocol protocol,  
 in integer spi  
) return TCCIPsec\_Result;**

Delete an SA from the SAD.

**external function f\_IPsec\_SADB\_flush () return TCCIPsec\_Result;**

Delete all SAs from the SAD.

##### Netlink XFRM API

The main attributes of the SAs are the communication addresses, the used protocol, the used IPsec protocol, the mode of usage (transport and tunneling mode), the SPI value, algorithms and keys. In addition, the NAT (Network Address Traversal), selectors (identification purpose) and the lifetime is also configurable.

Each function returns an integer value, that represents the outcome of the execution. If it is zero, the execution was successful. If the execution fails, an additional string is returned that contains details on the cause of the failure. If the failure was on the application side, the error describes that, while if the error was on the kernel side processing, it contains the error message received from the kernel.

To add an SA the following function may be used:

**external function f\_XFRM\_add\_sa(**

**in SAAddInfo pl\_sa\_info**

**) return XFRM\_Result;**

To delete an SA, the following function may be used:

**external function f\_XFRM\_delete\_sa(**

**in SADelInfo pl\_sa\_info**

**) return XFRM\_Result;**

To flush the complete SADB, the following function may be used:

**external function f\_XFRM\_flush\_sa()**

**return XFRM\_Result;**

#### Security Policy Database handling functions

Security Policies are identified by the source and destination addresses ranges and port values, the transport protocol (UDP or TCP) and the direction of the data traffic to which the policy is applied.

##### PF\_KEY API

The rule parameter specifies how IP packets should be handled.

**external function f\_IPsec\_SPDB\_add (  
 in charstring srcAddress,  
 in integer srcPrefixLen :=  
 c\_TCCIPsec\_prefixAll,  
 in integer srcPort := c\_TCCIPsec\_anyPort,  
 in charstring dstAddress,  
 in integer dstPrefixLen :=  
 c\_TCCIPsec\_prefixAll,  
 in integer dstPort := c\_TCCIPsec\_anyPort,  
 in TCCIPsec\_TranspProto transpProto := anyTranspProto,  
 in TCCIPsec\_PolicyDirection dir,  
 in TCCIPsec\_PolicyRule rule  
) return TCCIPsec\_Result;**

Add an SP (Security Policy) into the SPD (Security Policy Database).

**external function f\_IPsec\_SPDB\_delete (  
 in charstring srcAddress,  
 in integer srcPrefixLen :=  
 c\_TCCIPsec\_prefixAll,  
 in integer srcPort := c\_TCCIPsec\_anyPort,  
 in charstring dstAddress,  
 in integer dstPrefixLen :=  
 c\_TCCIPsec\_prefixAll,  
 in integer dstPort := c\_TCCIPsec\_anyPort,  
 in TCCIPsec\_TranspProto transpProto := anyTranspProto,  
 in TCCIPsec\_PolicyDirection dir  
) return TCCIPsec\_Result;**

Delete an SP from the SPD.

**external function f\_IPsec\_SPDB\_flush () return TCCIPsec\_Result;**

Delete all SPs from the SPD.

##### Netlink XFRM API

The main parameter of the SPs are the communication addresses, the transport protocol, the direction and the template list that identifies the SP. The template contains the SPI, the IPsec protocol and additional information.

To add an SP, the following function may be used:

**external function f\_XFRM\_add\_policy(**

**in SPAddInfo pl\_pol\_info**

**) return XFRM\_Result;**

To delete an SP, the following function may be used:

**external function f\_XFRM\_delete\_policy(**

**in SPDelInfo pl\_pol\_info**

**) return XFRM\_Result;**

To flush the SPDB, the following function may be used:

**external function f\_XFRM\_flush\_policy()**

**return XFRM\_Result;**

### Error messages

None

### Warning messages

“TCCIPsec: f\_\_IPsec\_\_SADB\_\_add: IPsec support was not specified during compilation”

The “-DUSE\_IPSEC” or “-DUSE\_KAME\_IPSEC” compiler switch should be added to the CPPFAGS variable in the Makefile if the kernel supports IPsec.

"TCCIPsec: f\_\_IPsec\_\_SPDB\_\_add: IPsec SPDB support was not specified during compilation"

The “-DUSE\_KAME\_IPSEC” compiler switch should be added to the CPPFLAGS variable in the Makefile in order to use SPD handling if the kernel contains the KAME IPsec implementation.

### Examples

testcase SA\_example () runs on test\_CT  
{  
 var TCCIPsec\_Result res;  
  
 res := f\_IPsec\_SADB\_flush ();  
 if ( res != ok ) { setverdict(fail); }  
 res := f\_IPsec\_SADB\_add ( "192.168.1.1", "192.168.1.2",  
 esp, 11001, { { hardLifetime := 180 }, { softLifetime := 60 } },  
 { encrAndAuth := {  
 ealgo := EALG\_3DESCBC, ekey := { text := "123456789012345678901234" },  
 aalgo := AALG\_MD5HMAC, akey := { text := "1234567890123456" } } } );  
 if ( res != ok ) { setverdict(fail); }  
  
 res := f\_IPsec\_SADB\_delete ( "192.168.1.1", "192.168.1.2", esp, 11001 );  
 log ( "f\_IPsec\_SADB\_delete returns: ", res );  
  
 select ( res ) {  
 case ( ok ) { setverdict(pass); }  
 case else { setverdict(fail); }  
 }  
}

testcase SP\_delete () runs on test\_CT  
{  
 var TCCIPsec\_Result res;  
  
 res := f\_IPsec\_SPDB\_flush ();  
 if ( res != ok ) { setverdict(fail); }  
  
 res := f\_IPsec\_SPDB\_add ( "192.168.1.1", -, 2001, "192.168.1.2", -, 3001,  
 tcpProto, outDir,  
 { ipSec := { { protocol := ah, mode := { transport := {} }, level := { unique := { id := 101 } } } } } );  
 if ( res != ok ) { setverdict(fail); }  
  
 res := f\_IPsec\_SPDB\_delete ( "192.168.1.1", -, 2001, "192.168.1.2", -, 3001, tcpProto, outDir );  
 log ( "f\_IPsec\_SPDB\_delete returns: ", res );  
  
 select ( res ) {  
 case ( ok ) { setverdict(pass); }  
 case else { setverdict(fail); }  
 }  
}

## XPath Support Functions

### Overview

XPath Support Functions enable you to get any information stored in an XML document by making XPath queries. XPath is a language for finding information in an XML document. XPath is used to navigate through elements and attributes in an XML document. You can find reference and any further information about XPath 1.0 (base of the implementation) on the page:

<http://www.w3.org/TR/xpath>

XPath Support Functions are implemented in the following files:

* TCCXPathSupport\_Functions.ttcn
* TCCXPathSupport.cc

### Function definitions

The following function was defined to enable you to use XPath queries in order to access the required data stored in the XML document.

**external function XPathQuery (**

**in universal charstring xml\_doc,**

**in universal charstring xpath\_query )**

**return universal charstring;**

Parameters:

*xml\_doc:*

the XML document to be processed. It is modeled by a pure TTCN-3 universal charstring and passed as an input parameter to this function

*xpath\_query:*

the XPath Query. With this technology we can access a node of the XML tree. It is passed to the function as a universal charstring

Return Value:

universal charstring - result of the query (information about a node of the XML tree)

Function to check the validity of the XML document

**external function XPathCheckXML (**

**in universal charstring xml\_doc,**

**out charstring error\_string)**

**return boolean;**

Parameters:

*xml\_doc:*

the XML document to be processed. It is modeled by a pure TTCN-3 universal charstring and passed as an input parameter to this function

*error\_string:*

The function returns the description of the error in that string

Return Value:

boolean – true if the document is valid, false otherwise.

### Building the TinyXPath external library

This TTCN-3 function is an application of the open-source TinyXPath external library.

TinyXPath is an XPath syntax decoder written in C++ built on top of the TinyXML package. TinyXML forms a tree form the XML document and TinyXPath uses this tree to apply XPath query on it.

So with TinyXPath you can apply an XPath syntax decoding to a TinyXML tree, and you can extract the required information from the XML document.

TinyXPath is covered by the zlib license.

In order to use the TinyXPath library, first you need to build the library, which you can import to your project.

You need to follow these instructions below.

First you have to get the whole TinyXPath folder and save it to your computer. You can find this folder in:

/vobs/ttcn/TCC\_Releases/Other/TinyXPath

You must remember the exact location of the directory (where you saved it).

Within the folder you can find subfolders containing:

source files: /source

header files: /include

library file: /lib

In order to generate this library file from source files we provide a Makefile. You can find it outside the subfolders, in the TinyXPath directory. You can use two commands - *make, make clean*.

First command is to build the library. The working directory must be:

<where you saved the dir>/TinyXPath

And then you type only: *make.*

You can find the library file (libtinyxpath.a) in the TinyXPath/lib subfolder.

### Preparing your TTCN-3 environment for using TinyXPath

#### Modification of the TTCN-3 files in your project

You need to add two files

TCCXPathSupport.cc

TCCXPathSupport\_Functions.ttcn

to your project. These contain the XPathSupport function declaration and definition.

In order to use the implemented function in your module you have to import definitions from the module containing the definition of XPathSupport function. You can use line below in the beginning of the module:

import from TCCXPathSupport\_Functions all;

#### Modification of the Makefile of your project

Finally, your generated Makefile needs some changes to be able to find the external library.

1.) Before every command in the Makefile you have to insert the next row specifying the place where you saved the TinyXPath folder:

TINYXPATH\_DIR = <path to the dir>

For example, if you saved this folder straight to the project folder, you can write:

TINYXPATH\_DIR = ./TinyXPath

2.) Next you have to add text to the end of the line starting with CPPFLAGS:

-I$(TINYXPATH\_DIR)/include

3.) Finally, at the section of $(TARGET): $(OBJECTS) you will find this two lines:

-L$(TTCN3\_DIR)/lib -l$(TTCN3\_LIB) \

-L$(OPENSSL\_DIR)/lib-lcrypto $($(PLATFORM)\_LIBS)

You have to insert a line between them according to this:

-L$(TTCN3\_DIR)/lib -l$(TTCN3\_LIB) \

-L$(TINYXPATH\_DIR)/lib -ltinyxpath \

-L$(OPENSSL\_DIR)/lib-lcrypto $($(PLATFORM)\_LIBS)

Now with this modified Makefile your project will be able to find and use the TinyXPath library.

### Error messages

None

### Warning messages

None

### Examples

module TCCXPathSupport\_Example {

import from TCCXPathSupport\_Functions all;

control {

var universal charstring doc := "<a><b val='123'><b />

<c/><!--- 122.0 --><d /></b><!-- 500.0 -->

<x target='xyz'>sub text</x></a>";

log ("The XML document:");

log (doc);

var universal charstring q := "sum(//\*/comment())";

log ("The XPath query:");

log (q);

var universal charstring res := XPathQuery (doc, q);

log ("The result of the query:");

log (res);

}

}

## System functions

### Overview

The system functions implement various operating system and process related functions

Conversion Functions are implemented in the following files:

* TCCSystem\_Functions.ttcn
* TCCSystem.cc

## Function definitions

**external function f\_SYS\_getpid() return integer;**

Returns the pid of the process.

### Error messages

None

### Warning messages

None

### Examples

None

## FileSystem functions

### Overview

The FileSystem functions get information about the file system that contains the file named by the filename argument

FileSystem Functions are implemented in the following files:

* TCCFileSystem\_Functions.ttcn
* TCCFileSystem.cc

## Function definitions

**external function f\_FS\_bsize(in charstring p\_name) return integer**

Returns the file system block size or -1 on error

**external function f\_FS\_block(in charstring p\_name) return integer**

Returns the total number of blocks in the file system or -1 on error

**external function f\_FS\_bfree(in charstring p\_name) return integer**

Returns the total number of free blocks in the file system or -1 on error

**external function f\_FS\_dspacerate(in charstring p\_name) return integer**

Returns the free space rate in the file system or -1 on error

### Error messages

None

### Warning messages

None

### Examples

testcase f\_test\_FS\_bsize() runs on FileSystem\_CT

{

log(" ----------- f\_FS\_bsize ----------- ");

log("Block size (f\_FS\_bsize): ", f\_FS\_bsize("./"));

// no check

setverdict(pass);

}

testcase f\_test\_FS\_block() runs on FileSystem\_CT

{

log(" ----------- f\_FS\_block ----------- ");

log("Number of blocks (f\_FS\_block): ", f\_FS\_block("./"));

// no check

setverdict(pass);

}

testcase f\_test\_FS\_bfree() runs on FileSystem\_CT

{

log(" ----------- f\_FS\_bfree ----------- ");

log("Number of free blocks (f\_FS\_bfree): ", f\_FS\_bfree("./"));

// no check

setverdict(pass);

}

testcase f\_test\_FS\_dspacerate() runs on FileSystem\_CT

{

log(" ----------- f\_FS\_dspacerate ----------- ");

log("The free space rate (f\_FS\_dspacerate): ", f\_FS\_dspacerate("./"));

// no check

setverdict(pass);

}

## Regexp functions

### Overview

The system functions implement non-TTCN-3 regular expression support functions

Regexp Functions are implemented in the following files:

* TCCRegexp\_Functions.ttcn
* TCCRegexp.cc

### Function definitions

**external function f\_pcre\_regexp(in charstring instr, in charstring expression, in integer groupno) return charstring;**

This function returns the substring of the input character string instr, which is the content of n-th group matching to the expression. The expression is a Perl compatible regular expression pattern. The number of the group to be returned is specified by groupno, which shall be a positive integer or zero. Group numbers are assigned by the order of occurrences of the opening bracket of a group and counted starting from 1 by step 1. If the groupno is zero, a substring matched by the entire pattern is returned. If no substring fulfilling all conditions (i.e. pattern and group number) is found within the input string, an empty string is returned.

**external function f\_pcre\_regexp\_list(in charstring instr,in charstring expression) return charstring\_list;**

This function returns a list of substrings of the input character string instr, which are the content of n-th group matching to the expression. The expression is a Perl compatible regular expression pattern. Group numbers are assigned by the order of occurrences of the opening bracket of a group and counted starting from 1 by step 1.

**external function f\_pcre\_regexp\_list\_all\_matches(in charstring instr,in charstring expression) return t\_list\_of\_charstring\_list;**

This function returns the full list of the substring of the input character string instr, which is the content of n-th group matching to the expression. The expression is a Perl compatible regular expression pattern. Group numbers are assigned by the order of occurrences of the opening bracket of a group and counted starting from 1 by step 1. If no substring fulfilling all conditions is found within the input string, an empty record of is returned.

The functions are based on the PCRE library. See: www.pcre.org

### Error messages

Compilation of the pcre regexp failled at position x. Reason: yyyyy"

The supplied expression is not a valid PCRE expression.

### Warning messages

None

### Examples

f\_pcre\_regexp(“abc”, “(a|(z))(bc)” ,0)

returns “abc”

f\_pcre\_regexp(“abc”, “(a|(z))(bc)” ,1)

returns “a”

f\_pcre\_regexp(“abc”, “(a|(z))(bc)” ,2)

returns “”. The second sub expression is the “(z)”

f\_pcre\_regexp(“abc”, “(a|(z))(bc)” ,3)

returns “bc”

f\_pcre\_regexp\_list(“abc”, “(a|(z))(bc)”)

returns {“abc” , ”a” , ”” , ”bc”}

f\_pcre\_regexp\_list\_all\_matches("a1 b1 a2", "(a)(\\d)")

returns { { "a1", "a", "1" }, { "a2", "a", "2" } }

# Terminology

No specific terminology used.

## Abbreviations

ES ETSI Standard

ETSI European Telecommunications Standards Institute

RFC Request for Comments

TTCN-3 Testing and Test Control Notation version 3

# References

1. ETSI ES 201 873-1 v3.1.1 (06/2005)  
   The Testing and Test Control Nota­tion version 3. Part 1: Core Language
2. 1/1531-CRL 113 200/6 Uen  
   Installation Guide for TITAN TTCN-3 Test Executor
3. 1/198 17-CRL 113 200/6 Uen  
   User Guide for TITAN TTCN-3 Test Executor
4. RFC 2045  
   Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies
5. RFC 2849  
   The LDAP Data Interchange Format (LDIF) – Technical Specification
6. NaturalDocs document for TCC Useful Functions  
   (/vobs/ttcn/TCC\_Common/Libraries/TCCUsefulFunctions\_  
   CNL113472/doc/apidoc)
7. An example algorithm set for the 3GPP authentication and key generation functions f1, f1\*, f2, f3, f4, f5 and f5\*;   
   Document 2: Algorithm Specification  
   3GPP TS 35.206 (v7.0.0 2007-06)
8. 3GPP TS 33.102 V9.1.0 (2009-12)  
   3rd Generation Partnership Project; Technical Specification Group  
   Services and System Aspects; 3G Security; Security architecture (Release 9)
9. 3GPP TS 34.108 V4.7.0  
   Common test environments for User Equipment (UE) conformance testing (Release 4) 2003.06.
10. Specification of the 3GPP Confidentiality and Integrity Algorithms  
    128-EEA3 & 128-EIA3. Document 1: 128-EEA3 and 128-EIA3  
    Specification, Version 1.6, 1st July, 2011
11. Specification of the 3GPP Confidentiality and Integrity Algorithms  
    128-EEA3 & 128-EIA3. Document 2: ZUC Specification  
    Version 1.6, 28th June 2011
12. GSM 03.38 V7.2.0  
    Digital cellular telecommunications system (Phase 2+)  
    Alphabets and language-specific information
13. ETR 060  
    Guidelines for using Abstract Syntax Notation One (ASN.1) in  
    telecommunication application protocols
14. ITU-T Q.763 (12/1999)  
    SERIES Q: SWITCHING AND SIGNALLING  
    Specifications of Signalling System No. 7 – ISDN user part  
    Signalling System No. 7 – ISDN user part formats and codes
15. 3GPP TS 24.008 V12.0.0 (2012-12)  
    3rd Generation Partnership Project;  
    Technical Specification Group Core Network and Terminals;  
    Mobile radio interface Layer 3 specification;  
    Core network protocols; Stage 3  
    (Release 12)
16. 3GPP TS 33.401 V12.14.0 (2015-03)  
    3rd Generation Partnership Project;  
    Technical Specification Group Services and System Aspects;  
    3GPP System Architecture Evolution (SAE);  
    Security architecture  
    (Release 12)