

International Telecommunication Union

**ITU-T**

TELECOMMUNICATION  
STANDARDIZATION SECTOR  
OF ITU

**Z.166**

(05/2012)

SERIES Z: LANGUAGES AND GENERAL SOFTWARE  
ASPECTS FOR TELECOMMUNICATION SYSTEMS

Formal description techniques (FDT) – Testing and Test  
Control Notation (TTCN)

---

**Testing and Test Control Notation version 3:  
TTCN-3 control interface (TCI)**

Recommendation ITU-T Z.166



ITU-T Z-SERIES RECOMMENDATIONS

LANGUAGES AND GENERAL SOFTWARE ASPECTS FOR TELECOMMUNICATION SYSTEMS

FORMAL DESCRIPTION TECHNIQUES (FDT)	
Specification and Description Language (SDL)	Z.100–Z.109
Application of formal description techniques	Z.110–Z.119
Message Sequence Chart (MSC)	Z.120–Z.129
User Requirements Notation (URN)	Z.150–Z.159
<b>Testing and Test Control Notation (TTCN)</b>	<b>Z.160–Z.179</b>
PROGRAMMING LANGUAGES	
CHILL: The ITU-T high level language	Z.200–Z.209
MAN-MACHINE LANGUAGE	
General principles	Z.300–Z.309
Basic syntax and dialogue procedures	Z.310–Z.319
Extended MML for visual display terminals	Z.320–Z.329
Specification of the man-machine interface	Z.330–Z.349
Data-oriented human-machine interfaces	Z.350–Z.359
Human-machine interfaces for the management of telecommunications networks	Z.360–Z.379
QUALITY	
Quality of telecommunication software	Z.400–Z.409
Quality aspects of protocol-related Recommendations	Z.450–Z.459
METHODS	
Methods for validation and testing	Z.500–Z.519
MIDDLEWARE	
Processing environment architectures	Z.600–Z.609

*For further details, please refer to the list of ITU-T Recommendations.*

## Recommendation ITU-T Z.166

### Testing and Test Control Notation version 3: TTCN-3 control interface (TCI)

#### Summary

Recommendation ITU-T Z.166 specifies the control interfaces for TTCN-3 test system implementations. The TTCN-3 control interfaces provide a standardized adaptation for management, test component handling and encoding/decoding of a test system to a particular test platform. This Recommendation defines the interfaces as a set of operations independent of a target language.

The interfaces are defined to be compatible with the TTCN-3 standards (see clause 2). The interface definition uses the CORBA interface definition language (IDL) to specify the TCI completely. Clauses 8 and 9 present language mappings for this abstract specification to the target languages Java and ANSI C. A summary of the IDL-based interface specification is provided in Annex A.

The first revision of the Recommendation contains amendments (i.e., C++ and C# language mappings), numerous clarifications, corrigenda and editorial corrections.

The first revision of the Recommendation corrects an error in the code specification of Recommendation ITU-T Z.166 (2011) TCI-CH provided interface in clause 8.5.3.1 in such a way that `tciTestComponentRunningReq`, `tciTestComponentDoneReq`, `tciTestComponentAliveReq`, and `tciTestComponentKilledReq` all return `public Boolean` as result.

This second revision of the Recommendation contains amendments, clarifications, corrigenda and editorial corrections.

This Recommendation is technically aligned with ETSI ES 201 873-6 V4.4.1 (2012).

#### History

Edition	Recommendation	Approval	Study Group
1.0	ITU-T Z.145	2006-03-16	17
2.0	ITU-T Z.166	2007-11-13	17
3.0	ITU-T Z.166	2011-03-16	17
4.0	ITU-T Z.166	2012-05-29	17

## FOREWORD

The International Telecommunication Union (ITU) is the United Nations specialized agency in the field of telecommunications, information and communication technologies (ICTs). The ITU Telecommunication Standardization Sector (ITU-T) is a permanent organ of ITU. ITU-T is responsible for studying technical, operating and tariff questions and issuing Recommendations on them with a view to standardizing telecommunications on a worldwide basis.

The World Telecommunication Standardization Assembly (WTSA), which meets every four years, establishes the topics for study by the ITU-T study groups which, in turn, produce Recommendations on these topics.

The approval of ITU-T Recommendations is covered by the procedure laid down in WTSA Resolution 1.

In some areas of information technology which fall within ITU-T's purview, the necessary standards are prepared on a collaborative basis with ISO and IEC.

## NOTE

In this Recommendation, the expression "Administration" is used for conciseness to indicate both a telecommunication administration and a recognized operating agency.

Compliance with this Recommendation is voluntary. However, the Recommendation may contain certain mandatory provisions (to ensure, e.g., interoperability or applicability) and compliance with the Recommendation is achieved when all of these mandatory provisions are met. The words "shall" or some other obligatory language such as "must" and the negative equivalents are used to express requirements. The use of such words does not suggest that compliance with the Recommendation is required of any party.

## INTELLECTUAL PROPERTY RIGHTS

ITU draws attention to the possibility that the practice or implementation of this Recommendation may involve the use of a claimed Intellectual Property Right. ITU takes no position concerning the evidence, validity or applicability of claimed Intellectual Property Rights, whether asserted by ITU members or others outside of the Recommendation development process.

As of the date of approval of this Recommendation, ITU had not received notice of intellectual property, protected by patents, which may be required to implement this Recommendation. However, implementers are cautioned that this may not represent the latest information and are therefore strongly urged to consult the TSB patent database at <http://www.itu.int/ITU-T/ipr/>.

© ITU 2012

All rights reserved. No part of this publication may be reproduced, by any means whatsoever, without the prior written permission of ITU.

## Table of Contents

	<b>Page</b>
1	Scope ..... 1
2	References..... 1
2.1	Normative references..... 1
2.2	Informative references..... 2
3	Definitions and abbreviations ..... 2
3.1	Definitions ..... 2
3.2	Abbreviations and acronyms ..... 4
4	Introduction ..... 4
5	Compliance ..... 5
6	General structure of a TTCN-3 test system ..... 5
6.1	Entities in a TTCN-3 test system..... 5
6.2	Execution requirements for a TTCN-3 test system ..... 9
7	TTCN-3 control interface and operations..... 9
7.1	Overview of the TCI..... 9
7.2	TCI data ..... 13
7.3	TCI operations ..... 24
8	Java language mapping..... 113
8.1	Introduction ..... 113
8.2	Names and scopes..... 113
8.3	Type mapping..... 113
8.4	Constants ..... 128
8.5	Mapping of interfaces..... 130
8.6	Optional parameters..... 137
8.7	TCI initialization ..... 137
8.8	Error handling..... 137
9	ANSI C language mapping..... 138
9.1	Introduction ..... 138
9.2	Value interfaces ..... 138
9.3	Logging interface..... 142
9.4	Operation interfaces..... 144
9.5	Data..... 150
9.6	Miscellaneous ..... 152
9.7	Optional parameters..... 152
10	C++ language mapping..... 153
10.1	Introduction ..... 153
10.2	Names and scopes..... 153
10.3	Memory management..... 153

	<b>Page</b>
10.4	Error handling..... 153
10.5	Type mapping..... 153
10.6	Operations mapping..... 177
11	W3C XML mapping..... 188
11.1	Introduction ..... 188
11.2	Scopes..... 188
11.3	Type mapping..... 189
11.4	Mapping of the operations on the logging interface..... 210
12	C# mapping..... 229
12.1	Introduction ..... 229
12.2	Names and scopes..... 229
12.3	Null value mapping ..... 230
12.4	Type mapping..... 230
12.5	Mapping of interfaces..... 245
12.6	Optional parameters..... 253
12.7	Error handling..... 253
Annex A – IDL specification of TCI ..... 255	
Annex B – XML mapping for TCI TL provided ..... 271	
B.1	TCI-TL XML schema for simple types..... 271
B.2	TCI-TL XML schema for types ..... 272
B.3	TCI-TL XML schema for values..... 274
B.4	TCI-TL XML schema for templates..... 279
B.5	TCI-TL XML schema for Events ..... 286
B.6	TCI-TL XML schema for a log..... 305
Appendix I – Use scenarios ..... 308	
I.1	Initialization, collecting information, logging..... 308
I.2	Execution of test cases and control ..... 311
I.3	Component handling ..... 314
I.4	Termination of test cases and control..... 321
I.5	Communication ..... 326
Bibliography..... 331	

# Recommendation ITU-T Z.166

## Testing and Test Control Notation version 3: TTCN-3 control interface (TCI)

### 1 Scope

This Recommendation specifies the control interfaces for TTCN-3 test system implementations. The TTCN-3 control interfaces provide a standardized adaptation for management, test component handling and encoding/decoding of a test system to a particular test platform. The present document defines the interfaces as a set of operations independent of a target language.

The interfaces are defined to be compatible with the TTCN-3 standard (see clause 22). The interface definition uses the CORBA interface definition language (IDL) to specify the TCI completely. Clauses 8, 9, and 9.7 present language mappings for this abstract specification to the target languages Java, ANSI C, and C++. A summary of the IDL-based interface specification is provided in Annex A.

### 2 References

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the reference document (including any amendments) applies.

Referenced documents which are not found to be publicly available in the expected location might be found at <http://docbox.etsi.org/Reference>.

NOTE – While any hyperlinks included in this clause were valid at the time of publication ETSI cannot guarantee their long-term validity.

#### 2.1 Normative references

The following ITU-T Recommendations and other references contain provisions which, through reference in this text, constitute provisions of this Recommendation. At the time of publication, the editions indicated were valid. All Recommendations and other references are subject to revision; users of this Recommendation are therefore encouraged to investigate the possibility of applying the most recent edition of the Recommendations and other references listed below. A list of the currently valid ITU-T Recommendations is regularly published. The reference to a document within this Recommendation does not give it, as a stand-alone document, the status of a Recommendation.

- [1] Recommendation ITU-T Z.161 (2012), *Testing and Test Control Notation version 3: TTCN-3 core language*.

ETSI ES 201 873-1 (2012), *Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language*.  
<[http://webapp.etsi.org/workprogram/Report\\_WorkItem.asp?WKI\\_ID=35092](http://webapp.etsi.org/workprogram/Report_WorkItem.asp?WKI_ID=35092)>

- [2] Recommendation ITU-T Z.164 (2012), *Testing and Test Control Notation version 3: TTCN-3 operational semantics*.

ETSI ES 201 873-4 (2012), *Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 4: TTCN-3 Operational Semantics*.  
<[http://webapp.etsi.org/workprogram/Report\\_WorkItem.asp?WKI\\_ID=35095](http://webapp.etsi.org/workprogram/Report_WorkItem.asp?WKI_ID=35095)>

- [3] Recommendation ITU-T Z.165 (2012), *Testing and Test Control Notation version 3: TTCN-3 runtime interface (TRI)*.

ETSI ES 201 873-5 (2012), *Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 5: TTCN-3 Runtime Interface (TRI)*.  
<[http://webapp.etsi.org/workprogram/Report\\_WorkItem.asp?WKI\\_ID=35096](http://webapp.etsi.org/workprogram/Report_WorkItem.asp?WKI_ID=35096)>

- [4] Recommendation ITU-T X.290 (1995), *OSI conformance testing methodology and framework for protocol Recommendations for ITU-T applications – General concepts*.  
ISO/IEC 9646-1:1994, *Information technology – Open Systems Interconnection – Conformance testing methodology and framework – Part 1: General concepts*.  
<[http://webstore.iec.ch/webstore/webstore.nsf/ArtNum\\_PK/39613?OpenDocument](http://webstore.iec.ch/webstore/webstore.nsf/ArtNum_PK/39613?OpenDocument)>
- [5] ISO/IEC 10646:2011, *Information technology – Universal Coded Character Set (UCS)*.  
<[http://www.iso.org/iso/iso\\_catalogue/catalogue\\_tc/catalogue\\_detail.htm?csnumber=51273](http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=51273)>
- [6] Object Management Group (OMG) (2002), *The Common Object Request Broker: Architecture and Specification, Version 3.0, formal/02-06-01*.  
<<http://www.omg.org/cgi-bin/doc?formal/02-06-01>>
- [7] Sun Microsystems (2011), *Java Language Specification*.  
<[http://java.sun.com/docs/books/jls/third\\_edition/html/j3TOC.html](http://java.sun.com/docs/books/jls/third_edition/html/j3TOC.html)>
- [8] ISO/IEC 9899:2011, *Information technology – Programming languages – C*.  
<[http://www.iso.org/iso/home/store/catalogue\\_tc/catalogue\\_detail.htm?csnumber=57853](http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=57853)>
- [9] ISO/IEC 14882:2011, *Information technology – Programming languages – C++*.  
<[http://www.iso.org/iso/catalogue\\_detail.htm?csnumber=50372](http://www.iso.org/iso/catalogue_detail.htm?csnumber=50372)>
- [10] W3C Recommendation (2004), *XML Schema Part 0: Primer*.  
<<http://www.w3.org/TR/xmlschema-0/>>
- [11] W3C Recommendation (2004), *XML Schema Part 1: Structures*.  
<<http://www.w3.org/TR/xmlschema-1/>>
- [12] W3C Recommendation (2004), *XML Schema Part 2: Datatypes*.  
<<http://www.w3.org/TR/xmlschema-2/>>
- [13] ECMA-334 (2006), *C# Language Specification*.  
<<http://www.ecma-international.org/publications/standards/Ecma-334.htm>>

## 2.2 Informative references

None.

## 3 Definitions and abbreviations

### 3.1 Definitions

For the purposes of this Recommendation, the terms and definitions given in Recommendation ITU-T X.290 [4] and the following apply:

**3.1.1 abstract test suite (ATS):** Test suite composed of abstract test cases, which are specified by TTCN-3 module(s).

**3.1.2 codec:** Encoder/decoder entity used for encoding and decoding data to be transmitted and received, respectively.

**3.1.3 coding/decoding (CD):** Entity that administers the value and type handling incl. encoding and decoding in the TTCN-3 test system.

**3.1.4 component handling (CH):** Entity that administers the handling of test components in the TTCN-3 test system.



**3.1.5 communication port:** Abstract mechanism facilitating communication between test components.

NOTE – A communication port is modelled as a FIFO queue in the receiving direction. Ports can be message-based, procedure-based or a mixture of the two.

**3.1.6 control component:** Component that executes the behaviour of the control part of a TTCN-3 module.

**3.1.7 executable test suite (ETS):** Refer to Recommendation ITU-T X.290 [4].

**3.1.8 implementation extra information for testing (IXIT):** Refer to Recommendation ITU-T X.290 [4].

**3.1.9 platform adaptor (PA):** Entity that adapts the TTCN-3 executable to a particular execution platform.

NOTE – The platform adaptor creates a single notion of time for a TTCN-3 test system, and implements both, explicit and implicit, timers as well as external functions.

**3.1.10 real test system interface:** Refer to Recommendation ITU-T X.290 [4].

**3.1.11 SUT adaptor (SA):** Entity that adapts the TTCN-3 communication operations with the SUT based on an abstract test system interface.

NOTE – It implements the real test system interface.

**3.1.12 system under test (SUT):** Refer to Recommendation ITU-T X.290 [4].

**3.1.13 Testing and Test Control Notation version 3 (TTCN-3):** Refer to Recommendation ITU-T X.290 [4].

**3.1.14 test case:** Refer to Recommendation ITU-T X.290 [4].

**3.1.15 test event:** Either sent or received test data (message or procedure call) on a communication port that is part of the test system interface as well as timeout events of timers.

**3.1.16 test logging (TL):** Entity which provides logging information about test execution (including also the information provided by the TTCN-3 log statement).

**3.1.17 test management (TM):** Entity which provides a user interface to as well as the administration of the TTCN-3 test system.

**3.1.18 test management and control (TMC):** Set of entities providing test management and control; consists of the test management (TM), the component handling (CH), the test logging (TL) and the coding/decoding (CD).

NOTE – The TMC is an implementation of TCI.

**3.1.19 test system:** Refer to Recommendation ITU-T X.290 [4].

**3.1.20 test system interface (TSI):** Test component that provides a mapping of the ports available in the (abstract) TTCN-3 test system to those offered by a real test system.

**3.1.21 TTCN-3 executable (TE):** Part of a test system that deals with interpretation or execution of a TTCN-3 ETS.

**3.1.22 TTCN-3 control interfaces (TCI):** Four interfaces that define the interaction of the TTCN-3 executable with the test management, the coding and decoding, the test component handling and the logging in a test system.

**3.1.23 TTCN-3 runtime interface (TRI):** Two interfaces that define the interaction of the TTCN-3 executable between the SUT and the platform adaptor (PA) and the system adaptor (SA) in a test system.

### 3.2 Abbreviations and acronyms

For the purposes of this Recommendation, the following abbreviations and acronyms apply:

ATS	Abstract Test Suite
CD	(External) Coding/Decoding
CH	Component Handler
CORBA	Common Object Request Broker Architecture
ETS	Executable Test Suite
FIFO	First In First Out
IDL	Interface Definition Language
IXIT	Implementation eXtra Information for Testing
MTC	Main Test Component
OMG	Object Management Group
PA	Platform Adaptor
PTC	Parallel Test Component
SA	SUT Adaptor
SUT	System Under Test
TCI	TTCN-3 Control Interfaces
TE	TTCN-3 Executable
TL	Test Logging
TM	Test Management
TMC	Test Management and Control
TRI	TTCN-3 Runtime Interface
TSI	Test System Interface
TTCN-3	Testing and Test Control Notation version 3
UML	Unified Modelling Language
W3C	World Wide Web Consortium
XML	Extensible Markup Language

## 4 Introduction

This Recommendation consists of two distinct parts, the first part describes the structure of a TTCN-3 test system implementation and the second part presents the TTCN-3 control interfaces specification.

The first part introduces the decomposition of a TTCN-3 test system into four main entities:

- Test management and control (TMC).
- TTCN-3 executable (TE).
- SUT adaptor (SA).
- Platform adaptor (PA).

The TMC consists itself of three entities: test management (TM), coder/decoder (CD), and test component handler (CH). In addition, the interaction between these entities, i.e., the corresponding interfaces, is defined.

The second part of the present document specifies the TTCN-3 control interfaces (TCI). The interfaces are defined in terms of operations implemented as part of one entity and called by other test system entities. For each operation, the interface specification defines associated data structures, the intended effect on the test system and any constraints on the usage of the operation. Note that these interface specifications only define interactions between the TE and TM, TE and CD, and TE and CH. For interactions between the TE and SA and the TE and PA please refer to the TTCN-3 runtime interface specification ([3]).

## **5 Compliance**

The minimum required for a TCI compliant TTCN-3 test system is to adhere to the interface specification stated in the present document. The TTCN-3 semantics in the test system must adhere to the operational semantics defined in [2]. In addition, one language mapping must be supported. For example, if a vendor supports Java, the TCI operation calls and implementations, which are part of the TTCN-3 executable, must comply with the IDL to Java mapping specified in the present document. For the logging interface, the XML mapping can be used instead of the Java or the C mapping.

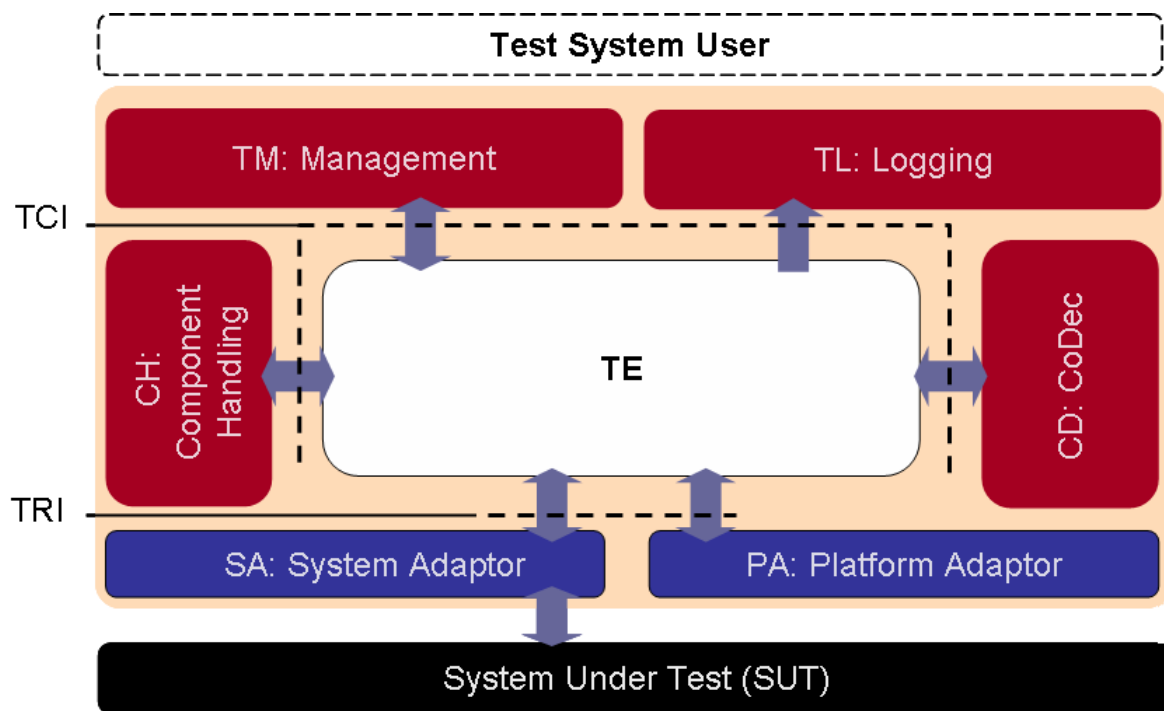
## **6 General structure of a TTCN-3 test system**

A TTCN-3 test system can be thought of conceptually as a set of interacting entities. Each entity implements specific test system functionality. These entities:

- manage test execution;
- interpret or execute compiled TTCN-3 code;
- realize proper communication with the SUT;
- administer types, values and test components;
- implement external functions; and
- handle timer operations.

### **6.1 Entities in a TTCN-3 test system**

The structure of a TTCN-3 test system implementation is illustrated in Figure 1.



**Figure 1 – General structure of a TTCN-3 test system**

As shown in Figure 1, the TTCN-3 executable (TE), also referred to as the executable test suite (ETS), interprets and executes TTCN-3 modules. Various TE structural elements can be identified: control, behaviour, components, types, values and queues. The structural elements within the TE represent functionality that is defined within a TTCN-3 module or by the TTCN-3 standard ([1]) itself. For example, the structural element "Control" represents the control part within a TTCN-3 module, while the structural element "Queues" represents the requirement on a TTCN-3 executable that each port of a test component maintains its own port queue. While the first is specified within a TTCN-3 module, the latter is required by the TTCN-3 specification.

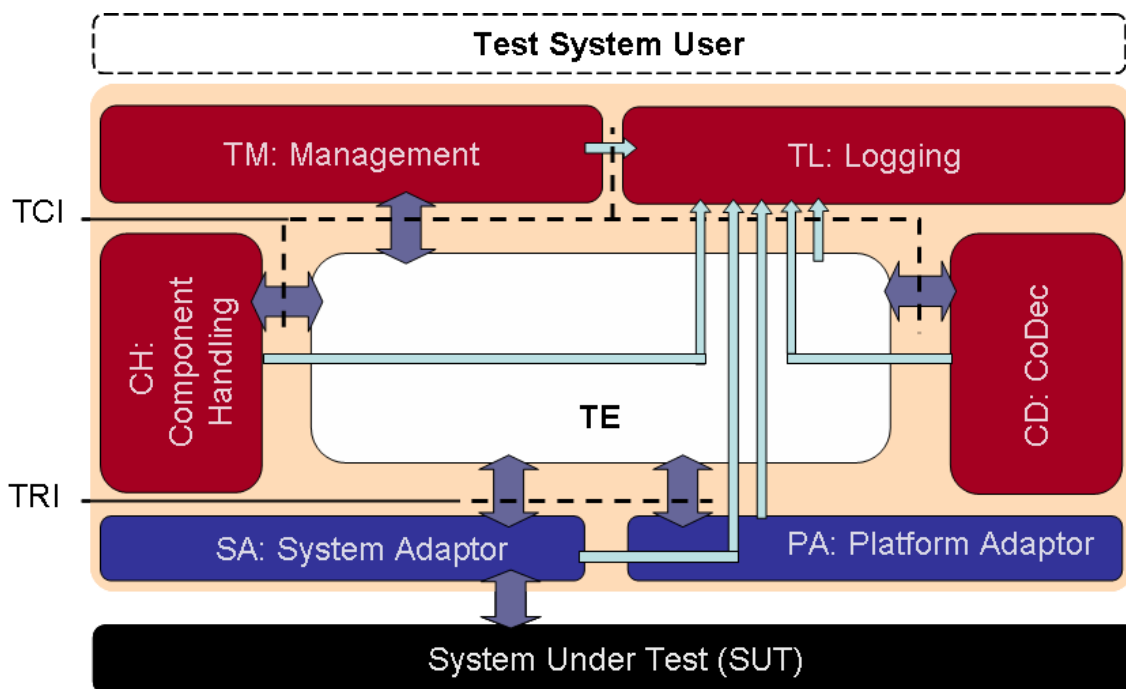
Refinement of the TE, as shown in Figure 1, is provided as an aid in defining the TTCN-3 control interfaces. The TE would typically correspond in a test system implementation either to the executable code produced by a TTCN-3 compiler or by a TTCN-3 interpreter.

The TE may be executed in a centralized or in a distributed manner. That is, on a single test device or across several test devices respectively. Although the structural entities of the TE implement a complete TTCN-3 module, single structural entities might be distributed over several test devices.

The TE implements a TTCN-3 module on an abstract level. The other entities of a TTCN-3 test system make these abstract concepts concrete. For example, the abstract concept of sending a message or receiving a timeout cannot be implemented within the TE. The remaining part of the test system implements the encoding of the message and its sending over concrete physical means or measuring the time and determining when a timer has expired, respectively.

The SA and PA and their interaction with the TE are defined in [3]. The TCI specification defines the interaction between the TE and the TMC.

The logging interface provides logging capabilities to all elements of the test system architecture, i.e., the TE, the TM, the CH, the CD, the SA and the PA are able to log information on the test execution via TL. Figure 2 represents a more detailed view on TL.



**Figure 2 – Detailed view on TL**

### 6.1.1 Test management and control (TMC)

The TMC entity includes functionality related to management of:

- test execution;
- components;
- encoding and decoding; and
- logging.

#### 6.1.1.1 Test management (TM)

The TM entity is responsible for the overall management of a test system. After the test system has been initialized, test execution starts within the TM entity. The entity is responsible for the proper invocation of TTCN-3 modules, i.e., propagating module parameters such as IXIT information to the TE if necessary. Typically, this entity would also implement a test system user interface.

#### 6.1.1.2 Coding and decoding (CD)

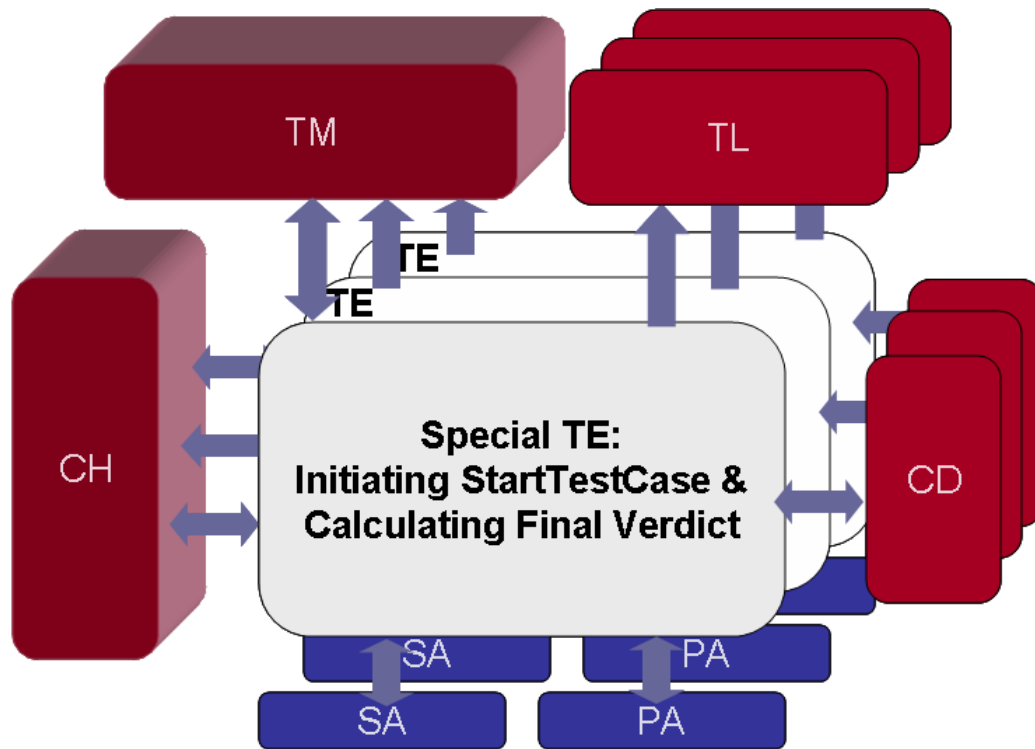
The CD entity is responsible for the external encoding and decoding of TTCN-3 values into bitstrings suitable to be sent to the system under test (SUT). Whenever external codecs are used, the TE determines which codecs shall be used. It passes the TTCN-3 data to the appropriate encoder to obtain the encoded data. Received data is decoded in the CD entity by using the appropriate decoder, which translates the received data into TTCN-3 values.

#### 6.1.1.3 Component handling (CH)

The TE can be distributed among several test devices. The CH implements communication between distributed test system entities. The CH entity provides the means to synchronize test system entities which might be distributed onto several nodes.

NOTE 1 – Nodes and test devices are used as synonyms.

The general structure of a test system distributed among several nodes is depicted in Figure 3.



**Figure 3 – General structure of a distributed TTCN-3 test system**

Each node within a test system includes the TE, SA, PA, CD and TL entities. The entities CH and TM mediate the test management and test component handling between the TEs on each node. The TE which starts a test case is a special TE. It shall calculate the final test case verdict. Besides this, all TEs are handled the same.

NOTE 2 – As stated in [2], a test system executes at most one test case at a given point in time, i.e., a TTCN-3 module cannot execute multiple test cases at the same time.

The creation of the MTC, PTCs and the control component in TEs is controlled by CH. Please note the special role of the system component, which exists only conceptually and not as a running test component in a TE. System ports, i.e., the ports of the system component, may be distributed among several nodes. Further, test components on different nodes may have access to the same physical port of the SUT, i.e., they may be mapped to the same port of the test system interface.

EXAMPLE – Access to remote real SUT ports can be realized by TEs via local proxies.

Communication between TTCN-3 components is either message or procedure based. Therefore, the CH adapts message and procedure based communication of TTCN-3 components to the particular execution platform of the test system. It is aware of connections between TTCN-3 test component communication ports. It propagates send request operations from one TTCN-3 component to another TTCN-3 component. The receiving component may reside in a different instance of the same TE located on a different node. It then notifies the TE of any received test events by enqueueing them in the port queues of the TE.

Procedure based communication operations between TTCN-3 components are also visible at the CH. The CH shall distinguish between the different kinds of procedure-based communication, i.e., call, reply, and exception, and shall propagate them in the appropriate manner to the TE where the target component resides. TTCN-3 procedure based communication semantics, i.e., the effect of such operation on TTCN-3 test component execution, are to be handled in the TE.

Additional communication is needed to implement the distribution of test components onto several nodes. Component management communication includes the indication of the creation of test components, the starting of execution of a test component, verdict distribution, as well as

component termination indication. The CH does not implement the TTCN-3 component behaviour. Rather, it implements the communication between several components implemented by a TE.

#### **6.1.1.4 Test logging (TL)**

The TL entity performs test event logging and presentation to the test system user. It provides the logging of information about the test execution such as which test components have been created, started and terminated, which data is sent to the SUT, received from the SUT and matched to TTCN-3 templates, which timers have been started, stopped or timed out, etc.

#### **6.1.2 TTCN-3 executable (TE)**

The TE entity executes or interprets a TTCN-3 module. Conceptually, the TE can be decomposed into six interacting entities: a Control, Behaviour, Component, Type, Value and Queue entity. This structural decomposition of the TE is defined in [3]. The terminology for TE defined in [3] is used within the present document.

#### **6.1.3 SUT adaptor (SA)**

The SA is the implementation of the system under test adaptor (SA) as defined in [3]. The terminology for SA defined in [3] is used within the present document.

#### **6.1.4 Platform adaptor (PA)**

The PA is the implementation of the platform adaptor (PA) as defined in [3]. The terminology for PA defined in [3] is used within the present document.

### **6.2 Execution requirements for a TTCN-3 test system**

Each TCI operation call shall be treated as an atomic operation in the calling entity. The called entity, which implements a TCI operation, shall return control to the calling entity as soon as its intended effect has been accomplished or if the operation cannot be completed successfully. The called entity shall not block in the implementation of procedure-based communication.

As stated before, no assumption is made as to whether the TTCN-3 test system or individual entities are implemented in a single executable or process or whether they are distributed among different processes or even test devices.

A TCI implementation shall fulfil the above mentioned requirements.

## **7 TTCN-3 control interface and operations**

This clause defines a set of abstract data types used to represent data communicated between the TE and the TMC. In addition, it defines TCI operations in terms of their signatures, when they are to be used and what their effects on the TTCN-3 test system are.

This definition also includes a more detailed description of the input parameters required for each TCI operation call and its return value.

### **7.1 Overview of the TCI**

The TCI defines the interaction between the TTCN-3 executable (TE), component handling (CH), the test management (TM), the coding/decoding (CD), the test logging (TL) entities within a TTCN-3 test system. It provides means for the TE to:

- manage test execution;
- distribute execution of test components among different test devices;
- encode and decode test data; and
- logging of information about test execution.

The TCI consists of four sub-interfaces:

- **TCI test management interface (TCI-TM):** This interface includes all operations needed to manage test execution, provide module parameters and external constants and provide test event logging.
- **TCI component handling interface (TCI-CH):** This interface consists of operations needed to implement the management of, and communication between TTCN-3 test components in a centralized or distributed test system. It includes operations to create, start and stop test components, establish connection between TTCN-3 components, manage test components and their verdicts, and handle message and procedure based communication between TTCN-3 components.
- **TCI coding/decoding interface (TCI-CD):** This interface includes all operations needed to retrieve and access codecs, i.e., encoders or decoders, for encoding data to be sent, defined using the TTCN-3 encode attribute, and to decode received data.
- **TCI test logging interface (TCI-TL):** This interface includes all operations needed to retrieve information about test execution and to control the level of detail of this information.

All interfaces are bidirectional so that calling and called parts reside in the TE and in the TMC of the test system. The provided interfaces (those operations which an interface offers to the TE) and the required operations (those operation which an interface needs to use from the TE) are combined into the respective provided and required subinterface for each interface, i.e., TCI-TM Provided/TCI-TM Required, TCI-CH Provided/TCI-CH Required, TCI-CD Provided/TCI-CD Required, and TCI-TL Provided/TCI-TL Required.

### 7.1.1 Correlation between TTCN-3 and TCI operation invocations

For some TTCN-3 operation invocations, there is a direct correlation to a TCI operation invocation, which is shown in Table 1. Some of the TTCN-3 operations correlate to a pair of TCI operation request and TCI operation to implement the propagation of TTCN-3 operations through the test system. For the other TCI operation invocations there is an indirect correlation – they are needed to implement the TTCN-3 semantics of underlying concepts.

#### 7.1.1.1 TTCN 3 operations with TCI operation equivalent

The correlation shown for TTCN-3 communication operations (i.e., send, call, reply, and raise) only holds if these operations are invoked on a test component port connected to another test component port. The correlation for communication operations that are invoked on test component ports that are mapped to test system interface ports is defined in [3].

**Table 1 – Correlation between TTCN-3 communication operations and TCI operation invocations**

<b>TTCN-3 Operation Name</b>	<b>TCI Operation Name</b>	<b>TCI Interface Name</b>
send	tciSendConnected (see Note 1)	TCI-CH Provided
	tciSendConnectedBC (see Note 2)	
	tciSendConnectedMC (see Note 3)	
	tciEnqueueMsgConnected	TCI-CH Required
call	tciCallConnected (see Note 1)	TCI-CH Provided
	tciCallConnectedBC (see Note 2)	
	tciCallConnectedMC (see Note 3)	



**Table 1 – Correlation between TTCN-3 communication operations and TCI operation invocations**

<b>TTCN-3 Operation Name</b>	<b>TCI Operation Name</b>	<b>TCI Interface Name</b>
	tciEnqueueCallConnected	TCI-CH Required
reply	tciReplyConnected (see Note 1)	TCI-CH Provided
	tciReplyConnectedBC (see Note 2)	
	tciReplyConnectedMC (see Note 3)	
	tciEnqueueReplyConnected	TCI-CH Required
raise	tciRaiseConnected (see Note 1)	TCI-CH Provided
	tciRaiseConnectedBC (see Note 2)	
	tciRaiseConnectedMC (see Note 3)	
	tciEnqueueRaiseConnected	TCI-CH Required
log	tliLog	TCI-TL Provided
NOTE 1 – For unicast communication.		
NOTE 2 – For broadcast communication.		
NOTE 3 – For multicast communication.		

#### 7.1.1.2 TTCN 3 operations with TCI operation pair equivalent

The correlation for TTCN-3 test case, test component and port operations is shown below. The initiating TE issues a TCI request operation to the TCI-CH, which propagates the respective TCI operation on the TE(s) which has (have) to perform it.

**Table 2 – Correlation between TTCN-3 test case, test component and port operations and TCI operation invocations**

<b>TTCN-3 Operation Name</b>	<b>TCI Operation Name</b>	<b>TCI Interface Name</b>
create	tciCreateTestComponentReq	TCI-CH Provided
	tciCreateTestComponent	TCI-CH Required
start (a component)	tciStartTestComponentReq	TCI-CH Provided
	tciStartTestComponent	TCI-CH Required
stop (a component)	tciStopTestComponentReq	TCI-CH Provided
	tciStopTestComponent	TCI-CH Required
kill	tciKillTestComponentReq	TCI-CH Provided
	tciKillTestComponent	TCI-CH Required
connect	tciConnectReq	TCI-CH Provided
	tciConnect	TCI-CH Required
disconnect	tciDisconnectReq	TCI-CH Provided
	tciDisconnect	TCI-CH Required

**Table 2 – Correlation between TTCN-3 test case, test component and port operations and TCI operation invocations**

<b>TTCN-3 Operation Name</b>	<b>TCI Operation Name</b>	<b>TCI Interface Name</b>
map	tciMapReq (see Note 1)	TCI-CH Provided
	tciMapParamReq (see Note 2)	
	tciMap (see Note 1)	TCI-CH Required
	tciMapParam (see Note 2)	
unmap	tciUnmapReq (see Note 1)	TCI-CH Provided
	tciUnmapParamReq (see Note 2)	
	tciUnmap (see Note 1)	TCI-CH Required
	tciUnmapParam (see Note 2)	
running	tciTestComponentRunningReq	TCI-CH Provided
	tciTestComponentRunning	TCI-CH Required
alive	tciTestComponentAliveReq	TCI-CH Provided
	tciTestComponentAlive	TCI-CH Required
done	tciTestComponentDoneReq	TCI-CH Provided
	tciTestComponentDone	TCI-CH Required
killed	tciTestComponentKilledReq	TCI-CH Provided
	tciTestComponentKilled	TCI-CH Required
mtc	tciGetMTCReq	TCI-CH Provided
	tciGetMTC	TCI-CH Required
execute	tciTestCaseExecuteReq	TCI-CH Provided
	tciTestCaseExecute	TCI-CH Required
NOTE 1 – For statement without configuration parameter.		
NOTE 2 – For statement with configuration parameter.		

### 7.1.1.3 TTCN 3 operations without direct TCI operation equivalent

For some TTCN-3 operation invocations, there is no direct correlation to TCI operation invocations as the ones shown in Table 1. These TTCN-3 operation invocations are realized by a series of TCI operation invocations as described in this clause.

#### 7.1.1.3.1 Test case stop operation

When the testcase.stop operation is invoked from the TE, the following actions need to be taken by the TE:

- the overall verdict should be set to USER\_ERROR with the message given to the invocation of the testcase stop operation as the verdict reason by invoking tciSetVerdict();
- a reference to the mtc should be obtained by invoking triGetMtcReq() in the CH interface; and
- via TLI, testcase.stop shall be logged with tliTcTerminated() with verdict USER\_ERROR;
- the mtc should be stopped by invoking triStopTestComponentReq() in the CH with the obtained reference to the mtc.

## 7.2 TCI data

The TCI specification defines a set of abstract data types. These describe, at a very high level, which kind of data shall be passed from a calling to a called entity. The abstract data types are used to determine:

- how TTCN-3 data is passed from a TE to an encoder, to encode TTCN-3 value representations into a bitstring; and in the reverse case;
- how data passed from a decoder to the TE shall be decoded from a bitstring into its TTCN-3 value representation.

For these abstract data types a set of operations is defined to process the data by the coder/decoder.

The concrete representation of these abstract data types as well as the definition of basic data types like `string` and `boolean` are defined in the respective language mappings in clauses 8, 9 and 9.7.

Notice that the values for any identifier data type shall be unique in the test system implementation where uniqueness is defined as being globally distinct at any point in time. This guarantees that different objects, e.g., two timers, are identified by different identifiers and identifiers are not reused.

### 7.2.1 General abstract data types

The following abstract data types are defined and used for the definition of TCI operations.

#### 7.2.1.1 Management

<code>TciModuleIdType</code>	A value of <code>TciModuleIdType</code> is the name of a TTCN-3 module as specified in the TTCN-3 ATS. This abstract type is used for module handling.
<code>TciModuleParameterIdType</code>	A value of <code>TciModuleParameterIdType</code> is the qualified name of a TTCN-3 module parameter as specified in the TTCN-3 ATS. This abstract type is used for module parameter handling.
<code>TciTestCaseIdType</code>	A value of <code>TciTestCaseIdType</code> is the qualified name of a TTCN-3 testcase as specified in the TTCN-3 ATS. This abstract type is used for testcase handling.
<code>TciTestCaseIdListType</code>	A value of <code>TciTestCaseIdListType</code> is a list of <code>TciTestCaseIdType</code> . This abstract type is used when retrieving the list of test cases in a TTCN-3 module.
<code>TciModuleIdListType</code>	A value of type <code>TciModuleIdListType</code> is a list of <code>TciModuleIdType</code> . This abstract type is used when retrieving the list of modules which are imported by a TTCN-3 module.
<code>TciModuleParameterType</code>	A value of type <code>TciModuleParameterType</code> is a structure of <code>TciModuleParameterIdType</code> and <code>Value</code> . This abstract type is used to represent the parameter name and the default value of a module parameter.
<code>TciModuleParameterListType</code>	A value of type <code>TciModuleParameterListType</code> is a list of <code>TciModuleParameterType</code> . This abstract type is used when retrieving the module parameters of a TTCN-3 module.
<code>TciParameterType</code>	A value of type <code>TciParameterType</code> includes a TTCN-3 <code>Value</code> , which can be absent, and a value of <code>TciParameterPassingModeType</code> to represent the parameter passing mode specified for the parameter in the TTCN-3 ATS.

<code>TciParameterPassingModeType</code>	A value of type <code>TciParameterPassingModeType</code> is either <code>IN</code> , <code>INOUT</code> , or <code>OUT</code> . This abstract type is used when starting a test case or when the termination of a test case is indicated.
<code>TciParameterListType</code>	A value of type <code>TciParameterListType</code> is a list of <code>TciParameterType</code> . This abstract type is used when starting a test case or when the termination of a test case is indicated.
<code>TciParameterTypeType</code>	A value of type <code>TciParameterTypeType</code> is a structure of <code>Type</code> and <code>TciParameterPassingModeType</code> . This abstract type is used to represent the type and the parameter passing mode of a test case parameter.
<code>TciParameterTypeListType</code>	A value of type <code>TciParameterTypeListType</code> is a list of <code>TciParameterTypeType</code> . This abstract type is used to represent the list of parameters of a test case.
<code>TciTestComponentKindType</code>	A value of type <code>TciTestComponentKindType</code> is a literal of the set of kinds of TTCN-3 test components, i.e., <code>CONTROL</code> , <code>MTC</code> , <code>PTC</code> , <code>SYSTEM</code> , and <code>PTC_ALIVE</code> . This abstract type is used for component handling.
<code>TciTypeClassType</code>	A value of type <code>TciTypeClassType</code> is a literal of the set of type classes in TTCN-3 such as <code>boolean</code> , <code>float</code> , <code>record</code> , etc. This abstract type is used for value handling.

### 7.2.1.2 Communication

`TciBehaviourIdType` A value of type `TciBehaviourIdType` identifies a TTCN-3 behaviour functions.

Additional abstract data types with the prefix `Tri` are taken from [3]: `TriPortIdType`, `TriPortIdListType`, `TriComponentIdType`, `TriComponentIdListType`, `TriAddressType`, `TriAddressListType`, and `TriMessageType`.

### 7.2.2 Abstract TTCN-3 data types and values

This clause defines the set of abstract data types that build up the TTCN-3 type and value representation. Functionality of each data type is defined by an accompanying set of operations. Operations on or using this abstract data type return either a value of this abstract type or a basic type like `boolean`.

All operations have been defined using the interface description language (IDL). Concrete language mappings for the operations on the abstract data types are given in clauses 8, 9 and 9.7. In certain languages, the application of an operation on an abstract data type is represented by passing (either by-value or by-reference, depending on the mapping) the concrete value as a parameter to the operation. Other languages might choose other referencing method to the concrete value, e.g., by considering the value as an object on which a method corresponding to the operation is invoked. To indicate the inability to perform a certain task or to indicate the absence of an optional parameter in the following, the distinct value `null` is used. It can be considered as being a reserved value indicating a special value. The language mappings will define a concrete representation of this distinct value `null`.

The abstract TTCN-3 type and value representation consists of two parts:

- an abstract data type `Type` that represents all TTCN-3 types in a TTCN-3 module;
- different abstract data types that represent TTCN-3 values, i.e., TTCN-3 values of a given TTCN-3 type. This can be either values of TTCN-3 predefined types or of TTCN-3 user-defined types.

For accessing, evaluating, and coding the TTCN-3 data the test system uses the abstract data type `Type` and the different abstract value data types. Therefore, these abstract data types define the abstraction level between the TTCN-3 executable (TE) and the remaining test system using the TCI interfaces.

### 7.2.2.1 Abstract TTCN-3 data types

According to the present document TTCN-3 types, either predefined or user-defined, will be represented at the TCI interfaces using the abstract data type `Type`.

For the abstract data type `Type` a set of operations is defined to:

- reference predefined and user-defined TTCN-3 data types; and
- create and maintain TTCN-3 values.

The following operations are defined for the abstract data type `Type`:

`TciModuleIdType getDefiningModule()` Returns the module identifier of the module in which type is defined. Returns the distinct value null if type is a TTCN-3 base type, e.g., boolean, integer, etc.).

`TString getName()` Returns the name of the type as defined in the TTCN-3 module. If the type is a nested type without explicit name, the TE has to create an additional unique identifier for this type which is consistently used in TRI/TCI.

NOTE 1 – The creation of identifiers for nested types is tool dependent.

NOTE 2 – The naming for a nested type without explicit name can follow the rules defined in clauses 6.2.1.1 and 6.2.3.2 of [1], e.g., `TypeIdOrExpression.ElementId` and `TypeId[-]`, respectively.

`TciTypeClassType getTypeClass()` Returns the type class of the respective type. A value of `TciTypeClassType` can have one of the following constants: ADDRESS, ANYTYPE, ARRAY, BITSTRING, BOOLEAN, CHARSTRING, COMPONENT, ENUMERATED, FLOAT, HEXSTRING, INTEGER, OCTETSTRING, RECORD, RECORD\_OF, SET, SET\_OF, UNION, UNIVERSAL\_CHARSTRING, VERDICT.

`Value newInstance()` Returns a freshly created value of the given type. This initial value of the created value is undefined.

NOTE 3 – Newly created instances of empty record types are considered to be initialized.

`TString getTypeEncoding()` Returns the type encoding attribute as defined in the TTCN-3 module.

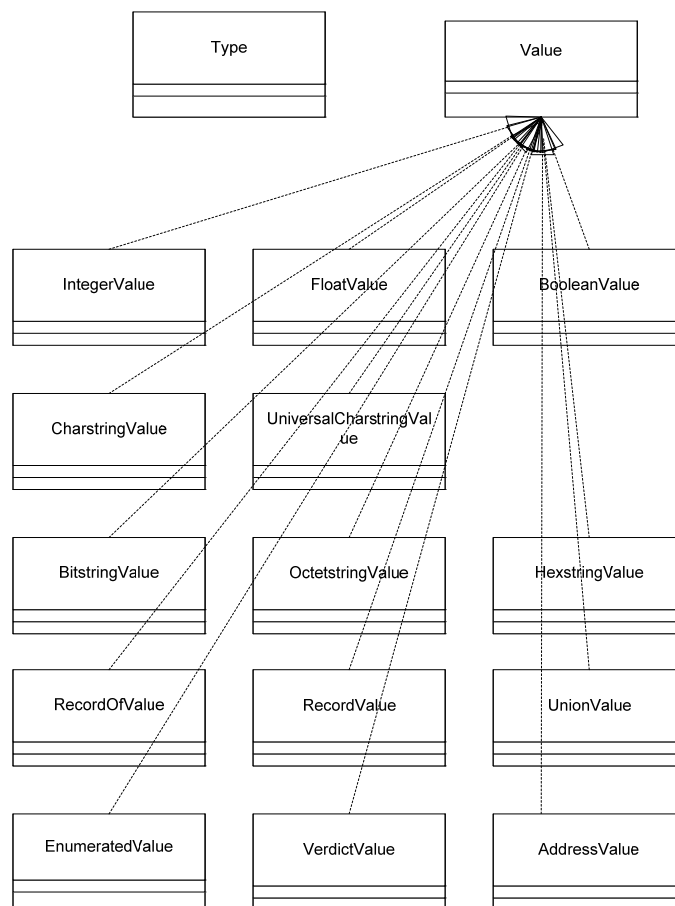
`TString getTypeEncodingVariant()` This operation returns the value encoding variant attribute as defined in the TTCN-3 module, if any. If no encoding variant attribute is defined the distinct value null is returned.

`TStringseq getTypeExtension()` Returns the type extension attribute as defined in the TTCN-3 module.

### 7.2.2.2 Abstract TTCN-3 values

According to the present document, TTCN-3 values are represented at the TCI interfaces via numerous abstract data types.

Figure 4 presents the hierarchy between the abstract data types for TTCN-3 values (short: abstract values).



**Figure 4 – Hierarchy of abstract values**

As shown in Figure 4, all TTCN-3 abstract values share the same base abstract data type `value`. All operations defined on this common base data type are implicitly defined also for the abstract value types derived from it.

#### 7.2.2.2.1 The abstract data type `value`

The following operations are defined on the base abstract data type `value`. The concrete representations of these operations are defined in the respective language mapping sections:

<code>Type</code> <code>getType()</code>	Returns the type of the specified value.
<code>TBoolean</code> <code>notPresent()</code>	Returns <code>true</code> if the specified value is omit, <code>false</code> otherwise.
<code>TString</code> <code>getValueEncoding()</code>	Returns the value encoding attribute as defined in the TTCN-3 module, if any. If no encoding attribute is defined the distinct value <code>null</code> is returned.
<code>TString</code> <code>getValueEncodingVariant()</code>	Returns the value encoding variant attribute as defined in the TTCN-3 module, if any. If no encoding variant attribute is defined the distinct value <code>null</code> is returned.

#### 7.2.2.2.2 The abstract data type `IntegerValue`

The abstract data type `IntegerValue` is based on the abstract data type `value`. It represents TTCN-3 integer values.

The following operations are defined on the abstract data type `IntegerValue`:

<code>TInteger</code> <code>getInt()</code>	Returns the integer value of this TTCN-3 integer.
---	---

`void setInt(in TInteger value)`                Sets this `IntegerValue` to `value`.

#### **7.2.2.2.3 The abstract data type `FloatValue`**

The abstract data type `FloatValue` is based on the abstract data type `Value`. It represents TTCN-3 float values.

The following operations are defined on the abstract data type `FloatValue`:

`TFloat getFloat()`                                Returns the float value of this TTCN-3 float.

`void setFloat(in TFloat value)`                Sets this `FloatValue` to `value`.

#### **7.2.2.2.4 The abstract data type `BooleanValue`**

The abstract data type `BooleanValue` is based on the abstract data type `Value`. It represents TTCN-3 boolean values.

The following operations are defined on the abstract data type `BooleanValue`:

`TBoolean getBoolean ()`                        Returns the boolean value of the TTCN-3 boolean.

`void setBoolean(in TBoolean value)`        Sets this boolean value to `value`.

#### **7.2.2.2.5 The abstract data type `CharstringValue`**

The abstract data type `CharstringValue` is based on the abstract data type `Value`. It represents TTCN-3 charstring values. `TChar` is a character within a charstring value.

The following operations are defined on the abstract data type `CharstringValue`:

`TString getString()`                            Returns the string value of the TTCN-3 charstring. The textual representation of the empty TTCN-3 charstring is "", while its length is zero.

`void setString(in TString value)`            Sets this `CharstringValue` to `value`.

`TChar getChar(in TInteger position)`        Returns the char value of the TTCN-3 charstring at position. Position 0 denotes the first char of the TTCN-3 charstring. Valid values for position are from 0 to `length - 1`.

`void setChar(in TInteger position, in TChar value)`        Set the character at position to `value`. Valid values for position are from 0 to `length - 1`.

`TInteger getLength()`                        Returns the length of this `CharstringValue` in chars, zero if the value of this `CharstringValue` is omit.

`void setLength(in TInteger len)`            `setLength` first resets this `CharstringValue` to its initial value and afterwards sets the length of this `CharstringValue` in chars to `len`.

#### **7.2.2.2.6 The abstract data type `UniversalCharstringValue`**

The abstract data type `UniversalCharstringValue` is based on the abstract data type `Value`. It represents TTCN-3 universal charstring values. `TUniversalChar` is a character within a universal charstring value.

The following operations are defined on the abstract data type `UniversalCharstringValue`:

`TString getString()`                            Returns the textual representation of this `UniversalCharstringValue`, as defined in TTCN-3.

<code>void setString(in TString value)</code>	Sets the value of this <code>UniversalCharstringValue</code> according to the textual representation as defined by <code>value</code> .
<code>TUniversalChar getChar(in TInteger position)</code>	Returns the universal char value of the TTCN-3 universal charstring at <code>position</code> . <code>Position 0</code> denotes the first <code>TUniversalChar</code> of the TTCN-3 universal charstring. Valid values for <code>position</code> are from <code>0</code> to <code>length - 1</code> .
<code>void setChar(in TInteger position, in TUniversalChar value)</code>	Sets the universal char at <code>position</code> to <code>value</code> . Valid values for <code>position</code> are from <code>0</code> to <code>length - 1</code> .
<code>TInteger getLength()</code>	Returns the length of this universal charstring value in universal chars, zero if the value of this universal charstring value is omit.
<code>void setLength(in TInteger len)</code>	<code>setLength</code> first resets this <code>UniversalCharstringValue</code> to its initial value and afterwards sets the length of this <code>UniversalCharstringValue</code> in universal chars to <code>len</code> .

#### 7.2.2.2.7 The abstract data type `BitstringValue`

The abstract data type `BitstringValue` is based on the abstract data type `Value`. It represents TTCN-3 bitstring values:

The following operations are defined on the abstract data type `BitstringValue`.

<code>TString getString()</code>	Returns the textual representation of this <code>BitstringValue</code> , as defined in TTCN-3. E.g., the textual representation of <code>0101</code> is <code>'0101'B</code> . The textual representation of the empty TTCN-3 bitstring is <code>'B</code> , while its length is zero.
<code>void setString(in TString value)</code>	Sets the value of this <code>BitstringValue</code> according to the textual representation as defined by <code>value</code> . E.g., the value of this <code>BitstringValue</code> is <code>0101</code> if the textual representation in <code>value</code> is <code>'0101'B</code> .
<code>TChar getBit(in TInteger position)</code>	Returns the value ( <code>0   1</code> ) at <code>position</code> of this TTCN-3 bitstring as a character. <code>Position 0</code> denotes the first bit of the TTCN-3 bitstring. Valid values for <code>position</code> are from <code>0</code> to <code>length - 1</code> .
<code>void setBit(in TInteger position, in TInteger value)</code>	Sets the bit at <code>position</code> to the value ( <code>0   1</code> ). <code>Position 0</code> denotes the first bit in this <code>BitstringValue</code> . Valid values for <code>position</code> are from <code>0</code> to <code>length - 1</code> .
<code>TInteger getLength()</code>	Returns the length of this <code>BitstringValue</code> in bits, zero if the value of this <code>BitstringValue</code> is omit.
<code>void setLength(in TInteger len)</code>	<code>setLength</code> first resets this <code>BitstringValue</code> to its initial value and afterwards sets the length of this <code>BitstringValue</code> in bits to <code>len</code> .



### 7.2.2.2.8 The abstract data type `OctetstringValue`

The abstract data type `OctetstringValue` is based on the abstract data type `Value`. It represents TTCN-3 `octetstring` values.

The following operations are defined on the abstract data type `OctetstringValue`:

<code>TString getString()</code>	Returns the textual representation of this <code>OctetstringValue</code> , as defined in TTCN-3. E.g., the textual representation of <code>0xCAFFEE</code> is <code>'CAFFEE'O</code> . The textual representation of the empty TTCN-3 <code>octetstring</code> is <code>'O</code> , while its length is zero.
<code>void setString(in TString value)</code>	Sets the value of this <code>OctetstringValue</code> according to the textual representation as defined by <code>value</code> . E.g., The value of this <code>OctetstringValue</code> is <code>0xCAFFEE</code> if the textual representation in <code>value</code> is <code>'CAFFEE'O</code> .
<code>TChar getOctet(in TInteger position)</code>	Returns the value (0..255) at position of this TTCN-3 <code>octetstring</code> . Position 0 denotes the first octet of the TTCN-3 <code>octetstring</code> . Valid values for position are from 0 to <code>length - 1</code> .
<code>void setOctet(in TInteger position, in TInteger value)</code>	Sets the octet at <code>position</code> to <code>value</code> (0..255). Position 0 denotes the first octet in the <code>octetstring</code> . Valid values for position are from 0 to <code>length - 1</code> .
<code>TInteger getLength()</code>	Returns the length of this <code>OctetstringValue</code> in octets, zero if the value of this <code>OctetstringValue</code> is <code>omit</code> .
<code>void setLength(in TInteger len)</code>	<code>setLength</code> first resets this <code>OctetstringValue</code> to its initial value and afterwards sets the length of this <code>OctetstringValue</code> in octets to <code>len</code> .

### 7.2.2.2.9 The abstract data type `HexstringValue`

The abstract data type `HexstringValue` is based on the abstract data type `Value`. It represents TTCN-3 `hexstring` values.

The following operations are defined on the abstract data type `HexstringValue`:

<code>TString getString()</code>	Returns the textual representation of this <code>HexstringValue</code> , as defined in TTCN-3. E.g., the textual representation of <code>0xAFFEE</code> is <code>'AFFEE'H</code> . The textual representation of the empty TTCN-3 <code>hexstring</code> is <code>'H</code> , while its length is zero.
<code>void setString(in TString value)</code>	Sets the value of this <code>HexstringValue</code> according to the textual representation as defined by <code>value</code> . E.g., The value of this <code>HexstringValue</code> is <code>0xAFFEE</code> if the textual representation in <code>value</code> is <code>'AFFEE'H</code> .
<code>TChar getHex(in TInteger position)</code>	Returns the value (0..15) at position of this TTCN-3 <code>hexstring</code> . Position 0 denotes the first hex digits of the TTCN-3 <code>hexstring</code> . Valid values for position are from 0 to <code>length - 1</code> .
<code>void setHex(in TInteger position, in TInteger value)</code>	

Sets the hex digit at position to value (0..15). Position 0 denotes the first octet in the hexstring. Valid values for position are from 0 to length - 1.

TInteger getLength()

Returns the length of this HexstringValue in octets, zero if the value of this HexstringValue is omit.

void setLength(in TInteger len)

setLength first resets this HexstringValue to its initial value and afterwards sets the length of this HexstringValue in hex digits to len.

#### 7.2.2.2.10 The abstract data type RecordValue

The abstract data type RecordValue is based on the abstract data type Value. It specifies how to get and set the TTCN-3 record type.

NOTE – Newly created instances of empty record types are considered to be initialized.

The same abstract data type applies for values whose type class is SET. The distinction between record and set is only relevant at matching time.

The following operations are defined on the abstract data type RecordValue:

Value getField(in TString fieldName) Returns the value of the field named fieldName. The return value is the common abstract base type Value, as a record field can have any type defined in TTCN-3. If the field cannot be obtained from the record the distinct value null is returned.

void setField(in TString fieldName, in Value value)

Sets the field named fieldName of the record to value. No assumption shall be made on how a field is stored in a record. An internal implementation might choose to store a reference to this value or to copy the value. It is safe to assume that the value is copied. Therefore it should be assumed that subsequent modifications of value will not be considered in the record.

TStringSeq getFieldNames()

Returns a sequence of string of field names, the empty sequence, if the record has no fields.

void setFieldOmitted(in TString fieldName)

Mark the referenced field of the record as being omitted.

#### 7.2.2.2.11 The abstract data type RecordOfValue

The abstract data type RecordOfValue is based on the abstract data type Value. It specifies how to get and set elements in TTCN-3 record of types. The same abstract data type applies for value whose type class is ARRAY or SET\_OF. The distinction between record of, set of, and array is only relevant at matching time.

The following operations are defined on the abstract data type RecordOfValue:

Value getField(in TInteger position) Returns the value of the record of at position if position is between zero and length - 1, the distinct value null otherwise. Also for array values indices start from 0, independent of the lower index

bound. The return value is the common abstract base type `Value`, as a record of can have fields of any type defined in TTCN-3.

`void setField(in TInteger position, in Value value)`

Sets the field at `position` to `value`. If `position` is greater than `(length - 1)` the record of is extended to have the length `(position + 1)`. The record of elements between the original position at `length` and `position - 1` is set to `omit`. No assumption shall be made on how a field is stored in a record of. An internal implementation might choose to store a reference to this value or to copy the value. It is safe to assume that the value is copied. Therefore, it should be assumed that subsequent modifications of `value` will not be considered in the record of. Also for array values indices start from 0, independent of the lower index bound.

`void appendField(in Value value)`

Appends the value at the end of the record of, i.e., at position `length`. No assumption shall be made on how a field is stored in a record of. An internal implementation might choose to store a reference to this value or to copy the value. It is safe to assume that the value is copied. Therefore, it should be assumed that subsequent modifications of `value` will not be considered in the record of.

`Type getElementType()`

Returns the `Type` of the elements of this record of.

`TInteger getLength()`

Returns the actual length of the record of `value`, zero if the record of `value` is `omit`.

`void setLength(in TInteger len)`

Sets the length of the record of to `len`. If `len` is greater than the original length, newly created elements have the value `omit`. If `len` is less or equal than the original length this operation is ignored.

`TInteger getOffset()`

Returns the lowest possible index. For a record of or set of `value` this is always 0. For an array value, this is the lower index bound used in the type definition.

#### 7.2.2.2.12 The abstract data type `UnionValue`

The abstract data type `UnionValue` is based on the abstract data type `Value`. It specifies how to get and set variants in a TTCN-3 union type. The TTCN-3 anytype is represented by a `UnionValue` where the type class of the type obtained by `getType()` is `ANYTYPE`. For details on type classes see clause 7.2.2.1.

The following operations are defined on the abstract data type `UnionValue`:

`Value getVariant(in TString variantName)`

Returns the value of the TTCN-3 union `variantName`, if `variantName` equals the result of `getPresentVariantName`, the distinct value `null`

otherwise. `variantName` denotes the name of the union variant as defined in the TTCN-3 module.

`void setVariant(in TString variantName, in Value value)`

Sets `variantName` of the union to `value`. If `variantName` is not defined for this union this operation is ignored. If another variant was selected the new variant is selected instead.

`TString getPresentVariantName()`

Returns a String representing the currently selected variant name in the given TTCN-3 union. The distinct value null is returned if no variant is selected.

`TStringSeq getVariantNames()`

Returns a sequence of string of variant names, the distinct value null, if the union has no fields. If the `UnionValue` represents the TTCN-3 anytype, i.e., the type class of the type obtained by `getType()` is ANYTYPE, all predefined and user-defined TTCN-3 types is returned.

#### 7.2.2.2.13 The abstract data type `EnumeratedValue`

The abstract data type `EnumeratedValue` is based on the abstract data type `Value`. It specifies how TTCN-3 `enumerated` can be set and get.

The following operations are defined on the abstract data type `EnumeratedValue`:

`TString getEnum()`

Returns the string identifier of this `EnumeratedValue`. This identifier equals the identifier in the TTCN-3 specification.

`void setEnum(in TString enumValue)`

Sets the enum to `enumValue`. If `enumValue` is not an allowed value for this enumeration the operation is ignored.

`TInteger getInt()`

Returns the integer value of this `EnumeratedValue`. This integer equals the user-assigned integer value in the TTCN-3 specification or the automatically assigned integer value.

`setInt(in TInteger intValue)`

Sets the integer value of this `EnumeratedValue`. This integer should equal the user-assigned integer value in the TTCN-3 specification or the automatically assigned integer value. If `intValue` is not an allowed value for this enumeration the operation is ignored.

#### 7.2.2.2.14 The abstract data type `VerdictValue`

The abstract data type `VerdictValue` is based on the abstract data type `Value`. It specifies how TTCN-3 `verdict` can be set and get.

The following operations are defined on the abstract data type `VerdictValue`:

`TInteger getVerdict()`

Returns the integer value for this `VerdictValue`. The integer is one of the following constants: `ERROR`, `FAIL`, `INCONC`, `NONE`, `PASS`, `USER_ERROR`.

`void setVerdict(in TInteger verdict)`

Sets this `VerdictValue` to `verdict`. Note that a `VerdictValue` can be set to any of the above mentioned verdicts at any time. The `VerdictValue` does not perform any verdict calculations as defined in TTCN-3. For example, it is legal to set the `VerdictValue` first to `INCONC` and then to `PASS`.

#### 7.2.2.2.15 The abstract data type `AddressValue`

The following operations are defined on the base abstract data type `AddressValue`. The concrete representations of these operations are defined in the respective language mapping sections:

<code>Value getAddress()</code>	Returns the address value, which will no longer be of type class <code>ADDRESS</code> but rather of the actual type used for address.
<code>void setAddress(in Value value)</code>	Sets this address value to <code>value</code> .

### 7.2.3 Abstract logging types

#### 7.2.3.1 The abstract data type `TciValueTemplate`

The following operations are defined on the abstract data type `TciValueTemplate`. The concrete representations of these operations are defined in the respective language mapping sections:

<code>TBoolean isOmit()</code>	Returns <code>true</code> if the template is an omit template.
<code>TBoolean isAny()</code>	Returns <code>true</code> if the template is an any template.
<code>TBoolean isAnyOrOmit()</code>	Returns <code>true</code> if the template is an any or omit template.
<code>TString getTemplateDef()</code>	Returns the definition of that template.

#### 7.2.3.2 The abstract data type `TciNonValueTemplate`

The following operations are defined on the abstract data type `TciNonValueTemplate`. The concrete representations of these operations are defined in the respective language mapping sections:

<code>TBoolean isAny()</code>	Returns <code>true</code> if the template is an any template.
<code>TBoolean isAll()</code>	Returns <code>true</code> if the template is an all template.
<code>TString getTemplateDef()</code>	Returns the definition of that template.

#### 7.2.3.3 The Value List and Mismatch Types

The following abstract data types are defined and used for the logging of differences between values and templates:

<code>TciValueList</code>	A value of <code>TciValueList</code> is a list of values.
<code>TciValueDifference</code>	A value of <code>TciValueDifference</code> is a structure containing a value, a template, and a description for the reason of this difference.
<code>TciValueDifferenceList</code>	A value of <code>TciValueDifferenceList</code> is a sequence of value differences.

The following operations are defined on the abstract data type `TciValueList`. The concrete representations of these operations are defined in the respective language mapping sections:

<code>TInteger size()</code>	Returns the number of values in this list.
<code>TBoolean isEmpty()</code>	Returns <code>true</code> if this list contains no values.
<code>Value get(in TInteger index)</code>	Returns the value at the specified position.

The following operations are defined on the abstract data type `TciValueDifference`. The concrete representations of these operations are defined in the respective language mapping sections:

<code>Value getValue()</code>	Returns the value of the <code>TciValueDifference</code> .
<code>TciValueTemplate getTciValueTemplate()</code>	Returns the template of the <code>TciValueDifference</code> .
<code>String getDescription()</code>	Returns the description of the mismatch.

The following operations are defined on the abstract data type `TciValueDifferenceList`. The concrete representations of these operations are defined in the respective language mapping sections:

<code>TInteger size()</code>	Returns the number of values in the list.
<code>TBoolean isEmpty()</code>	Returns <code>true</code> if the list contains no values.
<code>TciValueDifference get(in TInteger index)</code>	Returns the <code>TciValueDifference</code> at the specified position.

#### 7.2.3.4 The Status Types

The following abstract data types are defined and used for the logging of component and timer status:

<code>ComponentStatusType</code>	A value of <code>ComponentStatusType</code> is either "inactiveC", "runningC", "stoppedC", "killedC", or "nullC".
<code>TimerStatusType</code>	A value of <code>TimerStatusType</code> is either "runningT", "inactiveT", "expiredT", or "nullC".
<code>PortStatusType</code>	A value of <code>PortStatusType</code> is either "startedP", "haltedP", or "stoppedP".

### 7.3 TCI operations

This clause specifies the operations that a TTCN-3 executable shall provide to a test system (*required operations*) and which functionality shall be provided by the test system to the TTCN-3 executable (*provided operations*).

The terms "required" and "provided" reflect the fact that the present document defines the requirements on a TTCN-3 executable from a user's point of view. The user "requires" from a TTCN-3 executable certain functionality to build a complete TTCN-3-based test system. To fulfil its task the TTCN-3 executable has to inform the user on certain events where the user has to "provide" this possibility to the TTCN-3 executable.

All operation definitions in this clause are defined using the interface definition language (IDL). Concrete language mappings are defined in clauses 8, 9 and 9.7. Annex B provides for the logging interface an alternative mapping to XML.

For every TCI operation call all *in*, *inout*, and *out* parameters listed in the particular operation definition are mandatory. The value of an *in* parameter is specified by the calling entity. Calling entity refers to the direction of the call. For operations on a *required* interface the calling entity is

the test system while the called entity is the TTCN-3 executable. For operations on a *provided* interface the calling entity is the TTCN-3 executable while the test system is the called entity.

Similarly, the value of an *out* parameter is specified by the called entity. In the case of an *inout* parameter, a value is first specified by the calling entity but may be replaced with a new value by the called entity. Note that although TTCN-3 also uses *in*, *inout*, and *out* for signature definitions the denotations used in TCI IDL specification are not related to those in a TTCN-3 specification.

Operation calls should use a reserved value to indicate the absence of parameters. The reserved values for these types are defined in each language mapping and will be subsequently referred to as the null value.

In addition, the null value will also be used to indicate the inability to perform a certain task.

As this clause specifies interfaces only and does not suggest concrete implementations on how to perform the specified functionality the term entity will be used to identify the part of the test system implementation that implements this interface and performs the requested functionality. For example, the calling entity in the `tcISendConnected` operation is the TE, i.e., the part of test system implementation that provides the TE functionality.

All functions in the interface are described using the following template. Descriptions that are not applicable for certain operations are removed.

<b>Signature</b>	IDL Signature
<b>In Parameters</b>	Description of data passed as parameters to the operation from the calling entity to the called entity.
<b>Out Parameters</b>	Description of data passed as parameters to the operation from the called entity to the calling entity.
<b>InOut Parameters</b>	Description of data passed as parameters to the operation from the calling entity to the called entity and from the called entity back to the calling entity.
<b>Return Value</b>	Description of data returned from the operation to the calling entity.
<b>Constraint</b>	Description of any constraints when the operation can be called.
<b>Effect</b>	Behaviour required of the called entity before the operation may return.

### 7.3.1 The TCI-TM interface

The TCI test management interface (TCI-TM) describes the operations a TTCN-3 executable is required to implement and the operations a test management implementation shall provide to the TE (Figure 5).

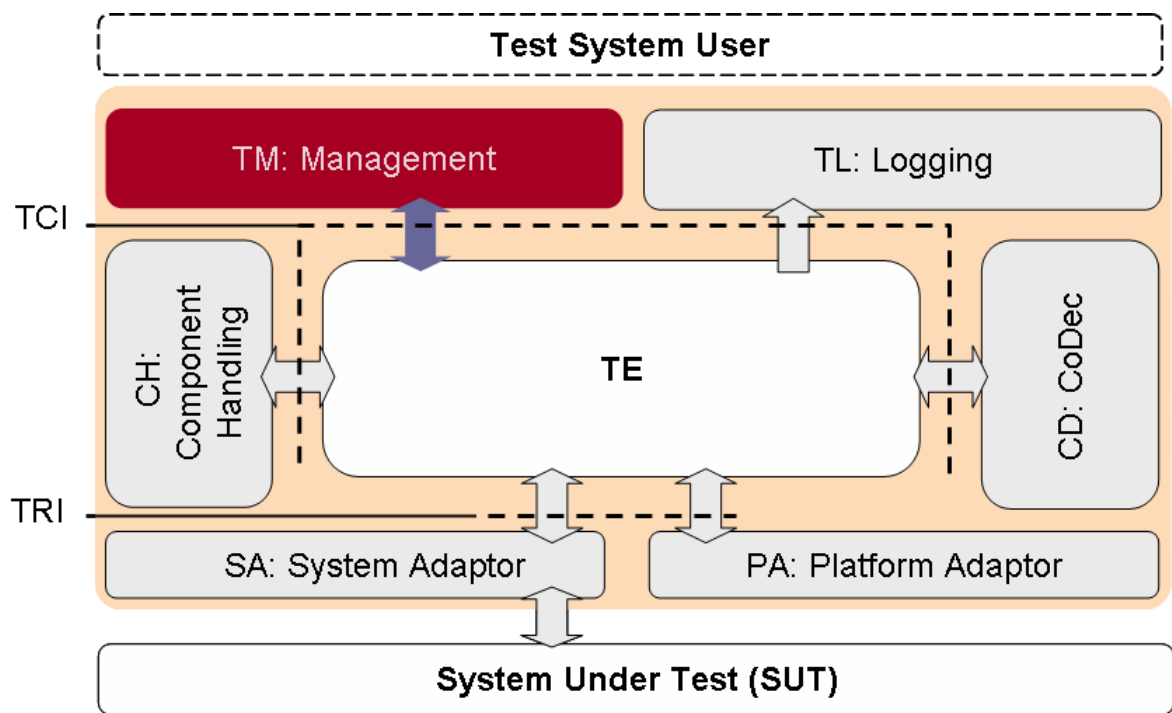


Figure 5 – The TCI-TM interface

A test management implementation provides overall test management to the test system user. It requires from the TE the presence of operations to start and stop test execution of a TTCN-3 module or of certain test cases in a TTCN-3 module. In turn it provides operations to the TE for resolving module parameter at runtime and the indication of execution termination.

Appendix I illustrates the usage and sequential ordering of operation calls by either the TE or the test management.

### 7.3.1.1 TCI-TM required

This clause specifies the operations the TM requires from the TE. In addition to the operations specified in this clause, a test management requires the operations as required at the TCI-CD interface.

#### 7.3.1.1.1 tciRootModule

<b>Signature</b>	void tciRootModule (in TciModuleIdType moduleName)	
<b>In Parameters</b>	moduleName	The moduleName denotes the module identifiers as defined in the TTCN-3 module.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be used only if neither the control part nor a test case is currently being executed.	
<b>Effect</b>	tciRootModule selects the indicated module for execution through a subsequent call using tciStartTestCase or tciStartControl. A tciError will be issued by the TE if no such module exists.	



### 7.3.1.1.2 tciGetImportedModules

<b>Signature</b>	TciModuleIdListType tciGetImportedModules()
<b>Return Value</b>	A list of all imported modules of the root module. The modules are ordered as they appear in the TTCN-3 module. If no imported modules exist, an empty module list is returned.
<b>Constraint</b>	Shall be used only if a root module has been set before.
<b>Effect</b>	The TE provides to the management a list of imported modules of the root module. If no imported module exists, an empty module list is returned. If the TE cannot provide a list, the distinct null value shall be returned.

### 7.3.1.1.3 tciGetModuleParameters

<b>Signature</b>	TciModuleParameterListType tciGetModuleParameters (in TciModuleIdType moduleName)	
<b>In Parameters</b>	moduleName	The moduleName denotes the module identifiers for which the module parameters should be retrieved.
<b>Return Value</b>	A list of all module parameters of the identified module. The parameters are ordered as they appear in the TTCN-3 module. If no parameters exist, an empty module parameter list is returned.	
<b>Constraint</b>	Shall be used only if a root module has been set before.	
<b>Effect</b>	The TE provides to the management a list of module parameters of the identified module. If no module parameters exist, an empty module parameter list is returned. If the TE cannot provide a list, the distinct null value shall be returned.	

### 7.3.1.1.4 tciGetTestCases

<b>Signature</b>	TciTestCaseIdListType tciGetTestCases ()
<b>Return Value</b>	A list of all test cases that are either defined in or imported into the root module.
<b>Constraint</b>	Shall be used only if a root module has been set before.
<b>Effect</b>	The TE provides to the management a list of test cases. If no test cases exist, an empty test case list is returned. If the TE cannot provide a list, the distinct null value shall be returned.

### 7.3.1.1.5 tciGetTestCaseParameters

<b>Signature</b>	TciParameterTypeListType tciGetTestCaseParameters (in TciTestCaseIdType testCaseId)	
<b>In Parameters</b>	testCaseId	A test case identifier as defined in the TTCN-3 module.
<b>Return Value</b>	A list of all parameter types of the given test case. The parameter types are ordered as they appear in the TTCN-3 signature of the test case. If no parameters exist, an empty parameter type list is returned.	
<b>Constraint</b>	Shall be used only if a root module has been set before.	
<b>Effect</b>	The TE provides to the management a list of parameter types of the given test case. If no test case parameters exist, an empty parameter type list is returned. If the TE cannot provide a list, the distinct null value shall be returned.	

### 7.3.1.1.6 tciGetTestCaseTSI

<b>Signature</b>	TriPortIdListType tciGetTestCaseTSI (in TciTestCaseIdType testCaseId)	
<b>In Parameters</b>	testCaseId	A test case identifier as defined in the TTCN-3 module.
<b>Return Value</b>	A list of all system ports of the given test case that have been declared in the definition of the system component for the test case, i.e., the TSI ports. If a system component has not been explicitly defined for the test case, then the list contains all communication ports of the MTC test component. The ports are ordered as they appear in the respective TTCN-3 component type declaration. If no system ports exist, an empty list, i.e., a list of length zero is returned.	
<b>Constraint</b>	Shall be used only if a root module has been set before.	
<b>Effect</b>	The TE provides to the management a list of system ports of the given test case. If no system ports exist, an empty port list is returned. If the TE cannot provide a list, the distinct null value shall be returned.	

### 7.3.1.1.7 tciStartTestCase

<b>Signature</b>	void tciStartTestCase(in TciTestCaseIdType testCaseId, in TciParameterListType parameterList)	
<b>In Parameters</b>	testCaseId	A test case identifier as defined in the TTCN-3 module.
	parameterList	A list of Values where each value defines a parameter from the parameter list as defined in the TTCN-3 test case definition. The parameters in parameterList are ordered as they appear in the TTCN-3 signature of the test case. If no parameters have to be passed either the null value or an empty parameterList, i.e., a list of length zero shall be passed.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called only if a module has been selected before. Only testCaseIds for test cases that are declared in the currently selected TTCN-3 module shall be passed. Test cases that are imported in a referenced module cannot be started. To start imported test cases the referenced (imported) module must be selected first using the tciRootModule operation.	
<b>Effect</b>	tciStartTestCase starts a testcase in the currently selected module with the given parameters. A tciError will be issued by the TE if no such test case exists. All in and inout test case parameters in parameterList contain Value. All out test case parameters in parameterList shall contain the distinct value of null since they are only of relevance when the test case terminates.	

### 7.3.1.1.8 tciStopTestCase

<b>Signature</b>	void tciStopTestCase()	
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called only if a module has been selected before.	
<b>Effect</b>	tciStopTestCase stops the testcase currently being executed. If the TE is not executing a test case, the operation will be ignored. If the control part is being executed, tciStopTestCase will stop execution of the currently executed test case, i.e., the execution of the test case that has recently been indicated using the provided operation tciTestCaseStarted. A possible executing control part will continue execution as if the test case has stopped normally and returned with verdict ERROR.	

### 7.3.1.1.9 tciStartControl

<b>Signature</b>	TriComponentId tciStartControl()
<b>Return Value</b>	A TriComponentId that represents the test component the module control part is executed on. If the TE cannot start control part of the selected module the distinct value null will be returned.
<b>Constraint</b>	Shall be called only if a module has been selected before.
<b>Effect</b>	Starts the control part of the selected module. The control part will start TTCN-3 test cases as described in TTCN-3. While executing the control part the TE will call the <i>provided</i> operation tciTestCaseStarted and tciTestCaseTerminated for every test case that has been started and that has terminated. After termination of the control part the TE will call the <i>provided</i> operation tciControlPartTerminated.

### 7.3.1.1.10 tciStopControl

<b>Signature</b>	void tciStopControl()
<b>Return Value</b>	void
<b>Constraint</b>	Shall only be called if a module has been selected before.
<b>Effect</b>	tciStopControl stops execution of the control part. If no control part is currently being executed the operation will be ignored. If a test case has been started directly this will stop execution of the current test case as if tciStopTestCase has been called.

### 7.3.1.2 TCI-TM provided

This clause specifies the operations the TM has to provide to the TE.

#### 7.3.1.2.1 tciTestCaseStarted

<b>Signature</b>	void tciTestCaseStarted(in TciTestCaseIdType testCaseId, in TciParameterListType parameterList, in TFloat timer)	
<b>In Parameters</b>	testCaseId	A test case identifier as defined in the TTCN-3 module.
	parameterList	A list of values that are part of the test case signature. The parameters in parameterList are ordered as they appear in the TTCN-3 test case declaration.
	timer	A float value representing the duration of the test case timer.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall only be called after either the control part of the module or a test case has been started using the <i>required</i> operations tciStartControl or tciStartTestCase.	
<b>Effect</b>	tciTestCaseStarted indicates to the TM that a test case with testCaseId has been started. It will not be distinguished whether the test case has been started explicitly using the <i>required</i> operation tciStartTestCase or implicitly while executing the control part.	

### 7.3.1.2.2 tciTestCaseTerminated

<b>Signature</b>	void tciTestCaseTerminated(in VerdictValue verdict, in TciParameterListType parameterList)	
<b>In Parameters</b>	verdict	The final verdict of the test case.
	parameterList	A list of values that are part of the test case signature. The parameters in parameterList are ordered as they appear in the TTCN-3 test case declaration.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall only be called after either the control part of the module or a test case has been started using the <i>required</i> operations tciStartControl or tciStartTestCase.	
<b>Effect</b>	This operation will be called by the TE to indicate the test management that the test case that has been currently executed on the MTC has terminated and that the final verdict was verdict. On the invocation of a tciTestCaseTerminated operation all <i>out</i> and <i>inout</i> test case parameters contain Values. All in test case parameters contain the distinct value of null because they are only of relevance to the test case start but not in the reply to the call.	

### 7.3.1.2.3 tciControlTerminated

<b>Signature</b>	void tciControlTerminated ()	
<b>Return Value</b>	void	
<b>Constraint</b>	Shall only be called when the module execution has been started using the tciStartControl operation.	
<b>Effect</b>	This operation will be called by the TE to indicate the test management that the control part of the selected module has just terminated execution.	

### 7.3.1.2.4 tciGetModulePar

<b>Signature</b>	Value tciGetModulePar (in TciModuleParameterIdType parameterId)	
<b>In Parameters</b>	parameterId	The identifier of the module parameter as defined in the TTCN-3 module.
<b>Return Value</b>	A value.	
<b>Constraint</b>	This operation shall be called whenever the TE needs to access the value of a module parameter. Every accessed module parameter will be resolved only once between a tciStartTestCase and tciTestCaseTerminated pair if a test case has been started explicitly or between a tciStartControl and tciControlTerminated pair if the control part of a module has been started.	
<b>Effect</b>	The management provides to the TE a Value for the indicated parameterId. Every call of tciGetModulePar () will return the same value throughout the execution of an explicitly started test case or throughout the execution of a control part If the management cannot provide a TTCN-3 value, the distinct null value shall be returned.	

### 7.3.1.2.5 tciLog

<b>Signature</b>	void tciLog (in TriComponentIdType testComponentId, in TString message)	
<b>In Parameters</b>	testComponentId	Identifier of the component that logs the message.
	message	A string value, i.e., the message to be logged.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by the TE when the TTCN-3 statement log will be executed, either in the control part of a module or within the test case.	
<b>Effect</b>	The TM presents testComponentId and message to the user, how this done is not within the scope of the present document.	

### 7.3.1.2.6 tciError

<b>Signature</b>	void tciError(in TString message)	
<b>In Parameters</b>	message	A string value, i.e., the error message.
<b>Return Value</b>	void	
<b>Constraint</b>	Can be called at any time by the TE to indicate an unrecoverable error situation. This error situation could either be indicated by the CH or the CD or could occur within the TE.	
<b>Effect</b>	The TE indicates the occurrence of an unrecoverable error situation. message contains a reason phrase that might be communicated to the test system user. It is up to the test management to terminate execution of test cases or control parts if running. The test management has to take explicit measures to terminate test execution immediately.	

## 7.3.2 The TCI-CD interface

The TCI codec interface (TCI-CD) describes the operations a TTCN-3 executable is required to implement and the operations a codec implementation for a certain encoding scheme shall provide to the TE (see Figure 6).



### 7.3.2.1.1 getTypeForName

<b>Signature</b>	Type getTypeForName(in TString typeName)	
<b>In Parameters</b>	typeName	The TTCN-3 name of the type as defined in the TTCN-3 module. The following are reserved type names and will return a predefined type: "integer" "float" "bitstring" "hexstring" "octetstring" "charstring" "universal charstring" "boolean" "verdicttype" typeName has to be the fully qualified type name, i.e., module.typeName
<b>Return Value</b>	A type representing the requested TTCN-3 type.	
<b>Constraint</b>	---	
<b>Effect</b>	Returns a type representing a TTCN-3 type. Predefined TTCN-3 types can be retrieved from the TE by using the TTCN-3 keywords for the predefined types. In this case typeName denotes to the basic TTCN-3 type like "charstring", "bitstring", etc. Returns the distinct value null if the requested type cannot be returned. Note that the anytype and address cannot be obtained with module set to null. Although they are predefined types they might be distinct between modules. For example, address can either be the unmodified predefined type, or a user-defined type in a module. Other predefined types cannot be redefined.	

### 7.3.2.1.2 getInteger

<b>Signature</b>	Type getInteger()
<b>Return Value</b>	An instance of Type representing a TTCN-3 integer type.
<b>Effect</b>	Constructs and returns a basic TTCN-3 integer type.

### 7.3.2.1.3 getFloat

<b>Signature</b>	Type getFloat()
<b>Return Value</b>	An instance of Type representing a TTCN-3 float type.
<b>Effect</b>	Constructs and returns a basic TTCN-3 float type.

### 7.3.2.1.4 getBoolean

<b>Signature</b>	Type getBoolean()
<b>Return Value</b>	An instance of Type representing a TTCN-3 boolean type.
<b>Effect</b>	Constructs and returns a basic TTCN-3 boolean type.

### 7.3.2.1.5 Void

### 7.3.2.1.6 getCharstring

<b>Signature</b>	Type getCharstring ()
<b>Return Value</b>	An instance of Type representing a TTCN-3 charstring type.
<b>Effect</b>	Constructs and returns a basic TTCN-3 charstringtype.

### 7.3.2.1.7 getUniversalCharstring

<b>Signature</b>	Type getUniversalCharstring ()
<b>Return Value</b>	An instance of Type representing a TTCN-3 universal charstring type.
<b>Effect</b>	Constructs and returns a basic TTCN-3 universal charstring type.

### 7.3.2.1.8 getHexstring

<b>Signature</b>	Type getHexstring ()
<b>Return Value</b>	An instance of Type representing a TTCN-3 hexstring type.
<b>Effect</b>	Constructs and returns a basic TTCN-3 hexstring type.

### 7.3.2.1.9 getBitstring

<b>Signature</b>	Type getBitstring()
<b>Return Value</b>	An instance of Type representing a TTCN-3 bitstring type.
<b>Effect</b>	Constructs and returns a basic TTCN-3 bitstring type.

### 7.3.2.1.10 getOctetstring

<b>Signature</b>	Type getOctetstring ()
<b>Return Value</b>	An instance of Type representing a TTCN-3 octetstring type.
<b>Effect</b>	Constructs and returns a basic TTCN-3 octetstring type.

### 7.3.2.1.11 getVerdict

<b>Signature</b>	Type getVerdict ()
<b>Return Value</b>	An instance of Type representing a TTCN-3 verdict type.
<b>Effect</b>	Constructs and returns a basic TTCN-3 verdict type.

### 7.3.2.1.12 tciErrorReq

<b>Signature</b>	void tciErrorReq(in TString message)	
<b>In Parameters</b>	Message	A string value, i.e., the error phrase describing the problem.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called whenever an error situation has occurred.	
<b>Effect</b>	The TE will be notified about an unrecoverable error situation within the CD and forward the error indication to the test management.	



### 7.3.2.2 TCI-CD provided

This clause specifies the operations the TM shall provide to the TE.

#### 7.3.2.2.1 decode

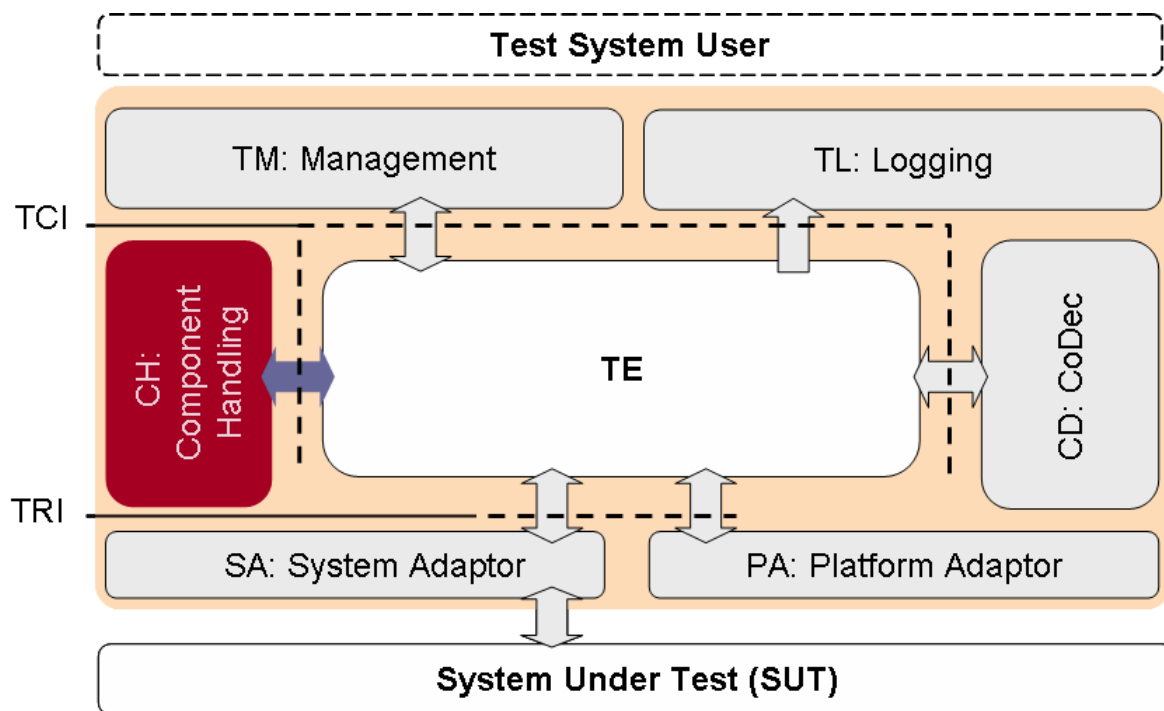
<b>Signature</b>	Value decode(in TriMessageType message, in Type decodingHypothesis)	
<b>In Parameters</b>	message	The encoded message to be decoded.
	decodingHypothesis	The hypothesis the decoding can be based on.
<b>Return Value</b>	Returns the decoded value, if the value is of a compatible type as the decodingHypothesis, else the distinct value null.	
<b>Constraint</b>	This operation shall be called whenever the TE has to decode an encoded value. The TE might decode immediately after reception of an encoded value, or might for performance considerations postpone the decoding until the actual access of the encoded value.	
<b>Effect</b>	This operation decodes message according to the encoding rules and returns a TTCN-3 value. The decodingHypothesis shall be used to determine whether the encoded value can be decoded. If an encoding rule is not self-sufficient, i.e., if the encoded message does not inherently contain its type decodingHypothesis shall be used. If the encoded value can be decoded without the decoding hypothesis, the distinct null value shall be returned if the type determined from the encoded message is not compatible with the decoding hypothesis.	

#### 7.3.2.2.2 encode

<b>Signature</b>	TriMessageType encode(in Value value)	
<b>In Parameters</b>	value	The value to be encoded.
<b>Return Value</b>	Returns an encoded TriMessage for the specified encoding rule.	
<b>Constraint</b>	This operation shall be called whenever the TE has to encode a Value.	
<b>Effect</b>	Returns an encoded TriMessage according to the encoding rules.	

### 7.3.3 The TCI-CH interface

The TCI component handling interface (TCI-CH) describes the operations a TTCN-3 executable is required to implement and the operations a component handling implementation shall provide to the TE (Figure 7).



**Figure 7 – The TCI-CH interface**

A component handling implementation distributes TTCN-3 configuration operations like create, connect and start and intercomponent communication like send on a connected port among one or more TTCN-3 executables participating in a test session. Note that although multiple instances of a TE might participate in a test session this is not mandatory.

The basic principle is that TCI-CH is not *implementing* any kind of TTCN-3 functionality. Instead it will be informed by the TE that for example a test component shall be created. Based on component handling (CH) internal knowledge the request for creation of a test component will be transmitted to another (remote) participating TE. This second (remote) participating TE will create the TTCN-3 component and will provide a handle back to the requesting (local) TE. The requesting (local) TE can now operate on the created test component via this component handle.

Within the operation definitions the terms local TE and remote TE is used to highlight the fact that a test system implementation might be distributed over several test devices, each of them hosting a complete TE. The terms "local" and "remote" always refer to the interfaces currently being described. For convenience, the term "local" refers always to the TE being either the callee of an operation (for *required* operations) or the caller of an operation (for *provided* operations). While the TE is conceptually considered as being distributed, the CH is considered to be non-distributed. This can either be achieved using a centralized architecture or by using a middleware-platform that abstracts from distribution aspects. Although the TE might be distributed over different physical devices, there might be configurations where only one, non-distributed TE will participate in a test session. In this case the term "local" and "remote" refer to the same TE instance.

Appendix I illustrates the usage and sequential ordering of operation calls by either the TE or the CH.

Although all TTCN-3 executables participating in a test session are equal, there is a distinct TE\*. This TE\* is the TE where the explicit `tcStartTestCase()` or `tcStartControl()` has been processed. The reason for this distinction is, that TE\* shall calculate the global verdict. TE\* will notify the test management upon termination of test execution and shall provide then the global verdict of the test case.

### 7.3.3.1 TCI-CH required

This clause specifies the operations the CH requires from the TE. In addition to the operations specified in this clause, all *required* operations of the TCI-CD interface are also required.

#### 7.3.3.1.1 tciEnqueueMsgConnected

<b>Signature</b>	void tciEnqueueMsgConnected (in TriPortIdType sender, in TriComponentIdType receiver, in Value rcvdMessage)	
<b>In Parameters</b>	sender	Port identifier at the sending component via which the message is sent.
	receiver	Identifier of the receiving component.
	rcvdMessage	The value to be enqueued.
<b>Return Value</b>	void	
<b>Constraint</b>	This operation shall be called by the CH at the local TE when at remote TE a <i>provided</i> tciSendConnected has been called.	
<b>Effect</b>	The TE enqueues the received value into the local port queue of the indicated receiver component.	

#### 7.3.3.1.2 tciEnqueueCallConnected

<b>Signature</b>	void tciEnqueueCallConnected (in TriPortIdType sender, in TriComponentIdType receiver, in TriSignatureIdType signature, in TciParameterListType parameterList)	
<b>In Parameters</b>	sender	Port identifier at the sending component via which the message is sent.
	receiver	Identifier of the receiving component.
	signature	Identifier of the signature of the procedure call.
	parameterList	A list of value parameters which are part of the indicated signature. The parameters in parameterList are ordered as they appear in the TTCN-3 signature declaration.
<b>Return Value</b>	void	
<b>Constraint</b>	This operation shall be called by the CH at the local TE when at a remote TE a <i>provided</i> tciCallConnected has been called. All <i>in</i> and <i>inout</i> procedure parameters contain values. All <i>out</i> procedure parameters shall contain the distinct value of null because they are only of relevance in a reply to the procedure call but not in the procedure call itself. The procedure parameters are the parameters specified in the TTCN-3 signature template.	
<b>Effect</b>	The TE enqueues the calls at the local port queue of the indicated receiver component.	

### 7.3.3.1.3 tciEnqueueReplyConnected

<b>Signature</b>	void tciEnqueueReplyConnected (in TriPortIdType sender, in TriComponentIdType receiver, in TriSignatureIdType signature, in TciParameterListType parameterList, in Value returnValue)	
<b>In Parameters</b>	sender	Identifier of the port sending the reply.
	receiver	Identifier of the component receiving the reply.
	signature	Identifier of the signature of the procedure call.
	parameterList	A list of value parameters which are part of the indicated signature. The parameters in parameterList are ordered as they appear in the TTCN-3 signature declaration.
	returnValue	(Optional) return value of the procedure call.
<b>Return Value</b>	void	
<b>Constraint</b>	This operation shall be called by the CH at the local TE when at a remote TE a <i>provided</i> tciReplyConnected has been called. All <i>out</i> and <i>inout</i> procedure parameters and the return value contain values. All <i>in</i> procedure parameters shall contain the distinct value of null since they are only of relevance to the procedure call but not in the reply to the call. The parameterList contains procedure call parameters. These parameters are the parameters specified in the TTCN-3 signature template. If no return type has been defined for the procedure signature in the TTCN-3 ATS, the distinct value null shall be passed for the returnValue.	
<b>Effect</b>	The TE enqueues the reply at the local port queue of the indicated receiver component.	

### 7.3.3.1.4 tciEnqueueRaiseConnected

<b>Signature</b>	void tciEnqueueRaiseConnected (in TriPortIdType sender, in TriComponentIdType receiver, in TriSignatureIdType signature, in Value exception)	
<b>In Parameters</b>	sender	Identifier of the port sending the reply.
	receiver	Identifier of the component receiving the reply.
	signature	Identifier of the signature of the procedure call.
	exception	The exception.
<b>Return Value</b>	void	
<b>Constraint</b>	This operation shall be called by the CH at the local TE when at a remote TE a <i>provided</i> tciRaiseConnected has been called.	
<b>Effect</b>	The TE enqueues the exception at the local port queue of the indicated receiver component.	

### 7.3.3.1.5 tciCreateTestComponent

<b>Signature</b>	TriComponentIdType tciCreateTestComponent (in TciTestComponentKindType kind,  in Type componentType), in TString name, in Value hostId)	
<b>In Parameters</b>	kind	The kind of component that shall be created (any kind except of SYSTEM).
	componentType	Identifier of the TTCN-3 component type that shall be created.
	name	Name of the component that shall be created.
	hostId	Value identifying the remote TE where the component shall be deployed.
<b>Return Value</b>	A TriComponentIdType value for the created component.	
<b>Constraint</b>	This operation shall be called by the CH at the local TE when at a remote TE a <i>provided tciCreateTestComponentReq</i> has been called. componentType shall be set to the distinct value null if a test component of kind control shall be created. name shall be set to the distinct value null if no name is given in the TTCN-3 create statement. If a non-null hostId is given, this hostId should be used to identify the remote TE in which to call tciCreateTestComponent.	
<b>Effect</b>	The TE creates a TTCN-3 test component of the componentType and passes a TriComponentIdType reference back to the CH. The CH communicates the reference back to the remote TE.	

### 7.3.3.1.6 tciStartTestComponent

<b>Signature</b>	void tciStartTestComponent(in TriComponentIdType component, in TciBehaviourIdType behaviour, in TciParameterListType parameterList)	
<b>In Parameters</b>	component	Identifier of the component to be started. Refers to an identifier previously created by a call of tciCreateTestComponent.
	behaviour	Identifier of the behaviour to be started on the component.
	parameterList	A list of Values where each value defines a parameter from the parameter list as defined in the TTCN-3 function declaration of the function being started. The parameters in parameterList are ordered as they appear in the TTCN-3 signature of the test case. If no parameters have to be passed either the null value or an empty parameterList, i.e., a list of length zero shall be passed.
<b>Return Value</b>	void	
<b>Constraint</b>	This operation shall be called by the CH at the local TE when at a remote TE a <i>provided tciStartTestComponentReq</i> has been called.	
<b>Effect</b>	The TE shall start the indicated behaviour on the indicated component.	

### 7.3.3.1.7 tciStopTestComponent

<b>Signature</b>	void tciStopTestComponent(in TriComponentIdType component)	
<b>In Parameters</b>	component	Identifier of the component to be stopped.
<b>Return Value</b>	void	
<b>Constraint</b>	This operation shall be called by the CH at the local TE when at a remote TE a <i>provided</i> tciStopTestComponentReq has been called.	
<b>Effect</b>	The TE shall stop the indicated behaviour on the indicated component.	

### 7.3.3.1.8 tciConnect

<b>Signature</b>	void tciConnect (in TriPortIdType fromPort, in TriPortIdType toPort)	
<b>In Parameters</b>	fromPort	Identifier of the test component port to be connected from.
	toPort	Identifier of the test component port to be connected to.
<b>Return Value</b>	void	
<b>Constraint</b>	This operation shall be called by the CH at the local TE when at a remote TE a <i>provided</i> tciConnectReq has been called.	
<b>Effect</b>	The TE shall connect the indicated ports to one another.	

### 7.3.3.1.9 tciDisconnect

<b>Signature</b>	void tciDisconnect (in TriPortIdType fromPort, in TriPortIdType toPort)	
<b>In Parameters</b>	fromPort	Identifier of the test component port to be disconnected.
	toPort	Identifier of the test component port to be disconnected.
<b>Return Value</b>	void	
<b>Constraint</b>	This operation shall be called by the CH at the local TE when at a remote TE a <i>provided</i> tciDisconnectReq has been called.	
<b>Effect</b>	The TE shall disconnect the indicated ports.	

### 7.3.3.1.10 tciMap

<b>Signature</b>	void tciMap (in TriPortIdType fromPort, in TriPortIdType toPort)	
<b>In Parameters</b>	fromPort	Identifier of the test component port to be mapped from.
	toPort	Identifier of the test component port to be mapped to.
<b>Return Value</b>	void	
<b>Constraint</b>	This operation shall be called by the CH at the local TE when at a remote TE a <i>provided</i> tciMapReq has been called.	
<b>Effect</b>	The TE shall map the indicated ports to one another.	

### 7.3.3.1.11 tciMapParam

<b>Signature</b>	void tciMapParam (in TriPortIdType fromPort, in TriPortIdType toPort in TriParameterListType paramList)	
<b>In Parameters</b>	fromPort	Identifier of the test component port to be mapped from.
	toPort	Identifier of the test component port to be mapped to.
	paramList	Configuration parameter list.
<b>Return Value</b>	void	
<b>Constraint</b>	This operation shall be called by the CH at the local TE when at a remote TE a <i>provided</i> tciMapParamReq has been called.	
<b>Effect</b>	The TE shall map the indicated ports to one another.	

### 7.3.3.1.12 tciUnmap

<b>Signature</b>	void tciUnmap (in TriPortIdType fromPort, in TriPortIdType toPort)	
<b>In Parameters</b>	fromPort	Identifier of the test component port to be unmapped.
	toPort	Identifier of the test component port to be unmapped.
<b>Return Value</b>	void	
<b>Constraint</b>	This operation shall be called by the CH at the local TE when at a remote TE a <i>provided</i> tciUnmapReq has been called.	
<b>Effect</b>	The TE shall unmap the indicated ports.	

### 7.3.3.1.13 tciUnmapParam

<b>Signature</b>	void tciUnmapParam (in TriPortIdType fromPort, in TriPortIdType toPort in TriParameterListType paramList)	
<b>In Parameters</b>	fromPort	Identifier of the test component port to be unmapped.
	toPort	Identifier of the test component port to be unmapped.
	paramList	Configuration parameter.
<b>Return Value</b>	void	
<b>Constraint</b>	This operation shall be called by the CH at the local TE when at a remote TE a <i>provided</i> tciUnmapParamReq has been called.	
<b>Effect</b>	The TE shall unmap the indicated ports.	

### 7.3.3.1.14 tciTestComponentTerminated

<b>Signature</b>	void tciTestComponentTerminated (in TriComponentIdType component,  in VerdictValue verdict)	
<b>In Parameters</b>	component	Identifier of the component that has terminated.
	verdict	Verdict after termination of the component.
<b>Return Value</b>	void	
<b>Constraint</b>	This operation shall be called by the CH at the local TE when at a remote TE a <i>provided</i> tciTestComponentTerminatedReq has been called.	
<b>Effect</b>	The local TE is notified of the termination of the indicated test component on a remote TE. Because the out values of <i>inout</i> and <i>out</i> parameters of a function being executed on a test component have no effect on that test component ([1]), the tciTestComponentTerminated operation does not have a parameterList parameter.	

### 7.3.3.1.15 tciTestComponentRunning

<b>Signature</b>	TBoolean tciTestComponentRunning (in TriComponentIdType component)	
<b>In Parameters</b>	component	Identifier of the component to be checked for running.
<b>Return Value</b>	true if the indicated component is still executing a behaviour, false otherwise.	
<b>Constraint</b>	This operation shall be called by the CH at the local TE when at a remote TE a <i>provided</i> tciTestComponentRunningReq has been called.	
<b>Effect</b>	The local TE determines whether the indicated component is executing a test behaviour. If the component is executing a behaviour true will be returned. In any other case, e.g., test component has finished execution, or test component has not been started, etc., false will be returned. After the operation returns, the CH will communicate the value back to the remote TE.	

### 7.3.3.1.16 tciTestComponentDone

<b>Signature</b>	TBoolean tciTestComponentDone (in TriComponentIdType comp)	
<b>In Parameters</b>	comp	Identifier of the component to be checked for done.
<b>Return Value</b>	true if the indicated component has completed executing its behaviour, false otherwise.	
<b>Constraint</b>	This operation shall be called by the CH at the local TE when at a remote TE a <i>provided</i> tciTestComponentDoneReq has been called.	
<b>Effect</b>	The local TE determines whether the indicated component has completed executing its test behaviour. If the component has completed its behaviour true will be returned. In any other case, e.g., test component has not been started, or test component is still executing, false will be returned. After the operation returns, the CH will communicate the value back to the remote TE.	



### 7.3.3.1.17 tciGetMTC

<b>Signature</b>	<code>TriComponentIdType tciGetMTC()</code>
<b>Return Value</b>	A <code>TriComponentIdType</code> value of the MTC if the MTC executes on the local TE, the distinct value <code>null</code> otherwise.
<b>Constraint</b>	This operation can be called by the CH at the appropriate local TE when at a remote TE a <i>provided</i> <code>tciGetMTCReq</code> has been called.
<b>Effect</b>	The local TE determines whether the MTC is executing on the local TE. If the MTC executes on the local TE the component id of the MTC is being returned. If the MTC is not executed on the local TE the distinct value <code>null</code> will be returned. The operation will have no effect on the execution of the MTC. After the operation returns, the CH will communicate the value back to the remote TE.

### 7.3.3.1.18 tciExecuteTestCase

<b>Signature</b>	<code>void tciExecuteTestCase (in TciTestCaseIdType testCaseId, in TriPortIdListType tsiPortList)</code>	
<b>In Parameters</b>	<code>testCaseId</code>	A test case identifier as defined in the TTCN-3 module.
	<code>tsiPortList</code>	Contains all ports that have been declared in the definition of the system component for the test case, i.e., the TSI ports. If a system component has not been explicitly defined for the test case, then the <code>tsiPortList</code> contains all communication ports of the MTC. The ports in <code>tsiPortList</code> are ordered as they appear in the respective TTCN-3 component type declaration. If no ports have to be passed either the <code>null</code> value or an empty <code>tsiPortList</code> , i.e., a list of length zero shall be passed.
<b>Return Value</b>	<code>void</code>	
<b>Constraint</b>	This operation shall be called by the CH at the appropriate local TE when at a remote TE a <i>provided</i> <code>tciExecuteTestCaseReq</code> has been called.	
<b>Effect</b>	The local TE determines whether static connections to the SUT and the initialization of communication means for TSI ports should be done.	

### 7.3.3.1.19 tciReset

<b>Signature</b>	<code>void tciReset ()</code>
<b>Return Value</b>	<code>void</code>
<b>Constraint</b>	This operation shall be called by the CH at appropriate local TEs when at a remote TE a <i>provided</i> <code>tciResetReq</code> has been called.
<b>Effect</b>	The TE can decide to take any means to reset the test system locally.

### 7.3.3.1.20 tciKillTestComponent

<b>Signature</b>	<code>void tciKillTestComponent (in TriComponentIdType comp)</code>	
<b>In Parameters</b>	<code>comp</code>	Identifier of the component to be killed.
<b>Return Value</b>	<code>void</code>	
<b>Constraint</b>	This operation shall be called by the CH at the local TE when at a remote TE a <i>provided</i> <code>tciKillTestComponentReq</code> has been called.	
<b>Effect</b>	The TE stops the behaviour on the indicated component if necessary and transfers it into the killed state.	

### 7.3.3.1.21 tciTestComponentAlive

<b>Signature</b>	TBoolean tciTestComponentAlive (in TriComponentIdType comp)	
<b>In Parameters</b>	comp	Identifier of the component to be checked for being alive.
<b>Return Value</b>	true if the indicated component is alive, false otherwise.	
<b>Constraint</b>	This operation shall be called by the CH at the local TE when at a remote TE a <i>provided</i> tciTestComponentAliveReq has been called.	
<b>Effect</b>	The local TE determines whether the indicated component is alive. After the operation returns, the CH will communicate the value back to the remote TE.	

### 7.3.3.1.22 tciTestComponentKilled

<b>Signature</b>	TBoolean tciTestComponentKilled (in TriComponentIdType comp)	
<b>In Parameters</b>	comp	Identifier of the component to be checked for being killed.
<b>Return Value</b>	true if the indicated component has been killed, false otherwise.	
<b>Constraint</b>	This operation shall be called by the CH at the local TE when at a remote TE a <i>provided</i> tciTestComponentKilledReq has been called.	
<b>Effect</b>	The local TE determines whether the indicated component is in the killed state. If it is, true will be returned. In any other case, false will be returned. After the operation returns, the CH will communicate the value back to the remote TE.	

### 7.3.3.2 TCI-CH provided

This clause specifies the operations the CH shall provide to the TE.

#### 7.3.3.2.1 tciSendConnected

<b>Signature</b>	void tciSendConnected (in TriPortIdType sender, in TriComponentIdType receiver, in Value sendMessage)	
<b>In Parameters</b>	sender	Port identifier at the sending component via which the message is sent.
	receiver	Identifier of the receiving component.
	sendMessage	The message to be sent.
<b>Return Value</b>	void	
<b>Constraint</b>	This operation shall be called by the TE when it executes a TTCN-3 unicast send operation on a component port, which has been connected to another component port.	
<b>Effect</b>	Sends an asynchronous transmission only to the given receiver component. CH transmits the message to the remote TE on which receiver is being executed and enqueues the data in the remote TE.	

### 7.3.3.2.2 tciSendConnectedBC

<b>Signature</b>	void tciSendConnectedBC (in TriPortIdType sender, in Value sendMessage)	
<b>In Parameters</b>	sender	Port identifier at the sending component via which the message is sent.
	sendMessage	The message to be sent.
<b>Return Value</b>	void	
<b>Constraint</b>	This operation shall be called by the TE when it executes a TTCN-3 broadcast send operation on a component port, which has been connected to other component ports.	
<b>Effect</b>	Sends an asynchronous transmission to all components being connected to this port. CH transmits the message to all remote TEs on which receivers are being executed and enqueues the data in the remote TEs.	

### 7.3.3.2.3 tciSendConnectedMC

<b>Signature</b>	void tciSendConnectedMC (in TriPortIdType sender, in TriComponentIdListType receivers, in Value sendMessage)	
<b>In Parameters</b>	sender	Port identifier at the sending component via which the message is sent.
	receivers	Identifiers of the receiving components.
	sendMessage	The message to be sent.
<b>Return Value</b>	void	
<b>Constraint</b>	This operation shall be called by the TE when it executes a TTCN-3 multicast send operation on a component port, which has been connected to other component ports.	
<b>Effect</b>	Sends an asynchronous transmission to all given receiver components. CH transmits the message to all remote TEs on which receivers are being executed and enqueues the data in the remote TEs.	

### 7.3.3.2.4 tciCallConnected

<b>Signature</b>	void tciCallConnected (in TriPortIdType sender, in TriComponentIdType receiver, in TriSignatureIdType signature, in TciParameterListType parameterList)	
<b>In Parameters</b>	sender	Port identifier at the sending component via which the message is sent.
	receiver	Identifier of the receiving component.
	signature	Identifier of the signature of the procedure call.
	parameterList	A list of value parameters which are part of the indicated signature. The parameters in parameterList are ordered as they appear in the TTCN-3 signature declaration.
<b>Return Value</b>	void	

<b>Constraint</b>	This operation shall be called by the TE when it executes a TTCN-3 unicast call operation on a component port, which has been connected to another component port. All <i>in</i> and <i>inout</i> procedure parameters contain values. All <i>out</i> procedure parameters shall contain the distinct value of <code>null</code> because they are only of relevance in a reply to the procedure call but not in the procedure call itself. The procedure parameters are the parameters specified in the TTCN-3 signature template.
<b>Effect</b>	On invocation of this operation the TE can initiate the procedure call corresponding to the signature identifier <code>signature</code> at the called component receiver. The <code>tciCallConnected</code> operation shall return without waiting for the return of the issued procedure call. Note that an optional timeout value, which can be specified in the TTCN-3 ATS for a call operation, is not included in the <code>tciCallConnected</code> operation signature. The TE is responsible to address this issue by starting a timer for the TTCN-3 call operation in the PA with a separate TRI operation call, i.e., <code>triStartTimer</code> . CH transmits the call to the remote TE on which <code>receiver</code> is being executed and enqueues the call in the remote TE.

### 7.3.3.2.5 tciCallConnectedBC

<b>Signature</b>	void <code>tciCallConnectedBC</code> (in <code>TriPortIdType</code> <code>sender</code> , in <code>TriSignatureIdType</code> <code>signature</code> , in <code>TciParameterListType</code> <code>parameterList</code> )	
<b>In Parameters</b>	<code>sender</code>	Port identifier at the sending component via which the message is sent.
	<code>signature</code>	Identifier of the signature of the procedure call.
	<code>parameterList</code>	A list of value parameters which are part of the indicated signature. The parameters in <code>parameterList</code> are ordered as they appear in the TTCN-3 signature declaration.
<b>Return Value</b>	void	
<b>Constraint</b>	This operation shall be called by the TE when it executes a TTCN-3 broadcast call operation on a component port, which has been connected to other component ports. All <i>in</i> and <i>inout</i> procedure parameters contain values. All <i>out</i> procedure parameters shall contain the distinct value of <code>null</code> because they are only of relevance in a reply to the procedure call but not in the procedure call itself. The procedure parameters are the parameters specified in the TTCN-3 signature template.	
<b>Effect</b>	On invocation of this operation the TE can initiate the procedure call corresponding to the signature identifier <code>signature</code> at the called component receiver. The <code>tciCallConnected</code> operation shall return without waiting for the return of the issued procedure call. Note that an optional timeout value, which can be specified in the TTCN-3 ATS for a call operation, is not included in the <code>tciCallConnected</code> operation signature. The TE is responsible to address this issue by starting a timer for the TTCN-3 call operation in the PA with a separate TRI operation call, i.e., <code>triStartTimer</code> . CH transmits the call to all remote TEs on which a <code>receiver</code> is being executed and enqueues the call in the remote TEs.	

### 7.3.3.2.6 tciCallConnectedMC

<b>Signature</b>	void tciCallConnectedMC (in TriPortIdType sender, in TriComponentIdListType receivers, in TriSignatureIdType signature, in TciParameterListType parameterList)	
<b>In Parameters</b>	sender	Port identifier at the sending component via which the message is sent.
	receivers	Identifier of the receiving components.
	signature	Identifier of the signature of the procedure call.
	parameterList	A list of value parameters which are part of the indicated signature. The parameters in parameterList are ordered as they appear in the TTCN-3 signature declaration.
<b>Return Value</b>	void	
<b>Constraint</b>	This operation shall be called by the TE when it executes a TTCN-3 multicast call operation on a component port, which has been connected to other component ports. All <i>in</i> and <i>inout</i> procedure parameters contain values. All <i>out</i> procedure parameters shall contain the distinct value of null because they are only of relevance in a reply to the procedure call but not in the procedure call itself. The procedure parameters are the parameters specified in the TTCN-3 signature template.	
<b>Effect</b>	On invocation of this operation the TE can initiate the procedure call corresponding to the signature identifier signature at the called component receiver. The tciCallConnected operation shall return without waiting for the return of the issued procedure call. Note that an optional timeout value, which can be specified in the TTCN-3 ATS for a call operation, is not included in the tciCallConnected operation signature. The TE is responsible to address this issue by starting a timer for the TTCN-3 call operation in the PA with a separate TRI operation call, i.e., triStartTimer. CH transmits the call to all remote TEs on which a receiver is being executed and enqueues the call in the remote TEs.	

### 7.3.3.2.7 tciReplyConnected

<b>Signature</b>	void tciReplyConnected (in TriPortIdType sender, in TriComponentIdType receiver, in TriSignatureIdType signature, in TciParameterListType parameterList, in Value returnValue)	
<b>In Parameters</b>	sender	Identifier of the port sending the reply.
	receiver	Identifier of the component receiving the reply.
	signature	Identifier of the signature of the procedure call.
	parameterList	A list of encoded parameters which are part of the indicated signature. The parameters in parameterList are ordered as they appear in the TTCN-3 signature declaration.
	returnValue	(Optional) return value of the procedure call.
<b>Return Value</b>	void	
<b>Constraint</b>	This operation shall be called by the TE when it executes a TTCN-3 unicast reply operation on a component port which has been connected to another component port. All <i>out</i> and <i>inout</i> procedure parameters and the return value contain values. All <i>in</i> procedure parameters shall contain the distinct value of null since they are only of relevance to the procedure call but not in the reply to the call. The parameterList	

	contains procedure call parameters. These parameters are the parameters specified in the TTCN-3 signature template. If no return type has been defined for the procedure signature in the TTCN-3 ATS, the distinct value <code>null</code> shall be passed for the return value.
<b>Effect</b>	On invocation of this operation the CH can issue the reply to a procedure call corresponding to the signature identifier <code>signature</code> and component identifier <code>receiver</code> . CH transmits the reply to the remote TE on which <code>receiver</code> is being executed and enqueues the reply in the remote TE.

### 7.3.3.2.8 tciReplyConnectedBC

<b>Signature</b>	void tciReplyConnectedBC (in TriPortIdType sender, in TriSignatureIdType signature, in TciParameterListType parameterList, in Value returnValue)	
<b>In Parameters</b>	sender	Identifier of the port sending the reply.
	signature	Identifier of the signature of the procedure call.
	parameterList	A list of encoded parameters which are part of the indicated signature. The parameters in <code>parameterList</code> are ordered as they appear in the TTCN-3 signature declaration.
	returnValue	(Optional) return value of the procedure call.
<b>Return Value</b>	void	
<b>Constraint</b>	This operation shall be called by the TE when it executes a TTCN-3 broadcast reply operation on a component port which has been connected to other component ports. All <i>out</i> and <i>inout</i> procedure parameters and the return value contain values. All <i>in</i> procedure parameters shall contain the distinct value of <code>null</code> since they are only of relevance to the procedure call but not in the reply to the call. The <code>parameterList</code> contains procedure call parameters. These parameters are the parameters specified in the TTCN-3 signature template. If no return type has been defined for the procedure signature in the TTCN-3 ATS, the distinct value <code>null</code> shall be passed for the return value.	
<b>Effect</b>	On invocation of this operation the CH can issue the reply to a procedure call corresponding to the signature identifier <code>signature</code> and all components connected to <code>sender</code> . CH transmits the exception to all remote TEs on which receivers are being executed and enqueues the exception in the remote TEs.	

### 7.3.3.2.9 tciReplyConnectedMC

<b>Signature</b>	void tciReplyConnectedMC (in TriPortIdType sender, in TriComponentIdListType receivers, in TriSignatureIdType signature, in TciParameterListType parameterList, in Value returnValue)	
<b>In Parameters</b>	sender	Identifier of the port sending the reply.
	receivers	Identifier of the components receiving the reply.
	signature	Identifier of the signature of the procedure call.
	parameterList	A list of encoded parameters which are part of the indicated signature. The parameters in <code>parameterList</code> are ordered as they appear in the TTCN-3 signature declaration.
	returnValue	(Optional) return value of the procedure call.

<b>Return Value</b>	void
<b>Constraint</b>	This operation shall be called by the TE when it executes a TTCN-3 multicast reply operation on a component port which has been connected to other component ports. All <i>out</i> and <i>inout</i> procedure parameters and the return value contain values. All <i>in</i> procedure parameters shall contain the distinct value of <code>null</code> since they are only of relevance to the procedure call but not in the reply to the call. The <code>parameterList</code> contains procedure call parameters. These parameters are the parameters specified in the TTCN-3 signature template. If no return type has been defined for the procedure signature in the TTCN-3 ATS, the distinct value <code>null</code> shall be passed for the return value.
<b>Effect</b>	On invocation of this operation the CH can issue the reply to a procedure call corresponding to the signature identifier <code>signature</code> and one of the component identifier in <code>receivers</code> . CH transmits the reply to the remote TEs on which <code>receivers</code> are being executed and enqueues the reply in the remote TEs.

### 7.3.3.2.10 tciRaiseConnected

<b>Signature</b>	void tciRaiseConnected (in TriPortIdType sender, in TriComponentIdType receiver, in TriSignatureIdType signature, in Value exception)	
<b>In Parameters</b>	sender	Identifier of the port sending the reply.
	receiver	Identifier of the component receiving the reply.
	signature	Identifier of the signature of the procedure call.
	exception	The exception value.
<b>Return Value</b>	void	
<b>Constraint</b>	This operation shall be called by the TE when it executes a TTCN-3 unicast raise operation on a component port which has been connected to another component port.	
<b>Effect</b>	On invocation of this operation the CH can raise an exception to a procedure call corresponding to the signature identifier <code>signature</code> and component identifier <code>receiver</code> . CH transmits the exception to the remote TE on which <code>receiver</code> is being executed and enqueues the exception in the remote TE.	

### 7.3.3.2.11 tciRaiseConnectedBC

<b>Signature</b>	void tciRaiseConnectedBC (in TriPortIdType sender, in TriSignatureIdType signature, in Value exception)	
<b>In Parameters</b>	sender	Identifier of the port sending the reply.
	signature	Identifier of the signature of the procedure call.
	exception	The exception value.
<b>Return Value</b>	void	
<b>Constraint</b>	This operation shall be called by the TE when it executes a TTCN-3 broadcast raise operation on a component port which has been connected to other component ports.	
<b>Effect</b>	On invocation of this operation the CH can raise an exception to a procedure call corresponding to the signature identifier <code>signature</code> and all components connected to <code>sender</code> . CH transmits the exception to all remote TEs on which receivers are being executed and enqueues the exception in the remote TEs.	

### 7.3.3.2.12 tciRaiseConnectedMC

<b>Signature</b>	void tciRaiseConnectedMC (in TriPortIdType sender, in TriComponentIdListType receiver, in TriSignatureIdType signature, in Value exception)	
<b>In Parameters</b>	sender	Identifier of the port sending the reply.
	receivers	Identifiers of the component receiving the reply.
	signature	Identifier of the signature of the procedure call.
	exception	The exception value.
<b>Return Value</b>	void	
<b>Constraint</b>	This operation shall be called by the TE when it executes a TTCN-3 multicast raise operation on a component port which has been connected to another component port.	
<b>Effect</b>	On invocation of this operation the CH can raise an exception to a procedure call corresponding to the signature identifier <code>signature</code> and one of the component identifier <code>receivers</code> . CH transmits the exception to all remote TEs on which <code>receivers</code> are being executed and enqueues the exception in the remote TEs.	

### 7.3.3.2.13 tciCreateTestComponentReq

<b>Signature</b>	TriComponentIdType tciCreateTestComponentReq (in TciTestComponentKindType kind, in Type componentType, in TString name)	
<b>In Parameters</b>	kind	The kind of component that shall be created (any kind except of SYSTEM).
	componentType	Identifier of the TTCN-3 component type that shall be created.
<b>Return Value</b>	A TriComponentIdType value for the created component.	
<b>Constraint</b>	This operation shall be called from the TE when a component has to be created, either explicitly when the TTCN-3 create operation is called or implicitly when the master test component (MTC) or a control component has to be created. name shall be set to the distinct value null if no name is given in the TTCN-3 create statement.	
<b>Effect</b>	CH transmits the component creation request to the remote TE and calls there the tciCreateTestComponent operation to obtain a component identifier for this component.	

### 7.3.3.2.14 tciStartTestComponentReq

<b>Signature</b>	void tciStartTestComponentReq(in TriComponentIdType component, in TciBehaviourIdType behaviour, in TciParameterListType parameterList)	
<b>In Parameters</b>	component	Identifier of the component to be started.
	behaviour	Identifier of the behaviour to be started on the component.
	parameterList	A list of Values where each value defines a parameter from the parameter list as defined in the TTCN-3 function declaration of the function being started. The parameters in parameterList are ordered as they appear in the TTCN-3 signature of the test case. If no parameters have to be passed either the null value or an empty parameterList, i.e., a list of length zero shall be passed.



<b>Return Value</b>	void
<b>Constraint</b>	This operation shall be called by the TE when it executes the TTCN-3 start operation.
<b>Effect</b>	CH transmits the start component request to the remote TE and calls there the <code>tciStartTestComponent</code> operation.

### 7.3.3.2.15 `tciStopTestComponentReq`

<b>Signature</b>	void <code>tciStopTestComponentReq</code> (in <code>TriComponentIdType</code> component)	
<b>In Parameters</b>	component	Identifier of the component to be stopped.
<b>Return Value</b>	void	
<b>Constraint</b>	This operation shall be called by the TE when it executes the TTCN-3 stop operation.	
<b>Effect</b>	CH transmits the stop component request to the remote TE and calls there the <code>tciStopTestComponent</code> operation.	

### 7.3.3.2.16 `tciConnectReq`

<b>Signature</b>	void <code>tciConnectReq</code> (in <code>TriPortIdType</code> fromPort, in <code>TriPortIdType</code> toPort)	
<b>In Parameters</b>	fromPort	Identifier of the test component port to be connected from.
	toPort	Identifier of the test component port to be connected to.
<b>Return Value</b>	void	
<b>Constraint</b>	This operation shall be called by the TE when it executes a TTCN-3 connect operation.	
<b>Effect</b>	CH transmits the connection request to the remote TE where it calls the <code>tciConnect</code> operation to establish a logical connection between the two indicated ports. Note that both ports can be on remote TEs. In this case, the operation returns only after calling the <code>tciConnect</code> operation on both remote TEs.	

### 7.3.3.2.17 `tciDisconnectReq`

<b>Signature</b>	void <code>tciDisconnectReq</code> (in <code>TriPortIdType</code> fromPort, in <code>TriPortIdType</code> toPort)	
<b>In Parameters</b>	fromPort	Identifier of the test component port to be disconnected.
	toPort	Identifier of the test component port to be disconnected.
<b>Return Value</b>	void	
<b>Constraint</b>	This operation shall be called by the TE when it executes a TTCN-3 disconnect operation.	
<b>Effect</b>	CH transmits the disconnect request to the remote TE where it calls the <code>tciDisconnect</code> operation to tear down the logical connection between the two indicated ports. Note that both ports can be on remote TEs. In this case, the operation returns only after calling the <code>tciDisconnect</code> operation on both remote TEs.	

### 7.3.3.2.18 `tciMapReq`

<b>Signature</b>	void <code>tciMapReq</code> (in <code>TriPortIdType</code> fromPort, in <code>TriPortIdType</code> toPort)	
<b>In Parameters</b>	fromPort	Identifier of the test component port to be mapped from.
	toPort	Identifier of the test component port to be mapped to.
<b>Return Value</b>	void	

<b>Constraint</b>	This operation shall be called by the TE when it executes a TTCN-3 map operation.
<b>Effect</b>	CH transmits the map request to the remote TE where it calls the <code>tciParam</code> operation to establish a logical connection between the two indicated ports.

### 7.3.3.2.19 tciParamReq

<b>Signature</b>	void tciParamReq (in TriPortIdType fromPort, in TriPortIdType toPort, in TriParameterListType paramList)	
<b>In Parameters</b>	fromPort	Identifier of the test component port to be mapped from.
	toPort	Identifier of the test component port to be mapped to.
	paramList	Configuration parameter list.
<b>Return Value</b>	void	
<b>Constraint</b>	This operation shall be called by the TE when it executes a TTCN-3 map operation including parameters.	
<b>Effect</b>	CH transmits the map request to the remote TE where it calls the <code>tciParam</code> operation to establish a logical connection between the two indicated ports.	

### 7.3.3.2.20 tciParamReq

<b>Signature</b>	void tciParamReq (in TriPortIdType fromPort, in TriPortIdType toPort)	
<b>In Parameters</b>	fromPort	Identifier of the test component port to be unmapped.
	toPort	Identifier of the test component port to be unmapped.
<b>Return Value</b>	void	
<b>Constraint</b>	This operation shall be called by the TE when it executes a TTCN-3 unmap operation.	
<b>Effect</b>	CH transmits the unmap request to the remote TE where it calls the <code>tciParam</code> operation to tear down the logical connection between the two indicated ports.	

### 7.3.3.2.21 tciParamReq

<b>Signature</b>	void tciParamReq (in TriPortIdType fromPort, in TriPortIdType toPort, in TriParameterListType paramList)	
<b>In Parameters</b>	fromPort	Identifier of the test component port to be unmapped.
	toPort	Identifier of the test component port to be unmapped.
	paramList	Configuration parameter list.
<b>Return Value</b>	void	
<b>Constraint</b>	This operation shall be called by the TE when it executes a TTCN-3 unmap operation including parameters.	
<b>Effect</b>	CH transmits the unmap request to the remote TE where it calls the <code>tciParam</code> operation to teardown the connection between the two indicated ports.	

### 7.3.3.2.22 tciTestComponentTerminatedReq

<b>Signature</b>	void tciTestComponentTerminatedReq (in TriComponentIdType component,  in VerdictValue verdict)	
<b>In Parameters</b>	component	Identifier of the component that has terminated.
	verdict	Verdict after termination of the component.
<b>Return Value</b>	void	
<b>Constraint</b>	This operation shall be called by the TE when a test component terminates execution, either explicitly with the TTNC-3 stop operation or implicitly, if it has reached the last statement.	
<b>Effect</b>	The CH is notified of the termination of the indicated test component. Because the out values of <i>inout</i> and <i>out</i> parameters of a function being executed on a test component have no effect on that test component ([1]), the tciTestComponentTerminateReq operation does not have a parameterList parameter. CH communicates the termination of the indicated component to all participating TEs and to the special TE*, which keeps track of the overall verdict.	

### 7.3.3.2.23 tciTestComponentRunningReq

<b>Signature</b>	TBoolean tciTestComponentRunningReq (in TriComponentIdType component)	
<b>In Parameters</b>	component	Identifier of the component to be checked for running.
<b>Return Value</b>	true if the indicated component is still executing a behaviour, false otherwise.	
<b>Constraint</b>	This operation shall be called by the TE when it executes a TTCN-3 running operation.	
<b>Effect</b>	CH transmits the running request to the remote TE having the test component to be checked, where it calls the tciTestComponentRunning operation to check the execution status of the indicated test component.	

### 7.3.3.2.24 tciTestComponentDoneReq

<b>Signature</b>	TBoolean tciTestComponentDoneReq (in TriComponentIdType comp)	
<b>In Parameters</b>	comp	Identifier of the component to be checked for done.
<b>Return Value</b>	true if the indicated component has completed executing its behaviour, false otherwise.	
<b>Constraint</b>	This operation shall be called by the TE when it executes a TTCN-3 done operation.	
<b>Effect</b>	CH transmits the done request to the remote TE having the test component to be checked, where it calls the tciTestComponentDone operation to check the status of the indicated test component.	

### 7.3.3.2.25 tciGetMTCReq

<b>Signature</b>	TriComponentIdType tciGetMTCReq()	
<b>Return Value</b>	A TriComponentIdType value of the MTC.	
<b>Constraint</b>	This operation shall be called by the TE when it executes a TTCN-3 mtc operation.	
<b>Effect</b>	The CH determines the component id of the MTC.	

### 7.3.3.2.26 tciExecuteTestCaseReq

<b>Signature</b>	void tciExecuteTestCaseReq (in TciTestCaseIdType testCaseId, in TriPortIdListType tsiPortList)	
<b>In Parameters</b>	testCaseId	A test case identifier as defined in the TTCN-3 module.
	tsiPortList	tsiPortList contains all ports that have been declared in the definition of the system component for the test case, i.e., the TSI ports. If a system component has not been explicitly defined for the test case, then the tsiPortList contains all communication ports of the MTC. The ports in tsiPortList are ordered as they appear in the respective TTCN-3 component type declaration.  If no ports have to be passed either the null value or an empty tsiPortList, i.e., a list of length zero shall be passed.
<b>Return Value</b>	void	
<b>Constraint</b>	This operation can be called by the TE immediately before it starts the test case behaviour on the MTC (in course of a TTCN-3 execute operation).	
<b>Effect</b>	CH transmits the execute test case request to the remote TEs having system ports of the indicated test case. Static connections to the SUT and the initialization of communication means for TSI ports can be set up.	

### 7.3.3.2.27 tciResetReq

<b>Signature</b>	void tciResetReq ()	
<b>Return Value</b>	void	
<b>Constraint</b>	This operation can be called by the TE at any time to reset the test system.	
<b>Effect</b>	CH transmits the reset request to all involved TEs.	

### 7.3.3.2.28 tciKillTestComponentReq

<b>Signature</b>	void tciKillTestComponentReq (in TriComponentIdType comp)	
<b>In Parameters</b>	comp	Identifier of the component to be killed.
<b>Return Value</b>	void	
<b>Constraint</b>	This operation shall be called by the TE when it executes the TTCN-3 kill operation.	
<b>Effect</b>	CH transmits the kill component request to the remote TE and calls there the tciKillTestComponent operation.	

### 7.3.3.2.29 tciTestComponentAliveReq

<b>Signature</b>	TBoolean tciTestComponentAliveReq (in TriComponentIdType comp)	
<b>In Parameters</b>	comp	Identifier of the component to be checked for being alive.
<b>Return Value</b>	true if the indicated component is alive, false otherwise.	
<b>Constraint</b>	This operation shall be called by the TE when it executes the TTCN-3 alive operation.	
<b>Effect</b>	CH transmits the request to the remote TE that created the test component in question, where it calls the tciTestComponentAlive operation to check the status of the indicated test component.	

### 7.3.3.2.30 tciTestComponentKilledReq

<b>Signature</b>	TBoolean tciTestComponentKilledReq (in TriComponentIdType comp)	
<b>In Parameters</b>	comp	Identifier of the component to be checked for being killed.
<b>Return Value</b>	true if the indicated component has been killed, false otherwise.	
<b>Constraint</b>	This operation shall be called by the TE when it executes the TTCN-3 killed operation.	
<b>Effect</b>	CH transmits the request to the remote TE that created the test component in question, where it calls the tciTestComponentKilled operation to check the status of the indicated test component.	

### 7.3.4 The TCI-TL interface

The TCI test logging interface (TCI-TL) describes the operations a TTCN-3 executable is required to implement and the operations a test logging implementation shall provide to the TE (Figure 8).

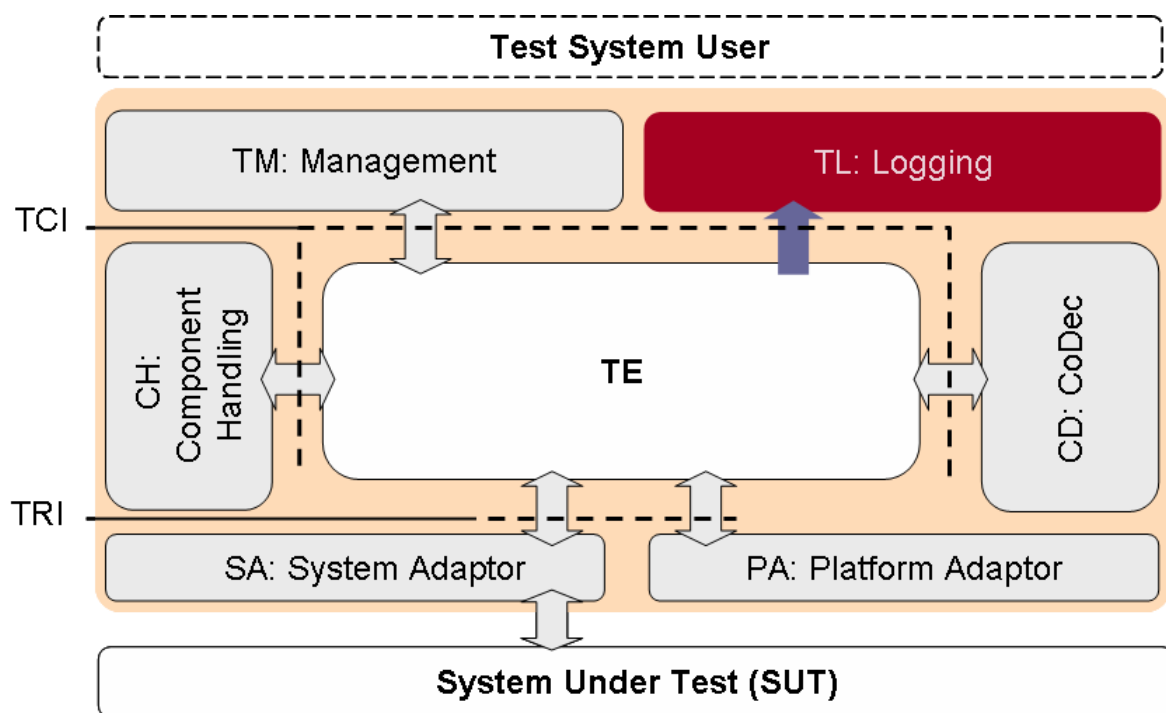


Figure 8 – The TCI-TL interface

The logging provides for all TTCN-3 level operations an operation to log the respective event being performed by the TE, the SA, the PA, the CH or the CD to the user.

#### 7.3.4.1 TCI-TL provided

This clause specifies the operations the TL shall provide to the TE.

### 7.3.4.1.1 tliTcExecute

<b>Signature</b>	void tliTcExecute(in TString am, in TInteger ts, in TString src, in TInteger line, in TriComponentIdType c, in TciTestCaseIdType tcId, in TciParameterListType triPars, in TriTimerDurationType dur)	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	tcId	The testcase to be executed.
	triPars	The list of parameters required by the testcase.
	dur	Duration of the execution.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by TE to log the execute test case request.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

### 7.3.4.1.2 tliTcStart

<b>Signature</b>	void tliTcStart(in TString am, in TInteger ts, in TString src, in TInteger line, in TriComponentIdType c, in TciTestCaseIdType tcId, in TciParameterListType tciPars, in TriTimerDurationType dur)	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	tcId	The testcase to be executed.
	tciPars	The list of parameters required by the testcase.
	dur	Duration of the execution.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by TE to log the start of a testcase. This event occurs before the testcase is started.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

### 7.3.4.1.3 tliTcStop

<b>Signature</b>	void tliTcStop(in TString am, in TInteger ts, in TString src, in TInteger line, in TriComponentIdType c, in TString reason)	
<b>In Parameters</b>	Am	An additional message.
	Ts	The time when the event is produced.
	Src	The source file of the test specification.
	Line	The line number where the request is performed.
	C	The component which produces this event.
	reason	The optional reason of the setverdict statement
<b>Return Value</b>	Void	
<b>Constraint</b>	Shall be called by TE to log the stop of a testcase.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

### 7.3.4.1.4 tliTcStarted

<b>Signature</b>	void tliTcStarted(in TString am, in TInteger ts, in TString src, in TInteger line, in TriComponentIdType c, in TciTestCaseIdType tcId, in TciParameterListType tciPars, in TriTimerDurationType dur)	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	tcId	The testcase to be executed.
	tciPars	The list of parameters required by the testcase.
	dur	Duration of the execution.
<b>Return Value</b>	Void	
<b>Constraint</b>	Shall be called by TM or TE to log the start of a testcase. This event occurs after the testcase was started.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

### 7.3.4.1.5 tliTcTerminated

<b>Signature</b>	void tliTcTerminated(in TString am, in TInteger ts, in TString src, in TInteger line, in TriComponentIdType c, in TciTestCaseIdType tcId, in TciParameterListType tciPars, in VerdictValue verdict, in TString reason)	
------------------	--	--

<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	tcId	The testcase to be executed.
	tciPars	The list of parameters required by the testcase.
	verdict	The verdict of the testcase.
	reason	The optional reason of the setverdict statement
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by TM or TE to log the termination of a testcase. This event occurs after the testcase terminated.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

#### 7.3.4.1.6 tliCtrlStart

<b>Signature</b>	void tliCtrlStart(in TString am, in TInteger ts, in TString src, in TInteger line, in TriComponentIdType c)	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by TE to log the start of the control part. This event occurs before the control is started. If the control is not represented by a TRI component, c is null.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

#### 7.3.4.1.7 tliCtrlStop

<b>Signature</b>	void tliCtrlStop(in TString am, in TInteger ts, in TString src, in TInteger line, in TriComponentIdType c)	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by TE to log the stop of the control part. This event occurs before the control is stopped. If the control is not represented by a TRI component, c is null.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	



### 7.3.4.1.8 tliCtrlTerminated

<b>Signature</b>	void tliCtrlTerminated (in TString am, in TInteger ts, in TString src, in TInteger line, in TriComponentIdType c)	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by TM or TE to log the termination of the control part. This event occurs after the control has terminated. If the control is not represented by a TRI component, c is null.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

### 7.3.4.1.9 tliMSend\_m

<b>Signature</b>	void tliMSend_m(in TString am, in TInteger ts, in TString src, in TInteger line, in TriComponentIdType c, in TriPortIdType at, in TriPortIdType to, in Value msgValue, in Value addrValue, in TciStatusType encoderFailure, in TriMessageType msg, in TriAddressType address, in TriStatusType transmissionFailure)	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	at	The port via which the message is sent.
	to	The port to which the message is sent.
	msgValue	The value to be encoded and sent.
	addrValue	The address value of the destination within the SUT.
	encoderFailure	The failure message which might occur at encoding.
	msg	The encoded message.
	address	The address of the destination within the SUT.
	transmissionFailure	The failure message which might occur at transmission.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by SA or TE to log a unicast send operation. This event occurs after sending. This event is used for logging the communication with the SUT.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

### 7.3.4.1.10 tliMSend\_m\_BC

<b>Signature</b>	<pre>void tliMSend_m_BC(in TString am, in TInteger ts,                   in TString src, in TInteger line,                   in TriComponentIdType c,                   in TriPortIdType at, in TriPortIdType to,                   in Value msgValue,                   in TciStatusType encoderFailure,                   in TriMessageType msg,                   in TriStatusType transmissionFailure)</pre>	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	at	The port via which the message is sent.
	to	The port to which the message is sent.
	msgValue	The value to be encoded and sent.
	encoderFailure	The failure message which might occur at encoding.
	msg	The encoded message.
	transmissionFailure	The failure message which might occur at transmission.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by SA or TE to log a broadcast send operation. This event occurs after sending. This event is used for logging the communication with the SUT.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

### 7.3.4.1.11 tliMSend\_m\_MC

<b>Signature</b>	<pre>void tliMSend_m_MC(in TString am, in TInteger ts,                   in TString src, in TInteger line,                   in TriComponentIdType c,                   in TriPortIdType at, in TriPortIdType to,                   in Value msgValue,                   in TciValueList addrValues,                   in TciStatusType encoderFailure,                   in TriMessageType msg,                   in TriAddressListType addresses,                   in TriStatusType transmissionFailure)</pre>	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	at	The port via which the message is sent.
	to	The port to which the message is sent.
	msgValue	The value to be encoded and sent.

	addrValues	The address values of the destinations within the SUT.
	encoderFailure	The failure message which might occur at encoding.
	msg	The encoded message.
	addresses	The addresses of the destinations within the SUT.
	transmissionFailure	The failure message which might occur at transmission.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by SA or TE to log a multicast send operation. This event occurs after sending. This event is used for logging the communication with the SUT.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

#### 7.3.4.1.12 tliMSend\_c

<b>Signature</b>	<pre>void tliMSend_c(in TString am, in TInteger ts,                in TString src, in TInteger line,                in TriComponentIdType c,                in TriPortIdType at, in TriPortIdType to,                in Value msgValue,                in TriStatusType transmissionFailure)</pre>	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	at	The port via which the message is sent.
	msgValue	The value to be encoded and sent.
	to	The component which will receive the message.
	transmissionFailure	The failure message which might occur at transmission.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by CH or TE to log a unicast send operation. This event occurs after sending. This event is used for logging the intercomponent communication.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

#### 7.3.4.1.13 tliMSend\_c\_BC

<b>Signature</b>	<pre>void tliMSend_c_BC(in TString am, in TInteger ts,                   in TString src, in TInteger line,                   in TriComponentIdType c,                   in TriPortIdType at, in TriPortIdListType to,                   in Value msgValue,                   in TriStatusType transmissionFailure)</pre>	
------------------	--	--

<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	at	The port via which the message is sent.
	to	The ports to which the message is sent.
	msgValue	The value to be encoded and sent.
	transmissionFailure	The failure message which might occur at transmission.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by CH or TE to log a broadcast send operation. This event occurs after sending. This event is used for logging the intercomponent communication.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

#### 7.3.4.1.14 tliMSend\_c\_MC

<b>Signature</b>	<pre>void tliMSend_c_MC(in TString am, in TInteger ts,                   in TString src, in TInteger line,                   in TriComponentIdType c,                   in TriPortIdType at, in TriPortIdListType to,                   in Value msgValue,                   in TriStatusType transmissionFailure)</pre>	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	at	The port via which the message is sent.
	to	The port to which the message is sent.
	msgValue	The value to be encoded and sent.
	transmissionFailure	The failure message which might occur at transmission.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by CH or TE to log a multicast send operation. This event occurs after sending. This event is used for logging the intercomponent communication.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

#### 7.3.4.1.15 tliMDetected\_m

<b>Signature</b>	<pre>void tliMDetected_m(in TString am, in TInteger ts,                    in TString src, in TInteger line,                    in TriComponentIdType c,                    in TriPortIdType at, in TriPortIdType from,                    in TriMessageType msg,                    in TriAddressType address)</pre>	
------------------	---	--

<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	at	The port via which the message is received.
	from	The port from which the message has been sent.
	msg	The received encoded message.
	address	The address of the source within the SUT.
<b>Return Value</b>	Void	
<b>Constraint</b>	Shall be called by SA or TE to log the enqueueing of a message. This event occurs after the message is enqueued. This event is used for logging the communication with the SUT.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

#### 7.3.4.1.16 tliMDetected\_c

<b>Signature</b>	<pre>void tliMDetected_c(in TString am, in TInteger ts,                    in TString src, in TInteger line,                    in TriComponentIdType c,                    in TriPortIdType at, in TriPortIdType from,                    in Value msgValue)</pre>	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	at	The port via which the message is received.
	from	The port from which the message has been sent.
	msgValue	The received message.
<b>Return Value</b>	Void	
<b>Constraint</b>	Shall be called by CH or TE to log the enqueueing of a message. This event occurs after the message is enqueued. This event is used for logging the intercomponent communication.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

#### 7.3.4.1.17 tliMMismatch\_m

<b>Signature</b>	<pre>void tliMMismatch_m(in TString am, in TInteger ts,                     in TString src, in TInteger line,                     in TriComponentIdType c,                     in TriPortIdType at, in Value msgValue,                     in TciValueTemplate msgTpl,                     in TciValueDifferenceList diffs,                     in Value addrValue,                     in TciValueTemplate addressTpl)</pre>	
------------------	---	--

<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	at	The port via which the message is received.
	msgValue	The message which is checked against the template.
	msgTmpl	The template used to check the message match.
	diffs	The difference/the mismatch between message and template
	addrValue	The address value of the source within the SUT.
	addressTmpl	The expected address of the source within the SUT.
<b>Return Value</b>	Void	
<b>Constraint</b>	Shall be called by TE to log the mismatch of a template. This event occurs after checking a template match. This event is used for logging the communication with the SUT.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

#### 7.3.4.1.18 tliMMismatch\_c

<b>Signature</b>	<pre>void tliMMismatch_c(in TString am, in TInteger ts,                     in TString src, in TInteger line,                     in TriComponentIdType c,                     in TriPortIdType at, in Value msgValue,                     in TciValueTemplate msgTmpl,                     in TciValueDifferenceList diffs,                     in TriComponentIdType from,                     in TciNonValueTemplate fromTmpl)</pre>	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	at	The port via which the message is received.
	msgValue	The message which is checked against the template.
	msgTmpl	The template used to check the message match.
	diffs	The difference/the mismatch between message and template
	from	The component which sent the message.
	fromTmpl	The expected sender component.
<b>Return Value</b>	Void	
<b>Constraint</b>	Shall be called by TE to log the mismatch of a template. This event occurs after checking a template match. This event is used for logging the intercomponent communication.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

### 7.3.4.1.19 tliMReceive\_m

<b>Signature</b>	<pre>void tliMReceive_m(in TString am, in TInteger ts,                   in TString src, in TInteger line,                   in TriComponentIdType c,                   in TriPortIdType at,                   in Value msgValue,                   in TciValueTemplate msgTmpl,                   in Value addrValue,                   in TciValueTemplate addressTmpl)</pre>	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	at	The port via which the message is received.
	msgValue	The message which is checked against the template.
	msgTmpl	The template used to check the message match.
	addrValue	The address value of the source within the SUT.
	addressTmpl	The expected address of the source within the SUT.
<b>Return Value</b>	Void	
<b>Constraint</b>	Shall be called by TE to log the receiving of a message. This event occurs after checking a template match. This event is used for logging the communication with SUT.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

### 7.3.4.1.20 tliMReceive\_c

<b>Signature</b>	<pre>void tliMReceive_c(in TString am, in TInteger ts,                   in TString src, in TInteger line,                   in TriComponentIdType c, in TriPortIdType at,                   in Value msgValue,                   in TciValueTemplate msgTmpl,                   in TriComponentIdType from,                   in TciNonValueTemplate fromTmpl)</pre>	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	at	The port via which the message is received.
	msg	The message which is checked against the template.
	msgTmpl	The template used to check the message match.
	from	The component which sent the message.
	fromTmpl	The expected sender component.
<b>Return Value</b>	Void	

<b>Constraint</b>	Shall be called by TE to log the receive of a message. This event occurs after checking a template match. This event is used for logging the intercomponent communication.
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.

#### 7.3.4.1.21 tliPrCall\_m

<b>Signature</b>	<pre>void tliPrCall_m(in TString am, in TInteger ts, in TString src,                 in TInteger line, in TriComponentIdType c,                 in TriPortIdType at, in TriPortIdType to,                 in TriSignatureIdType signature,                 in TciParameterListType tciPars,                 in Value addrValue,                 in TciStatusType encoderFailure,                 in TriParameterListType triPars,                 in TriAddressType address,                 in TriStatusType transmissionFailure)</pre>	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	at	The port via which the call is invoked.
	to	The port to which the call is sent.
	signature	The signature of the called operation.
	tciPars	The parameters of the called operation.
	addrValue	The address value of the destination within the SUT.
	encoderFailure	The failure message which might occur at encoding.
	triPars	The encoded parameters.
	address	The address of the destination within the SUT.
transmissionFailure	The failure message which might occur at transmission.	
<b>Return Value</b>	Void	
<b>Constraint</b>	Shall be called by SA or TE to log a unicast call operation. This event occurs after call execution. This event is used for logging the communication with the SUT.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

#### 7.3.4.1.22 tliPrCall\_m\_BC

<b>Signature</b>	<pre>void tliPrCall_m_BC(in TString am, in TInteger ts,                    in TString src,                    in TInteger line, in TriComponentIdType c,                    in TriPortIdType at, in TriPortIdType to,                    in TriSignatureIdType signature,                    in TciParameterListType tciPars,                    in TciStatusType encoderFailure,                    in TriParameterListType triPars,                    in TriStatusType transmissionFailure)</pre>	
------------------	--	--



<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	at	The port via which the call is invoked.
	to	The port to which the call is sent.
	signature	The signature of the called operation.
	tciPars	The parameters of the called operation.
	encoderFailure	The failure message which might occur at encoding.
	triPars	The encoded parameters.
	transmissionFailure	The failure message which might occur at transmission.
<b>Return Value</b>	Void	
<b>Constraint</b>	Shall be called by SA or TE to log a broadcast call operation. This event occurs after call execution. This event is used for logging the communication with the SUT.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

#### 7.3.4.1.23 tliPrCall\_m\_MC

<b>Signature</b>	<pre>void tliPrCall_m_MC(in TString am, in TInteger ts,                     in TString src,                     in TInteger line, in TriComponentIdType c,                     in TriPortIdType at, in TriPortIdType to,                     in TriSignatureIdType signature,                     in TciParameterListType tciPars,                     in TciValueList addrValues,                     in TciStatusType encoderFailure,                     in TriParameterListType triPars,                     in TriAddressListType addresses,                     in TriStatusType transmissionFailure)</pre>	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	at	The port via which the call is invoked.
	to	The port to which the call is sent.
	signature	The signature of the called operation.
	tciPars	The parameters of the called operation.
	addrValues	The address values of the destinations within the SUT.
	encoderFailure	The failure message which might occur at encoding.
	triPars	The encoded parameters.
	addresses	The addresses of the destinations within the SUT.
	transmissionFailure	The failure message which might occur at transmission.

<b>Return Value</b>	Void
<b>Constraint</b>	Shall be called by SA or TE to log a multicast call operation. This event occurs after call execution. This event is used for logging the communication with the SUT.
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.

#### 7.3.4.1.24 tliPrCall\_c

<b>Signature</b>	<pre>void tliPrCall_c(in TString am, in TInteger ts, in TString src,                 in TInteger line, in TriComponentIdType c,                 in TriPortIdType at, in TriPortIdType to,                 in TriSignatureIdType signature,                 in TciParameterListType tciPars,                 in TriStatusType transmissionFailure)</pre>	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	at	The port via which the call is invoked.
	to	The port to which the call is sent.
	signature	The signature of the called operation.
	tciPars	The parameters of the called operation.
transmissionFailure	The failure message which might occur at transmission.	
<b>Return Value</b>	Void	
<b>Constraint</b>	Shall be called by CH or TE to log a unicast call operation. This event occurs after call execution. This event is used for logging the intercomponent communication.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

#### 7.3.4.1.25 tliPrCall\_c\_BC

<b>Signature</b>	<pre>void tliPrCall_c_BC(in TString am, in TInteger ts,                    in TString src,                    in TInteger line, in TriComponentIdType c,                    in TriPortIdType at, in TriPortIdListType to,                    in TriSignatureIdType signature,                    in TciParameterListType tciPars,                    in TriStatusType transmissionFailure)</pre>
------------------	--

<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	at	The port via which the call is invoked.
	to	The port list to which the call is sent.
	signature	The signature of the called operation.
	tciPars	The parameters of the called operation.
	transmissionFailure	The failure message which might occur at transmission.
<b>Return Value</b>	Void	
<b>Constraint</b>	Shall be called by CH or TE to log a broadcast call operation. This event occurs after call execution. This event is used for logging the intercomponent communication.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

#### 7.3.4.1.26 tliPrCall\_c\_MC

<b>Signature</b>	<pre>void tliPrCall_c_MC(in TString am, in TInteger ts,                   in TString src,                   in TInteger line, in TriComponentIdType c,                   in TriPortIdType at,                   in TriPortIdListType to,                   in TriSignatureIdType signature,                   in TciParameterListType tciPars,                   in TriStatusType transmissionFailure)</pre>	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	at	The port via which the call is invoked.
	to	The port list to which the call is sent.
	signature	The signature of the called operation.
	tciPars	The parameters of the called operation.
	transmissionFailure	The failure message which might occur at transmission.
<b>Return Value</b>	Void	
<b>Constraint</b>	Shall be called by CH or TE to log a multicast call operation. This event occurs after call execution. This event is used for logging the intercomponent communication.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

### 7.3.4.1.27 tliPrGetCallDetected\_m

<b>Signature</b>	<pre>void tliPrGetCallDetected_m(in TString am, in TInteger ts,                              in TString src,                              in TInteger line,                              in TriComponentIdType c,                              in TriPortIdType at,                              in TriPortIdType from,                              in TriSignatureIdType signature,                              in TriParameterListType triPars,                              in TriAddressType address)</pre>	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	at	The port via which the call is received.
	from	The port from which the call has been sent.
	signature	The signature of the detected call.
	triPars	The encoded parameters of detected call.
	address	The address of the destination within the SUT.
<b>Return Value</b>	Void	
<b>Constraint</b>	Shall be called by SA or TE to log the getcall enqueue operation. This event occurs after call is enqueued. This event is used for logging the communication with the SUT.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

### 7.3.4.1.28 tliPrGetCallDetected\_c

<b>Signature</b>	<pre>void tliPrGetCallDetected_c(in TString am, in TInteger ts,                              in TString src,                              in TInteger line,                              in TriComponentIdType c,                              in TriPortIdType at,                              in TriPortIdType from,                              in TriSignatureIdType signature,                              in TciParameterListType tciPars)</pre>	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	at	The port via which the call is received.
	from	The port from which the call has been sent.
	signature	The signature of the called operation.
	tciPars	The encoded parameters of detected call.
<b>Return Value</b>	Void	

<b>Constraint</b>	Shall be called by CH or TE to log the getcall enqueue operation. This event occurs after call is enqueued. This event is used for logging the intercomponent communication.
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.

#### 7.3.4.1.29 tliPrGetCallMismatch\_m

<b>Signature</b>	<pre>void tliPrGetCallMismatch_m(in TString am, in TInteger ts,                            in TString src,                            in TInteger line,                            in TriComponentIdType c,                            in TriPortIdType at,                            in TriSignatureIdType signature,                            in TciParameterListType tciPars,                            in TciValueTemplate parsTpl,                            in TciValueDifferenceList diffs,                            in Value addrValue,                            in TciValueTemplate addressTpl)</pre>	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	at	The port via which the call is received.
	signature	The signature of the detected call.
	tciPars	The parameters of detected call.
	parsTpl	The template used to check the parameter match.
	diffs	The difference/the mismatch between call and template
	addrValue	The address value of the source within the SUT.
	addressTpl	The expected address of the source within the SUT.
<b>Return Value</b>	Void	
<b>Constraint</b>	Shall be called by TE to log the mismatch of a getcall. This event occurs after getcall is checked against a template. This event is used for logging the communication with the SUT.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

### 7.3.4.1.30 tliPrGetCallMismatch\_c

<b>Signature</b>	<pre>void tliPrGetCallMismatch_c(in TString am, in TInteger ts,                            in TString src,                            in TInteger line,                            in TriComponentIdType c,                            in TriPortIdType at,                            in TriSignatureIdType signature,                            in TciParameterListType tciPars,                            in TciValueTemplate parsTpl,                            in TciValueDifferenceList diffs,                            in TriComponentIdType from,                            in TciNonValueTemplate fromTpl)</pre>	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	at	The port via which the call is received.
	signature	The signature of the detected call.
	tciPars	The parameters of detected call.
	parsTpl	The template used to check the parameter match.
	diffs	The difference/the mismatch between message and template
	from	The component which called the operation.
	fromTpl	The expected calling component.
<b>Return Value</b>	Void	
<b>Constraint</b>	Shall be called by TE to log the mismatch of a getcall. This event occurs after getcall is checked against a template. This event is used for logging the intercomponent communication.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

### 7.3.4.1.31 tliPrGetCall\_m

<b>Signature</b>	<pre>void tliPrGetCall_m(in TString am, in TInteger ts,                    in TString src,                    in TInteger line, in TriComponentIdType c,                    in TriPortIdType at,                    in TriSignatureIdType signature,                    in TciParameterListType tciPars,                    in TciValueTemplate parsTpl,                    in Value addrValue,                    in TciValueTemplate addressTpl)</pre>	
------------------	--	--

<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	at	The port via which the call is received.
	signature	The signature of the detected call.
	tciParam	The parameters of detected call.
	parTpl	The template used to check the parameter match.
	addrValue	The address value of the source within the SUT.
	addressTpl	The expected address of the source within the SUT.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by TE to log getting a call. This event occurs after getcall has matched against a template. This event is used for logging the communication with the SUT.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

#### 7.3.4.1.32 tliPrGetCall\_c

<b>Signature</b>	<pre>void tliPrGetCall_c(in TString am, in TInteger ts,                    in TString src,                    in TInteger line,                    in TriComponentIdType c,                    in TriPortIdType at,                    in TriSignatureIdType signature,                    in TciParameterListType tciParam,                    in TciValueTemplate parTpl,                    in TriComponentIdType from,                    in TciNonValueTemplate fromTpl)</pre>	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	at	The port via which the call is received.
	signature	The signature of the detected call.
	tciParam	The parameters of detected call.
	parTpl	The template used to check the parameter match.
	from	The component which called the operation.
	fromTpl	The expected calling component.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by TE to log getting a call. This event occurs after getcall has matched against a template. This event is used for logging the intercomponent communication.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

### 7.3.4.1.33 tliPrReply\_m

<b>Signature</b>	<pre>void tliPrReply_m(in TString am, in TInteger ts, in TString src,                  in TInteger line, in TriComponentIdType c,                  in TriPortIdType at, in TriPortIdType to,                  in TriSignatureIdType signature,                  in TciParameterListType tciPars,                  in Value replValue,                  in Value addrValue,                  in TciStatusType encoderFailure,                  in TriParameterListType triPars,                  in TriParameterType repl,                  in TriAddressType address,                  in TriStatusType transmissionFailure)</pre>	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	at	The port via which the reply is sent.
	to	The port to which the reply is sent.
	signature	The signature relating to the reply.
	tciPars	The signature parameters relating to the reply.
	replValue	The reply to be sent.
	addrValue	The address value of the destination within the SUT.
	encoderFailure	The failure message which might occur at encoding.
	triPars	The encoded parameters.
	repl	The encoded reply.
	address	The address of the destination within the SUT.
transmissionFailure	The failure message which might occur at transmission.	
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by SA or TE to log a unicast reply operation. This event occurs after reply execution. This event is used for logging the communication with the SUT.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

### 7.3.4.1.34 tliPrReply\_m\_BC

<b>Signature</b>	<pre>void tliPrReply_m_BC(in TString am, in TInteger ts,                     in TString src,                     in TInteger line, in TriComponentIdType c,                     in TriPortIdType at, in TriPortIdType to,                     in TriSignatureIdType signature,                     in TciParameterListType tciPars,                     in Value replValue,                     in TciStatusType encoderFailure,                     in TriParameterListType triPars,                     in TriParameterType repl,                     in TriStatusType transmissionFailure)</pre>	
------------------	---	--



<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	at	The port via which the reply is sent.
	to	The port to which the reply is sent.
	signature	The signature relating to the reply.
	tciPars	The signature parameters relating to the reply.
	replValue	The reply to be sent.
	encoderFailure	The failure message which might occur at encoding.
	triPars	The encoded parameters.
	repl	The encoded reply.
	transmissionFailure	The failure message which might occur at transmission.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by SA or TE to log a broadcast reply operation. This event occurs after reply execution. This event is used for logging the communication with the SUT.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

#### 7.3.4.1.35 tliPrReply\_m\_MC

<b>Signature</b>	<pre>void tliPrReply_m_MC(in TString am, in TInteger ts,                     in TString src,                     in TInteger line, in TriComponentIdType c,                     in TriPortIdType at, in TriPortIdType to,                     in TriSignatureIdType signature,                     in TciParameterListType tciPars,                     in Value replValue,                     in TciValueList addrValues,                     in TciStatusType encoderFailure,                     in TriParameterListType triPars,                     in TriParameterType repl,                     in TriAddressListType addresses,                     in TriStatusType transmissionFailure)</pre>
------------------	--

<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	at	The port via which the reply is sent.
	to	The port to which the reply is sent.
	signature	The signature relating to the reply.
	tciPars	The signature parameters relating to the reply.
	replValue	The reply to be sent.
	addrValues	The address values of the destinations within the SUT.
	encoderFailure	The failure message which might occur at encoding.
	triPars	The encoded parameters.
	repl	The encoded reply.
	addresses	The addresses of the destinations within the SUT.
transmissionFailure	The failure message which might occur at transmission.	
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by SA or TE to log a multicast reply operation. This event occurs after reply execution. This event is used for logging the communication with the SUT.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

#### 7.3.4.1.36 tliPrReply\_c

<b>Signature</b>	void tliPrReply_c(in TString am, in TInteger ts, in TString src, in TInteger line, in TriComponentIdType c, in TriPortIdType at, in TriPortIdType to, in TriSignatureIdType signature, in Value parsValue, in Value replValue, in TriStatusType transmissionFailure)	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	at	The port via which the reply is sent.
	to	The port to which the reply is sent.
	signature	The signature relating to the reply.
	parsValue	The signature parameters relating to the reply.
	replValue	The reply to be sent.
	transmissionFailure	The failure message which might occur at transmission.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by CH or TE to log a unicast reply operation. This event occurs after reply execution. This event is used for logging the intercomponent communication.	

<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.
---------------	---

#### 7.3.4.1.37 tliPrReply\_c\_BC

<b>Signature</b>	<pre>void tliPrReply_c_BC(in TString am, in TInteger ts,                     in TString src,                     in TInteger line, in TriComponentIdType c,                     in TriPortIdType at,                     in TriPortIdListType to,                     in TriSignatureIdType signature,                     in Value parsValue, in Value replValue,                     in TriStatusType transmissionFailure)</pre>	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	at	The port via which the reply is sent.
	to	The port list to which the reply is sent.
	signature	The signature relating to the reply.
	parsValue	The signature parameters relating to the reply.
	replValue	The reply to be sent.
	transmissionFailure	The failure message which might occur at transmission.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by CH or TE to log a broadcast reply operation. This event occurs after reply execution. This event is used for logging the intercomponent communication.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

#### 7.3.4.1.38 tliPrReply\_c\_MC

<b>Signature</b>	<pre>void tliPrReply_c_MC(in TString am, in TInteger ts,                     in TString src,                     in TInteger line, in TriComponentIdType c,                     in TriPortIdType at,                     in TriPortIdListType to,                     in TriSignatureIdType signature,                     in Value parsValue, in Value replValue,                     in TriStatusType transmissionFailure)</pre>	
------------------	--	--

<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	at	The port via which the reply is sent.
	to	The port list to which the reply is sent.
	signature	The signature relating to the reply.
	parsValue	The signature parameters relating to the reply.
	replValue	The reply to be sent.
	transmissionFailure	The failure message which might occur at transmission.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by CH or TE to log a multicast reply operation. This event occurs after reply execution. This event is used for logging the intercomponent communication.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

#### 7.3.4.1.39 tliPrGetReplyDetected\_m

<b>Signature</b>	<pre>void tliPrGetReplyDetected_m(in TString am, in TInteger ts,                              in TString src,                              in TInteger line,                              in TriComponentIdType c,                              in TriPortIdType at,                              in TriPortIdType from,                              in TriSignatureIdType signature,                              in TriParameterListType triPars,                              in TriParameterType repl,                              in TriAddressType address)</pre>	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	at	The port via which the reply is received.
	from	The port from which the reply has been sent.
	signature	The signature relating to the reply.
	triPars	The encoded parameters of detected reply.
	repl	The received encoded reply.
	address	The address of the source within the SUT.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by SA or TE to log the getreply enqueue operation. This event occurs after getreply is enqueued. This event is used for logging the communication with the SUT.	

<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.
---------------	---

#### 7.3.4.1.40 tliPrGetReplyDetected\_c

<b>Signature</b>	<pre>void tliPrGetReplyDetected_c(in TString am, in TInteger ts,                              in TString src,                              in TInteger line,                              in TriComponentIdType c,                              in TriPortIdType at,                              in TriPortIdType from,                              in TriSignatureIdType signature,                              in TciParameterListType tciPars,                              in Value replValue)</pre>	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	at	The port via which the reply is received.
	from	The port from which the reply has been sent.
	signature	The signature relating to the reply.
	tciPars	The encoded parameters of detected reply.
	replValue	The received reply.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by CH or TE to log the getreply enqueue operation. This event occurs after getreply is enqueued. This event is used for logging the intercomponent communication.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

#### 7.3.4.1.41 tliPrGetReplyMismatch\_m

<b>Signature</b>	<pre>void tliPrGetReplyMismatch_m(in TString am, in TInteger ts,                               in TString src,                               in TInteger line,                               in TriComponentIdType c,                               in TriPortIdType at,                               in TriSignatureIdType signature,                               in TciParameterListType tciPars,                               in TciValueTemplate parsTpl,                               in Value replValue,                               in TciValueTemplate replyTpl,                               in TciValueDifferenceList diffs,                               in Value addrValue,                               in TciValueTemplate addressTpl)</pre>	
------------------	--	--

<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	at	The port via which the reply is received.
	signature	The signature relating to the reply.
	tciPars	The signature parameters relating to the reply.
	parsTpl	The template used to check the parameter match.
	replValue	The received reply.
	replyTpl	The template used to check the reply match.
	diffs	The difference/the mismatch between reply and template
	addrValue	The address value of the source within the SUT.
	addressTpl	The expected address of the source within the SUT.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by TE to log the mismatch of a getreply operation. This event occurs after getreply is checked against a template. This event is used for logging the communication with SUT.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

#### 7.3.4.1.42 tliPrGetReplyMismatch\_c

<b>Signature</b>	<pre>void tliPrGetReplyMismatch_c(in TString am, in TInteger ts,                              in TString src,                              in TInteger line,                              in TriComponentIdType c,                              in TriPortIdType at,                              in TriSignatureIdType signature,                              in TciParameterListType tciPars,                              in TciValueTemplate parsTpl,                              in Value replValue,                              in TciValueTemplate replyTpl,                              in TciValueDifferenceList diffs,                              in TriComponentIdType from,                              in TciNonValueTemplate fromTpl)</pre>
------------------	--

<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	at	The port via which the reply is received.
	signature	The signature relating to the reply.
	tciPars	The signature parameters relating to the reply.
	parsTmpl	The template used to check the parameter match.
	repl	The received reply.
	replyTmpl	The template used to check the reply match.
	diffs	The difference/the mismatch between reply and template
	from	The component which sent the reply.
	fromTmpl	The expected replying component.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by TE to log the mismatch of a getreply operation. This event occurs after getreply is checked against a template. This event is used for logging the intercomponent communication.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

#### 7.3.4.1.43 tliPrGetReply\_m

<b>Signature</b>	<pre>void tliPrGetReply_m(in TString am, in TInteger ts,                     in TString src,                     in TInteger line, in TriComponentIdType c,                     in TriPortIdType at,                     in TriSignatureIdType signature,                     in TciParameterListType tciPars,                     in TciValueTemplate parsTmpl,                     in Value replValue,                     in TciValueTemplate replyTmpl,                     in Value addrValue,                     in TciValueTemplate addressTmpl)</pre>
------------------	--

<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	at	The port via which the reply is received.
	signature	The signature relating to the reply.
	tciPars	The signature parameters relating to the reply.
	parsTmpl	The template used to check the parameter match.
	replValue	The received reply.
	replyTmpl	The template used to check the reply match.
	addrValue	The address value of the source within the SUT.
	addressTmpl	The expected address of the source within the SUT.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by TE to log getting a reply. This event occurs after getreply is checked against a template. This event is used for logging the communication with SUT.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

#### 7.3.4.1.44 tliPrGetReply\_c

<b>Signature</b>	<pre>void tliPrGetReply_c(in TString am, in TInteger ts, in TString src,                     in TInteger line, in TriComponentIdType c,                     in TriPortIdType at,                     in TriSignatureIdType signature,                     in TciParameterListType tciPars,                     in TciValueTemplate parsTmpl,                     in Value replValue,                     in TciValueTemplate replyTmpl,                     in TriComponentIdType from,                     in TciNonValueTemplate fromTmpl)</pre>	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	at	The port via which the reply is received.
	signature	The signature relating to the reply.
	tciPars	The signature parameters relating to the reply.
	parsTmpl	The template used to check the parameter match.
	replValue	The received reply.
	replyTmpl	The template used to check the reply match.
	from	The component which sent the reply.
	fromTmpl	The expected replying component.



<b>Return Value</b>	void
<b>Constraint</b>	Shall be called by TE to log getting a reply. This event occurs after getreply is checked against a template. This event is used for logging the intercomponent communication.
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.

#### 7.3.4.1.45 tliPrRaise\_m

<b>Signature</b>	<pre>void tliPrRaise_m(in TString am, in TInteger ts,                  in TString src,                  in TInteger line, in TriComponentIdType c,                  in TriPortIdType at, in TriPortIdType to,                  in TriSignatureIdType signature,                  in TciParameterListType tciPars,                  in Value excValue,                  in Value addrValue,                  in TciStatusType encoderFailure,                  in TriExceptionType exc,                  in TriAddressType address,                  in TriStatusType transmissionFailure)</pre>	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	at	The port via which the exception is sent.
	to	The port to which the exception is sent.
	signature	The signature relating to the exception.
	tciPars	The signature parameters relating to the exception.
	excValue	The exception to be sent.
	addrValue	The address value of the destination within the SUT.
	encoderFailure	The failure message which might occur at encoding.
	exc	The encoded exception.
	address	The address of the destination within the SUT.
transmissionFailure	The failure message which might occur at transmission.	
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by SA or TE to log a unicast raise operation. This event occurs after reply execution. This event is used for logging the communication with the SUT.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

### 7.3.4.1.46 tliPrRaise\_m\_BC

<b>Signature</b>	<pre>void tliPrRaise_m_BC(in TString am, in TInteger ts,                     in TString src,                     in TInteger line, in TriComponentIdType c,                     in TriPortIdType at, in TriPortIdType to,                     in TriSignatureIdType signature,                     in TciParameterListType tciPars,                     in Value excValue,                     in TciStatusType encoderFailure,                     in TriExceptionType exc,                     in TriStatusType transmissionFailure)</pre>	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	at	The port via which the exception is sent.
	to	The port to which the exception is sent.
	signature	The signature relating to the exception.
	tciPars	The signature parameters relating to the exception.
	excValue	The exception to be sent.
	encoderFailure	The failure message which might occur at encoding.
	exc	The encoded exception.
	transmissionFailure	The failure message which might occur at transmission.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by SA or TE to log a broadcast raise operation. This event occurs after reply execution. This event is used for logging the communication with the SUT.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

### 7.3.4.1.47 tliPrRaise\_m\_MC

<b>Signature</b>	<pre>void tliPrRaise_m_MC(in TString am, in TInteger ts,                     in TString src,                     in TInteger line, in TriComponentIdType c,                     in TriPortIdType at, in TriPortIdType to,                     in TriSignatureIdType signature,                     in TciParameterListType tciPars,                     in Value excValue,                     in TciValueList addrValues,                     in TciStatusType encoderFailure,                     in TriExceptionType exc,                     in TriAddressListType addresses,                     in TriStatusType transmissionFailure)</pre>	
------------------	---	--

<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	at	The port via which the exception is sent.
	to	The port to which the exception is sent.
	signature	The signature relating to the exception.
	tciPars	The signature parameters relating to the exception.
	excValue	The exception to be sent.
	addrValues	The address values of the destinations within the SUT.
	encoderFailure	The failure message which might occur at encoding.
	exc	The encoded exception.
	addresses	The addresses of the destinations within the SUT.
transmissionFailure	The failure message which might occur at transmission.	
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by SA or TE to log a multicast raise operation. This event occurs after reply execution. This event is used for logging the communication with the SUT.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

#### 7.3.4.1.48 tliPrRaise\_c

<b>Signature</b>	<pre>void tliPrRaise_c(in TString am, in TInteger ts,                  in TString src,                  in TInteger line, in TriComponentIdType c,                  in TriPortIdType at, in TriPortIdType to,                  in TriSignatureIdType signature,                  in TciParameterListType tciPars,                  in Value excValue,                  in TriStatusType transmissionFailure)</pre>	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	at	The port via which the exception is sent.
	to	The port to which the exception is sent.
	signature	The signature relating to the exception.
	tciPars	The signature parameters relating to the exception.
	excValue	The exception to be sent.
	transmissionFailure	The failure message which might occur at transmission.

<b>Return Value</b>	void
<b>Constraint</b>	Shall be called by CH or TE to log a unicast raise operation. This event occurs after reply execution. This event is used for logging the intercomponent communication.
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.

#### 7.3.4.1.49 tliPrRaise\_c\_BC

<b>Signature</b>	<pre>void tliPrRaise_c_BC(in TString am, in TInteger ts,                     in TString src,                     in TInteger line,                     in TriComponentIdType c,                     in TriPortIdType at,                     in TriPortIdListType to,                     in TriSignatureIdType signature,                     in TciParameterListType tciPars,                     in Value excValue,                     in TriStatusType transmissionFailure)</pre>	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	at	The port via which the exception is sent.
	to	The port list to which the exception is sent.
	signature	The signature relating to the exception.
	tciPars	The signature parameters relating to the exception.
	excValue	The exception to be sent.
	transmissionFailure	The failure message which might occur at transmission.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by CH or TE to log a broadcast raise operation. This event occurs after reply execution. This event is used for logging the intercomponent communication.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

#### 7.3.4.1.50 tliPrRaise\_c\_MC

<b>Signature</b>	<pre>void tliPrRaise_c_MC(in TString am, in TInteger ts,                     in TString src,                     in TInteger line,                     in TriComponentIdType c,                     in TriPortIdType at,                     in TriPortIdListType to,                     in TriSignatureIdType signature,                     in TciParameterListType tciPars,                     in Value excValue,                     in TriStatusType transmissionFailure)</pre>	
------------------	--	--

<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	at	The port via which the exception is sent.
	to	The port list to which the exception is sent.
	signature	The signature relating to the exception.
	tciPars	The signature parameters relating to the exception.
	excValue	The exception to be sent.
	transmissionFailure	The failure message which might occur at transmission.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by CH or TE to log a multicast raise operation. This event occurs after reply execution. This event is used for logging the intercomponent communication.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

#### 7.3.4.1.51 tliPrCatchDetected\_m

<b>Signature</b>	<pre>void tliPrCatchDetected_m(in TString am, in TInteger ts,                           in TString src,                           in TInteger line,                           in TriComponentIdType c,                           in TriPortIdType at,                           in TriPortIdType from,                           in TriSignatureIdType signature,                           in TriExceptionType exc,                           in TriAddressType address)</pre>	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	at	The port via which the exception is received.
	from	The port from which the exception has been sent.
	signature	The signature relating to the exception.
	exc	The exception caught.
	address	The address of the source within the SUT.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by SA or TE to log the catch enqueue operation. This event occurs after catch is enqueued. This event is used for logging the communication with the SUT.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

### 7.3.4.1.52 tliPrCatchDetected\_c

<b>Signature</b>	void tliPrCatchDetected_c(in TString am, in TInteger ts, in TString src, in TInteger line, in TriComponentIdType c, in TriPortIdType at, in TriPortIdType from, in TriSignatureIdType signature, in Value excValue)	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	at	The port via which the exception is received.
	from	The port from which the exception has been sent.
	signature	The signature relating to the exception.
	excValue	The caught exception.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by CH or TE to log the catch enqueue operation. This event occurs after catch is enqueued. This event is used for logging the intercomponent communication.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

### 7.3.4.1.53 tliPrCatchMismatch\_m

<b>Signature</b>	void tliPrCatchMismatch_m(in TString am, in TInteger ts, in TString src, in TInteger line, in TriComponentIdType c, in TriPortIdType at, in TriSignatureIdType signature, in Value excValue, in TciValueTemplate excTmpl, in TciValueDifferenceList diffs, in Value addrValue, in TciValueTemplate addressTmpl)	
------------------	--	--

<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	at	The port via which the exception is received.
	signature	The signature relating to the exception.
	excValue	The received exception.
	excTpl	The template used to check the exception match.
	diffs	The difference/the mismatch between exception and template
	addrValue	The address value of the source within the SUT.
	addressTpl	The expected address of the source within the SUT.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by TE to log the mismatch of a catch operation. This event occurs after catch is checked against a template. This event is used for logging the communication with SUT.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

#### 7.3.4.1.54 tliPrCatchMismatch\_c

<b>Signature</b>	<pre>void tliPrCatchMismatch_c(in TString am, in TInteger ts,                           in TString src,                           in TInteger line,                           in TriComponentIdType c,                           in TriPortIdType at,                           in TriSignatureIdType signature,                           in Value excValue,                           in TciValueTemplate excTpl,                           in TciValueDifferenceList diffs,                           in TriComponentIdType from,                           in TciNonValueTemplate fromTpl)</pre>	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	at	The port via which the exception is received.
	signature	The signature relating to the exception.
	excValue	The received exception.
	excTpl	The template used to check the exception match.
	diffs	The difference/the mismatch between exception and template
	from	The component which sent the reply.
	fromTpl	The expected replying component.
<b>Return Value</b>	void	

<b>Constraint</b>	Shall be called by TE to log the mismatch of a catch operation. This event occurs after catch is checked against a template. This event is used for logging the intercomponent communication.
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.

#### 7.3.4.1.55 tliPrCatch m

<b>Signature</b>	<pre>void tliPrCatch_m(in TString am, in TInteger ts,                  in TString src,                  in TInteger line, in TriComponentIdType c,                  in TriPortIdType at,                  in TriSignatureIdType signature,                  in Value excValue,                  in TciValueTemplate excTmpl,                  in Value addrValue,                  in TciValueTemplate addressTmpl)</pre>	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	at	The port via which the exception is received.
	signature	The signature relating to the exception.
	excValue	The received exception.
	excTmpl	The template used to check the exception match.
	addrValue	The address value of the source within the SUT.
	addressTmpl	The expected address of the source within the SUT.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by SA or TE to log catching an exception. This event occurs after catch is checked against a template. This event is used for logging the communication with SUT.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

#### 7.3.4.1.56 tliPrCatch c

<b>Signature</b>	<pre>void tliPrCatch_c(in TString am, in TInteger ts,                  in TString src,                  in TInteger line, in TriComponentIdType c,                  in TriPortIdType at,                  in TriSignatureIdType signature,                  in Value excValue,                  in TciValueTemplate excTmpl,                  in TriComponentIdType from,                  in TciNonValueTemplate fromTmpl)</pre>	
------------------	---	--



<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	at	The port via which the exception is received.
	signature	The signature relating to the exception.
	excValue	The received exception.
	excTmpl	The template used to check the exception match.
	from	The component which sent the reply.
	fromTmpl	The expected replying component.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by CH or TE to log catching an exception. This event occurs after catch is checked against a template. This event is used for logging the intercomponent communication.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

#### 7.3.4.1.57 tliPrCatchTimeoutDetected

<b>Signature</b>	<pre>void tliPrCatchTimeoutDetected(in TString am, in TInteger ts,                                in TString src,                                in TInteger line,                                in TriComponentIdType c,                                in TriPortIdType at,                                in TriSignatureIdType signature)</pre>	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	at	The port via which the exception is received.
	signature	The signature relating to the exception.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by PA or TE to log the detection of a catch timeout. This event occurs after the timeout is enqueued.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

### 7.3.4.1.58 tliPrCatchTimeout

<b>Signature</b>	void tliPrCatchTimeout (in TString am, in TInteger ts, in TString src, in TInteger line, in TriComponentIdType c, in TriPortIdType at, in TriSignatureIdType signature)	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	at	The port via which the exception is received.
	signature	The signature relating to the exception.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by TE to log catching a timeout. This event occurs after the catch timeout has been performed.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

### 7.3.4.1.59 tliCCreate

<b>Signature</b>	void tliCCreate(in TString am, in TInteger ts, in TString src, in TInteger line, in TriComponentIdType c, in TriComponentIdType comp, in TString name, in TBoolean alive)	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	comp	The component which is created.
	name	The name of the component which is created.
	alive	If the component is an alive component.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by TE to log the create component operation. This event occurs after component creation.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

### 7.3.4.1.60 tliCStart

<b>Signature</b>	void tliCStart(in TString am, in TInteger ts, in TString src, in TInteger line, in TriComponentIdType c, in TriComponentIdType comp, in TciBehaviourIdType beh, in TciParameterListType tciPars)	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	comp	The component which is started.
	beh	The behaviour being started on the component.
	tciPars	The parameters of the started behaviour.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by TE to log the start component operation. This event occurs after component start.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

### 7.3.4.1.61 tliCRunning

<b>Signature</b>	void tliCRunning(in TString am, in TInteger ts, in TString src, in TInteger line, in TriComponentIdType c, in TriComponentIdType comp, in ComponentStatusType status)	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	comp	The component which is checked to be running.
	status	The status of this component.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by TE to log the running component operation. This event occurs after component running.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

### 7.3.4.1.62 tliCAlive

<b>Signature</b>	void tliCAlive(in TString am, in TInteger ts, in TString src, in TInteger line, in TriComponentIdType c, in TriComponentIdType comp, in ComponentStatusType status)	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	comp	The component which is checked to be running.
	status	The status of this component.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by TE to log the alive component operation. This event occurs after component alive.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

### 7.3.4.1.63 tliCStop

<b>Signature</b>	void tliCStop(in TString am, in TInteger ts, in TString src, in TInteger line, in TriComponentIdType c, in TriComponentIdType comp)	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	comp	The component which is stopped.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by TE to log the stop component operation. This event occurs after component stop.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

### 7.3.4.1.64 tliCKill

<b>Signature</b>	void tliCKill(in TString am, in TInteger ts, in TString src, in TInteger line, in TriComponentIdType c, in TriComponentIdType comp)	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.

	comp	The component which is killed.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by TE to log the kill component operation. This event occurs after component kill.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

#### 7.3.4.1.65 tliCDoneMismatch

<b>Signature</b>	void tliCDoneMismatch(in TString am, in TInteger ts, in TString src, in TInteger line, in TriComponentIdType c, in TriComponentIdType comp, in TciNonValueTemplate compTmpl)	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	comp	The first component that did not match.
	compTmpl	The template used to check the done match.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by TE to log the mismatch of a done component operation. This event occurs after done is checked against a template.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

#### 7.3.4.1.66 tliCDone

<b>Signature</b>	void tliCDone (in TString am, in TInteger ts, in TString src, in TInteger line, in TriComponentIdType c, in TciNonValueTemplate compTmpl)	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	compTmpl	The template used to check the done match.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by TE to log the done component operation. This event occurs after the done operation.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

### 7.3.4.1.67 tliCKilledMismatch

<b>Signature</b>	void tliCKilledMismatch(in TString am, in TInteger ts, in TString src, in TInteger line, in TriComponentIdType c, in TriComponentIdType comp, in TciNonValueTemplate compTpl)	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	comp	The first component that did not match.
	compTpl	The template used to check the killed match.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by TE to log the mismatch of a killed component operation. This event occurs after killed is checked against a template.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

### 7.3.4.1.68 tliCKilled

<b>Signature</b>	void tliCKilled (in TString am, in TInteger ts, in TString src, in TInteger line, in TriComponentIdType c, in TciNonValueTemplate compTpl)	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	compTpl	The template used to check the killed match.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by TE to log the killed component operation. This event occurs after the killed operation.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

### 7.3.4.1.69 tliCTerminated

<b>Signature</b>	void tliCTerminated(in TString am, in TInteger ts, in TString src, in TInteger line, in TriComponentIdType c, in VerdictValue verdict, in TString reason)	
------------------	---	--

<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	verdict	The verdict of the component.
	reason	The optional reason of the setverdict statement
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by TE to log the termination of a component. This event occurs after the termination of the component.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

#### 7.3.4.1.70 tliPConnect

<b>Signature</b>	<pre>void tliPConnect(in TString am, in TInteger ts, in TString src,                 in TInteger line, in TriComponentIdType c,                 in TriPortIdType port1,                 in TriPortIdType port2)</pre>	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	port1	The first port to be connected.
	port2	The second port to be connected.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by CH or TE to log the connect operation. This event occurs after the connect operation.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

#### 7.3.4.1.71 tliPDisconnect

<b>Signature</b>	<pre>void tliPDisconnect(in TString am, in TInteger ts,                    in TString src,                    in TInteger line, in TriComponentIdType c,                    in TriPortIdType port1,                    in TriPortIdType port2)</pre>	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	port1	The first port to be disconnected.

	port2	The second port to be disconnected.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by CH or TE to log the disconnect operation. This event occurs after the disconnect operation.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

#### 7.3.4.1.72 tliPMap

<b>Signature</b>	void tliPMap(in TString am, in TInteger ts, in TString src, in TInteger line, in TriComponentIdType c, in TriPortIdType port1, in TriPortIdType port2)	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	port1	The first port to be mapped.
	port2	The second port to be mapped.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by SA or TE to log the map operation. This event occurs after the map operation.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

#### 7.3.4.1.73 tliPMapParam

<b>Signature</b>	void tliPMapParam(in TString am, in TInteger ts, in TString src, in TInteger line, in TriComponentIdType c, in TriPortIdType port1, in TriPortIdType port2, in TciParameterListType tciPars, in TciStatusType encoderFailure, in TriParameterListType triPars)	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	port1	The first port to be mapped.
	port2	The second port to be mapped.
	tciPars	The configuration parameter list.
	encoderFailure	The failure message which might occur at encoding.
	triPars	The encoded configuration parameter list.
<b>Return Value</b>	void	



<b>Constraint</b>	Shall be called by SA or TE to log the map operation. This event occurs after the map operation including parameters.
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.

#### 7.3.4.1.74 tliPUnmap

<b>Signature</b>	void tliPUnmap(in TString am, in TInteger ts, in TString src, in TInteger line, in TriComponentIdType c, in TriPortIdType port1, in TriPortIdType port2)	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	port1	The first port to be unmapped.
	port2	The second port to be unmapped.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by SA or TE to log the unmap operation. This event occurs after the unmap operation.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

#### 7.3.4.1.75 tliPUnmapParam

<b>Signature</b>	void tliPUnmapParam(in TString am, in TInteger ts, in TString src, in TInteger line, in TriComponentIdType c, in TriPortIdType port1, in TriPortIdType port2, in TciParameterListType tciPars, in TciStatusType encoderFailure, in TriParameterListType triPars)	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	port1	The first port to be unmapped.
	port2	The second port to be unmapped.
	tciPars	The configuration parameter list.
	encoderFailure	The failure message which might occur at encoding.
	triPars	The encoded configuration parameter list.
<b>Return Value</b>	void	

<b>Constraint</b>	Shall be called by SA or TE to log the unmap operation. This event occurs after the unmap operation including parameters.
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.

#### 7.3.4.1.76 tliPClear

<b>Signature</b>	void tliPClear(in TString am, in TInteger ts, in TString src, in TInteger line, in TriComponentIdType c, in TriPortIdType port)	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	port	The port to be cleared.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by TE to log the port clear operation. This event occurs after the port clear operation.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

#### 7.3.4.1.77 tliPStart

<b>Signature</b>	void tliPStart(in TString am, in TInteger ts, in TString src, in TInteger line, in TriComponentIdType c, in TriPortIdType port)	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	port	The port to be started.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by TE to log the port start operation. This event occurs after the port start operation.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

#### 7.3.4.1.78 tliPStop

<b>Signature</b>	void tliPStop(in TString am, in TInteger ts, in TString src, in TInteger line, in TriComponentIdType c, in TriPortIdType port)	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.

	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	port	The port to be stopped.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by TE to log the port stop operation. This event occurs after the port stop operation.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

#### 7.3.4.1.79 tliPHalt

<b>Signature</b>	void tliPHalt(in TString am, in TInteger ts, in TString src, in TInteger line, in TriComponentIdType c, in TriPortIdType port)	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	port	The port to be stopped.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by TE to log the port halt operation. This event occurs after the port halt operation.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

#### 7.3.4.1.80 tliEncode

<b>Signature</b>	void tliEncode(in TString am, in TInteger ts, in TString src, in TInteger line, in TriComponentIdType c, in Value val, in TciStatusType encoderFailure, in TriMessageType msg, in TString codec)	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	value	The value to be encoded.
	encoderFailure	The failure message which might occur at encoding.
	msg	The encoded value.
codec	The used encoder.	
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by CD or TE to log the encode operation.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

### 7.3.4.1.81 tliDecode

<b>Signature</b>	void tliDecode(in TString am, in TInteger ts, in TString src, in TInteger line, in TriComponentIdType c, in TriMessageType msg, in TciStatusType decoderFailure, in Value val, in TString codec)	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	msg	The value to be decoded.
	decoderFailure	The failure message which might occur at decoding.
	val	The decoded value.
	codec	The used decoder.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by CD or TE to log the decode operation.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

### 7.3.4.1.82 tliTTimeoutDetected

<b>Signature</b>	void tliTTimeoutDetected(in TString am, in TInteger ts, in TString src, in TInteger line, in TriComponentIdType c, in TriTimerIdType timer)	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	timer	The timer that timed out.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by PA or TE to log the detection of a timeout. This event occurs after timeout is enqueued.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

### 7.3.4.1.83 tliTTimeoutMismatch

<b>Signature</b>	void tliTTimeoutMismatch(in TString am, in TInteger ts, in TString src, in TInteger line, in TriComponentIdType c, in TriTimerIdType timer, in TciNonValueTemplate timerTpl)	
------------------	---	--

<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	timer	The first timer that did not match.
	timerTmpl	The timer template that did not match.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by TE to log a timeout mismatch. This event occurs after a timeout match failed.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

#### 7.3.4.1.84 tliTTimeout

<b>Signature</b>	<pre>void tliTTimeout(in TString am, in TInteger ts,                 in TString src,                 in TInteger line,                 in TriComponentIdType c,                 in TriTimerIdType timer,                 in TciNonValueTemplate timerTmpl)</pre>	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	timer	The timer that matched.
	timerTmpl	The timer template that matched.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by TE to log a timeout match. This event occurs after a timeout matched.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

#### 7.3.4.1.85 tliTStart

<b>Signature</b>	<pre>void tliTStart(in TString am, in TInteger ts, in TString src,                in TInteger line, in TriComponentIdType c,                in TriTimerIdType timer,                in TriTimerDurationType dur)</pre>	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	timer	The timer that is started.

	dur	The timer duration.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by PA or TE to log the start of a timer. This event occurs after the start timer operation.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

#### 7.3.4.1.86 tliTStop

<b>Signature</b>	void tliTStop(in TString am, in TInteger ts, in TString src, in TInteger line, in TriComponentIdType c, in TriTimerIdType timer, in TriTimerDurationType dur)	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	timer	The timer that is stopped.
	dur	The duration of the timer when it was stopped.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by PA or TE to log the stop of a timer. This event occurs after the stop timer operation.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

#### 7.3.4.1.87 tliTRead

<b>Signature</b>	void tliTRead(in TString am, in TInteger ts, in TString src, in TInteger line, in TriComponentIdType c, in TriTimerIdType timer, in TriTimerDurationType elapsed)	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	timer	The timer that is started.
	elapsed	The elapsed time of the timer.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by PA or TE to log the reading of a timer. This event occurs after the read timer operation.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

### 7.3.4.1.88 tliTRunning

<b>Signature</b>	void tliTRunning(in TString am, in TInteger ts, in TString src, in TInteger line, in TriComponentIdType c, in TriTimerIdType timer, in TimerStatusType status)	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	timer	The timer which is checked to be running.
	status	The status of this component.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by PA or TE to log the running timer operation. This event occurs after the running timer operation.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

### 7.3.4.1.89 tliSEnter

<b>Signature</b>	void tliSEnter(in TString am, in TInteger ts, in TString src, in TInteger line, in TriComponentIdType c, in QualifiedName name, in TciParameterListType tciPars, in TString kind)	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	name	The name of the scope.
	tciPars	The parameters of the scope.
	kind	The kind of the scope.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by TE to log the entering of a scope. This event occurs after the scoped has been entered.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

### 7.3.4.1.90 tliSLeave

<b>Signature</b>	void tliSLeave(in TString am, in TInteger ts, in TString src, in TInteger line, in TriComponentIdType c, in QualifiedName name, in TciParameterListType tciPars, in Value returnValue, in TString kind)	
------------------	---	--

<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	name	The name of the scope.
	tciPars	The parameters of the scope.
	returnValue	The return value of the scope.
	kind	The kind of the scope.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by TE to log the leaving of a scope. This event occurs after the scoped has been left.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

#### 7.3.4.1.91 tliVar

<b>Signature</b>	void tliVar(in TString am, in TInteger ts, in TString src, in TInteger line, in TriComponentIdType c, in QualifiedName name, in Value varValue)	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	name	The name of the variable.
	varValue	The new value of the variable.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by TE to log the modification of the value of a variable. This event occurs after the value has been changed.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

#### 7.3.4.1.92 tliModulePar

<b>Signature</b>	void tliModulePar(in TString am, in TInteger ts, in TString src, in TInteger line, in TriComponentIdType c, in QualifiedName name, in Value parValue)	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	name	The name of the module parameter.



	parValue	The value of the module parameter.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by TE to log the value of a module parameter. This event occurs after the access to the value of a module parameter.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

#### 7.3.4.1.93 tliGetVerdict

<b>Signature</b>	void tliGetVerdict(in TString am, in TInteger ts, in TString src, in TInteger line, in TriComponentIdType c, in VerdictValue verdict)	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	verdict	The current value of the local verdict.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by TE to log the getverdict operation. This event occurs after the getverdict operation.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

#### 7.3.4.1.94 tliSetVerdict

<b>Signature</b>	void tliSetVerdict(in TString am, in TInteger ts, in TString src, in TInteger line, in TriComponentIdType c, in VerdictValue verdict, in TString reason)	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	verdict	The value to be set to the local verdict.
	reason	The optional reason of the setverdict statement.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by TE to log the setverdict operation or the occurrence of a runtime error. If used to log the setverdict operation, then this event occurs after the setverdict operation.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

### 7.3.4.1.95 tliLog

<b>Signature</b>	void tliLog (in TString am, in TInteger ts, in TString src, in TInteger line, in TriComponentIdType c, in TString log)	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	log	The string to be logged.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by TM or TE to log the TTCN-3 statement log. This event occurs after the TTCN-3 log operation.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

### 7.3.4.1.96 tliAEnter

<b>Signature</b>	void tliAEnter(in TString am, in TInteger ts, in TString src, in TInteger line, in TriComponentIdType c)	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by TE to log entering an alt. This event occurs after an alt has been entered.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

### 7.3.4.1.97 tliALeave

<b>Signature</b>	void tliALeave(in TString am, in TInteger ts, in TString src, in TInteger line, in TriComponentIdType c)	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by TE to log leaving an alt. This event occurs after the alt has been leaved.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

### 7.3.4.1.98 tliANomatch

<b>Signature</b>	void tliANomatch (in TString am, in TInteger ts, in TString src, in TInteger line, in TriComponentIdType c)	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by TE to log the nomatch of an alt. This event occurs after the alt has not matched.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

### 7.3.4.1.99 tliARepeat

<b>Signature</b>	void tliARepeat (in TString am, in TInteger ts, in TString src, in TInteger line, in TriComponentIdType c)	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by TE to log repeating an alt. This event occurs when the alt has been repeated.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

### 7.3.4.1.100 tliADefaults

<b>Signature</b>	void tliADefaults (in TString am, in TInteger ts, in TString src, in TInteger line, in TriComponentIdType c)	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by TE to log entering the default section. This event occurs after the default section has been entered.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

### 7.3.4.1.101 tliAActivate

<b>Signature</b>	<pre>void tliAActivate(in TString am, in TInteger ts,                  in TString src,                  in TInteger line, in TriComponentIdType c,                  in QualifiedName name,                  in TciParameterListType tciPars,                  in Value ref)</pre>	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	name	The name of the default.
	tciPars	The parameter of the default.
	ref	The resulting default reference.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by TE to log the activation of a default. This event occurs after the default activation.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

### 7.3.4.1.102 tliADeactivate

<b>Signature</b>	<pre>void tliADeactivate(in TString am, in TInteger ts,                    in TString src,                    in TInteger line, in TriComponentIdType c,                    in Value ref)</pre>	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	ref	The default reference.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by TE to log the deactivation of a default. This event occurs after the default deactivation.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

### 7.3.4.1.103 tliAwait

<b>Signature</b>	void tliAwait(in TString am, in TInteger ts, in TString src, in TInteger line, in TriComponentId c)	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by TE to log that the component awaits events for a new snapshot.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

### 7.3.4.1.104 tliAction

<b>Signature</b>	void tliAction(in TString am, in TInteger ts, in TString src, in TInteger line, in TriComponentIdType c, in TString action)	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	action	The action to be performed.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by TE to log that the component executed an SUT action.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

### 7.3.4.1.105 tliMatch

<b>Signature</b>	void tliMatch(in TString am, in TInteger ts, in TString src, in TInteger line, in TriComponentIdType c, in Value expr, in TciValueTemplate tpl)	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	expr	The expression to be matched with tpl.
	tpl	The template to be matched with expr.
<b>Return Value</b>	void	

<b>Constraint</b>	Shall be called by TE to log that the component successfully executed a match operation.
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.

#### 7.3.4.1.106 tliMatchMismatch

<b>Signature</b>	void tliMatchMismatch(in TString am, in TInteger ts, in TString src, in TInteger line, in TriComponentIdType c, in Value expr, in TciValueTemplate tpl, in TciValueDifferenceList diffs)	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	expr	The expression to be matched with tpl.
	tpl	The template to be matched with expr.
	diffs	The difference/the mismatch between expr and tpl.
<b>Return Value</b>	void	
<b>Constraint</b>	Shall be called by TE to log that the component unsuccessfully executed a match operation – a mismatch occurred.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

#### 7.3.4.1.107 tliInfo

<b>Signature</b>	void tliInfo (in TString am, in TInteger ts, in TString src, in TInteger line, in TriComponentIdType c, in TInteger level, in TString info)	
<b>In Parameters</b>	am	An additional message.
	ts	The time when the event is produced.
	src	The source file of the test specification.
	line	The line number where the request is performed.
	c	The component which produces this event.
	level	The information level.
	info	The information.
	<b>Return Value</b>	void
<b>Constraint</b>	Can be called by TE to log additional information during test execution. The generation of this event is tool dependent as well as the usage of the parameters level and info.	
<b>Effect</b>	The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document.	

## 8 Java language mapping

### 8.1 Introduction

This clause introduces the TCI Java language mapping. For efficiency reasons a dedicated language mapping is introduced instead of using the OMG IDL [6] to Java language [7].

The Java language mapping for the TTCN-3 control interface defines how the IDL definitions described in clause 7 are mapped to the Java language. The language mapping is independent of the used Java version as only basic Java language constructs are used.

### 8.2 Names and scopes

#### 8.2.1 Names

Although there are no conflicts between identifiers used in the IDL definition and the Java language some naming translation rules are applied to the IDL identifiers.

Java interfaces or class identifiers are omitting the trailing `Type` used in the IDL definition.

EXAMPLE – The IDL type `tciTestCaseIdType` maps to `TciTestCaseId` in Java.

The resulting mapping conforms to the standard Java coding conventions.

#### 8.2.2 Scopes

The IDL module `tciInterface` is mapped to the Java package `org.etsi.ttcn3.tci`. All IDL type declarations within this module are mapped to Java classes or interface declarations within this package.

### 8.3 Type mapping

#### 8.3.1 Basic type mapping

Table 3 gives an overview on how the native types `TBoolean`, `TFloat`, `TInteger`, `TString`, and `TStringSeq` are mapped to the Java types.

**Table 3 – Basic type mappings**

IDL Type	Java Type
<code>TBoolean</code>	<code>boolean</code>
<code>TFloat</code>	<code>float</code>
<code>TInteger</code>	<code>int</code>
<code>TString</code>	<code>java.lang.String</code>
<code>TStringSeq</code>	<code>java.lang.String[]</code>

#### **boolean**

The IDL `TBoolean` type is mapped to the java basic type `boolean`.

#### **float**

The IDL `TFloat` type is mapped to the java basic type `float`.

#### **char**

The IDL `TChar` type is mapped to the java basic type `char`.

#### **int**

The IDL `TInteger` type is mapped to the java basic type `int`.

## String

The IDL `tstring` type is mapped to the `java.lang.String` class without range checking or bounds for characters in the string. All possible strings defined in TTCN-3 can be converted to `java.lang.String`.

## String[]

The IDL `tstringseq` type is mapped to an array of the `java.lang.String` class.

## Universal Char

The IDL `tuniversalchar` type is mapped to a java basic type `int`. The integer uses the canonical form as defined in ISO/IEC 10646 [5], clause 6.2.

### 8.3.2 Structured type mapping

The TCI IDL description defines user defined types as native types. In the Java language mapping these types are mapped to Java interfaces. The interfaces define methods and attributes being available for objects implementing this interface.

#### 8.3.2.1 TciParameterType

`TciParameterType` is mapped to the following interface:

```
// TCI IDL TciParameterType
package org.etsi.ttcn.tci;
public interface TciParameter {
    public String    getParameterName();
    public void     setParameterName(String name);
    public int      getParameterPassingMode();
    public void     setParameterPassingMode(TciParameterPassingMode mode);
    public Value    getParameter();
    public void     setParameter(Value parameter);
}
```

#### Methods:

- `getParameterName` Returns the parameter name as defined in the TTCN-3 specification.
- `setParameterName` Sets the name of this `TciParameter` parameter to `name`.
- `getParameterPassingMode` Returns the parameter passing mode of this parameter.
- `setParameterPassingMode` Sets the parameter mode of this `TriParameter` parameter to `mode`.
- `getParameter` Returns the `Value` parameter of this `TciParameter`, or the `null` object if the parameter contains the distinct value `null`.
- `setParameter` Sets the `Value` parameter of this `TciParameter` to `parameter`. If the distinct value `null` shall be set to indicate that this parameter holds no value, the Java `null` shall be passed as `parameter`.

#### 8.3.2.2 TciParameterPassingModeType

`TciParameterPassingModeType` is mapped to the following interface:

```
// TCI IDL TciParameterPassingModeType
package org.etsi.ttcn.tci;
public interface TciParameterPassingMode {
    public final static int TCI_IN      = 0;
    public final static int TCI_INOUT   = 1;
    public final static int TCI_OUT     = 2;
}
```



## Constants:

- TCI\_IN Will be used to indicate that a TciParameter is an in parameter.
- TCI\_INOUT Will be used to indicate that a TciParameter is an inout parameter.
- TCI\_OUT Will be used to indicate that a TciParameter is an out parameter.

### 8.3.2.3 TciParameterListType

**TciParameterListType** is mapped to the following interface:

```
// TCI IDL TciParameterListType
package org.etsi.ttcn.tci;
public interface TciParameterList {
    public int size() ;
    public boolean isEmpty() ;
    public java.util.Enumeration getParameters() ;
    public TciParameter get(int index) ;
    public void clear() ;
    public void add(TciParameter parameter) ;
    public void setParameters(TciParameter[] parameters) ;
}
```

## Methods:

- size Returns the number of parameters in this list.
- isEmpty Returns true if this list contains no parameters.
- getParameters Returns an Enumeration over the parameters in the list. The enumeration provides the parameters in the same order as they appear in the list.
- get Returns the TciParameter at the specified position.
- clear Removes all parameters from this TciParameterList.
- add Adds parameter to the end of this TciParameterList.
- setParameter Fills this TciParameterList with parameters.

### 8.3.2.4 TciTypeClassType

**TciTypeClassType** is mapped to the following interface:

```
// TCI IDL TciTypeClassType
package org.etsi.ttcn.tci;
public interface TciTypeClass {
    public final static int ADDRESS = 0 ;
    public final static int ANYTYPE = 1 ;
    public final static int BITSTRING = 2 ;
    public final static int BOOLEAN = 3 ;
    public final static int CHARSTRING = 5 ;
    public final static int COMPONENT = 6 ;
    public final static int ENUMERATED = 7 ;
    public final static int FLOAT = 8 ;
    public final static int HEXSTRING = 9 ;
    public final static int INTEGER = 10 ;
    public final static int OCTETSTRING = 12 ;
    public final static int RECORD = 13 ;
    public final static int RECORD_OF = 14 ;
    public final static int ARRAY = 15 ;
    public final static int SET = 16 ;
    public final static int SET_OF = 17 ;
    public final static int UNION = 18 ;
    public final static int UNIVERSAL_CHARSTRING = 20 ;
    public final static int VERDICT = 21 ;
}
```

### 8.3.2.5 TciTestComponentKindType

**TciTestComponentKindType** is mapped to the following interface:

```
// TCI IDL TciTestComponentKindType
public interface TciTestComponentKind {
    public final static int TCI_CTRL_COMP      = 0;
    public final static int TCI_MTC_COMP      = 1;
    public final static int TCI_PTC_COMP      = 2;
    public final static int TCI_SYSTEM_COMP   = 3;
    public final static int TCI_ALIVE_COMP    = 4;
}

```

### 8.3.2.6 TciBehaviourIdType

**TciBehaviourIdType** is mapped to the following interface:

```
// TCI IDL TciBehaviourIdType
package org.etsi.ttcn.tci;
public interface TciBehaviourId extends QualifiedName {
}

```

### 8.3.2.7 TciTestCaseIdType

**TciTestCaseIdType** is mapped to the following interface:

```
// TCI IDL TciTestCaseIdType
package org.etsi.ttcn.tci;
public interface TciTestCaseId extends QualifiedName {
}

```

### 8.3.2.8 TciModuleIdType

**TciModuleIdType** is mapped to the following interface:

```
// TCI IDL TciModuleIdType
package org.etsi.ttcn.tci;
public interface TciModuleId extends QualifiedName {
}

```

### 8.3.2.9 TciModuleParameterIdType

**TciModuleParameterIdType** is mapped to the following interface:

```
// TCI IDL TciModuleParameterIdType
package org.etsi.ttcn.tci;
public interface TciModuleParameterId extends QualifiedName {
}

```

### 8.3.2.10 TciModuleParameterListType

**TciModuleParameterListType** is mapped to the following interface:

```
// TCI IDL TciModuleParameterListType
package org.etsi.ttcn.tci;
public interface TciModuleParameterList {
    public int size() ;
    public boolean isEmpty() ;
    public java.util.Enumeration getModuleParameters() ;
    public TciModuleParameter get(int index) ;
}

```

#### Methods:

- `size` Returns the number of module parameters in this list.
- `isEmpty` Returns `true` if this list contains no module parameters.
- `getModuleParameters` Returns an `Enumeration` over the module parameters in the list. The enumeration provides the module parameters in the same order as they appear in the list.
- `get` Returns the `TciModuleParameter` at the specified position.

### 8.3.2.11 TciModuleParameterType

**TciModuleParameterType** is mapped to the following interface:

```
// TCI IDL TciModuleParameterType
package org.etsi.ttcn.tci;
public interface TciModuleParameter {
    public String      getModuleParameterName();
    public Value      getDefaultValue();
}
```

#### Methods:

- `getModuleParameterName` Returns the module parameter name as defined in the TTCN-3 specification.
- `getDefaultValue` Returns the default Value module parameter of this TciModuleParameter, or the null object if the module parameter contains the distinct value null.

### 8.3.2.12 TciParameterTypeListType

**TciParameterTypeListType** is mapped to the following interface:

```
// TCI IDL TciParameterTypeListType
package org.etsi.ttcn.tci;
public interface TciParameterTypeList {
    public int      size();
    public boolean  isEmpty();
    public java.util.Enumeration getParameterTypes();
    public TciParameterType get(int index);
}
```

#### Methods:

- `size` Returns the number of parameter types in this list.
- `isEmpty` Returns true if this list contains no parameter types.
- `getParameterTypes` Returns an Enumeration over the parameter types in the list. The enumeration provides the parameter types in the same order as they appear in the list.
- `get` Returns the TciParameterType at the specified position.

### 8.3.2.13 TciParameterTypeType

**TciParameterTypeType** is mapped to the following interface:

```
// TCI IDL TciParameterTypeType
package org.etsi.ttcn.tci;
public interface TciParameterType {
    public Type      getParameterType();
    public int      getParameterPassingMode();
}
```

#### Methods:

- `getParameterType` Returns the Type of the parameter.
- `getParameterPassingMode` Returns the parameter passing mode of this parameter.

### 8.3.2.14 TciModuleIdListType

**TciModuleIdListType** is mapped to the following interface:

```
// TCI IDL TciModuleIdListType
package org.etsi.ttcn.tci;
public interface TciModuleIdList {
    public int      size();
}
```

```

    public boolean          isEmpty() ;
    public java.util.Enumeration tciGetImportedModules() ;
    public TciModuleId     get(int index) ;
}

```

### Methods:

- `size` Returns the number of modules in this list.
- `isEmpty` Returns `true` if this list contains no modules.
- `tciGetImportedModules` Returns an `Enumeration` over the modules in the list. The enumeration provides the modules in the same order as they appear in the list.
- `get` Returns the `TciModuleId` at the specified position.

### 8.3.2.15 TciTestCaseIdListType

`TciTestCaseIdType` is mapped to the following interface:

```

// TCI IDL TciTestCaseIdListType
package org.etsi.ttcn.tci;
public interface TciTestCaseIdList {
    public int          size() ;
    public boolean     isEmpty() ;
    public java.util.Enumeration tciGetTestCases() ;
    public TciTestCaseId get(int index) ;
}

```

### Methods:

- `size` Returns the number of test cases in this list.
- `isEmpty` Returns `true` if this list contains no test cases.
- `tciGetTestCases` Returns an `Enumeration` over the test cases in the list. The enumeration provides the test cases in the same order as they appear in the list.
- `get` Returns the `TciTestCaseId` at the specified position.

### 8.3.3 Abstract type mapping

The TTCN-3 data types are modelled in Java using the abstract type mapping as defined in this clause. The `Type` interface defines only operations used to retrieve in TTCN-3 defined types. No TTCN-3 types can be constructed using the `Type` interface. Types are modelled using the single interface `Type`, which provides methods to identify types and to retrieve values of a given type.

#### 8.3.3.1 Type

`Type` is mapped to the following interface:

```

// TCI IDL Type
package org.etsi.ttcn.tci;
public interface Type {
    public TciModuleId getDefiningModule ();
    public String      getName ();
    public int         getTypeClass ();
    public Value       newInstance ();
    public String      getTypeEncoding ();
    public String      getTypeEncodingVariant ();
    public String[]    getTypeExtension();
}

```

## Methods:

- `getDefiningModule` Returns the module identifier of the module the type has been defined in. If the type represents a TTCN-3 base type the distinct value `null` will be returned.
- `getName` Returns name of the type as defined in the TTCN-3 module.
- `getTypeClass` Returns the type class of the respective type. A value of `TciTypeClassType` can have one of the following constants: `ADDRESS`, `ANYTYPE`, `BITSTRING`, `BOOLEAN`, `CHARSTRING`, `COMPONENT`, `ENUMERATED`, `FLOAT`, `HEXSTRING`, `INTEGER`, `OCTETSTRING`, `RECORD`, `RECORD_OF`, `ARRAY`, `SET`, `SET_OF`, `UNION`, `UNIVERSAL_CHARSTRING`, `VERDICT`.
- `newInstance` Returns a freshly created value of the given type. This initial value of the created value is undefined.
- `getTypeEncoding` Returns the type encoding attribute as defined in the TTCN-3 module.
- `getTypeEncodingVariant` This operation returns the value encoding variant attribute as defined in the TTCN-3 module, if any. If no encoding variant attribute has been defined the distinct value `null` will be returned.
- `getTypeExtension` Returns the type extension attribute as defined in the TTCN-3 module.

### 8.3.4 Abstract value mapping

TTCN-3 values can be retrieved from the TE and constructed using the Value interface. The value mapping interface is constructed hierarchically with Value as the basic interface. Specialized interfaces for different types of values have been defined.

#### 8.3.4.1 Value

`value` is mapped to the following interface:

```
// TCI IDL Value
package org.etsi.ttcn.tci;
public interface Value {
    public Type    getType() ;
    public boolean notPresent() ;
    public String  getValueEncoding() ;
    public String  getValueEncodingVariant();
}
```

## Methods:

- `getType` Returns the type of the specified value.
- `notPresent` Returns `true` if the specified value is omit, `false` otherwise.
- `getValueEncoding` This operation returns the value encoding attribute as defined in the TTCN-3 module, if any. If no encoding attribute has been defined the distinct value `null` will be returned.
- `getValueEncodingVariant` This operation returns the value encoding variant attribute as defined in TTCN-3, if any. If no encoding variant attribute has been defined the distinct value `null` will be returned.

#### 8.3.4.2 IntegerValue

`IntegerValue` type is mapped to the following interface:

```
// IntegerValue
package org.etsi.ttcn.tci;
public interface IntegerValue {
    public void    setInteger(int value);
    public int     getInteger();
}

```

#### Methods:

- `setInteger`            Sets this IntegerValue to the int value `value`.
- `getInteger`            Returns the int value represented by this IntegerValue.

### 8.3.4.3 FloatValue

**FloatValue** type is mapped to the following interface:

```
// FloatValue
package org.etsi.ttcn.tci;
public interface FloatValue {
    public void    setFloat(float value);
    public float   getFloat();
}

```

#### Methods:

- `setFloat`            Sets this FloatValue to the float value `value`.
- `getFloat`            Returns the float value represented by this FloatValue.

### 8.3.4.4 BooleanValue

**BooleanValue** type is mapped to the following interface:

```
// BooleanValue
package org.etsi.ttcn.tci;
public interface BooleanValue {
    public void    setBoolean(boolean value);
    public boolean getBoolean();
}

```

#### Methods:

- `setBoolean`           Sets this BooleanValue to the boolean value `value`.
- `getBoolean`           Returns the boolean value represented by this BooleanValue.

### 8.3.4.5 CharstringValue

**CharstringValue** is mapped to the following interface:

```
// TCI IDL CharstringValue
package org.etsi.ttcn.tci;
public interface CharstringValue {
    String  getString ();
    void    setString (String value);
    char    getChar (int position);
    void    setChar (int position, char value);
    int     getLength ();
    void    setLength (int len);
}

```

#### Methods:

- `getString`           Returns the string of the TTCN-3 charstring. The textual representation of the empty TTCN-3 charstring is "", while its length is zero.
- `setString`           Sets this CharstringValue to `value`.

- `getChar` Returns the char value of the TTCN-3 charstring at position. position 0 denotes the first char of the TTCN-3 charstring. Valid values for position are 0 to length - 1.
- `setChar` Set the char at position to value. Valid values for position are 0 to length - 1.
- `getLength` Returns the length of this CharstringValue in chars, zero if the value of this CharstringValue is omit.
- `setLength` Sets the length of this CharstringValue in chars to len.

### 8.3.4.6 BitstringValue

**BitstringValue** is mapped to the following interface:

```
// TCI IDL BitstringValue
package org.etsi.ttcn.tci;
public interface BitstringValue {
    String getString ();
    void    setString (String value);
    int     getBit (int position);
    void    setBit (int position, int value);
    int     getLength ();
    void    setLength (int len);
}
```

#### Methods:

- `getString` Returns the textual representation of this BitstringValue, as defined in TTCN-3. E.g., the textual representation of 0101 is '0101'B. The textual representation of the empty TTCN-3 bitstring is ''B, while its length is zero.
- `setString` Sets the value of this BitstringValue according to the textual representation as defined by value. E.g., The value of this BitstringValue will be 0101 if the textual representation in value is '0101'B.
- `getBit` Returns the value (0 | 1) at position of this TTCN-3 bitstring. position 0 denotes the first bit of the TTCN-3 bitstring. Valid values for position are 0 to length - 1.
- `setBit` Set the bit at position to value (0 | 1). position 0 denotes the first bit in this BitstringValue. Valid values for position are 0 to length - 1.
- `getLength` Returns the length of this BitstringValue in bits, zero if the value of this BitstringValue is omit.
- `setLength` Sets the length of this BitstringValue in bits to len.

### 8.3.4.7 OctetstringValue

**OctetstringValue** is mapped to the following interface:

```
// TCI IDL OctetstringValue
package org.etsi.ttcn.tci;
public interface OctetstringValue {
    String getString ();
    void    setString (String value);
    int     getOctet (int position);
    void    setOctet (int position, int value);
    int     getLength ();
    void    setLength (int len);
}
```

## Methods:

- `getString` Returns the textual representation of this `OctetstringValue`, as defined in TTCN-3. E.g., the textual representation of `0xCAFFEE` is `'CAFFEE'`. The textual representation of the empty TTCN-3 octetstring is `''`, while its length is zero.
- `setString` Sets the value of this `OctetstringValue` according to the textual representation as defined by `value`. E.g., the value of this `OctetstringValue` will be `0xCAFFEE` if the textual representation in `value` is `'CAFFEE'`.
- `getOctet` Returns the value (0..255) at position of this TTCN-3 octetstring. position 0 denotes the first octet of the TTCN-3 octetstring. Valid values for position are 0 to `length - 1`.
- `setOctet` Set the octet at position to value (0..255). position 0 denotes the first octet in the octetstring. Valid values for position are 0 to `length - 1`.
- `getLength` Returns the length of this `OctetstringValue` in octets, zero if the value of this `OctetstringValue` is `omit`.
- `setLength` Sets the length of this `OctetstringValue` in octets to `len`.

### 8.3.4.8 UniversalCharstringValue

`UniversalCharstringValue` is mapped to the following interface:

```
// TCI IDL UniversalCharstringValue
package org.etsi.ttcn.tci;
public interface UniversalCharstringValue {
    String getString ();
    void   setString (String value);
    int    getChar  (int position);
    void   setChar  (int position, int value);
    int    getLength ();
    void   setLength (int len);
}
```

## Methods:

- `getString` Returns the textual representation of this `UniversalCharstringValue`, as defined in TTCN-3.
- `setString` Sets the value of this `UniversalCharstringValue` according to the textual representation as defined by `value`.
- `getChar` Returns the `UniversalChar` value of the TTCN-3 universal charstring at position. position 0 denotes the first `UniversalChar` of the TTCN-3 universal charstring. Valid values for position are 0 to `length - 1`.
- `setChar` Set the `UniversalChar` at position to `value`. Valid values for position are 0 to `length - 1`.
- `getLength` Returns the length of this `UniversalCharstringValue` in `UniversalChars`, zero if the value of this `UniversalCharstringValue` is `omit`.
- `setLength` Sets the length of this `UniversalCharstringValue` in `UniversalChars` to `len`.



### 8.3.4.9 HexstringValue

**HexstringValue** is mapped to the following interface:

```
// TCI IDL HexstringValue
package org.etsi.ttcn.tci;
public interface HexstringValue {
    String getString ();
    void setString (String value);
    int getHex (int position);
    void setHex (int position, int value);
    int getLength ();
    void setLength (int len);
}
```

#### Methods:

- `getString` Returns the textual representation of this `HexstringValue`, as defined in TTCN-3. E.g., the textual representation of `0xAFFEE` is `'AFFEE'H`. The textual representation of the empty TTCN-3 hexstring is `''H`, while its length is zero.
- `setString` Sets the value of this `HexstringValue` according to the textual representation as defined by `value`. E.g., the value of this `HexstringValue` will be `0xAFFEE` if the textual representation in `value` is `'AFFEE'H`.
- `getHex` Returns the value (`0...15`) at `position` of this TTCN-3 hexstring. `position 0` denotes the first hex digits of the TTCN-3 hexstring. Valid values for `position` are `0` to `length - 1`.
- `setHex` Set the hex digit at `position` to `value (0...16)`. `position 0` denotes the first octet in the hexstring. Valid values for `position` are `0` to `length - 1`.
- `getLength` Returns the length of this `HexstringValue` in octets, zero if the value of this `HexstringValue` is `omit`.
- `setLength` Sets the length of this `HexstringValue` in hex digits to `len`.

### 8.3.4.10 RecordValue

**RecordValue** is mapped to the following interface:

```
// TCI IDL RecordValue
package org.etsi.ttcn.tci;
public interface RecordValue {
    public Value getField(String fieldName) ;
    public void setField(String fieldName, Value value) ;
    public String[] getFieldNames() ;
    public void setFieldOmitted(String fieldName)
}
}
```

#### Methods:

- `getField` Returns the value of the field named `fieldName`. The return value is the common abstract base type `Value`, as a record field can have any type defined in TTCN-3. If the field cannot be obtained from the record the distinct value `null` will be returned.
- `setField` Set the field named `fieldName` of the record to `value`. No assumption shall be made on how a field is stored in a record. An internal implementation might choose to store a reference to this value or to copy the value. It is safe to assume that the value will be

copied. Therefore it should be assumed that subsequent modifications of `value` will not be considered in the record.

- `getFieldNames` Returns an array of `String` of field names, the empty sequence, if the record has no fields.
- `setFieldOmitted` Set the field named `fieldName` of the record to `omit`.

### 8.3.4.11 RecordOfValue

`RecordOfValue` is mapped to the following interface:

```
// TCI IDL RecordOfValue
package org.etsi.ttcn.tci;
public interface RecordOfValue {
    public Value    getField(String fieldName) ;
    public void    setField(int position, Value value) ;
    public void    appendField(Value value) ;
    public Type    getElementType() ;
    public int     getLength() ;
    public void    setLength(int len) ;
    public int     getOffset() ;
}
```

#### Methods:

- `getField` Returns the value of the record `of` at position `position` if `position` is between zero and `length - 1`, the distinct value `null` otherwise. The return value is the common abstract base type `Value`, as a record `of` can have fields of any type defined in TTCN-3.
- `setField` Sets the field at position `position` to `value`. If `position` is greater than `(length - 1)` the record `of` will be extended to have the length `(position + 1)`. The record `of` elements between the original position at `length` and `position - 1` will be set to `OMIT`. No assumption shall be made on how a field is stored in a record `of`. An internal implementation might choose to store a reference to this value or to copy the value. It is safe to assume that the value will be copied. Therefore it should be assumed that subsequent modifications of `value` will not be considered in the record `of`.
- `appendField` Appends the value at the end of the record `of`, i.e., at position `length`. No assumption shall be made on how a field is stored in a record `of`. An internal implementation might choose to store a reference to this value or to copy the value. It is safe to assume that the value will be copied. Therefore it should be assumed that subsequent modifications of `value` will not be considered in the record `of`.
- `getElementType` This operation will return the `Type` of the elements of this record `of`.
- `getLength` Returns the actual length of the record `of` value, zero if the record `of` value is `OMIT`.
- `setLength` Set the length of the record `of` to `len`. If `len` is greater than the original length, newly created elements have the value `OMIT`. If `len` is less or equal than the original length this operation will be ignored.
- `getOffset` Returns the lowest possible index. For a record `of` or set of value this is always 0. For an array value, this is the lower index bound used in the type definition.

### 8.3.4.12 UnionValue

**UnionValue** is mapped to the following interface:

```
// TCI IDL UnionValue
package org.etsi.ttcn.tci;
public interface UnionValue {
    Value      getVariant (String variantName);
    void       setVariant (String variantName, Value value);
    String     getPresentVariantName ();
    String[]   getVariantNames ();
}
```

#### Methods:

- **getVariant** Returns the value of the TTCN-3 union `variantName`, if `variantName` equals the result of `getPresentVariantName`, the distinct value `null` otherwise. `variantName` denotes the name of the union variant as defined in the TTCN-3 module.
- **setVariant** Sets `variantName` of the union to `value`. If `variantName` is not defined for this union this operation will be ignored. If another variant was selected the new variant will be selected instead.
- **getPresentVariantName** Returns the variant name that has a value in this union set as a `String`. The distinct value `null` will be returned if no variant is selected.
- **getVariantNames** Returns an array of `String` of variant names, the empty sequence, if the union has no fields. If the `UnionValue` represents the TTCN-3 `anytype`, i.e., the type class of the type obtained by `getType()` is `ANYTYPE`, all predefined and user-defined TTCN-3 types will be returned.

### 8.3.4.13 EnumeratedValue

**EnumeratedValue** is mapped to the following interface:

```
// TCI IDL EnumeratedValue
package org.etsi.ttcn.tci;
public interface EnumeratedValue {
    String  getEnum ();
    void    setEnum (String enumValue);
    int     getInt ();
    void    setInt (int intValue);
}
```

#### Methods:

- **getEnum** Returns the string identifier of this `EnumeratedValue`. This identifier equals the identifier in the TTCN-3 specification.
- **setEnum** Set the enum to `enumValue`. If `enumValue` is not an allowed value for this enumeration the operation will be ignored.
- **getInt** Returns the integer value of this `EnumeratedValue`. This integer equals the user-assigned integer value in the TTCN-3 specification or the automatically assigned integer value.
- **setInt** Sets the integer value of this `EnumeratedValue`. This integer should equal the user-assigned integer value in the TTCN-3 specification or the automatically assigned integer value. If `intValue` is not an allowed value for this enumeration the operation is ignored.

### 8.3.4.14 VerdictValue

**VerdictValue** is mapped to the following interface:

```
// TCI IDL VerdictValue
package org.etsi.ttcn.tci;
public interface VerdictValue {
    public static final int NONE = 0;
    public static final int PASS = 1;
    public static final int INCONC = 2;
    public static final int FAIL = 3;
    public static final int ERROR = 4;

    public int    getVerdict() ;
    public void   setVerdict(int verdict) ;
}
```

#### Methods:

- `getVerdict` Returns the integer value for this `VerdictValue`. The integer is one of the following constants: `ERROR`, `FAIL`, `INCONC`, `NONE`, `PASS`, `USER_ERROR`.
- `setVerdict` Sets this `VerdictValue` to `verdict`. Note that a `VerdictValue` can be set to any of the above mentioned verdicts at any time. The `VerdictValue` does not perform any verdict calculations as defined in TTCN-3. For example, it is legal to set the `VerdictValue` first to `INCONC` and then to `PASS`.

### 8.3.4.15 AddressValue

**AddressValue** is mapped to the following interface:

```
// TCI IDL Address_Value
> package org.etsi.ttcn.tci;
> public interface AddressValue {
> public int getAddress() ;
> public void setAddress(Value value) ;
> }
>
```

#### Methods:

- `getAddress` Returns the value represented by this `AddressValue`.
- `setAddress` Sets this `AddressValue` to the value `value`.

## 8.3.5 Abstract logging types mapping

Additional types are defined to ease the logging of matches between values and templates.

### 8.3.5.1 TciValueTemplate

**TciValueTemplate** is mapped to the following interface:

```
// TCI IDL TciValueTemplate
package org.etsi.ttcn.tci;
public interface TciValueTemplate {
    public boolean isOmit();
    public boolean isAny();
    public boolean isAnyOrOmit();
    public String getTemplateDef();
}
```

#### Methods:

- `isOmit` Returns `true` if the template is `omit`, `false` otherwise.
- `isAny` Returns `true` if the template is `any`, `false` otherwise.

- `isAnyOrOmit` Returns `true` if the template is `anyoromit`, `false` otherwise.
- `getTemplateDef` This operation returns the template definition.

### 8.3.5.2 TciNonValueTemplate

**TciNonValueTemplate** is mapped to the following interface:

```
// TCI IDL TciNonValueTemplate
package org.etsi.ttcn.tci;
public interface TciNonValueTemplate {
    public boolean isAny();
    public boolean isAll();
    public String getTemplateDef();
}
```

#### Methods:

- `isAny` Returns `true` if the template is `any`, `false` otherwise.
- `isAll` Returns `true` if the template is `all`, `false` otherwise.
- `getTemplateDef` This operation returns the template definition.

### 8.3.5.3 TciValueList

**TciValueList** is mapped to the following interface:

```
// TCI IDL TciValueList
package org.etsi.ttcn.tci;
public interface TciValueList{
    public int size() ;
    public boolean isEmpty() ;
    public Value get(int index) ;
}
```

#### Methods:

- `size` Returns the number of values in this list.
- `isEmpty` Returns `true` if this list contains no values.
- `get` Returns the `Value` at the specified position.

### 8.3.5.4 TciValueDifference

**TciValueDifference** is mapped to the following interface:

```
// TCI IDL TciValueDifference
package org.etsi.ttcn.tci;
public interface TciValueDifference {
    public Value getValue();
    public TciValueTemplate getTciValueTemplate();
    public String getDescription()
}
```

#### Methods:

- `getValue` Returns the value of this `TciValueDifference`.
- `getTciValueTemplate` Returns the template of this `TciValueDifference`.
- `getDescription` Returns the description of the mismatch.

### 8.3.5.5 TciValueDifferenceList

**TciValueDifferenceList** is mapped to the following interface:

```
// TCI IDL TciValueDifferenceList
package org.etsi.ttcn.tci;
public interface TciValueDifferenceList{
    public int size() ;
    public boolean isEmpty() ;
    public TciValueDifference get(int index) ;
}
```

```
}
```

### Methods:

- `size` Returns the number of differences in this list.
- `isEmpty` Returns `true` if this list contains no differences.
- `get` Returns the `TciValueDifference` at the specified position.

#### 8.3.5.6 ComponentStatus

**ComponentStatus** is mapped to the following interface:

```
// TCI IDL ComponentStatus
package org.etsi.ttcn.tci;
public interface ComponentStatus {
    public static final int INACTIVE_C = 0;
    public static final int RUNNING_C = 1;
    public static final int STOPPED_C = 2;
    public static final int KILLED_C = 3;
    public static final int NULL_C = 4;

    public int      getComponentStatus() ;
    public void     setComponentStatus (int componentStatus) ;
}
```

#### 8.3.5.7 TimerStatus

**TimerStatus** is mapped to the following interface:

```
// TCI IDL TimerStatus
package org.etsi.ttcn.tci;
public interface TimerStatus {
    public static final int RUNNING_T = 0;
    public static final int INACTIVE_T = 1;
    public static final int EXPIRED_T = 2;
    public static final int NULL_T = 3;

    public int      getTimerStatus() ;
    public void     setTimerStatus (int timerStatus) ;
}
```

#### 8.3.5.8 TciStatus

**TciStatus** is mapped to the following interface:

```
// TCI IDL TciStatus
package org.etsi.ttcn.tci;
public interface TciStatus {
    public static final int TCI_OK = 0;
    public static final int TCI_ERROR = -1;

    public int      getTciStatus() ;
    public void     setTciStatus (int tciStatus) ;
}
```

### 8.4 Constants

Within this Java language mapping constants have been specified. All constants are defined `public final static` and are accessible from every object from every package. The constants defined within this clause are not defined with the IDL clause. Instead they result from the specification of the TCI IDL types marked as native.

The following constants can be used to determine the parameter passing mode of TTCN-3 parameters (see also clause 8.3.2.3).

- `org.etsi.ttcn.tci.TciParameterPassingMode.TCI_IN;`
- `org.etsi.ttcn.tci.TciParameterPassingMode.TCI_INOUT;`

- `org.etsi.ttcn.tci.TciParameterPassingMode.TCI_OUT`.

For the distinct parameter value `null`, the encoded parameter value shall be set to Java `null`.

The following constants shall be used for value handling (see also clause 8.3.2.4).

- `org.etsi.ttcn.tci.TciTypeClass.ADDRESS`;
- `org.etsi.ttcn.tci.TciTypeClass.ANYTYPE`;
- `org.etsi.ttcn.tci.TciTypeClass.BITSTRING`;
- `org.etsi.ttcn.tci.TciTypeClass.BOOLEAN`;
- `org.etsi.ttcn.tci.TciTypeClass.CHARSTRING`;
- `org.etsi.ttcn.tci.TciTypeClass.COMPONENT`;
- `org.etsi.ttcn.tci.TciTypeClass.ENUMERATED`;
- `org.etsi.ttcn.tci.TciTypeClass.FLOAT`;
- `org.etsi.ttcn.tci.TciTypeClass.HEXSTRING`;
- `org.etsi.ttcn.tci.TciTypeClass.INTEGER`;
- `org.etsi.ttcn.tci.TciTypeClass.OCTETSTRING`;
- `org.etsi.ttcn.tci.TciTypeClass.RECORD`;
- `org.etsi.ttcn.tci.TciTypeClass.RECORD_OF`;
- `org.etsi.ttcn.tci.TciTypeClass.SET`;
- `org.etsi.ttcn.tci.TciTypeClass.SET_OF`;
- `org.etsi.ttcn.tci.TciTypeClass.ARRAY`;
- `org.etsi.ttcn.tci.TciTypeClass.UNION`;
- `org.etsi.ttcn.tci.TciTypeClass.UNIVERSAL_CHARSTRING`;
- `org.etsi.ttcn.tci.TciTypeClass.VERDICT`.

The following constants shall be used for component handling (see also clause 8.3.2.5).

- `org.etsi.ttcn.tci.TciTestComponentKind.TCI_CTRL_COMP`;
- `org.etsi.ttcn.tci.TciTestComponentKind.TCI_MTC_COMP`;
- `org.etsi.ttcn.tci.TciTestComponentKind.TCI_PTC_COMP`;
- `org.etsi.ttcn.tci.TciTestComponentKind.TCI_SYSTEM_COMP`;
- `org.etsi.ttcn.tci.TciTestComponentKind.TCI_ALIVE_COMP`.

The following constants shall be used for component status (see also clause 8.3.5.6).

- `org.etsi.ttcn.tci.ComponentStatus.INACTIVE_C`;
- `org.etsi.ttcn.tci.ComponentStatus.RUNNING_C`;
- `org.etsi.ttcn.tci.ComponentStatus.STOPPED_C`;
- `org.etsi.ttcn.tci.ComponentStatus.KILLED_C`;
- `org.etsi.ttcn.tci.ComponentStatus.NULL_C`.

The following constants shall be used for timer status (see also clause 8.3.5.7).

- `org.etsi.ttcn.tci.TimerStatus.RUNNING_T`;
- `org.etsi.ttcn.tci.TimerStatus.INACTIVE_T`;
- `org.etsi.ttcn.tci.TimerStatus.EXPIRED_T`;
- `org.etsi.ttcn.tci.TimerStatus.NULL_T`.

The following constants shall be used for TCI status (see also clause 8.3.5.8).

- `org.etsi.ttcn.tci.TciStatus.TCI_OK`;
- `org.etsi.ttcn.tci.TciStatus.TCI_ERROR`.

## 8.5 Mapping of interfaces

The TCI IDL definition defines four interfaces, the **TCI-TM**, the **TCI-CH**, the **TCI-CD**, and the **TCI-TL** interface. The operations are defined for different directions within this interface, i.e., some operations can only be called by the TTCN-3 executable (TE), the system adaptor (SA) or the platform adaptor (PA) on the test management and control (TMC) while others can only be called by the TMC on the TE. This is reflected by dividing the TCI IDL interfaces in two sub interfaces, each suffixed by Required or Provided.

**Table 4 – Sub interfaces**

Calling/ Called	TE	TM	CD	CH	SA	PA	TL
TE	–	TCI-TM Provided	TCI-CD Provided	TCI-CH Provided	See [3]	See [3]	TCI-TL Provided
TM	TCI-TM Required	–	–	–	–	–	TCI-TL Provided
CD	TCI-CD Required	–	–	–	–	–	TCI-TL Provided
CH	TCI-CH Required	–	–	–	–	–	TCI-TL Provided
SA	See [3]	–	–	–	–	–	TCI-TL Provided
PA	See [3]	–	–	–	–	–	TCI-TL Provided
TL	–	–	–	–	–	–	–

All methods defined in these interfaces should behave as defined in clause 7.

### 8.5.1 The TCI-TM interface

#### 8.5.1.1 TCI-TM provided

The TCI-TM Provided interface is mapped to the following interface:

```
// TCI-TM
// TE -> TM
package org.etsi.ttcn.tci;
public interface TciTMProvided {
    public void tciTestCaseStarted (TciTestCaseId testCaseId, TciParameterList parameterList, Float
timer);
    public void tciTestCaseTerminated (VerdictValue verdict, TciParameterList parameterList);
    public void tciControlTerminated ();
    public Value tciGetModulePar (TciModuleParameterId parameterId);
    public void tciLog (TriComponentId testComponentId, String message);
    public void tciError (String message);
}
```

#### 8.5.1.2 TCI-TM required

The TCI-TM Required interface is mapped to the following interface:

```
// TCI-TM
// TM -> TE
package org.etsi.ttcn.tci;
public interface TciTMRequired {
    public void tciRootModule (TciModuleId moduleName) ;
    public TciModuleIdList tciGetImportedModules ();
    public TciModuleParameterList tciGetModuleParameters (TciModuleId moduleId);
    public TciTestCaseIdList tciGetTestCases ();
    public TciParameterTypeList tciGetTestCaseParameters (TciTestCaseId testCaseId);
    public TriPortIdList tciGetTestCaseTSI (TciTestCaseId testCaseId);
}
```



```

    public void                tciStartTestCase
                               (String testCaseId, TciParameterList parameterList ) ;
    public void                tciStopTestCase () ;
    public TriComponentId      tciStartControl () ;
    public void                tciStopControl () ;
}

```

## 8.5.2 The TCI-CD interface

### 8.5.2.1 TCI-CD provided

The TCI-CD Provided interface is mapped to the following interface:

```

// TCI-CD
// TE -> CD
package org.etsi.ttcn.tci;
public interface TciCDProvided {
    public Value        decode (TriMessage message, Type decodingHypothesis );
    public TriMessage   encode (Value value);
}

```

### 8.5.2.2 TCI-CD required

The TCI-CD Required interface is mapped to the following interface:

```

// TCI-CD
// CD -> TE
package org.etsi.ttcn.tci;
public interface TciCDRequired {
    public Type        getTypeForName (String typeName);
    public Type        getInteger ();
    public Type        getFloat ();
    public Type        getBoolean ();
    public Type        getCharstring ();
    public Type        getUniversalCharstring ();
    public Type        getHexstring ();
    public Type        getBitstring ();
    public Type        getOctetstring ();
    public Type        getVerdict ();
    public void        tciErrorReq (String message);
}

```

## 8.5.3 The TCI-CH interface

### 8.5.3.1 TCI-CH provided

The TCI-CH Provided interface is mapped to the following interface:

```

// TciCHProvided
// TE -> CH
package org.etsi.ttcn.tci;
public interface TciCHProvided {
    public void tciSendConnected (TriPortId sender, TriComponentId receiver, Value sendMessage) ;
    public void tciSendConnectedBC (TriPortId sender, Value sendMessage) ;
    public void tciSendConnectedMC (TriPortId sender, TriComponentIdList receivers,
                                   Value sendMessage) ;

    public void tciCallConnected(TriPortId sender,
                                TriComponentId receiver,
                                TriSignatureId signature,
                                TciParameterList parameterList) ;
    public void tciCallConnectedBC(TriPortId sender,
                                   TriSignatureId signature,
                                   TciParameterList parameterList) ;
    public void tciCallConnectedMC(TriPortId sender,
                                   TriComponentIdList receivers,
                                   TriSignatureId signature,
                                   TciParameterList parameterList) ;

    public void tciReplyConnected(TriPortId sender,
                                  TriComponentId receiver,
                                  TriSignatureId signature,
                                  TciParameterList parameterList,

```

```

        Value returnValue) ;
public void    tciReplyConnectedBC(TriPortId sender,
        TriSignatureId signature,
        TciParameterList parameterList,
        Value returnValue) ;
public void    tciReplyConnectedMC(TriPortId sender,
        TriComponentIdList receivers,
        TriSignatureId signature,
        TciParameterList parameterList,
        Value returnValue) ;

public void    tciRaiseConnected(TriPortId sender,
        TriComponentId receiver,
        TriSignatureId signature,
        Value except) ;
public void    tciRaiseConnectedBC(TriPortId sender,
        TriSignatureId signature,
        Value except) ;
public void    tciRaiseConnectedMC(TriPortId sender,
        TriComponentIdList receivers,
        TriSignatureId signature,
        Value except) ;

public TriComponentId    tciCreateTestComponentReq(int kind,
        Type componentType,
        String name,
        Value hostId) ;

public void    tciStartTestComponentReq(TriComponentId comp,
        TciBehaviourId behaviour,
        TciParameterList parameterList) ;

public void    tciStopTestComponentReq(TriComponentId comp) ;

public void    tciConnectReq(TriPortId fromPort, TriPortId toPort) ;

public void    tciDisconnectReq(TriPortId fromPort, TriPortId toPort) ;

public void    tciTestComponentTerminatedReq(TriComponentId comp, VerdictValue verdict) ;

public boolean tciTestComponentRunningReq(TriComponentId comp) ;

public TriComponentId    tciGetMTCReq() ;

public void    tciMapReq(TriPortId fromPort, TriPortId toPort);

public void    tciMapParamReq(TriPortId fromPort, TriPortId toPort,
        TciParameterList parameterList);

public void    tciUnmapReq(TriPortId fromPort, TriPortId toPort);

public void    tciUnmapParamReq(TriPortId fromPort, TriPortId toPort,
        TciParameterList parameterList);

public void    tciExecuteTestCaseReq(TriComponentId testComponentId,
        TriPortIdList tsiPortList);

public void    tciResetReq() ;

public boolean tciTestComponentDoneReq(TriComponentId testComponentId) ;

public void    tciKillTestComponentReq(TriComponentId component)

public boolean tciTestComponentAliveReq (TriComponentId component)

public boolean tciTestComponentKilledReq (TriComponentId component)
}

```

### 8.5.3.2 TCI-CH required

The TCI-CH Required interface is mapped to the following interface:

```

// TciCHRequired
// CH -> TE
package org.etsi.ttcn.tci;
public interface TciCHRequired extends TciCDRequired {
    public void    tciEnqueueMsgConnected(TriPortId sender,
        TriComponentId receiver,

```

```

        Value receivedMessage) ;

public void    tciEnqueueCallConnected(TriPortId sender,
                                       TriComponentId receiver,
                                       TriSignatureId signature,
                                       TciParameterList parameterList) ;

public void    tciEnqueueReplyConnected(TriPortId sender,
                                       TriComponentId receiver,
                                       TriSignatureId signature,
                                       TciParameterList parameterList,
                                       Value returnValue) ;

public void    tciEnqueueRaiseConnected(TriPortId sender,
                                       TriComponentId receiver,
                                       TriSignatureId signature,
                                       Value except) ;

public TriComponentId    tciCreateTestComponent(int kind, Type componentType, String name) ;

public void    tciStartTestComponent (TriComponentId comp,
                                       TciBehaviourId behaviour,
                                       TciParameterList parameterList) ;

public void    tciStopTestComponent (TriComponentId comp) ;

public void    tciConnect (TriPortId fromPort, TriPortId toPort) ;

public void    tciDisconnect (TriPortId fromPort, TriPortId toPort) ;

public void    tciTestComponentTerminated (TriComponentId comp, VerdictValue verdict);

public boolean tciTestComponentRunning (TriComponentId comp);

public boolean tciTestComponentDone (TriComponentId comp);

public TriComponentId    tciGetMTC ();

public void    tciExecuteTestCase (TciTestCaseId testCaseId, TriPortIdList tsiPortList);

public void    tciReset ();

public void    tciMap (TriPortId fromPort, TriPortId toPort);

public void    tciMapParam (TriPortId fromPort, TriPortId toPort,
                           TciParameterList parameterList);

public void    tciUnmap (TriPortId fromPort, TriPortId toPort);

public void    tciUnmapParam (TriPortId fromPort, TriPortId toPort,
                              TciParameterList parameterList);

public void    tciKillTestComponent (TriComponentId component)

public boolean tciTestComponentAlive (TriComponentId component)

public boolean tciTestComponentKilled (TriComponentId component)
}

```

## 8.5.4 The TCI-TL interface

### 8.5.4.1 TCI-TL provided

The TCI-TL Provided interface is mapped to the following interface:

```

// TCI-TL
// TE, TM,CH,CD, SA,PA -> TL
package org.etsi.ttcn.tci;
public interface TciTLProvided {
    public void tliTcExecute(String am, int ts, String src, int line, TriComponentId c,
                           TciTestCaseId tcId, TriParameterList triPars, TriTimerDuration dur);
    public void tliTcStart(String am, int ts, String src, int line, TriComponentId c,
                          TciTestCaseId tcId, TciParameterList tciPars, TriTimerDuration dur);
    public void tliTcStop(String am, int ts, String src, int line, TriComponentId c, String reason);
    public void tliTcStarted(String am, int ts, String src, int line, TriComponentId c,
                            TciTestCaseId tcId, TciParameterList tciPars, TriTimerDuration dur);
    public void tliTcTerminated(String am, int ts, String src, int line, TriComponentId c,
                               TciTestCaseId tcId, TciParameterList tciPars, VerdictValue verdict, String reason);
}

```

```

public void tliCtrlStart(String am, int ts, String src, int line, TriComponentId c);
public void tliCtrlStop(String am, int ts, String src, int line, TriComponentId c);
public void tliCtrlTerminated(String am, int ts, String src, int line, TriComponentId c);
public void tliMSend_m(String am, int ts, String src, int line, TriComponentId c,
    TriPortId at, TriPortId to, Value msgValue, Value addrValue,
    TciStatus encoderFailure, TriMessage msg, TriAddress address,
    TriStatus transmissionFailure);
public void tliMSend_m_BC(String am, int ts, String src, int line, TriComponentId c,
    TriPortId at, TriPortId to, Value msgValue
    TciStatus encoderFailure, TriMessage msg, TriStatus transmissionFailure) ;
public void tliMSend_m_MC(String am, int ts, String src, int line, TriComponentId c,
    TriPortId at, TriPortId to, Value msgValue, TciValueList addrValues,
    TciStatus encoderFailure, TriMessage msg, TriAddressList addresses,
    TriStatus transmissionFailure);
public void tliMSend_c(String am, int ts, String src, int line, TriComponentId c,
    TriPortId at, TriPortId to, Value msgValue, TriStatus transmissionFailure);
public void tliMSend_c_BC(String am, int ts, String src, int line, TriComponentId c,
    TriPortId at, TriPortIdList to, Value msgValue, TriStatus transmissionFailure);
public void tliMSend_c_MC(String am, int ts, String src, int line, TriComponentId c,
    TriPortId at, TriPortIdList to, Value msgValue, TriStatus transmissionFailure);
public void tliMDetected_m(String am, int ts, String src, int line, TriComponentId c,
    TriPortId at, TriPortId from, TriMessage msg, TriAddress address);
public void tliMDetected_c(String am, int ts, String src, int line, TriComponentId c,
    TriPortId at, TriPortId from, Value msgValue );
public void tliMMismatch_m(String am, int ts, String src, int line, TriComponentId c,
    TriPortId at, Value msgValue, TciValueTemplate msgTmpl, TciValueDifferenceList diffs,
    Value addrValue, TciValueTemplate addressTmpl);
public void tliMMismatch_c(String am, int ts, String src, int line, TriComponentId c,
    TriPortId at, Value msgValue, TciValueTemplate msgTmpl, TciValueDifferenceList diffs,
    TriComponentId from, TciNonValueTemplate fromTmpl);
public void tliMReceive_m(String am, int ts, String src, int line, TriComponentId c,
    TriPortId at, Value msgValue, TciValueTemplate msgTmpl,
    Value addrValue, TciValueTemplate addressTmpl);
public void tliMReceive_c(String am, int ts, String src, int line, TriComponentId c,
    TriPortId at, Value msgValue, TciValueTemplate msgTmpl,
    TriComponentId from, TciNonValueTemplate fromTmpl);
public void tliPrCall_m(String am, int ts, String src, int line, TriComponentId c,
    TriPortId at, TriPortId to, TriSignatureId signature, TciParameterList tciPars,
    Value addrValue, TciStatus encoderFailure, TriParameterList triPars,
    TriAddress address, TriStatus transmissionFailure);
public void tliPrCall_m_BC(String am, int ts, String src, int line, TriComponentId c,
    TriPortId at, TriPortId to, TriSignatureId signature, TciParameterList tciPars,
    TciStatus encoderFailure, TriParameterList triPars,
    TriStatus transmissionFailure);
public void tliPrCall_m_MC(String am, int ts, String src, int line, TriComponentId c,
    TriPortId at, TriPortId to, TriSignatureId signature, TciParameterList tciPars,
    TciValueList addrValues, TciStatus encoderFailure, TriParameterList triPars,
    TriAddressList addresses, TriStatus transmissionFailure);
public void tliPrCall_c(String am, int ts, String src, int line, TriComponentId c,
    TriPortId at, TriPortId to, TriSignatureId signature, TciParameterList tciPars,
    TriStatus transmissionFailure);
public void tliPrCall_c_BC(String am, int ts, String src, int line, TriComponentId c,
    TriPortId at, TriPortIdList to, TriSignatureId signature, TciParameterList tciPars,
    TriStatus transmissionFailure);
public void tliPrCall_c_MC(String am, int ts, String src, int line, TriComponentId c,
    TriPortId at, TriPortIdList to, TriSignatureId signature, TciParameterList tciPars,
    TriStatus transmissionFailure);
public void tliPrGetCallDetected_m(String am, int ts, String src, int line, TriComponentId c,
    TriPortId at, TriPortId from, TriSignatureId signature, TriParameterList triPars,
    TriAddress address);
public void tliPrGetCallDetected_c(String am, int ts, String src, int line, TriComponentId c,
    TriPortId at, TriPortId from, TriSignatureId signature, TciParameterList tciPars );
public void tliPrGetCallMismatch_m(String am, int ts, String src, int line, TriComponentId c,
    TriPortId at, TriSignatureId signature,
    TciParameterList tciPars, TciValueTemplate parsTmpl, TciValueDifferenceList diffs,
    Value addrValue, TciValueTemplate addressTmpl);
public void tliPrGetCallMismatch_c(String am, int ts, String src, int line, TriComponentId c,
    TriPortId at, TriSignatureId signature,
    TciParameterList tciPars, TciValueTemplate parsTmpl, TciValueDifferenceList diffs,
    TriComponentId from, TciNonValueTemplate fromTmpl);
public void tliPrGetCall_m(String am, int ts, String src, int line, TriComponentId c,
    TriPortId at, TriSignatureId signature,
    TciParameterList tciPars, TciValueTemplate parsTmpl,
    Value addrValue, TciValueTemplate addressTmpl);
public void tliPrGetCall_c(String am, int ts, String src, int line, TriComponentId c,
    TriPortId at, TriSignatureId signature,
    TciParameterList tciPars, TciValueTemplate parsTmpl,
    TriComponentId from, TciNonValueTemplate fromTmpl);
public void tliPrReply_m(String am, int ts, String src, int line, TriComponentId c,
    TriPortId at, TriPortId to, TriSignatureId signature, TciParameterList tciPars,

```

```

        Value replValue, Value addrValue,
        TciStatus encoderFailure, TriParameterList triPars,
        TriParameter repl, TriAddress address, TriStatus transmissionFailure);
public void tliPrReply_m_BC(String am, int ts, String src, int line, TriComponentId c,
    TriPortId at, TriPortId to, TriSignatureId signature, TciParameterList tciPars,
    Value replValue, TciStatus encoderFailure,
    TriParameterList triPars, TriParameter repl, TriStatus transmissionFailure);
public void tliPrReply_m_MC(String am, int ts, String src, int line, TriComponentId c,
    TriPortId at, TriPortId to, TriSignatureId signature, TciParameterList tciPars,
    Value replValue, TciValueList addrValues,
    TciStatus encoderFailure, TriParameterList triPars,
    TriParameter repl, TriAddressList addresses, TriStatus transmissionFailure);
public void tliPrReply_c(String am, int ts, String src, int line, TriComponentId c,
    TriPortId at, TriPortId to, TriSignatureId signature,
    TciParameterList tciPars, Value replValue,
    TriStatus transmissionFailure);
public void tliPrReply_c_BC(String am, int ts, String src, int line, TriComponentId c,
    TriPortId at, TriPortIdList to, TriSignatureId signature, TciParameterList tciPars,
    Value replValue, TriStatus transmissionFailure);
public void tliPrReply_c_MC(String am, int ts, String src, int line, TriComponentId c,
    TriPortId at, TriPortIdList to, TriSignatureId signature, TciParameterList tciPars,
    Value replValue, TriStatus transmissionFailure);
public void tliPrGetReplyDetected_m(String am, int ts, String src, int line, TriComponentId c,
    TriPortId at, TriPortId from, TriSignatureId signature, in TriParameterListType triPars,
    TriParameter repl, TriAddress address);
public void tliPrGetReplyDetected_c(String am, int ts, String src, int line, TriComponentId c,
    TriPortId at, TriPortId from, TriSignatureId signature, in TciParameterList tciPars,
    Value replValue);
public void tliPrGetReplyMismatch_m(String am, int ts, String src, int line, TriComponentId c,
    TriPortId at, TriSignatureId signature,
    TciParameterList tciPars, in TciValueTemplate parsTmpl,
    Value replValue, TciValueTemplate replyTmpl, TciValueDifferenceList diffs,
    Value addrValue, TciValueTemplate addressTmpl);
public void tliPrGetReplyMismatch_c(String am, int ts, String src, int line, TriComponentId c,
    TriPortId at, TriSignatureId signature,
    TciParameterList tciPars, in TciValueTemplate parsTmpl,
    Value replValue, TciValueTemplate replyTmpl, TciValueDifferenceList diffs,
    TriComponentId from, TciNonValueTemplate fromTmpl);
public void tliPrGetReply_m(String am, int ts, String src, int line, TriComponentId c,
    TriPortId at, TriSignatureId signature,
    TciParameterList tciPars, in TciValueTemplate parsTmpl,
    Value replValue, TciValueTemplate replyTmpl,
    Value addrValue, TciValueTemplate addressTmpl);
public void tliPrGetReply_c(String am, int ts, String src, int line, TriComponentId c,
    TriPortId at, TriSignatureId signature,
    TciParameterList tciPars, in TciValueTemplate parsTmpl,
    Value replValue, TciValueTemplate replyTmpl,
    TriComponentId from, TciNonValueTemplate fromTmpl);
public void tliPrRaise_m(String am, int ts, String src, int line, TriComponentId c,
    TriPortId at, TriPortId to,
    TriSignatureId signature, TciParameterList tciPars, Value excValue,
    Value addrValue, TciStatus encoderFailure, TriException exc,
    TriAddress address, TriStatus transmissionFailure);
public void tliPrRaise_m_BC(String am, int ts, String src, int line, TriComponentId c,
    TriPortId at, TriPortId to,
    TriSignatureId signature, TciParameterList tciPars, Value excValue,
    TciStatus encoderFailure, TriException exc, TriStatus transmissionFailure);
public void tliPrRaise_m_MC(String am, int ts, String src, int line, TriComponentId c,
    TriPortId at, TriPortId to,
    TriSignatureId signature, TciParameterList tciPars, Value excValue,
    TciValueList addrValues, TciStatus encoderFailure, TriException exc,
    TriAddressList addresses, TriStatus transmissionFailure);
public void tliPrRaise_c(String am, int ts, String src, int line, TriComponentId c,
    TriPortId at, TriPortId to, TriSignatureId signature,
    TciParameterList tciPars, Value excValue, TriStatus transmissionFailure);
public void tliPrRaise_c_BC(String am, int ts, String src, int line, TriComponentId c,
    TriPortId at, TriPortIdList to, TriSignatureId signature, TciParameterList tciPars,
    Value excValue, TriStatus transmissionFailure);
public void tliPrRaise_c_MC(String am, int ts, String src, int line, TriComponentId c,
    TriPortId at, TriPortIdList to, TriSignatureId signature, TciParameterList tciPars,
    Value excValue, TriStatus transmissionFailure);
public void tliPrCatchDetected_m(String am, int ts, String src, int line, TriComponentId c,
    TriPortId at, TriPortId from, TriSignatureId signature,
    TriException exc, TriAddress address);
public void tliPrCatchDetected_c(String am, int ts, String src, int line, TriComponentId c,
    TriPortId at, TriPortId from, TriSignatureId signature,
    Value excValue);
public void tliPrCatchMismatch_m(String am, int ts, String src, int line, TriComponentId c,
    TriPortId at, TriSignatureId signature,
    Value excValue, TciValueTemplate excTmpl, TciValueDifferenceList diffs,

```

```

    Value addrValue, TciValueTemplate addressTpl);
public void tliPrCatchMismatch_c(String am, int ts, String src, int line, TriComponentId c,
    TriPortId at, TriSignatureId signature,
    Value excValue, TciValueTemplate excTpl, TciValueDifferenceList diffs,
    TriComponentId from, TciNonValueTemplate fromTpl);
public void tliPrCatch_m(String am, int ts, String src, int line, TriComponentId c,
    TriPortId at, TriSignatureId signature,
    Value excValue, TciValueTemplate excTpl,
    Value addrValue, TciValueTemplate addressTpl);
public void tliPrCatch_c(String am, int ts, String src, int line, TriComponentId c,
    TriPortId at, TriSignatureId signature,
    Value excValue, TciValueTemplate excTpl,
    TriComponentId from, TciNonValueTemplate fromTpl);
public void tliPrCatchTimeoutDetected(String am, int ts, String src, int line, TriComponentId c,
    TriPortId at, TriSignatureId signature);
public void tliPrCatchTimeout(String am, int ts, String src, int line, TriComponentId c,
    TriPortId at, TriSignatureId signature);
public void tliCCreate(String am, int ts, String src, int line, TriComponentId c,
    TriComponentId comp, String name, Boolean alive);
public void tliCStart(String am, int ts, String src, int line, TriComponentId c,
    TriComponentId comp, TciBehaviourId name, TciParameterList tciPars);
public void tliCRunning(String am, int ts, String src, int line, TriComponentId c,
    TriComponentId comp, ComponentStatus status);
public void tliCAlive(String am, int ts, String src, int line, TriComponentId c,
    TriComponentId comp, ComponentStatus status);
public void tliCStop(String am, int ts, String src, int line, TriComponentId c,
    TriComponentId comp);
public void tliCKill(String am, int ts, String src, int line, TriComponentId c,
    TriComponentId comp);
public void tliCDoneMismatch(String am, int ts, String src, int line, TriComponentId c,
    TriComponentId comp, TciNonValueTemplate compTpl);
public void tliCDone(String am, int ts, String src, int line, TriComponentId c,
    TciNonValueTemplate compTpl);
public void tliCKilledMismatch(String am, int ts, String src, int line, TriComponentId c,
    TriComponentId comp, TciNonValueTemplate compTpl);
public void tliCKilled(String am, int ts, String src, int line, TriComponentId c,
    TciNonValueTemplate compTpl);
public void tliCTerminated(String am, int ts, String src, int line, TriComponentId c,
    VerdictValue verdict, String reason);
public void tliPConnect(String am, int ts, String src, int line, TriComponentId c,
    TriPortId port1, TriPortId port2);
public void tliPDisconnect(String am, int ts, String src, int line, TriComponentId c,
    TriPortId port1, TriPortId port2);
public void tliPMap(String am, int ts, String src, int line, TriComponentId c,
    TriPortId port1, TriPortId port2);
public void tliPMapParam(String am, int ts, String src, int line, TriComponentId c,
    TriPortId port1, TriPortId port2, TciParameterList tciPars,
    TciStatus encoderFailure, TriParameterList triPars);
public void tliPUnmap(String am, int ts, String src, int line, TriComponentId c,
    TriPortId port1, TriPortId port2);
public void tliPUnmapParam(String am, int ts, String src, int line, TriComponentId c,
    TriPortId port1, TriPortId port2, TciParameterList tciPars,
    TciStatus encoderFailure, TriParameterList triPars);
public void tliPClear(String am, int ts, String src, int line, TriComponentId c,
    TriPortId port);
public void tliPStart(String am, int ts, String src, int line, TriComponentId c,
    TriPortId port);
public void tliPStop(String am, int ts, String src, int line, TriComponentId c,
    TriPortId port);
public void tliPHalt(String am, int ts, String src, int line, TriComponentId c,
    TriPortId port);
public void tliEncode(String am, int ts, String src, int line, TriComponentId c,
    Value val, TciStatus encoderFailure, TriMessage msg, String codec);
public void tliDecode(String am, int ts, String src, int line, TriComponentId c,
    TriMessage msg, TciStatus decoderFailure, Value val, String codec);
public void tliTimeoutDetected(String am, int ts, String src, int line, TriComponentId c,
    TriTimerId timer);
public void tliTimeoutMismatch(String am, int ts, String src, int line, TriComponentId c,
    TriTimerId timer, TciNonValueTemplate timerTpl);
public void tliTimeout(String am, int ts, String src, int line, TriComponentId c,
    TriTimerId timer, TciNonValueTemplate timerTpl);
public void tliTStart(String am, int ts, String src, int line, TriComponentId c,
    TriTimerId timer, TriTimerDuration dur);
public void tliTStop(String am, int ts, String src, int line, TriComponentId c,
    TriTimerId timer, in TriTimerDuration dur);
public void tliTRead(String am, int ts, String src, int line, TriComponentId c,
    TriTimerId timer, TriTimerDuration elapsed);
public void tliTRunning(String am, int ts, String src, int line, TriComponentId c,
    TriTimerId timer, TimerStatus status);
public void tliSEnter(String am, int ts, String src, int line, TriComponentId c,

```

```

        QualifiedName name, TciParameterList tciPars, String kind);
public void tliSLeave(String am, int ts, String src, int line, TriComponentId c,
    QualifiedName name, TciParameterList tciPars, Value returnValue, String kind);
public void tliVar(String am, int ts, String src, int line, TriComponentId c,
    QualifiedName name, Value varValue);
public void tliModulePar(String am, int ts, String src, int line, TriComponentId c,
    QualifiedName name, Value parValue);
public void tliGetVerdict(String am, int ts, String src, int line, TriComponentId c,
    VerdictValue verdict);
public void tliSetVerdict(String am, int ts, String src, int line, TriComponentId c,
    VerdictValue verdict, String reason);
public void tliLog(String am, int ts, String src, int line, TriComponentId c,
    Tstring log);
public void tliAEnter(String am, int ts, String src, int line, TriComponentId c);
public void tliALeave(String am, int ts, String src, int line, TriComponentId c);
public void tliADefaults(String am, int ts, String src, int line, TriComponentId c);
public void tliAActivate(String am, int ts, String src, int line, TriComponentId c,
    QualifiedName name, TciParameterList tciPars, Value ref);
public void tliADeactivate(String am, int ts, String src, int line, TriComponentId c,
    Value ref);
public void tliANomatch(String am, int ts, String src, int line, TriComponentId c);
public void tliARepeat(String am, int ts, String src, int line, TriComponentId c);
public void tliAWait(String am, int ts, String src, int line, TriComponentId c);
public void tliAction(String am, int ts, String src, int line, TriComponentId c, String action);
public void tliMatch(String am, int ts, String src, int line, TriComponentId c, Value expr,
    TciValueTemplate tpl);
public void tliMatchMismatch(String am, int ts, String src, int line, TriComponentId c,
    Value expr, TciValueTemplate tpl, TciValueDifferenceList diffs);
public void tliInfo (String am, int ts, String src, int line, TriComponent c,
    int level, String info)
}

```

## 8.6 Optional parameters

Clause 7.3 defines that a reserved value shall be used to indicate the absence of an optional parameter. For the Java language mapping the Java null value shall be used to indicate the absence of an optional value. For example, if in the `tciReplyConnected` operation the value parameter shall be omitted the operation invocation shall be `tciReplyConnected (sender, receiver, signature, parameterList, null)`.

## 8.7 TCI initialization

All methods are non-static, i.e., operations can only be called on objects. As the present document does not define concrete implementation strategies of TE, TM, CD and CH the mechanism how the TE, the TM, the CD or the CH get to know the handles on the respective objects is out of scope of the present document.

Tool vendors shall provide methods to the developers of TM, CD and CH to register the TE, TM, CD and CH to their respective communication partner.

## 8.8 Error handling

All operations called from the TM, CH or CD that return have succeeded. If an erroneous situation has been identified by the TE a test case error will be communicated to the user using the procedures as defined in clause 7.3.1.2.6 (`tciError`). If an operation called by the TE in the TM, CH, CD, or TL produces an error, this erroneous situation should be communicated to the TE using the procedures as defined in clause 7.3.2.1.12 (`tciErrorReq`).

Beside this error handling no additional error handling is defined with this Java language mapping. In particular, no exception handling mechanisms are defined.

## 9 ANSI C language mapping

### 9.1 Introduction

This clause defines the TCI ANSI-C [8] language mapping for the TCI data specified in clause 7.2 and for the TCI operations specified in clause 7.3.

### 9.2 Value interfaces

TCI IDL Interface	ANSI C representation	Notes and comments
Type		
TciModuleIdType getDefiningModule()	TciModuleIdType tciGetDefiningModule(Type inst)	
Tstring getName()	String tciGetName(Type inst)	String type reused from IDL (OMG recommendation)
TciTypeClassType getTypeClass()	TciTypeClassType tciGetTypeClass (Type inst)	
Value newInstance()	Value tciNewInstance(Type inst)	
Tstring getTypeEncoding()	String tciGetTypeEncoding(Type inst)	
TstringSeq getTypeExtension()	String* getTypeExtension(Type inst)	
Tstring getTypeEncodingVariant()	String tciGetTypeEncodingVariant(Type inst)	
Value		
Tstring getValueEncoding()	String tciGetValueEncoding(Value inst)	
Tstring getValueEncodingVariant()	String tciGetValueEncodingVariant(Value inst)	
Type getType()	Type tciGetType(Value inst)	
Tboolean notPresent()	Boolean tciNotPresent(Value inst)	Boolean type reused from IDL (OMG recommendation)
	void tciSetNull(Value inst)	For optional parameters of operations, see clause 9.7
	Boolean tciIsNull(Value inst)	For optional parameters of operations, see clause 9.7. Boolean type reused from IDL (OMG recommendation)



TCI IDL Interface	ANSI C representation	Notes and comments
<b>IntegerValue</b>		
Tinteger getInt()	String tciGetIntAbs(Value inst)	Returns the (10-base) integer absolute value as an ASCII string.
	unsigned long int tciGetIntNumberOfDigits (Value inst)	Returns the number of digits in an integer value.
	Boolean tciGetIntSign(Value inst)	Returns true if the number is positive, false otherwise.
	char tciGetIntDigit (Value inst, unsigned long int position)	Returns the value of the digit at position 'position', where position 0 is the least significant digit.
void setInt(in Tinteger value)	void tciSetIntAbs(Value inst, String value)	Sets the (10-base) absolute value of the integer via an ASCII string. The first digit must not be 0 unless the value is 0.
	Void tciSetIntNumberOfDigits (Value inst, unsigned long int dig_num)	Sets the number of digits in an integer value.
	void tciSetIntSign (Value inst, Boolean sign)	Sets the sign to + (true) or – (false).
	void tciSetIntDigit (Value inst, unsigned long int position, char digit)	Sets the value of the digit at position 'position', where position 0 is the least significant digit.
<b>FloatValue</b>		
Tfloat getFloat()	double tciGetFloatValue(Value inst)	
void setFloat(in Tfloat value)	void tciSetFloatValue(Value inst, double value)	
<b>BooleanValue</b>		
Tboolean getBoolean()	Boolean tciGetBooleanValue(Value inst)	
void setBoolean (in Tboolean value)	void tciSetBooleanValue (Value inst, Boolean value)	

<b>TCI IDL Interface</b>	<b>ANSI C representation</b>	<b>Notes and comments</b>
<b>CharstringValue</b>		
Tstring getString()	TciCharStringValue tciGetCStringValue(Value inst)	
void setString(in Tstring value)	void tciSetCStringValue (Value inst, TciCharStringValue value)	
Tchar getChar (in Tinteger position)	char tciGetCStringCharValue (Value inst, long int position)	
void setChar (in Tinteger position, in Tchar value)	void tciSetCStringCharValue (Value inst, long int position, char value)	
Tinteger getLength()	unsigned long int tciGetCStringLength (Value inst)	
void setLength(in Tinteger len)	void tciSetCStringLength (Value inst, unsigned long int len)	
<b>UniversalCharstringValue</b>		
Tstring getString()	TciUCStringValue tciGetUCStringValue(Value inst)	
void setString(in Tstring value)	void tciSetUCStringValue (Value inst, TciUCStringValue value)	
TuniversalChar getChar (in Tinteger position)	void tciGetUCStringCharValue (Value inst, unsigned long int position, TciUCValue result)	
void setChar (in Tinteger position, in TuniversalChar value)	void tciSetUCStringCharValue (Value inst, unsigned long int position, TciUCValue value)	
Tinteger getLength()	unsigned long int tciGetUCStringLength(Value inst)	
void setLength(in Tinteger len)	void tciSetUCStringLength (Value inst, unsigned long int len)	
<b>BitstringValue</b>		
Tstring getString()	String tciGetBStringValue(Value inst)	
void setString(in Tstring value)	void tciSetBStringValue (Value inst, String value)	
Tchar getBit (in integer position)	int tciGetBStringBitValue (Value inst, long int position)	
void setBit (in Tinteger position, in Tinteger value)	void tciSetBStringBitValue (Value inst, unsigned long int position, int value)	
Tinteger getLength()	unsigned long int tciGetBStringLength(Value inst)	
void setLength(in Tinteger len)	void tciSetBStringLength (Value inst, long int len)	
<b>HexstringValue</b>		
Tstring getString()	String tciGetHStringValue(Value inst)	
void setString(in Tstring value)	void tciSetHStringValue (Value inst, String value)	

<b>TCI IDL Interface</b>	<b>ANSI C representation</b>	<b>Notes and comments</b>
Tchar getHex (in Tinteger position)	int tciGetHStringHexValue (Value inst, unsigned long int position)	
void setBit (in Tinteger position, in Tinteger value)	void tciSetHStringHexValue (Value inst, unsigned long int position, int value)	
Tinteger getLength()	long int tciGetHStringLength(Value inst)	
void setLength(in Tinteger len)	void tciSetHStringLength (Value inst, unsigned long int len)	
<b>OctetstringValue</b>		
Tstring getString()	String tciGetOStringValue(Value inst)	
void setString(in Tstring value)	void tciSetOStringValue (Value inst, String value)	
Tchar getOctet(in Tinteger position)	int tciGetOStringOctetValue (Value inst, unsigned long int position)	
void setOctet (in Tinteger position, in Tinteger value)	void tciSetOStringOctetValue (Value inst, unsigned long int position, int value)	
Tinteger getLength()	unsigned long int tciGetOStringLength(Value inst)	
void setLength(in Tinteger len)	void tciSetOStringLength (Value inst, unsigned long int len)	
<b>RecordValue</b>		
Value getField(in Tstring fieldName int)	Value tciGetRecFieldValue (Value inst, String fieldName int)	
void setField (in Tstring fieldName int, in Value value)	void tciSetRecFieldValue (Value inst, String fieldName int, Value value)	
Tstring[] getFieldNames()	char** tciGetRecFieldNames(Value inst)	Returns a NULL-terminated array of the field names.
void setFieldOmitted (in Tstring fieldName int)	void setFieldOmitted (Value inst, String fieldName int)	
<b>RecordOfValue</b>		
Value getField(in Tinteger position)	Value tciGetRecOfFieldValue (Value inst, unsigned long int position)	
void setField (in Tinteger position, in Value value)	void tciSetRecOfFieldValue (Value inst, unsigned long int position, Value value)	
void appendField(in Value value)	void tciAppendRecOfFieldValue (Value inst, Value value)	
Type getElementType()	Type tciGetRecOfElementType(Value inst)	

<b>TCI IDL Interface</b>	<b>ANSI C representation</b>	<b>Notes and comments</b>
Tinteger getLength()	unsigned long int tciGetRecOfLength(Value inst)	
void setLength(in Tinteger len)	void tciSetRecOfLength (Value inst, unsigned long int len)	
Tinteger getOffset()	unsigned long int tciGetOffset(Value inst)	
<b>UnionValue</b>		
Value getVariant (in Tstring variantName)	Value tciGetUnionVariant (Value inst, String variantName)	
void setVariant (in Tstring variantName, in Value value)	void tciSetUnionVariant (Value inst, String variantName, Value value)	
Tstring getPresentVariantName()	String tciGetUnionPresentVariantName (Value inst)	
Tstring[] getVariantNames()	char** tciGetUnionVariantNames(Value inst)	Returns a NULL-terminated array of the field names.
<b>EnumeratedValue</b>		
Tstring getEnum()	String tciGetEnumValue(Value inst)	
void setEnum(in Tstring enumValue)	void tciSetEnumValue (Value inst, String enumValue)	
Tinteger getInt()	unsigned long tciGetEnumInt(Value inst);	
setInt(in Tinteger intValue)	void tciSetEnumInt(Value inst, unsigned long intValue);	
<b>VerdictValue</b>		
Tinteger getVerdict()	int tciGetVerdictValue(Value inst)	
void setVerdict(in Tinteger verdict)	void tciSetVerdictValue(Value inst, int verdict)	
<b>AddressValue</b>		
Value getAddress()	Value tciGetAddressValue(Value inst)	
void setAddress(in Value value)	void tciSetAddressValue(Value inst, Value value)	

### 9.3 Logging interface

<b>TCI IDL Interface</b>	<b>ANSI C representation</b>	<b>Notes and comments</b>
<b>TciValueTemplate</b>		
Tboolean isOmit()	Boolean tciIsOmit(TciValueTemplate inst)	Boolean type reused from IDL (OMG recommendation)

<b>TCI IDL Interface</b>	<b>ANSI C representation</b>	<b>Notes and comments</b>
Tboolean isAny()	Boolean tciIsAny(TciValueTemplate inst)	Boolean type reused from IDL (OMG recommendation)
Tboolean isAnyOrOmit()	Boolean tciIsAnyOrOmit(TciValueTemplate inst)	Boolean type reused from IDL (OMG recommendation)
Tstring getTemplateDef()	String tciGetTemplateDef(TciValueTemplate inst)	String type reused from IDL (OMG recommendation)
<b>TciNonValueTemplate</b>		
Tboolean isAny()	Boolean tciIsAnyNonValue(TciNonValueTemplate inst)	Boolean type reused from IDL (OMG recommendation)
Tboolean isAll()	Boolean tciIsAllNonValue(TciNonValueTemplate inst)	Boolean type reused from IDL (OMG recommendation)
Tstring getTemplateDef()	String tciGetTemplateDefNonValue(TciNonValueTemplate inst)	String type reused from IDL (OMG recommendation)
<b>TciValueList</b>		
Tinteger size()	int size(TciValueList inst)	
Tboolean isEmpty()	Boolean isEmpty(TciValueList inst)	Boolean type reused from IDL (OMG recommendation)
Value get(Tinteger index)	Value get(TciValueList inst, int index)	
<b>TciValueDifference</b>		
Value getValue()	Value getValue(TciValueDifference inst)	
TciValueTemplate getTciValueTemplate()	TciValueTemplate getTciValueTemplate(TciValueDifference inst)	
Tstring getDescription()	String getDescription(TciValueDifference inst)	String type reused from IDL (OMG recommendation)
<b>TciValueDifferenceList</b>		
Tinteger size()	int size(TciValueDifferenceList inst)	
Tboolean isEmpty()	Boolean isEmpty(TciValueDifferenceList inst)	Boolean type reused from IDL (OMG recommendation)
TciValueDifference get(Tinteger index)	TciValueDifference get(TciValueDifferenceList inst, int index)	

## 9.4 Operation interfaces

### 9.4.1 The TCI-TM interface

#### 9.4.1.1 TCI-TM provided

The TCI-TM Provided interface is mapped to the following interface:

```
void tciTestCaseStarted
    (TciTestCaseIdType testCaseId, TciParameterListType parameterList, double timer)
void tciTestCaseTerminated (VerdictValue verdict, TciParameterListType parameterlist)
void tciControlTerminated()
Value tciGetModulePar (TciModuleParameterIdType parameterId)
void tciLog(String message)
void tciError(String message)
```

#### 9.4.1.2 TCI-TM required

The TCI-TM Required interface is mapped to the following interface:

```
void tciRootModule(String moduleId)
TciModuleIdListType tciGetImportedModules()
TciModuleParameterListType tciGetModuleParameters(TciModuleIdType moduleName)
TciTestCaseIdListType tciGetTestCases()
TciParameterTypeListType tciGetTestCaseParameters (TciTestCaseIdType testCaseId)
TriPortIdList tciGetTestCaseTSI (TciTestCaseIdType testCaseId)
void tciStartTestCase (TciTestCaseIdType testCaseId, TciParameterListType parameterlist)
void tciStopTestCase()
TriComponentId tciStartControl()
void tciStopControl()
```

### 9.4.2 The TCI-CD interface

#### 9.4.2.1 TCI-CD provided

The TCI-CD Provided interface is mapped to the following interface:

```
Value tciDecode (BinaryString message, Type decHypothesis)
BinaryString tciEncode(Value value)
```

NOTE – BinaryString type reused from TRI.

#### 9.4.2.2 TCI-CD required

The TCI-CD Required interface is mapped to the following interface:

```
Type tciGetTypeForName(String typeName)
Type tciGetIntegerType()
Type tciGetFloatType()
Type tciGetBooleanType()
Type tciGetCharType()
Type tciGetUniversalCharType()
Type tciGetTciCharstringType()
Type tciGetUniversalCharstringType()
Type tciGetHexstringType()
Type tciGetBitstringType()
Type tciGetOctetstringType()
Type tciGetVerdictType()
void tciErrorReq(String message)
```

### 9.4.3 The TCI-CH interface

#### 9.4.3.1 TCI-CH provided

The TCI-CH Provided interface is mapped to the following interface:

```
void tciSendConnected(TriPortId sender, TriComponentId receiver, Value sendMessage)
void tciSendConnectedBC(TriPortId sender, Value sendMessage)
void tciSendConnectedMC
    (TriPortId sender, TriComponentIdList receivers, Value sendMessage)
void tciCallConnected
```

```

(TriPortId sender, TriComponentId receiver, TriSignatureId signature,
TciParameterListType parameterList)
void          tciCallConnectedBC
(TriPortId sender, TriSignatureId signature, TciParameterListType parameterList)
void          tciCallConnectedMC
(TriPortId sender, TriComponentIdList receivers, TriSignatureId signature,
TciParameterListType parameterList)
void          tciReplyConnected
(TriPortId sender, TriComponentId receiver, TriSignatureId signature,
TciParameterListType parameterList, Value returnValue)
void          tciReplyConnectedBC
(TriPortId sender, TriSignatureId signature, TciParameterListType parameterList,
Value returnValue)
void          tciReplyConnectedMC
(TriPortId sender, TriComponentIdList receivers, TriSignatureId signature,
TciParameterListType parameterList, Value returnValue)
void          tciRaiseConnected
(TriPortId sender, TriComponentId receiver, TriSignatureId signature, Value exception)
void          tciRaiseConnectedBC
(TriPortId sender, TriSignatureId signature, Value exception)
void          tciRaiseConnectedMC
(TriPortId sender, TriComponentIdList receivers, TriSignatureId signature, Value exception)
TriComponentId tciCreateTestComponentReq
(TciTestComponentKindType kind, Type componentType, String name, Value hostId)
void          tciStartTestComponentReq
(TriComponentId component, TciBehaviourIdType behaviour, TciParameterListType parameterList)
void          tciStopTestComponentReq (TriComponentId component)
void          tciConnectReq(TriPortId fromPort, TriPortId toPort)
void          tciDisconnectReq(TriPortId fromPort, TriPortId toPort)
void          tciMapReq(TriPortId fromPort, TriPortId toPort)
void          tciMapParamReq
(TriPortId fromPort, TriPortId toPort, TciParameterListType parameterList)
void          tciUnmapReq(TriPortId fromPort, TriPortId toPort)
void          tciUnmapParamReq
(TriPortId fromPort, TriPortId toPort, TciParameterListType parameterList)
void          tciTestComponentTerminatedReq(TriComponentId component, VerdictValue verdict)
Boolean      tciTestComponentRunningReq(TriComponentId component)
Boolean      tciTestComponentDoneReq(TriComponentId component)
TriComponentId tciGetMTCReq()
void          tciExecuteTestCaseReq(TciTestCaseIdType testCaseId, TriPortIdList tsiPortList)
void          tciResetReq()
void          tciKillTestComponentReq(TriComponentId component)
Boolean      tciTestComponentAliveReq (TriComponentId component)
Boolean      tciTestComponentKilledReq (TriComponentId component)

```

### 9.4.3.2 TCI-CH required

The TCI-CH Required interface is mapped to the following interface:

```

void          tciEnqueueMsgConnected
(TriPortId sender, TriComponentId receiver, Value rcvdMessage)
void          tciEnqueueCallConnected
(TriPortId sender, TriComponentId receiver, TriSignatureId signature,
TciParameterListType parameterList)
void          tciEnqueueReplyConnected
(TriPortId sender, TriComponentId receiver, TriSignatureId signature,
TciParameterListType parameterList, Value returnValue)
void          tciEnqueueRaiseConnected
(TriPortId sender, TriComponentId receiver, TriSignatureId signature, Value exception)
TriComponentId tciCreateTestComponent
(TciTestComponentKindType kind, Type componentType, String name)
void          tciStartTestComponent
(TriComponentId component, TciBehaviourIdType behaviour, TciParameterListType parameterList)
void          tciStopTestComponent(TriComponentId component)
void          tciConnect(TriPortId fromPort, TriPortId toPort)
void          tciDisconnect(TriPortId fromPort, TriPortId toPort)
void          tciMap (TriPortId fromPort, TriPortId toPort)
void          tciMapParam
(TriPortId fromPort, TriPortId toPort, TciParameterListType parameterList)
void          tciUnmap(TriPortId fromPort, TriPortId toPort)
void          tciUnmapParam
(TriPortId fromPort, TriPortId toPort, TciParameterListType parameterList)
void          tciTestComponentTerminated(TriComponentId component, VerdictValue verdict)
Boolean      tciTestComponentRunning(TriComponentId component)
Boolean      tciTestComponentDone(TriComponentId component)
TriComponentId tciGetMTC()
void          tciExecuteTestCase(TciTestCaseIdType testCaseId, TriPortIdList tsiPortList)
void          tciReset()

```

```

void          tciKillTestComponent(TriComponentId component)
Boolean      tciTestComponentAlive(TriComponentId component)
Boolean      tciTestComponentKilled(TriComponentId component)

```

## 9.4.4 The TCI-TL interface

### 9.4.4.1 TCI-TL provided

The TCI-TL Provided interface is mapped to the following interface:

```

void tliTcExecute
  (String am, int ts, String src, int line, TriComponentId c, TciTestCaseIdType tcId,
  TciParameterListType tciPars, TriTimerDuration dur)
void tliTcStart
  (String am, int ts, String src, int line, TriComponentId c, TciTestCaseIdType tcId,
  TciParameterListType tciPars, TriTimerDuration dur)
void tliTcStop
  (String am, int ts, String src, int line, TriComponentId c, String reason)
void tliTcStarted
  (String am, int ts, String src, int line, TriComponentId c, TciTestCaseIdType tcId,
  TciParameterListType tciPars, TriTimerDuration dur)
void tliTcTerminated
  (String am, int ts, String src, int line, TriComponentId c, TciTestCaseIdType tcId,
  TciParameterListType tciPars, VerdictValue verdict, String reason)

void tliCtrlStart(String am, int ts, String src, int line, TriComponentId c)
void tliCtrlStop(String am, int ts, String src, int line, TriComponentId c)
void tliCtrlTerminated(String am, int ts, String src, int line, TriComponentId c)

void tliMSend_m
  (String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriPortId to,
  Value msgValue, Value addrValue, TciStatus encoderFailure, TriMessage msg,
  TriAddress address, TriStatus transmissionFailure)
void tliMSend_m_BC
  (String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriPortId to,
  Value msgValue, TciStatus encoderFailure, TriMessage msg, TriStatus transmissionFailure)
void tliMSend_m_MC
  (String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriPortId to,
  Value msgValue, TciValueList addrValues, TciStatus encoderFailure, TriMessage msg,
  TriAddressList addresses, TriStatus transmissionFailure)
void tliMSend_c
  (String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriPortId to,
  Value msgValue, TriStatus transmissionFailure)
void tliMSend_c_BC
  (String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriPortIdList to,
  Value msgValue, TriStatus transmissionFailure)
void tliMSend_c_MC
  (String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriPortIdList to,
  Value msgValue, TriStatus transmissionFailure)
void tliMDetected_m
  (String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriPortId from,
  TriMessage msg, TriAddress address)
void tliMDetected_c
  (String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriPortId from,
  Value msgValue)
void tliMMismatch_m
  (String am, int ts, String src, int line, TriComponentId c, TriPortId at, Value msgValue,
  TciValueTemplate msgTmpl, TciValueDifferenceList diffs, Value addrValue,
  TciValueTemplate addressTmpl)
void tliMMismatch_c
  (String am, int ts, String src, int line, TriComponentId c, TriPortId at, Value msgValue,
  TciValueTemplate msgTmpl, TciValueDifferenceList diffs, TriComponentId from,
  TciNonValueTemplate fromTmpl)
void tliMReceive_m
  (String am, int ts, String src, int line, TriComponentId c, TriPortId at, Value msgValue,
  TciValueTemplate msgTmpl, Value addrValue, TciValueTemplate addressTmpl)
void tliMReceive_c
  (String am, int ts, String src, int line, TriComponentId c, TriPortId at, Value msgValue,
  TciValueTemplate msgTmpl, TriComponentId from, TciNonValueTemplate fromTmpl)

void tliPrCall_m
  (String am, int ts, String src, int line, TriComponentId c, TriPortId at,
  TriPortId to, TriSignatureId signature,
  TciParameterListType tciPars, Value addrValue, TciStatus encoderFailure,
  TriParameterList triPars, TriAddress address, TriStatus transmissionFailure)
void tliPrCall_m_BC
  (String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriPortId to,
  TriSignatureId signature, TciParameterListType tciPars, TciStatus encoderFailure,

```



```

TriParameterList triPars, TriStatus transmissionFailure)
void tliPrCall_m_MC
    (String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriPortId to,
TriSignatureId signature, TciParameterListType tciPars, TciValueList addrValues,
TciStatus encoderFailure, TriParameterList triPars, TriAddressList addresses,
TriStatus transmissionFailure)
void tliPrCall_c
    (String am, int ts, String src, int line, TriComponentId c, TriPortId at,
TriPortId to, TriSignatureId signature,
TciParameterListType tciPars, TriStatus transmissionFailure)
void tliPrCall_c_BC
    (String am, int ts, String srcint line, TriComponentId c, TriPortId at, TriPortIdList to,
TriSignatureId signature, TciParameterListType tciPars, TriStatus transmissionFailure)
void tliPrCall_c_MC
    (String am, int ts, String srcint line, TriComponentId c, TriPortId at, TriPortIdList to,
TriSignatureId signature, TciParameterListType tciPars, TriStatus transmissionFailure)
void tliPrGetCallDetected_m
    (String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriPortId from,
TriSignatureId signature, TriParameterList triPars, TriAddress address)
void tliPrGetCallDetected_c
    (String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriPortId from,
TriSignatureId signature, TciParameterListType tciPars)
void tliPrGetCallMismatch_m
    (String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriSignatureId signature,
TciParameterListType tciPars, TciValueTemplate parsTmpl, TciValueDifferenceList diffs,
Value addrValue, TciValueTemplate addressTmpl)
void tliPrGetCallMismatch_c
    (String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriSignatureId signature,
TciParameterListType tciPars, TciValueTemplate parsTmpl, TciValueDifferenceList diffs,
TriComponentId from, TciNonValueTemplate fromTmpl)
void tliPrGetCall_m
    (String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriSignatureId signature,
TciParameterListType tciPars, TciValueTemplate parsTmpl, Value addrValue,
TciValueTemplate addressTmpl)
void tliPrGetCall_c
    (String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriSignatureId signature,
TciParameterListType tciPars, TciValueTemplate parsTmpl, TriComponentId from,
TciNonValueTemplate fromTmpl)
void tliPrReply_m
    (String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriPortId to,
TriSignatureId signature, TciParameterListType tciPars, Value replValue,
Value addrValue, TciStatus encoderFailure, TriParameterList triPars,
TriParameter repl, TriAddress address, TriStatus transmissionFailure)
void tliPrReply_m_BC
    (String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriPortId to,
TriSignatureId signature, TciParameterListType tciPars, Value replValue,
TciStatus encoderFailure, TriParameterList triPars, TriParameter repl,
TriStatus transmissionFailure)
void tliPrReply_m_MC
    (String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriPortId to,
TriSignatureId signature, TciParameterListType tciPars, Value replValue,
TciValueList addrValues, TriStatus encoderFailure, TriParameterList triPars,
TriParameter repl, TriAddressList addresses, TciStatus transmissionFailure)
void tliPrReply_c
    (String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriPortId to,
TriSignatureId signature, TciParameterListType tciPars, Value replValue,
TriStatus transmissionFailure)
void tliPrReply_c_BC
    (String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriPortIdList to,
TriSignatureId signature, TciParameterListType tciPars, Value replValue,
TriStatus transmissionFailure)
void tliPrReply_c_MC
    (String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriPortIdList to,
TriSignatureId signature, TciParameterListType tciPars, Value replValue,
TriStatus transmissionFailure)
void tliPrGetReplyDetected_m
    (String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriPortId from,
TriSignatureId signature, TriParameterList triPars, TriParameter repl, TriAddress address)
void tliPrGetReplyDetected_c
    (String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriPortId from,
TriSignatureId signature, TciParameterListType tciPars, Value replValue)
void tliPrGetReplyMismatch_m
    (String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriSignatureId signature,
TciParameterListType tciPars, TciValueTemplate parsTmpl, Value replValue,
TciValueTemplate replyTmpl, TciValueDifferenceList diffs, Value addrValue,
TciValueTemplate addressTmpl)
void tliPrGetReplyMismatch_c
    (String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriSignatureId signature,
TciParameterListType tciPars, TciValueTemplate parsTmpl, Value replValue,
TciValueTemplate replyTmpl, TciValueDifferenceList diffs, TriComponentId from,

```

```

TciNonValueTemplate fromTmpl)
void tliPrGetReply_m
    (String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriSignatureId signature,
TciParameterListType tciPars, TciValueTemplate parsTmpl, Value replValue,
TciValueTemplate replyTmpl, Value addrValue, TciValueTemplate addressTmpl)
void tliPrGetReply_c
    (String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriSignatureId signature,
TciParameterListType tciPars, TciValueTemplate parsTmpl, Value replValue,
TciValueTemplate replyTmpl, TriComponentId from, TciNonValueTemplate fromTmpl)
void tliPrRaise_m
    (String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriPortId to,
TriSignatureId signature, TciParameterListType tciPars, Value excValue, Value addrValue,
TciStatus encoderFailure, TriException exc, TriAddress address, TriStatus transmissionFailure)
void tliPrRaise_m_BC
    (String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriPortId to,
TriSignatureId signature, TciParameterListType tciPars, Value excValue,
TciStatus encoderFailure, TriException exc, TriStatus transmissionFailure)
void tliPrRaise_m_MC
    (String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriPortId to,
TriSignatureId signature, TciParameterListType tciPars, Value excValue,
TciValueList addrValues, TciStatus encoderFailure, TriException exc,
TriAddressList addresses, TriStatus transmissionFailure)
void tliPrRaise_c
    (String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriPortId to,
TriSignatureId signature, TciParameterListType tciPars, Value excValue,
TriStatus transmissionFailure)
void tliPrRaise_c_BC
    (String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriPortIdList to,
TriSignatureId signature, TciParameterListType tciPars, Value excValue,
TriStatus transmissionFailure)
void tliPrRaise_c_MC
    (String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriPortIdList to,
TriSignatureId signature, TciParameterListType tciPars, Value excValue,
TriStatus transmissionFailure)
void tliPrCatchDetected_m
    (String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriPortId from,
TriSignatureId signature, TriException exc, TriAddress address)
void tliPrCatchDetected_c
    (String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriPortId from,
TriSignatureId signature, Value excValue)
void tliPrCatchMismatch_m
    (String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriSignatureId signature,
Value excValue, TciValueTemplate excTmpl, TciValueDifferenceList diffs, Value addrValue,
TciValueTemplate addressTmpl)
void tliPrCatchMismatch_c
    (String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriSignatureId signature,
Value excValue, TciValueTemplate excTmpl, TciValueDifferenceList diffs, TriComponentId from,
TciNonValueTemplate fromTmpl)
void tliPrCatch_m
    (String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriSignatureId signature,
Value excValue, TciValueTemplate excTmpl, Value addrValue, TciValueTemplate addressTmpl)
void tliPrCatch_c
    (String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriSignatureId signature,
Value excValue, TciValueTemplate excTmpl, TriComponentId from, TciNonValueTemplate fromTmpl)
void tliPrCatchTimeoutDetected
    (String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriSignatureId signature)
void tliPrCatchTimeout
    (String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriSignatureId signature)

void tliCCreate
    (String am, int ts, String src, int line, TriComponentId c, TriComponentId comp, String name,
Boolean alive)
void tliCStart
    (String am, int ts, String src, int line, TriComponentId c, TriComponentId comp,
TciBehaviourIdType name, TciParameterListType tciPars)
void tliCRunning
    (String am, int ts, String src, int line, TriComponentId c, TriComponentId comp,
ComponentStatus status)
void tliCAlive
    (String am, int ts, String src, int line, TriComponentId c, TriComponentId comp,
ComponentStatus status)
void tliCStop
    (String am, int ts, String src, int line, TriComponentId c, TriComponentId comp)
void tliCKill
    (String am, int ts, String src, int line, TriComponentId c, TriComponentId comp)
void tliCDoneMismatch
    (String am, int ts, String src, int line, TriComponentId c, TriComponentId comp,
TciNonValueTemplate compTmpl)
void tliCDone
    (String am, int ts, String src, int line, TriComponentId c, TciNonValueTemplate compTmpl)

```

```

void tliCTerminated
  (String am, int ts, String src, int line, TriComponentId c, VerdictValue verdict, String reason)
void tliCKilledMismatch
  (String am, int ts, String src, int line, TriComponentId c, TriComponentId comp,
TciNonValueTemplate compTpl)
void tliCKilled
  (String am, int ts, String src, int line, TriComponentId c, TciNonValueTemplate compTpl)

void tliPConnect
  (String am, int ts, String src, int line, TriComponentId c, TriPortId port1, TriPortId port2)
void tliPDisconnect
  (String am, int ts, String src, int line, TriComponentId c, TriPortId port1,
TriPortId port2)
void tliPMap
  (String am, int ts, String src, int line, TriComponentId c, TriPortId port1, TriPortId port2)
void tliPMapParam
  (String am, int ts, String src, int line, TriComponentId c, TriPortId port1, TriPortId port2,
TciParameterListType tciPars, TciStatus encoderFailure, TriParameterList triPars)
void tliPUnmap
  (String am, int ts, String src, int line, TriComponentId c, TriPortId port1,
TriPortId port2)
void tliPUnmapParam
  (String am, int ts, String src, int line, TriComponentId c, TriPortId port1,
TriPortId port2, TciParameterListType tciPars, TciStatus encoderFailure, TriParameterList triPars)
void tliPClear
  (String am, int ts, String src, int line, TriComponentId c, TriPortId port)
void tliPStart
  (String am, int ts, String src, int line, TriComponentId c, TriPortId port)
void tliPStop
  (String am, int ts, String src, int line, TriComponentId c, TriPortId port)
void tliPHalt
  (String am, int ts, String src, int line, TriComponentId c, TriPortId port)

void tliEncode
  (String am, int ts, String src, int line, TriComponentId c, Value val, TciStatus encoderFailure,
TriMessage msg, String codec)
void tliDecode
  (String am, int ts, String src, int line, TriComponentId c, TriMessage msg,
TciStatus decoderFailure, Value val, String codec)

void tliTimeoutDetected
  (String am, int ts, String src, int line, TriComponentId c, TriTimerId timer)
void tliTimeoutMismatch
  (String am, int ts, String src, int line, TriComponentId c, TriTimerId timer,
TciNonValueTemplate timerTpl)
void tliTimeout
  (String am, int ts, String src, int line, TriComponentId c, TriTimerId timer,
TciNonValueTemplate timerTpl)
void tliTStart
  (String am, int ts, String src, int line, TriComponentId c, TriTimerId timer, TriTimerDuration dur)
void tliTStop
  (String am, int ts, String src, int line, TriComponentId c, TriTimerId timer, TriTimerDuration dur)
void tliTRead
  (String am, int ts, String src, int line, TriComponentId c, TriTimerId timer,
TriTimerDuration elapsed)
void tliTRunning
  (String am, int ts, String src, int line, TriComponentId c, TriTimerId timer, TimerStatus status)

void tliSEnter
  (String am, int ts, String src, int line, TriComponentId c, QualifiedName name,
TciParameterListType tciPars, String kind)
void tliSLeave
  (String am, int ts, String src, int line, TriComponentId c, QualifiedName name,
TciParameterListType tciPars, Value returnValue, String kind)

void tliVar
  (String am, int ts, String src, int line, TriComponentId c, QualifiedName name, Value varValue)
void tliModulePar
  (String am, int ts, String src, int line, TriComponentId c, QualifiedName name, Value parValue)

void tliGetVerdict(String am, int ts, String src, int line, TriComponentId c, VerdictValue verdict)
void tliSetVerdict
  (String am, int ts, String src, int line, TriComponentId c, VerdictValue verdict, String reason)

void tliLog(String am, int ts, String src, int line, TriComponentId c, String log)

void tliAEnter(String am, int ts, String src, int line, TriComponentId c)
void tliALeave(String am, int ts, String src, int line, TriComponentId c)
void tliADefaults(String am, int ts, String src, int line, TriComponentId c)
void tliAActivate

```

```

(String am, int ts, String src, int line, TriComponentId c, QualifiedName name,
TciParameterListType tciPars, Value ref)
void tliAdeactivate(String am, int ts, String src, int line, TriComponentId c, Value ref)
void tliANomatch(String am, int ts, String src, int line, TriComponentId c)
void tliARepeat(String am, int ts, String src, int line, TriComponentId c)
void tliAwait(String am, int ts, String src, int line, TriComponentId c)

void tliAction(String am, int ts, String src, int line, TriComponentId c, String action)

void tliMatch
  (String am, int ts, String src, int line, TriComponentId c, Value expr, TciValueTemplate tmpl)
void tliMatchMismatch
  (String am, int ts, String src, int line, TriComponentId c, Value expr, TciValueTemplate tmpl,
TciValueDifferenceList diffs);

void tliInfo
  (String am, int ts, String src, int line, TriComponentId c, int level, String info)

```

## 9.5 Data

TCI IDL ADT	ANSI C representation (Type definition)	Notes and comments
TciModuleIdType	QualifiedName	
TciModuleParameterType	typedef struct TciModuleParameterType { String parName; Value defaultValue; } TciModuleParameterType;	
TciModuleParameterListType	typedef struct TciModuleParameterListType { long int length; TciModuleParameterType *modParList; } TciModuleParameterListType;	
TciParameterType	typedef struct TciParameterType { String parName; TciParameterPassingModeType parPassMode; Value parValue; } TciParameterType;	
TciParameterPassingModeType	typedef enum { TCI_IN_PAR = 0, TCI_INOUT_PAR = 1, TCI_OUT_PAR = 2 } TciParameterPassingModeType;	
TciParameterListType	typedef struct TciParameterListType { long int length; TciParameterType *parList; } TciParameterListType;	length 0 shall be interpreted as "empty list".
TciParameterTypeListType	typedef struct TciParameterTypeListType { long int length; TciParameterTypeType *parList; } TciParameterTypeListType;	length 0 shall be interpreted as "empty list".

TCI IDL ADT	ANSI C representation (Type definition)	Notes and comments
TciParameterTypeType	<pre>typedef struct TciParameterTypeType {     Type parameterType;     TciParameterPassingModeType mode; } TciParameterTypeType;</pre>	
TciTestCaseIdListType	<pre>typedef struct TciTestCaseIdListType {     long int          length;     QualifiedName *idList; } TciTestCaseIdListType;</pre>	length 0 shall be interpreted as "empty list".
TciTypeClassType	<pre>typedef enum {     TCI_ADDRESS_TYPE = 0,     TCI_ANYTYPE_TYPE = 1,     TCI_BITSTRING_TYPE = 2,     TCI_BOOLEAN_TYPE = 3,     TCI_CHARSTRING_TYPE = 5,     TCI_COMPONENT_TYPE = 6,     TCI_ENUMERATED_TYPE = 7,     TCI_FLOAT_TYPE = 8,     TCI_HEXSTRING_TYPE = 9,     TCI_INTEGER_TYPE = 10,     TCI_OCTETSTRING_TYPE = 12,     TCI_RECORD_TYPE = 13,     TCI_RECORD_OF_TYPE = 14,     TCI_ARRAY_TYPE = 15,     TCI_SET_TYPE = 16,     TCI_SET_OF_TYPE = 17,     TCI_UNION_TYPE = 18,     TCI_UNIVERSAL_CHARSTRING_TYPE = 20,     TCI_VERDICT_TYPE = 21 } TciTypeClassType;</pre>	
TciTestComponentKindType	<pre>typedef enum {     TCI_CTRL_COMP,     TCI_MTC_COMP,     TCI_PTC_COMP,     TCI_SYS_COMP,     TCI_ALIVE_COMP } TciTestComponentKindType;</pre>	
TciBehaviourIdType	QualifiedName	
TciValueDifference	<pre>typedef struct TciValueDifference {     Value val;     TciValueTemplate tmpl;     String desc; } TciValueDifference;</pre>	
TciValueDifferenceList	<pre>typedef struct TciValueDifferenceList {     long int          length;     TciValueDifference* diffList; } TciValueDifferenceList;</pre>	length 0 shall be interpreted as "empty list".

## 9.6 Miscellaneous

TCI concept	ANSI C representation	Notes and comments
<b>Verdict representation</b>		
NONE	<code>const int TCI_VERDICT_NONE = 0</code>	Since the VerdictValue interface is defined in terms of integers, consensus must be established on which value defines which verdict.
PASS	<code>const int TCI_VERDICT_PASS = 1</code>	
INCONC	<code>const int TCI_VERDICT_INCONC = 2</code>	
FAIL	<code>const int TCI_VERDICT_FAIL = 3</code>	
ERROR	<code>const int TCI_VERDICT_ERROR = 4</code>	
USER_ERROR	<code>const int TCI_VERDICT_USER_ERROR = 5</code>	
<b>ComponentStatus</b>		
INACTIVE_C	<code>const int TCI_INACTIVE_C = 0</code>	
RUNNING_C	<code>const int TCI_RUNNING_C = 1</code>	
STOPPED_C	<code>const int TCI_STOPPED_C = 2</code>	
KILLED_C	<code>const int TCI_KILLED_C = 3</code>	
NULL_C	<code>const int TCI_NULL_C = 4</code>	
<b>TimerStatus</b>		
RUNNING_T	<code>const int TCI_RUNNING_T = 0</code>	
INACTIVE_T	<code>const int TCI_INACTIVE_T = 1</code>	
EXPIRED_T	<code>const int TCI_EXPIRED_T = 2</code>	
NULL_T	<code>const int TCI_NULL_T = 3</code>	
TciStatus		
TCI_OK	<code>const int TCI_OK = 0</code>	
TCI_ERROR	<code>const int TCI_ERROR = -1</code>	
<b>CharstringValue representation</b>		
TciCharString	<pre>typedef struct TciCharStringValue {     unsigned long int length;     char*          string; } TciCharStringValue</pre>	
<b>Universal Character[string] representation</b>		
Universal Char	<code>typedef unsigned char TciUCValue[4]</code>	
Universal Charstring	<pre>typedef struct TciUCStringValue {     unsigned long int length;     TciUCValue     *string; } TciUCStringValue;</pre>	

## 9.7 Optional parameters

Clause 7.4 defines that a reserved value shall be used to indicate the absence of an optional parameter. For the C language mapping an explicit null shall be used. The function `tciSetNull` can be used to set a value to null and `tciIsNull` can be used to check whether a value represents null. `tciIsNull` returns true if the value is null, false otherwise.

For example, if in the `tcReplyConnected` operation the value parameter shall be omitted, then a value `reply` shall be created and set to null; the operation invocation shall be:

```
tcSetNull(reply);  
tcReplyConnected (sender, receiver, signature, parameterList, reply).
```

## 10 C++ language mapping

### 10.1 Introduction

This clause introduces the TCI C++ language [9] mapping for the definitions given in clause 7.

### 10.2 Names and scopes

The namespace `ORG_ETSI_TTCN3_TCI` has been defined for the TCI C++ mapping, in order to avoid conflicts with the different names used, for example, in the C mapping.

C++ class identifiers are omitting the trailing "Type" at the end of the abstract definitions, e.g., the type `TciModuleIdType` is mapped to `TciModuleId` in C++.

### 10.3 Memory management

A general policy for memory management is not defined in this mapping. However, parameters are passed as pointers (or references) where possible, and a clone method has been added to the definition of every interface. The clone method can be used by the receiving entity to make a local copy where needed.

### 10.4 Error handling

No additional error handling has been defined for this mapping.

### 10.5 Type mapping

This clause introduces the TRI C++ language mapping for the abstract types defined in clause 7.2. The following concepts have been used:

- Pure virtual classes have been used following the concept of an interface.
- C++ types have been encapsulated under abstract definitions like `Tfloat` or `Tinteger`.

#### 10.5.1 Encapsulated C++ types

The following types have been defined in order to keep the definitions of data types and operations as general as possible:

- Boolean type definition: `typedef bool Tboolean`
- Integer type definition: `typedef long int Tinteger`
- Size type definition: `typedef unsigned long int Tsize`
- Index type definition: `typedef unsigned long int Tindex`
- Float type definition: `typedef double Tfloat`
- String type definition: `typedef std::string Tstring`
- Universal string type definition: `typedef std::wstring TuniversalString`
- Bit type definition: `typedef unsigned char Tbit`
- Char type definition: `typedef unsigned char Tchar`

## 10.5.2 General abstract data types

### 10.5.2.1 TciBehaviourId

Identifies a TTCN-3 behaviour functions. It is mapped to the following pure virtual class:

```
class TciBehaviourId: ORG_ETSI_TTCN3_TRI::QualifiedName {
public:

    virtual ~TciBehaviourId ();
    virtual Tboolean operator== (const TciBehaviourId &bid) const =0;
    virtual TciBehaviourId * clone () const =0;
    virtual Tboolean operator< (const TciBehaviourId &bid) const =0;
}
```

#### 10.5.2.1.1 Methods

- ~TciBehaviourId
- Destructor
- operator==
- Returns true if both objects are equal
- clone
- Return a copy of the TciBehaviourId
- operator<
- Operator < overload

### 10.5.2.2 TciModuleId

A value of TciModuleId specifies the name of a TTCN-3 module. It is mapped to the following pure virtual class:

```
class TciModuleId {
public:
    virtual ~TciModuleId ()
    virtual const Tstring & getObjectname() const = 0;
    virtual void setObjectName (const Tstring &p_name)=0;
    virtual Tboolean operator== (const TciModuleId &mid) const =0;
    virtual TciModuleId * clone () const =0;
    virtual Tboolean operator< (const TciModuleId &mid) const =0;
}
```

#### 10.5.2.2.1 Methods

- ~TciModuleId
- Destructor
- getObjectname
- Get the moduleId name
- setObjectName
- Set the moduleId name
- operator==
- Returns true if both objects are equal
- clone
- Return a copy of the TciModuleId
- operator<
- Operator < overload



### 10.5.2.3 TciModuleParameterId

A value of TciModuleParameterId specifies the name of a TTCN-3 module parameter as defined in the TTCN-3 module. It is mapped to the following pure virtual class:

```
class TciModuleParameterId : ORG_ETSI_TTCN3_TRI::QualifiedName {
public:
    virtual ~TciModuleParameterId ();
    virtual Tboolean operator== (const TciModuleParameterId &mparId) const =0;
    virtual TciModuleParameterId * clone () const =0;
    virtual Tboolean operator< (const TciModuleParameterId &mparId) const =0;
}
```

#### 10.5.2.3.1 Methods

- ~TciModuleParameterId
- Destructor
- operator==
- Returns true if both objects are equal
- clone
- Return a copy of the TciModuleParameterId
- operator<
- Operator < overload

### 10.5.2.4 TciTestCaseId

A value of TciModuleParameterId specifies the name of a TTCN-3 testcase as defined in the TTCN-3 module. It is mapped to the following pure virtual class:

```
class TciTestCaseId : TciBehaviourId {
public:
    virtual ~TciTestCaseId();
    virtual Tboolean operator== (const TciTestCaseId &tcid) const =0;
    virtual TciTestCaseId * clone () const =0;
    virtual Tboolean operator< (const TciTestCaseId &tcid) const =0;
}
```

#### 10.5.2.4.1 Methods

- ~TciTestCaseId
- Destructor
- operator==
- Returns true if both objects are equal
- clone
- Return a copy of the TciTestCaseId
- operator<
- Operator < overload

### 10.5.2.5 TciModuleIdList

A value of TciModuleIdList defines a list of TciModuleId elements. It is mapped to the following pure virtual class:

```
class TciModuleIdList {
public:
    virtual ~TciModuleIdList();
    virtual Tsize size () const =0;
    virtual Tboolean empty () const =0;
    virtual const TciModuleId *get (Tsize p_index) const =0;
```

```

virtual void clear ()=0;
virtual void push_back (const TciModuleId &comp)=0;
virtual Tboolean operator== (const TciModuleIdList &midList) const =0;
virtual TciModuleIdList * clone () const =0;
virtual Tboolean operator< (const TciModuleIdList &midList) const =0;
}

```

### 10.5.2.5.1 Methods

- ~TciModuleIdList
- Destructor
- size
- Return the size of the list
- empty
- Return true if the list is empty
- get
- Return the requested element
- clear
- Remove all the components from this list
- push\_back
- Add a component to the end of this list
- operator==
- Returns true if both objects are equal
- clone
- Return a copy of the TciModuleId
- operator<
- Operator < overload

### 10.5.2.6 TciModuleParameter

This abstract type is used to represent the parameter name and the default value of a module parameter. It is mapped to the following pure virtual class:

```

class TciModuleParameter {
public:
    virtual ~TciModuleParameter ();
    virtual const TciValue & getDefaultValue () const =0;
    virtual const Tstring & getModuleParameterName () const =0;
    virtual const TciModuleParameterId & getTciModuleParameterId () const =0;
    virtual Tboolean operator== (const TciModuleParameter &mpar) const =0;
    virtual TciModuleParameter * clone () const =0;
    virtual Tboolean operator< (const TciModuleParameter &mpar) const =0;
}

```

### 10.5.2.6.1 Methods

- ~TciModuleParameter
- Destructor
- getDefaultValue
- Return default value of the parameter
- getModuleParameterName
- Return parameter name
- getTciModuleParameterId

- Get the name of the module parameter as defined in the TTCN-3 module
- operator==
- Returns true if both objects are equal
- clone
- Return a copy of the TciModuleParameter
- operator<
- Operator < overload

### 10.5.2.7 TciModuleParameterList

A value of TciModuleParameterList is a list of TciModuleParameter elements. It is mapped to the following pure virtual class:

```
class TciModuleParameterList {
public:
    virtual ~TciModuleParameterList ();
    virtual Tsize size () const =0;
    virtual Tboolean empty () const =0;
    virtual const TciModuleParameter *get (Tindex p_index) const =0;
    virtual void clear ()=0;
    virtual void push_back (const TciModuleParameter &comp)=0;
    virtual Tboolean operator== (const TciModuleParameterList &mparList) const =0;
    virtual TciModuleParameterList * clone () const =0;
    virtual Tboolean operator< (const TciModuleParameterList &mparList) const =0;
}

```

#### 10.5.2.7.1 Methods

- ~TciModuleParameterList
- Destructor
- size
- Return the size of the list
- empty
- Return true if the list is empty
- get
- Retrieve the specified element
- clear
- Remove all components from this list
- push\_back
- Add a component to the end of this list
- operator==
- Returns true if both objects are equal
- clone
- Return a copy of the TciModuleParameterList
- operator<
- Operator < overload

### 10.5.2.8 TciParameterPassingMode

Defines the parameter passing mode. It is mapped to an enumeration:

```
typedef enum
{

```

```

    IN = 0,
    OUT = 1,
    INOUT = 2
} TciParameterPassingMode;

```

### 10.5.2.9 TciParameter

Includes a TTCN-3 Value and a TciParameterPassingMode. It is mapped to the following pure virtual class:

```

class TciParameter {
public:
    virtual ~TciParameter ();
    virtual const TciValue & getValue () const =0;
    virtual void setValue (TciValue &value)=0;
    virtual const TciParameterPassingMode &getParameterPassingMode () const =0;
    virtual void setParameterPassingMode (const TciParameterPassingMode &mode)=0;
    virtual const Tstring & getParameterName () const =0;
    virtual void setParameterName (const Tstring &name)=0;
    virtual Tboolean operator== (const TciParameter &param) const =0;
    virtual TciParameter * clone () const =0;
    virtual Tboolean operator< (const TciParameter &param) const =0;
}

```

#### 10.5.2.9.1 Methods

- ~TciParameter
- Destructor
- getValue
- Retrieve the TTCN-3 value
- setValue
- Set the TTCN-3 value
- getParameterPassingMode
- Return the parameter passing mode
- setParameterPassingMode
- Set the parameter passing mode
- getParameterName
- Return the name of the parameter
- setParameterName
- Set the name of the parameter
- operator==
- Returns true if both objects are equal
- clone
- Return a copy of the TciParameter
- operator<
- Operator < overload

#### 10.5.2.10 TciParameterList

Defines a list of TciParameter elements. It is mapped to the following pure virtual class:

```

class TciParameterList {
public:
    virtual ~TciParameterList ();
    virtual Tsize size () const =0;
    virtual Tboolean empty () const =0;
}

```

```

virtual const TciParameter *get (Tindex p_index) const =0;
virtual void clear ()=0;
virtual void push_back (const TciParameter &comp)=0;
virtual Tboolean operator== (const TciParameterList &param) const =0;
virtual TciParameterList * clone () const =0;
virtual Tboolean operator< (const TciParameterList &param) const =0;
}

```

### 10.5.2.10.1 Methods

- ~TciParameterList
- Destructor
- size
- Return the size of the list
- empty
- Return true if the list is empty
- get
- Get the specified element
- clear
- Remove all the components from this list
- push\_back
- Add a component to the end of this list
- operator==
- Returns true if both objects are equal
- clone
- Return a copy of the TciParameterList
- operator<
- Operator < overload

### 10.5.2.11 TciParameterType

Includes a TTCN-3 Type and a TciParameterPassingMode. It is mapped to the following pure virtual class:

```

class TciParameterType {
public:
    virtual ~TciParameterType ();
    virtual const TciType & getType () const =0;
    virtual const TciParameterPassingMode & getParameterPassingMode () const =0;
    virtual Tboolean operator== (const TciParameterType &parType) const =0;
    virtual TciParameterType * clone () const =0;
    virtual Tboolean operator< (const TciParameterType &parType) const =0;
}

```

### 10.5.2.11.1 Methods

- ~TciParameterType
- Destructor
- getType
- Return the TTCN-3 Type
- getParameterPassingMode
- Get the parameter passing mode
- operator==

- Returns true if both objects are equal
- clone
- Return a copy of the TciParameterType
- operator<
- Operator < overload

### 10.5.2.12 TciParameterTypeList

Specifies a list of TciParameterType elements. It is mapped to the following pure virtual class:

```
class TciParameterTypeList {
public:
    virtual ~TciParameterTypeList ();
    virtual Tsize size () const =0;
    virtual Tboolean empty () const =0;
    virtual const TciParameterType *get (Tindex p_position) const =0;
    virtual void clear ()=0;
    virtual void push_back (const TciParameterType &comp)=0;
    virtual Tboolean operator== (const TciParameterTypeList &ptypeList) const =0;
    virtual TciParameterTypeList * clone () const =0;
    virtual Tboolean operator< (const TciParameterTypeList &ptypeList) const =0;
}

```

#### 10.5.2.12.1 Methods

- ~TciParameterTypeList
- Destructor
- size
- Return the size of the list
- empty
- Returns true if the list is empty
- get
- Return the requested element
- clear
- Remove all the components from this list
- push\_back
- Add a component to the end of this list
- operator==
- Returns true if both objects are equal
- clone ()
- Returns a copy of the TciParameterTypeList
- operator< (const TciParameterTypeList &ptypeList)
- Operator < overload

### 10.5.2.13 TciTestComponentKind

Defines the test component kind. It is mapped to an enumeration:

```
typedef enum
{
    SYSTEM_COMP = 0,
    PTC_COMP = 1,
    PTC_ALIVE_COMP = 2,
    MTC_COMP = 3,
    CTRL_COMP = 4
}

```

```
} TciTestComponentKind;
```

### 10.5.2.14 TciTypeClass

Defines the type class. It is mapped to an enumeration:

```
typedef enum
{
    TCI_ADDRESS = 0,
    TCI_ANYTYPE = 1,
    TCI_BITSTRING = 2,
    TCI_BOOLEAN = 3,
    TCI_CHARSTRING = 5,
    TCI_COMPONENT = 6,
    TCI_ENUMERATED = 7,
    TCI_FLOAT = 8,
    TCI_HEXSTRING = 9,
    TCI_INTEGER = 10,
    TCI_OCTETSTRING = 12,
    TCI_RECORD = 13,
    TCI_RECORD_OF = 14,
    TCI_ARRAY = 15,
    TCI_SET = 16,
    TCI_SET_OF = 17,
    TCI_UNION = 18,
    TCI_UNIVERSAL_CHARSTRING = 20,
    TCI_VERDICT = 21
} TciTypeClass;
```

### 10.5.2.15 TciTestCaseIdList

Specifies a list of TciTestCaseId elements. It is mapped to the following pure virtual class:

```
class TciTestCaseIdList {
public:
    virtual ~TciTestCaseIdList ();
    virtual Tsize size () const =0;
    virtual Tboolean empty () const =0;
    virtual const TciTestCaseId *get (Tindex p_position) const =0;
    virtual void clear ()=0;
    virtual void push_back (const TciTestCaseId &comp)=0;
    virtual Tboolean operator== (const TciTestCaseIdList &ptypeList) const =0;
    virtual TciTestCaseIdList * clone () const =0;
    virtual Tboolean operator< (const TciTestCaseIdList &ptypeList) const =0;
}

```

#### 10.5.2.15.1 Methods

- ~TciTestCaseIdList
- Destructor
- size
- Return the size of the list
- empty
- Returns true if the list is empty
- get
- Return the requested element
- clear
- Remove all the components from this list
- push\_back
- Add a component to the end of this list
- operator==
- Returns true if both objects are equal

- clone ()
- Returns a copy of the TciTestCaseIdList
- operator< (const TciTestCaseIdList &ptypeList)
- Operator < overload

### 10.5.3 Abstract TTCN-3 data types and values

#### 10.5.3.1 TciType

A value of TciType represents one of the TTCN-3 types in a TTCN-3 module. It is mapped to the following pure virtual class:

```
class TciType {
public:
    virtual ~TciType ();
    virtual const TciModuleId & getDefiningModule () const =0;
    virtual const Tstring & getName () const =0;
    virtual const TciTypeClass & getTypeClass () const =0;
    virtual const Tstring & getTypeEncoding () const =0;
    virtual const Tstring & getTypeEncodingVariant () const =0;
    virtual const std::vector<Tstring*> & getTypeExtension() const =0;
    virtual TciValue * newInstance () const =0;
    virtual Tboolean operator== (const TciType &typ) const =0;
    virtual TciType * clone () const =0;
    virtual Tboolean operator< (const TciType &typ) const =0;
}

```

##### 10.5.3.1.1 Methods

- ~TciType
- Destructor
- getDefiningModule
- Return the defining module as defined in the TTCN-3 module
- getName
- Return type name as defined in the TTCN-3 module
- getTypeClass
- Return this type class
- getTypeEncoding
- Return type encoding as defined in the TTCN-3 module
- getTypeEncodingVariant
- Return encoding variant as defined in the TTCN-3 module
- getTypeExtension
- Return type extension as defined in the TTCN-3 module
- newInstance
- Return a new Value instance of this type
- operator==
- Return true if the types are equal
- clone
- Return a copy of the TciType
- operator<
- Operator < overload



### 10.5.3.2 TciValue

A value of TciValue represents TTCN-3 values for a given type. It is mapped to the following pure virtual class:

```
class TciValue {
public:
    virtual ~TciValue ();
    virtual const TciType & getType () const =0;
    virtual const Tstring & getValueEncoding () const =0;
    virtual const Tstring & getValueEncodingVariant () const =0;
    virtual Tboolean notPresent () const =0;
    virtual Tboolean operator== (const TciValue &val) const =0;
    virtual TciValue * clone () const =0;
    virtual Tboolean operator< (const TciValue &val) const =0;
}
```

#### 10.5.3.2.1 Methods

- ~TciValue
- Destructor
- getType
- Returns the type of the specified value
- getValueEncoding
- Returns the value encoding attribute as defined in the TTCN-3 module
- getValueEncodingVariant
- Returns the value encoding variant attribute as defined in the TTCN-3 module
- notPresent
- Returns true if the specified value is omit
- operator==
- Returns true if both objects are equal
- clone
- Return a copy of the TciValue
- operator<
- Operator < overload

### 10.5.3.3 IntegerValue

TTCN-3 integer value support. It is mapped to the following pure virtual class:

```
class IntegerValue : public virtual TciValue {
public:
    virtual ~IntegerValue ();
    virtual Tinteger getInt () const =0;
    virtual void setInt (Tinteger p_value)=0;
    virtual Tboolean operator== (const IntegerValue &intVal) const =0;
    virtual IntegerValue * clone () const =0;
    virtual Tboolean operator< (const IntegerValue &intVal) const =0;
}
```

#### 10.5.3.3.1 Methods

- ~IntegerValue
- Destructor
- getInt3333
- Return integer value

- setInt
- Set integer value
- operator==
- Returns true if both objects are equal
- clone
- Return a copy of the IntegerValue
- operator<
- Operator < overload

### 10.5.3.4 FloatValue

TTCN-3 float value support. It is mapped to the following pure virtual class:

```
class FloatValue : public virtual TciValue {
public:
    virtual ~FloatValue ();
    virtual Tfloat getFloat () const =0;
    virtual void setFloat (Tfloat p_floatValue)=0;
    virtual Tboolean operator== (const FloatValue &floatVal) const =0;
    virtual FloatValue * clone () const =0;
    virtual Tboolean operator< (const FloatValue &floatVal) const =0;
}
```

#### 10.5.3.4.1 Methods

- ~FloatValue
- Destructor
- getFloat
- Return the float value
- setFloat
- Set float value
- operator==
- Returns true if both objects are equal
- clone
- Return a copy of the FloatValue
- operator<
- Operator < overload

### 10.5.3.5 BooleanValue

TTCN-3 boolean values support. It is mapped to the following pure virtual class:

```
class BooleanValue : public virtual TciValue {
public:
    virtual ~BooleanValue ();
    virtual Tboolean getBoolean () const =0;
    virtual void setBoolean (Tboolean p_booleanValue)=0;
    virtual Tboolean operator== (const BooleanValue &booleanVal) const =0;
    virtual BooleanValue * clone () const =0;
    virtual Tboolean operator< (const BooleanValue &booleanVal) const =0;
}
```

#### 10.5.3.5.1 Methods

- ~BooleanValue

- Destructor
- getBoolean
- Return the boolean value
- setBoolean
- Set the variable to booleanValue
- operator==
- Returns true if both objects are equal
- clone
- Return a copy of the BooleanValue
- operator<
- Operator < overload

### 10.5.3.6 CharstringValue

TTCN-3 charstring value support. It is mapped to the following pure virtual class:

```
class CharstringValue : public virtual TciValue {
public:
    virtual ~CharstringValue ();
    virtual char getChar (Tindex p_position) const =0;
    virtual Tsize getLength () const =0;
    virtual const Tstring & getString () const =0;
    virtual void setChar (Tsize p_position, char p_char)=0;
    virtual void setLength (Tsize p_length)=0;
    virtual void setString (const Tstring &p_charValue)=0;
    virtual Tboolean operator== (const CharstringValue &charStr) const =0;
    virtual CharstringValue * clone () const =0;
    virtual Tboolean operator< (const CharstringValue &charStr) const =0;
}

```

#### 10.5.3.6.1 Methods

- ~CharstringValue
- Destructor
- getChar
- Return the char at the specified position
- getLength
- Return length of the string
- getString
- Return the value of the string
- setChar
- Set the char at the specified position
- setLength
- Set length of the string
- setString
- Set the value of the string
- operator==
- Returns true if both objects are equal
- clone

- Return a copy of the CharstringValue
- operator<
- Operator < overload

### 10.5.3.7 UniversalCharstringValue

TTCN-3 universal charstring value support. It is mapped to the following pure virtual class:

```
class UniversalCharstringValue : public virtual TciValue {
public:
    virtual ~UniversalCharstringValue ();
    virtual wchar_t getChar (Tindex p_position) const =0;
    virtual Tsize getLength () const =0;
    virtual const TuniversalString & getString () const =0;
    virtual void setChar (Tindex p_position, const wchar_t p_ucValue)=0;
    virtual void setLength (Tsize p_length)=0;
    virtual void setString (const TuniversalString &p_ucsValue)=0;
    virtual Tboolean operator== (const UniversalCharstringValue &uniCharstr) const =0;
    virtual UniversalCharstringValue * clone () const =0;
    virtual Tboolean operator< (const UniversalCharstringValue &uniCharstr) const =0;
}
```

#### 10.5.3.7.1 Methods

- ~UniversalCharstringValue
- Destructor
- getChar
- Return the requested element
- getLength
- Return the length of the universal charstring
- getString
- Return the textual representation of the string
- setChar
- Set the char at the specified position
- setLength
- Set the length of the string
- setString
- Set the value of the string
- operator==
- Returns true if both objects are equal
- clone
- Return a copy of the UniversalCharstringValue
- operator<
- Operator < overload

### 10.5.3.8 BitstringValue

TTCN-3 bitstring value support. It is mapped to the following pure virtual class:

```
class BitstringValue : public virtual TciValue {
public:
    virtual ~BitstringValue ();
    virtual Tbit getBit (Tindex p_position) const =0;
    virtual Tsize getLength () const =0;
    virtual const Tstring & getString () const =0;
```

```

virtual void setBit (Tindex p_position, Tbit p_bsValue)=0;
virtual void setLength (Tindex p_new_length)=0;
virtual void setString (const Tstring &p_bsValue)=0;
virtual Tboolean operator== (const BitstringValue &bitStr) const =0;
virtual BitstringValue * clone () const =0;
virtual Tboolean operator< (const BitstringValue &bitStr) const =0;
}

```

### 10.5.3.8.1 Methods

- ~BitstringValue
- Destructor
- getBit
- Returns the bit at the specified position
- getLength
- Returns the length of the string
- getString
- Set the value of the string
- setBit
- Set the bit value at the specified position
- setLength
- Set the length of the string
- setString
- Set the string value
- operator==
- Returns true if both objects are equal
- clone
- Return a copy of the BitstringValue
- operator<
- Operator < overload

### 10.5.3.9 OctetstringValue

TTCN-3 octetstring value support. It is mapped to the following pure virtual class:

```

class OctetstringValue : public virtual TciValue {
public:
    virtual ~OctetstringValue ();
    virtual Tsize getLength () const =0;
    virtual const Tchar getOctet (Tindex p_position) const =0;
    virtual const Tstring &getString () const =0;
    virtual void setLength (Tsize p_length)=0;
    virtual void setOctet (Tindex p_position, Tchar p_ochar)=0;
    virtual void setString (const Tstring &p_osValue)=0;
    virtual Tboolean operator== (const OctetstringValue &octStr) const =0;
    virtual OctetstringValue * clone () const =0;
    virtual Tboolean operator< (const OctetstringValue &octStr) const =0;
}

```

#### 10.5.3.9.1 Methods

- ~OctetstringValue
- Destructor
- getLength

- Return the length of the string
- `getOctet`
- Return the textual representation of the octetchar at the specified position
- `getString`
- Set the string value
- `setLength`
- Set the length of the string
- `setOctet`
- Set the char at specified position
- `setString`
- Set the value of the string
- `operator==`
- Returns true if both objects are equal
- `clone`
- Return a copy of the `OctetstringValue`
- `operator<`
- Operator < overload

### 10.5.3.10 HexstringValue

TTCN-3 hexstring value support. It is mapped to the following pure virtual class:

```
class HexstringValue : public virtual TciValue {
public:
    virtual ~HexstringValue ();
    virtual Tchar getHex (Tindex p_position) const =0;
    virtual Tsize getLength () const =0;
    virtual const Tstring & getString () const =0;
    virtual void setHex (Tindex p_position, Tchar p_hcValue)=0;
    virtual void setLength (Tsize p_length)=0;
    virtual void setString (const Tstring &p_hsValue)=0;
    virtual Tboolean operator== (const HexstringValue &hexStr) const =0;
    virtual HexstringValue * clone () const =0;
    virtual Tboolean operator< (const HexstringValue &hexStr) const =0;
}
```

#### 10.5.3.10.1 Methods

- `~HexstringValue`
- Destructor
- `getHex`
- Return the element at the specified position
- `getLength`
- Return the length of the string
- `getString`
- Return the string value
- `setHex`
- Set the hex value at the specified position
- `setLength`

- Set the length of the string
- setString
- Set the value of the string
- operator==
- Returns true if both objects are equal
- clone
- Return a copy of the HexstringValue
- operator<
- Operator < overload

### 10.5.3.11 RecordValue

TTCN-3 record value support. It is mapped to the following pure virtual class:

```
class RecordValue : public virtual TciValue {
public:
    virtual ~RecordValue ();
    virtual const TciValue &getField (const Tstring &p_field_name) const =0;
    virtual void setField (const Tstring &p_field_name,const TciValue &p_new_value)=0;
    virtual const std::vector< Tstring *> &getFieldNames () const =0;
    virtual void setFieldOmitted (const Tstring &fieldName)=0;
    virtual Tboolean operator== (const RecordValue &rec) const =0;
    virtual RecordValue * clone () const =0;
    virtual Tboolean operator< (const RecordValue &rec) const =0;
}
```

#### 10.5.3.11.1 Methods

- ~RecordValue
- Destructor
- getField
- Return a reference to the field name
- setField
- Set the value of a field
- getFieldNames
- Return a list which containing the names of all the fields
- setFieldOmitted
- Set omit in one field
- operator==
- Returns true if both objects are equal
- clone
- Return a copy of the RecordValue
- operator<
- Operator < overload

### 10.5.3.12 RecordOfValue

TTCN-3 record of value support. It is mapped to the following pure virtual class:

```
class RecordOfValue : public virtual TciValue {
public:
    virtual ~RecordOfValue ();
    virtual const TciValue &getField (Tindex p_position)=0;
    virtual void setField (Tindex p_position, const TciValue &p_value)=0;
    virtual void appendField (const TciValue &p_value)=0;
    virtual const TciType &getElementType () const =0;
}
```

```

virtual Tsize getLength () const =0;
virtual void setLength (Tsize p_length)=0;
virtual Tindex getOffset() const =0;
virtual Tboolean operator== (const RecordOfValue &recOf) const =0;
virtual RecordOfValue * clone () const =0;
virtual Tboolean operator< (const RecordOfValue &recOf) const =0;
}

```

### 10.5.3.12.1 Methods

~RecordOfValue

Destructor

getField

Return the field at the specified position

setField

Set the value at the specified position

appendField

Add a value at the end of the record of

getElementType

Return the type of the elements of this record of

getLength

Return the length of the object

setLength

Set length of the record of

getOffset

For arrays, return the lower index bound used in the type definition of arrays. Return 0 for record of and set of

operator==

Returns true if both objects are equal

clone

Return a copy of the RecordOfValue

operator<

Operator < overload

### 10.5.3.13 UnionValue

TTCN-3 union value support. It is mapped to the following pure virtual class:

```

class UnionValue : public virtual TciValue {
public:
    virtual ~UnionValue ()
    virtual void setVariant (const Tstring &p_variantName, const TciValue &p_value)=0;
    virtual const TciValue & getVariant (const Tstring &p_variantName) const =0;
    virtual const Tstring & getPresentVariantName () const =0;
    virtual const std::set< Tstring * > & getVariantNames () const =0;
    virtual Tboolean operator== (const UnionValue &unionVal) const =0;
    virtual UnionValue * clone () const =0;
    virtual Tboolean operator< (const UnionValue &unionVal) const =0;
}

```

#### 10.5.3.13.1 Methods

- ~UnionValue
- Destructor
- setVariant
- Set the variant name to a value
- getVariant
- Return the value of the variant if exists



- `getPresentVariantName`
- Return the name of the current variant value. null if no initialized
- `getVariantNames`
- Return a list which contains the variant names as defined in the TTCN-3 module
- `operator==`
- Returns true if both objects are equal
- `clone`
- Return a copy of the UnionValue
- `operator<`
- Operator < overload

#### 10.5.3.14 EnumeratedValue

TTCN-3 enumerated value support. It is mapped to the following pure virtual class:

```
class EnumeratedValue : public virtual TciValue {
public:
    virtual ~EnumeratedValue ();
    virtual const Tstring & getEnum () const =0;
    virtual void setEnum (const Tstring &p_value)=0;
    virtual Tinteger getInt () const =0;
    virtual void setInt(Tinteger p_int);
    virtual Tboolean operator== (const EnumeratedValue &enumVal) const =0;
    virtual EnumeratedValue * clone () const =0;
    virtual Tboolean operator< (const EnumeratedValue &enumVal) const =0;
}
```

##### 10.5.3.14.1 Methods

- `~EnumeratedValue`
- Destructor
- `getEnum`
- Return current value
- `setEnum`
- Set the enumeration value
- `getInt`
- Return current integer value
- `setInt`
- Set the integer value
- `operator==`
- Returns true if both objects are equal
- `clone`
- Return a copy of the EnumeratedValue
- `operator<`
- Operator < overload

#### 10.5.3.15 VerdictValue

TTCN-3 verdict value support. It is mapped to the following pure virtual class:

```
class VerdictValue : public virtual TciValue {
public:
    virtual ~VerdictValue ();
    virtual const VerdictValueEnum & getVerdict () const =0;
    virtual void setVerdict (const VerdictValueEnum & p_enum)=0;
    virtual Tboolean operator== (const VerdictValue &verdictVal) const =0;
    virtual VerdictValue * clone () const =0;
    virtual Tboolean operator< (const VerdictValue &verdictVal) const =0;
}
```

### 10.5.3.15.1 Methods

- ~VerdictValue
- Destructor
- getVerdict
- Return the value of the verdict
- setVerdict
- Set the value of the verdict
- operator==
- Returns true if both objects are equal
- clone
- Return a copy of the VerdictValue
- operator<
- Operator < overload

### 10.5.3.16 VerdictValueEnum

Defines verdict values as an enumeration:

```
typedef enum
{
    NONE = 0,
    PASS = 1,
    FAIL = 2,
    INCONC = 3,
    ERROR = 4,
    USER_ERROR = 5
} VerdictValueEnum;
```

### 10.5.3.17 AddressValue

TTCN-3 address value support. It is mapped to the following pure virtual class:

```
class AddressValue {
public:
    virtual ~AddressValue ();
    virtual const TciValue & getAddress () const =0;
    virtual void setAddress ( const TciValue& T)=0;
    virtual Tboolean operator== (const AddressValue &addr) const =0;
    virtual AddressValue * cloneAddressValue () const =0;
    virtual Tboolean operator< (const AddressValue &addr) const =0;
}
```

#### 10.5.3.17.1 Methods

- ~AddressValue
- Destructor
- getAddress
- Return the value of the address
- setAddress
- Set the value of the address
- operator==
- Returns true if both objects are equal
- clone
- Return a copy of the AddressValue
- operator<

- Operator < overload

## 10.5.4 Abstract logging types

### 10.5.4.1 TciValueTemplate

Interface that defines the concrete operations of the TTCN-3 template. It is mapped to the following pure virtual class:

```
class TciValueTemplate {
public:
    virtual ~TciValueTemplate ();
    virtual Tboolean isOmit () const =0;
    virtual Tboolean isAny () const =0;
    virtual Tboolean isAnyOrOmit () const =0;
    virtual const Tstring & getTemplateDef () const =0;
    virtual Tboolean operator== (const TciValueTemplate &vtempl) const =0;
    virtual TciValueTemplate * clone () const =0;
    virtual Tboolean operator< (const TciValueTemplate &vtempl) const =0;
}

```

#### 10.5.4.1.1 Methods

- ~TciValueTemplate ()
- Destructor
- isOmit ()
- Return true if value of template is omit
- isAny ()
- Return true if value of template is any
- isAnyOrOmit ()
- Return true value of template if any or omit
- getTemplateDef ()
- Return the template definition as defined in the TTCN-3 module
- operator== (const TciValueTemplate &vtempl)
- Returns true if both objects are equal
- clone ()
- Return a copy of the TciValueTemplate
- operator< (const TciValueTemplate &vtempl)
- Operator < overload

### 10.5.4.2 TciNonValueTemplate

Support *all* and *any* matching mechanisms over TTCN-3 components and timers. It is mapped to the following pure virtual class:

```
class TciNonValueTemplate {
public:
    virtual ~TciNonValueTemplate ();
    virtual Tboolean isAny () const =0;
    virtual Tboolean isAll () const =0;
    virtual const Tstring & getTemplateDef () const =0;
    virtual Tboolean operator== (const TciNonValueTemplate &nvtempl) const =0;
    virtual TciNonValueTemplate * clone () const =0;
    virtual Tboolean operator< (const TciNonValueTemplate &nvtempl) const =0;
}

```

#### 10.5.4.2.1 Methods

- ~TciNonValueTemplate ()

- Destructor
- `isAny ()`
- Return true if value is any
- `isAll ()`
- Return true if is value all
- `getTemplateDef ()`
- Return template definition as defined in the TTCN-3 module
- `operator== (const TciNonValueTemplate &nvtempl)`
- Returns true if both objects are equal
- `clone ()`
- Return a copy of the `TciNonValueTemplate`
- `operator< (const TciNonValueTemplate &nvtempl)`
- Operator < overload

### 10.5.4.3 TciValueList

A list of `TciValues`. It is mapped to the following pure virtual class:

```
class TciValueList {
public:
    virtual ~TciValueList (void);
    virtual Tsize size () const =0;
    virtual Tboolean empty () const =0;
    virtual const TciValue *get (Tindex index) const =0;
    virtual void clear ()=0;
    virtual void add (const TciValue &comp)=0;
    virtual Tboolean operator== (const TciValueList &valList) const =0;
    virtual TciValueList * clone () const =0;
    virtual Tboolean operator< (const TciValueList &valList) const =0;
}
```

#### 10.5.4.3.1 Methods

- `~TciValueList ()`
- Destructor
- `size ()`
- Return the size of the list
- `empty ()`
- Return true if the list is empty
- `get (Tindex index)`
- Return the value at the specified position
- `clear ()`
- Remove all the elements from this list
- `add (const TciValue &comp)`
- Add an element to the end of this list
- `operator== (const TciValueList &valList)`
- Returns true if both objects are equal
- `clone ()`
- Return a copy of the `TciValueList`
- `operator< (const TciValueList &valList)`

- Operator < overload

#### 10.5.4.4 TciValueDifference

Represents the differences during a match operation. It is mapped to the following pure virtual class:

```
class TciValueDifference {
public:
    virtual ~TciValueDifference ();
    virtual const TciValue & getValue () const =0;
    virtual void setValue (TciValue &val)=0;
    virtual const TciValueTemplate & getTciValueTemplate () const =0;
    virtual void setTciValueTemplate (TciValueTemplate &valT)=0;
    virtual const Tstring & getDescription () const =0;
    virtual void setDescription (const Tstring &descr)=0;
    virtual Tboolean operator== (const TciValueDifference &vdiff) const =0;
    virtual TciValueDifference * clone () const =0;
    virtual Tboolean operator< (const TciValueDifference &vdiff) const =0;
}
```

##### 10.5.4.4.1 Methods

- ~TciValueDifference ()
- Destructor
- getValue ()
- Return the value definition
- setValue (TciValue &val)
- Set the value definition
- getTciValueTemplate ()
- Return the template definition
- setTciValueTemplate (TciValueTemplate &valT)
- Set the template definition
- getDescription ()
- Return a string which describes the difference
- setDescription (const Tstring &descr)
- Set description
- operator== (const TciValueDifference &vdiff)
- Returns true if both objects are equal
- clone ()
- Return a copy of the TciValueDifference
- operator< (const TciValueDifference &vdiff)
- Operator < overload

#### 10.5.4.5 TciValueDifferenceList

Collection of TciValueDifferences. It is mapped to the following pure virtual class:

```
class TciValueDifferenceList {
public:
    virtual ~TciValueDifferenceList ();
    virtual Tsize size () const =0;
    virtual Tboolean empty () const =0;
    virtual const TciValueDifference *get (Tinteger p_position) const =0;
    virtual void clear ()=0;
    virtual void add (const TciValueDifference &comp)=0;
    virtual Tboolean operator== (const TciValueDifferenceList &vdiff) const =0;
    virtual TciValueDifferenceList * clone () const =0;
}
```

```

    virtual Tboolean operator< (const TciValueDifferenceList &vdList) const =0;
}

```

#### 10.5.4.5.1 Methods

- `~TciValueDifferenceList ()`
- Destructor
- `size ()`
- Return the size of the list
- `empty ()`
- Return true if this list contains no elements
- `get (Tinteger p_position)`
- Return the requested difference
- `clear ()`
- Remove all the components from this list
- `add (const TciValueDifference &comp)`
- Add a component to the end of the list
- `operator== (const TciValueDifferenceList &vdList)`
- Returns true if both objects are equal
- `clone ()`
- Return a copy of the TciValueDifferenceList
- `operator< (const TciValueDifferenceList &vdList)`
- Operator < overload

#### 10.5.4.6 ComponentStatus

Defines component status as an enumeration:

```

typedef enum
{
    INACTIVE_C = 0,
    RUNNING_C = 1,
    STOPPED_C = 2,
    KILLED_C = 3
    NULL_C = 4
} ComponentStatus;

```

#### 10.5.4.7 TimerStatus

Defines timer status as an enumeration:

```

typedef enum
{
    RUNNING_T = 0,
    INACTIVE_T = 1,
    EXPIRED_T = 2
    NULL_T = 3
} TimerStatus;

```

#### 10.5.4.8 TciStatus

Defines TCI status as an enumeration:

```

typedef enum
{
    TCI_OK = 0,
    TCI_ERROR = -1
} TciStatus;

```

## 10.6 Operations mapping

### 10.6.1 TCI-TM

#### 10.6.1.1 TciTmRequired

Specifies the operations the TM requires from TE. It is mapped to the following interface:

```
//Destructor
virtual ~TciTmRequired ();

//Selects the indicated module for execution
virtual void tciRootModule (const TciModuleId *moduleName)=0;

//The TE provides to the management a list of imported modules of the root module
virtual const TciModuleIdList * getImportedModules () const =0;

//The TE provides to the management a list of module parameters of the identified module
virtual const TciModuleParameterList * tciGetModuleParameters (const TciModuleId *moduleName)=0;

//The TE provides to the management a list of test cases
virtual const TciTestCaseIdList * tciGetTestCases () const =0;

//The TE provides to the management a list of parameter types of the given test case
virtual const TciParameterTypeList * tciGetTestCaseParameters (const TciTestCaseId *testCaseId)
const =0;

//The TE provides to the management a list of system ports of the given test case
virtual const TriPortIdList * tciGetTestCaseTSI (const TciTestCaseId &testCaseId) const =0;

//Starts a testcase in the currently selected module with the given parameters
virtual void tciStartTestCase (const TciTestCaseId *testCaseId, const TciParameterList
*parameterList)=0;

//Stops the testcase currently being executed
virtual void tciStopTestCase ()=0;

//Starts the control part of the selected module
virtual const TriComponentId * tciStartControl ()=0;

//Stops execution of the control part
virtual void tciStopControl ()=0;
```

#### 10.6.1.2 TciTmProvided

Specifies the operation the TM has to provide to the TE. It is mapped to the following interface:

```
//Destructor
virtual ~TciTmProvided ();

//Indicates to the TM that a test case with testCaseId has been started
virtual void tciTestCaseStarted (const TciTestCaseId &testCaseId, const TciParameterList
&parameterList, const Tfloat &timer)=0;

//Called to indicate that the test case has terminated execution
virtual void tciTestCaseTerminated (const VerdictValue &verdict, const TciParameterList
&parameterList)=0;

//Called to indicate that the control part of the selected module has just terminated execution
virtual void tciControlTerminated ()=0;

//The management provides to the TE a Value for the indicated parameterId
virtual TciValue * tciGetModulePar (const TciModuleParameterId &parameterId)=0;

//Indicates the occurrence of an unrecoverable error situation
virtual void tciError (const Tstring &message)=0;

//The TE indicates a message of a test component
virtual void tciLog (const TriComponentId &testComponentId, const Tstring &message)=0;
```

### 10.6.2 TCI-CD

#### 10.6.2.1 TciCdRequired

This class defines the TCI\_CD required interface. It is mapped to the following interface:

```

//Destructor
virtual ~TciCdRequired ();

//Returns a type representing a ttcn type
virtual const TciType * getTypeForName (const Tstring typeName) const =0;

//Constructs and returns a basic TTCN-3 integer type
virtual const TciType & getInteger () const =0;

//Constructs and returns a basic TTCN-3 float type
virtual const TciType & getFloat () const =0;

//Constructs and returns a basic TTCN-3 boolean type
virtual const TciType & getBoolean () const =0;

//Constructs and returns a basic TTCN-3 charstring type
virtual const TciType & getCharstring () const =0;

//Constructs and returns a basic TTCN-3 universal charstring type
virtual const TciType & getUniversalCharstring () const =0;

//Constructs and returns a basic TTCN-3 hexstring type
virtual const TciType & getHexstring () const =0;

//Constructs and returns a basic TTCN-3 bitstring type
virtual const TciType & getBitstring () const =0;

//Constructs and returns a basic TTCN-3 octetstring type
virtual const TciType & getOctetstring () const =0;

//Constructs and returns a basic TTCN-3 verdict type
virtual const TciType & getVerdict () const =0;

//The TE will be notified about an unrecoverable error situation within the CD
virtual void tciErrorReq (const Tstring message)=0;

```

### 10.6.2.2 TciCdProvided

This class defines the TCI\_CD provided interface. It is mapped to the following interface:

```

//Destructor
virtual ~TciCdProvided ();

//This operation is called whenever the TE has to decode and encode value
virtual TciValue * decode (const TriMessage *p_message, const TciType *p_decodingHypothesis)=0;

//This operation is called whenever the TE has to encode a Value
virtual TriMessage * encode (const TciValue *p_value)=0;

```

## 10.6.3 TCI-CH

### 10.6.3.1 TciChRequired

This class defines the TCI\_CH required interface. It is mapped to the following interface:

```

//Default destructor
virtual ~TciChRequired ();

//This operation is called by the CH at the local TE when at a remote TE a provided
//tciSendConnected has been called
virtual void tciEnqueueMsgConnected (const TriPortId *sender, const TriComponentId *receiver,
const TciValue *rcvdMessage)=0;

//This operation is called by the CH at the local TE when at a remote TE a provided
//tciCallConnected has been called
virtual void tciEnqueueCallConnected (const TriPortId *sender, const TriComponentId *receiver,
const TriSignatureId *signature, const TciParameterList *parameterList)=0;

//This operation is called by the CH at the local TE when at a remote TE a provided
//tciReplyConnected has been called
virtual void tciEnqueueReplyConnected (const TriPortId *sender, const TriComponentId *receiver,
const TriSignatureId *signature, const TciParameterList *parameterList,
const TciValue *returnValue)=0;

//This operation is called by the CH at the local TE when at a remote TE a provided
//tciRaiseConnected has been called
virtual void tciEnqueueRaiseConnected (const TriPortId *sender, const TriComponentId *receiver,
const TriSignatureId *signature, const TciValue *exception)=0;

```



```

//This operation is called by the CH at the local TE when at a remote TE a provided
//tciCreateTestComponentReq has been called
virtual const TriComponentId * tciCreateTestComponent (const TciTestComponentKind *kind,
const TciType *componentType, const Tstring *name)=0;

//This operation is called by the CH at the local TE when at a remote TE a provided
//tciStartTestComponentReq has been called
virtual void tciStartTestComponent (const TriComponentId *component,
const TciBehaviourId *behaviour, const TciParameterList *parameterList)=0;

//This operation is called by the CH at the local TE when at a remote TE a provided
//tciStopTestComponentReq has been called
virtual void tciStopTestComponent (const TriComponentId *component)=0;

//This operation is called by the CH at the local TE when at a remote TE a provided tciConnect
//has been called
virtual void tciConnect (const TriPortId *fromPort, const TriPortId *toPort)

//This operation is called by the CH at the local TE when at a remote TE a provided
//tciDisconnect has been called
virtual void tciDisconnect (const TriPortId *fromPort, const TriPortId *toPort)=0;

//This operation is called by the CH at the local TE when at a remote TE a provided tciMapReq
//has been called
virtual void tciMap (const TriPortId *fromPort, const TriPortId *toPort)=0;

//This operation is called by the CH at the local TE when at a remote TE a provided
//tciMapParamReq has been called
virtual void tciMapParam (const TriPortId *fromPort, const TriPortId *toPort,
const TciParameterList *parameterList)=0

//This operation is called by the CH at the local TE when at a remote TE a provided tciUnmapReq
//has been called
virtual void tciUnmap (const TriPortId *fromPort, const TriPortId *toPort)=0;

//This operation is called by the CH at the local TE when at a remote TE a provided
//tciUnmapParamReq has been called
virtual void tciUnmapParam (const TriPortId *fromPort, const TriPortId *toPort,
const TciParameterList *parameterList)=0

//This operation is called by the CH at the local TE when at a remote TE a provided tciUnmapReq
//has been called
virtual void tciTestComponentTerminated (const TriComponentId *component,
const VerdictValue *verdict) const =0

//This operation is called by the CH at the local TE when at a remote TE a provided
//tciTestComponentRunningReq has been called
virtual Tboolean tciTestComponentRunning (const TriComponentId *component) const =0

//This operation is called by the CH at the local TE when at a remote TE a provided
//tciTestComponentDoneReq has been called
virtual Tboolean tciTestComponentDone (const TriComponentId *comp) const =0

//This operation can be called by the CH at the appropriate local TE when at a remote TE a
//provided tciGetMTCReq has been called
virtual const TriComponentId * tciGetMTC () const =0

//This operation is called by the CH at the appropriate local TE when at a remote TE a provided
//tciExecuteTestCaseReq has been called
virtual void tciExecuteTestCase (const TciTestCaseId *testCaseId,
const TriPortIdList *tsiPortList)=0;

//This operation is called by the CH at appropriate local TEs when at a remote TE a provided
//tciResetReq has been called
virtual void tciReset ()=0;

//This operation is called by the CH at the local TE when at a remote TE a provided
//tciKillTestComponentReq has been called
virtual void tciKillTestComponent (const TriComponentId *comp)=0;

//This operation is called by the CH at the local TE when at a remote TE a provided
//tciTestComponentAliveReq has been called
virtual Tboolean tciTestComponentAlive (const TriComponentId *comp) const =0

//This operation is called by the CH at the local TE when at a remote TE a provided
//tciTestComponentKilledReq has been called
virtual Tboolean tciTestComponentKilled (const TriComponentId *comp) const =0

```

### 10.6.3.2 TciChProvided

This class defines the TCI\_CH provided interface. It is mapped to the following interface:

```
//Destructor
virtual ~TciChProvided ();

//Called by the TE when it executes a TTCN-3 unicast send operation on a component port
virtual void tciSendConnected (const TriPortId *sender, const TriComponentId *receiver,
    const TciValue *sendMessage)=0;

//Called by the TE when it executes a TTCN-3 broadcast send operation on a component port
virtual void tciSendConnectedBC (const TriPortId *sender, const TciValue *sendMessage)=0;

//Called by the TE when it executes a TTCN-3 multicast send operation on a component port
virtual void tciSendConnectedMC (const TriPortId *sender, const TriComponentIdList *receivers,
    const TciValue *sendMessage)=0;

//Called by the TE when it executes a TTCN-3 unicast call operation on a component port
virtual void tciCallConnected (const TriPortId *sender, const TriComponentId *receiver,
    const TriSignatureId *signature, const TciParameterList *parameterList)=0;

//Called by the TE when it executes a TTCN-3 broadcast call operation on a component port
virtual void tciCallConnectedBC (const TriPortId *sender, const TriSignatureId *signature,
    const TciParameterList *parameterList)=0;

//Called by the TE when it executes a TTCN-3 multicast call operation on a component port
virtual void tciCallConnectedMC (const TriPortId *sender, const TriComponentIdList *receivers,
    const TriSignatureId *signature, const TciParameterList *parameterList)=0;

//Called by the TE when it executes a TTCN-3 unicast reply operation on a component port
virtual void tciReplyConnected (const TriPortId *sender, const TriComponentId *receiver,
    const TriSignatureId *signature, const TciParameterList *parameterList,
    const TciValue *returnValue)=0;

//Called by the TE when it executes a TTCN-3 broadcast reply operation on a component port
virtual void tciReplyConnectedBC (const TriPortId *sender, const TriSignatureId *signature,
    const TciParameterList *parameterList, const TciValue *returnValue)=0;

//Called by the TE when it executes a TTCN-3 multicast reply operation on a component
virtual void tciReplyConnectedMC (const TriPortId *sender, const TriComponentIdList *receivers,
    const TriSignatureId *signature, const TciParameterList *parameterList,
    const TciValue *returnValue)=0;

//Called by the TE when it executes a TTCN-3 unicast raise operation on a component port
virtual void tciRaiseConnected (const TriPortId *sender, const TriComponentId *receiver,
    const TriSignatureId *signature, const TciValue *exception)=0;

//Called by the TE when it executes a TTCN-3 broadcast raise operation on a component portvirtual
void tciRaiseConnectedBC (const TriPortId *sender, const TriSignatureId *signature,
    const TciValue *exception)=0;

//Called by the TE when it executes a TTCN-3 multicast raise operation on a component
virtual void tciRaiseConnectedMC (const TriPortId *sender, const TriComponentIdList *receiver,
    const TriSignatureId *signature, const TciValue *exception)=0;

//Called from the TE when a component has to be created
virtual const TriComponentId * tciCreateTestComponentReq (const TciTestComponentKind *kind,
    const QualifiedName *componentType, const Tstring &name, const TciValue *hostId)=0;

//Called by the TE when it executes the TTCN-3 start operation
virtual void tciStartTestComponentReq (const TriComponentId *component,
    const TciBehaviourId *behaviour, const TciParameterList *parameterList)=0;

//Called by the TE when it executes the TTCN-3 stop operation
virtual void tciStopTestComponentReq (const TriComponentId *component)=0;

//Called by the TE when it executes a TTCN-3 connect operation
virtual void tciConnectReq (const TriPortId *fromPort, const TriPortId *toPort)=0;

//Called by the TE when it executes a TTCN-3 disconnect operation
virtual void tciDisconnectReq (const TriPortId *fromPort, const TriPortId *toPort)=0;

//Called by the TE when it executes a TTCN-3 map operation
virtual void tciMapReq (const TriPortId *fromPort, const TriPortId *toPort)=0;

//Called by the TE when it executes a TTCN-3 map operation including parameters
virtual void tciMapParamReq (const TriPortId *fromPort, const TriPortId *toPort,
    const TciParameterList *parameterList)=0
```

```

//Called by the TE when it executes a TTCN-3 unmap operation
virtual void tciUnmapReq (const TriPortId *fromPort, const TriPortId *toPort)=0;

//Called by the TE when it executes a TTCN-3 unmap operation including parameters
virtual void tciUnmapParamReq (const TriPortId *fromPort, const TriPortId *toPort,
    const TciParameterList *parameterList)=0

//Called by the TE when a test component terminates execution
virtual void tciTestComponentTerminatedReq (const TriComponentId *component,
    const VerdictValue *verdict)=0;

//Called by the TE when it executes a TTCN-3 running operation
virtual Tboolean tciTestComponentRunningReq (const TriComponentId *component) const =0

//Called by the TE when it executes a TTCN-3 done operation
virtual Tboolean tciTestComponentDoneReq (const TriComponentId *comp) const =0

//Called by the TE when it executes a TTCN-3 mtc operation
virtual const TriComponentId * tciGetMTCReq () const =0

//Called by the TE immediately before it starts the test case behaviour on the MTC
virtual void tciExecuteTestCaseReq (const TciTestCaseId *testCaseId,
    const TriPortIdList *tsiPortList)=0;

//Called by the TE at any time to reset the test system
virtual void tciResetReq ()=0;

//Called by the TE when it executes the TTCN-3 kill operation
virtual void tciKillTestComponentReq (const TriComponentId *comp)=0;

//Called by the TE when it executes the TTCN-3 alive operation
virtual Tboolean tciTestComponentAliveReq (const TriComponentId *comp) const =0

//Called by the TE when it executes the TTCN-3 killed operation
virtual Tboolean tciTestComponentKilledReq (const TriComponentId *comp) const =0

```

## 10.6.4 TCI-TL

### 10.6.4.1 TciTlProvided

This class defines the TCI\_TL provided Tinterface:

```

//Default constructor
TciTlProvided ();

// Destructor
virtual ~TciTlProvided ();

//Called by TE to log the execute test case request
virtual void tliTcExecute (const Tstring &am, const timeval ts, const Tstring src, const Tinteger
line, const TriComponentId *c, const TciTestCaseId *tcId, const TriParameterList *triPars, const
TriTimerDuration *dur)=0;

//Called by TE to log the start of a testcase. This event occurs before the testcase is started
virtual void tliTcStart (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger
line, const TriComponentId *c, const TciTestCaseId *tcId, const TciParameterList *tciPars, const
TriTimerDuration *dur)=0;

//Called by TE to log the stop of a testcase
virtual void tliTcStop (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger
line, const TriComponentId *c, const Tstring &reason)=0;

//Called by TE to log the start of a testcase
virtual void tliTcStarted (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger
line, const TriComponentId *c, const TciTestCaseId *tcId, const TciParameterList *tciPars, const
TriTimerDuration *dur)=0;

//Called by TE to log the termination of a testcase
virtual void tliTcTerminated (const Tstring &am, const timeval ts, const Tstring &src, const
Tinteger line, const TriComponentId *c, const TciTestCaseId *tcId, const TciParameterList *tciPars,
const VerdictValue *verdict, const Tstring &reason)=0;

//Called by TE to log the start of the control part
virtual void tliCtrlStart (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger
line, const TriComponentId *c)=0;

//Called by TE to log the stop of the control part. This event occurs after the control has
//stopped. If the control is not represented by TRI component, c is null

```

```

virtual void tliCtrlStop (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger
line, const TriComponentId *c)=0;

//Called by TE to log the termination of the control part
virtual void tliCtrlTerminated (const Tstring &am, const timeval ts, const Tstring &src, const
Tinteger line, const TriComponentId *c)=0;

//Called by TE to log a unicast send operation
virtual void tliMSend_m (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger
line, const TriComponentId *c, const TriPortId *at, const TriPortId *to, const TciValue *msgValue,
const TriAddress *address, const TciStatus *encoderFailure, const TriMessage *msg, const TriStatus
*transmissionFailure)=0;

//Called by TE to log a broadcast send operation
virtual void tliMSend_m_BC (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger
line, const TriComponentId *c, const TriPortId *at, const TriPortId *to, const TciValue *msgValue,
const TciStatus *encoderFailure, const TriMessage *msg, const TriStatus *transmissionFailure)=0;

//Called by TE to log a multicast send operation
virtual void tliMSend_m_MC (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger
line, const TriComponentId *c, const TriPortId *at, const TriPortId *to, const TciValue *msgValue,
const TriAddressList *addresses, const TciStatus *encoderFailure, const TriMessage *msg, const
TriStatus *transmissionFailure)=0;

//Called by TE to log a unicast send operation
virtual void tliMSend_c (const Tstring &am, const timeval ts, const Tstring src, const Tinteger
line, const TriComponentId *c, const TriPortId *at, const TriPortId *to, const TciValue *msgValue,
const TriStatus *transmissionFailure)=0;

//Called by TE to log a broadcast send operation
virtual void tliMSend_c_BC (const Tstring &am, const timeval ts, const Tstring src, const Tinteger
line, const TriComponentId *c, const TriPortId *at, const TriPortIdList *to, const TciValue
*msgValue, const TriStatus *transmissionFailure)=0;

//Called by TE to log a multicast send operation
virtual void tliMSend_c_MC (const Tstring &am, const timeval ts, const Tstring src, const Tinteger
line, const TriComponentId *c, const TriPortId *at, const TriPortIdList *to, const TciValue
*msgValue, const TriStatus *transmissionFailure)=0;

//Called by TE to log the enqueueing of a message
virtual void tliMDetected_m (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger
line, const TriComponentId *c, const TriPortId *at, const TriPortId *from, const TriMessage *msg,
const TriAddress *address)=0;

//Called by CH to log the enqueueing of a message
virtual void tliMDetected_c (const Tstring &am, const timeval ts, const Tstring src, const Tinteger
line, const TriComponentId *c, const TriPortId *at, const TriPortId *from, const TciValue
*msgValue)=0;

//Called by TE to log the mismatch of a template
virtual void tliMMismatch_m (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger
line, const TriComponentId *c, const TriPortId *at, const TciValue *msgValue, const TciValueTemplate
*msgTpl, const TciValueDifferenceList *diffs, const TriAddress *address, const TciValueTemplate
*addressTpl)=0;

//Called by TE to log the mismatch of a template
virtual void tliMMismatch_c (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger
line, const TriComponentId *c, const TriPortId *at, const TciValue *msgValue, const TciValueTemplate
*msgTpl, const TciValueDifferenceList *diffs, const TriComponentId *from, const TciNonValueTemplate
*fromTpl)=0;

// Called by TE to log the receiving of a message
virtual void tliMRecieve_m (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger
line, const TriComponentId *c, const TriPortId *at, const TciValue *msgValue, const TciValueTemplate
*msgTpl, const TriAddress *address, const TciValueTemplate *addressTpl)=0;

//Called by TE to log the mismatch of a template
virtual void tliMReceive_c (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger
line, const TriComponentId *c, const TriPortId *at, const TciValue *msgValue, const TciValueTemplate
*msgTpl, const TriComponentId *fromComp, const TciNonValueTemplate *fromTpl)=0;

//Called by TE to log a unicast call operation
virtual void tliPrCall_m (const Tstring &am, const timeval ts, const Tstring src, const Tinteger
line, const TriComponentId *c, const TriPortId *at, const TriPortId *to, const TriSignatureId
*signature, const TciParameterList *tciPars, const TriAddress *address, const TciStatus
*encoderFailure, const TriParameterList *triPars, const TriStatus *transmissionFailure)=0;

//Called by TE to log a broadcast call operation
virtual void tliPrCall_m_BC (const Tstring &am, const timeval ts, const Tstring src, const Tinteger
line, const TriComponentId *c, const TriPortId *at, const TriPortId *to, const TriSignatureId

```

```

*signature, const TciParameterList *tciPars, const TciStatus *encoderFailure, const TriParameterList
*triPars, const TriStatus *transmissionFailure)=0;

//Called by TE to log a multicast call operation
virtual void tliPrCall_m_MC (const Tstring &am, const timeval ts, const Tstring src, const Tinteger
line, const TriComponentId *c, const TriPortId *at, const TriPortId *to, const TriSignatureId
*signature, const TciParameterList *tciPars, const TriAddressList *addresses, const TciStatus
*encoderFailure, const TriParameterList *triPars, const TriStatus *transmissionFailure)=0;

//Called by TE to log a unicast call operation
virtual void tliPrCall_c (const Tstring &am, const timeval ts, const Tstring src, const Tinteger
line, const TriComponentId *c, const TriPortId *at, const TriPortId *to, const TriSignatureId
*signature, const TciParameterList *tciPars, const TriStatus *transmissionFailure)=0;

//Called by TE to log a broadcast call operation
virtual void tliPrCall_c_BC (const Tstring &am, const timeval ts, const Tstring src, const Tinteger
line, const TriComponentId *c, const TriPortId *at, const TriPortIdList *to, const TriSignatureId
*signature, const TciParameterList *tciPars, const TriStatus *transmissionFailure)=0;

//Called by TE to log a multicast call operation
virtual void tliPrCall_c_MC (const Tstring &am, const timeval ts, const Tstring src, const Tinteger
line, const TriComponentId *c, const TriPortId *at, const TriPortIdList *to, const TriSignatureId
*signature, const TciParameterList *tciPars, const TriStatus *transmissionFailure)=0;

//Called by TE to log the getcall enqueue operation
virtual void tliPrGetCallDetected_m (const Tstring &am, const timeval ts, const Tstring src, const
Tinteger line, const TriComponentId *c, const TriPortId *at, const TriPortId *from, const
TriSignatureId *signature, const TriParameterList *triPars, const TriAddress *address)=0;

//Called by TE to log the getcall enqueue operation
virtual void tliPrGetCallDetected_c (const Tstring &am, const timeval ts, const Tstring src, const
Tinteger line, const TriComponentId *c, const TriPortId *at, const TriPortId *from, const
TriSignatureId *signature, const TciParameterList *tciPars)=0;

//Called by TE to log the mismatch of a getcall
virtual void tliPrGetCallMismatch_m (const Tstring &am, const timeval ts, const Tstring src, const
Tinteger line, const TriComponentId *c, const TriPortId *at, const TriSignatureId *signature, const
TciParameterList *tciPars, const TciValueTemplate *parsTpl, const TciValueDifferenceList *diffs,
const TriAddress *address, const TciValueTemplate *addressTpl)=0;

//Called by TE to log the mismatch of a getcall
virtual void tliPrGetCallMismatch_c (const Tstring &am, const timeval ts, const Tstring src, const
Tinteger line, const TriComponentId *c, const TriPortId *at, const TriSignatureId *signature, const
TciParameterList *tciPars, const TciValueTemplate *parsTpl, const TciValueDifferenceList *diffs,
const TriComponentId *from, const TciValueTemplate *fromTpl)=0;

//Called by TE to log getting a call
virtual void tliPrGetCall_m (const Tstring &am, const timeval ts, const Tstring src, const Tinteger
line, const TriComponentId *c, const TriPortId *at, const TriSignatureId *signature, const
TciParameterList *tciPars, const TciValueTemplate *parsTpl, const TriAddress *address, const
TciValueTemplate *addressTpl)=0;

//Called by TE to log getting a call
virtual void tliPrGetCall_c (const Tstring &am, const timeval ts, const Tstring src, const Tinteger
line, const TriComponentId *c, const TriPortId *at, const TriSignatureId *signature, const
TciParameterList *tciPars, const TciValueTemplate *parsTpl, const TriComponentId *from, const
TciNonValueTemplate *fromTpl)=0;

//Called by TE to log a unicast reply operation
virtual void tliPrReply_m (const Tstring &am, const timeval ts, const Tstring src, const Tinteger
line, const TriComponentId *c, const TriPortId *at, const TriPortId *to, const TriSignatureId
*signature, const TciParameterList *tciPars, const TciValue *replValue, const TriAddress *address,
const TciStatus *encoderFailure, const TriParameterList *triPars, const TriParameter *repl, const
TriStatus *transmissionFailure)=0;

//Called by TE to log a broadcast reply operation
virtual void tliPrReply_m_BC (const Tstring &am, const timeval ts, const Tstring src, const Tinteger
line, const TriComponentId *c, const TriPortId *at, const TriPortId *to, const TriSignatureId
*signature, const TciParameterList *tciPars, const TciValue *replValue, const TciStatus
*encoderFailure, const TriParameterList *triPars, const TriParameter *repl, const TriStatus
*transmissionFailure)=0;

//Called by TE to log a multicast reply operation
virtual void tliPrReply_m_MC (const Tstring &am, const timeval ts, const Tstring src, const Tinteger
line, const TriComponentId *c, const TriPortId *at, const TriPortId *to, const TriSignatureId
*signature, const TciParameterList *tciPars, const TciValue *replValue, const TriAddressList
*addresses, const TciStatus *encoderFailure, const TriParameterList *triPars, const TriParameter
*repl, const TriStatus *transmissionFailure)=0;

//Called by TE to log a unicast reply operation

```

```

virtual void tliPrReply_c (const Tstring &am, const timeval ts, const Tstring src, const Tinteger
line, const TriComponentId *c, const TriPortId *at, const TriPortId *to, const TriSignatureId
*signature, const TciValue *parsValue, const TciValue *replValue, const TriStatus
*transmissionFailure)=0;

//Called by TE to log a broadcast reply operation
virtual void tliPrReply_c_BC (const Tstring &am, const timeval ts, const Tstring src, const Tinteger
line, const TriComponentId *c, const TriPortId *at, const TriPortIdList *to, const TriSignatureId
*signature, const TciValue *parsValue, const TciValue *replValue, const TriStatus
*transmissionFailure)=0;

//Called by TE to log og a multicast reply operation
virtual void tliPrReply_c_MC (const Tstring &am, const timeval ts, const Tstring src, const Tinteger
line, const TriComponentId *c, const TriPortId *at, const TriPortIdList *to, const TriSignatureId
*signature, const TciValue *parsValue, const TciValue *replValue, const TriStatus
*transmissionFailure)=0;

//Called by TE to log the getreply enqueue operation
virtual void tliPrGetReplyDetected_m (const Tstring &am, const timeval ts, const Tstring src, const
Tinteger line, const TriComponentId *c, const TriPortId *at, const TriPortId *from, const
TriSignatureId *signature, const TriParameterList *triPars, const TriParameter *repl, const
TriAddress *address)=0;

//Called by CH to log the getreply enqueue operation
virtual void tliPrGetReplyDetected_c (const Tstring &am, const timeval ts, const Tstring src, const
Tinteger line, const TriComponentId *c, const TriPortId *at, const TriPortId *from, const
TriSignatureId *signature, const TriParameterList *tciPars, const TciValue *replValue)=0;

//Called by TE to log the mismatch of a getreply operation
virtual void tliPrGetReplyMismatch_m (const Tstring &am, const timeval ts, const Tstring src, const
Tinteger line, const TriComponentId *c, const TriPortId *at, const TriSignatureId *signature, const
TciParameterList *tciPars, const TciValueTemplate *parsTmpl, const TciValue *replValue, const
TciValueTemplate *replyTmpl, const TciValueDifferenceList *diffs, const TriAddress *address, const
TciValueTemplate *addressTmpl)=0;

//Called by TE to log the mismatch of a getreply operation
virtual void tliPrGetReplyMismatch_c (const Tstring &am, const timeval ts, const Tstring src, const
Tinteger line, const TriComponentId *c, const TriPortId *at, const TriSignatureId *signature, const
TciParameterList *tciPars, const TciValueTemplate *parsTmpl, const TciValue *replValue, const
TciValueTemplate *replyTmpl, const TciValueDifferenceList *diffs, const TriComponentId *from, const
TciNonValueTemplate *fromTmpl)=0;

//Called by TE to log getting a reply
virtual void tliPrGetReply_m (const Tstring &am, const timeval ts, const Tstring src, const Tinteger
line, const TriComponentId *c, const TriPortId *at, const TriSignatureId *signature, const
TciParameterList *tciPars, const TciValueTemplate *parsTmpl, const TciValue *replValue, const
TciValueTemplate *replyTmpl, const TriAddress *address, const TciValueTemplate *addressTmpl)=0;

//Called by TE to log getting a reply
virtual void tliPrGetReply_c (const Tstring &am, const timeval ts, const Tstring src, const Tinteger
line, const TriComponentId *c, const TriPortId *at, const TriSignatureId *signature, const
TciParameterList *tciPars, const TciValueTemplate *parsTmpl, const TciValue *replValue, const
TciValueTemplate *replyTmpl, const TriComponentId *from, const TciNonValueTemplate *fromTmpl)=0;

//Called by TE to log a unicast raise operation
virtual void tliPrRaise_m (const Tstring &am, const timeval ts, const Tstring src, const Tinteger
line, const TriComponentId *c, const TriPortId *at, const TriPortId *to, const TriSignatureId
*signature, const TciParameterList *tciPars, const TciValue *excValue, const TriAddress *address,
const TriStatus *encoderFailure, const TriException *exc, const TriStatus *transmissionFailure)=0;

//Called by TE to log a broadcast raise operation
virtual void tliPrRaise_m_BC (const Tstring &am, const timeval ts, const Tstring src, const Tinteger
line, const TriComponentId *c, const TriPortId *at, const TriPortId *to, const TriSignatureId
*signature, const TciParameterList *tciPars, const TciValue *excValue, const TriStatus
*encoderFailure, const TriException *exc, const TriStatus *transmissionFailure)=0;

//Called by TE to log a multicast raise operation
virtual void tliPrRaise_m_MC (const Tstring &am, const timeval ts, const Tstring src, const Tinteger
line, const TriComponentId *c, const TriPortId *at, const TriPortId *to, const TriSignatureId
*signature, const TciParameterList *tciPars, const TciValue *excValue, const TriAddressList
*addresses, const TriStatus *encoderFailure, const TriException *exc, const TriStatus
*transmissionFailure)=0;
//Called by TE to log a unicast raise operation
virtual void tliPrRaise_c (const Tstring &am, const timeval ts, const Tstring src, const Tinteger
line, const TriComponentId *c, const TriPortId *at, const TriPortId *to, const TriSignatureId
*signature, const TciParameterList *tciPars, const TciValue *excValue, const TriStatus
*transmissionFailure)=0;

//Called by TE to log a broadcast raise operation

```

```

virtual void tliPrRaise_c_BC (const Tstring &am, const timeval ts, const Tstring src, const Tinteger
line, const TriComponentId *c, const TriPortId *at, const TriPortIdList *to, const TriSignatureId
*signature, const TciParameterList *tciPars, const TciValue *excValue, const TriStatus
*transmissionFailure)=0;

//Called by TE to log a multicast raise operation
virtual void tliPrRaise_c_MC (const Tstring &am, const timeval ts, const Tstring src, const Tinteger
line, const TriComponentId *c, const TriPortId *at, const TriPortIdList *to, const TriSignatureId
*signature, const TciParameterList *tciPars, const TciValue *excValue, const TriStatus
*transmissionFailure)=0;

//Called by TE to log the catch enqueue operation
virtual void tliPrCatchDetected_m (const Tstring &am, const timeval ts, const Tstring src, const
Tinteger line, const TriComponentId *c, const TriPortId *at, const TriPortId *from, const
TriSignatureId *signature, const TriException *exc, const TriAddress *address)=0;

//Called by TE to log the catch enqueue operation
virtual void tliPrCatchDetected_c (const Tstring &am, const timeval ts, const Tstring src, const
Tinteger line, const TriComponentId *c, const TriPortId *at, const TriPortId *from, const
TriSignatureId *signature, const TciValue *excValue)=0;

//Called by TE to log the mismatch of a catch operation
virtual void tliPrCatchMismatch_m (const Tstring &am, const timeval ts, const Tstring src, const
Tinteger line, const TriComponentId *c, const TriPortId *at, const TriSignatureId *signature, const
TciValue *excValue, const TciValueTemplate *excTmpl, const TciValueDifferenceList *diffs, const
TriAddress *address, const TciValueTemplate *addressTmpl)=0;

//Called by TE to log the mismatch of a catch operation
virtual void tliPrCatchMismatch_c (const Tstring &am, const timeval ts, const Tstring src, const
Tinteger line, const TriComponentId *c, const TriPortId *at, const TriSignatureId *signature, const
TciValue *excValue, const TciValueTemplate *excTmpl, const TciValueDifferenceList *diffs, const
TriComponentId *from, const TciNonValueTemplate *fromTmpl)=0;

//Called by TE to log catching an exception
virtual void tliPrCatch_m (const Tstring &am, const timeval ts, const Tstring src, const Tinteger
line, const TriComponentId *c, const TriPortId *at, const TriSignatureId *signature, const TciValue
*excValue, const TciValueTemplate *excTmpl, const TriAddress *address, const TciValueTemplate
*addressTmpl)=0;

//Called by TE to log catching an exception
virtual void tliPrCatch_c (const Tstring &am, const timeval ts, const Tstring src, const Tinteger
line, const TriComponentId *c, const TriPortId *at, const TriSignatureId *signature, const TciValue
*excValue, const TciValueTemplate *excTmpl, const TriComponentId *from, const TciNonValueTemplate
*fromTmpl)=0;

//Called by TE to log the detection of a catch timeout
virtual void tliPrCatchTimeoutDetected (const Tstring &am, const timeval ts, const Tstring src,
const Tinteger line, const TriComponentId *c, const TriPortId *at, const TriSignatureId
*signature)=0;

//Called by TE to log catching a timeout
virtual void tliPrCatchTimeout (const Tstring &am, const timeval ts, const Tstring src, const
Tinteger line, const TriComponentId *c, const TriPortId *at, const TriSignatureId *signature)=0;

//Called by TE to log the create component operation
virtual void tliCCreate (const Tstring &am, const timeval ts, const Tstring src, const Tinteger
line, const TriComponentId *c, const TriComponentId *comp, const Tstring &name, const Tboolean
alive)=0;

//Called by TE to log the start component operation
virtual void tliCStart (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger
line, const TriComponentId *c, const TriComponentId *comp, const TciBehaviourId *beh, const
TciParameterList *tciPars)=0;

//Called by TE to log the running component operation
virtual void tliCRunning (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger
line, const TriComponentId *c, const TriComponentId *comp, const ComponentStatus status)=0;

//Called by TE to log the alive component operation
virtual void tliCALive (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger
line, const TriComponentId *c, const TriComponentId *comp, const ComponentStatus status)=0;

//Called by TE to log the stop component operation
virtual void tliCStop (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line,
const TriComponentId *c, const TriComponentId *comp)=0;

//Called by TE to log the kill component operation
virtual void tliCKill (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line,
const TriComponentId *c, const TriComponentId *comp)=0;

```

```

//Called by TE to log the mismatch of a done component operation
virtual void tliCDoneMismatch (const Tstring &am, const timeval ts, const Tstring &src, const
Tinteger line, const TriComponentId *c, const TriComponentId *comp, const TciNonValueTemplate
*compTpl)=0;

//Called by TE to log the done component operation
virtual void tliCDone (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line,
const TriComponentId *c, const TciNonValueTemplate *compTpl)=0;

//Called by TE to log the mismatch of a killed component operation
virtual void tliCKilledMismatch (const Tstring &am, const timeval ts, const Tstring &src, const
Tinteger line, const TriComponentId *c, const TciNonValueTemplate *compTpl)=0;

//Called by TE to log the killed component operation
virtual void tliCKilled (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger
line, const TriComponentId *c, const TciNonValueTemplate *compTpl)=0;

//Called by TE to log the termination of a component
virtual void tliCTerminated (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger
line, const TriComponentId *c, const VerdictValue *verdict, const Tstring &reason)=0;

//Called by TE to log the connect operation
virtual void tliPConnect (const Tstring &am, const timeval ts, const Tstring src, const Tinteger
line, const TriComponentId *c, const TriPortId *port1, const TriPortId *port2)=0;

//Called by TE to log the connect operation
virtual void tliPDisconnect (const Tstring &am, const timeval ts, const Tstring src, const Tinteger
line, const TriComponentId *c, const TriPortId *port1, const TriPortId *port2)=0;

//Called by TE to log the map operation
virtual void tliPMap (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line,
const TriComponentId *c, const TriPortId *port1, const TriPortId *port2)=0;

//Called by TE to log the map operation including param
virtual void tliPMapParam (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger
line, const TriComponentId *c, const TriPortId *port1, const TriPortId *port2, const
TciParameterList *tciPars, const TriStatus *encoderFailure,
const TriParameterList *triPars)=0

//Called by TE to log the unmap operation
virtual void tliPUnmap (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger
line, const TriComponentId *c, const TriPortId *port1, const TriPortId *port2)=0;

//Called by TE to log the unmap operation including param
virtual void tliPUnmapParam (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger
line, const TriComponentId *c, const TriPortId *port1, const TriPortId *port2, const
TciParameterList *tciPars, const TriStatus *encoderFailure,
const TriParameterList *triPars)=0

//Called by TE to log the port clear operation
virtual void tliPClear (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger
line, const TriComponentId *c, const TriPortId *port)=0;

//Called by TE to log the port start operation
virtual void tliPStart (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger
line, const TriComponentId *c, const TriPortId *port)=0;

//Called by TE to log the port stop operation
virtual void tliPStop (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line,
const TriComponentId *c, const TriPortId *port)=0;

//Called by TE to log the port stop operation
virtual void tliPHalt (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line,
const TriComponentId *c, const TriPortId *port)=0;

//Called by TE to log the encode operation
virtual void tliEncode (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger
line, const TriComponentId *c, const TciValue *val, const TciStatus *encoderFailure, const
TriMessage *msg, const Tstring &codec)=0;

//Called by TE to log the decode operation
virtual void tliDecode (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger
line, const TriComponentId *c, const TriMessage *msg, const TciStatus *decoderFailure, const
TciValue *val, const Tstring &codec)=0;

//Called by TE to log the detection of a timeout
virtual void tliTTimeoutDetected (const Tstring &am, const timeval ts, const Tstring &src, const
Tinteger line, const TriComponentId *c, const TriTimerId *timer)=0;

//Called by TE to log a timeout mismatch

```



```

virtual void tliTimeoutMismatch (const Tstring &am, const timeval ts, const Tstring &src, const
Tinteger line, const TriComponentId *c, const TriTimerId *timer, const TciNonValueTemplate
*timerTpl)=0;

//Called by TE to log a timeout match
virtual void tliTimeout (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger
line, const TriComponentId *c, const TriTimerId *timer, const TciNonValueTemplate *timerTpl)=0;

//Called by TE to log the start of a timer
virtual void tliTStart (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger
line, const TriComponentId *c, const TriTimerId *timer, const TriTimerDuration *dur)=0;

//Called by TE to log the stop of a timer
virtual void tliTStop (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line,
const TriComponentId *c, const TriTimerId *timer, const TriTimerDuration *dur)=0;

//Called by TE to log the reading of a timer
virtual void tliTRead (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line,
const TriComponentId *c, const TriTimerId *timer, const TriTimerDuration *elapsed)=0;

//Called by TE to log the running timer operation
virtual void tliTRunning (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger
line, const TriComponentId *c, const TriTimerId *timer, const TimerStatus status)=0;

//Called by TE to log the entering of a scope
virtual void tliSEnter (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger
line, const TriComponentId *c, const QualifiedName &name, const TciParameterList *tciPars, const
Tstring &kind)=0;

//Called by TE to log the leaving of a scope
virtual void tliSLeave (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger
line, const TriComponentId *c, const QualifiedName &name, const TciParameterList *tciPars, const
TciValue *returnValue, const Tstring &kind)=0;

//Called by TE to log the modification of the value of a variable
virtual void tliVVar (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line,
const TriComponentId *c, const QualifiedName &name, const TciValue *varValue)=0;

//Called by TE to log the value of a module parameter
virtual void tliModulePar (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger
line, const TriComponentId *c, const QualifiedName &name, const TciValue *parValue)=0;

//Called by TE to log the value of a module parameter
virtual void tliGetVerdict (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger
line, const TriComponentId *c, const VerdictValue *verdict)=0;

//Called by TE to log the setverdict operation
virtual void tliSetVerdict (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger
line, const TriComponentId *c, const VerdictValue *verdict, const Tstring &reason)=0;

//Called by TE to log the TTCN-3 statement log
virtual void tliLog (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line,
const TriComponentId *c, const Tstring *log)=0;

//Called by TE to log entering an alt
virtual void tliAEnter (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger
line, const TriComponentId *c)=0;

//Called by TE to log leaving an alt
virtual void tliALeave (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger
line, const TriComponentId *c)=0;

//Called by TE to log the nomatch of an alt
virtual void tliANomatch (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger
line, const TriComponentId *c)=0;

//Called by TE to log repeating an alt
virtual void tliARepeat (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger
line, const TriComponentId *c)=0;

//Called by TE to log entering the default section
virtual void tliADefaults (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger
line, const TriComponentId *c)=0;

//Called by TE to log the activation of a default
virtual void tliAActivate (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger
line, const TriComponentId *c, const QualifiedName &name, const TciParameterList *tciPars, const
TciValue *ref)=0;

//Called by TE to log the deactivation of a default

```

```

virtual void tliADeactivate (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger
line, const TriComponentId *c, const TciValue *ref)=0;

//Called by TE to log entering an alt
virtual void tliAwait (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line,
const TriComponentId *c)=0;

//Called by TE to log that the component executed an SUT action
virtual void tliAction (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger
line, const TriComponentId *c, const Tstring &action)=0;

//Called by TE to log that the component successfully executed a match operation
virtual void tliMatch (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line,
const TriComponentId *c, const TciValue &expr, const TciValueTemplate &tmpl)=0;

//Called by TE to log that the component executed a match operation, and a mismatch occurred
virtual void tliMatchMismatch (const Tstring &am, const timeval ts, const Tstring &src, const
Tinteger line, const TriComponentId *c, const TciValue &expr, const TciValueTemplate &tmpl, const
TciValueDifferenceList &diffs)=0;

//Can be called by the TE to log additional information during test execution
virtual void tliInfo (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line,
const TriComponentId *c, const Tinteger level, const Tstring &info)=0;

```

## 11 W3C XML mapping

### 11.1 Introduction

This clause introduces the TCI XML mapping [10], [11] and [12] for the logging interface of TCI. The XML mapping for the logging interface defines how the IDL definitions for TCI-TL described in clause 7 are mapped to XML. The complete schema definitions for this mapping are given in Annex B.

### 11.2 Scopes

The IDL module `tciInterface` is mapped to an XML schema with the name space [http://uri.etsi.org/ttcn-3/tci/TLI\\_v4\\_4\\_1.xsd](http://uri.etsi.org/ttcn-3/tci/TLI_v4_4_1.xsd).

This schema uses further schemas:

- [http://uri.etsi.org/ttcn-3/tci/SimpleTypes\\_v4\\_4\\_1.xsd](http://uri.etsi.org/ttcn-3/tci/SimpleTypes_v4_4_1.xsd) for the mapping of simple types to XML.
- [http://uri.etsi.org/ttcn-3/tci/Types\\_v4\\_4\\_1.xsd](http://uri.etsi.org/ttcn-3/tci/Types_v4_4_1.xsd) for the mapping of structured types to XML.
- [http://uri.etsi.org/ttcn-3/tci/Values\\_v4\\_4\\_1.xsd](http://uri.etsi.org/ttcn-3/tci/Values_v4_4_1.xsd) for the mapping of values to XML.
- [http://uri.etsi.org/ttcn-3/tci/Templates\\_v4\\_4\\_1.xsd](http://uri.etsi.org/ttcn-3/tci/Templates_v4_4_1.xsd) for the mapping of templates to XML.
- [http://uri.etsi.org/ttcn-3/tci/Events\\_v4\\_4\\_1.xsd](http://uri.etsi.org/ttcn-3/tci/Events_v4_4_1.xsd) for the mapping of logging events to XML.

### 11.3 Type mapping

#### 11.3.1 Mapping of simple types

##### 11.3.1.1 TBoolean

The IDL `TBoolean` type is mapped to the xsd basic type `boolean`.

##### 11.3.1.2 TString

The IDL `TString` type is mapped to the xsd basic type `string`.

### 11.3.1.3 TInteger

The IDL `TInteger` type is mapped to the xsd basic type `integer`.

### 11.3.1.4 TriTimerDurationType

The IDL `TriTimerDurationType` type is mapped to the xsd basic type `float`.

### 11.3.1.5 TciParameterPassingModeType

The IDL `TciParameterPassingModeType` type is mapped to the xsd basic type `string` with enumeration values "in", "out" and "inout".

### 11.3.1.6 TriStatusType

The IDL `TriStatusType` type is mapped to the xsd basic type `string` with enumeration values "TRI\_Ok" and "TRI\_Error".

### 11.3.1.7 TciStatusType

The IDL `TriStatusType` type is mapped to the xsd basic type `string` with enumeration values "TCI\_Ok" and "TCI\_Error".

### 11.3.1.8 ComponentStatusType

The IDL `ComponentStatusType` type is mapped to the xsd basic type `string` with enumeration values "inactiveC", "runningC", "stoppedC", "killedC" and "nullC".

### 11.3.1.9 TimerStatusType

The IDL `TimerStatusType` type is mapped to the xsd basic type `string` with enumeration values "runningT", "inactiveT", "expiredT", and "nullT".

### 11.3.1.10 PortStatusType

The IDL `PortStatusType` type is mapped to the xsd basic type `string` with enumeration values "startedP", "haltedP" and "stoppedP".

## 11.3.2 Complex type mapping

### 11.3.2.1 TriPortIdType

`TriPortIdType` is mapped to the following complex type:

```
<xsd:complexType name="TriPortIdType">
  <xsd:sequence>
    <xsd:element name="comp" type="Types:TriComponentIdType" />
    <xsd:element name="port" type="Types:Port"/>
  </xsd:sequence>
</xsd:complexType>
```

#### Elements:

- `comp` The TRI component identifier.
- `port` The identification of the port.

#### Attributes:

- none.

### 11.3.2.2 TriComponentIdType

`TriComponentIdType` is mapped to the following complex type:

```
<xsd:complexType name="TriComponentIdType">
  <xsd:sequence>
    <xsd:choice>
      <xsd:element name="null" type="Templates:null"/>
    </xsd:choice>
  </xsd:sequence>
</xsd:complexType>
```

```

        <xsd:element name="id" type="Types:Id"/>
    </xsd:choice>
</xsd:sequence>
</xsd:complexType>

```

#### Elements:

- `id` The identifier of the TRI component.
- `null` The `null` identifier. To be used if there is no TRI component identifier.

#### Attributes:

- `none`.

### 11.3.2.3 TriComponentIdListType

`TriComponentIdListType` is mapped to the following complex type:

```

<xsd:complexType name="TriComponentIdListType">
  <xsd:sequence>
    <xsd:element name="comp" type="Types:TriComponentIdType" minOccurs="0"
      maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>

```

#### Elements:

- `comp` The identifiers of TRI components in that list.

#### Attributes:

- `none`.

### 11.3.2.4 Port

`Port` is mapped to the following complex type:

```

<xsd:complexType name="Port">
  <xsd:sequence>
    <xsd:element name="id" type="Types:Id"/>
    <xsd:element name="index" type="xsd:int"/>
  </xsd:sequence>
</xsd:complexType>

```

#### Elements:

- `id` The port identifier.
- `port` The port index.

#### Attributes:

- `none`.

### 11.3.2.5 Id

`Id` is used as identification for components, ports and timers and is mapped to the following complex type:

```

<xsd:complexType name="Id">
  <xsd:sequence>
    <xsd:element name="name" type="SimpleTypes:TString"/>
    <xsd:element name="id" type="SimpleTypes:TString" minOccurs="0"/>
    <xsd:element name="type" type="SimpleTypes:TString" minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>

```

#### Elements:

- `name` The name of the component, port or timer.
- `id` The internal representation of the component, port or timer.
- `type` The type of the component, port or timer.

## Attributes:

- none.

### 11.3.2.6 TriMessageType

**TriMessageType** is mapped to the following complex type:

```
<xsd:complexType name="TriMessageType">
  <xsd:attribute name="val" type="xsd:hexBinary"/>
  <xsd:attribute name="paddingBits" type="xsd:integer" use="optional" default="0"/>
</xsd:complexType>
```

NOTE – paddingBits is optional with a default value of 0 and should only take values between 0 and 7. The relation between paddingBits and numberOfBits is: numberOfBits == (length(val-attribute)/2)\*8-paddingBits. In the byte-aligned case which is the typical one, the paddingBits attribute can be left out.

## Elements:

- val The encoded message.

## Attributes:

- none.

### 11.3.2.7 TriParameterType

**TriParameterType** is mapped to the following complex type:

```
<xsd:complexType name="TriParameterType">
  <xsd:element name="val" type="xsd:hexBinary"/>
  <xsd:attribute name="paddingBits" type="xsd:integer" use="optional" default="0"/>
  <xsd:attribute name="name" type="SimpleTypes:TString"/>
  <xsd:attribute name="mode" type="SimpleTypes:TciParameterPassingModeType"/>
</xsd:complexType>
```

NOTE – paddingBits is optional with a default value of 0 and should only take values between 0 and 7. The relation between paddingBits and numberOfBits is: numberOfBits == (length(val-attribute)/2)\*8-paddingBits. In the byte-aligned case which is the typical one, the paddingBits attribute can be left out.

## Elements:

- val The encoded parameter.

## Attributes:

- name The parameter name.
- mode The parameter passing mode.

### 11.3.2.8 TriParameterListType

**TriParameterListType** is mapped to the following complex type:

```
<xsd:complexType name="TriParameterListType">
  <xsd:sequence>
    <xsd:element name="par" type="Types:TriParameterType" minOccurs="0"
      maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>
```

## Sequence of Elements:

- par The parameters in that list.

## Attributes:

- none.

### 11.3.2.9 TriAddressType

**TriAddressType** is mapped to the following complex type:

```
<xsd:complexType name="TriAddressType" >
  <xsd:attribute name="val" type="xsd:hexBinary"/>
  <xsd:attribute name="paddingBits" type="xsd:integer" use="optional" default="0"/>
</xsd:complexType>
```

NOTE – paddingBits is optional with a default value of 0 and should only take values between 0 and 7. The relation between paddingBits and numberOfBits is: numberOfBits == (length(val-attribute)/2) \* 8 - paddingBits. In the byte-aligned case which is the typical one, the paddingBits attribute can be left out.

#### Elements:

- val The address value.

#### Attributes:

- none.

### 11.3.2.10 TriAddressListType

**TriAddressListType** is mapped to the following complex type:

```
<xsd:complexType name="TriAddressListType" >
  <xsd:sequence>
    <xsd:element name="addr" type="Types:TriAddressType" minOccurs="0"
      maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>
```

#### Elements:

- addr The addresses in that list.

#### Attributes:

- none.

### 11.3.2.11 TriExceptionType

**TriExceptionType** is mapped to the following complex type:

```
<xsd:complexType name="TriExceptionType" >
  <xsd:attribute name="val" type="xsd:hexBinary"/>
  <xsd:attribute name="paddingBits" type="xsd:integer" use="optional" default="0"/>
</xsd:complexType>
```

NOTE – paddingBits is optional with a default value of 0 and should only take values between 0 and 7. The relation between paddingBits and numberOfBits is: numberOfBits == (length(val-attribute)/2) \* 8 - paddingBits. In the byte-aligned case which is the typical one, the paddingBits attribute can be left out.

#### Elements:

- val The exception.

#### Attributes:

- none.

### 11.3.2.12 TriSignatureIdType

**TriSignatureIdType** is mapped to the following complex type:

```
<xsd:complexType name="TriSignatureIdType" >
  <xsd:attribute name="val" type="SimpleTypes:TString" use="required"/>
</xsd:complexType>
```

**Elements:**

- `val` The signature.

**Attributes:**

- none.

### 11.3.2.13 TriTimerIdType

**TriTimerIdType** is mapped to the following complex type:

```
<xsd:complexType name="TriTimerIdType">
  <xsd:sequence>
    <xsd:element name="id" type="Types:Id"/>
  </xsd:sequence>
</xsd:complexType>
```

**Elements:**

- `id` The identification of the timer.

**Attributes:**

- none.

### 11.3.2.14 TriTimerDurationType

**TriTimerDurationType** is mapped to the following simple type:

```
<xsd:simpleType name="TriTimerDurationType">
  <xsd:restriction base="xsd:float"/>
</xsd:simpleType>
```

### 11.3.2.15 QualifiedName

**QualifiedName** is used to fully qualify module parameters, variables, etc., and is mapped to the following complex type:

```
<xsd:complexType name="QualifiedName">
  <xsd:attribute name="moduleName" type="SimpleTypes:TString" use="required"/>
  <xsd:attribute name="baseName" type="SimpleTypes:TString" use="required"/>
</xsd:complexType>
```

**Elements:**

- `moduleName` The module name of the TTCN-3 module.
- `baseName` The name of the object that is fully qualified.

**Attributes:**

- none.

### 11.3.2.16 TciBehaviourIdType

**TciBehaviourIdType** is mapped to the following complex type:

```
<xsd:complexType name="TciBehaviourIdType">
  <xsd:sequence>
    <xsd:element name="name" type="Types:QualifiedName"/>
  </xsd:sequence>
</xsd:complexType>
```

**Elements:**

- `name` The qualified name of the behaviour.

### Attributes:

- none.

### 11.3.2.17 TciTestCaseIdType

**TciTestCaseIdType** is mapped to the following complex type:

```
<xsd:complexType name="TciTestCaseIdType">
  <xsd:sequence>
    <xsd:element name="name" type="Types:QualifiedName"/>
  </xsd:sequence>
</xsd:complexType>
```

### Elements:

- name The qualified name of the test case.

### Attributes:

- none.

### 11.3.2.18 TciParameterType

**TciParameterType** is mapped to the following complex type:

```
<xsd:complexType name="TciParameterType">
  <xsd:sequence>
    <xsd:element name="val" type="Values:Value"/>
  </xsd:sequence>
  <xsd:attribute name="name" type="SimpleTypes:TString"/>
  <xsd:attribute name="mode" type="SimpleTypes:TciParameterPassingModeType"/>
</xsd:complexType>
```

### Elements:

- val The encoded parameter.

### Attributes:

- name The parameter name.
- mode The parameter passing mode.

### 11.3.2.19 TciParameterListType

**TciParameterListType** is mapped to the following complex type:

```
<xsd:complexType name="TciParameterListType">
  <xsd:sequence>
    <xsd:element name="par" type="Types:TciParameterType"
      minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>
```

### Sequence of Elements:

- par The parameters in that list.

### Attributes:

- none.

## 11.3.3 Abstract value mapping

### 11.3.3.1 Value

**value** is mapped to the following complex type:

```
<xsd:complexType name="Value" mixed="true">
  <xsd:choice>
    <xsd:element name="integer" type="Values:IntegerValue"/>
    <xsd:element name="float" type="Values:FloatValue"/>
  </xsd:choice>
</xsd:complexType>
```



```

<xsd:element name="boolean" type="Values:BooleanValue"/>
<xsd:element name="verdicttype" type="Values:VerdictValue"/>
<xsd:element name="bitstring" type="Values:BitstringValue"/>
<xsd:element name="hexstring" type="Values:HexstringValue"/>
<xsd:element name="octetstring" type="Values:OctetstringValue"/>
<xsd:element name="charstring" type="Values:CharstringValue"/>
<xsd:element name="universal_charstring" type="Values:UniversalCharstringValue"/>
<xsd:element name="record" type="Values:RecordValue"/>
<xsd:element name="record_of" type="Values:RecordOfValue"/>
<xsd:element name="array" type="Values:ArrayValue"/>
<xsd:element name="set" type="Values:SetValue"/>
<xsd:element name="set_of" type="Values:SetOfValue"/>
<xsd:element name="enumerated" type="Values:EnumeratedValue"/>
<xsd:element name="union" type="Values:UnionValue"/>
<xsd:element name="anytype" type="Values:AnytypeValue"/>
<xsd:element name="address" type="Values:AddressValue"/>
</xsd:choice>
<xsd:attributeGroup ref="Values:ValueAtts"/>
</xsd:complexType>

<xsd:attributeGroup name="ValueAtts">
  <xsd:attribute name="name" type="SimpleTypes:TString" use="optional"/>
  <xsd:attribute name="type" type="SimpleTypes:TString" use="optional"/>
  <xsd:attribute name="module" type="SimpleTypes:TString" use="optional"/>
  <xsd:attribute name="annotation" type="SimpleTypes:TString" use="optional"/>
</xsd:attributeGroup>

```

### Choice of Elements:

- integer                    An integer value.
- float                     A float value.
- boolean                    A boolean value.
- verdicttype                A verdicttype value.
- bitstring                  A bitstring value.
- hexstring                  A hexstring value.
- octetstring                An octetstring value.
- charstring                 A charstring value.
- universal\_charstring      A universal charstring value.
- record                     A record value.
- record\_of                  A record of value.
- array                      An array value.
- set                         A set value.
- set\_of                     A set of value.
- enumerated                 An enumerated value.
- union                      A union value.
- anytype                    An anytype value.
- address                    An address value.

### Attributes:

- name                      The name of the value, if known.
- type                      The type of the value, if known.
- module                    The module of the value, if known.
- annotation                A helper attribute to provide additional matching/mismatching information, etc.

### 11.3.3.2 IntegerValue

**IntegerValue** is mapped to the following complex type:

```
<xsd:complexType name="IntegerValue">
  <xsd:choice >
    <xsd:element name="value" type="SimpleTypes:TString"/>
    <xsd:element name="null" type="Templates:null"/>
    <xsd:element name="omit" type="Templates:omit"/>
  </xsd:choice>
  <xsd:attributeGroup ref="Values:ValueAtts"/>
</xsd:complexType>
```

#### Simple Content:

- value The integer value as string.
- null If no value is given.
- omit If the value is omitted.

#### Attributes:

- The same attributes as those of Value.

### 11.3.3.3 FloatValue

**FloatValue** is mapped to the following complex type:

```
<xsd:complexType name="FloatValue">
  <xsd:simpleContent>
    <xsd:extension base="SimpleTypes:TString">
      <xsd:attributeGroup ref="Values:ValueAtts"/>
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>
```

#### Simple Content:

- value The float value as string.
- null If no value is given.
- omit If the value is omitted.

#### Attributes:

- The same attributes as those of Value.

### 11.3.3.4 BooleanValue

**BooleanValue** is mapped to the following complex type:

```
<xsd:complexType name="BooleanValue">
  <xsd:simpleContent>
    <xsd:extension base="SimpleTypes:TString">
      <xsd:attributeGroup ref="Values:ValueAtts"/>
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>
```

#### Simple Content:

- value The boolean value as string.
- null If no value is given.
- omit If the value is omitted.

#### Attributes:

- The same attributes as those of Value.

### 11.3.3.5 Void

### 11.3.3.6 VerdictValue

**VerdictValue** is mapped to the following complex type:

```
<xsd:complexType name="VerdictValue">
  <xsd:simpleContent>
    <xsd:extension base="SimpleTypes:TString">
      <xsd:attributeGroup ref="Values:ValueAtts"/>
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>
```

#### Simple Content:

- value The verdict value as string.
- null If no value is given.
- omit If the value is omitted.

#### Attributes:

- The same attributes as those of Value.

### 11.3.3.7 BitstringValue

**BitstringValue** is mapped to the following complex type:

```
<xsd:complexType name="BitstringValue">
  <xsd:simpleContent>
    <xsd:extension base="SimpleTypes:TString">
      <xsd:attributeGroup ref="Values:ValueAtts"/>
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>
```

#### Simple Content:

- value The bitstring value as string.
- null If no value is given.
- omit If the value is omitted.

#### Attributes:

- The same attributes as those of Value.

### 11.3.3.8 HexstringValue

**HexstringValue** is mapped to the following complex type:

```
<xsd:complexType name="HexstringValue">
  <xsd:simpleContent>
    <xsd:extension base="SimpleTypes:TString">
      <xsd:attributeGroup ref="Values:ValueAtts"/>
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>
```

#### Simple Content:

- value The hexstring value as string.
- null If no value is given.
- omit If the value is omitted.

#### Attributes:

- The same attributes as those of Value.

### 11.3.3.9 OctetstringValue

**OctetstringValue** is mapped to the following complex type:

```
<xsd:complexType name="OctetstringValue">
  <xsd:simpleContent>
    <xsd:extension base="SimpleTypes:TString">
      <xsd:attributeGroup ref="Values:ValueAtts"/>
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>
```

#### Simple Content:

- value The octetstring value as string.
- null If no value is given.
- omit If the value is omitted.

#### Attributes:

- The same attributes as those of Value.

### 11.3.3.10 CharstringValue

**CharstringValue** is mapped to the following complex type:

```
<xsd:complexType name="CharstringValue">
  <xsd:simpleContent>
    <xsd:extension base="SimpleTypes:TString">
      <xsd:attributeGroup ref="Values:ValueAtts"/>
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>
```

#### Simple Content:

- value The charstring value as string.
- null If no value is given.
- omit If the value is omitted.

#### Attributes:

- The same attributes as those of Value.

### 11.3.3.11 UniversalCharstringValue

**UniversalCharstringValue** is mapped to the following complex type:

```
<xsd:complexType name="UniversalCharstringValue">
  <xsd:simpleContent>
    <xsd:extension base="SimpleTypes:TString">
      <xsd:attributeGroup ref="Values:ValueAtts"/>
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>
```

#### Simple Content:

- value The universal charstring value as string.
- null If no value is given.
- omit If the value is omitted.

#### Attributes:

- The same attributes as those of Value.

### 11.3.3.12 RecordValue

**RecordValue** is mapped to the following complex type:

```
<xsd:complexType name="RecordValue">
  <xsd:choice>
    <xsd:choice minOccurs="0" maxOccurs="unbounded">
      <xsd:element name="integer" type="Values:IntegerValue"/>
      <xsd:element name="float" type="Values:FloatValue"/>
      <xsd:element name="boolean" type="Values:BooleanValue"/>
      <xsd:element name="verdicttype" type="Values:VerdictValue"/>
      <xsd:element name="bitstring" type="Values:BitstringValue"/>
      <xsd:element name="hexstring" type="Values:HexstringValue"/>
      <xsd:element name="octetstring" type="Values:OctetstringValue"/>
      <xsd:element name="charstring" type="Values:CharstringValue"/>
      <xsd:element name="universal_charstring"
        type="Values:UniversalCharstringValue"/>
      <xsd:element name="record" type="Values:RecordValue"/>
      <xsd:element name="record_of" type="Values:RecordOfValue"/>
      <xsd:element name="array" type="Values:ArrayValue"/>
      <xsd:element name="set" type="Values:SetValue"/>
      <xsd:element name="set_of" type="Values:SetOfValue"/>
      <xsd:element name="enumerated" type="Values:EnumeratedValue"/>
      <xsd:element name="union" type="Values:UnionValue"/>
      <xsd:element name="anytype" type="Values:AnytypeValue"/>
      <xsd:element name="address" type="Values:AddressValue"/>
    </xsd:choice>
    <xsd:element name="null" type="Templates:null"/>
    <xsd:element name="omit" type="Templates:omit"/>
  </xsd:choice>
  <xsd:attributeGroup ref="Values:ValueAtts"/>
</xsd:complexType>
```

#### Sequence of Elements:

- integer An integer value.
- float A float value.
- boolean A boolean value.
- verdicttype A verdicttype value.
- bitstring A bitstring value.
- hexstring A hexstring value.
- octetstring An octetstring value.
- charstring A charstring value.
- universal\_charstring A universal charstring value.
- record A record value.
- record\_of A record of value.
- array An array value.
- set A set value.
- set\_of A set of value.
- enumerated An enumerated value.
- union A union value.
- anytype An anytype value.
- address An address value.
- null If no field is given.
- omit If the field is omitted.

## Attributes:

- The same attributes as those of Value.

### 11.3.3.13 RecordOfValue

**RecordOfValue** is mapped to the following complex type:

```
<xsd:complexType name="RecordOfValue">
  <xsd:choice>
    <xsd:element name="integer" type="Values:IntegerValue" minOccurs="0"
      maxOccurs="unbounded"/>
    <xsd:element name="float" type="Values:FloatValue" minOccurs="0"
      maxOccurs="unbounded"/>
    <xsd:element name="boolean" type="Values:BooleanValue" minOccurs="0"
      maxOccurs="unbounded"/>
    <xsd:element name="verdicttype" type="Values:VerdictValue" minOccurs="0"
      maxOccurs="unbounded"/>
    <xsd:element name="bitstring" type="Values:BitstringValue"
      minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element name="hexstring" type="Values:HexstringValue"
      minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element name="octetstring" type="Values:OctetstringValue"
      minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element name="charstring" type="Values:CharstringValue"
      minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element name="universal_charstring"
      type="Values:UniversalCharstringValue" minOccurs="0"
      maxOccurs="unbounded"/>
    <xsd:element name="record" type="Values:RecordValue" minOccurs="0"
      maxOccurs="unbounded"/>
    <xsd:element name="record_of" type="Values:RecordOfValue"
      minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element name="array" type="Values:ArrayValue"
      minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element name="set" type="Values:SetValue" minOccurs="0"
      maxOccurs="unbounded"/>
    <xsd:element name="set_of" type="Values:SetOfValue"
      minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element name="enumerated" type="Values:EnumeratedValue"
      minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element name="union" type="Values:UnionValue" minOccurs="0"
      maxOccurs="unbounded"/>
    <xsd:element name="anytype" type="Values:AnytypeValue" minOccurs="0"
      maxOccurs="unbounded"/>
    <xsd:element name="address" type="Values:AddressValue" minOccurs="0"
      maxOccurs="unbounded"/>
    <xsd:element name="null" type="Templates:null"/>
    <xsd:element name="omit" type="Templates:omit"/>
  </xsd:choice>
  <xsd:attributeGroup ref="Values:ValueAtts"/>
</xsd:complexType>
```

#### Choice of Sequence of Elements:

- integer An integer value.
- float A float value.
- boolean A boolean value.
- verdicttype A verdicttype value.
- bitstring A bitstring value.
- hexstring A hexstring value.
- octetstring An octetstring value.
- charstring A charstring value.
- universal\_charstring A universal charstring value.
- record A record value.
- record\_of A record of value.

- array An array value.
- set A set value.
- set\_of A set of value.
- enumerated An enumerated value.
- union A union value.
- anytype An anytype value.
- address An address value.
- null If no field is given.
- omit If the field is omitted.

#### Attributes:

- The same attributes as those of Value.

### 11.3.3.14 ArrayValue

**ArrayValue** is mapped to the following complex type:

```

<xsd:complexType name="ArrayValue">
  <xsd:choice>
    <xsd:element name="integer" type="Values:IntegerValue" minOccurs="0"
      maxOccurs="unbounded"/>
    <xsd:element name="float" type="Values:FloatValue" minOccurs="0"
      maxOccurs="unbounded"/>
    <xsd:element name="boolean" type="Values:BooleanValue" minOccurs="0"
      maxOccurs="unbounded"/>
    <xsd:element name="verdicttype" type="Values:VerdictValue" minOccurs="0"
      maxOccurs="unbounded"/>
    <xsd:element name="bitstring" type="Values:BitstringValue"
      minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element name="hexstring" type="Values:HexstringValue"
      minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element name="octetstring" type="Values:OctetstringValue"
      minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element name="charstring" type="Values:CharstringValue"
      minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element name="universal_charstring"
      type="Values:UniversalCharstringValue" minOccurs="0"
      maxOccurs="unbounded"/>
    <xsd:element name="record" type="Values:RecordValue" minOccurs="0"
      maxOccurs="unbounded"/>
    <xsd:element name="record_of" type="Values:RecordOfValue"
      minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element name="array" type="Values:ArrayValue"
      minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element name="set" type="Values:SetValue" minOccurs="0"
      maxOccurs="unbounded"/>
    <xsd:element name="set_of" type="Values:SetOfValue"
      minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element name="enumerated" type="Values:EnumeratedValue"
      minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element name="union" type="Values:UnionValue" minOccurs="0"
      maxOccurs="unbounded"/>
    <xsd:element name="anytype" type="Values:AnytypeValue" minOccurs="0"
      maxOccurs="unbounded"/>
    <xsd:element name="address" type="Values:AddressValue" minOccurs="0"
      maxOccurs="unbounded"/>
    <xsd:element name="null" type="Templates:null"/>
    <xsd:element name="omit" type="Templates:omit"/>
  </xsd:choice>
  <xsd:attributeGroup ref="Values:ValueAtts"/>
</xsd:complexType>

```

#### Choice of Sequence of Elements:

- integer An integer value.
- float A float value.

- `boolean` A boolean value.
- `verdicttype` A verdicttype value.
- `bitstring` A bitstring value.
- `hexstring` A hexstring value.
- `octetstring` An octetstring value.
- `charstring` A charstring value.
- `universal_charstring` A universal charstring value.
- `record` A record value.
- `record_of` A record of value.
- `array` An array value.
- `set` A set value.
- `set_of` A set of value.
- `enumerated` An enumerated value.
- `union` A union value.
- `anytype` An anytype value.
- `address` An address value.
- `null` If no field is given.
- `omit` If the field is omitted.

#### Attributes:

- The same attributes as those of Value.

#### 11.3.3.15 SetValue

`SetValue` is mapped to the following complex type:

```
<xsd:complexType name="SetValue">
  <xsd:choice>
    <xsd:choice minOccurs="0" maxOccurs="unbounded">
      <xsd:element name="integer" type="Values:IntegerValue"/>
      <xsd:element name="float" type="Values:FloatValue"/>
      <xsd:element name="boolean" type="Values:BooleanValue"/>
      <xsd:element name="verdicttype" type="Values:VerdictValue"/>
      <xsd:element name="bitstring" type="Values:BitstringValue"/>
      <xsd:element name="hexstring" type="Values:HexstringValue"/>
      <xsd:element name="octetstring" type="Values:OctetstringValue"/>
      <xsd:element name="charstring" type="Values:CharstringValue"/>
      <xsd:element name="universal_charstring"
        type="Values:UniversalCharstringValue"/>
      <xsd:element name="record" type="Values:RecordValue"/>
      <xsd:element name="record_of" type="Values:RecordOfValue"/>
      <xsd:element name="array" type="Values:ArrayValue"/>
      <xsd:element name="set" type="Values:SetValue"/>
      <xsd:element name="set_of" type="Values:SetOfValue"/>
      <xsd:element name="enumerated" type="Values:EnumeratedValue"/>
      <xsd:element name="union" type="Values:UnionValue"/>
      <xsd:element name="anytype" type="Values:AnytypeValue"/>
      <xsd:element name="address" type="Values:AddressValue"/>
    </xsd:choice>
    <xsd:element name="null" type="Templates:null"/>
    <xsd:element name="omit" type="Templates:omit"/>
  </xsd:choice>
  <xsd:attributeGroup ref="Values:ValueAtts"/>
</xsd:complexType>
```

#### Sequence of Elements:

- `integer` An integer value.



- float A float value.
- boolean A boolean value.
- verdicttype A verdicttype value.
- bitstring A bitstring value.
- hexstring A hexstring value.
- octetstring An octetstring value.
- charstring A charstring value.
- universal\_charstring A universal charstring value.
- record A record value.
- record\_of A record of value.
- array An array value.
- set A set value.
- set\_of A set of value.
- enumerated An enumerated value.
- union A union value.
- anytype An anytype value.
- address An address value.
- null If no field is given.
- omit If the field is omitted.

#### Attributes:

- The same attributes as those of Value.

#### 11.3.3.16 SetOfValue

`SetOfValue` is mapped to the following complex type:

```
<xsd:complexType name="SetOfValue">
  <xsd:choice>
    <xsd:element name="integer" type="Values:IntegerValue" minOccurs="0"
      maxOccurs="unbounded"/>
    <xsd:element name="float" type="Values:FloatValue" minOccurs="0"
      maxOccurs="unbounded"/>
    <xsd:element name="boolean" type="Values:BooleanValue" minOccurs="0"
      maxOccurs="unbounded"/>
    <xsd:element name="verdicttype" type="Values:VerdictValue" minOccurs="0"
      maxOccurs="unbounded"/>
    <xsd:element name="bitstring" type="Values:BitstringValue"
      minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element name="hexstring" type="Values:HexstringValue"
      minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element name="octetstring" type="Values:OctetstringValue"
      minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element name="charstring" type="Values:CharstringValue"
      minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element name="universal_charstring"
      type="Values:UniversalCharstringValue" minOccurs="0"
      maxOccurs="unbounded"/>
    <xsd:element name="record" type="Values:RecordValue" minOccurs="0"
      maxOccurs="unbounded"/>
    <xsd:element name="record_of" type="Values:RecordOfValue"
      minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element name="array" type="Values:ArrayValue"
      minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element name="set" type="Values:SetValue" minOccurs="0"
      maxOccurs="unbounded"/>
    <xsd:element name="set_of" type="Values:SetOfValue"
      minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element name="enumerated" type="Values:EnumeratedValue"
```

```

        minOccurs="0" maxOccurs="unbounded"/>
<xsd:element name="union" type="Values:UnionValue" minOccurs="0"
    maxOccurs="unbounded"/>
<xsd:element name="anytype" type="Values:AnytypeValue" minOccurs="0"
    maxOccurs="unbounded"/>
<xsd:element name="address" type="Values:AddressValue" minOccurs="0"
    maxOccurs="unbounded"/>
<xsd:element name="null" type="Templates:null"/>
<xsd:element name="omit" type="Templates:omit"/> </xsd:choice>
<xsd:attributeGroup ref="Values:ValueAtts"/>
</xsd:complexType>

```

### Choice of Sequence of Elements:

- integer An integer value.
- float A float value.
- boolean A boolean value.
- verdicttype A verdicttype value.
- bitstring A bitstring value.
- hexstring A hexstring value.
- octetstring An octetstring value.
- charstring A charstring value.
- universal\_charstring A universal charstring value.
- record A record value.
- record\_of A record of value.
- array An array value.
- set A set value.
- set\_of A set of value.
- enumerated An enumerated value.
- union A union value.
- anytype An anytype value.
- address An address value.
- null If no field is given.
- omit If the field is omitted.

### Attributes:

- The same attributes as those of Value.

#### 11.3.3.17 EnumeratedValue

**EnumeratedValue** is mapped to the following complex type:

```

<xsd:complexType name="EnumeratedValue">
  <xsd:choice>
    <xsd:element name="value" type="SimpleTypes:TString"/>
    <xsd:element name="intValue" type="SimpleTypes:TInteger" minOccurs="0"/>
    <xsd:element name="null" type="Templates:null"/>
    <xsd:element name="omit" type="Templates:omit"/>
  </xsd:choice>
</xsd:sequence>
<xsd:attributeGroup ref="Values:ValueAtts"/>
</xsd:complexType>

```

### Sequence of Elements:

- value The enumeration value.

- intValue           The integer value.
- null                If no value is given.
- omit                If the value is omitted.

**Attributes:**

- The same attributes as those of Value.

**11.3.3.18 UnionValue**

UnionValue is mapped to the following complex type:

```
<xsd:complexType name="UnionValue">
  <xsd:choice>
    <xsd:element name="integer" type="Values:IntegerValue"/>
    <xsd:element name="float" type="Values:FloatValue"/>
    <xsd:element name="boolean" type="Values:BooleanValue"/>
    <xsd:element name="verdicttype" type="Values:VerdictValue"/>
    <xsd:element name="bitstring" type="Values:BitstringValue"/>
    <xsd:element name="hexstring" type="Values:HexstringValue"/>
    <xsd:element name="octetstring" type="Values:OctetstringValue"/>
    <xsd:element name="charstring" type="Values:CharstringValue"/>
    <xsd:element name="universal_charstring"
      type="Values:UniversalCharstringValue"/>
    <xsd:element name="record" type="Values:RecordValue"/>
    <xsd:element name="record_of" type="Values:RecordOfValue"/>
    <xsd:element name="array" type="Values:ArrayValue"/>
    <xsd:element name="set" type="Values:SetValue"/>
    <xsd:element name="set_of" type="Values:SetOfValue"/>
    <xsd:element name="enumerated" type="Values:EnumeratedValue"/>
    <xsd:element name="union" type="Values:UnionValue"/>
    <xsd:element name="anytype" type="Values:AnytypeValue"/>
    <xsd:element name="address" type="Values:AddressValue"/>
    <xsd:element name="null" type="Templates:null"/>
    <xsd:element name="omit" type="Templates:omit"/>
  </xsd:choice>
  <xsd:attributeGroup ref="Values:ValueAtts"/>
</xsd:complexType>
```

**Choice of Elements:**

- integer            An integer value.
- float              A float value.
- boolean            A boolean value.
- verdicttype        A verdicttype value.
- bitstring          A bitstring value.
- hexstring          A hexstring value.
- octetstring        An octetstring value.
- charstring         A charstring value.
- universal\_charstring A universal charstring value.
- record             A record value.
- record\_of          A record of value.
- array              An array value.
- set                 A set value.
- set\_of             A set of value.
- enumerated         An enumerated value.
- union              A union value.
- anytype            An anytype value.

- address An address value.

**Attributes:**

- The same attributes as those of Value.

**11.3.3.19 AnytypeValue**

**AnytypeValue** is mapped to the following complex type:

```
<xsd:complexType name="AnytypeValue">
  <xsd:choice minOccurs="0" maxOccurs="unbounded">
    <xsd:element name="integer" type="Values:IntegerValue"/>
    <xsd:element name="float" type="Values:FloatValue"/>
    <xsd:element name="boolean" type="Values:BooleanValue"/>
    <xsd:element name="verdicttype" type="Values:VerdictValue"/>
    <xsd:element name="bitstring" type="Values:BitstringValue"/>
    <xsd:element name="hexstring" type="Values:HexstringValue"/>
    <xsd:element name="octetstring" type="Values:OctetstringValue"/>
    <xsd:element name="charstring" type="Values:OctetstringValue"/>
    <xsd:element name="universal_charstring"
      type="Values:UniversalCharstringValue"/>
    <xsd:element name="record" type="Values:RecordValue"/>
    <xsd:element name="record_of" type="Values:RecordOfValue"/>
    <xsd:element name="array" type="Values:ArrayValue"/>
    <xsd:element name="set" type="Values:SetValue"/>
    <xsd:element name="set_of" type="Values:SetOfValue"/>
    <xsd:element name="enumerated" type="Values:EnumeratedValue"/>
    <xsd:element name="union" type="Values:UnionValue"/>
    <xsd:element name="address" type="Values:AddressValue"/>
    <xsd:element name="null" type="Templates:null"/>
    <xsd:element name="omit" type="Templates:omit"/>
  </xsd:choice>
  <xsd:attributeGroup ref="Values:ValueAtts"/>
</xsd:complexType>
```

**Choice of Elements:**

- integer An integer value.
- float A float value.
- boolean A boolean value.
- verdicttype A verdicttype value.
- bitstring A bitstring value.
- hexstring A hexstring value.
- octetstring An octetstring value.
- charstring A charstring value.
- universal\_charstring A universal charstring value.
- record A record value.
- record\_of A record of value.
- array An array value.
- set A set value.
- set\_of A set of value.
- enumerated An enumerated value.
- union A union value.
- address An address value.

**Attributes:**

- The same attributes as those of Value.

### 11.3.3.20 AddressValue

**AddressValue** is mapped to the following complex type:

```
<xsd:complexType name="AddressValue">
  <xsd:choice minOccurs="0" maxOccurs="unbounded">
    <xsd:element name="integer" type="Values:IntegerValue"/>
    <xsd:element name="float" type="Values:FloatValue"/>
    <xsd:element name="boolean" type="Values:BooleanValue"/>
    <xsd:element name="verdicttype" type="Values:VerdictValue"/>
    <xsd:element name="bitstring" type="Values:BitstringValue"/>
    <xsd:element name="hexstring" type="Values:HexstringValue"/>
    <xsd:element name="octetstring" type="Values:OctetstringValue"/>
    <xsd:element name="charstring" type="Values:OctetstringValue"/>
    <xsd:element name="universal_charstring"
      type="Values:UniversalCharstringValue"/>
    <xsd:element name="record" type="Values:RecordValue"/>
    <xsd:element name="record_of" type="Values:RecordOfValue"/>
    <xsd:element name="array" type="Values:ArrayValue"/>
    <xsd:element name="set" type="Values:SetValue"/>
    <xsd:element name="set_of" type="Values:SetOfValue"/>
    <xsd:element name="enumerated" type="Values:EnumeratedValue"/>
    <xsd:element name="union" type="Values:UnionValue"/>
    <xsd:element name="anytype" type="Values:AnytypeValue"/>
    <xsd:element name="null" type="Templates:null"/>
    <xsd:element name="omit" type="Templates:omit"/>
  </xsd:choice>
  <xsd:attributeGroup ref="Values:ValueAtts"/>
</xsd:complexType>
```

#### Choice of Elements:

- integer An integer value.
- float A float value.
- boolean A boolean value.
- verdicttype A verdicttype value.
- bitstring A bitstring value.
- hexstring A hexstring value.
- octetstring An octetstring value.
- charstring A charstring value.
- universal\_charstring A universal charstring value.
- record A record value.
- record\_of A record of value.
- array An array of value.
- set A set value.
- set\_of A set of value.
- enumerated An enumerated value.
- union A union value.
- anytype An anytype value.

#### Attributes:

- The same attributes as those of Value.

### 11.3.4 Abstract logging types mapping

Additional types are defined to ease the logging of matches between values and templates.

### 11.3.4.1 TciValueTemplate

**TciValueTemplate** is mapped to the following complex type:

```
<xsd:complexType name="TciValueTemplate">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Values:Value">
      <xsd:choice minOccurs="0">
        <xsd:element name="integer" type="Templates:IntegerTemplate"/>
        <xsd:element name="float" type="Templates:FloatTemplate"/>
        <xsd:element name="boolean" type="Templates:BooleanTemplate"/>
        <xsd:element name="verdicttype" type="Templates:VerdictTemplate"/>
        <xsd:element name="bitstring" type="Templates:BitstringTemplate"/>
        <xsd:element name="hexstring" type="Templates:HexstringTemplate"/>
        <xsd:element name="octetstring" type="Templates:OctetstringTemplate"/>
        <xsd:element name="charstring" type="Templates:CharstringTemplate"/>
        <xsd:element name="universal_charstring"
          type="Templates:UniversalCharstringTemplate"/>
        <xsd:element name="record" type="Templates:RecordTemplate"/>
        <xsd:element name="record_of" type="Templates:RecordOfTemplate"/>
        <xsd:element name="array" type="Templates:ArrayTemplate"/>
        <xsd:element name="set" type="Templates:SetTemplate"/>
        <xsd:element name="set_of" type="Templates:SetOfTemplate"/>
        <xsd:element name="enumerated" type="Templates:EnumeratedTemplate"/>
        <xsd:element name="union" type="Templates:UnionTemplate"/>
        <xsd:element name="anytype" type="Templates:AnytypeTemplate"/>
        <xsd:element name="address" type="Templates:AddressTemplate"/>
        <xsd:element name="omit" type="Templates:omit"/>
        <xsd:element name="any" type="Templates:any"/>
        <xsd:element name="anyoromit" type="Templates:anyoromit"/>
        <xsd:element name="templateDef" type="SimpleTypes:TString"/>
      </xsd:choice>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

#### Choice of Elements:

- integer An integer template.
- float A float template.
- boolean A boolean template.
- verdicttype A verdicttype template.
- bitstring A bitstring template.
- hexstring A hexstring template.
- octetstring An octetstring template.
- charstring A charstring template.
- universal\_charstring A universal charstring template.
- record A record template.
- record\_of A record of template.
- array An array template.
- set A set template.
- set\_of A set of template.
- enumerated An enumerated template.
- union A union template.
- anytype An anytype template.
- address An address template.
- omit An omit template.
- any An any template.

- anyoromit                      An anyoromit template.
- templateDef                    A complex template definition.

**Attributes:**

- none.

**11.3.4.2 TciNonValueTemplate**

**TciNonValueTemplate** is mapped to the following complex type:

```
<xsd:complexType name="TciNonValueTemplate">
  <xsd:sequence>
    <xsd:choice>
      <xsd:element name="any" type="Templates:any"/>
      <xsd:element name="all" type="Templates:all"/>
      <xsd:element name="templateDef" type="SimpleTypes:TString"/>
      <xsd:element name="null" type="Templates:null"/>
    </xsd:choice>
  </xsd:sequence>
</xsd:complexType>
```

**Choice of Elements:**

- any                              An any template.
- all                                An all template.
- templateDef                    A complex template definition.
- null                                No template is given.

**Attributes:**

- none.

**11.3.4.3 TciValueList**

**TciValueList** is mapped to the following complex type:

```
<xsd:complexType name="TciValueListType">
  <xsd:sequence>
    <xsd:element name="val" type="Values:Value"
      maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>
```

**Sequence of Elements:**

- val                                The values in the value list.

**Attributes:**

- none.

**11.3.4.4 TciValueDifference**

**TciValueDifference** is mapped to the following complex type:

```
<xsd:complexType name="TciValueDifference">
  <xsd:sequence>
    <xsd:element name="val" type="SimpleTypes:xpath"/>
    <xsd:element name="tpl"
type="SimpleTypes:xpath"/>
  </xsd:sequence>
  <xsd:attributeGroup ref="Values:ValueAtts"/>
  <xsd:attribute name="desc" type="SimpleTypes:TString"
use="optional"/>
</xsd:complexType>
```

### Sequence of Elements:

- `val` A reference to the mismatching value.
- `tmpl` A reference to the template.

### Attributes:

- The same attributes as those of Value.
- `desc` The reason of the mismatch.

#### 11.3.4.5 TciValueDifferenceList

**TciValueDifferenceList** is mapped to the following complex type:

```
<xsd:complexType name="TciValueDifferenceList">
  <xsd:sequence>
    <xsd:element name="diff" type="Templates:TciValueDifference"
      maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>
```

### Sequence of Elements:

- `diff` The value/template differences in the value difference list.

### Attributes:

- none.

## 11.4 Mapping of the operations on the logging interface

Every operation provided at the logging interface has a corresponding complex type definition in XML. These complex type definitions are extensions of Event.

### 11.4.1 Event

**Event** is mapped to the following complex type:

```
<!-- common definition for all events -->
<xsd:complexType name="Event" mixed="true">
  <xsd:sequence>
    <xsd:element name="am" type="SimpleTypes:TString"/>
  </xsd:sequence>
  <xsd:attribute name="ts" type="xsd:long" use="required"/>
  <xsd:attribute name="src" type="SimpleTypes:TString" use="optional"/>
  <xsd:attribute name="line" type="SimpleTypes:TInteger" use="optional"/>

  <!-- general identifier structure for test components, ports and timer -->
  <xsd:attribute name="name" type="SimpleTypes:TString" use="required"/>
  <xsd:attribute name="id" type="SimpleTypes:TString" use="required"/>
  <xsd:attribute name="type" type="SimpleTypes:TString" use="required"/>
</xsd:complexType>
```

### Elements:

- `am` A message, to be used for further information in the log.

### Attributes:

- `ts` The time when the event is produced.
- `src` The source file of the test specification.
- `line` The line number where the request is performed.
- `name` The name of the component which produces this event.
- `id` The id of the component which produces this event.
- `type` The type of the component which produces this event.



## 11.4.2 The TCI-TL interface

### 11.4.2.1 TCI-TL provided

The TCI-TL Provided interface is mapped to the following interface:

```
<!-- testcases -->
<xsd:complexType name="tliTcExecute">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:Event">
      <xsd:sequence>
        <xsd:element name="tcId" type="Types:TciTestCaseIdType"/>
        <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>
        <xsd:element name="dur" type="SimpleTypes:TriTimerDurationType" minOccurs="0"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliTcStart">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:Event">
      <xsd:sequence>
        <xsd:element name="tcId" type="Types:TciTestCaseIdType"/>
        <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>
        <xsd:element name="dur" type="SimpleTypes:TriTimerDurationType" minOccurs="0"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliTcStop">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:Event"/>
    <xsd:sequence>
      <xsd:element name="reason" type="SimpleTypes:TString" minOccurs="0"/>
    </xsd:sequence>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliTcStarted">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:Event">
      <xsd:sequence>
        <xsd:element name="tcId" type="Types:TciTestCaseIdType"/>
        <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>
        <xsd:element name="dur" type="SimpleTypes:TriTimerDurationType" minOccurs="0"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliTcTerminated">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:Event">
      <xsd:sequence>
        <xsd:element name="tcId" type="Types:TciTestCaseIdType"/>
        <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>
        <xsd:element name="verdict" type="Values:VerdictValue"/>
        <xsd:element name="reason" type="SimpleTypes:TString" minOccurs="0"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<!-- control -->
<xsd:complexType name="tliCtrlStart">
  <xsd:complexContent>
    <xsd:extension base="Events:Event"/>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliCtrlStop">
  <xsd:complexContent>
    <xsd:extension base="Events:Event"/>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliCtrlTerminated">
```

```

    <xsd:complexContent>
      <xsd:extension base="Events:Event"/>
    </xsd:complexContent>
  </xsd:complexType>

  <!-- asynchronous communication -->
  <xsd:complexType name="tliMSend_m">
    <xsd:complexContent mixed="true">
      <xsd:extension base="Events:Event">
        <xsd:sequence>
          <xsd:element name="at" type="Types:TriPortIdType"/>
          <xsd:element name="to" type="Types:TriPortIdType" minOccurs="0"/>
          <xsd:element name="msgValue" type="Values:Value"/>
          <xsd:element name="addrValue" type="Values:Value" minOccurs="0"/>
          <xsd:choice>
            <xsd:element name="encoder-failure" type="SimpleTypes:TciStatusType"
minOccurs="0"/>
            <xsd:sequence>
              <xsd:element name="msg" type="Types:TriMessageType" minOccurs="0"/>
              <xsd:element name="address" type="Types:TriAddressType" minOccurs="0"/>
              <xsd:element name="transmission-failure"
type="SimpleTypes:TriStatusType" minOccurs="0"/>
            </xsd:sequence>
          </xsd:choice>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="tliMSend_m_BC">
    <xsd:complexContent mixed="true">
      <xsd:extension base="Events:Event">
        <xsd:sequence>
          <xsd:element name="at" type="Types:TriPortIdType"/>
          <xsd:element name="to" type="Types:TriPortIdType" minOccurs="0"/>
          <xsd:element name="msgValue" type="Values:Value"/>
          <xsd:choice>
            <xsd:element name="encoder-failure" type="SimpleTypes:TciStatusType"
minOccurs="0"/>
            <xsd:sequence>
              <xsd:element name="msg" type="Types:TriMessageType" minOccurs="0"/>
              <xsd:element name="transmission-failure"
type="SimpleTypes:TriStatusType" minOccurs="0"/>
            </xsd:sequence>
          </xsd:choice>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="tliMSend_m_MC">
    <xsd:complexContent mixed="true">
      <xsd:extension base="Events:Event">
        <xsd:sequence>
          <xsd:element name="at" type="Types:TriPortIdType"/>
          <xsd:element name="to" type="Types:TriPortIdType" minOccurs="0"/>
          <xsd:element name="msgValue" type="Values:Value"/>
          <xsd:element name="addrValues" type="Types:TciValueListType" minOccurs="0"/>
          <xsd:choice>
            <xsd:element name="encoder-failure" type="SimpleTypes:TciStatusType"
minOccurs="0"/>
            <xsd:sequence>
              <xsd:element name="msg" type="Types:TriMessageType" minOccurs="0"/>
              <xsd:element name="addresses" type="Types:TriAddressListType"
minOccurs="0"/>
              <xsd:element name="transmission-failure"
type="SimpleTypes:TriStatusType" minOccurs="0"/>
            </xsd:sequence>
          </xsd:choice>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="tliMSend_c">
    <xsd:complexContent mixed="true">
      <xsd:extension base="Events:Event">
        <xsd:sequence>
          <xsd:element name="at" type="Types:TriPortIdType"/>
          <xsd:element name="to" type="Types:TriPortIdType" minOccurs="0"/>

```

```

                <xsd:element name="msgValue" type="Values:Value"/>
                <xsd:element name="transmission-failure" type="SimpleTypes:TriStatusType"
minOccurs="0"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliMSend_c_BC">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="at" type="Types:TriPortIdType"/>
                <xsd:element name="to" type="Types:TriPortIdListType" minOccurs="0"/>
                <xsd:element name="msgValue" type="Values:Value"/>
                <xsd:element name="transmission-failure" type="SimpleTypes:TriStatusType"
minOccurs="0"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliMSend_c_MC">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="at" type="Types:TriPortIdType"/>
                <xsd:element name="to" type="Types:TriPortIdListType" minOccurs="0"/>
                <xsd:element name="msgValue" type="Values:Value"/>
                <xsd:element name="transmission-failure" type="SimpleTypes:TriStatusType"
minOccurs="0"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliMDetected_m">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="at" type="Types:TriPortIdType"/>
                <xsd:element name="from" type="Types:TriPortIdType" minOccurs="0"/>
                <xsd:element name="msgValue" type="Types:TriMessageType"/>
                <xsd:element name="address" type="Types:TriAddressType" minOccurs="0"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliMDetected_c">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="at" type="Types:TriPortIdType"/>
                <xsd:element name="from" type="Types:TriPortIdType" minOccurs="0"/>
                <xsd:element name="msgValue" type="Values:Value"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliMMismatch_m">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="at" type="Types:TriPortIdType"/>
                <xsd:element name="msgValue" type="Values:Value"/>
                <xsd:element name="msgTpl" type="Templates:TciValueTemplate"/>
                <xsd:element name="diffs" type="Templates:TciValueDifferenceList"/>
                <xsd:element name="addrValue" type="Values:Value" minOccurs="0"/>
                <xsd:element name="addressTpl" type="Templates:TciValueTemplate"
minOccurs="0"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliMMismatch_c">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">

```

```

        <xsd:sequence>
            <xsd:element name="at" type="Types:TriPortIdType"/>
            <xsd:element name="msgValue" type="Values:Value"/>
            <xsd:element name="msgTmpl" type="Templates:TciValueTemplate"/>
            <xsd:element name="diffs" type="Templates:TciValueDifferenceList"/>
            <xsd:element name="from" type="Types:TriComponentIdType" minOccurs="0"/>
            <xsd:element name="fromTmpl" type="Templates:TciNonValueTemplate"
minOccurs="0"/>
        </xsd:sequence>
    </xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliMReceive_m">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="at" type="Types:TriPortIdType"/>
                <xsd:element name="msgValue" type="Values:Value" minOccurs="0"/>
                <xsd:element name="msgTmpl" type="Templates:TciValueTemplate"
minOccurs="0"/>
                <xsd:element name="addrValue" type="Values:Value" minOccurs="0"/>
                <xsd:element name="addressTmpl" type="Templates:TciValueTemplate"
minOccurs="0"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliMReceive_c">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="at" type="Types:TriPortIdType"/>
                <xsd:element name="msgValue" type="Values:Value" minOccurs="0"/>
                <xsd:element name="msgTmpl" type="Templates:TciValueTemplate"
minOccurs="0"/>
                <xsd:element name="from" type="Types:TriComponentIdType" minOccurs="0"/>
                <xsd:element name="fromTmpl" type="Templates:TciNonValueTemplate"
minOccurs="0"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<!-- synchronous communication -->
<xsd:complexType name="tliPrCall_m">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="at" type="Types:TriPortIdType"/>
                <xsd:element name="to" type="Types:TriPortIdType" minOccurs="0"/>
                <xsd:element name="signature" type="Types:TriSignatureIdType"/>
                <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>
                <xsd:element name="addrValue" type="Values:Value" minOccurs="0"/>
                <xsd:choice>
                    <xsd:element name="encoder-failure" type="SimpleTypes:TciStatusType"
minOccurs="0"/>
                    <xsd:sequence>
                        <xsd:element name="triPars" type="Types:TriParameterListType"
minOccurs="0"/>
                        <xsd:element name="address" type="Types:TriAddressType" minOccurs="0"/>
                        <xsd:element name="transmission-failure"
type="SimpleTypes:TriStatusType" minOccurs="0"/>
                    </xsd:sequence>
                </xsd:choice>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliPrCall_m_BC">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="at" type="Types:TriPortIdType"/>
                <xsd:element name="to" type="Types:TriPortIdType" minOccurs="0"/>
                <xsd:element name="signature" type="Types:TriSignatureIdType"/>
                <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>
                <xsd:choice>

```

```

minOccurs="0"/>
        <xsd:element name="encoder-failure" type="SimpleTypes:TciStatusType"
        <xsd:sequence>
minOccurs="0"/>
            <xsd:element name="triPars" type="Types:TriParameterListType"
            <xsd:element name="transmission-failure"
type="SimpleTypes:TriStatusType" minOccurs="0"/>
            </xsd:sequence>
        </xsd:choice>
    </xsd:sequence>
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliPrCall_m_MC">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="at" type="Types:TriPortIdType"/>
                <xsd:element name="to" type="Types:TriPortIdType" minOccurs="0"/>
                <xsd:element name="signature" type="Types:TriSignatureIdType"/>
                <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>
                <xsd:element name="addrValues" type="Types:TciValueListType" minOccurs="0"/>
                <xsd:choice>
                    <xsd:element name="encoder-failure" type="SimpleTypes:TciStatusType"
minOccurs="0"/>
                    <xsd:sequence>
minOccurs="0"/>
                        <xsd:element name="triPars" type="Types:TriParameterListType"
minOccurs="0"/>
                        <xsd:element name="addresses" type="Types:TriAddressListType"
minOccurs="0"/>
                        <xsd:element name="transmission-failure"
type="SimpleTypes:TriStatusType" minOccurs="0"/>
                    </xsd:choice>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>

<xsd:complexType name="tliPrCall_c">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="at" type="Types:TriPortIdType"/>
                <xsd:element name="to" type="Types:TriPortIdType" minOccurs="0"/>
                <xsd:element name="signature" type="Types:TriSignatureIdType"/>
                <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>
                <xsd:element name="transmission-failure" type="SimpleTypes:TriStatusType"
minOccurs="0"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliPrCall_c_BC">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="at" type="Types:TriPortIdType"/>
                <xsd:element name="to" type="Types:TriPortIdListType" minOccurs="0"/>
                <xsd:element name="signature" type="Types:TriSignatureIdType"/>
                <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>
                <xsd:element name="transmission-failure" type="SimpleTypes:TriStatusType"
minOccurs="0"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliPrCall_c_MC">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="at" type="Types:TriPortIdType"/>
                <xsd:element name="to" type="Types:TriPortIdListType" minOccurs="0"/>
                <xsd:element name="signature" type="Types:TriSignatureIdType"/>
                <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>
                <xsd:element name="transmission-failure" type="SimpleTypes:TriStatusType"
minOccurs="0"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

```

```

        </xsd:sequence>
    </xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliPrGetCallDetected_m">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="at" type="Types:TriPortIdType"/>
                <xsd:element name="from" type="Types:TriPortIdType" minOccurs="0"/>
                <xsd:element name="signature" type="Types:TriSignatureIdType"/>
                <xsd:element name="triPars" type="Types:TriParameterListType" minOccurs="0"/>
                <xsd:element name="address" type="Types:TriAddressType" minOccurs="0"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliPrGetCallDetected_c">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="at" type="Types:TriPortIdType"/>
                <xsd:element name="from" type="Types:TriPortIdType" minOccurs="0"/>
                <xsd:element name="signature" type="Types:TriSignatureIdType"/>
                <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliPrGetCallMismatch_m">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="at" type="Types:TriPortIdType"/>
                <xsd:element name="signature" type="Types:TriSignatureIdType"/>
                <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>
                <xsd:element name="parsTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>
                <xsd:element name="diffs" type="Templates:TciValueDifferenceList"/>
                <xsd:element name="addrValue" type="Values:Value" minOccurs="0"/>
                <xsd:element name="addressTmpl" type="Templates:TciValueTemplate"
minOccurs="0"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliPrGetCallMismatch_c">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="at" type="Types:TriPortIdType"/>
                <xsd:element name="signature" type="Types:TriSignatureIdType"/>
                <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>
                <xsd:element name="parsTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>
                <xsd:element name="diffs" type="Templates:TciValueDifferenceList"/>
                <xsd:element name="from" type="Types:TriComponentIdType" minOccurs="0"/>
                <xsd:element name="fromTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliPrGetCall_m">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="at" type="Types:TriPortIdType"/>
                <xsd:element name="signature" type="Types:TriSignatureIdType"/>
                <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>
                <xsd:element name="parsTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>
                <xsd:element name="addrValue" type="Values:Value" minOccurs="0"/>
                <xsd:element name="addressTmpl" type="Templates:TciValueTemplate"
minOccurs="0"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

```

```

<xsd:complexType name="tliPrGetCall_c">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:Event">
      <xsd:sequence>
        <xsd:element name="at" type="Types:TriPortIdType"/>
        <xsd:element name="signature" type="Types:TriSignatureIdType"/>
        <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>
        <xsd:element name="parsTpl" type="Templates:TciValueTemplate" minOccurs="0"/>
        <xsd:element name="from" type="Types:TriComponentIdType" minOccurs="0"/>
        <xsd:element name="fromTpl" type="Templates:TciNonValueTemplate"
minOccurs="0"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliPrReply_m">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:Event">
      <xsd:sequence>
        <xsd:element name="at" type="Types:TriPortIdType"/>
        <xsd:element name="to" type="Types:TriPortIdType" minOccurs="0"/>
        <xsd:element name="signature" type="Types:TriSignatureIdType"/>
        <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>
        <xsd:element name="replValue" type="Values:Value" minOccurs="0"/>
        <xsd:element name="addrValue" type="Values:Value" minOccurs="0"/>
        <xsd:choice>
          <xsd:element name="encoder-failure" type="SimpleTypes:TciStatusType"
minOccurs="0"/>
          <xsd:sequence>
            <xsd:element name="triPars" type="Types:TriParameterListType"
minOccurs="0"/>
            <xsd:element name="repl" type="Types:TriParameterType" minOccurs="0"/>
            <xsd:element name="address" type="Types:TriAddressType" minOccurs="0"/>
            <xsd:element name="transmission-failure"
              type="SimpleTypes:TciStatusType" minOccurs="0"/>
          </xsd:sequence>
        </xsd:choice>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliPrReply_m_BC">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:Event">
      <xsd:sequence>
        <xsd:element name="at" type="Types:TriPortIdType"/>
        <xsd:element name="to" type="Types:TriPortIdType" minOccurs="0"/>
        <xsd:element name="signature" type="Types:TriSignatureIdType"/>
        <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>
        <xsd:element name="replValue" type="Values:Value" minOccurs="0"/>
        <xsd:choice>
          <xsd:element name="encoder-failure" type="SimpleTypes:TciStatusType"
minOccurs="0"/>
          <xsd:sequence>
            <xsd:element name="triPars" type="Types:TriParameterListType"
minOccurs="0"/>
            <xsd:element name="repl" type="Types:TriParameterType" minOccurs="0"/>
            <xsd:element name="transmission-failure"
              type="SimpleTypes:TciStatusType" minOccurs="0"/>
          </xsd:sequence>
        </xsd:choice>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliPrReply_m_MC">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:Event">
      <xsd:sequence>
        <xsd:element name="at" type="Types:TriPortIdType"/>
        <xsd:element name="to" type="Types:TriPortIdType" minOccurs="0"/>
        <xsd:element name="signature" type="Types:TriSignatureIdType"/>
        <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>
        <xsd:element name="replValue" type="Values:Value" minOccurs="0"/>
        <xsd:element name="addrValues" type="Types:TciValueListType" minOccurs="0"/>
        <xsd:choice>

```

```

        <xsd:element name="encoder-failure" type="SimpleTypes:TciStatusType"
            minOccurs="0"/>
        <xsd:sequence>
            <xsd:element name="triPars" type="Types:TriParameterListType"
                minOccurs="0"/>
            <xsd:element name="repl" type="Types:TriParameterType" minOccurs="0"/>
            <xsd:element name="addresses" type="Types:TriAddressListType"
                minOccurs="0"/>
            <xsd:element name="transmission-failure"
                type="SimpleTypes:TriStatusType" minOccurs="0"/>
        </xsd:sequence>
    </xsd:choice>
</xsd:sequence>
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliPrReply_c">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="at" type="Types:TriPortIdType"/>
                <xsd:element name="to" type="Types:TriPortIdType" minOccurs="0"/>
                <xsd:element name="signature" type="Types:TriSignatureIdType"/>
                <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>
                <xsd:element name="replValue" type="Values:Value" minOccurs="0"/>
                <xsd:element name="transmission-failure" type="SimpleTypes:TriStatusType"
                    minOccurs="0"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliPrReply_c_BC">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="at" type="Types:TriPortIdType"/>
                <xsd:element name="to" type="Types:TriPortIdListType" minOccurs="0"/>
                <xsd:element name="signature" type="Types:TriSignatureIdType"/>
                <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>
                <xsd:element name="replValue" type="Values:Value" minOccurs="0"/>
                <xsd:element name="transmission-failure" type="SimpleTypes:TriStatusType"
                    minOccurs="0"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliPrReply_c_MC">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="at" type="Types:TriPortIdType"/>
                <xsd:element name="to" type="Types:TriPortIdListType" minOccurs="0"/>
                <xsd:element name="signature" type="Types:TriSignatureIdType"/>
                <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>
                <xsd:element name="replValue" type="Values:Value" minOccurs="0"/>
                <xsd:element name="transmission-failure" type="SimpleTypes:TriStatusType"
                    minOccurs="0"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliPrGetReplyDetected_m">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="at" type="Types:TriPortIdType"/>
                <xsd:element name="from" type="Types:TriPortIdType" minOccurs="0"/>
                <xsd:element name="signature" type="Types:TriSignatureIdType"/>
                <xsd:element name="triPars" type="Types:TriParameterListType" minOccurs="0"/>
                <xsd:element name="repl" type="Types:TriParameterType" minOccurs="0"/>
                <xsd:element name="address" type="Types:TriAddressType" minOccurs="0"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

```



```

<xsd:complexType name="tliPrGetReplyDetected_c">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:Event">
      <xsd:sequence>
        <xsd:element name="at" type="Types:TriPortIdType"/>
        <xsd:element name="from" type="Types:TriPortIdType" minOccurs="0"/>
        <xsd:element name="signature" type="Types:TriSignatureIdType"/>
        <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>
        <xsd:element name="replValue" type="Values:Value" minOccurs="0"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliPrGetReplyMismatch_m">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:Event">
      <xsd:sequence>
        <xsd:element name="at" type="Types:TriPortIdType"/>
        <xsd:element name="signature" type="Types:TriSignatureIdType"/>
        <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>
        <xsd:element name="parsTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>
        <xsd:element name="replValue" type="Values:Value" minOccurs="0"/>
        <xsd:element name="replTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>
        <xsd:element name="diffs" type="Templates:TciValueDifferenceList"/>
        <xsd:element name="addrValue" type="Values:Value" minOccurs="0"/>
        <xsd:element name="addressTmpl" type="Templates:TciValueTemplate"
minOccurs="0"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliPrGetReplyMismatch_c">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:Event">
      <xsd:sequence>
        <xsd:element name="at" type="Types:TriPortIdType"/>
        <xsd:element name="signature" type="Types:TriSignatureIdType"/>
        <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>
        <xsd:element name="parsTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>
        <xsd:element name="replValue" type="Values:Value" minOccurs="0"/>
        <xsd:element name="replTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>
        <xsd:element name="diffs" type="Templates:TciValueDifferenceList"/>
        <xsd:element name="from" type="Types:TriComponentIdType" minOccurs="0"/>
        <xsd:element name="fromTmpl" type="Templates:TciNonValueTemplate"
minOccurs="0"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliPrGetReply_m">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:Event">
      <xsd:sequence>
        <xsd:element name="at" type="Types:TriPortIdType"/>
        <xsd:element name="signature" type="Types:TriSignatureIdType"/>
        <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>
        <xsd:element name="parsTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>
        <xsd:element name="replValue" type="Values:Value" minOccurs="0"/>
        <xsd:element name="replTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>
        <xsd:element name="addrValue" type="Values:Value" minOccurs="0"/>
        <xsd:element name="addressTmpl" type="Templates:TciValueTemplate"
minOccurs="0"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliPrGetReply_c">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:Event">
      <xsd:sequence>
        <xsd:element name="at" type="Types:TriPortIdType"/>
        <xsd:element name="signature" type="Types:TriSignatureIdType"/>
        <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>
        <xsd:element name="parsTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>
        <xsd:element name="replValue" type="Values:Value" minOccurs="0"/>
        <xsd:element name="replTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>

```

```

                <xsd:element name="from" type="Types:TriComponentIdType" minOccurs="0"/>
                <xsd:element name="fromTmp1" type="Templates:TciNonValueTemplate"
minOccurs="0"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliPrRaise_m">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="at" type="Types:TriPortIdType"/>
                <xsd:element name="to" type="Types:TriPortIdType" minOccurs="0"/>
                <xsd:element name="signature" type="Types:TriSignatureIdType"/>
                <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>
                <xsd:element name="excValue" type="Values:Value" minOccurs="0"/>
                <xsd:element name="addrValue" type="Values:Value" minOccurs="0"/>
                <xsd:choice>
                    <xsd:element name="encoder-failure" type="SimpleTypes:TciStatusType"
minOccurs="0"/>
                    <xsd:sequence>
                        <xsd:element name="exc" type="Types:TriExceptionType" minOccurs="0"/>
                        <xsd:element name="address" type="Types:TriAddressType" minOccurs="0"/>
                        <xsd:element name="transmission-failure"
type="SimpleTypes:TriStatusType" minOccurs="0"/>
                    </xsd:sequence>
                </xsd:choice>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliPrRaise_m_BC">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="at" type="Types:TriPortIdType"/>
                <xsd:element name="to" type="Types:TriPortIdType" minOccurs="0"/>
                <xsd:element name="signature" type="Types:TriSignatureIdType"/>
                <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>
                <xsd:element name="excValue" type="Values:Value" minOccurs="0"/>
                <xsd:choice>
                    <xsd:element name="encoder-failure" type="SimpleTypes:TciStatusType"
minOccurs="0"/>
                    <xsd:sequence>
                        <xsd:element name="exc" type="Types:TriExceptionType" minOccurs="0"/>
                        <xsd:element name="transmission-failure"
type="SimpleTypes:TriStatusType" minOccurs="0"/>
                    </xsd:sequence>
                </xsd:choice>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliPrRaise_m_MC">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="at" type="Types:TriPortIdType"/>
                <xsd:element name="to" type="Types:TriPortIdType" minOccurs="0"/>
                <xsd:element name="signature" type="Types:TriSignatureIdType"/>
                <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>
                <xsd:element name="excValue" type="Values:Value" minOccurs="0"/>
                <xsd:element name="addrValues" type="Types:TciValueListType" minOccurs="0"/>
                <xsd:choice>
                    <xsd:element name="encoder-failure" type="SimpleTypes:TciStatusType"
minOccurs="0"/>
                    <xsd:sequence>
                        <xsd:element name="exc" type="Types:TriExceptionType" minOccurs="0"/>
                        <xsd:element name="addresses" type="Types:TriAddressListType"
minOccurs="0"/>
                        <xsd:element name="transmission-failure"
type="SimpleTypes:TriStatusType" minOccurs="0"/>
                    </xsd:sequence>
                </xsd:choice>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>

```

```

</xsd:complexType>

<xsd:complexType name="tliPrRaise_c">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:Event">
      <xsd:sequence>
        <xsd:element name="at" type="Types:TriPortIdType"/>
        <xsd:element name="to" type="Types:TriPortIdType" minOccurs="0"/>
        <xsd:element name="signature" type="Types:TriSignatureIdType"/>
        <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>
        <xsd:element name="excValue" type="Values:Value" minOccurs="0"/>
        <xsd:element name="transmission-failure" type="SimpleTypes:TriStatusType"
minOccurs="0"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliPrRaise_c_BC">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:Event">
      <xsd:sequence>
        <xsd:element name="at" type="Types:TriPortIdType"/>
        <xsd:element name="to" type="Types:TriPortIdListType" minOccurs="0"/>
        <xsd:element name="signature" type="Types:TriSignatureIdType"/>
        <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>
        <xsd:element name="excValue" type="Values:Value" minOccurs="0"/>
        <xsd:element name="transmission-failure" type="SimpleTypes:TriStatusType"
minOccurs="0"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliPrRaise_c_MC">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:Event">
      <xsd:sequence>
        <xsd:element name="at" type="Types:TriPortIdType"/>
        <xsd:element name="to" type="Types:TriPortIdListType" minOccurs="0"/>
        <xsd:element name="signature" type="Types:TriSignatureIdType"/>
        <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>
        <xsd:element name="excValue" type="Values:Value" minOccurs="0"/>
        <xsd:element name="transmission-failure" type="SimpleTypes:TriStatusType"
minOccurs="0"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliPrCatchDetected_m">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:Event">
      <xsd:sequence>
        <xsd:element name="at" type="Types:TriPortIdType"/>
        <xsd:element name="from" type="Types:TriPortIdType" minOccurs="0"/>
        <xsd:element name="signature" type="Types:TriSignatureIdType"/>
        <xsd:element name="exc" type="Types:TriExceptionType" minOccurs="0"/>
        <xsd:element name="address" type="Types:TriAddressType" minOccurs="0"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliPrCatchDetected_c">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:Event">
      <xsd:sequence>
        <xsd:element name="at" type="Types:TriPortIdType"/>
        <xsd:element name="from" type="Types:TriPortIdType" minOccurs="0"/>
        <xsd:element name="signature" type="Types:TriSignatureIdType"/>
        <xsd:element name="excValue" type="Values:Value" minOccurs="0"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliPrCatchMismatch_m">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:Event">

```

```

        <xsd:sequence>
          <xsd:element name="at" type="Types:TriPortIdType"/>
          <xsd:element name="signature" type="Types:TriSignatureIdType"/>
          <xsd:element name="excValue" type="Values:Value" minOccurs="0"/>
          <xsd:element name="excTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>
          <xsd:element name="diffs" type="Templates:TciValueDifferenceList"/>
          <xsd:element name="addrValue" type="Values:Value" minOccurs="0"/>
          <xsd:element name="addressTmpl" type="Templates:TciValueTemplate"
minOccurs="0"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="tliPrCatchMismatch_c">
    <xsd:complexContent mixed="true">
      <xsd:extension base="Events:Event">
        <xsd:sequence>
          <xsd:element name="at" type="Types:TriPortIdType"/>
          <xsd:element name="signature" type="Types:TriSignatureIdType"/>
          <xsd:element name="excValue" type="Values:Value" minOccurs="0"/>
          <xsd:element name="excTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>
          <xsd:element name="diffs" type="Templates:TciValueDifferenceList"/>
          <xsd:element name="from" type="Types:TriComponentIdType" minOccurs="0"/>
          <xsd:element name="fromTmpl" type="Templates:TciNonValueTemplate"
minOccurs="0"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="tliPrCatch_m">
    <xsd:complexContent mixed="true">
      <xsd:extension base="Events:Event">
        <xsd:sequence>
          <xsd:element name="at" type="Types:TriPortIdType"/>
          <xsd:element name="signature" type="Types:TriSignatureIdType"/>
          <xsd:element name="excValue" type="Values:Value"/>
          <xsd:element name="excTmpl" type="Templates:TciValueTemplate"/>
          <xsd:element name="addrValue" type="Values:Value" minOccurs="0"/>
          <xsd:element name="addressTmpl" type="Templates:TciValueTemplate"
minOccurs="0"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="tliPrCatch_c">
    <xsd:complexContent mixed="true">
      <xsd:extension base="Events:Event">
        <xsd:sequence>
          <xsd:element name="at" type="Types:TriPortIdType"/>
          <xsd:element name="signature" type="Types:TriSignatureIdType"/>
          <xsd:element name="excValue" type="Values:Value" minOccurs="0"/>
          <xsd:element name="excTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>
          <xsd:element name="from" type="Types:TriComponentIdType" minOccurs="0"/>
          <xsd:element name="fromTmpl" type="Templates:TciNonValueTemplate"
minOccurs="0"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="tliPrCatchTimeoutDetected">
    <xsd:complexContent mixed="true">
      <xsd:extension base="Events:Event">
        <xsd:sequence>
          <xsd:element name="at" type="Types:TriPortIdType"/>
          <xsd:element name="signature" type="Types:TriSignatureIdType"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="tliPrCatchTimeout">
    <xsd:complexContent mixed="true">
      <xsd:extension base="Events:Event">
        <xsd:sequence>
          <xsd:element name="at" type="Types:TriPortIdType"/>
          <xsd:element name="signature" type="Types:TriSignatureIdType"/>

```

```

        </xsd:sequence>
    </xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<!-- components -->
<xsd:complexType name="tliCCreate">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="comp" type="Types:TriComponentIdType"/>
                <xsd:element name="name" type="SimpleTypes:TString"/>
                <xsd:element name="hostId" type="Values:Value" minOccurs="0"/>
                <xsd:element name="alive" type="SimpleTypes:TBoolean"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliCStart">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="comp" type="Types:TriComponentIdType"/>
                <xsd:element name="name" type="Types:TciBehaviourIdType"/>
                <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliCRunning">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="comp" type="Types:TriComponentIdType"/>
                <xsd:element name="status" type="SimpleTypes:ComponentStatusType"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliCAlive">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="comp" type="Types:TriComponentIdType"/>
                <xsd:element name="status" type="SimpleTypes:ComponentStatusType"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliCStop">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="comp" type="Types:TriComponentIdType"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliCKill">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="comp" type="Types:TriComponentIdType"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliCDoneMismatch">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="comp" type="Types:TriComponentIdType"/>
                <xsd:element name="compTpl1" type="Templates:TciNonValueTemplate"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

```

```

        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="tliCKilledMismatch">
    <xsd:complexContent mixed="true">
      <xsd:extension base="Events:Event">
        <xsd:sequence>
          <xsd:element name="comp" type="Types:TriComponentIdType"/>
          <xsd:element name="compTpl" type="Templates:TciNonValueTemplate"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="tliCDone">
    <xsd:complexContent mixed="true">
      <xsd:extension base="Events:Event">
        <xsd:sequence>
          <xsd:element name="compTpl" type="Templates:TciNonValueTemplate"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="tliCKilled">
    <xsd:complexContent mixed="true">
      <xsd:extension base="Events:Event">
        <xsd:sequence>
          <xsd:element name="compTpl" type="Templates:TciNonValueTemplate"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="tliCTerminated">
    <xsd:complexContent mixed="true">
      <xsd:extension base="Events:Event">
        <xsd:sequence>
          <xsd:element name="verdict" type="Values:VerdictValue" />
          <xsd:element name="reason" type="SimpleTypes:TString" minOccurs="0"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <!-- ports -->
  <xsd:complexType name="tliPConnect">
    <xsd:complexContent mixed="true">
      <xsd:extension base="Events:PortConfiguration"/>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="tliPDisconnect">
    <xsd:complexContent mixed="true">
      <xsd:extension base="Events:PortConfiguration"/>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="tliPMap">
    <xsd:complexContent mixed="true">
      <xsd:extension base="Events:PortConfiguration"/>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="tliPMapParam">
    <xsd:complexContent mixed="true">
      <xsd:extension base="Events:tliPMap">
        <xsd:sequence>
          <xsd:element name="tciPars" type="Types:TciParameterListType" />
          <xsd:choice>
            <xsd:element name="encoder-failure" type="SimpleTypes:TciStatusType"
minOccurs="0"/>
            <xsd:element name="triPars" type="Types:TriParameterListType" />
          </xsd:choice>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

```

```

</xsd:complexType>

<xsd:complexType name="tliPUnmap">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:PortConfiguration"/>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliPUnmapParam">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:tliPUnmap">
      <xsd:sequence>
        <xsd:element name="tciPars" type="Types:TciParameterListType" />
        <xsd:choice>
          <xsd:element name="encoder-failure" type="SimpleTypes:TciStatusType"
minOccurs="0"/>
          <xsd:element name="triPars" type="Types:TriParameterListType"/>
        </xsd:choice>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliPClear">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:PortStatus"/>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliPStart">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:PortStatus"/>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliPStop">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:PortStatus"/>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliPHalt">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:PortStatus"/>
  </xsd:complexContent>
</xsd:complexType>

<!-- codec -->
<xsd:complexType name="tliEncode">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:Event">
      <xsd:sequence>
        <xsd:element name="val" type="Values:Value"/>
        <xsd:choice>
          <xsd:element name="msg" type="Types:TriMessageType"/>
          <xsd:element name="encoder-failure" type="SimpleTypes:TciStatusType"
minOccurs="0"/>
        </xsd:choice>
        <xsd:element name="codec" type="SimpleTypes:TString"
minOccurs="0"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliDecode" mixed="true">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:Event">
      <xsd:sequence>
        <xsd:element name="msg" type="Types:TriMessageType"/>
        <xsd:choice>
          <xsd:element name="decoder-failure" type="SimpleTypes:TciStatusType"
minOccurs="0"/>
          <xsd:element name="val" type="Values:Value"/>
        </xsd:choice>
        <xsd:element name="codec" type="SimpleTypes:TString"
minOccurs="0"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>

```

```

</xsd:complexType>

<!-- timers -->
<xsd:complexType name="tliTimeoutDetected">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:Event">
      <xsd:sequence>
        <xsd:element name="timer" type="Types:TriTimerIdType" />
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliTimeoutMismatch">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:Event">
      <xsd:sequence>
        <xsd:element name="timer" type="Types:TriTimerIdType" />
        <xsd:element name="timerTpl" type="Templates:TciNonValueTemplate" />
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliTimeout">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:Event">
      <xsd:sequence>
        <xsd:element name="timer" type="Types:TriTimerIdType" />
        <xsd:element name="timerTpl" type="Templates:TciNonValueTemplate" />
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliTStart">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:Event">
      <xsd:sequence>
        <xsd:element name="timer" type="Types:TriTimerIdType"/>
        <xsd:element name="dur" type="SimpleTypes:TriTimerDurationType"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliTStop">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:Event">
      <xsd:sequence>
        <xsd:element name="timer" type="Types:TriTimerIdType"/>
        <xsd:element name="dur" type="SimpleTypes:TriTimerDurationType"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliTRead">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:Event">
      <xsd:sequence>
        <xsd:element name="timer" type="Types:TriTimerIdType"/>
        <xsd:element name="elapsed" type="SimpleTypes:TriTimerDurationType"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliTRunning">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:Event">
      <xsd:sequence>
        <xsd:element name="timer" type="Types:TriTimerIdType"/>
      </xsd:sequence>
      <xsd:attribute name="status" type="SimpleTypes:TimerStatusType"/>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<!-- scope -->

```



```

<xsd:complexType name="tliSEnter">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:Event">
      <xsd:sequence>
        <xsd:element name="name" type="Types:QualifiedName" />
        <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>
        <xsd:element name="kind" type="SimpleTypes:TString"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliSLeave">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:Event">
      <xsd:sequence>
        <xsd:element name="name" type="Types:QualifiedName" />
        <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>
        <xsd:element name="returnValue" type="Values:Value" minOccurs="0"/>
        <xsd:element name="kind" type="SimpleTypes:TString"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<!-- variables and module parameter -->
<xsd:complexType name="tliVar">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:Event">
      <xsd:sequence>
        <xsd:element name="name" type="Types:QualifiedName" />
        <xsd:element name="val" type="Values:Value" minOccurs="0"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliModulePar">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:Event">
      <xsd:sequence>
        <xsd:element name="name" type="Types:QualifiedName" />
        <xsd:element name="val" type="Values:Value" minOccurs="0"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<!-- verdicts -->
<xsd:complexType name="tliGetVerdict">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:Event">
      <xsd:sequence>
        <xsd:element name="verdict" type="Values:VerdictValue"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliSetVerdict">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:Event">
      <xsd:sequence>
        <xsd:element name="verdict" type="Values:VerdictValue"/>
        <xsd:element name="reason" type="SimpleTypes:TString" minOccurs="0"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<!-- log -->
<xsd:complexType name="tliLog">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:Event">
      <xsd:sequence>
        <xsd:element name="log" type="SimpleTypes:TString"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

```

```

<!-- alt -->
<xsd:complexType name="tliAEnter">
  <xsd:complexContent>
    <xsd:extension base="Events:Event"/>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliALeave">
  <xsd:complexContent>
    <xsd:extension base="Events:Event"/>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliADefaults">
  <xsd:complexContent>
    <xsd:extension base="Events:Event"/>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliAActivate">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:Event">
      <xsd:sequence>
        <xsd:element name="name" type="Types:QualifiedName" />
        <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>
        <xsd:element name="ref" type="Values:Value"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliADeactivate">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:Event">
      <xsd:sequence>
        <xsd:element name="ref" type="Values:Value"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliANomatch">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:Event"/>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliARepeat">
  <xsd:complexContent>
    <xsd:extension base="Events:Event"/>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliAWait">
  <xsd:complexContent>
    <xsd:extension base="Events:Event"/>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliAction">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:Event">
      <xsd:sequence>
        <xsd:element name="action" type="SimpleTypes:TString"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliMatch">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:Event">
      <xsd:sequence>
        <xsd:element name="expr" type="Values:Value"/>
        <xsd:element name="tpl" type="Templates:TciValueTemplate"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

```

```

<xsd:complexType name="tliMatchMismatch">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:Event">
      <xsd:sequence>
        <xsd:element name="expr" type="Values:Value"/>
        <xsd:element name="tmpl" type="Templates:TciValueTemplate"/>
        <xsd:element name="diffs" type="Templates:TciValueDifferenceList"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliInfo">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:Event">
      <xsd:sequence>
        <xsd:element name="level" type="SimpleTypes:TInteger"/>
        <xsd:element name="info" type="SimpleTypes:TString"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

```

## 12 C# mapping

### 12.1 Introduction

The C# mapping for the TTCN-3 control interface defines how the IDL [6] definitions described in clause 7 are mapped to the .Net language C# [13].

### 12.2 Names and scopes

#### 12.2.1 Names

Although there are almost no conflicts between identifiers used in the IDL definition and C#, some naming translation rules are applied to the IDL identifiers.

C# interfaces are omitting the trailing Type used in the IDL definition. In addition to that, the capital letter "I" is added to the beginning of interface names.

EXAMPLE 1 – The IDL type `TciTestCaseIdType` maps to `ITciTestCaseId` in C#.

C# names of enumerated items start with a capital letter and the remaining letters are low-case letters. If the enumerated item name is composed of several words, each word starts with a capital letter.

EXAMPLE 2 – The identifier for boolean type defined in `TciTypeClassType` enumeration is `BooleanType` in C#.

The resulting mapping conforms to the standard C# coding conventions.

#### 12.2.2 Scopes

The TCI interfaces are mapped to the namespace `Etsi.Ttcn3.Tci`. All IDL type declarations are mapped to C# interface declarations within this namespace. The associated assembly file is `Etsi.Ttcn3.Tci.dll`.

### 12.3 Null value mapping

The distinct value `null` specified in the IDL definition is equal to `null` in C#.

### 12.4 Type mapping

#### 12.4.1 Basic type mapping

Table 5 gives an overview on how the used basic IDL types are mapped to the .NET types.

**Table 5 – Basic type mapping**

<b>IDL Type</b>	<b>C# Type/Interface</b>
TBoolean	bool
TChar	char
TFloat	double
TInteger	int/TciVerdict
TString	string
TStringSeq	string[]
TUniversalChar	uint

### **TBoolean**

The IDL `TBoolean` type is mapped to the C# type `bool`.

### **TFloat**

The IDL `TFloat` type is mapped to the C# type `double`.

### **TChar**

The IDL `TChar` type is mapped to the C# type `char`.

### **TInteger**

The IDL `TInteger` type is usually mapped to the C# type `int`. Only in case of operations defined for the IDL type `TciVerdictValue`, the IDL `TInteger` type is mapped to `Etsi.Ttcn3.Tci.TciVerdict` enumeration.

### **TString**

The IDL `TString` type is mapped to the C# class `string` without range checking or bounds for characters in the string. All possible strings defined in TTCN-3 can be converted to C# `string` class.

### **TStringSeq**

The IDL `TStringSeq` type is mapped to a string array.

### **TUniversalChar**

The IDL `TUniversalChar` type is mapped to the C# type `uint`. The integer uses the canonical form as defined in ISO/IEC 10646 [5], clause 6.2.

#### **12.4.1.1 TciVerdict**

In case of verdict operations, the IDL `TInteger` type is mapped to the `tciverdict` enumeration. This enumeration is defined as follows:

```
public enum TciVerdict {  
    None = 0,  
    Pass = 1,  
    Inconc = 2,  
    Fail = 3,  
    Error = 4  
    User_Error = 5  
}
```

## 12.4.2 Structured type mapping

The TCI IDL description defines user-defined types as native types. In the C# mapping, these types are mapped to C# interfaces. The interfaces define methods and properties being available for classes implementing this interface.

### 12.4.2.1 TciParameterPassingModeType

**TciParameterPassingModeType** is mapped to the following enumeration:

```
public enum TciParameterPassingMode {
    TciIn = 0,
    TciInOut = 1,
    TciOut = 2
}
```

### 12.4.2.2 TciParameterType

**TciParameterType** is mapped to the following interface:

```
public interface ITciParameter {
    string ParameterName { get; set; }
    TciParameterPassingMode ParameterPassingMode { get; set; }
    ITciValue Parameter { get; set; }
}
```

#### Members:

- **ParameterName**  
Gets or sets the parameter name.
- **ParameterPassingMode**  
Gets or sets the parameter passing mode of this parameter.
- **Parameter**  
Used for getting or setting value of the parameter. The parameter can be an instance of **ITciValue** or the distinct value **null**.

### 12.4.2.3 TciParameterListType

**TciParameterListType** is mapped to the following interface:

```
public interface ITciParameterList: System.Collections.IEnumerable {
    int Size { get; }
    bool IsEmpty { get; }
    ITciParameter this[int index] { get; }
    void Clear();
    void Add(ITciParameter comp);
}
```

#### Members:

- **Size**  
Returns the number of parameters in this list.
- **IsEmpty**  
Returns **true** if this list contains no parameters.
- **GetEnumerator**  
Inherited from **IEnumerable**. Returns an enumerator for this object and allows to use the list in a **foreach** loop.
- **operator**  
Returns a **ITciParameter** instance at the specified position. **IndexOutOfRangeException** is thrown if the index is less than zero or greater or equal to the list size.

- Clear  
Removes all parameters from the list.
- Add  
Adds a parameter to the end of the list.

#### 12.4.2.4 TciTypeClassType

**TciTypeClassType** is mapped to the following enumeration:

```
public enum TciTypeClass {
    Address = 0,
    Anytype = 1,
    Bitstring = 2,
    BooleanType = 3,
    Charstring = 5,
    Component = 6,
    Enumerated = 7,
    Float = 8,
    Hexstring = 9,
    IntegerType = 10,
    Octetstring = 12,
    Record = 13,
    RecordOf = 14,
    Array = 15
    Set = 16,
    SetOf = 17,
    Union = 18,
    UniversalCharstring = 20,
    Verdict = 21
}
```

#### 12.4.2.5 TciTestComponentKindType

**TciTestComponentKindType** is mapped to the following enumeration:

```
public enum TciTestComponentKind {
    TciCtrlComp = 0,
    TciMtcComp = 1,
    TciPtcComp = 2,
    TciSystemComp = 3,
    TciAliveComp = 4
}
```

#### 12.4.2.6 TciBehaviourIdType

**TciBehaviourIdType** C# mapping is derived from the Etsi.Ttcn3.Tri.IQualifiedName interface:

```
public interface ITciBehaviourId : Etsi.Ttcn3.Tri.IQualifiedName {}
```

#### 12.4.2.7 TciTestCaseIdType

**TciTestCaseIdType** C# mapping is derived from the Etsi.Ttcn3.Tri.IQualifiedName interface:

```
public interface ITciTestCaseId : Etsi.Ttcn3.Tri.IQualifiedName {}
```

#### 12.4.2.8 TciTestCaseIdListType

**TciTestCaseIdListType** is mapped to the following interface:

```
public interface ITciTestCaseIdList: System.Collections.IEnumerable {
    int Size { get; }
    bool IsEmpty { get; }
    ITciTestCaseId this[int index] { get; }
}
```

## Members:

- `Size`  
Returns the number of test case identifiers in this list.
- `IsEmpty`  
Returns true if this list contains no parameters.
- `GetEnumerator`  
Inherited from `IEnumerable`. Returns an enumerator for this object and allows to use the list in a foreach loop.
- `operator`  
Returns a `ITciTestCaseId` instance at the specified position. `IndexOutOfRangeException` is thrown if the index is less than zero or greater or equal to the list size.

### 12.4.2.9 TciModuleIdType

**TciModuleIdType** C# mapping is derived from the `Etsi.Ttcn3.Tri.IQualifiedName` interface:

```
public interface ITciModuleId : Etsi.Ttcn3.Tri.IQualifiedName {  
}
```

### 12.4.2.10 TciModuleIdListType

**TciModuleIdListType** is mapped to the following interface:

```
public interface ITciModuleIdList : System.Collections.IEnumerable {  
    int Size { get; }  
    bool IsEmpty { get; }  
    ITciModuleId this[int index] { get; }  
}
```

## Members:

- `Size`  
Returns the number of module identifiers in this list.
- `IsEmpty`  
Returns true if this list contains no parameters.
- `GetEnumerator`  
Inherited from `IEnumerable`. Returns an enumerator for this object and allows to use the list in a foreach loop.
- `operator`  
Returns a `ITciModuleId` instance at the specified position. `IndexOutOfRangeException` is thrown if the index is less than zero or greater or equal to the list size.

### 12.4.2.11 TciModuleParameterIdType

**TciModuleIdType** C# mapping is derived from the `Etsi.Ttcn3.Tri.IQualifiedName` interface:

```
public interface ITciModuleParameterId : Etsi.Ttcn3.Tri.IQualifiedName {  
}
```

### 12.4.2.12 TciModuleParameterType

**TciModuleParameterType** is mapped to the following interface:

```
public interface ITciModuleParameter {  
    ITciModuleParameterId ModuleParameterName { get; }  
}
```

```

    ITciValue DefaultValue { get; }
}

```

### Members:

- `ModuleParameterName`  
Returns the qualified module parameter name as defined in the TTCN-3 specification.
- `DefaultValue`  
Returns the default value of this `TciModuleParameter` or the distinct value `null` if the default value is not specified.

### 12.4.2.13 TciModuleParameterListType

`TciModuleParameterListType` is mapped to the following interface:

```

public interface ITciModuleParameterList: System.Collections.IEnumerable {
    int Size { get; }
    bool IsEmpty { get; }
    ITciModuleParameter this[int index] { get; }
}

```

### Members:

- `Size`  
Returns the number of module identifiers in this list.
- `IsEmpty`  
Returns true if this list contains no parameters.
- `GetEnumerator()`  
Inherited from `IEnumerable`. Returns an enumerator for this object and allows to use the list in a `foreach` loop.
- Indexing operator  
Returns a `ITciModuleId` instance at the specified position. `IndexOutOfRangeException` is thrown if the index is less than zero or greater or equal to the list size.

### 12.4.2.14 TciParameterTypeType

`TciParameterTypeType` is mapped to the following interface:

```

public interface ITciParameterType {
    ITciType ParameterType { get; }
    TciParameterPassingMode ParameterPassingMode { get; }
}

```

### Members:

- `ParameterType`  
Returns the type of the parameter.
- `ParameterPassingMode`  
Returns the passing mode of this parameter.

### 12.4.2.15 TciParameterTypeListType

`TciParameterListType` is mapped to the following interface:

```

public interface ITciParameterTypeList: System.Collections.IEnumerable {
    int Size { get; }
    bool IsEmpty { get; }
    ITciParameterType this[int index] { get; }
}

```



## Members:

- `Size`  
Returns the number of parameters in this list.
- `IsEmpty`  
Returns true if this list contains no parameters.
- `GetEnumerator`  
Inherited from `IEnumerable`. Returns an enumerator for this object and allows to use the list in a foreach loop.
- `Indexing operator`  
Returns a `ITciParameter` instance at the specified position. `IndexOutOfRangeException` is thrown if the index is less than zero or greater or equal to the list size.
- `Clear`  
Removes all parameters from the list.
- `Add`  
Adds a parameter to the end of the list.

### 12.4.3 Abstract type mapping

The TTCN-3 data types are modelled in C# using the abstract type mapping as defined in this clause. The `ITciType` interface defines only operations used to retrieve in TTCN-3 defined types. No TTCN-3 types can be constructed using the `ITciType` interface. Types are modelled using the single interface `ITciType`, that provides methods to identify types and to retrieve values of a given type.

#### 12.4.3.1 Type

The IDL type `type` is mapped to the following interface:

```
public interface ITciType {
    ITciModuleId DefiningModule { get; }
    string Name { get; }
    TciTypeClass TypeClass { get; }
    ITciValue NewInstance();
    string TypeEncoding { get; }
    string TypeEncodingVariant { get; }
    string[] TypeExtension { get; }
}
```

## Members:

- `DefiningModule`  
Returns the module identifier of the module the type has been defined in. If the type represents a TTCN-3 base type the distinct value null will be returned.
- `Name`  
Returns name of the type as defined in the TTCN-3 module.
- `TypeClass`  
Returns the type class of the respective type.
- `NewInstance`  
Returns a freshly created value of the given type. This initial value of the created value is undefined.

- `TypeEncoding`  
Returns the type encoding attribute as defined in the TTCN-3 module, if any. If no encoding attribute has been defined, the distinct value `null` will be returned.
- `TypeEncodingVariant`  
This property returns the type encoding variant attribute as defined in TTCN-3, if any. If no encoding variant attribute has been defined, the distinct value `null` will be returned.
- `TypeExtension`  
Returns the type extension attributes as defined in the TTCN-3 module. If no extension attributes have been defined, the distinct value `null` will be returned.

#### 12.4.4 Abstract value mapping

TTCN-3 values can be retrieved from the TE and constructed using the `ITciValue` interface. The value mapping interface is constructed hierarchically with `ITciValue` as the basic interface. Specialized interfaces for different types of values have been defined.

##### 12.4.4.1 Value

The IDL type `value` is mapped to the following interface:

```
public interface ITciValue {
    ITciType Type { get; }
    bool NotPresent { get; }
    string ValueEncoding { get; }
    string ValueEncodingVariant { get; }
}
```

##### Members:

- `Type`  
Returns the type of the specified value.
- `NotPresent`  
Returns `true` if the specified value is omit, false otherwise.
- `ValueEncoding`  
This property returns the value encoding attribute as defined in TTCN-3, if any. If no encoding attribute has been defined the distinct value `null` will be returned.
- `ValueEncodingVariant`  
This property returns the value encoding variant attribute as defined in TTCN-3, if any. If no encoding variant attribute has been defined the distinct value `null` will be returned.

##### 12.4.4.2 IntegerValue

`IntegerValue` is mapped to the following interface:

```
public interface ITciIntegerValue : ITciValue {
    long IntegerValue { get; set; }
    string StringValue { get; set; }
}
```

##### Members:

- `IntegerValue`  
Gets or sets the numeric value of the object. In case the numeric value exceeds the allowed value range of the `long` type, `long.MaxValue` or `long.MinValue` is returned.
- `StringValue`

Get or sets the value of the object. The string assigned to the property shall have the same format as TTCN-3 integer literals. The integer literal can be optionally preceded by a sign character ('+' or '-').

#### 12.4.4.3 FloatValue

**FloatValue** is mapped to the following interface:

```
public interface ITciFloatValue : ITciValue {
    double FloatValue { get; set; }
    string StringValue { get; set; }
}
```

##### Members:

- **FloatValue**  
Gets or sets the numeric value of the object. In case the numeric value exceeds the allowed value range of the `double` type, `double.MaxValue` or `double.MinValue` is returned.
- **StringValue**  
Get or sets the value of the object. The string assigned to the property shall have the same format as TTCN-3 float literals. The float literal can be optionally preceded by a sign character ('+' or '-').

#### 12.4.4.4 BooleanValue

**BooleanValue** is mapped to the following interface:

```
public interface ITciBooleanValue : ITciValue {
    bool BooleanValue { get; set; }
}
```

##### Members:

- **BooleanValue**  
Gets or sets the boolean value of the object.

#### 12.4.4.5 CharstringValue

**CharstringValue** is mapped to the following interface:

```
public interface ITciCharstringValue : ITciValue {
    string StringValue { get; set; }
    char this[int position] { get; set; }
    int Length { get; set; }
}
```

##### Members:

- **StringValue**  
Gets or sets the string value of the TTCN-3 charstring. Strings assigned to this property shall contain only characters allowed in TTCN-3 charstring type.
- **Indexing operator**  
Get or sets the character value of the TTCN-3 charstring at the specified position. `IndexOutOfRangeException` is thrown if the position is less than zero or greater or equal to the string length.
- **Length**  
Gets or sets the length of this `ITciCharstringValue` in characters. The property returns zero if the value of this object is `omit`. In case the new length is greater than the length of the current string, characters with ordinal value 0 are added to the end of the string. If the new length is less than the length of the current string, the current string is truncated.

#### 12.4.4.6 BitstringValue

**BitstringValue** is mapped to the following interface:

```
public interface ITciBitstringValue : ITciValue {
    string StringValue { get; set; }
    byte this[int position] { get; set; }
    int Length { get; set; }
}
```

##### Members:

- **StringValue**  
Gets or sets the string value of the TTCN-3 bitstring. The only allowed characters in the string passed to this property are '0' and '1'. The string returned by the property contains a sequence of '0' and '1' digits.
- **Indexing operator**  
Get or sets the value of the bit at the specified position. All non-zero values shall be interpreted as if the bit was present. `IndexOutOfRangeException` is thrown if the position is less than zero or greater or equal to the string length.
- **Length**  
Gets or sets the length of this `ITciBitstringValue` in bits. The property returns zero if the value of this object is `omit`. In case the new length is greater than the length of the current bitstring, the bitstring is padded with empty bits. If the new length is less than the length of the current bitstring, the current bitstring is truncated.

#### 12.4.4.7 OctetstringValue

**OctetstringValue** is mapped to the following interface:

```
public interface ITciOctetstringValue : ITciValue {
    string StringValue { get; set; }
    byte this[int position] { get; set; }
    int Length { get; set; }
}
```

##### Members:

- **StringValue**  
Gets or sets the string value of the TTCN-3 octetstring. The only allowed characters in the string passed to this property are hexadecimal digits. The length of the string passed to this property shall be even. The string returned by this property is a sequence of pairs of hexadecimal digits.
- **Indexing operator**  
Get or sets the value of the octet at the specified position. `IndexOutOfRangeException` is thrown if the position is less than zero or greater or equal to the string length.
- **Length**  
Gets or sets the length of this `ITciOctetstringValue` in octets. The property returns zero if the value of this object is `omit`. In case the new length is greater than the length of the current octetstring, the octetstring is padded with empty octets. If the new length is less than the length of the current octetstring, the current octetstring is truncated.

#### 12.4.4.8 UniversalCharstringValue

**UniversalCharstringValue** is mapped to the following interface:

```
public interface ITciUniversalCharstringValue : ITciValue {
    string StringValue { get; set; }
}
```

```

uint this[int position] { get; set; }
int Length { get; set; }
}

```

### Members:

- **StringValue**  
Gets or sets the string value of the TTCN-3 universal charstring. If the TTCN-3 universal charstring value contains characters that have higher ordinal value than `char.MaxValue`, these characters will be represented by a character `0xFFFFD` (the Unicode replacement character) in the string returned by this property.
- **Indexing operator**  
Get or sets the character value of the TTCN-3 universal charstring at the specified position. The unsigned number used by this property is character ordinal value. `IndexOutOfRangeException` is thrown if the position is less than zero or greater or equal to the string length.
- **Length**  
Gets or sets the length of this `ITciUniversalCharstringValue` in characters. The property returns zero if the value of this object is omit. In case the new length is greater than the length of the current string, characters with ordinal value 0 are added to the end of the string. If the new length is less than the length of the current string, the current string is truncated.

#### 12.4.4.9 HexstringValue

**HexstringValue** is mapped to the following interface:

```

public interface ITciHexstringValue : ITciValue {
    string StringValue { get; set; }
    byte this[int position] { get; set; }
    int Length { get; set; }
}

```

### Members:

- **StringValue**  
Gets or sets the string value of the TTCN-3 hexstring. The only allowed characters in the string passed to this property are hexadecimal digits. The string returned by this property is a sequence of hexadecimal digits.
- **Indexing operator**  
Get or sets the hex digit at the specified position. Only the lower four bits of the passed value are used in this assignment. The upper four bits as ignored. `IndexOutOfRangeException` is thrown if the position is less than zero or greater or equal to the string length.
- **Length**  
Gets or sets the length of this `ITciHexstringValue` in hex digits. The property returns zero if the value of this object is omit. In case the new length is greater than the length of the current hexstring, the hexstring is padded with zeroes. If the new length is less than the length of the current hexstring, the current hexstring is truncated.

#### 12.4.4.10 RecordValue

**RecordValue** is mapped to the following interface:

```

public interface ITciRecordValue : ITciValue {
    ITciValue GetField(string fieldName);
    void SetField(string fieldName, ITciValue value);
    string[] GetFieldNames();
}

```

```

    void SetFieldOmitted(string fieldName);
}

```

### Members:

- `GetField`  
Returns the value of the field named `fieldName`. The return value is the common abstract base type `ITciValue`, as a record field can have any type defined in TTCN-3. If the field cannot be obtained from the record the distinct value `null` will be returned.
- `SetField`  
Sets the field named `fieldName` of the record to value. No assumption shall be made on how a field is stored in a record. An internal implementation might choose to store a reference to this value or to copy the value. It is safe to assume that the value will be copied. Therefore it should be assumed that subsequent modifications of value will not be considered in the record.
- `GetFieldNames`  
Returns an array of `String` of field names, the empty sequence, if the record has no fields.
- `SetFieldOmitted`  
Sets the field named `fieldName` of the record to omit.

#### 12.4.4.11 RecordOfValue

`RecordOfValue` is mapped to the following interface:

```

public interface ITciRecordOfValue : ITciValue, System.Collections.IEnumerable {
    ITciValue this[int position] { get; set; }
    void AppendField(ITciValue value);
    ITciType ElementType { get; }
    int Length { get; set; }
    int Offset { get; }
}

```

### Members:

- Indexing operator  
Returns or sets the value of the record of at the specified position. The class of this property is the common abstract base interface `ITciValue`, as a record of can have fields of any type defined in TTCN-3. When getting the value an `ITciValue` instance is returned only if `position` is between zero and `(length - 1)`. The distinct value `null` is returned otherwise. When setting the value, if `position` is greater than the current length, the record of will be extended to have the length `(position + 1)`. The record of elements between the original position at `length` and `(position - 1)` will be set to omit. No assumption shall be made on how a field is stored in a `record of`. An internal implementation might choose to store a reference to this value or to copy the value. It is safe to assume that the value will be copied. Therefore it should be assumed that subsequent modifications of value will not be considered in the `record of`.
- `GetEnumerator`  
Inherited from `IEnumerable`. Returns an enumerator for this object and allows to use the object in a `foreach` loop.
- `appendField`  
Appends the value at the end of the `record of`, i.e., at position `length`. No assumption shall be made on how a field is stored in a `record of`. An internal implementation might choose to store a reference to this value or to copy the value. It is safe to assume that the value will

be copied. Therefore it should be assumed that subsequent modifications of `value` will not be considered in the `record` of.

- `ElementType`

Returns the type of the elements of this `record` of.

- `Length`

Gets or sets the actual length of the `record` of `value`. When getting the length, zero is returned if the `record` of `value` is `omit`. When setting the length, if the new length is greater than the original length, newly created elements have the value `omit`. If the new length is less or equal than the original length, this operation will be ignored.

- `Offset`

Returns the lowest possible index. For a `record` of or set of `value` this is always 0. For an array `value`, this is the lower index bound used in the type definition.

#### 12.4.4.12 UnionValue

**UnionValue** is mapped to the following interface:

```
public interface ITciUnionValue : ITciValue {
    ITciValue GetVariant(string variantName);
    void SetVariant(string variantName, ITciValue value);
    string PresentVariantName { get; }
    string[] GetVariantNames();
}
```

#### Members:

- `GetVariant`

Returns the value of the TTCN-3 union variant, if `variantName` equals the result of `GetPresentVariantName`. The distinct value `null` is returned otherwise. `variantName` denotes the name of the union variant as defined in TTCN-3.

- `SetVariant`

Sets `variantName` of the union to `value`. If `variantName` is not defined for this union this operation will be ignored. If another variant was selected the new variant will be selected instead.

- `GetPresentVariantName`

Returns the variant name that has a value in this union set as a `string`. The distinct value `null` will be returned if no variant is selected.

- `GetVariantNames`

Returns an array of `string` of variant names, the empty sequence, if the union has no fields. If the `UnionValue` represents the TTCN-3 `anytype`, i.e., the value of the `Type` property is `TciTypeClass.Anytype`, all predefined and user-defined TTCN-3 types will be returned.

#### 12.4.4.13 EnumeratedValue

**EnumeratedValue** is mapped to the following interface:

```
public interface ITciEnumeratedValue : ITciValue {
    string EnumValue { get; set; }
    int IntValue { get; set; }
}
```

#### Members:

- `EnumValue`

Returns or sets the enumerated value. The value of the property is equal to the identifier in the TTCN-3 specification. If the value assigned to the property is not an allowed value for this enumeration, the assignment will be ignored.

- `IntValue`

Returns or sets the integer value. This integer should equal the user-assigned integer value in the TTCN-3 specification or the automatically assigned integer value. If the integer assigned to the property is not allowed for this enumeration, the assignment will be ignored.

#### 12.4.4.14 VerdictValue

`VerdictValue` is mapped to the following interface:

```
public interface ITciVerdictValue : ITciValue {
    TciVerdict Verdict { get; set; }
}
```

##### Members:

- `Verdict`

Returns the value of this `VerdictValue`. Note that a `VerdictValue` can be set to any of the verdicts defined in the `TciVerdict` enumeration at any time. The `VerdictValue` does not perform any verdict calculations as defined in TTCN-3. For example, it is legal to set the `VerdictValue` first to `TciVerdict.ErrorVerdict` and then to `TciVerdict.Pass`.

#### 12.4.4.15 AddressValue

`AddressValue` is mapped to the following interface:

```
public interface ITciAddressValue : ITciValue {
    ITciValue Address { get; set; }
}
```

##### Members:

- `Address`

Gets or sets the value represented by this `AddressValue`.

### 12.4.5 Abstract logging types mapping

Additional types are defined to ease the logging of object states and matches between values and templates.

#### 12.4.5.1 TciValueTemplate

`TciValueTemplate` is mapped to the following interface:

```
public interface ITciValueTemplate {
    bool IsOmit { get; }
    bool IsAny { get; }
    bool IsAnyOrOmit { get; }
    string TemplateDef { get; }
}
```

##### Members:

- `IsOmit`

Returns `true` if the template is omit, `false` otherwise.

- `IsAny`

Returns `true` if the template is any, `false` otherwise.



- `IsAnyOrOmit`  
Returns `true` if the template is `AnyValueOrNone`, `false` otherwise.
- `TemplateDef`  
This property returns the template definition.

### 12.4.5.2 TciNonValueTemplate

**TciNonValueTemplate** is mapped to the following interface:

```
public interface ITciNonValueTemplate {
    bool IsAny { get; }
    bool IsAll { get; }
    string TemplateDef { get; }
}
```

#### Members:

- `IsAny`  
Returns `true` if the template is any, `false` otherwise.
- `IsAll`  
Returns `true` if the template is all, `false` otherwise.
- `TemplateDef`  
This operation returns the template definition.

### 12.4.5.3 TciValueList

**TciValueList** is mapped to the following interface:

```
public interface ITciValueList: IEnumerable {
    int Size { get; }
    bool IsEmpty { get; }
    ITciValue this[int index] { get; }
}
```

#### Members:

- `Size`  
Returns the number of values in this list.
- `IsEmpty`  
Returns `true` if this list contains no values.
- `GetEnumerator`  
Inherited from `IEnumerable`. Returns an enumerator for this object and allows to use the list in a `foreach` loop.
- Indexing operator  
Returns a `ITciValue` instance at the specified position. `IndexOutOfRangeException` is thrown if the index is less than zero or greater or equal to the list size.

### 12.4.5.4 TciValueDifference

**TciValueDifference** is mapped to the following interface:

```
public interface ITciValueDifference {
    ITciValue Value { get; }
    ITciValueTemplate ValueTemplate { get; }
    string Description { get; }
}
```

## Members:

- Value  
Returns the value of this `ITciValueDifference`.
- Template  
Returns the template of this `ITciValueDifference`.
- Description  
Returns the description of the mismatch.

### 12.4.5.5 TciValueDifferenceList

`TciValueDifferenceList` is mapped to the following interface:

```
public interface ITciValueDifferenceList : IEnumerable {  
    int Size { get; }  
    bool IsEmpty { get; }  
    ITciValueDifference this[int index] { get; }  
}
```

## Members:

- Size  
Returns the number of differences in this list.
- IsEmpty  
Returns `true` if this list contains no parameters.
- GetEnumerator  
Inherited from `IEnumerable`. Returns an enumerator for this object and allows to use the list in a `foreach` loop.
- Indexing operator  
Returns a `ITciValueDifference` instance at the specified position. `IndexOutOfRangeException` is thrown if the index is less than zero or greater or equal to the list size.

### 12.4.5.6 TciStatusType

`TciStatusType` is mapped to the following enumeration:

```
public enum TciStatus {  
    TciOk = 0,  
    TciError = -1  
}
```

### 12.4.5.7 ComponentStatusType

`ComponentStatusType` is mapped to the following enumeration:

```
public enum TciComponentStatus {  
    InactiveC = 0,  
    RunningC = 1,  
    StoppedC = 2,  
    KilledC = 3,  
    NullC = 4  
}
```

### 12.4.5.8 TimerStatusType

`TimerStatusType` is mapped to the following enumeration:

```
public enum TciTimerStatus {  
    RunningT = 0,  
}
```

```

    InactiveT = 1,
    ExpiredT = 2,
    NullT = 3
}

```

## 12.5 Mapping of interfaces

The TCI IDL definition defines four interfaces, the TCI-TM, the TCI-CH, the TCI-CD, and the TCI-TL interface. The operations are defined for different directions within this interface, i.e., some operations can only be called by the TTCN-3 executable (TE), the system adaptor (SA) or the platform adaptor (PA) on the test management and control (TMC) while others can only be called by the TMC on the TE. This is reflected by dividing the TCI IDL interfaces in two sub interfaces, each suffixed by Required or Provided.

**Table 6 – TCI sub-interfaces**

Calling	Called	Interface
TE	TMC	ITciTMProvided
TMC	TE	ITciTMRequired
TE	CD	ITciCDProvided
CD	TE	ITciCDRequired
TE	CH	ITciCHProvided
CH	TE	ITciCHRequired
TE, TMC, CD, CH, SA, PA	TL	ITciTLProvided

All methods defined in this interfaces should behave as defined in clause 7 of [1].

### 12.5.1 TCI-TM interface

#### 12.5.1.1 TCI-TM provided

The **TCI-TM provided** interface is mapped to the following interface:

```

public interface ITciTMProvided {
    void TciTestCaseStarted(ITciTestCaseId testCaseId,
        ITciParameterList parameterList, double timer);
    void TciTestCaseTerminated(ITciVerdictValue verdict,
        ITciParameterList parameterList);
    void TciControlTerminated();
    ITciValue TciGetModulePar(ITciModuleParameterId parameterId);
    void TciLog(Etsi.Ttcn3.Tri.ITriComponentId testComponentId,
        string message);
    void TciError(string message);
}

```

#### 12.5.1.2 TCI-TM required

The **TCI-TM required** interface is mapped to the following interface:

```

public interface ITciTMRequired {
    void TciRootModule(ITciModuleId moduleId);
    ITciModuleIdList TciGetImportedModules();
    ITciModuleParameterList TciGetModuleParameters(ITciModuleId moduleId);
    ITciTestCaseIdList TciGetTestCases();
    ITciParameterTypeList TciGetTestCaseParameters(ITciTestCaseId testCaseId);
    Etsi.Ttcn3.Tri.ITriPortIdList TciGetTestCaseTsi(
        ITciTestCaseId testCaseId);
    void TciStartTestCase(ITciTestCaseId testCaseId,
        ITciParameterList parameterList);
    void TciStopTestCase();
    Etsi.Ttcn3.Tri.ITriComponentId TciStartControl();
    void TciStopControl();
}

```

## 12.5.2 TCI-CD interface

### 12.5.2.1 TCI-CD provided

The **TCI-CD provided** interface is mapped to the following interface:

```
public interface ITciCDProvided {
    ITciValue Decode(Etsi.Ttcn3.Tri.ITriMessage message,
        ITciType decodingHypothesis);
    Etsi.Ttcn3.Tri.ITriMessage Encode(ITciValue value);
}
```

### 12.5.2.2 TCI-CD required

The **TCI-CD required** interface is mapped to the following interface:

```
public interface ITciCDRequired {
    ITciType GetTypeForName(string typeName);
    ITciType GetInteger();
    ITciType GetFloat();
    ITciType GetBoolean();
    ITciType GetCharstring();
    ITciType GetUniversalCharstring();
    ITciType GetHexstring();
    ITciType GetBitstring();
    ITciType GetOctetstring();
    ITciType GetVerdict();
    void TciErrorReq(string message);
}
```

## 12.5.3 TCI-CH interface

### 12.5.3.1 TCI-CH provided

The **TCI-CH provided** interface is mapped to the following interface:

```
public interface ITciCHProvided {
    void TciSendConnected(Etsi.Ttcn3.Tri.ITriPortId sender,
        Etsi.Ttcn3.Tri.ITriComponentId receiver, ITciValue sendMessage);
    void TciSendConnectedBC(Etsi.Ttcn3.Tri.ITriPortId sender,
        ITciValue sendMessage);
    void TciSendConnectedMC(Etsi.Ttcn3.Tri.ITriPortId sender,
        Etsi.Ttcn3.Tri.ITriComponentIdList receivers,
        ITciValue sendMessage);
    void TciCallConnected(Etsi.Ttcn3.Tri.ITriPortId sender,
        Etsi.Ttcn3.Tri.ITriComponentId receiver,
        Etsi.Ttcn3.Tri.ITriSignatureId signature,
        ITciParameterList parameterList);
    void TciCallConnectedBC(Etsi.Ttcn3.Tri.ITriPortId sender,
        Etsi.Ttcn3.Tri.ITriSignatureId signature,
        ITciParameterList parameterList);
    void TciCallConnectedMC(Etsi.Ttcn3.Tri.ITriPortId sender,
        Etsi.Ttcn3.Tri.ITriComponentIdList receivers,
        Etsi.Ttcn3.Tri.ITriSignatureId signature,
        ITciParameterList parameterList);
    void TciReplyConnected(Etsi.Ttcn3.Tri.ITriPortId sender,
        Etsi.Ttcn3.Tri.ITriComponentId receiver,
        Etsi.Ttcn3.Tri.ITriSignatureId signature,
        ITciParameterList parameterList, ITciValue returnValue);
    void TciReplyConnectedBC(Etsi.Ttcn3.Tri.ITriPortId sender,
        Etsi.Ttcn3.Tri.ITriSignatureId signature,
        ITciParameterList parameterList, ITciValue returnValue);
    void TciReplyConnectedMC(Etsi.Ttcn3.Tri.ITriPortId sender,
        Etsi.Ttcn3.Tri.ITriComponentIdList receivers,
        Etsi.Ttcn3.Tri.ITriSignatureId signature,
        ITciParameterList parameterList, ITciValue returnValue);
    void TciRaiseConnected(Etsi.Ttcn3.Tri.ITriPortId sender,
        Etsi.Ttcn3.Tri.ITriComponentId receiver,
        Etsi.Ttcn3.Tri.ITriSignatureId signature, ITciValue except);
    void TciRaiseConnectedBC(Etsi.Ttcn3.Tri.ITriPortId sender,
        Etsi.Ttcn3.Tri.ITriSignatureId signature, ITciValue except);
    void TciRaiseConnectedMC(Etsi.Ttcn3.Tri.ITriPortId sender,
        Etsi.Ttcn3.Tri.ITriComponentIdList receivers,
        Etsi.Ttcn3.Tri.ITriSignatureId signature, ITciValue except);
    Etsi.Ttcn3.Tri.ITriComponentId TciCreateTestComponentReq(int kind,
        ITciType componentType, string name, ITciValue hostId);
}
```

```

void TciStartTestComponentReq(Etsi.Ttcn3.Tri.ITriComponentId comp,
    ITciBehaviourId behavior, ITciParameterList parameterList);
void TciStopTestComponentReq(Etsi.Ttcn3.Tri.ITriComponentId comp);
void TciConnectReq(Etsi.Ttcn3.Tri.ITriPortId fromPort,
    Etsi.Ttcn3.Tri.ITriPortId toPort);
void TciDisconnectReq(Etsi.Ttcn3.Tri.ITriPortId fromPort,
    Etsi.Ttcn3.Tri.ITriPortId toPort);
void TciTestComponentTerminatedReq(Etsi.Ttcn3.Tri.ITriComponentId comp,
    ITciVerdictValue verdict);
bool TciTestComponentRunningReq(Etsi.Ttcn3.Tri.ITriComponentId comp);
Etsi.Ttcn3.Tri.ITriComponentId TciGetMmcReq();
void TciMapReq(Etsi.Ttcn3.Tri.ITriPortId fromPort,
    Etsi.Ttcn3.Tri.ITriPortId toPort);
void TciMapParamReq(Etsi.Ttcn3.Tri.ITriPortId fromPort,
    Etsi.Ttcn3.Tri.ITriPortId toPort, Etsi.Ttcn3.Tci.ITciParameterList parameterList);
void TciUnmapReq(Etsi.Ttcn3.Tri.ITriPortId fromPort,
    Etsi.Ttcn3.Tri.ITriPortId toPort);
void TciUnmapParamReq(Etsi.Ttcn3.Tri.ITriPortId fromPort,
    Etsi.Ttcn3.Tri.ITriPortId toPort, Etsi.Ttcn3.Tci.ITciParameterList parameterList);
void TciExecuteTestCaseReq(Etsi.Ttcn3.Tri.ITriComponentId component,
    Etsi.Ttcn3.Tri.ITriPortIdList tsiPortList);
void TciResetReq();
bool TciTestComponentDoneReq(Etsi.Ttcn3.Tri.ITriComponentId component);
void TciKillTestComponentReq(Etsi.Ttcn3.Tri.ITriComponentId component);
bool TciTestComponentAliveReq(Etsi.Ttcn3.Tri.ITriComponentId component);
bool TciTestComponentKilledReq(Etsi.Ttcn3.Tri.ITriComponentId component);
}

```

### 12.5.3.2 TCI-CH required

The **TCI-CH required** interface is mapped to the following interface:

```

public interface ITciCHRequired {
    void TciEnqueueMsgConnected(Etsi.Ttcn3.Tri.ITriPortId sender,
        Etsi.Ttcn3.Tri.ITriComponentId receiver,
        ITciValue receivedMessage);
    void TciEnqueueCallConnected(Etsi.Ttcn3.Tri.ITriPortId sender,
        Etsi.Ttcn3.Tri.ITriComponentId receiver,
        Etsi.Ttcn3.Tri.ITriSignatureId signature,
        ITciParameterList parameterList);
    void TciEnqueueReplyConnected(Etsi.Ttcn3.Tri.ITriPortId sender,
        Etsi.Ttcn3.Tri.ITriComponentId receiver,
        Etsi.Ttcn3.Tri.ITriSignatureId signature,
        ITciParameterList parameterList, ITciValue returnValue);
    void TciEnqueueRaiseConnected(Etsi.Ttcn3.Tri.ITriPortId sender,
        Etsi.Ttcn3.Tri.ITriComponentId receiver,
        Etsi.Ttcn3.Tri.ITriSignatureId signature, ITciValue except);
    Etsi.Ttcn3.Tri.ITriComponentId TciCreateTestComponent(int kind,
        ITciType componentType, string name);
    void TciStartTestComponent(Etsi.Ttcn3.Tri.ITriComponentId comp,
        ITciBehaviourId behavior, ITciParameterList parameterList);
    void TciStopTestComponent(Etsi.Ttcn3.Tri.ITriComponentId comp);
    void TciConnect(Etsi.Ttcn3.Tri.ITriPortId fromPort,
        Etsi.Ttcn3.Tri.ITriPortId toPort);
    void TciDisconnect(Etsi.Ttcn3.Tri.ITriPortId fromPort,
        Etsi.Ttcn3.Tri.ITriPortId toPort);
    void TciTestComponentTerminated(Etsi.Ttcn3.Tri.ITriComponentId comp,
        ITciVerdictValue verdict);
    bool TciTestComponentRunning(Etsi.Ttcn3.Tri.ITriComponentId comp);
    bool TciTestComponentDone(Etsi.Ttcn3.Tri.ITriComponentId comp);
    Etsi.Ttcn3.Tri.ITriComponentId TciGetMtc();
    void TciExecuteTestCase(ITciTestCaseId testCaseId,
        Etsi.Ttcn3.Tri.ITriPortIdList tsiPortList);
    void TciReset();
    void TciMap(Etsi.Ttcn3.Tri.ITriPortId fromPort,
        Etsi.Ttcn3.Tri.ITriPortId toPort);
}

```

```

void TciMapParam(Etsi.Ttcn3.Tri.ITriPortId fromPort,
    Etsi.Ttcn3.Tri.ITriPortId toPort, Etsi.Ttcn3.Tci.ITciParameterList parameterList);
void TciUnmap(Etsi.Ttcn3.Tri.ITriPortId fromPort,
    Etsi.Ttcn3.Tri.ITriPortId toPort);
void TciUnmapParam(Etsi.Ttcn3.Tri.ITriPortId fromPort,
    Etsi.Ttcn3.Tri.ITriPortId toPort, Etsi.Ttcn3.Tci.ITciParameterList parameterList);
void TciKillTestComponent(Etsi.Ttcn3.Tri.ITriComponentId component);
bool TciTestComponentAlive(Etsi.Ttcn3.Tri.ITriComponentId component);
bool TciTestComponentKilled(Etsi.Ttcn3.Tri.ITriComponentId component);
}

```

## 12.5.4 TCI-TL interface

### 12.5.4.1 TCI-TL provided

The TCI-TL provided interface is mapped to the following interface:

```

public interface ITciTLProvided {
    void TliTcExecute(string am, System.DateTime ts, string src, int line,
        Etsi.Ttcn3.Tri.ITriComponentId c, ITciTestCaseId tcId,
        ITciParameterList tciPars, Etsi.Ttcn3.Tri.ITriTimerDuration dur);
    void TliTcStart(string am, System.DateTime ts, string src, int line,
        Etsi.Ttcn3.Tri.ITriComponentId c, ITciTestCaseId tcId,
        ITciParameterList tciPars, Etsi.Ttcn3.Tri.ITriTimerDuration dur);
    void TliTcStop(string am, System.DateTime ts, string src, int line,
        Etsi.Ttcn3.Tri.ITriComponentId c, string reason);
    void TliTcStarted(string am, System.DateTime ts, string src, int line,
        Etsi.Ttcn3.Tri.ITriComponentId c, ITciTestCaseId tcId,
        ITciParameterList tciPars, Etsi.Ttcn3.Tri.ITriTimerDuration dur);
    void TliTcTerminated(string am, System.DateTime ts, string src, int line,
        Etsi.Ttcn3.Tri.ITriComponentId c, ITciTestCaseId tcId,
        ITciParameterList tciPars, ITciVerdictValue verdict,
        string reason);
    void TliCtrlStart(string am, System.DateTime ts, string src, int line,
        Etsi.Ttcn3.Tri.ITriComponentId c);
    void TliCtrlStop(string am, System.DateTime ts, string src, int line,
        Etsi.Ttcn3.Tri.ITriComponentId c);
    void TliCtrlTerminated(string am, System.DateTime ts, string src, int
        line, Etsi.Ttcn3.Tri.ITriComponentId c);
    void TliMSend_m(string am, System.DateTime ts, string src, int line,
        Etsi.Ttcn3.Tri.ITriComponentId c, Etsi.Ttcn3.Tri.ITriPortId at,
        Etsi.Ttcn3.Tri.ITriPortId to, ITciValue msgValue,
        ITciValue addrValue, TciStatus encoderFailure,
        Etsi.Ttcn3.Tri.ITriMessage msg, Etsi.Ttcn3.Tri.ITriAddress address,
        Etsi.Ttcn3.Tri.TriStatus transmissionFailure);
    void TliMSend_m_BC(string am, System.DateTime ts, string src, int line,
        Etsi.Ttcn3.Tri.ITriComponentId c, Etsi.Ttcn3.Tri.ITriPortId at,
        Etsi.Ttcn3.Tri.ITriPortId to, ITciValue msgValue,
        TciStatus encoderFailure, Etsi.Ttcn3.Tri.ITriMessage msg,
        Etsi.Ttcn3.Tri.TriStatus transmissionFailure);
    void TliMSend_m_MC(string am, System.DateTime ts, string src, int line,
        Etsi.Ttcn3.Tri.ITriComponentId c, Etsi.Ttcn3.Tri.ITriPortId at,
        Etsi.Ttcn3.Tri.ITriPortId to, ITciValue msgValue,
        ITciValueList addrValues, TciStatus encoderFailure,
        Etsi.Ttcn3.Tri.ITriMessage msg,
        Etsi.Ttcn3.Tri.ITriAddressList addresses,
        Etsi.Ttcn3.Tri.TriStatus transmissionFailure);
    void TliMSend_c(string am, System.DateTime ts, string src, int line,
        Etsi.Ttcn3.Tri.ITriComponentId c, Etsi.Ttcn3.Tri.ITriPortId at,
        Etsi.Ttcn3.Tri.ITriPortId to, ITciValue msgValue,
        Etsi.Ttcn3.Tri.TriStatus transmissionFailure);
    void TliMSend_c_BC(string am, System.DateTime ts, string src, int line,
        Etsi.Ttcn3.Tri.ITriComponentId c, Etsi.Ttcn3.Tri.ITriPortId at,
        Etsi.Ttcn3.Tri.ITriPortIdList to, ITciValue msgValue,
        Etsi.Ttcn3.Tri.TriStatus transmissionFailure);
    void TliMSend_c_MC(string am, System.DateTime ts, string src, int line,
        Etsi.Ttcn3.Tri.ITriComponentId c, Etsi.Ttcn3.Tri.ITriPortId at,
        Etsi.Ttcn3.Tri.ITriPortIdList to, ITciValue msgValue,
        Etsi.Ttcn3.Tri.TriStatus transmissionFailure);
    void TliMDetected_m(string am, System.DateTime ts, string src, int line,
        Etsi.Ttcn3.Tri.ITriComponentId c, Etsi.Ttcn3.Tri.ITriPortId at,
        Etsi.Ttcn3.Tri.ITriPortId from, Etsi.Ttcn3.Tri.ITriMessage msg,
        Etsi.Ttcn3.Tri.ITriAddress address);
    void TliMDetected_c(string am, System.DateTime ts, string src, int line,
        Etsi.Ttcn3.Tri.ITriComponentId c, Etsi.Ttcn3.Tri.ITriPortId at,
        Etsi.Ttcn3.Tri.ITriPortId from, ITciValue msgValue);
    void TliMMismatch_m(string am, System.DateTime ts, string src, int line,
        Etsi.Ttcn3.Tri.ITriComponentId c, Etsi.Ttcn3.Tri.ITriPortId at,
        ITciValue msgValue, ITciValueTemplate msgTmpl,

```

```

    ITciValueDifferenceList diffs, ITciValue address,
    ITciValueTemplate addressTmpl);
void TliMMismatch_c(string am, System.DateTime ts, string src, int line,
    Etsi.Ttcn3.Tri.ITriComponentId c, Etsi.Ttcn3.Tri.ITriPortId at,
    ITciValue msgValue, ITciValueTemplate msgTmpl,
    ITciValueDifferenceList diffs, Etsi.Ttcn3.Tri.ITriComponentId from,
    ITciNonValueTemplate fromTmpl);
void TliMReceive_m(string am, System.DateTime ts, string src, int line,
    Etsi.Ttcn3.Tri.ITriComponentId c, Etsi.Ttcn3.Tri.ITriPortId at,
    ITciValue msgValue, ITciValueTemplate msgTmpl, ITciValue address,
    ITciValueTemplate addressTmpl);
void TliMReceive_c(string am, System.DateTime ts, string src, int line,
    Etsi.Ttcn3.Tri.ITriComponentId c, Etsi.Ttcn3.Tri.ITriPortId at,
    ITciValue msgValue, ITciValueTemplate msgTmpl,
    Etsi.Ttcn3.Tri.ITriComponentId fromComp,
    ITciNonValueTemplate fromTmpl);
void TliPrCall_m(string am, System.DateTime ts, string src, int line,
    Etsi.Ttcn3.Tri.ITriComponentId c, Etsi.Ttcn3.Tri.ITriPortId at,
    Etsi.Ttcn3.Tri.ITriPortId to,
    Etsi.Ttcn3.Tri.ITriSignatureId signature,
    ITciParameterList tciPars, ITciValue addrValue,
    TciStatus encoderFailure, Etsi.Ttcn3.Tri.ITriParameterList triPars,
    Etsi.Ttcn3.Tri.ITriAddress address,
    Etsi.Ttcn3.Tri.TriStatus transmissionFailure);
void TliPrCall_m_BC(string am, System.DateTime ts, string src, int line,
    Etsi.Ttcn3.Tri.ITriComponentId c, Etsi.Ttcn3.Tri.ITriPortId at,
    Etsi.Ttcn3.Tri.ITriPortId to,
    Etsi.Ttcn3.Tri.ITriSignatureId signature, ITciParameterList tciPars,
    TciStatus encoderFailure, Etsi.Ttcn3.Tri.ITriParameterList triPars,
    Etsi.Ttcn3.Tri.TriStatus transmissionFailure);
void TliPrCall_m_MC(string am, System.DateTime ts, string src, int line,
    Etsi.Ttcn3.Tri.ITriComponentId c, Etsi.Ttcn3.Tri.ITriPortId at,
    Etsi.Ttcn3.Tri.ITriPortId to,
    Etsi.Ttcn3.Tri.ITriSignatureId signature, ITciParameterList tciPars,
    ITciValueList addrValues, TciStatus encoderFailure,
    Etsi.Ttcn3.Tri.ITriParameterList triPars,
    Etsi.Ttcn3.Tri.ITriAddressList addresses,
    Etsi.Ttcn3.Tri.TriStatus transmissionFailure);
void TliPrCall_c(string am, System.DateTime ts, string src, int line,
    Etsi.Ttcn3.Tri.ITriComponentId c, Etsi.Ttcn3.Tri.ITriPortId at,
    Etsi.Ttcn3.Tri.ITriPortId to,
    Etsi.Ttcn3.Tri.ITriSignatureId signature,
    ITciParameterList tciPars,
    Etsi.Ttcn3.Tri.TriStatus transmissionFailure);
void TliPrCall_c_BC(string am, System.DateTime ts, string src, int line,
    Etsi.Ttcn3.Tri.ITriComponentId c, Etsi.Ttcn3.Tri.ITriPortId at,
    Etsi.Ttcn3.Tri.ITriPortIdList to,
    Etsi.Ttcn3.Tri.ITriSignatureId signature, ITciParameterList tciPars,
    Etsi.Ttcn3.Tri.TriStatus transmissionFailure);
void TliPrCall_c_MC(string am, System.DateTime ts, string src, int line,
    Etsi.Ttcn3.Tri.ITriComponentId c, Etsi.Ttcn3.Tri.ITriPortId at,
    Etsi.Ttcn3.Tri.ITriPortIdList to,
    Etsi.Ttcn3.Tri.ITriSignatureId signature, ITciParameterList tciPars,
    Etsi.Ttcn3.Tri.TriStatus transmissionFailure);
void TliPrGetCallDetected_m(string am, System.DateTime ts, string src,
    int line, Etsi.Ttcn3.Tri.ITriComponentId c,
    Etsi.Ttcn3.Tri.ITriPortId at, Etsi.Ttcn3.Tri.ITriPortId from,
    Etsi.Ttcn3.Tri.ITriSignatureId signature,
    Etsi.Ttcn3.Tri.ITriParameterList triPars,
    Etsi.Ttcn3.Tri.ITriAddress address);
void TliPrGetCallDetected_c(string am, System.DateTime ts, string src,
    int line, Etsi.Ttcn3.Tri.ITriComponentId c,
    Etsi.Ttcn3.Tri.ITriPortId at, Etsi.Ttcn3.Tri.ITriPortId from,
    Etsi.Ttcn3.Tri.ITriSignatureId signature,
    ITciParameterList tciPars);
void TliPrGetCallMismatch_m(string am, System.DateTime ts, string src,
    int line, Etsi.Ttcn3.Tri.ITriComponentId c,
    Etsi.Ttcn3.Tri.ITriPortId at,
    Etsi.Ttcn3.Tri.ITriSignatureId signature, ITciParameterList tciPars,
    ITciValueTemplate parsTmpl, ITciValueDifferenceList diffs,
    ITciValue address, ITciValueTemplate addressTmpl);
void TliPrGetCallMismatch_c(string am, System.DateTime ts, string src,
    int line, Etsi.Ttcn3.Tri.ITriComponentId c,
    Etsi.Ttcn3.Tri.ITriPortId at,
    Etsi.Ttcn3.Tri.ITriSignatureId signature,
    ITciParameterList tciPars, ITciValueTemplate parsTmpl,
    ITciValueDifferenceList diffs, Etsi.Ttcn3.Tri.ITriComponentId from,
    ITciNonValueTemplate fromTmpl);
void TliPrGetCall_m(string am, System.DateTime ts, string src, int line,
    Etsi.Ttcn3.Tri.ITriComponentId c, Etsi.Ttcn3.Tri.ITriPortId at,

```

```

    Etsi.Ttcn3.Tri.ITriSignatureId signature,
    ITciParameterList tciPars, ITciValueTemplate parsTpl,
    ITciValue address, ITciValueTemplate addressTpl);
void TliPrGetCall_c(string am, System.DateTime ts, string src, int line,
    Etsi.Ttcn3.Tri.ITriComponentId c, Etsi.Ttcn3.Tri.ITriPortId at,
    Etsi.Ttcn3.Tri.ITriSignatureId signature,
    ITciParameterList tciPars, ITciValueTemplate parsTpl,
    Etsi.Ttcn3.Tri.ITriComponentId from, ITciNonValueTemplate fromTpl);
void TliPrReply_m(string am, System.DateTime ts, string src, int line,
    Etsi.Ttcn3.Tri.ITriComponentId c, Etsi.Ttcn3.Tri.ITriPortId at,
    Etsi.Ttcn3.Tri.ITriPortId to,
    Etsi.Ttcn3.Tri.ITriSignatureId signature, ITciParameterList tciPars,
    ITciValue replValue, ITciValue addrValue, TciStatus encoderFailure,
    Etsi.Ttcn3.Tri.ITriParameterList triPars,
    Etsi.Ttcn3.Tri.ITriParameter repl,
    Etsi.Ttcn3.Tri.ITriAddress address,
    Etsi.Ttcn3.Tri.TriStatus transmissionFailure);
void TliPrReply_m_BC(string am, System.DateTime ts, string src, int line,
    Etsi.Ttcn3.Tri.ITriComponentId c, Etsi.Ttcn3.Tri.ITriPortId at,
    Etsi.Ttcn3.Tri.ITriPortId to,
    Etsi.Ttcn3.Tri.ITriSignatureId signature, ITciParameterList tciPars,
    ITciValue replValue, TciStatus encoderFailure,
    Etsi.Ttcn3.Tri.ITriParameterList triPars,
    Etsi.Ttcn3.Tri.ITriParameter repl,
    Etsi.Ttcn3.Tri.TriStatus transmissionFailure);
void TliPrReply_m_MC(string am, System.DateTime ts, string src, int line,
    Etsi.Ttcn3.Tri.ITriComponentId c, Etsi.Ttcn3.Tri.ITriPortId at,
    Etsi.Ttcn3.Tri.ITriPortId to,
    Etsi.Ttcn3.Tri.ITriSignatureId signature, ITciParameterList tciPars,
    ITciValue replValue, ITciValueList addrValues,
    TciStatus encoderFailure, Etsi.Ttcn3.Tri.ITriParameterList triPars,
    Etsi.Ttcn3.Tri.ITriParameter repl,
    Etsi.Ttcn3.Tri.ITriAddressList addresses,
    Etsi.Ttcn3.Tri.TriStatus transmissionFailure);
void TliPrReply_c(string am, System.DateTime ts, string src, int line,
    Etsi.Ttcn3.Tri.ITriComponentId c, Etsi.Ttcn3.Tri.ITriPortId at,
    Etsi.Ttcn3.Tri.ITriPortId to,
    Etsi.Ttcn3.Tri.ITriSignatureId signature, ITciParameterList tciPars,
    ITciValue replValue, Etsi.Ttcn3.Tri.TriStatus transmissionFailure);
void TliPrReply_c_BC(string am, System.DateTime ts, string src, int line,
    Etsi.Ttcn3.Tri.ITriComponentId c, Etsi.Ttcn3.Tri.ITriPortId at,
    Etsi.Ttcn3.Tri.ITriPortIdList to,
    Etsi.Ttcn3.Tri.ITriSignatureId signature, ITciParameterList tciPars,
    ITciValue replValue, Etsi.Ttcn3.Tri.TriStatus transmissionFailure);
void TliPrReply_c_MC(string am, System.DateTime ts, string src, int line,
    Etsi.Ttcn3.Tri.ITriComponentId c, Etsi.Ttcn3.Tri.ITriPortId at,
    Etsi.Ttcn3.Tri.ITriPortIdList to,
    Etsi.Ttcn3.Tri.ITriSignatureId signature, ITciParameterList tciPars,
    ITciValue replValue, Etsi.Ttcn3.Tri.TriStatus transmissionFailure);
void TliPrGetReplyDetected_m(string am, System.DateTime ts, string src,
    int line, Etsi.Ttcn3.Tri.ITriComponentId c,
    Etsi.Ttcn3.Tri.ITriPortId at, Etsi.Ttcn3.Tri.ITriPortId from,
    Etsi.Ttcn3.Tri.ITriSignatureId signature,
    Etsi.Ttcn3.Tri.ITriParameterList triPars,
    Etsi.Ttcn3.Tri.ITriParameter repl,
    Etsi.Ttcn3.Tri.ITriAddress address);
void TliPrGetReplyDetected_c(string am, System.DateTime ts, string src,
    int line, Etsi.Ttcn3.Tri.ITriComponentId c,
    Etsi.Ttcn3.Tri.ITriPortId at, Etsi.Ttcn3.Tri.ITriPortId from,
    Etsi.Ttcn3.Tri.ITriSignatureId signature, ITciParameterList tciPars,
    ITciValue replValue);
void TliPrGetReplyMismatch_m(string am, System.DateTime ts, string src,
    int line, Etsi.Ttcn3.Tri.ITriComponentId c,
    Etsi.Ttcn3.Tri.ITriPortId at,
    Etsi.Ttcn3.Tri.ITriSignatureId signature, ITciParameterList tciPars,
    ITciValueTemplate parsTpl, ITciValue replValue,
    ITciValueTemplate replyTpl, ITciValueDifferenceList diffs,
    ITciValue address, ITciValueTemplate addressTpl);
void TliPrGetReplyMismatch_c(string am, System.DateTime ts, string src,
    int line, Etsi.Ttcn3.Tri.ITriComponentId c,
    Etsi.Ttcn3.Tri.ITriPortId at,
    Etsi.Ttcn3.Tri.ITriSignatureId signature, ITciParameterList tciPars,
    ITciValueTemplate parsTpl, ITciValue replValue,
    ITciValueTemplate replyTpl, ITciValueDifferenceList diffs,
    Etsi.Ttcn3.Tri.ITriComponentId from, ITciNonValueTemplate fromTpl);
void TliPrGetReply_m(string am, System.DateTime ts, string src, int line,
    Etsi.Ttcn3.Tri.ITriComponentId c, Etsi.Ttcn3.Tri.ITriPortId at,
    Etsi.Ttcn3.Tri.ITriSignatureId signature, ITciParameterList tciPars,
    ITciValueTemplate parsTpl, ITciValue replValue,
    ITciValueTemplate replyTpl, ITciValue address,

```



```

    ITciValueTemplate addressTpl);
void TliPrGetReply_c(string am, System.DateTime ts, string src, int line,
    Etsi.Ttcn3.Tri.ITriComponentId c, Etsi.Ttcn3.Tri.ITriPortId at,
    Etsi.Ttcn3.Tri.ITriSignatureId signature,
    ITciParameterList tciPars, ITciValueTemplate parsTpl,
    ITciValue replValue, ITciValueTemplate replyTpl,
    Etsi.Ttcn3.Tri.ITriComponentId from, ITciNonValueTemplate fromTpl);
void TliPrRaise_m(string am, System.DateTime ts, string src, int line,
    Etsi.Ttcn3.Tri.ITriComponentId c, Etsi.Ttcn3.Tri.ITriPortId at,
    Etsi.Ttcn3.Tri.ITriPortId to,
    Etsi.Ttcn3.Tri.ITriSignatureId signature, ITciParameterList tciPars,
    ITciValue excValue, ITciValue addrValue, TciStatus encoderFailure,
    Etsi.Ttcn3.Tri.ITriException exc,
    Etsi.Ttcn3.Tri.ITriAddress address,
    Etsi.Ttcn3.Tri.TriStatus transmissionFailure);
void TliPrRaise_m_BC(string am, System.DateTime ts, string src, int line,
    Etsi.Ttcn3.Tri.ITriComponentId c, Etsi.Ttcn3.Tri.ITriPortId at,
    Etsi.Ttcn3.Tri.ITriPortId to,
    Etsi.Ttcn3.Tri.ITriSignatureId signature, ITciParameterList tciPars,
    ITciValue excValue, TciStatus encoderFailure,
    Etsi.Ttcn3.Tri.ITriException exc,
    Etsi.Ttcn3.Tri.TriStatus transmissionFailure);
void TliPrRaise_m_MC(string am, System.DateTime ts, string src, int line,
    Etsi.Ttcn3.Tri.ITriComponentId c, Etsi.Ttcn3.Tri.ITriPortId at,
    Etsi.Ttcn3.Tri.ITriPortId to,
    Etsi.Ttcn3.Tri.ITriSignatureId signature, ITciParameterList tciPars,
    ITciValue excValue, ITciValueList addrValues,
    TciStatus encoderFailure, Etsi.Ttcn3.Tri.ITriException exc,
    Etsi.Ttcn3.Tri.ITriAddressList addresses,
    Etsi.Ttcn3.Tri.TriStatus transmissionFailure);
void TliPrRaise_c(string am, System.DateTime ts, string src, int line,
    Etsi.Ttcn3.Tri.ITriComponentId c, Etsi.Ttcn3.Tri.ITriPortId at,
    Etsi.Ttcn3.Tri.ITriPortId to,
    Etsi.Ttcn3.Tri.ITriSignatureId signature, ITciParameterList tciPars,
    ITciValue excValue, Etsi.Ttcn3.Tri.TriStatus transmissionFailure);
void TliPrRaise_c_BC(string am, System.DateTime ts, string src, int line,
    Etsi.Ttcn3.Tri.ITriComponentId c, Etsi.Ttcn3.Tri.ITriPortId at,
    Etsi.Ttcn3.Tri.ITriPortIdList to,
    Etsi.Ttcn3.Tri.ITriSignatureId signature,
    ITciParameterList tciPars, ITciValue excValue,
    Etsi.Ttcn3.Tri.TriStatus transmissionFailure);
void TliPrRaise_c_MC(string am, System.DateTime ts, string src, int line,
    Etsi.Ttcn3.Tri.ITriComponentId c, Etsi.Ttcn3.Tri.ITriPortId at,
    Etsi.Ttcn3.Tri.ITriPortIdList to,
    Etsi.Ttcn3.Tri.ITriSignatureId signature, ITciParameterList tciPars,
    ITciValue excValue, Etsi.Ttcn3.Tri.TriStatus transmissionFailure);
void TliPrCatchDetected_m(string am, System.DateTime ts, string src,
    int line, Etsi.Ttcn3.Tri.ITriComponentId c,
    Etsi.Ttcn3.Tri.ITriPortId at, Etsi.Ttcn3.Tri.ITriPortId from,
    Etsi.Ttcn3.Tri.ITriSignatureId signature,
    Etsi.Ttcn3.Tri.ITriException exc,
    Etsi.Ttcn3.Tri.ITriAddress address);
void TliPrCatchDetected_c(string am, System.DateTime ts, string src,
    int line, Etsi.Ttcn3.Tri.ITriComponentId c,
    Etsi.Ttcn3.Tri.ITriPortId at, Etsi.Ttcn3.Tri.ITriPortId from,
    Etsi.Ttcn3.Tri.ITriSignatureId signature, ITciValue excValue);
void TliPrCatchMismatch_m(string am, System.DateTime ts, string src,
    int line, Etsi.Ttcn3.Tri.ITriComponentId c,
    Etsi.Ttcn3.Tri.ITriPortId at,
    Etsi.Ttcn3.Tri.ITriSignatureId signature, ITciValue excValue,
    ITciValueTemplate excTpl, ITciValueDifferenceList diffs,
    ITciValue address, ITciValueTemplate addressTpl);
void TliPrCatchMismatch_c(string am, System.DateTime ts, string src,
    int line, Etsi.Ttcn3.Tri.ITriComponentId c,
    Etsi.Ttcn3.Tri.ITriPortId at,
    Etsi.Ttcn3.Tri.ITriSignatureId signature,
    ITciValue excValue, ITciValueTemplate excTpl,
    ITciValueDifferenceList diffs, Etsi.Ttcn3.Tri.ITriComponentId from,
    ITciNonValueTemplate fromTpl);
void TliPrCatch_m(string am, System.DateTime ts, string src, int line,
    Etsi.Ttcn3.Tri.ITriComponentId c, Etsi.Ttcn3.Tri.ITriPortId at,
    Etsi.Ttcn3.Tri.ITriSignatureId signature,
    ITciValue excValue, ITciValueTemplate excTpl, ITciValue address,
    ITciValueTemplate addressTpl);
void TliPrCatch_c(string am, System.DateTime ts, string src, int line,
    Etsi.Ttcn3.Tri.ITriComponentId c, Etsi.Ttcn3.Tri.ITriPortId at,
    Etsi.Ttcn3.Tri.ITriSignatureId signature,
    ITciValue excValue, ITciValueTemplate excTpl,
    Etsi.Ttcn3.Tri.ITriComponentId from, ITciNonValueTemplate fromTpl);
void TliPrCatchTimeoutDetected(string am, System.DateTime ts, string src,

```

```

    int line, Etsi.Ttcn3.Tri.ITriComponentId c,
    Etsi.Ttcn3.Tri.ITriPortId at,
    Etsi.Ttcn3.Tri.ITriSignatureId signature);
void TliPrCatchTimeout(string am, System.DateTime ts, string src,
    int line, Etsi.Ttcn3.Tri.ITriComponentId c,
    Etsi.Ttcn3.Tri.ITriPortId at,
    Etsi.Ttcn3.Tri.ITriSignatureId signature);
void TliCCreate(string am, System.DateTime ts, string src, int line,
    Etsi.Ttcn3.Tri.ITriComponentId c,
    Etsi.Ttcn3.Tri.ITriComponentId comp, string name, bool alive);
void TliCStart(string am, System.DateTime ts, string src, int line,
    Etsi.Ttcn3.Tri.ITriComponentId c,
    Etsi.Ttcn3.Tri.ITriComponentId comp, ITciBehaviourId name,
    ITciParameterList tciPars);
void TliCRunning(string am, System.DateTime ts, string src, int line,
    Etsi.Ttcn3.Tri.ITriComponentId c,
    Etsi.Ttcn3.Tri.ITriComponentId comp, TciComponentStatus status);
void TliCAlive(string am, System.DateTime ts, string src, int line,
    Etsi.Ttcn3.Tri.ITriComponentId c,
    Etsi.Ttcn3.Tri.ITriComponentId comp, TciComponentStatus status);
void TliCStop(string am, System.DateTime ts, string src, int line,
    Etsi.Ttcn3.Tri.ITriComponentId c,
    Etsi.Ttcn3.Tri.ITriComponentId comp);
void TliCKill(string am, System.DateTime ts, string src, int line,
    Etsi.Ttcn3.Tri.ITriComponentId c,
    Etsi.Ttcn3.Tri.ITriComponentId comp);
void TliCDoneMismatch(string am, System.DateTime ts, string src, int line,
    Etsi.Ttcn3.Tri.ITriComponentId c,
    Etsi.Ttcn3.Tri.ITriComponentId comp, ITciNonValueTemplate compTmpl);
void TliCDone(string am, System.DateTime ts, string src, int line,
    Etsi.Ttcn3.Tri.ITriComponentId c, ITciNonValueTemplate compTmpl);
void TliCKilledMismatch(string am, System.DateTime ts, string src,
    int line, Etsi.Ttcn3.Tri.ITriComponentId c,
    Etsi.Ttcn3.Tri.ITriComponentId comp, ITciNonValueTemplate compTmpl);
void TliCKilled(string am, System.DateTime ts, string src, int line,
    Etsi.Ttcn3.Tri.ITriComponentId c, ITciNonValueTemplate compTmpl);
void TliCTerminated(string am, System.DateTime ts, string src, int line,
    Etsi.Ttcn3.Tri.ITriComponentId c, ITciVerdictValue verdict,
    string reason);
void TliPConnect(string am, System.DateTime ts, string src, int line,
    Etsi.Ttcn3.Tri.ITriComponentId c, Etsi.Ttcn3.Tri.ITriPortId port1,
    Etsi.Ttcn3.Tri.ITriPortId port2);
void TliPDisconnect(string am, System.DateTime ts, string src, int line,
    Etsi.Ttcn3.Tri.ITriComponentId c, Etsi.Ttcn3.Tri.ITriPortId port1,
    Etsi.Ttcn3.Tri.ITriPortId port2);
void TliPMap(string am, System.DateTime ts, string src, int line,
    Etsi.Ttcn3.Tri.ITriComponentId c, Etsi.Ttcn3.Tri.ITriPortId port1,
    Etsi.Ttcn3.Tri.ITriPortId port2);
void TliPUnmap(string am, System.DateTime ts, string src, int line,
    Etsi.Ttcn3.Tri.ITriComponentId c, Etsi.Ttcn3.Tri.ITriPortId port1,
    Etsi.Ttcn3.Tri.ITriPortId port2);
void TliPClear(string am, System.DateTime ts, string src, int line,
    Etsi.Ttcn3.Tri.ITriComponentId c, Etsi.Ttcn3.Tri.ITriPortId port);
void TliPStart(string am, System.DateTime ts, string src, int line,
    Etsi.Ttcn3.Tri.ITriComponentId c, Etsi.Ttcn3.Tri.ITriPortId port);
void TliPStop(string am, System.DateTime ts, string src, int line,
    Etsi.Ttcn3.Tri.ITriComponentId c, Etsi.Ttcn3.Tri.ITriPortId port);
void TliPHalt(string am, System.DateTime ts, string src, int line,
    Etsi.Ttcn3.Tri.ITriComponentId c, Etsi.Ttcn3.Tri.ITriPortId port);
void TliEncode(string am, System.DateTime ts, string src, int line,
    Etsi.Ttcn3.Tri.ITriComponentId c, ITciValue val,
    TciStatus encoderFailure, Etsi.Ttcn3.Tri.ITriMessage msg,
    string codec);
void TliDecode(string am, System.DateTime ts, string src, int line,
    Etsi.Ttcn3.Tri.ITriComponentId c, Etsi.Ttcn3.Tri.ITriMessage msg,
    TciStatus decoderFailure, ITciValue val, string codec);
void TliTimeoutDetected(string am, System.DateTime ts, string src,
    int line, Etsi.Ttcn3.Tri.ITriComponentId c,
    Etsi.Ttcn3.Tri.ITriTimerId timer);
void TliTimeoutMismatch(string am, System.DateTime ts, string src,
    int line, Etsi.Ttcn3.Tri.ITriComponentId c,
    Etsi.Ttcn3.Tri.ITriTimerId timer, ITciNonValueTemplate timerTmpl);
void TliTimeout(string am, System.DateTime ts, string src, int line,
    Etsi.Ttcn3.Tri.ITriComponentId c, Etsi.Ttcn3.Tri.ITriTimerId timer,
    ITciNonValueTemplate timerTmpl);
void TliTStart(string am, System.DateTime ts, string src, int line,
    Etsi.Ttcn3.Tri.ITriComponentId c, Etsi.Ttcn3.Tri.ITriTimerId timer,
    Etsi.Ttcn3.Tri.ITriTimerDuration dur);
void TliTStop(string am, System.DateTime ts, string src, int line,
    Etsi.Ttcn3.Tri.ITriComponentId c, Etsi.Ttcn3.Tri.ITriTimerId timer,

```

```

    Etsi.Ttcn3.Tri.ITriTimerDuration dur);
void TliTRead(string am, System.DateTime ts, string src, int line,
    Etsi.Ttcn3.Tri.ITriComponentId c, Etsi.Ttcn3.Tri.ITriTimerId timer,
    Etsi.Ttcn3.Tri.ITriTimerDuration elapsed);
void TliTRunning(string am, System.DateTime ts, string src, int line,
    Etsi.Ttcn3.Tri.ITriComponentId c, Etsi.Ttcn3.Tri.ITriTimerId timer,
    TciTimerStatus status);
void TliSEnter(string am, System.DateTime ts, string src, int line,
    Etsi.Ttcn3.Tri.ITriComponentId c,
    Etsi.Ttcn3.Tri.IQualifiedName name, ITciParameterList tciPars,
    string kind);
void TliSLeave(string am, System.DateTime ts, string src, int line,
    Etsi.Ttcn3.Tri.ITriComponentId c,
    Etsi.Ttcn3.Tri.IQualifiedName name, ITciParameterList tciPars,
    ITciValue returnValue, string kind);
void TliVAr(string am, System.DateTime ts, string src, int line,
    Etsi.Ttcn3.Tri.ITriComponentId c,
    Etsi.Ttcn3.Tri.IQualifiedName name, ITciValue varValue);
void TliModulePar(string am, System.DateTime ts, string src, int line,
    Etsi.Ttcn3.Tri.ITriComponentId c,
    Etsi.Ttcn3.Tri.IQualifiedName name, ITciValue parValue);
void TliGetVerdict(string am, System.DateTime ts, string src, int line,
    Etsi.Ttcn3.Tri.ITriComponentId c, ITciVerdictValue verdict);
void TliSetVerdict(string am, System.DateTime ts, string src, int line,
    Etsi.Ttcn3.Tri.ITriComponentId c, ITciVerdictValue verdict,
    string reason);
void TliLog(string am, System.DateTime ts, string src, int line,
    Etsi.Ttcn3.Tri.ITriComponentId c, string log);
void TliAEnter(string am, System.DateTime ts, string src, int line,
    Etsi.Ttcn3.Tri.ITriComponentId c);
void TliALeave(string am, System.DateTime ts, string src, int line,
    Etsi.Ttcn3.Tri.ITriComponentId c);
void TliADefaults(string am, System.DateTime ts, string src, int line,
    Etsi.Ttcn3.Tri.ITriComponentId c);
void TliAActivate(string am, System.DateTime ts, string src, int line,
    Etsi.Ttcn3.Tri.ITriComponentId c,
    Etsi.Ttcn3.Tri.IQualifiedName name, ITciParameterList tciPars,
    ITciValue expr);
void TliAdeactivate(string am, System.DateTime ts, string src, int line,
    Etsi.Ttcn3.Tri.ITriComponentId c, ITciValue expr);
void TliANomatch(string am, System.DateTime ts, string src, int line,
    Etsi.Ttcn3.Tri.ITriComponentId c);
void TliARepeat(string am, System.DateTime ts, string src, int line,
    Etsi.Ttcn3.Tri.ITriComponentId c);
void TliAWait(string am, System.DateTime ts, string src, int line,
    Etsi.Ttcn3.Tri.ITriComponentId c);
void TliAction(string am, System.DateTime ts, string src, int line,
    Etsi.Ttcn3.Tri.ITriComponentId c, string action);
void TliMatch(string am, System.DateTime ts, string src, int line,
    Etsi.Ttcn3.Tri.ITriComponentId c, ITciValue expr,
    ITciValueTemplate tpl);
void TliMatchMismatch(string am, System.DateTime ts, string src, int line,
    Etsi.Ttcn3.Tri.ITriComponentId c, ITciValue expr,
    ITciValueTemplate tpl, ITciValueDifferenceList diffs);
void TliInfo(string am, System.DateTime ts, string src, int line,
    Etsi.Ttcn3.Tri.ITriComponentId c, int level, string info);
}

```

## 12.6 Optional parameters

Clause 7.3 in [1] defines that a reserved value shall be used to indicate the absence of an optional parameter. For the C# language mapping the distinct value null shall be used to indicate the absence of an optional value. For example, if in the `TciReplyConnected` operation the value parameter shall be omitted the operation invocation shall be `TciReplyConnected (sender, receiver, signature, parameterList, null)`.

## 12.7 Error handling

All operations called from the TM, CH or CD that return have succeeded. If an erroneous situation has been identified by the TE a test case error will be communicated to the user using the procedures as defined in clause 7.3.1.2.6 (`TciError`). If an operation called by the TE in the TM,

CH, CD, or TL produces an error, this erroneous situation should be communicated to the TE using the procedures as defined in clause 7.3.2.1.12 (TciErrorReq).

Beside this error handling and exceptions specified for indexing operators no additional error handling is defined in the C# mapping.

## Annex A IDL specification of TCI

(This annex forms an integral part of this Recommendation.)

This annex defines the TTCN-3 control interfaces using the interface definition language (IDL).

```
// *****
// * Interface definitions for the TTCN-3 Control Interfaces
// *****

module tciInterface {

    /* Forward declaration */
    interface Value;
    interface Type;

    // *****
    // * Data types taken from the TRI definitions
    // *****

    // Connection
    native TriPortIdType ;
    native TriPortIdListType;
    native TriComponentIdType ;
    native TriComponentIdListType;

    // Communications
    native TriMessageType;
    native TriParameterType;
    native TriParameterListType;
    native TriAddressType;
    native TriAddressListType;
    native TriExceptionType;
    native TriSignatureIdType;

    // Miscellaneous
    native TriStatusType;
    native TriTimerIdType;
    native TriTimerDurationType;

    native TciStatusType;

    // *****
    // * General Abstract Data Types
    // *****

    // Basic definitions
    native TBoolean;
    native TFloat;
    native TChar;
    native TInteger;
    native TString;
    native TUniversalChar;
    typedef sequence <TString> TStringSeq;

    struct QualifiedName {
        TString moduleName;
        TString baseName;
    };

    // General TCI abstract data types
    typedef QualifiedName TciBehaviourIdType;
    typedef QualifiedName TciModuleIdType;
    typedef QualifiedName TciModuleParameterIdType;
    typedef QualifiedName TciTestCaseIdType;

    enum TciParameterPassingModeType {
        IN_MODE,
        OUT_MODE,
        INOUT_MODE
    };
};
```

```

struct TciParameterType {
    TciModuleParameterIdType parameterName;
    Value parameterValue;
    TciParameterPassingModeType mode;
};

typedef sequence <TciParameterType> TciParameterListType;

struct TciParameterTypeType {
    Type parameterType;
    TciParameterPassingModeType mode;
};

typedef sequence <TciParameterTypeType> TciParameterTypeListType;

struct TciModuleParameterType {
    TciModuleParameterIdType parameterName;
    Value defaultValue;
};

typedef sequence <TciModuleIdType> TciModuleIdListType ;

typedef sequence <TciModuleParameterType> TciModuleParameterListType;

typedef sequence <TciTestCaseIdType> TciTestCaseIdListType;

enum TciTestComponentKindType {
    CONTROL,
    MTC,
    PTC,
    SYSTEM,
    PTC_ALIVE
};

enum ComponentStatusType{
    inactiveC,
    runningC,
    stoppedC,
    killedC,
    nullC
};

enum TimerStatusType{
    runningT,
    inactiveT,
    expiredT,
    nullT
};

enum PortStatusType{
    startedP,
    haltedP,
    stoppedP
};

enum TciTypeClassType {
    ADDRESS_CLASS,
    ANYTYPE_CLASS,
    BITSTRING_CLASS,
    BOOLEAN_CLASS,
    CHARSTRING_CLASS,
    COMPONENT_CLASS,
    ENUMERATED_CLASS,
    FLOAT_CLASS,
    HEXSTRING_CLASS,
    INTEGER_CLASS,
    OCTETSTRING_CLASS,
    RECORD_CLASS,
    RECORDOF_CLASS,
    ARRAY_CLASS,
    SET_CLASS,
    SETOF_CLASS,
    UNION_CLASS,
    UNIVERSALCHARSTRING_CLASS,
    VERDICT_CLASS
};

// *****

```

```

// * Abstract TTCN-3 Data Types And Values
// *****

// Abstract data type "Type"
interface Type {
    TciModuleIdType  getDefiningModule ();
    TString          getName ();
    TciTypeClassType getTypeClass ();
    Value            newInstance ();
    TString          getTypeEncoding ();
    TString          getTypeEncodingVariant ();
    TStringSeq       getTypextension ();
};

// Abstract TTCN-3 Values
interface Value {
    TString  getValueEncoding ();
    TString  getValueEncodingVariant ();
    Type     getType ();
    TBoolean notPresent ();
};

interface RecordOfValue : Value {
    Value  getField (in TInteger position);
    void   setField (
        in TInteger position,
        in Value value
    );
    void   appendField (in Value value);
    Type   getElementType ();
    TInteger getLength ();
    void    setLength (in TInteger len);
    TInteger getOffset ();
};

interface RecordValue : Value {
    Value  getField (in TString fieldName);
    void   setField (
        in TString fieldName,
        in Value value
    );
    TStringSeq getFieldNames ();
    void    setFieldOmitted (in TString fieldName);
};

interface VerdictValue : Value {
    TInteger getVerdict ();
    void     setVerdict (in TInteger verdict);
};

interface BitstringValue : Value {
    TString  getString ();
    void     setString (in TString value);
    TInteger getBit (in TInteger position);
    void     setBit (
        in TInteger position,
        in TInteger value
    );
    TInteger getLength ();
    void     setLength (in TInteger len);
};

interface OctetstringValue : Value {
    TString  getString ();
    void     setString (in TString value);
    TInteger getOctet (in TInteger position);
    void     setOctet (
        in TInteger position,
        in TInteger value
    );
    TInteger getLength ();
    void     setLength (in TInteger len);
};

interface FloatValue : Value {
    TFloat getFloat ();
    void   setFloat (in TFloat value);
};

interface HexstringValue : Value {

```

```

TString getString ();
void setString (in TString value);
TInteger getHex (in TInteger position);
void setHex (
    in TInteger position,
    in TInteger value
);
TInteger getLength ();
void setLength (in TInteger len);
};

interface EnumeratedValue : Value {
    void setEnum (in TString enumValue);
    TString getEnum ();
};

interface IntegerValue : Value {
    TInteger getInt ();
    void setInt (in TInteger value);
};

interface CharValue : Value {
    TChar getChar ();
    void setChar (in TChar value);
};

interface CharstringValue : Value {
    TString getString ();
    void setString (in TString value);
    TChar getChar (in TInteger position);
    void setChar (
        in TInteger position,
        in TChar value
    );
    TInteger getLength ();
    void setLength (in TInteger len);
};

interface BooleanValue : Value {
    TBoolean getBoolean ();
    void setBoolean (in TBoolean value);
};

interface UniversalCharValue : Value {
    TUniversalChar getUniversalChar ();
    void setUniversalChar (in TUniversalChar value);
};

interface UniversalCharstringValue : Value {
    TString getString ();
    void setString (in TString value);
    TUniversalChar getChar (in TInteger position);
    void setChar (
        in TInteger position,
        in TUniversalChar value
    );
    TInteger getLength ();
    void setLength (in TInteger len);
};

interface UnionValue : Value {
    Value getVariant (in TString variantName);
    void setVariant (
        in TString variantName,
        in Value value
    );
    TString getPresentVariantName ();
    TStringSeq getVariantNames ();
};

// *****
// * Abstract Logging Types
// *****

interface TciValueTemplate : Value {
    TBoolean isOmit ();
    TBoolean isAny();
    TBoolean isAnyOrOmit();
    TString getTemplateDef();
};

```



```

};

interface TciNonValueTemplate {
    TBoolean isAny();
    TBoolean isAll();
    TString getTemplateDef();
};

typedef sequence <Value> TciValueListType;

struct TciValueDifferenceType
{
    TString desc;
    Value val;
    TciValueTemplate tmpl;
};

typedef sequence <TciValueDifferenceType> TciValueDifferenceListType;

interface TciValueList {
    attribute TciValueListType inst;
    TInteger size();
    TBoolean isEmpty();
    Value get(in TInteger index);
};

interface TciValueDifference {
    attribute TciValueDifferenceType inst;
    Value getValue();
    TciValueTemplate getTciValueTemplate();
    TString getDescription();
};

interface TciValueDifferenceList {
    attribute TciValueDifferenceListType inst;
    TInteger size();
    TBoolean isEmpty();
    TciValueDifference get(in TInteger index);
};

// *****
// Coding Decoding Interface
// - Required
// *****

interface TCI_CD_Required {
    Type getTypeForName (in TString typeName);
    Type getInteger ();
    Type getFloat ();
    Type getBoolean ();
    Type getChar ();
    Type getUniversalChar ();
    Type getCharstring ();
    Type getUniversalCharstring ();
    Type getHexstring ();
    Type getBitstring ();
    Type getOctetstring ();
    Type getVerdict ();
    void tciErrorReq (in TString message);
};

// *****
// Coding Decoding interface
// - Provided
// *****

interface TCI_CD_Provided {
    Value decode (
        in TriMessageType message,
        in Type decodingHypothesis
    );
    TriMessageType encode (in Value value);
};

// *****
// Test Management Interface
// - Required
// *****

```

```

interface TCI_TM_Required : TCI_CD_Required {
    void tciRootModule (in TciModuleIdType moduleName);
    TciModuleIdListType tciGetImportedModules();
    TciModuleParameterListType tciGetModuleParameters (in TciModuleIdType moduleName);
    TciTestCaseIdListType tciGetTestCases ();
    TciParameterTypeListType tciGetTestCaseParameters (
        in TciTestCaseIdType testCaseId
    );
    TriPortIdListType tciGetTestCaseTSI (
        in TciTestCaseIdType testCaseId
    );
    void tciStartTestCase (
        in TciTestCaseIdType testCaseId,
        in TciParameterListType parameterList
    );
    void tciStopTestCase ();
    TriComponentIdType tciStartControl ();
    void tciStopControl ();
};

// *****
// Test Management Interface
// - Provided
// *****

interface TCI_TM_Provided {
    void tciTestCaseStarted (
        in TciTestCaseIdType testCaseId,
        in TciParameterListType parameterList,
        in TFloat timer
    );
    void tciTestCaseTerminated (
        in VerdictValue verdict,
        in TciParameterListType parameterList
    );
    void tciControlTerminated ();
    Value tciGetModulePar (
        in TciModuleParameterIdType parameterId
    );
    void tciLog (
        in TriComponentIdType testComponentId,
        in TString message
    );
    void tciError (in TString message);
};

// *****
// Component Handling Interface
// - Required
// *****

interface TCI_CH_Required : TCI_CD_Required {
    void tciEnqueueMsgConnected (
        in TriPortIdType sender,
        in TriComponentIdType receiver,
        in Value receivedMessage
    );
    void tciEnqueueCallConnected (
        in TriPortIdType sender,
        in TriComponentIdType receiver,
        in TriSignatureIdType signature,
        in TciParameterListType parameterList
    );
    void tciEnqueueReplyConnected (
        in TriPortIdType sender,
        in TriComponentIdType receiver,
        in TriSignatureIdType signature,
        in TciParameterListType parameterList,
        in Value returnValue
    );
    void tciEnqueueRaiseConnected (
        in TriPortIdType sender,
        in TriComponentIdType receiver,
        in TriSignatureIdType signature,
        in Value except
    );
    TriComponentIdType tciCreateTestComponent (
        in TciTestComponentKindType kind,
        in Type componentType,

```

```

        in TString name,
        in Value hostId
    );
void tciStartTestComponent (
    in TriComponentIdType comp,
    in TciBehaviourIdType behavior,
    in TciParameterListType parameterList
);
void tciStopTestComponent (
    in TriComponentIdType comp
);
void tciConnect (
    in TriPortIdType fromPort,
    in TriPortIdType toPort
);
void tciDisconnect (
    in TriPortIdType fromPort,
    in TriPortIdType toPort
);
void tciTestComponentTerminated (
    in TriComponentIdType comp,
    in VerdictValue verdict
);
TBoolean tciTestComponentRunning (
    in TriComponentIdType comp
);
TriComponentIdType tciGetMTC ();
void tciMap (
    in TriPortIdType fromPort,
    in TriPortIdType toPort
);
void tciUnmap (
    in TriPortIdType fromPort,
    in TriPortIdType toPort
);
void tciExecuteTestCase (
    in TciTestCaseIdType testCaseId,
    in TriPortIdListType tsiPortList
);
TBoolean tciTestComponentDone (
    in TriComponentIdType comp
);
void tciReset ();
};

// *****
// Component Handling Interface
// - Provided
// *****

interface TCI_CH_Provided {
    void tciSendConnected (
        in TriPortIdType sender,
        in TriComponentIdType receiver,
        in Value sendMessage
    );
    void tciSendConnectedBC (
        in TriPortIdType sender,
        in Value sendMessage
    );
    void tciSendConnectedMC (
        in TriPortIdType sender,
        in TriComponentIdListType receivers,
        in Value sendMessage
    );

    void tciCallConnected (
        in TriPortIdType sender,
        in TriComponentIdType receiver,
        in TriSignatureIdType signature,
        in TciParameterListType parameterList
    );
    void tciCallConnectedBC (
        in TriPortIdType sender,
        in TriSignatureIdType signature,
        in TciParameterListType parameterList
    );
    void tciCallConnectedMC (
        in TriPortIdType sender,
        in TriComponentIdListType receivers,

```

```

        in TriSignatureIdType signature,
        in TciParameterListType parameterList
    );

void tciReplyConnected (
    in TriPortIdType sender,
    in TriComponentIdType receiver,
    in TriSignatureIdType signature,
    in TciParameterListType parameterList,
    in Value returnValue
);

void tciReplyConnectedBC (
    in TriPortIdType sender,
    in TriSignatureIdType signature,
    in TciParameterListType parameterList,
    in Value returnValue
);

void tciReplyConnectedMC (
    in TriPortIdType sender,
    in TriComponentIdListType receivers,
    in TriSignatureIdType signature,
    in TciParameterListType parameterList,
    in Value returnValue
);

void tciRaiseConnected (
    in TriPortIdType sender,
    in TriComponentIdType receiver,
    in TriSignatureIdType signature,
    in Value except
);

void tciRaiseConnectedBC (
    in TriPortIdType sender,
    in TriSignatureIdType signature,
    in Value except
);

void tciRaiseConnectedMC (
    in TriPortIdType sender,
    in TriComponentIdListType receivers,
    in TriSignatureIdType signature,
    in Value except
);

TriComponentIdType tciCreateTestComponentReq (
    in TciTestComponentKindType kind,
    in Type componentType,
    in TString name
);

void tciStartTestComponentReq (
    in TriComponentIdType comp,
    in TciBehaviourIdType behavior,
    in TciParameterListType parameterList
);

void tciStopTestComponentReq (
    in TriComponentIdType comp
);

void tciConnectReq (
    in TriPortIdType fromPort,
    in TriPortIdType toPort
);

void tciDisconnectReq (
    in TriPortIdType fromPort,
    in TriPortIdType toPort
);

void tciTestComponentTerminatedReq (
    in TriComponentIdType comp,
    in VerdictValue verdict
);

TBoolean tciTestComponentRunningReq (
    in TriComponentIdType comp
);

TriComponentIdType tciGetMTCReq ();

void tciMapReq (
    in TriPortIdType fromPort,
    in TriPortIdType toPort
);

void tciUnmapReq (
    in TriPortIdType fromPort,
    in TriPortIdType toPort
);

```

```

void tciExecuteTestCaseReq (
    in TciTestCaseIdType testCaseId,
    in TriPortIdListType tsiPortList
);
void tciResetReq ();
TBoolean tciTestComponentDoneReq (
    in TriComponentIdType comp
);
};

// *****
// Test Logging Interface
// - Provided
// *****

interface TCI_TL_Provided {
    void tliTcExecute(
        in TString am, in TInteger ts, in TString src, in TInteger line,
        in TriComponentIdType c, in TciTestCaseIdType tcId,
        in TciParameterListType tciPars, in TriTimerDurationType dur
    );
    void tliTcStart(
        in TString am, in TInteger ts, in TString src, in TInteger line,
        in TriComponentIdType c, in TciTestCaseIdType tcId,
        in TciParameterListType tciPars, in TriTimerDurationType dur
    );
    void tliTcStop(
        in TString am, in TInteger ts, in TString src, in TInteger line,
        in TriComponentIdType c, in TString reason
    );
    void tliTcStarted(
        in TString am, in TInteger ts, in TString src, in TInteger line,
        in TriComponentIdType c, in TciTestCaseIdType tcId,
        in TciParameterListType tciPars, in TriTimerDurationType dur
    );
    void tliTcTerminated(
        in TString am, in TInteger ts, in TString src, in TInteger line,
        in TriComponentIdType c, in TciTestCaseIdType tcId,
        in TciParameterListType tciPars, in VerdictValue verdict, in TString reason);

    void tliCtrlStart(
        in TString am, in TInteger ts, in TString src, in TInteger line,
        in TriComponentIdType c
    );
    void tliCtrlStop(
        in TString am, in TInteger ts, in TString src, in TInteger line,
        in TriComponentIdType c
    );
    void tliCtrlTerminated(
        in TString am, in TInteger ts, in TString src, in TInteger line,
        in TriComponentIdType c);

    void tliMSend_m(
        in TString am, in TInteger ts, in TString src, in TInteger line,
        in TriComponentIdType c, in TriPortIdType at, in TriPortIdType to, in Value msgValue,
        in Value addrValue, in TciStatusType encoderFailure,
        in TriMessageType msg, in TriAddressType address, in TriStatusType transmissionFailure
    );
    void tliMSend_m_BC(
        in TString am, in TInteger ts, in TString src, in TInteger line,
        in TriComponentIdType c, in TriPortIdType at, in TriPortIdType to, in Value msgValue,
        in TciStatusType encoderFailure, in TriMessageType msg,
        in TriStatusType transmissionFailure
    );
    void tliMSend_m_MC(
        in TString am, in TInteger ts, in TString src, in TInteger line,
        in TriComponentIdType c, in TriPortIdType at, in TriPortIdType to, in Value msgValue,
        in TciValueList addrValues, in TciStatusType encoderFailure,
        in TriMessageType msg, in TriAddressListType addresses,
        in TriStatusType transmissionFailure
    );

    void tliMSend_c(
        in TString am, in TInteger ts, in TString src, in TInteger line,
        in TriComponentIdType c, in TriPortIdType at, in TriPortIdType to, in Value msgValue,
        in TriStatusType transmissionFailure
    );
    void tliMSend_c_BC(
        in TString am, in TInteger ts, in TString src, in TInteger line,
        in TriComponentIdType c, in TriPortIdType at, in TriPortIdListType to, in Value msgValue,

```

```

        in TriStatusType transmissionFailure
    );
void tliMSend_c_MC(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in TriPortIdType at, in TriPortIdListType to, in Value msgValue,
    in TriStatusType transmissionFailure);

void tliMDetected_m(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in TriPortIdType at, in TriPortIdType from,
    in TriMessageType msg,
    in TriAddressType address
);
void tliMDetected_c(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in TriPortIdType at, in TriPortIdType from, in Value msgValue
);
void tliMMismatch_m(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in TriPortIdType at, in Value msgValue,
    in TciValueTemplate msgTpl, in TciValueDifferenceList diffs,
    in Value addrValue, in TciValueTemplate addressTpl
);
void tliMMismatch_c(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in TriPortIdType at, in Value msgValue,
    in TciValueTemplate msgTpl, in TciValueDifferenceList diffs,
    in TriComponentIdType from, in TciNonValueTemplate fromTpl
);
void tliMReceive_m(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in TriPortIdType at, in Value msgValue,
    in TciValueTemplate msgTpl, in Value addrValue,
    in TciValueTemplate addressTpl
);
void tliMReceive_c(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in TriPortIdType at, in Value msgValue,
    in TciValueTemplate msgTpl, in TriComponentIdType from,
    in TciNonValueTemplate fromTpl
);

void tliPrCall_m(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in TriPortIdType at, in TriPortIdType to,
    in TriSignatureIdType signature, in TciParameterListType tciPars,
    in Value addrValue, in TciStatusType encoderFailure,
    in TriParameterListType triPars, in TriAddressType address,
    in TriStatusType transmissionFailure
);
void tliPrCall_m_BC(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in TriPortIdType at, in TriPortIdType to,
    in TriSignatureIdType signature, in TciParameterListType tciPars,
    in TciStatusType encoderFailure, in TriParameterListType triPars,
    in TriStatusType transmissionFailure
);
void tliPrCall_m_MC(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in TriPortIdType at, in TriPortIdType to,
    in TriSignatureIdType signature, in TciParameterListType tciPars,
    in TciValueList addrValues, in TciStatusType encoderFailure,
    in TriParameterListType triPars, in TriAddressListType addresses,
    in TriStatusType transmissionFailure
);

void tliPrCall_c(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in TriPortIdType at, in TriPortIdType to,
    in TriSignatureIdType signature, in TciParameterListType tciPars,
    in TriStatusType transmissionFailure
);
void tliPrCall_c_BC(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in TriPortIdType at, in TriPortIdListType to,
    in TriSignatureIdType signature, in TciParameterListType tciPars,
    in TriStatusType transmissionFailure
);
void tliPrCall_c_MC(
    in TString am, in TInteger ts, in TString src, in TInteger line,

```

```

        in TriComponentIdType c, in TriPortIdType at, in TriPortIdListType to,
        in TriSignatureIdType signature, in TciParameterListType tciPars,
        in TriStatusType transmissionFailure
    );

void tliPrGetCallDetected_m(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in TriPortIdType at, in TriPortIdType from,
    in TriSignatureIdType signature, in TriParameterListType triPars,
    in TriAddressType address
);

void tliPrGetCallDetected_c(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in TriPortIdType at, in TriPortIdType from,
    in TriSignatureIdType signature, in TciParameterListType tciPars
);

void tliPrGetCallMismatch_m(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in TriPortIdType at,
    in TriSignatureIdType signature, in TciParameterListType tciPars,
    in TciValueTemplate parsTpl, in TciValueDifferenceList diffs,
    in Value addrValue, in TciValueTemplate addressTpl
);

void tliPrGetCallMismatch_c(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in TriPortIdType at,
    in TriSignatureIdType signature, in TciParameterListType tciPars,
    in TciValueTemplate parsTpl, in TciValueDifferenceList diffs,
    in TriComponentIdType from, in TciNonValueTemplate fromTpl
);

void tliPrGetCall_m(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in TriPortIdType at,
    in TriSignatureIdType signature, in TciParameterListType tciPars,
    in TciValueTemplate parsTpl, in Value addrValue,
    in TciValueTemplate addressTpl
);

void tliPrGetCall_c(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in TriPortIdType at,
    in TriSignatureIdType signature, in TciParameterListType tciPars,
    in TciValueTemplate parsTpl, in TriComponentIdType from,
    in TciNonValueTemplate fromTpl
);

void tliPrReply_m(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in TriPortIdType at, in TriPortIdType to,
    in TriSignatureIdType signature, in TciParameterListType tciPars,
    in Value replValue, in Value addrValue,
    in TciStatusType encoderFailure, in TriParameterListType triPars,
    in TriParameterType repl, in TriAddressType address, in TriStatusType transmissionFailure
);

void tliPrReply_m_BC(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in TriPortIdType at, in TriPortIdType to,
    in TriSignatureIdType signature, in TciParameterListType tciPars, in Value replValue,
    in TciStatusType encoderFailure, in TriParameterListType triPars,
    in TriParameterType repl, in TriStatusType transmissionFailure
);

void tliPrReply_m_MC(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in TriPortIdType at, in TriPortIdType to,
    in TriSignatureIdType signature, in TciParameterListType tciPars, in Value replValue,
    in TciValueListType addrValues, in TciStatusType encoderFailure,
    in TriParameterListType triPars, in TriParameterType repl,
    in TriAddressListType addresses, in TriStatusType transmissionFailure
);

void tliPrReply_c(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in TriPortIdType at, in TriPortIdType to,
    in TriSignatureIdType signature, in TciParameterListType tciPars,
    in Value replValue, in TriStatusType transmissionFailure
);

void tliPrReply_c_BC(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in TriPortIdType at, in TriPortIdListType to,
    in TriSignatureIdType signature, in Value parsValue, in Value replValue,
    in TriStatusType transmissionFailure
);

```

```

);
void tliPrReply_c_MC(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in TriPortIdType at, in TriPortIdListType to,
    in TriSignatureIdType signature, in Value parsValue, in Value replValue,
    in TriStatusType transmissionFailure
);

void tliPrGetReplyDetected_m(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in TriPortIdType at, in TriPortIdType from,
    in TriSignatureIdType signature, in TriParameterListType triPars,
    in TriParameterType repl, in TriAddressType address
);

void tliPrGetReplyDetected_c(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in TriPortIdType at, in TriPortIdType from,
    in TriSignatureIdType signature, in TciParameterListType tciPars,
    in Value replValue
);

void tliPrGetReplyMismatch_m(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in TriPortIdType at,
    in TriSignatureIdType signature,
    in TciParameterListType tciPars, in TciValueTemplate parsTmpl,
    in Value replValue, in TciValueTemplate replyTmpl,
    in TciValueDifferenceList diffs, in Value addrValue,
    in TciValueTemplate addressTmpl
);

void tliPrGetReplyMismatch_c(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in TriPortIdType at,
    in TriSignatureIdType signature,
    in TciParameterListType tciPars, in TciValueTemplate parsTmpl,
    in Value replValue, in TciValueTemplate replyTmpl,
    in TciValueDifferenceList diffs, in TriComponentIdType from,
    in TciNonValueTemplate fromTmpl
);

void tliPrGetReply_m(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in TriPortIdType at,
    in TriSignatureIdType signature,
    in TciParameterListType tciPars, in TciValueTemplate parsTmpl,
    in Value replValue, in TciValueTemplate replyTmpl,
    in Value addrValue, in TciValueTemplate addressTmpl
);

void tliPrGetReply_c(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in TriPortIdType at,
    in TriSignatureIdType signature,
    in TciParameterListType tciPars, in TciValueTemplate parsTmpl,
    in Value replValue, in TciValueTemplate replyTmpl,
    in TriComponentIdType from, in TciNonValueTemplate fromTmpl
);

void tliPrRaise_m(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in TriPortIdType at, in TriPortIdType to,
    in TriSignatureIdType signature, in TciParameterListType tciPars,
    in Value excValue, in Value addrValue, in TciStatusType encoderFailure,
    in TriExceptionType exc, in TriAddressType address, in TriStatusType transmissionFailure
);

void tliPrRaise_m_BC(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in TriPortIdType at, in TriPortIdType to,
    in TriSignatureIdType signature, in TciParameterListType tciPars,
    in Value excValue, in TciStatusType encoderFailure, in TriExceptionType exc,
    in TriStatusType transmissionFailure
);

void tliPrRaise_m_MC(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in TriPortIdType at, in TriPortIdType to,
    in TriSignatureIdType signature, in TciParameterListType tciPars,
    in Value excValue, in TciValueListType addrValues,
    in TciStatusType encoderFailure, in TriExceptionType exc,
    in TriAddressListType addresses, in TriStatusType transmissionFailure
);

void tliPrRaise_c(
    in TString am, in TInteger ts, in TString src, in TInteger line,

```



```

        in TriComponentIdType c, in TriPortIdType at, in TriPortIdType to,
        in TriSignatureIdType signature, in TciParameterListType tciPars,
        in Value excValue, in TriStatusType transmissionFailure
    );
void tliPrRaise_c_BC(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in TriPortIdType at, in TriPortIdListType to,
    in TriSignatureIdType signature, in TciParameterListType tciPars,
    in Value excValue, in TriStatusType transmissionFailure
);
void tliPrRaise_c_MC(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in TriPortIdType at, in TriPortIdListType to,
    in TriSignatureIdType signature, in TciParameterListType tciPars,
    in Value excValue, in TriStatusType transmissionFailure
);

void tliPrCatchDetected_m(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in TriPortIdType at, in TriPortIdType from,
    in TriSignatureIdType signature,
    in TriExceptionType exc, in TriAddressType address
);
void tliPrCatchDetected_c(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in TriPortIdType at, in TriPortIdType from,
    in TriSignatureIdType signature, in Value excValue
);
void tliPrCatchMismatch_m(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in TriPortIdType at,
    in TriSignatureIdType signature,
    in Value excValue, in TciValueTemplate excTmpl,
    in TciValueDifferenceList diffs, in Value addrValue,
    in TciValueTemplate addressTmpl
);
void tliPrCatchMismatch_c(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in TriPortIdType at,
    in TriSignatureIdType signature,
    in Value excValue, in TciValueTemplate excTmpl,
    in TciValueDifferenceList diffs, in TriComponentIdType from,
    in TciNonValueTemplate fromTmpl
);
void tliPrCatch_m(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in TriPortIdType at,
    in TriSignatureIdType signature,
    in Value excValue, in TciValueTemplate excTmpl,
    in Value addrValue, in TciValueTemplate addressTmpl);

void tliPrCatch_c(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in TriPortIdType at,
    in TriSignatureIdType signature,
    in Value excValue, in TciValueTemplate excTmpl,
    in TriComponentIdType from, in TciNonValueTemplate fromTmpl
);
void tliPrCatchTimeoutDetected(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in TriPortIdType at,
    in TriSignatureIdType signature
);
void tliPrCatchTimeout(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in TriPortIdType at,
    in TriSignatureIdType signature
);
void tliCCreate(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in TriComponentIdType comp,
    in TString name, in TBoolean alive
);
void tlicStart(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in TriComponentIdType comp,
    in TciBehaviourIdType name, in TciParameterListType tciPars
);
void tlicRunning(
    in TString am, in TInteger ts, in TString src, in TInteger line,

```

```

        in TriComponentIdType c, in TriComponentIdType comp, in ComponentStatusType status
    );
void tliCAlive(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c,
    in TriComponentIdType comp, in ComponentStatusType status
);
void tliCStop(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in TriComponentIdType comp
);
void tliCKill(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in TriComponentIdType comp
);
void tliCDoneMismatch(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in TriComponentIdType comp, in TciNonValueTemplate compTpl
);
void tliCKilledMismatch(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in TriComponentIdType comp, in TciNonValueTemplate compTpl
);
void tliCDone(in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in TciNonValueTemplate compTpl
);
void tliCKilled(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in TciNonValueTemplate compTpl
);
void tliCTerminated(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in VerdictValue verdict, in TString reason
);
void tliPConnect(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in TriPortIdType port1, in TriPortIdType port2
);
void tliPDisconnect(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in TriPortIdType port1,
    in TriPortIdType port2
);
void tliPMap(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in TriPortIdType port1, in TriPortIdType port2
);
void tliPUnmap(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in TriPortIdType port1,
    in TriPortIdType port2
);
void tliPClear(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in TriPortIdType port
);
void tliPStart(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in TriPortIdType port
);
void tliPStop(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in TriPortIdType port
);
void tliPHalt(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in TriPortIdType port
);
void tliEncode(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in Value val, in TciStatusType encoderFailure,
    in TriMessageType msg, in TString codec
);
void tliDecode(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in TriMessageType msg,
    in TciStatusType decoderFailure, in Value val, in TString codec
);
void tliTTimeoutDetected(
    in TString am, in TInteger ts, in TString src, in TInteger line,

```

```

        in TriComponentIdType c, in TriTimerIdType timer
    );
void tliTimeoutMismatch(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in TriTimerIdType timer, in TciNonValueTemplate timerTpl
);
void tliTimeout(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in TriTimerIdType timer, in TciNonValueTemplate timerTpl
);
void tliTStart(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in TriTimerIdType timer,
    in TriTimerDurationType dur
);
void tliTStop(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in TriTimerIdType timer, in TriTimerDurationType dur
);
void tliTRead(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in TriTimerIdType timer,
    in TriTimerDurationType elapsed
);
void tliTRunning(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in TriTimerIdType timer, in TimerStatusType status
);
void tliSEnter(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in QualifiedName name, in TciParameterListType tciPars,
    in TString kind
);
void tliSLeave(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in QualifiedName name, in TciParameterListType tciPars,
    in Value returnValue, in TString kind
);
void tliVar(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in QualifiedName name, in Value varValue
);
void tliModulePar(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in QualifiedName name, in Value parValue
);
void tliGetVerdict(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in VerdictValue verdict
);
void tliSetVerdict(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in VerdictValue verdict, in TString reason
);
void tliLog(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in TString log
);
void tliAEnter(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c
);
void tliALeave(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c
);
void tliADefaults(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c
);
void tliAActivate(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in QualifiedName name, in TciParameterListType tciPars,
    in Value ref
);
void tliAdeactivate(
    in TString am, in TInteger ts, in TString src, in TInteger line,
    in TriComponentIdType c, in Value ref
);
void tliANomatch(

```



## Annex B

### XML mapping for TCI TL provided

(This annex forms an integral part of this Recommendation.)

This annex defines a mapping for the logging interface of TCI using extensible markup language (XML) schema definitions.

#### B.1 TCI-TL XML schema for simple types

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://uri.etsi.org/ttcn-3/tci/SimpleTypes_v4_4_1.xsd"
  xmlns:SimpleTypes="http://uri.etsi.org/ttcn-3/tci/SimpleTypes_v4_4_1.xsd"
  elementFormDefault="qualified">

  <!-- Basic definitions -->
  <xsd:simpleType name="xpath">
    <!-- this string should be XPATH compliant -->
    <xsd:restriction base="xsd:string"/>
  </xsd:simpleType>

  <xsd:simpleType name="TBoolean">
    <xsd:restriction base="xsd:boolean"/>
  </xsd:simpleType>

  <xsd:simpleType name="TString">
    <xsd:restriction base="xsd:string"/>
  </xsd:simpleType>

  <xsd:simpleType name="TInteger">
    <xsd:restriction base="xsd:integer"/>
  </xsd:simpleType>

  <!-- Miscellaneous -->
  <xsd:simpleType name="TriTimerDurationType">
    <xsd:restriction base="xsd:float"/>
  </xsd:simpleType>

  <xsd:simpleType name="TciParameterPassingModeType">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="in"/>
      <xsd:enumeration value="inout"/>
      <xsd:enumeration value="out"/>
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:simpleType name="TriStatusType">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="TRI_Ok"/>
      <xsd:enumeration value="TRI_Error"/>
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:simpleType name="TciStatusType">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="TCI_Ok"/>
      <xsd:enumeration value="TCI_Error"/>
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:simpleType name="ComponentStatusType">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="inactiveC"/>
      <xsd:enumeration value="runningC"/>
      <xsd:enumeration value="stoppedC"/>
      <xsd:enumeration value="killedC"/>
      <xsd:enumeration value="nullC"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:schema>
```

```

<xsd:simpleType name="TimerStatusType">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="runningT"/>
    <xsd:enumeration value="inactiveT"/>
    <xsd:enumeration value="expiredT"/>
    <xsd:enumeration value="nullT"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="PortStatusType">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="startedP"/>
    <xsd:enumeration value="haltedP"/>
    <xsd:enumeration value="stoppedP"/>
  </xsd:restriction>
</xsd:simpleType>
</xsd:schema>

```

## B.2 TCI-TL XML schema for types

```

<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://uri.etsi.org/ttcn-3/tci/Types_v4_4_.xsd"
  xmlns:Types="http://uri.etsi.org/ttcn-3/tci/Types_v4_4_.xsd"
  xmlns:SimpleTypes="http://uri.etsi.org/ttcn-3/tci/SimpleTypes_v4_4_.xsd"
  xmlns:Values="http://uri.etsi.org/ttcn-3/tci/Values_v4_4_.xsd"
  xmlns:Templates="http://uri.etsi.org/ttcn-3/tci/Templates_v4_4_.xsd"
  elementFormDefault="qualified">

  <xsd:import namespace="http://uri.etsi.org/ttcn-3/tci/Values_v4_4_.xsd"
    schemaLocation="Values_v4_4_.xsd"/>
  <xsd:import namespace="http://uri.etsi.org/ttcn-3/tci/SimpleTypes_v4_4_1.xsd"
    schemaLocation="SimpleTypes_v4_4_1.xsd"/>
  <xsd:import namespace="http://uri.etsi.org/ttcn-3/tci/Templates_v4_4_1.xsd"
    schemaLocation="Templates_v4_4_1.xsd"/>

  <!-- Connection -->
  <xsd:complexType name="TriPortIdType">
    <xsd:sequence>
      <xsd:element name="comp" type="Types:TriComponentIdType" />
      <xsd:element name="port" type="Types:Port" />
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="TriPortIdListType">
    <xsd:sequence>
      <xsd:element name="comp" type="Types:TriPortIdType" minOccurs="0"
        maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="Port">
    <xsd:sequence>
      <xsd:element name="id" type="Types:Id" />
      <xsd:element name="index" type="xsd:int" minOccurs="0" />
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="TriComponentIdType">
    <xsd:sequence>
      <xsd:choice>
        <xsd:element name="null" type="Templates:null"/>
        <xsd:element name="id" type="Types:Id" />
      </xsd:choice>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="TriComponentIdListType">
    <xsd:sequence>
      <xsd:element name="comp" type="Types:TriComponentIdType" minOccurs="0"
        maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>

  <!-- Communication -->
  <xsd:complexType name="TriMessageType">
    <xsd:attribute name="val" type="xsd:hexBinary"/>
  </xsd:complexType>

```

```

<xsd:complexType name="TriParameterType">
  <xsd:sequence>
    <xsd:element name="val" type="xsd:hexBinary" />
  </xsd:sequence>
  <xsd:attribute name="name" type="SimpleTypes:TString"/>
  <xsd:attribute name="mode" type="SimpleTypes:TciParameterPassingModeType"/>
</xsd:complexType>

<xsd:complexType name="TriParameterListType">
  <xsd:sequence>
    <xsd:element name="par" type="Types:TriParameterType" minOccurs="0"
maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="TriExceptionType">
  <xsd:attribute name="val" type="xsd:hexBinary"/>
</xsd:complexType>

<xsd:complexType name="TciValueListType">
  <xsd:complexContent>
    <xsd:extension base="Values:RecordValue"/>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="TriSignatureIdType">
  <xsd:attribute name="val" type="SimpleTypes:TString" use="required"/>
</xsd:complexType>

<xsd:complexType name="TriAddressType">
  <xsd:attribute name="val" type="xsd:hexBinary"/>
</xsd:complexType>

<xsd:complexType name="TriAddressListType">
  <xsd:sequence>
    <xsd:element name="addr" type="Types:TriAddressType" minOccurs="0"
maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>

<!-- Miscellaneous -->
<xsd:complexType name="TriTimerIdType">
  <xsd:sequence>
    <xsd:element name="id" type="Types:Id" />
  </xsd:sequence>
</xsd:complexType>

<!-- Basic definitions -->
<xsd:complexType name="QualifiedName">
  <xsd:attribute name="moduleName" type="SimpleTypes:TString" use="required"/>
  <xsd:attribute name="baseName" type="SimpleTypes:TString" use="required"/>
</xsd:complexType>

<!-- general TCI abstract data types -->
<xsd:complexType name="TciBehaviourIdType">
  <xsd:sequence>
    <xsd:element name="name" type="Types:QualifiedName" />
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="TciTestCaseIdType">
  <xsd:sequence>
    <xsd:element name="name" type="Types:QualifiedName" />
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="TciParameterType">
  <xsd:sequence>
    <xsd:element name="val" type="Values:Value" />
  </xsd:sequence>
  <xsd:attribute name="name" type="SimpleTypes:TString"/>
  <xsd:attribute name="mode" type="SimpleTypes:TciParameterPassingModeType"/>
</xsd:complexType>

<xsd:complexType name="TciParameterListType">
  <xsd:sequence>
    <xsd:element name="par" type="Types:TciParameterType" minOccurs="0"
maxOccurs="unbounded"/>

```

```

        </xsd:sequence>
    </xsd:complexType>

    <!-- general identifier structure for test components, ports and timer -->
    <xsd:complexType name="Id">
        <xsd:sequence>
            <xsd:element name="name" type="SimpleTypes:TString" />
            <xsd:element name="id" type="SimpleTypes:TString" minOccurs="0"/>
            <xsd:element name="type" type="SimpleTypes:TString" minOccurs="0"/>
        </xsd:sequence>
    </xsd:complexType>

</xsd:schema>

```

### B.3 TCI-TL XML schema for values

```

<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://uri.etsi.org/ttcn-3/tci/Values_v4_4_1.xsd"
    xmlns:Values="http://uri.etsi.org/ttcn-3/tci/Values_v4_4_1.xsd"
    xmlns:Templates="http://uri.etsi.org/ttcn-3/tci/Templates_v4_4_1.xsd"
    xmlns:SimpleTypes="http://uri.etsi.org/ttcn-3/tci/SimpleTypes_v4_4_1.xsd"
    elementFormDefault="qualified">

    <xsd:import namespace="http://uri.etsi.org/ttcn-3/tci/Templates_v4_4_1.xsd"
        schemaLocation="Templates_v4_4_1.xsd"/>
    <xsd:import namespace="http://uri.etsi.org/ttcn-3/tci/SimpleTypes_v4_4_1.xsd"
        schemaLocation="SimpleTypes_v4_4_1.xsd"/>

    <xsd:attributeGroup name="ValueAtts">
        <xsd:attribute name="name" type="SimpleTypes:TString" use="optional"/>
        <xsd:attribute name="type" type="SimpleTypes:TString" use="optional"/>
        <xsd:attribute name="module" type="SimpleTypes:TString" use="optional"/>
        <xsd:attribute name="annotation" type="SimpleTypes:TString" use="optional"/>
    </xsd:attributeGroup>

    <xsd:complexType name="Value" mixed="true">
        <xsd:choice>
            <xsd:element name="integer" type="Values:IntegerValue"/>
            <xsd:element name="float" type="Values:FloatValue"/>
            <xsd:element name="boolean" type="Values:BooleanValue"/>
            <xsd:element name="verdicttype" type="Values:VerdictValue"/>
            <xsd:element name="bitstring" type="Values:BitstringValue"/>
            <xsd:element name="hexstring" type="Values:HexstringValue"/>
            <xsd:element name="octetstring" type="Values:OctetstringValue"/>
            <xsd:element name="charstring" type="Values:CharstringValue"/>
            <xsd:element name="universal_charstring" type="Values:UniversalCharstringValue"/>
            <xsd:element name="record" type="Values:RecordValue"/>
            <xsd:element name="record_of" type="Values:RecordOfValue"/>
            <xsd:element name="array" type="Values:ArrayValue"/>
            <xsd:element name="set" type="Values:SetValue"/>
            <xsd:element name="set_of" type="Values:SetOfValue"/>
            <xsd:element name="enumerated" type="Values:EnumeratedValue"/>
            <xsd:element name="union" type="Values:UnionValue"/>
            <xsd:element name="anytype" type="Values:AnytypeValue"/>
            <xsd:element name="address" type="Values:AddressValue"/>
        </xsd:choice>
        <xsd:attributeGroup ref="Values:ValueAtts"/>
    </xsd:complexType>

    <!-- general event elements -->
    <xsd:complexType name="IntegerValue">
        <xsd:choice>
            <xsd:element name="value" type="SimpleTypes:TString"/>
            <xsd:element name="null" type="Templates:null"/>
            <xsd:element name="omit" type="Templates:omit"/>
        </xsd:choice>
        <xsd:attributeGroup ref="Values:ValueAtts"/>
    </xsd:complexType>

    <xsd:complexType name="FloatValue">
        <xsd:choice>
            <xsd:element name="value" type="SimpleTypes:TString"/>
            <xsd:element name="null" type="Templates:null"/>
            <xsd:element name="omit" type="Templates:omit"/>
        </xsd:choice>
        <xsd:attributeGroup ref="Values:ValueAtts"/>
    </xsd:complexType>

```



```

<xsd:complexType name="BooleanValue">
  <xsd:choice>
    <xsd:element name="value" type="SimpleTypes:TString"/>
    <xsd:element name="null" type="Templates:null"/>
    <xsd:element name="omit" type="Templates:omit"/>
  </xsd:choice>
  <xsd:attributeGroup ref="Values:ValueAtts"/>
</xsd:complexType>

<xsd:complexType name="VerdictValue">
  <xsd:choice>
    <xsd:element name="value" type="SimpleTypes:TString"/>
    <xsd:element name="null" type="Templates:null"/>
    <xsd:element name="omit" type="Templates:omit"/>
  </xsd:choice>
  <xsd:attributeGroup ref="Values:ValueAtts"/>
</xsd:complexType>

<xsd:complexType name="BitstringValue">
  <xsd:choice>
    <xsd:element name="value" type="SimpleTypes:TString"/>
    <xsd:element name="null" type="Templates:null"/>
    <xsd:element name="omit" type="Templates:omit"/>
  </xsd:choice>
  <xsd:attributeGroup ref="Values:ValueAtts"/>
</xsd:complexType>

<xsd:complexType name="HexstringValue">
  <xsd:choice>
    <xsd:element name="value" type="SimpleTypes:TString"/>
    <xsd:element name="null" type="Templates:null"/>
    <xsd:element name="omit" type="Templates:omit"/>
  </xsd:choice>
  <xsd:attributeGroup ref="Values:ValueAtts"/>
</xsd:complexType>

<xsd:complexType name="OctetstringValue">
  <xsd:choice>
    <xsd:element name="value" type="SimpleTypes:TString"/>
    <xsd:element name="null" type="Templates:null"/>
    <xsd:element name="omit" type="Templates:omit"/>
  </xsd:choice>
  <xsd:attributeGroup ref="Values:ValueAtts"/>
</xsd:complexType>

<xsd:complexType name="CharstringValue">
  <xsd:choice>
    <xsd:element name="value" type="SimpleTypes:TString"/>
    <xsd:element name="null" type="Templates:null"/>
    <xsd:element name="omit" type="Templates:omit"/>
  </xsd:choice>
  <xsd:attributeGroup ref="Values:ValueAtts"/>
</xsd:complexType>

<xsd:complexType name="UniversalCharstringValue">
  <xsd:choice>
    <xsd:element name="value" type="SimpleTypes:TString"/>
    <xsd:element name="null" type="Templates:null"/>
    <xsd:element name="omit" type="Templates:omit"/>
  </xsd:choice>
  <xsd:attributeGroup ref="Values:ValueAtts"/>
</xsd:complexType>

<xsd:complexType name="RecordValue">
  <xsd:choice>
    <xsd:choice minOccurs="0" maxOccurs="unbounded">
      <xsd:element name="integer" type="Values:IntegerValue"/>
      <xsd:element name="float" type="Values:FloatValue"/>
      <xsd:element name="boolean" type="Values:BooleanValue"/>
      <xsd:element name="verdicttype" type="Values:VerdictValue"/>
      <xsd:element name="bitstring" type="Values:BitstringValue"/>
      <xsd:element name="hexstring" type="Values:HexstringValue"/>
      <xsd:element name="octetstring" type="Values:OctetstringValue"/>
      <xsd:element name="charstring" type="Values:CharstringValue"/>
      <xsd:element name="universal_charstring"
        type="Values:UniversalCharstringValue"/>
      <xsd:element name="record" type="Values:RecordValue"/>
      <xsd:element name="record_of" type="Values:RecordOfValue"/>
      <xsd:element name="array" type="Values:ArrayValue"/>
    </xsd:choice>
  </xsd:choice>

```

```

        <xsd:element name="set" type="Values:SetValue"/>
        <xsd:element name="set_of" type="Values:SetOfValue"/>
        <xsd:element name="enumerated" type="Values:EnumeratedValue"/>
        <xsd:element name="union" type="Values:UnionValue"/>
        <xsd:element name="anytype" type="Values:AnytypeValue"/>
        <xsd:element name="address" type="Values:AddressValue"/>
    </xsd:choice>
    <xsd:element name="null" type="Templates:null"/>
    <xsd:element name="omit" type="Templates:omit"/>
</xsd:choice>
<xsd:attributeGroup ref="Values:ValueAtts"/>
</xsd:complexType>

<xsd:complexType name="RecordOfValue">
    <xsd:choice>
        <xsd:element name="integer" type="Values:IntegerValue" minOccurs="0"
            maxOccurs="unbounded"/>
        <xsd:element name="float" type="Values:FloatValue" minOccurs="0"
            maxOccurs="unbounded"/>
        <xsd:element name="boolean" type="Values:BooleanValue" minOccurs="0"
            maxOccurs="unbounded"/>
        <xsd:element name="verdicttype" type="Values:VerdictValue" minOccurs="0"
            maxOccurs="unbounded"/>
        <xsd:element name="bitstring" type="Values:BitstringValue"
            minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="hexstring" type="Values:HexstringValue"
            minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="octetstring" type="Values:OctetstringValue"
            minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="charstring" type="Values:CharstringValue"
            minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="universal_charstring"
            type="Values:UniversalCharstringValue" minOccurs="0"
            maxOccurs="unbounded"/>
        <xsd:element name="record" type="Values:RecordValue" minOccurs="0"
            maxOccurs="unbounded"/>
        <xsd:element name="record_of" type="Values:RecordOfValue"
            minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="array" type="Values:ArrayValue"
            minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="set" type="Values:SetValue" minOccurs="0"
            maxOccurs="unbounded"/>
        <xsd:element name="set_of" type="Values:SetOfValue"
            minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="enumerated" type="Values:EnumeratedValue"
            minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="union" type="Values:UnionValue" minOccurs="0"
            maxOccurs="unbounded"/>
        <xsd:element name="anytype" type="Values:AnytypeValue" minOccurs="0"
            maxOccurs="unbounded"/>
        <xsd:element name="address" type="Values:AddressValue" minOccurs="0"
            maxOccurs="unbounded"/>
        <xsd:element name="null" type="Templates:null"/>
        <xsd:element name="omit" type="Templates:omit"/>
    </xsd:choice>
    <xsd:attributeGroup ref="Values:ValueAtts"/>
</xsd:complexType>

<xsd:complexType name="ArrayValue">
    <xsd:choice>
        <xsd:element name="integer" type="Values:IntegerValue" minOccurs="0"
            maxOccurs="unbounded"/>
        <xsd:element name="float" type="Values:FloatValue" minOccurs="0"
            maxOccurs="unbounded"/>
        <xsd:element name="boolean" type="Values:BooleanValue" minOccurs="0"
            maxOccurs="unbounded"/>
        <xsd:element name="verdicttype" type="Values:VerdictValue" minOccurs="0"
            maxOccurs="unbounded"/>
        <xsd:element name="bitstring" type="Values:BitstringValue"
            minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="hexstring" type="Values:HexstringValue"
            minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="octetstring" type="Values:OctetstringValue"
            minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="charstring" type="Values:CharstringValue"
            minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="universal_charstring"
            type="Values:UniversalCharstringValue" minOccurs="0"
            maxOccurs="unbounded"/>
    </xsd:choice>
</xsd:complexType>

```

```

<xsd:element name="record" type="Values:RecordValue" minOccurs="0"
  maxOccurs="unbounded"/>
<xsd:element name="record_of" type="Values:RecordOfValue"
  minOccurs="0" maxOccurs="unbounded"/>
<xsd:element name="array" type="Values:ArrayValue"
  minOccurs="0" maxOccurs="unbounded"/>
<xsd:element name="set" type="Values:SetValue" minOccurs="0"
  maxOccurs="unbounded"/>
<xsd:element name="set_of" type="Values:SetOfValue"
  minOccurs="0" maxOccurs="unbounded"/>
<xsd:element name="enumerated" type="Values:EnumeratedValue"
  minOccurs="0" maxOccurs="unbounded"/>
<xsd:element name="union" type="Values:UnionValue" minOccurs="0"
  maxOccurs="unbounded"/>
<xsd:element name="anytype" type="Values:AnytypeValue" minOccurs="0"
  maxOccurs="unbounded"/>
<xsd:element name="address" type="Values:AddressValue" minOccurs="0"
  maxOccurs="unbounded"/>
<xsd:element name="null" type="Templates:null"/>
<xsd:element name="omit" type="Templates:omit"/>
</xsd:choice>
<xsd:attributeGroup ref="Values:ValueAtts"/>
</xsd:complexType>

<xsd:complexType name="SetValue">
  <xsd:choice>
    <xsd:choice minOccurs="0" maxOccurs="unbounded">
      <xsd:element name="integer" type="Values:IntegerValue"/>
      <xsd:element name="float" type="Values:FloatValue"/>
      <xsd:element name="boolean" type="Values:BooleanValue"/>
      <xsd:element name="verdicttype" type="Values:VerdictValue"/>
      <xsd:element name="bitstring" type="Values:BitstringValue"/>
      <xsd:element name="hexstring" type="Values:HexstringValue"/>
      <xsd:element name="octetstring" type="Values:OctetstringValue"/>
      <xsd:element name="charstring" type="Values:CharstringValue"/>
      <xsd:element name="universal_charstring"
        type="Values:UniversalCharstringValue"/>
      <xsd:element name="record" type="Values:RecordValue"/>
      <xsd:element name="record_of" type="Values:RecordOfValue"/>
      <xsd:element name="array" type="Values:ArrayValue"/>
      <xsd:element name="set" type="Values:SetValue"/>
      <xsd:element name="set_of" type="Values:SetOfValue"/>
      <xsd:element name="enumerated" type="Values:EnumeratedValue"/>
      <xsd:element name="union" type="Values:UnionValue"/>
      <xsd:element name="anytype" type="Values:AnytypeValue"/>
      <xsd:element name="address" type="Values:AddressValue"/>
    </xsd:choice>
    <xsd:element name="null" type="Templates:null"/>
    <xsd:element name="omit" type="Templates:omit"/>
  </xsd:choice>
  <xsd:attributeGroup ref="Values:ValueAtts"/>
</xsd:complexType>

<xsd:complexType name="SetOfValue">
  <xsd:choice>
    <xsd:element name="integer" type="Values:IntegerValue" minOccurs="0"
      maxOccurs="unbounded"/>
    <xsd:element name="float" type="Values:FloatValue" minOccurs="0"
      maxOccurs="unbounded"/>
    <xsd:element name="boolean" type="Values:BooleanValue" minOccurs="0"
      maxOccurs="unbounded"/>
    <xsd:element name="verdicttype" type="Values:VerdictValue" minOccurs="0"
      maxOccurs="unbounded"/>
    <xsd:element name="bitstring" type="Values:BitstringValue"
      minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element name="hexstring" type="Values:HexstringValue"
      minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element name="octetstring" type="Values:OctetstringValue"
      minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element name="charstring" type="Values:CharstringValue"
      minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element name="universal_charstring"
      type="Values:UniversalCharstringValue" minOccurs="0"
      maxOccurs="unbounded"/>
    <xsd:element name="record" type="Values:RecordValue" minOccurs="0"
      maxOccurs="unbounded"/>
    <xsd:element name="record_of" type="Values:RecordOfValue"
      minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element name="array" type="Values:ArrayValue"
      minOccurs="0" maxOccurs="unbounded"/>
  </xsd:choice>

```

```

    <xsd:element name="set" type="Values:SetValue" minOccurs="0"
      maxOccurs="unbounded"/>
    <xsd:element name="set_of" type="Values:SetOfValue"
      minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element name="enumerated" type="Values:EnumeratedValue"
      minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element name="union" type="Values:UnionValue" minOccurs="0"
      maxOccurs="unbounded"/>
    <xsd:element name="anytype" type="Values:AnytypeValue" minOccurs="0"
      maxOccurs="unbounded"/>
    <xsd:element name="address" type="Values:AddressValue" minOccurs="0"
      maxOccurs="unbounded"/>
    <xsd:element name="null" type="Templates:null"/>
    <xsd:element name="omit" type="Templates:omit"/>
  </xsd:choice>
  <xsd:attributeGroup ref="Values:ValueAtts"/>
</xsd:complexType>

<xsd:complexType name="EnumeratedValue">
  <xsd:choice>
    <xsd:element name="value" type="SimpleTypes:TString"/>
    <xsd:element name="intValue" type="SimpleTypes:TInteger" minOccurs="0"/>
    <xsd:element name="null" type="Templates:null"/>
    <xsd:element name="omit" type="Templates:omit"/>
  </xsd:choice>
  <xsd:attributeGroup ref="Values:ValueAtts"/>
</xsd:complexType>

<xsd:complexType name="UnionValue">
  <xsd:choice>
    <xsd:element name="integer" type="Values:IntegerValue"/>
    <xsd:element name="float" type="Values:FloatValue"/>
    <xsd:element name="boolean" type="Values:BooleanValue"/>
    <xsd:element name="verdicttype" type="Values:VerdictValue"/>
    <xsd:element name="bitstring" type="Values:BitstringValue"/>
    <xsd:element name="hexstring" type="Values:HexstringValue"/>
    <xsd:element name="octetstring" type="Values:OctetstringValue"/>
    <xsd:element name="charstring" type="Values:CharstringValue"/>
    <xsd:element name="universal_charstring"
      type="Values:UniversalCharstringValue"/>
    <xsd:element name="record" type="Values:RecordValue"/>
    <xsd:element name="record_of" type="Values:RecordOfValue"/>
    <xsd:element name="array" type="Values:ArrayValue"/>
    <xsd:element name="set" type="Values:SetValue"/>
    <xsd:element name="set_of" type="Values:SetOfValue"/>
    <xsd:element name="enumerated" type="Values:EnumeratedValue"/>
    <xsd:element name="union" type="Values:UnionValue"/>
    <xsd:element name="anytype" type="Values:AnytypeValue"/>
    <xsd:element name="address" type="Values:AddressValue"/>
    <xsd:element name="null" type="Templates:null"/>
    <xsd:element name="omit" type="Templates:omit"/>
  </xsd:choice>
  <xsd:attributeGroup ref="Values:ValueAtts"/>
</xsd:complexType>

<xsd:complexType name="AnytypeValue">
  <xsd:choice minOccurs="0" maxOccurs="unbounded">
    <xsd:element name="integer" type="Values:IntegerValue"/>
    <xsd:element name="float" type="Values:FloatValue"/>
    <xsd:element name="boolean" type="Values:BooleanValue"/>
    <xsd:element name="verdicttype" type="Values:VerdictValue"/>
    <xsd:element name="bitstring" type="Values:BitstringValue"/>
    <xsd:element name="hexstring" type="Values:HexstringValue"/>
    <xsd:element name="octetstring" type="Values:OctetstringValue"/>
    <xsd:element name="charstring" type="Values:OctetstringValue"/>
    <xsd:element name="universal_charstring"
      type="Values:UniversalCharstringValue"/>
    <xsd:element name="record" type="Values:RecordValue"/>
    <xsd:element name="record_of" type="Values:RecordOfValue"/>
    <xsd:element name="array" type="Values:ArrayValue"/>
    <xsd:element name="set" type="Values:SetValue"/>
    <xsd:element name="set_of" type="Values:SetOfValue"/>
    <xsd:element name="enumerated" type="Values:EnumeratedValue"/>
    <xsd:element name="union" type="Values:UnionValue"/>
    <xsd:element name="address" type="Values:AddressValue"/>
    <xsd:element name="null" type="Templates:null"/>
    <xsd:element name="omit" type="Templates:omit"/>
  </xsd:choice>
  <xsd:attributeGroup ref="Values:ValueAtts"/>
</xsd:complexType>

```

```

<xsd:complexType name="AddressValue">
  <xsd:choice minOccurs="0" maxOccurs="unbounded">
    <xsd:element name="integer" type="Values:IntegerValue"/>
    <xsd:element name="float" type="Values:FloatValue"/>
    <xsd:element name="boolean" type="Values:BooleanValue"/>
    <xsd:element name="verdicttype" type="Values:VerdictValue"/>
    <xsd:element name="bitstring" type="Values:BitstringValue"/>
    <xsd:element name="hexstring" type="Values:HexstringValue"/>
    <xsd:element name="octetstring" type="Values:OctetstringValue"/>
    <xsd:element name="charstring" type="Values:OctetstringValue"/>
    <xsd:element name="universal_charstring"
      type="Values:UniversalCharstringValue"/>
    <xsd:element name="record" type="Values:RecordValue"/>
    <xsd:element name="record_of" type="Values:RecordOfValue"/>
    <xsd:element name="array" type="Values:ArrayValue"/>
    <xsd:element name="set" type="Values:SetValue"/>
    <xsd:element name="set_of" type="Values:SetOfValue"/>
    <xsd:element name="enumerated" type="Values:EnumeratedValue"/>
    <xsd:element name="union" type="Values:UnionValue"/>
    <xsd:element name="anytype" type="Values:AnytypeValue"/>
    <xsd:element name="null" type="Templates:null"/>
    <xsd:element name="omit" type="Templates:omit"/>
  </xsd:choice>
  <xsd:attributeGroup ref="Values:ValueAtts"/>
</xsd:complexType>
</xsd:schema>

```

## B.4 TCI-TL XML schema for templates

```

<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://uri.etsi.org/ttcn-3/tci/Templates_v4_4_1.xsd"
  xmlns:Templates="http://uri.etsi.org/ttcn-3/tci/Templates_v4_4_1.xsd"
  xmlns:Values="http://uri.etsi.org/ttcn-3/tci/Values_v4_4_1.xsd"
  xmlns:SimpleTypes="http://uri.etsi.org/ttcn-3/tci/SimpleTypes_v4_4_1.xsd"
  elementFormDefault="qualified">

  <xsd:import namespace="http://uri.etsi.org/ttcn-3/tci/Values_v4_4_1.xsd"
    schemaLocation="Values_v4_4_1.xsd"/>
  <xsd:import namespace="http://uri.etsi.org/ttcn-3/tci/SimpleTypes_v4_4_1.xsd"
    schemaLocation="SimpleTypes_v4_4_1.xsd"/>

  <xsd:complexType name="TciValueTemplate">
    <xsd:complexContent mixed="true">
      <xsd:extension base="Values:Value">
        <xsd:choice>
          <xsd:element name="integer" type="Templates:IntegerTemplate"/>
          <xsd:element name="float" type="Templates:FloatTemplate"/>
          <xsd:element name="boolean" type="Templates:BooleanTemplate"/>
          <xsd:element name="verdicttype" type="Templates:VerdictTemplate"/>
          <xsd:element name="bitstring" type="Templates:BitstringValue"/>
          <xsd:element name="hexstring" type="Templates:HexstringValue"/>
          <xsd:element name="octetstring" type="Templates:OctetstringValue"/>
          <xsd:element name="charstring" type="Templates:CharstringValue"/>
          <xsd:element name="universal_charstring"
            type="Templates:UniversalCharstringValue"/>
          <xsd:element name="record" type="Templates:RecordTemplate"/>
          <xsd:element name="record_of" type="Templates:RecordOfTemplate"/>
          <xsd:element name="array" type="Values:ArrayValue"/>
          <xsd:element name="set" type="Templates:SetTemplate"/>
          <xsd:element name="set_of" type="Templates:SetOfTemplate"/>
          <xsd:element name="enumerated" type="Templates:EnumeratedTemplate"/>
          <xsd:element name="union" type="Templates:UnionTemplate"/>
          <xsd:element name="anytype" type="Templates:AnytypeTemplate"/>
          <xsd:element name="address" type="Templates:AddressTemplate"/>
          <xsd:element name="omit" type="Templates:omit"/>
          <xsd:element name="any" type="Templates:any"/>
          <xsd:element name="anyoromit" type="Templates:anyoromit"/>
          <xsd:element name="templateDef" type="SimpleTypes:TString"/>
        </xsd:choice>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="omit">
    <xsd:attributeGroup ref="Values:ValueAtts"/>
  </xsd:complexType>

```

```

<xsd:complexType name="any">
  <xsd:attributeGroup ref="Values:ValueAtts"/>
</xsd:complexType>

<xsd:complexType name="anyoromit">
  <xsd:attributeGroup ref="Values:ValueAtts"/>
</xsd:complexType>

<xsd:complexType name="TciNonValueTemplate">
  <xsd:sequence>
    <xsd:choice>
      <xsd:element name="any" type="Templates:any"/>
      <xsd:element name="all" type="Templates:all"/>
      <xsd:element name="templateDef" type="SimpleTypes:TString"/>
      <xsd:element name="null" type="Templates:null"/>
    </xsd:choice>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="all">
  <xsd:attributeGroup ref="Values:ValueAtts"/>
</xsd:complexType>

<xsd:complexType name="null">
  <xsd:attributeGroup ref="Values:ValueAtts"/>
</xsd:complexType>

<xsd:complexType name="TciValueDifference">
  <xsd:sequence>
    <xsd:element name="val" type="SimpleTypes:xpath"/>
    <xsd:element name="tmpl" type="SimpleTypes:xpath"/>
  </xsd:sequence>
  <xsd:attributeGroup ref="Values:ValueAtts"/>
  <xsd:attribute name="desc" type="SimpleTypes:TString" use="optional"/>
</xsd:complexType>

<xsd:complexType name="TciValueDifferenceList">
  <xsd:sequence>
    <xsd:element name="diff" type="Templates:TciValueDifference"
maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="IntegerTemplate">
  <xsd:choice>
    <xsd:element name="value" type="SimpleTypes:TString"/>
    <xsd:element name="templateDef" type="SimpleTypes:TString"/>
    <xsd:element name="omit" type="Templates:omit"/>
    <xsd:element name="any" type="Templates:any"/>
    <xsd:element name="anyoromit" type="Templates:anyoromit"/>
    <xsd:element name="null" type="Templates:null"/>
  </xsd:choice>
  <xsd:attributeGroup ref="Values:ValueAtts"/>
</xsd:complexType>

<xsd:complexType name="FloatTemplate">
  <xsd:choice>
    <xsd:element name="value" type="SimpleTypes:TString"/>
    <xsd:element name="templateDef" type="SimpleTypes:TString"/>
    <xsd:element name="omit" type="Templates:omit"/>
    <xsd:element name="any" type="Templates:any"/>
    <xsd:element name="anyoromit" type="Templates:anyoromit"/>
    <xsd:element name="null" type="Templates:null"/>
  </xsd:choice>
  <xsd:attributeGroup ref="Values:ValueAtts"/>
</xsd:complexType>

<xsd:complexType name="BooleanTemplate">
  <xsd:choice>
    <xsd:element name="value" type="SimpleTypes:TString"/>
    <xsd:element name="templateDef" type="SimpleTypes:TString"/>
    <xsd:element name="omit" type="Templates:omit"/>
    <xsd:element name="any" type="Templates:any"/>
    <xsd:element name="anyoromit" type="Templates:anyoromit"/>
    <xsd:element name="null" type="Templates:null"/>
  </xsd:choice>
  <xsd:attributeGroup ref="Values:ValueAtts"/>
</xsd:complexType>

```

```

<xsd:complexType name="BitstringTemplate">
  <xsd:choice>
    <xsd:element name="value" type="SimpleTypes:TString"/>
    <xsd:element name="templateDef" type="SimpleTypes:TString"/>
    <xsd:element name="omit" type="Templates:omit"/>
    <xsd:element name="any" type="Templates:any"/>
    <xsd:element name="anyoromit" type="Templates:anyoromit"/>
    <xsd:element name="null" type="Templates:null"/>
  </xsd:choice>
  <xsd:attributeGroup ref="Values:ValueAtts"/>
</xsd:complexType>

<xsd:complexType name="HexstringTemplate">
  <xsd:choice>
    <xsd:element name="value" type="SimpleTypes:TString"/>
    <xsd:element name="templateDef" type="SimpleTypes:TString"/>
    <xsd:element name="omit" type="Templates:omit"/>
    <xsd:element name="any" type="Templates:any"/>
    <xsd:element name="anyoromit" type="Templates:anyoromit"/>
    <xsd:element name="null" type="Templates:null"/>
  </xsd:choice>
  <xsd:attributeGroup ref="Values:ValueAtts"/>
</xsd:complexType>

<xsd:complexType name="OctetstringTemplate">
  <xsd:choice>
    <xsd:element name="value" type="SimpleTypes:TString"/>
    <xsd:element name="templateDef" type="SimpleTypes:TString"/>
    <xsd:element name="omit" type="Templates:omit"/>
    <xsd:element name="any" type="Templates:any"/>
    <xsd:element name="anyoromit" type="Templates:anyoromit"/>
    <xsd:element name="null" type="Templates:null"/>
  </xsd:choice>
  <xsd:attributeGroup ref="Values:ValueAtts"/>
</xsd:complexType>

<xsd:complexType name="CharstringTemplate">
  <xsd:choice>
    <xsd:element name="value" type="SimpleTypes:TString"/>
    <xsd:element name="templateDef" type="SimpleTypes:TString"/>
    <xsd:element name="omit" type="Templates:omit"/>
    <xsd:element name="any" type="Templates:any"/>
    <xsd:element name="anyoromit" type="Templates:anyoromit"/>
    <xsd:element name="null" type="Templates:null"/>
  </xsd:choice>
  <xsd:attributeGroup ref="Values:ValueAtts"/>
</xsd:complexType>

<xsd:complexType name="UniversalCharstringTemplate">
  <xsd:choice>
    <xsd:element name="value" type="SimpleTypes:TString"/>
    <xsd:element name="templateDef" type="SimpleTypes:TString"/>
    <xsd:element name="omit" type="Templates:omit"/>
    <xsd:element name="any" type="Templates:any"/>
    <xsd:element name="anyoromit" type="Templates:anyoromit"/>
    <xsd:element name="null" type="Templates:null"/>
  </xsd:choice>
  <xsd:attributeGroup ref="Values:ValueAtts"/>
</xsd:complexType>

<xsd:complexType name="VerdictTemplate">
  <xsd:choice>
    <xsd:element name="value" type="SimpleTypes:TString"/>
    <xsd:element name="templateDef" type="SimpleTypes:TString"/>
    <xsd:element name="omit" type="Templates:omit"/>
    <xsd:element name="any" type="Templates:any"/>
    <xsd:element name="anyoromit" type="Templates:anyoromit"/>
    <xsd:element name="null" type="Templates:null"/>
  </xsd:choice>
  <xsd:attributeGroup ref="Values:ValueAtts"/>
</xsd:complexType>

<xsd:complexType name="RecordTemplate">
  <xsd:complexContent>
    <xsd:extension base="Values:RecordValue">
      <xsd:choice>
        <xsd:choice minOccurs="0">
          <xsd:element name="integer" type="Templates:IntegerTemplate"/>
          <xsd:element name="float" type="Templates:FloatTemplate"/>
        </xsd:choice>
      </xsd:choice>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

```

```

<xsd:element name="boolean" type="Templates:BooleanTemplate"/>
<xsd:element name="verdicttype" type="Templates:VerdictTemplate"/>
<xsd:element name="bitstring" type="Templates:BitstringTemplate"/>
<xsd:element name="hexstring" type="Templates:HexstringTemplate"/>
<xsd:element name="octetstring" type="Templates:OctetstringTemplate"/>
<xsd:element name="charstring" type="Templates:CharstringTemplate"/>
<xsd:element name="universal_charstring"
  type="Templates:UniversalCharstringTemplate"/>
<xsd:element name="record" type="Templates:RecordTemplate"/>
<xsd:element name="record_of" type="Templates:RecordOfTemplate"/>
<xsd:element name="array" type="Values:ArrayValue"/>
<xsd:element name="set" type="Templates:SetTemplate"/>
<xsd:element name="set_of" type="Templates:SetOfTemplate"/>
<xsd:element name="enumerated" type="Templates:EnumeratedTemplate"/>
<xsd:element name="union" type="Templates:UnionTemplate"/>
<xsd:element name="anytype" type="Templates:AnytypeTemplate"/>
<xsd:element name="address" type="Templates:AddressTemplate"/>
</xsd:choice>
<xsd:element name="omit" type="Templates:omit"/>
<xsd:element name="any" type="Templates:any"/>
<xsd:element name="anyoromit" type="Templates:anyoromit"/>
<xsd:element name="templateDef" type="SimpleTypes:TString"/>
<xsd:element name="null" type="Templates:null"/>
</xsd:choice>
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="RecordOfTemplate">
  <xsd:complexContent>
    <xsd:extension base="Values:RecordOfValue">
      <xsd:choice minOccurs="0" maxOccurs="unbounded">
        <xsd:element name="integer" type="Templates:IntegerTemplate" minOccurs="0"
          maxOccurs="unbounded"/>
        <xsd:element name="float" type="Templates:FloatTemplate" minOccurs="0"
          maxOccurs="unbounded"/>
        <xsd:element name="boolean" type="Templates:BooleanTemplate" minOccurs="0"
          maxOccurs="unbounded"/>
        <xsd:element name="verdicttype" type="Templates:VerdictTemplate" minOccurs="0"
          maxOccurs="unbounded"/>
        <xsd:element name="bitstring" type="Templates:BitstringTemplate"
          minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="hexstring" type="Templates:HexstringTemplate"
          minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="octetstring" type="Templates:OctetstringTemplate"
          minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="charstring" type="Templates:CharstringTemplate"
          minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="universal_charstring"
          type="Templates:UniversalCharstringTemplate" minOccurs="0"
          maxOccurs="unbounded"/>
        <xsd:element name="record" type="Templates:RecordTemplate" minOccurs="0"
          maxOccurs="unbounded"/>
        <xsd:element name="record_of" type="Templates:RecordOfTemplate"
          minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="array" type="Templates:ArrayTemplate" minOccurs="0"
          maxOccurs="unbounded"/>
        <xsd:element name="set" type="Templates:SetTemplate" minOccurs="0"
          maxOccurs="unbounded"/>
        <xsd:element name="set_of" type="Templates:SetOfTemplate"
          minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="enumerated" type="Templates:EnumeratedTemplate"
          minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="union" type="Templates:UnionTemplate" minOccurs="0"
          maxOccurs="unbounded"/>
        <xsd:element name="anytype" type="Templates:AnytypeTemplate" minOccurs="0"
          maxOccurs="unbounded"/>
        <xsd:element name="address" type="Templates:AddressTemplate" minOccurs="0"
          maxOccurs="unbounded"/>
        <xsd:element name="omit" type="Templates:omit"/>
        <xsd:element name="any" type="Templates:any"/>
        <xsd:element name="anyoromit" type="Templates:anyoromit"/>
        <xsd:element name="templateDef" type="SimpleTypes:TString"/>
        <xsd:element name="null" type="Templates:null"/>
      </xsd:choice>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="ArrayTemplate">

```



```

<xsd:complexContent>
  <xsd:extension base="Values:ArrayValue">
    <xsd:choice minOccurs="0" maxOccurs="unbounded">
      <xsd:element name="integer" type="Templates:IntegerTemplate" minOccurs="0"
        maxOccurs="unbounded"/>
      <xsd:element name="float" type="Templates:FloatTemplate" minOccurs="0"
        maxOccurs="unbounded"/>
      <xsd:element name="boolean" type="Templates:BooleanTemplate" minOccurs="0"
        maxOccurs="unbounded"/>
      <xsd:element name="verdicttype" type="Templates:VerdictTemplate" minOccurs="0"
        maxOccurs="unbounded"/>
      <xsd:element name="bitstring" type="Templates:BitstringTemplate"
        minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element name="hexstring" type="Templates:HexstringTemplate"
        minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element name="octetstring" type="Templates:OctetstringTemplate"
        minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element name="charstring" type="Templates:CharstringTemplate"
        minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element name="universal_charstring"
        type="Templates:UniversalCharstringTemplate" minOccurs="0"
        maxOccurs="unbounded"/>
      <xsd:element name="record" type="Templates:RecordTemplate" minOccurs="0"
        maxOccurs="unbounded"/>
      <xsd:element name="record_of" type="Templates:RecordOfTemplate"
        minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element name="array" type="Templates:ArrayTemplate" minOccurs="0"
        maxOccurs="unbounded"/>
      <xsd:element name="set" type="Templates:SetTemplate" minOccurs="0"
        maxOccurs="unbounded"/>
      <xsd:element name="set_of" type="Templates:SetOfTemplate"
        minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element name="enumerated" type="Templates:EnumeratedTemplate"
        minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element name="union" type="Templates:UnionTemplate" minOccurs="0"
        maxOccurs="unbounded"/>
      <xsd:element name="anytype" type="Templates:AnytypeTemplate" minOccurs="0"
        maxOccurs="unbounded"/>
      <xsd:element name="address" type="Templates:AddressTemplate" minOccurs="0"
        maxOccurs="unbounded"/>
      <xsd:element name="omit" type="Templates:omit"/>
      <xsd:element name="any" type="Templates:any"/>
      <xsd:element name="anyoromit" type="Templates:anyoromit"/>
      <xsd:element name="templateDef" type="SimpleTypes:TString"/>
      <xsd:element name="null" type="Templates:null"/>
    </xsd:choice>
  </xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="SetTemplate">
  <xsd:complexContent>
    <xsd:extension base="Values:SetValue">
      <xsd:choice>
        <xsd:choice minOccurs="0">
          <xsd:element name="integer" type="Templates:IntegerTemplate"/>
          <xsd:element name="float" type="Templates:FloatTemplate"/>
          <xsd:element name="boolean" type="Templates:BooleanTemplate"/>
          <xsd:element name="verdicttype" type="Templates:VerdictTemplate"/>
          <xsd:element name="bitstring" type="Templates:BitstringTemplate"/>
          <xsd:element name="hexstring" type="Templates:HexstringTemplate"/>
          <xsd:element name="octetstring" type="Templates:OctetstringTemplate"/>
          <xsd:element name="charstring" type="Templates:CharstringTemplate"/>
          <xsd:element name="universal_charstring"
            type="Templates:UniversalCharstringTemplate"/>
          <xsd:element name="record" type="Templates:RecordTemplate"/>
          <xsd:element name="record_of" type="Templates:RecordOfTemplate"/>
          <xsd:element name="array" type="Templates:ArrayTemplate"/>
          <xsd:element name="set" type="Templates:SetTemplate"/>
          <xsd:element name="set_of" type="Templates:SetOfTemplate"/>
          <xsd:element name="enumerated" type="Templates:EnumeratedTemplate"/>
          <xsd:element name="union" type="Templates:UnionTemplate"/>
          <xsd:element name="anytype" type="Templates:AnytypeTemplate"/>
          <xsd:element name="address" type="Templates:AddressTemplate"/>
        </xsd:choice>
        <xsd:element name="omit" type="Templates:omit"/>
        <xsd:element name="any" type="Templates:any"/>
        <xsd:element name="anyoromit" type="Templates:anyoromit"/>
        <xsd:element name="templateDef" type="SimpleTypes:TString"/>
        <xsd:element name="null" type="Templates:null"/>
      </xsd:choice>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

```

```

        </xsd:choice>
    </xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="SetOfTemplate">
    <xsd:complexContent>
        <xsd:extension base="Values:SetOfValue">
            <xsd:choice minOccurs="0" maxOccurs="unbounded">
                <xsd:element name="integer" type="Templates:IntegerTemplate" minOccurs="0"
                    maxOccurs="unbounded"/>
                <xsd:element name="float" type="Templates:FloatTemplate" minOccurs="0"
                    maxOccurs="unbounded"/>
                <xsd:element name="boolean" type="Templates:BooleanTemplate" minOccurs="0"
                    maxOccurs="unbounded"/>
                <xsd:element name="verdicttype" type="Templates:VerdictTemplate" minOccurs="0"
                    maxOccurs="unbounded"/>
                <xsd:element name="bitstring" type="Templates:BitstringTemplate"
                    minOccurs="0" maxOccurs="unbounded"/>
                <xsd:element name="hexstring" type="Templates:HexstringTemplate"
                    minOccurs="0" maxOccurs="unbounded"/>
                <xsd:element name="octetstring" type="Templates:OctetstringTemplate"
                    minOccurs="0" maxOccurs="unbounded"/>
                <xsd:element name="charstring" type="Templates:CharstringTemplate"
                    minOccurs="0" maxOccurs="unbounded"/>
                <xsd:element name="universal_charstring"
                    type="Templates:UniversalCharstringTemplate" minOccurs="0"
                    maxOccurs="unbounded"/>
                <xsd:element name="record" type="Templates:RecordTemplate" minOccurs="0"
                    maxOccurs="unbounded"/>
                <xsd:element name="record_of" type="Templates:RecordOfTemplate"
                    minOccurs="0" maxOccurs="unbounded"/>
                <xsd:element name="array" type="Templates:ArrayTemplate" minOccurs="0"
                    maxOccurs="unbounded"/>
                <xsd:element name="set" type="Templates:SetTemplate" minOccurs="0"
                    maxOccurs="unbounded"/>
                <xsd:element name="set_of" type="Templates:SetOfTemplate"
                    minOccurs="0" maxOccurs="unbounded"/>
                <xsd:element name="enumerated" type="Templates:EnumeratedTemplate"
                    minOccurs="0" maxOccurs="unbounded"/>
                <xsd:element name="union" type="Templates:UnionTemplate" minOccurs="0"
                    maxOccurs="unbounded"/>
                <xsd:element name="anytype" type="Templates:AnytypeTemplate" minOccurs="0"
                    maxOccurs="unbounded"/>
                <xsd:element name="address" type="Templates:AddressTemplate" minOccurs="0"
                    maxOccurs="unbounded"/>
                <xsd:element name="omit" type="Templates:omit"/>
                <xsd:element name="any" type="Templates:any"/>
                <xsd:element name="anyoromit" type="Templates:anyoromit"/>
                <xsd:element name="templateDef" type="SimpleTypes:TString"/>
                <xsd:element name="null" type="Templates:null"/>
            </xsd:choice>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="EnumeratedTemplate">
    <xsd:complexContent>
        <xsd:extension base="Values:EnumeratedValue">
            <xsd:choice minOccurs="0">
                <xsd:element name="value" type="SimpleTypes:TString"/>
                <xsd:element name="templateDef" type="SimpleTypes:TString"/>
                <xsd:element name="omit" type="Templates:omit"/>
                <xsd:element name="any" type="Templates:any"/>
                <xsd:element name="anyoromit" type="Templates:anyoromit"/>
                <xsd:element name="null" type="Templates:null"/>
            </xsd:choice>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="UnionTemplate">
    <xsd:complexContent>
        <xsd:extension base="Values:UnionValue">
            <xsd:choice minOccurs="0">
                <xsd:element name="integer" type="Templates:IntegerTemplate"/>
                <xsd:element name="float" type="Templates:FloatTemplate"/>
                <xsd:element name="boolean" type="Templates:BooleanTemplate"/>
                <xsd:element name="verdicttype" type="Templates:VerdictTemplate"/>
                <xsd:element name="bitstring" type="Templates:BitstringTemplate"/>
            </xsd:choice>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

```

```

<xsd:element name="hexstring" type="Templates:HexstringTemplate"/>
<xsd:element name="octetstring" type="Templates:OctetstringTemplate"/>
<xsd:element name="charstring" type="Templates:CharstringTemplate"/>
<xsd:element name="universal_charstring"
  type="Templates:UniversalCharstringTemplate"/>
<xsd:element name="record" type="Templates:RecordTemplate"/>
<xsd:element name="record_of" type="Templates:RecordOfTemplate"/>
<xsd:element name="array" type="Templates:ArrayTemplate"/>
<xsd:element name="set" type="Templates:SetTemplate"/>
<xsd:element name="set_of" type="Templates:SetOfTemplate"/>
<xsd:element name="enumerated" type="Templates:EnumeratedTemplate"/>
<xsd:element name="union" type="Templates:UnionTemplate"/>
<xsd:element name="anytype" type="Templates:AnytypeTemplate"/>
<xsd:element name="address" type="Templates:AddressTemplate"/>
<xsd:element name="omit" type="Templates:omit"/>
<xsd:element name="any" type="Templates:any"/>
<xsd:element name="anyoromit" type="Templates:anyoromit"/>
<xsd:element name="templateDef" type="SimpleTypes:TString"/>
<xsd:element name="null" type="Templates:null"/>
</xsd:choice>
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="AnytypeTemplate">
  <xsd:complexContent>
    <xsd:extension base="Values:AnytypeValue">
      <xsd:choice minOccurs="0">
        <xsd:element name="integer" type="Templates:IntegerTemplate"/>
        <xsd:element name="float" type="Templates:FloatTemplate"/>
        <xsd:element name="boolean" type="Templates:BooleanTemplate"/>
        <xsd:element name="verdicttype" type="Templates:VerdictTemplate"/>
        <xsd:element name="bitstring" type="Templates:BitstringTemplate"/>
        <xsd:element name="hexstring" type="Templates:HexstringTemplate"/>
        <xsd:element name="octetstring" type="Templates:OctetstringTemplate"/>
        <xsd:element name="charstring" type="Templates:CharstringTemplate"/>
        <xsd:element name="universal_charstring"
          type="Templates:UniversalCharstringTemplate"/>
        <xsd:element name="record" type="Templates:RecordTemplate"/>
        <xsd:element name="record_of" type="Templates:RecordOfTemplate"/>
        <xsd:element name="array" type="Templates:ArrayTemplate"/>
        <xsd:element name="set" type="Templates:SetTemplate"/>
        <xsd:element name="set_of" type="Templates:SetOfTemplate"/>
        <xsd:element name="enumerated" type="Templates:EnumeratedTemplate"/>
        <xsd:element name="union" type="Templates:UnionTemplate"/>
        <xsd:element name="address" type="Templates:AddressTemplate"/>
        <xsd:element name="omit" type="Templates:omit"/>
        <xsd:element name="any" type="Templates:any"/>
        <xsd:element name="anyoromit" type="Templates:anyoromit"/>
        <xsd:element name="templateDef" type="SimpleTypes:TString"/>
        <xsd:element name="null" type="Templates:null"/>
      </xsd:choice>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="AddressTemplate">
  <xsd:complexContent>
    <xsd:extension base="Values:AnytypeValue">
      <xsd:choice minOccurs="0">
        <xsd:element name="integer" type="Templates:IntegerTemplate"/>
        <xsd:element name="float" type="Templates:FloatTemplate"/>
        <xsd:element name="boolean" type="Templates:BooleanTemplate"/>
        <xsd:element name="bitstring" type="Templates:BitstringTemplate"/>
        <xsd:element name="hexstring" type="Templates:HexstringTemplate"/>
        <xsd:element name="octetstring" type="Templates:OctetstringTemplate"/>
        <xsd:element name="charstring" type="Templates:CharstringTemplate"/>
        <xsd:element name="universal_charstring"
          type="Templates:UniversalCharstringTemplate"/>
        <xsd:element name="record" type="Templates:RecordTemplate"/>
        <xsd:element name="record_of" type="Templates:RecordOfTemplate"/>
        <xsd:element name="array" type="Templates:ArrayTemplate"/>
        <xsd:element name="set" type="Templates:SetTemplate"/>
        <xsd:element name="set_of" type="Templates:SetOfTemplate"/>
        <xsd:element name="enumerated" type="Templates:EnumeratedTemplate"/>
        <xsd:element name="union" type="Templates:UnionTemplate"/>
        <xsd:element name="anytype" type="Templates:AnytypeTemplate"/>
        <xsd:element name="omit" type="Templates:omit"/>
        <xsd:element name="any" type="Templates:any"/>
        <xsd:element name="anyoromit" type="Templates:anyoromit"/>
      </xsd:choice>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

```

```

        <xsd:element name="templateDef" type="SimpleTypes:TString"/>
        <xsd:element name="null" type="Templates:null"/>
    </xsd:choice>
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>
</xsd:schema>

```

## B.5 TCI-TL XML schema for Events

```

<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://uri.etsi.org/ttcn-3/tci/Events_v4_4_1.xsd"
  xmlns:Events="http://uri.etsi.org/ttcn-3/tci/Events_v4_4_1.xsd"
  xmlns:Types="http://uri.etsi.org/ttcn-3/tci/Types_v4_4_1.xsd"
  xmlns:Templates="http://uri.etsi.org/ttcn-3/tci/Templates_v4_4_1.xsd"
  xmlns:SimpleTypes="http://uri.etsi.org/ttcn-3/tci/SimpleTypes_v4_4_1.xsd"
  xmlns:Values="http://uri.etsi.org/ttcn-3/tci/Values_v4_4_1.xsd"
  elementFormDefault="qualified">

  <xsd:import namespace="http://uri.etsi.org/ttcn-3/tci/SimpleTypes_v4_4_1.xsd"
    schemaLocation="SimpleTypes_v4_4_1.xsd"/>
  <xsd:import namespace="http://uri.etsi.org/ttcn-3/tci/Types_v4_4_1.xsd"
    schemaLocation="Types_v4_4_1.xsd"/>
  <xsd:import namespace="http://uri.etsi.org/ttcn-3/tci/Values_v4_4_1.xsd"
    schemaLocation="Values_v4_4_1.xsd"/>
  <xsd:import namespace="http://uri.etsi.org/ttcn-3/tci/Templates_v4_4_1.xsd"
    schemaLocation="Templates_v4_4_1.xsd"/>

  <!-- common definition for all events -->
  <xsd:complexType name="Event" mixed="true">
    <xsd:sequence>
      <xsd:element name="am" type="SimpleTypes:TString"/>
    </xsd:sequence>
    <xsd:attribute name="ts" type="xsd:long" use="required"/>
    <xsd:attribute name="src" type="SimpleTypes:TString" use="optional"/>
    <xsd:attribute name="line" type="SimpleTypes:TInteger" use="optional"/>

    <!-- general identifier structure for test components, ports and timer -->
    <xsd:attribute name="name" type="SimpleTypes:TString" use="required"/>
    <xsd:attribute name="id" type="SimpleTypes:TString" use="required"/>
    <xsd:attribute name="type" type="SimpleTypes:TString" use="required"/>
  </xsd:complexType>

  <!-- this event is extended by all port configuration events -->
  <xsd:complexType name="PortConfiguration">
    <xsd:complexContent mixed="true">
      <xsd:extension base="Events:Event">
        <xsd:sequence>
          <xsd:element name="port1" type="Types:TriPortIdType" />
          <xsd:element name="port2" type="Types:TriPortIdType" />
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <!-- this event is extended by all port status events -->
  <xsd:complexType name="PortStatus">
    <xsd:complexContent mixed="true">
      <xsd:extension base="Events:Event">
        <xsd:sequence>
          <xsd:element name="port" type="Types:TriPortIdType"/>
          <xsd:element name="stat" type="SimpleTypes:PortStatusType" minOccurs="0"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <!-- testcases -->
  <xsd:complexType name="tliTcExecute">
    <xsd:complexContent mixed="true">
      <xsd:extension base="Events:Event">
        <xsd:sequence>
          <xsd:element name="tcId" type="Types:TciTestCaseIdType"/>
          <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>
          <xsd:element name="dur" type="SimpleTypes:TriTimerDurationType" minOccurs="0"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

```

```

    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliTcStart">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:Event">
      <xsd:sequence>
        <xsd:element name="tcId" type="Types:TciTestCaseIdType"/>
        <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>
        <xsd:element name="dur" type="SimpleTypes:TriTimerDurationType" minOccurs="0"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliTcStop">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:Event">
      <xsd:sequence>
        <xsd:element name="reason" type="SimpleTypes:TString" minOccurs="0"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliTcStarted">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:Event">
      <xsd:sequence>
        <xsd:element name="tcId" type="Types:TciTestCaseIdType"/>
        <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>
        <xsd:element name="dur" type="SimpleTypes:TriTimerDurationType" minOccurs="0"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliTcTerminated">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:Event">
      <xsd:sequence>
        <xsd:element name="tcId" type="Types:TciTestCaseIdType"/>
        <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>
        <xsd:element name="verdict" type="Values:VerdictValue"/>
        <xsd:element name="reason" type="SimpleTypes:TString" minOccurs="0"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<!-- control -->
<xsd:complexType name="tliCtrlStart">
  <xsd:complexContent>
    <xsd:extension base="Events:Event"/>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliCtrlStop">
  <xsd:complexContent>
    <xsd:extension base="Events:Event"/>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliCtrlTerminated">
  <xsd:complexContent>
    <xsd:extension base="Events:Event"/>
  </xsd:complexContent>
</xsd:complexType>

<!-- asynchronous communication -->
<xsd:complexType name="tliMSend_m">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:Event">
      <xsd:sequence>
        <xsd:element name="at" type="Types:TriPortIdType"/>
        <xsd:element name="to" type="Types:TriPortIdType" minOccurs="0"/>
        <xsd:element name="msgValue" type="Values:Value"/>
        <xsd:element name="addrValue" type="Values:Value" minOccurs="0"/>
        <xsd:choice>

```

```

        <xsd:element name="encoder-failure" type="SimpleTypes:TciStatusType"
minOccurs="0"/>
        <xsd:sequence>
            <xsd:element name="msg" type="Types:TriMessageType" minOccurs="0"/>
            <xsd:element name="address" type="Types:TriAddressType" minOccurs="0"/>
            <xsd:element name="transmission-failure"
type="SimpleTypes:TriStatusType" minOccurs="0"/>
        </xsd:sequence>
    </xsd:choice>
</xsd:sequence>
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliMSend_m_BC">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="at" type="Types:TriPortIdType"/>
                <xsd:element name="to" type="Types:TriPortIdType" minOccurs="0"/>
                <xsd:element name="msgValue" type="Values:Value"/>
            <xsd:choice>
                <xsd:element name="encoder-failure" type="SimpleTypes:TciStatusType"
minOccurs="0"/>
            </xsd:choice>
            <xsd:sequence>
                <xsd:element name="msg" type="Types:TriMessageType" minOccurs="0"/>
                <xsd:element name="transmission-failure"
type="SimpleTypes:TriStatusType" minOccurs="0"/>
            </xsd:sequence>
        </xsd:choice>
    </xsd:sequence>
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliMSend_m_MC">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="at" type="Types:TriPortIdType"/>
                <xsd:element name="to" type="Types:TriPortIdType" minOccurs="0"/>
                <xsd:element name="msgValue" type="Values:Value"/>
                <xsd:element name="addrValues" type="Types:TciValueListType" minOccurs="0"/>
            <xsd:choice>
                <xsd:element name="encoder-failure" type="SimpleTypes:TciStatusType"
minOccurs="0"/>
            </xsd:choice>
            <xsd:sequence>
                <xsd:element name="msg" type="Types:TriMessageType" minOccurs="0"/>
                <xsd:element name="addresses" type="Types:TriAddressListType"
minOccurs="0"/>
            </xsd:sequence>
            <xsd:element name="transmission-failure"
type="SimpleTypes:TriStatusType" minOccurs="0"/>
        </xsd:choice>
    </xsd:sequence>
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliMSend_c">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="at" type="Types:TriPortIdType"/>
                <xsd:element name="to" type="Types:TriPortIdType" minOccurs="0"/>
                <xsd:element name="msgValue" type="Values:Value"/>
                <xsd:element name="transmission-failure" type="SimpleTypes:TriStatusType"
minOccurs="0"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliMSend_c_BC">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="at" type="Types:TriPortIdType"/>
                <xsd:element name="to" type="Types:TriPortIdListType" minOccurs="0"/>
                <xsd:element name="msgValue" type="Values:Value"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

```

```

        <xsd:element name="transmission-failure" type="SimpleTypes:TriStatusType"
minOccurs="0"/>
    </xsd:sequence>
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliMSend_c_MC">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="at" type="Types:TriPortIdType"/>
                <xsd:element name="to" type="Types:TriPortIdListType" minOccurs="0"/>
                <xsd:element name="msgValue" type="Values:Value"/>
                <xsd:element name="transmission-failure" type="SimpleTypes:TriStatusType"
minOccurs="0"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliMDetected_m">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="at" type="Types:TriPortIdType"/>
                <xsd:element name="from" type="Types:TriPortIdType" minOccurs="0"/>
                <xsd:element name="msgValue" type="Types:TriMessageType"/>
                <xsd:element name="address" type="Types:TriAddressType" minOccurs="0"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliMDetected_c">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="at" type="Types:TriPortIdType"/>
                <xsd:element name="from" type="Types:TriPortIdType" minOccurs="0"/>
                <xsd:element name="msgValue" type="Values:Value"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliMMismatch_m">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="at" type="Types:TriPortIdType"/>
                <xsd:element name="msgValue" type="Values:Value"/>
                <xsd:element name="msgTmpl" type="Templates:TciValueTemplate"/>
                <xsd:element name="diffs" type="Templates:TciValueDifferenceList"/>
                <xsd:element name="addrValue" type="Values:Value" minOccurs="0"/>
                <xsd:element name="addressTmpl" type="Templates:TciValueTemplate"
minOccurs="0"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliMMismatch_c">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="at" type="Types:TriPortIdType"/>
                <xsd:element name="msgValue" type="Values:Value"/>
                <xsd:element name="msgTmpl" type="Templates:TciValueTemplate"/>
                <xsd:element name="diffs" type="Templates:TciValueDifferenceList"/>
                <xsd:element name="from" type="Types:TriComponentIdType" minOccurs="0"/>
                <xsd:element name="fromTmpl" type="Templates:TciNonValueTemplate"
minOccurs="0"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliMReceive_m">
    <xsd:complexContent mixed="true">

```

```

        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="at" type="Types:TriPortIdType"/>
                <xsd:element name="msgValue" type="Values:Value" minOccurs="0"/>
                <xsd:element name="msgTpl" type="Templates:TciValueTemplate"
minOccurs="0"/>
                <xsd:element name="addrValue" type="Values:Value" minOccurs="0"/>
                <xsd:element name="addressTpl" type="Templates:TciValueTemplate"
minOccurs="0"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliMReceive_c">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="at" type="Types:TriPortIdType"/>
                <xsd:element name="msgValue" type="Values:Value" minOccurs="0"/>
                <xsd:element name="msgTpl" type="Templates:TciValueTemplate"
minOccurs="0"/>
                <xsd:element name="from" type="Types:TriComponentIdType" minOccurs="0"/>
                <xsd:element name="fromTpl" type="Templates:TciNonValueTemplate"
minOccurs="0"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<!-- synchronous communication -->
<xsd:complexType name="tliPrCall_m">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="at" type="Types:TriPortIdType"/>
                <xsd:element name="to" type="Types:TriPortIdType" minOccurs="0"/>
                <xsd:element name="signature" type="Types:TriSignatureIdType"/>
                <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>
                <xsd:element name="addrValue" type="Values:Value" minOccurs="0"/>
                <xsd:choice>
                    <xsd:element name="encoder-failure" type="SimpleTypes:TciStatusType"
minOccurs="0"/>
                    <xsd:sequence>
                        <xsd:element name="triPars" type="Types:TriParameterListType"
minOccurs="0"/>
                        <xsd:element name="address" type="Types:TriAddressType" minOccurs="0"/>
                        <xsd:element name="transmission-failure"
type="SimpleTypes:TriStatusType" minOccurs="0"/>
                    </xsd:sequence>
                </xsd:choice>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliPrCall_m_BC">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="at" type="Types:TriPortIdType"/>
                <xsd:element name="to" type="Types:TriPortIdType" minOccurs="0"/>
                <xsd:element name="signature" type="Types:TriSignatureIdType"/>
                <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>
                <xsd:choice>
                    <xsd:element name="encoder-failure" type="SimpleTypes:TciStatusType"
minOccurs="0"/>
                    <xsd:sequence>
                        <xsd:element name="triPars" type="Types:TriParameterListType"
minOccurs="0"/>
                        <xsd:element name="transmission-failure"
type="SimpleTypes:TriStatusType" minOccurs="0"/>
                    </xsd:sequence>
                </xsd:choice>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliPrCall_m_MC">

```



```

<xsd:complexContent mixed="true">
  <xsd:extension base="Events:Event">
    <xsd:sequence>
      <xsd:element name="at" type="Types:TriPortIdType"/>
      <xsd:element name="to" type="Types:TriPortIdType" minOccurs="0"/>
      <xsd:element name="signature" type="Types:TriSignatureIdType"/>
      <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>
      <xsd:element name="addrValues" type="Types:TciValueListType" minOccurs="0"/>
      <xsd:choice>
        <xsd:element name="encoder-failure" type="SimpleTypes:TciStatusType"
minOccurs="0"/>
        <xsd:sequence>
          <xsd:element name="triPars" type="Types:TriParameterListType"
minOccurs="0"/>
          <xsd:element name="addresses" type="Types:TriAddressListType"
minOccurs="0"/>
          <xsd:element name="transmission-failure"
type="SimpleTypes:TriStatusType" minOccurs="0"/>
        </xsd:sequence>
      </xsd:choice>
    </xsd:sequence>
  </xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliPrCall_c">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:Event">
      <xsd:sequence>
        <xsd:element name="at" type="Types:TriPortIdType"/>
        <xsd:element name="to" type="Types:TriPortIdType" minOccurs="0"/>
        <xsd:element name="signature" type="Types:TriSignatureIdType"/>
        <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>
        <xsd:element name="transmission-failure" type="SimpleTypes:TriStatusType"
minOccurs="0"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliPrCall_c_BC">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:Event">
      <xsd:sequence>
        <xsd:element name="at" type="Types:TriPortIdType"/>
        <xsd:element name="to" type="Types:TriPortIdListType" minOccurs="0"/>
        <xsd:element name="signature" type="Types:TriSignatureIdType"/>
        <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>
        <xsd:element name="transmission-failure" type="SimpleTypes:TriStatusType"
minOccurs="0"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliPrCall_c_MC">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:Event">
      <xsd:sequence>
        <xsd:element name="at" type="Types:TriPortIdType"/>
        <xsd:element name="to" type="Types:TriPortIdListType" minOccurs="0"/>
        <xsd:element name="signature" type="Types:TriSignatureIdType"/>
        <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>
        <xsd:element name="transmission-failure" type="SimpleTypes:TriStatusType"
minOccurs="0"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliPrGetCallDetected_m">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:Event">
      <xsd:sequence>
        <xsd:element name="at" type="Types:TriPortIdType"/>
        <xsd:element name="from" type="Types:TriPortIdType" minOccurs="0"/>
        <xsd:element name="signature" type="Types:TriSignatureIdType"/>
        <xsd:element name="triPars" type="Types:TriParameterListType" minOccurs="0"/>
        <xsd:element name="address" type="Types:TriAddressType" minOccurs="0"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

```

```

        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliPrGetCallDetected_c">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="at" type="Types:TriPortIdType"/>
                <xsd:element name="from" type="Types:TriPortIdType" minOccurs="0"/>
                <xsd:element name="signature" type="Types:TriSignatureIdType"/>
                <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliPrGetCallMismatch_m">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="at" type="Types:TriPortIdType"/>
                <xsd:element name="signature" type="Types:TriSignatureIdType"/>
                <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>
                <xsd:element name="parsTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>
                <xsd:element name="diffs" type="Templates:TciValueDifferenceList"/>
                <xsd:element name="addrValue" type="Values:Value" minOccurs="0"/>
                <xsd:element name="addressTmpl" type="Templates:TciValueTemplate"
minOccurs="0"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliPrGetCallMismatch_c">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="at" type="Types:TriPortIdType"/>
                <xsd:element name="signature" type="Types:TriSignatureIdType"/>
                <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>
                <xsd:element name="parsTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>
                <xsd:element name="diffs" type="Templates:TciValueDifferenceList"/>
                <xsd:element name="from" type="Types:TriComponentIdType" minOccurs="0"/>
                <xsd:element name="fromTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliPrGetCall_m">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="at" type="Types:TriPortIdType"/>
                <xsd:element name="signature" type="Types:TriSignatureIdType"/>
                <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>
                <xsd:element name="parsTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>
                <xsd:element name="addrValue" type="Values:Value" minOccurs="0"/>
                <xsd:element name="addressTmpl" type="Templates:TciValueTemplate"
minOccurs="0"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliPrGetCall_c">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="at" type="Types:TriPortIdType"/>
                <xsd:element name="signature" type="Types:TriSignatureIdType"/>
                <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>
                <xsd:element name="parsTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>
                <xsd:element name="from" type="Types:TriComponentIdType" minOccurs="0"/>
                <xsd:element name="fromTmpl" type="Templates:TciNonValueTemplate"
minOccurs="0"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>

```

```

</xsd:complexType>

<xsd:complexType name="tliPrReply_m">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:Event">
      <xsd:sequence>
        <xsd:element name="at" type="Types:TriPortIdType"/>
        <xsd:element name="to" type="Types:TriPortIdType" minOccurs="0"/>
        <xsd:element name="signature" type="Types:TriSignatureIdType"/>
        <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>
        <xsd:element name="replValue" type="Values:Value" minOccurs="0"/>
        <xsd:element name="addrValue" type="Values:Value" minOccurs="0"/>
        <xsd:choice>
          <xsd:element name="encoder-failure" type="SimpleTypes:TciStatusType"
            minOccurs="0"/>
          <xsd:sequence>
            <xsd:element name="triPars" type="Types:TriParameterListType"
              minOccurs="0"/>
            <xsd:element name="repl" type="Types:TriParameterType" minOccurs="0"/>
            <xsd:element name="address" type="Types:TriAddressType" minOccurs="0"/>
            <xsd:element name="transmission-failure"
              type="SimpleTypes:TciStatusType" minOccurs="0"/>
          </xsd:sequence>
        </xsd:choice>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliPrReply_m_BC">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:Event">
      <xsd:sequence>
        <xsd:element name="at" type="Types:TriPortIdType"/>
        <xsd:element name="to" type="Types:TriPortIdType" minOccurs="0"/>
        <xsd:element name="signature" type="Types:TriSignatureIdType"/>
        <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>
        <xsd:element name="replValue" type="Values:Value" minOccurs="0"/>
        <xsd:choice>
          <xsd:element name="encoder-failure" type="SimpleTypes:TciStatusType"
            minOccurs="0"/>
          <xsd:sequence>
            <xsd:element name="triPars" type="Types:TriParameterListType"
              minOccurs="0"/>
            <xsd:element name="repl" type="Types:TriParameterType" minOccurs="0"/>
            <xsd:element name="transmission-failure"
              type="SimpleTypes:TciStatusType" minOccurs="0"/>
          </xsd:sequence>
        </xsd:choice>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliPrReply_m_MC">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:Event">
      <xsd:sequence>
        <xsd:element name="at" type="Types:TriPortIdType"/>
        <xsd:element name="to" type="Types:TriPortIdType" minOccurs="0"/>
        <xsd:element name="signature" type="Types:TriSignatureIdType"/>
        <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>
        <xsd:element name="replValue" type="Values:Value" minOccurs="0"/>
        <xsd:element name="addrValues" type="Types:TciValueListType" minOccurs="0"/>
        <xsd:choice>
          <xsd:element name="encoder-failure" type="SimpleTypes:TciStatusType"
            minOccurs="0"/>
          <xsd:sequence>
            <xsd:element name="triPars" type="Types:TriParameterListType"
              minOccurs="0"/>
            <xsd:element name="repl" type="Types:TriParameterType" minOccurs="0"/>
            <xsd:element name="addresses" type="Types:TriAddressListType"
              minOccurs="0"/>
            <xsd:element name="transmission-failure"
              type="SimpleTypes:TciStatusType" minOccurs="0"/>
          </xsd:sequence>
        </xsd:choice>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>

```

```

</xsd:complexType>

<xsd:complexType name="tliPrReply_c">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:Event">
      <xsd:sequence>
        <xsd:element name="at" type="Types:TriPortIdType"/>
        <xsd:element name="to" type="Types:TriPortIdType" minOccurs="0"/>
        <xsd:element name="signature" type="Types:TriSignatureIdType"/>
        <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>
        <xsd:element name="replValue" type="Values:Value" minOccurs="0"/>
        <xsd:element name="transmission-failure" type="SimpleTypes:TriStatusType"
minOccurs="0"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliPrReply_c_BC">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:Event">
      <xsd:sequence>
        <xsd:element name="at" type="Types:TriPortIdType"/>
        <xsd:element name="to" type="Types:TriPortIdListType" minOccurs="0"/>
        <xsd:element name="signature" type="Types:TriSignatureIdType"/>
        <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>
        <xsd:element name="replValue" type="Values:Value" minOccurs="0"/>
        <xsd:element name="transmission-failure" type="SimpleTypes:TriStatusType"
minOccurs="0"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliPrReply_c_MC">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:Event">
      <xsd:sequence>
        <xsd:element name="at" type="Types:TriPortIdType"/>
        <xsd:element name="to" type="Types:TriPortIdListType" minOccurs="0"/>
        <xsd:element name="signature" type="Types:TriSignatureIdType"/>
        <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>
        <xsd:element name="replValue" type="Values:Value" minOccurs="0"/>
        <xsd:element name="transmission-failure" type="SimpleTypes:TriStatusType"
minOccurs="0"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliPrGetReplyDetected_m">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:Event">
      <xsd:sequence>
        <xsd:element name="at" type="Types:TriPortIdType"/>
        <xsd:element name="from" type="Types:TriPortIdType" minOccurs="0"/>
        <xsd:element name="signature" type="Types:TriSignatureIdType"/>
        <xsd:element name="triPars" type="Types:TriParameterListType" minOccurs="0"/>
        <xsd:element name="repl" type="Types:TriParameterType" minOccurs="0"/>
        <xsd:element name="address" type="Types:TriAddressType" minOccurs="0"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliPrGetReplyDetected_c">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:Event">
      <xsd:sequence>
        <xsd:element name="at" type="Types:TriPortIdType"/>
        <xsd:element name="from" type="Types:TriPortIdType" minOccurs="0"/>
        <xsd:element name="signature" type="Types:TriSignatureIdType"/>
        <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>
        <xsd:element name="replValue" type="Values:Value" minOccurs="0"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliPrGetReplyMismatch_m">

```

```

<xsd:complexContent mixed="true">
  <xsd:extension base="Events:Event">
    <xsd:sequence>
      <xsd:element name="at" type="Types:TriPortIdType"/>
      <xsd:element name="signature" type="Types:TriSignatureIdType"/>
      <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>
      <xsd:element name="parsTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>
      <xsd:element name="replValue" type="Values:Value" minOccurs="0"/>
      <xsd:element name="replTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>
      <xsd:element name="diffs" type="Templates:TciValueDifferenceList"/>
      <xsd:element name="addrValue" type="Values:Value" minOccurs="0"/>
      <xsd:element name="addressTmpl" type="Templates:TciValueTemplate"
minOccurs="0"/>
    </xsd:sequence>
  </xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliPrGetReplyMismatch_c">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:Event">
      <xsd:sequence>
        <xsd:element name="at" type="Types:TriPortIdType"/>
        <xsd:element name="signature" type="Types:TriSignatureIdType"/>
        <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>
        <xsd:element name="parsTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>
        <xsd:element name="replValue" type="Values:Value" minOccurs="0"/>
        <xsd:element name="replTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>
        <xsd:element name="diffs" type="Templates:TciValueDifferenceList"/>
        <xsd:element name="from" type="Types:TriComponentIdType" minOccurs="0"/>
        <xsd:element name="fromTmpl" type="Templates:TciNonValueTemplate"
minOccurs="0"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliPrGetReply_m">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:Event">
      <xsd:sequence>
        <xsd:element name="at" type="Types:TriPortIdType"/>
        <xsd:element name="signature" type="Types:TriSignatureIdType"/>
        <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>
        <xsd:element name="parsTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>
        <xsd:element name="replValue" type="Values:Value" minOccurs="0"/>
        <xsd:element name="replTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>
        <xsd:element name="addrValue" type="Values:Value" minOccurs="0"/>
        <xsd:element name="addressTmpl" type="Templates:TciValueTemplate"
minOccurs="0"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliPrGetReply_c">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:Event">
      <xsd:sequence>
        <xsd:element name="at" type="Types:TriPortIdType"/>
        <xsd:element name="signature" type="Types:TriSignatureIdType"/>
        <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>
        <xsd:element name="parsTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>
        <xsd:element name="replValue" type="Values:Value" minOccurs="0"/>
        <xsd:element name="replTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>
        <xsd:element name="from" type="Types:TriComponentIdType" minOccurs="0"/>
        <xsd:element name="fromTmpl" type="Templates:TciNonValueTemplate"
minOccurs="0"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliPrRaise_m">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:Event">
      <xsd:sequence>
        <xsd:element name="at" type="Types:TriPortIdType"/>
        <xsd:element name="to" type="Types:TriPortIdType" minOccurs="0"/>
        <xsd:element name="signature" type="Types:TriSignatureIdType"/>

```

```

        <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>
        <xsd:element name="excValue" type="Values:Value" minOccurs="0"/>
        <xsd:element name="addrValue" type="Values:Value" minOccurs="0"/>
        <xsd:choice>
            <xsd:element name="encoder-failure" type="SimpleTypes:TciStatusType"
minOccurs="0"/>
            <xsd:sequence>
                <xsd:element name="exc" type="Types:TriExceptionType" minOccurs="0"/>
                <xsd:element name="address" type="Types:TriAddressType" minOccurs="0"/>
                <xsd:element name="transmission-failure"
type="SimpleTypes:TriStatusType" minOccurs="0"/>
            </xsd:sequence>
        </xsd:choice>
    </xsd:sequence>
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliPrRaise_m_BC">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="at" type="Types:TriPortIdType"/>
                <xsd:element name="to" type="Types:TriPortIdType" minOccurs="0"/>
                <xsd:element name="signature" type="Types:TriSignatureIdType"/>
                <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>
                <xsd:element name="excValue" type="Values:Value" minOccurs="0"/>
                <xsd:choice>
                    <xsd:element name="encoder-failure" type="SimpleTypes:TciStatusType"
minOccurs="0"/>
                    <xsd:sequence>
                        <xsd:element name="exc" type="Types:TriExceptionType" minOccurs="0"/>
                        <xsd:element name="transmission-failure"
type="SimpleTypes:TriStatusType" minOccurs="0"/>
                    </xsd:sequence>
                </xsd:choice>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliPrRaise_m_MC">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="at" type="Types:TriPortIdType"/>
                <xsd:element name="to" type="Types:TriPortIdType" minOccurs="0"/>
                <xsd:element name="signature" type="Types:TriSignatureIdType"/>
                <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>
                <xsd:element name="excValue" type="Values:Value" minOccurs="0"/>
                <xsd:element name="addrValues" type="Types:TciValueListType" minOccurs="0"/>
                <xsd:choice>
                    <xsd:element name="encoder-failure" type="SimpleTypes:TciStatusType"
minOccurs="0"/>
                    <xsd:sequence>
                        <xsd:element name="exc" type="Types:TriExceptionType" minOccurs="0"/>
                        <xsd:element name="addresses" type="Types:TriAddressListType"
minOccurs="0"/>
                        <xsd:element name="transmission-failure"
type="SimpleTypes:TriStatusType" minOccurs="0"/>
                    </xsd:sequence>
                </xsd:choice>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliPrRaise_c">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="at" type="Types:TriPortIdType"/>
                <xsd:element name="to" type="Types:TriPortIdType" minOccurs="0"/>
                <xsd:element name="signature" type="Types:TriSignatureIdType"/>
                <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>
                <xsd:element name="excValue" type="Values:Value" minOccurs="0"/>
                <xsd:element name="transmission-failure" type="SimpleTypes:TriStatusType"
minOccurs="0"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

```

```

    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliPrRaise_c_BC">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:Event">
      <xsd:sequence>
        <xsd:element name="at" type="Types:TriPortIdType"/>
        <xsd:element name="to" type="Types:TriPortIdListType" minOccurs="0"/>
        <xsd:element name="signature" type="Types:TriSignatureIdType"/>
        <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>
        <xsd:element name="excValue" type="Values:Value" minOccurs="0"/>
        <xsd:element name="transmission-failure" type="SimpleTypes:TriStatusType"
minOccurs="0"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliPrRaise_c_MC">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:Event">
      <xsd:sequence>
        <xsd:element name="at" type="Types:TriPortIdType"/>
        <xsd:element name="to" type="Types:TriPortIdListType" minOccurs="0"/>
        <xsd:element name="signature" type="Types:TriSignatureIdType"/>
        <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>
        <xsd:element name="excValue" type="Values:Value" minOccurs="0"/>
        <xsd:element name="transmission-failure" type="SimpleTypes:TriStatusType"
minOccurs="0"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliPrCatchDetected_m">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:Event">
      <xsd:sequence>
        <xsd:element name="at" type="Types:TriPortIdType"/>
        <xsd:element name="from" type="Types:TriPortIdType" minOccurs="0"/>
        <xsd:element name="signature" type="Types:TriSignatureIdType"/>
        <xsd:element name="exc" type="Types:TriExceptionType" minOccurs="0"/>
        <xsd:element name="address" type="Types:TriAddressType" minOccurs="0"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliPrCatchDetected_c">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:Event">
      <xsd:sequence>
        <xsd:element name="at" type="Types:TriPortIdType"/>
        <xsd:element name="from" type="Types:TriPortIdType" minOccurs="0"/>
        <xsd:element name="signature" type="Types:TriSignatureIdType"/>
        <xsd:element name="excValue" type="Values:Value" minOccurs="0"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliPrCatchMismatch_m">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:Event">
      <xsd:sequence>
        <xsd:element name="at" type="Types:TriPortIdType"/>
        <xsd:element name="signature" type="Types:TriSignatureIdType"/>
        <xsd:element name="excValue" type="Values:Value" minOccurs="0"/>
        <xsd:element name="excTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>
        <xsd:element name="diffs" type="Templates:TciValueDifferenceList"/>
        <xsd:element name="addrValue" type="Values:Value" minOccurs="0"/>
        <xsd:element name="addressTmpl" type="Templates:TciValueTemplate"
minOccurs="0"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliPrCatchMismatch_c">

```

```

    <xsd:complexContent mixed="true">
      <xsd:extension base="Events:Event">
        <xsd:sequence>
          <xsd:element name="at" type="Types:TriPortIdType"/>
          <xsd:element name="signature" type="Types:TriSignatureIdType"/>
          <xsd:element name="excValue" type="Values:Value" minOccurs="0"/>
          <xsd:element name="excTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>
          <xsd:element name="diffs" type="Templates:TciValueDifferenceList"/>
          <xsd:element name="from" type="Types:TriComponentIdType" minOccurs="0"/>
          <xsd:element name="fromTmpl" type="Templates:TciNonValueTemplate"
minOccurs="0"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="tliPrCatch_m">
    <xsd:complexContent mixed="true">
      <xsd:extension base="Events:Event">
        <xsd:sequence>
          <xsd:element name="at" type="Types:TriPortIdType"/>
          <xsd:element name="signature" type="Types:TriSignatureIdType"/>
          <xsd:element name="excValue" type="Values:Value"/>
          <xsd:element name="excTmpl" type="Templates:TciValueTemplate"/>
          <xsd:element name="addrValue" type="Values:Value" minOccurs="0"/>
          <xsd:element name="addressTmpl" type="Templates:TciValueTemplate"
minOccurs="0"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="tliPrCatch_c">
    <xsd:complexContent mixed="true">
      <xsd:extension base="Events:Event">
        <xsd:sequence>
          <xsd:element name="at" type="Types:TriPortIdType"/>
          <xsd:element name="signature" type="Types:TriSignatureIdType"/>
          <xsd:element name="excValue" type="Values:Value" minOccurs="0"/>
          <xsd:element name="excTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>
          <xsd:element name="from" type="Types:TriComponentIdType" minOccurs="0"/>
          <xsd:element name="fromTmpl" type="Templates:TciNonValueTemplate"
minOccurs="0"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="tliPrCatchTimeoutDetected">
    <xsd:complexContent mixed="true">
      <xsd:extension base="Events:Event">
        <xsd:sequence>
          <xsd:element name="at" type="Types:TriPortIdType"/>
          <xsd:element name="signature" type="Types:TriSignatureIdType"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="tliPrCatchTimeout">
    <xsd:complexContent mixed="true">
      <xsd:extension base="Events:Event">
        <xsd:sequence>
          <xsd:element name="at" type="Types:TriPortIdType"/>
          <xsd:element name="signature" type="Types:TriSignatureIdType"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <!-- components -->
  <xsd:complexType name="tliCCreate">
    <xsd:complexContent mixed="true">
      <xsd:extension base="Events:Event">
        <xsd:sequence>
          <xsd:element name="comp" type="Types:TriComponentIdType"/>
          <xsd:element name="name" type="SimpleTypes:TString"/>
          <xsd:element name="hostId" type="Values:Value" minOccurs="0"/>
          <xsd:element name="alive" type="SimpleTypes:TBoolean"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

```



```

        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tlicStart">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="comp" type="Types:TriComponentIdType"/>
                <xsd:element name="name" type="Types:TciBehaviourIdType"/>
                <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tlicRunning">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="comp" type="Types:TriComponentIdType"/>
                <xsd:element name="status" type="SimpleTypes:ComponentStatusType"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tlicAlive">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="comp" type="Types:TriComponentIdType"/>
                <xsd:element name="status" type="SimpleTypes:ComponentStatusType"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tlicStop">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="comp" type="Types:TriComponentIdType"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tlicKill">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="comp" type="Types:TriComponentIdType"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tlicDoneMismatch">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="comp" type="Types:TriComponentIdType"/>
                <xsd:element name="compTpl" type="Templates:TciNonValueTemplate"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tlicKilledMismatch">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="comp" type="Types:TriComponentIdType"/>
                <xsd:element name="compTpl" type="Templates:TciNonValueTemplate"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

```

```

<xsd:complexType name="tliCDone">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:Event">
      <xsd:sequence>
        <xsd:element name="compTmpl" type="Templates:TciNonValueTemplate"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliCKilled">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:Event">
      <xsd:sequence>
        <xsd:element name="compTmpl" type="Templates:TciNonValueTemplate"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliCTerminated">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:Event">
      <xsd:sequence>
        <xsd:element name="verdict" type="Values:VerdictValue" />
        <xsd:element name="reason" type="SimpleTypes:TString" minOccurs="0"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<!-- ports -->
<xsd:complexType name="tliPConnect">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:PortConfiguration"/>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliPDisconnect">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:PortConfiguration"/>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliPMap">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:PortConfiguration"/>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliPMapParam">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:tliPMap">
      <xsd:sequence>
        <xsd:element name="tciPars" type="Types:TciParameterListType" />
        <xsd:choice>
          <xsd:element name="encoder-failure" type="SimpleTypes:TciStatusType"
minOccurs="0"/>
          <xsd:element name="triPars" type="Types:TriParameterListType" />
        </xsd:choice>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliPUnmap">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:PortConfiguration"/>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliPUnmapParam">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:tliPUnmap">
      <xsd:sequence>
        <xsd:element name="tciPars" type="Types:TciParameterListType" />
        <xsd:choice>
          <xsd:element name="encoder-failure" type="SimpleTypes:TciStatusType"
minOccurs="0"/>

```

```

        <xsd:element name="triPars" type="Types:TriParameterListType"/>
    </xsd:choice>
</xsd:sequence>
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliPClear">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:PortStatus"/>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliPStart">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:PortStatus"/>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliPStop">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:PortStatus"/>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliPHalt">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:PortStatus"/>
    </xsd:complexContent>
</xsd:complexType>

<!-- codec -->
<xsd:complexType name="tliEncode">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="val" type="Values:Value"/>
                <xsd:choice>
                    <xsd:element name="msg" type="Types:TriMessageType"/>
                    <xsd:element name="encoder-failure" type="SimpleTypes:TciStatusType"
minOccurs="0"/>
                </xsd:choice>
                <xsd:element name="codec" type="SimpleTypes:TString"
                    minOccurs="0"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliDecode" mixed="true">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="msg" type="Types:TriMessageType"/>
                <xsd:choice>
                    <xsd:element name="decoder-failure" type="SimpleTypes:TciStatusType"
minOccurs="0"/>
                    <xsd:element name="val" type="Values:Value"/>
                </xsd:choice>
                <xsd:element name="codec" type="SimpleTypes:TString"
                    minOccurs="0"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<!-- timers -->
<xsd:complexType name="tliTimeoutDetected">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="timer" type="Types:TriTimerIdType" />
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliTimeoutMismatch">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">

```

```

        <xsd:sequence>
            <xsd:element name="timer" type="Types:TriTimerIdType" />
            <xsd:element name="timerTmpl" type="Templates:TciNonValueTemplate" />
        </xsd:sequence>
    </xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliTTimeout">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="timer" type="Types:TriTimerIdType" />
                <xsd:element name="timerTmpl" type="Templates:TciNonValueTemplate" />
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliTStart">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="timer" type="Types:TriTimerIdType"/>
                <xsd:element name="dur" type="SimpleTypes:TriTimerDurationType"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliTStop">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="timer" type="Types:TriTimerIdType"/>
                <xsd:element name="dur" type="SimpleTypes:TriTimerDurationType"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliTRead">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="timer" type="Types:TriTimerIdType"/>
                <xsd:element name="elapsed" type="SimpleTypes:TriTimerDurationType"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliTRunning">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="timer" type="Types:TriTimerIdType"/>
            </xsd:sequence>
            <xsd:attribute name="status" type="SimpleTypes:TimerStatusType"/>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<!-- scope -->
<xsd:complexType name="tliSEnter">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="name" type="Types:QualifiedName" />
                <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>
                <xsd:element name="kind" type="SimpleTypes:TString"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliSLeave">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>

```

```

        <xsd:element name="name" type="Types:QualifiedName" />
        <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>
        <xsd:element name="returnValue" type="Values:Value" minOccurs="0"/>
        <xsd:element name="kind" type="SimpleTypes:TString"/>
    </xsd:sequence>
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<!-- variables and module parameter -->
<xsd:complexType name="tliVar">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="name" type="Types:QualifiedName" />
                <xsd:element name="val" type="Values:Value" minOccurs="0"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliModulePar">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="name" type="Types:QualifiedName" />
                <xsd:element name="val" type="Values:Value" minOccurs="0"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<!-- verdicts -->
<xsd:complexType name="tliGetVerdict">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="verdict" type="Values:VerdictValue"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliSetVerdict">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="verdict" type="Values:VerdictValue"/>
                <xsd:element name="reason" type="SimpleTypes:TString" minOccurs="0"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<!-- log -->
<xsd:complexType name="tliLog">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="log" type="SimpleTypes:TString"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<!-- alt -->
<xsd:complexType name="tliAEnter">
    <xsd:complexContent>
        <xsd:extension base="Events:Event"/>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliALeave">
    <xsd:complexContent>
        <xsd:extension base="Events:Event"/>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliADefaults">
    <xsd:complexContent>

```

```

        <xsd:extension base="Events:Event"/>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliAActivate">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="name" type="Types:QualifiedName" />
                <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>
                <xsd:element name="ref" type="Values:Value"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliADeactivate">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="ref" type="Values:Value"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliANomatch">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event"/>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliARepeat">
    <xsd:complexContent>
        <xsd:extension base="Events:Event"/>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliAwait">
    <xsd:complexContent>
        <xsd:extension base="Events:Event"/>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliAction">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="action" type="SimpleTypes:TString"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliMatch">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="expr" type="Values:Value"/>
                <xsd:element name="tmpl" type="Templates:TciValueTemplate"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliMatchMismatch">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">
            <xsd:sequence>
                <xsd:element name="expr" type="Values:Value"/>
                <xsd:element name="tmpl" type="Templates:TciValueTemplate"/>
                <xsd:element name="diffs" type="Templates:TciValueDifferenceList"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tliInfo">
    <xsd:complexContent mixed="true">
        <xsd:extension base="Events:Event">

```

```

        <xsd:sequence>
            <xsd:element name="level" type="SimpleTypes:TInteger"/>
            <xsd:element name="info" type="SimpleTypes:TString"/>
        </xsd:sequence>
    </xsd:extension>
</xsd:complexContent>
</xsd:complexType>
</xsd:schema>

```

## B.6 TCI-TL XML schema for a log

```

<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://uri.etsi.org/ttcn-3/tci/TLI_v4_4_1.xsd"
    xmlns:TLI="http://uri.etsi.org/ttcn-3/tci/TLI_v4_4_1.xsd"
    xmlns:Events="http://uri.etsi.org/ttcn-3/tci/Events_v4_4_1.xsd"
    elementFormDefault="qualified">

    <xsd:import namespace="http://uri.etsi.org/ttcn-3/tci/Events_v4_4_1.xsd"
        schemaLocation="Events_v4_4_1.xsd"/>

    <xsd:element name="logfile" type="TLI:LogModule"/>
    <xsd:complexType name="LogModule">
        <xsd:sequence>
            <xsd:element name="header" type="TLI:Header"/>
            <xsd:element name="body" type="TLI:Body"/>
            <xsd:element name="trailer" type="TLI:Trailer" minOccurs="0"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="Header">
        <xsd:sequence>
            <!-- logging version -->
            <xsd:element name="version" type="xsd:string"/>
            <!-- begin of the log -->
            <xsd:element name="ts" type="xsd:long"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="Trailer">
        <xsd:choice>
            <xsd:any namespace="##any" processContents="skip" minOccurs="0" maxOccurs="unbounded"/>
        </xsd:choice>
    </xsd:complexType>

    <xsd:complexType name="Body">
        <xsd:choice maxOccurs="unbounded">

            <!-- test cases operations -->
            <xsd:element name="tliTcExecute" type="Events:tliTcExecute"/>
            <xsd:element name="tliTcStart" type="Events:tliTcStart"/>
            <xsd:element name="tliTcStop" type="Events:tliTcStop"/>
            <xsd:element name="tliTcStarted" type="Events:tliTcStarted"/>
            <xsd:element name="tliTcTerminated" type="Events:tliTcTerminated"/>

            <!-- control operations -->
            <xsd:element name="tliCtrlStart" type="Events:tliCtrlStart"/>
            <xsd:element name="tliCtrlStop" type="Events:tliCtrlStop"/>
            <xsd:element name="tliCtrlTerminated" type="Events:tliCtrlTerminated"/>

            <!-- asynchronous communication -->
            <xsd:element name="tliMSend_m" type="Events:tliMSend_m"/>
            <xsd:element name="tliMSend_c" type="Events:tliMSend_c"/>
            <xsd:element name="tliMSend_m_BC" type="Events:tliMSend_m_BC"/>
            <xsd:element name="tliMSend_c_BC" type="Events:tliMSend_c_BC"/>
            <xsd:element name="tliMSend_m_MC" type="Events:tliMSend_m_MC"/>
            <xsd:element name="tliMSend_c_MC" type="Events:tliMSend_c_MC"/>
            <xsd:element name="tliMDetected_m" type="Events:tliMDetected_m"/>
            <xsd:element name="tliMDetected_c" type="Events:tliMDetected_c"/>
            <xsd:element name="tliMMismatch_m" type="Events:tliMMismatch_m"/>
            <xsd:element name="tliMMismatch_c" type="Events:tliMMismatch_c"/>
            <xsd:element name="tliMReceive_m" type="Events:tliMReceive_m"/>
            <xsd:element name="tliMReceive_c" type="Events:tliMReceive_c"/>

            <!-- synchronous communication -->
            <xsd:element name="tliPrCall_m" type="Events:tliPrCall_m"/>
            <xsd:element name="tliPrCall_c" type="Events:tliPrCall_c"/>
            <xsd:element name="tliPrCall_m_BC" type="Events:tliPrCall_m_BC"/>
            <xsd:element name="tliPrCall_c_BC" type="Events:tliPrCall_c_BC"/>
            <xsd:element name="tliPrCall_m_MC" type="Events:tliPrCall_m_MC"/>

```

```

<xsd:element name="tliPrCall_c_MC" type="Events:tliPrCall_c_MC"/>

<xsd:element name="tliPrGetCallDetected_m" type="Events:tliPrGetCallDetected_m"/>
<xsd:element name="tliPrGetCallDetected_c" type="Events:tliPrGetCallDetected_c"/>
<xsd:element name="tliPrGetCallMismatch_m" type="Events:tliPrGetCallMismatch_m"/>
<xsd:element name="tliPrGetCallMismatch_c" type="Events:tliPrGetCallMismatch_c"/>
<xsd:element name="tliPrGetCall_m" type="Events:tliPrGetCall_m"/>
<xsd:element name="tliPrGetCall_c" type="Events:tliPrGetCall_c"/>

<xsd:element name="tliPrReply_m" type="Events:tliPrReply_m"/>
<xsd:element name="tliPrReply_c" type="Events:tliPrReply_c"/>
<xsd:element name="tliPrReply_m_BC" type="Events:tliPrReply_m_BC"/>
<xsd:element name="tliPrReply_c_BC" type="Events:tliPrReply_c_BC"/>
<xsd:element name="tliPrReply_m_MC" type="Events:tliPrReply_m_MC"/>
<xsd:element name="tliPrReply_c_MC" type="Events:tliPrReply_c_MC"/>

<xsd:element name="tliPrGetReplyDetected_m" type="Events:tliPrGetReplyDetected_m"/>
<xsd:element name="tliPrGetReplyDetected_c" type="Events:tliPrGetReplyDetected_c"/>
<xsd:element name="tliPrGetReplyMismatch_m" type="Events:tliPrGetReplyMismatch_m"/>
<xsd:element name="tliPrGetReplyMismatch_c" type="Events:tliPrGetReplyMismatch_c"/>
<xsd:element name="tliPrGetReply_m" type="Events:tliPrGetReply_m"/>
<xsd:element name="tliPrGetReply_c" type="Events:tliPrGetReply_c"/>

<xsd:element name="tliPrRaise_m" type="Events:tliPrRaise_m"/>
<xsd:element name="tliPrRaise_c" type="Events:tliPrRaise_c"/>
<xsd:element name="tliPrRaise_m_BC" type="Events:tliPrRaise_m_BC"/>
<xsd:element name="tliPrRaise_c_BC" type="Events:tliPrRaise_c_BC"/>
<xsd:element name="tliPrRaise_m_MC" type="Events:tliPrRaise_m_MC"/>
<xsd:element name="tliPrRaise_c_MC" type="Events:tliPrRaise_c_MC"/>

<xsd:element name="tliPrCatchDetected_m" type="Events:tliPrCatchDetected_m"/>
<xsd:element name="tliPrCatchDetected_c" type="Events:tliPrCatchDetected_c"/>
<xsd:element name="tliPrCatchMismatch_m" type="Events:tliPrCatchMismatch_m"/>
<xsd:element name="tliPrCatchMismatch_c" type="Events:tliPrCatchMismatch_c"/>
<xsd:element name="tliPrCatch_m" type="Events:tliPrCatch_m"/>
<xsd:element name="tliPrCatch_c" type="Events:tliPrCatch_c"/>

<xsd:element name="tliPrCatchTimeoutDetected"
                type="Events:tliPrCatchTimeoutDetected"/>
<xsd:element name="tliPrCatchTimeout" type="Events:tliPrCatchTimeout"/>

<!-- components -->
<xsd:element name="tliCCreate" type="Events:tliCCreate"/>
<xsd:element name="tliCStart" type="Events:tliCStart"/>
<xsd:element name="tliCRunning" type="Events:tliCRunning"/>
<xsd:element name="tliCALive" type="Events:tliCALive"/>
<xsd:element name="tliCStop" type="Events:tliCStop"/>
<xsd:element name="tliCKill" type="Events:tliCKill"/>
<xsd:element name="tliCDoneMismatch" type="Events:tliCDoneMismatch"/>
<xsd:element name="tliCDone" type="Events:tliCDone"/>
<xsd:element name="tliCKilledMismatch" type="Events:tliCKilledMismatch"/>
<xsd:element name="tliCKilled" type="Events:tliCKilled"/>
<xsd:element name="tliCTerminated" type="Events:tliCTerminated"/>

<!-- ports -->
<xsd:element name="tliPConnect" type="Events:tliPConnect"/>
<xsd:element name="tliPDisconnect" type="Events:tliPDisconnect"/>
<xsd:element name="tliPMap" type="Events:tliPMap"/>
<xsd:element name="tliPMapParam" type="Events:tliPMapParam"/>
<xsd:element name="tliPUnmap" type="Events:tliPUnmap"/>
<xsd:element name="tliPUnmapParam" type="Events:tliPUnmapParam"/>
<xsd:element name="tliPClear" type="Events:tliPClear"/>
<xsd:element name="tliPStart" type="Events:tliPStart"/>
<xsd:element name="tliPStop" type="Events:tliPStop"/>
<xsd:element name="tliPHalt" type="Events:tliPHalt"/>

<!-- codec -->
<xsd:element name="tliDecode" type="Events:tliDecode"/>
<xsd:element name="tliEncode" type="Events:tliEncode"/>

<!-- timers -->
<xsd:element name="tliTTimeoutDetected" type="Events:tliTTimeoutDetected"/>
<xsd:element name="tliTTimeoutMismatch" type="Events:tliTTimeoutMismatch"/>
<xsd:element name="tliTTimeout" type="Events:tliTTimeout"/>
<xsd:element name="tliTStart" type="Events:tliTStart"/>
<xsd:element name="tliTStop" type="Events:tliTStop"/>
<xsd:element name="tliTRead" type="Events:tliTRead"/>
<xsd:element name="tliTRunning" type="Events:tliTRunning"/>

<!-- scopes -->

```



```

<xsd:element name="tliSEnter" type="Events:tliSEnter"/>
<xsd:element name="tliSLeave" type="Events:tliSLeave"/>

<!-- statements -->
<xsd:element name="tliVar" type="Events:tliVar"/>
<xsd:element name="tliModulePar" type="Events:tliModulePar"/>
<xsd:element name="tliGetVerdict" type="Events:tliGetVerdict"/>
<xsd:element name="tliSetVerdict" type="Events:tliSetVerdict"/>
<xsd:element name="tliLog" type="Events:tliLog"/>

<!-- alt -->
<xsd:element name="tliAEnter" type="Events:tliAEnter"/>
<xsd:element name="tliALeave" type="Events:tliALeave"/>
<xsd:element name="tliADefaults" type="Events:tliADefaults"/>
<xsd:element name="tliAActivate" type="Events:tliAActivate"/>
<xsd:element name="tliADeactivate" type="Events:tliADeactivate"/>
<xsd:element name="tliANomatch" type="Events:tliANomatch"/>
<xsd:element name="tliARepeat" type="Events:tliARepeat"/>
<xsd:element name="tliAWait" type="Events:tliAWait"/>

<!-- action -->
<xsd:element name="tliAction" type="Events:tliAction"/>

<!-- match -->
<xsd:element name="tliMatch" type="Events:tliMatch"/>
<xsd:element name="tliMatchMismatch" type="Events:tliMatchMismatch"/>

<!-- info -->
<xsd:element name="tliInfo" type="Events:tliInfo"/>
</xsd:choice>
</xsd:complexType>
</xsd:schema>

```

## Appendix I

### Use scenarios

(This appendix does not form an integral part of this Recommendation.)

This appendix contains use scenarios that should help users of the TCI and tool vendors providing the TCI understand the semantics of the operations defined within the present document.

The scenarios are defined in terms of UML sequence diagrams. The sequence diagram shows the interactions between the TCI entities. The scenarios are explained and where applicable underpinned with a TTCN-3 fragment corresponding to the scenario.

#### **I.1 Initialization, collecting information, logging**

##### **I.1.1 Use scenario: initialization**

The scenario in Figure I.1 shows the initialization phase for a test system when a TTCN-3 module is to be selected for execution. At first, a root module has to be set with `tciRootModule`. The module parameters of the root module can be obtained with `tciGetModuleParameters`. Module parameter information can be used to ask the test system user for concrete values for each module parameter. The list of test cases available in the root module can be retrieved with `tciGetTestCases`. These test cases can be directly executed from the test management. Their parameters and their test system interface can be obtained with `tciGetTestCaseParameters` and `tciGetTestCaseTSI`, respectively.

### I.1.1.1 Sequence diagram

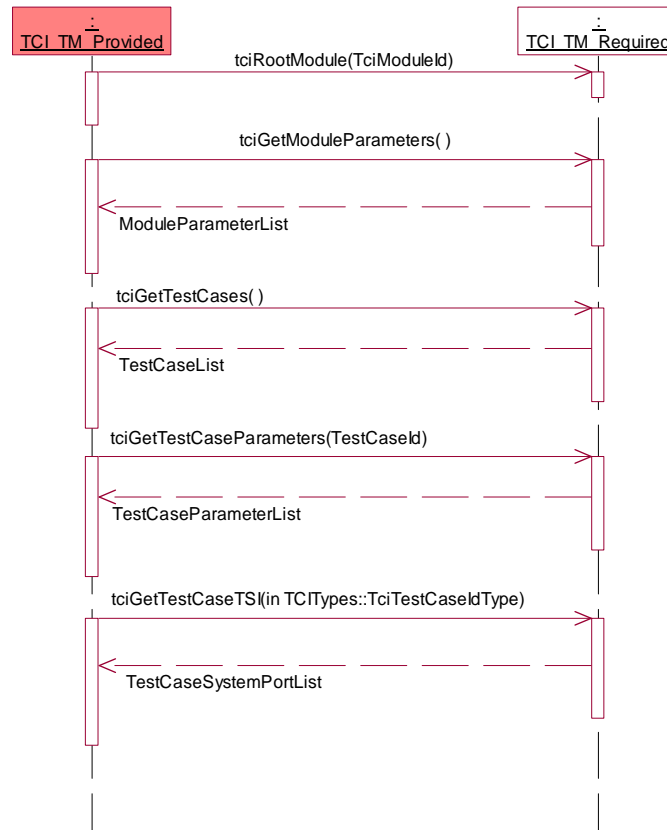


Figure I.1 – Use scenario – initialization

### I.1.1.2 TTCN-3 fragment

The initialization is outside the scope of TTCN-3.

### I.1.2 Use scenario: requesting module parameters

The scenario in Figure I.2 shows how a test component requests the actual value of a module parameter needed for the execution of its test behaviour. At first, the type of a module parameter is requested, then the value can be constructed by the TM and given to the TE.

### I.1.2.1 Sequence diagram

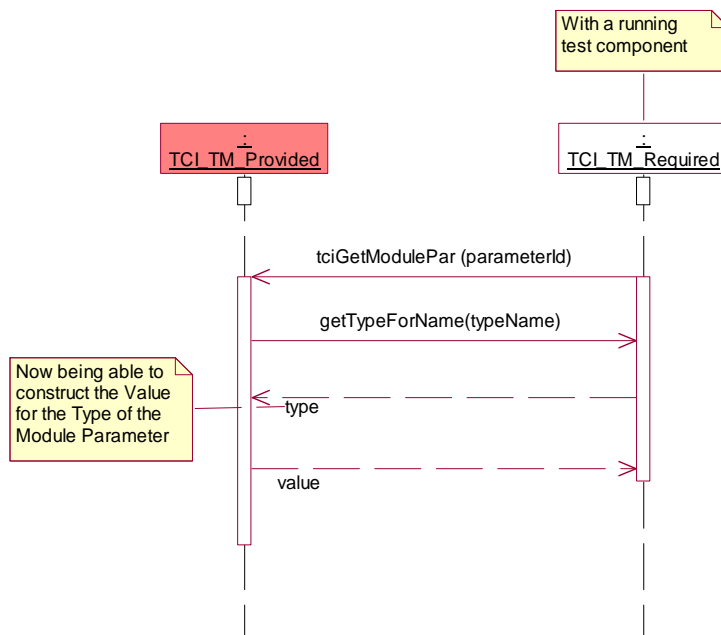


Figure I.2 – Use scenario – requesting module Pars

### I.1.2.2 TTCN-3 fragment

```
module AModule {  
  ...  
  modulepar {  
    integer AModulePar  
  }  
  ...  
  function AFunction (...) ... {  
    integer x;  
    ...  
    x:= 2+AModulePar; // an expression with a module parameter  
    ...  
  }  
  ...  
}
```

### I.1.3 Use scenario: logging

The scenario in Figure I.3 shows logging of information during the execution of a test behaviour by a test component. The message to be logged is propagated to the test logging.

### I.1.3.1 Sequence diagram

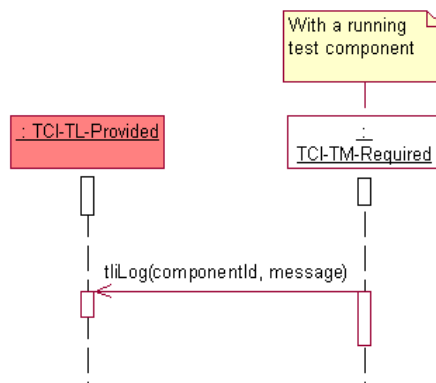


Figure I.3 – Use scenario – logging

### I.1.3.2 TTCN-3 fragment

```

module AModule {
  ...
  function AFunction (...) ... {
    ...
    log("AMessage");
    ...
  }
  ...
}
  
```

## I.2 Execution of test cases and control

### I.2.1 Use scenario: execution of control

The scenario in Figure I.4 shows the sequence of operations to execute the control part of a TTCN-3 module. The module containing the control part is selected first, then the control is started, then it is executed until the execution is terminated by TE.

#### I.2.1.1 Sequence diagram

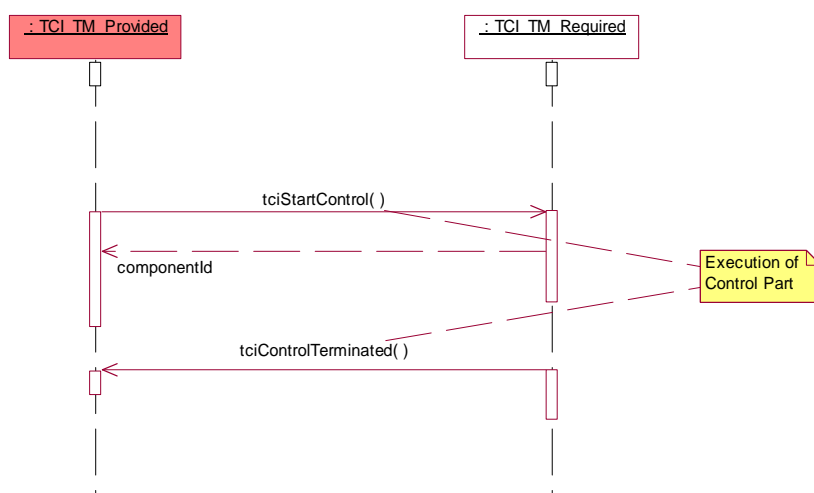


Figure I.4 – Use scenario – execution of control

#### I.2.1.2 TTCN-3 fragment

```

module AModule {
  ...
  
```

```

control {
  ...
}
...
}

```

## I.2.2 Use scenario: test case execution within control

The scenario in Figure I.5 shows how a test case is executed within the control part.

### I.2.2.1 Sequence diagram

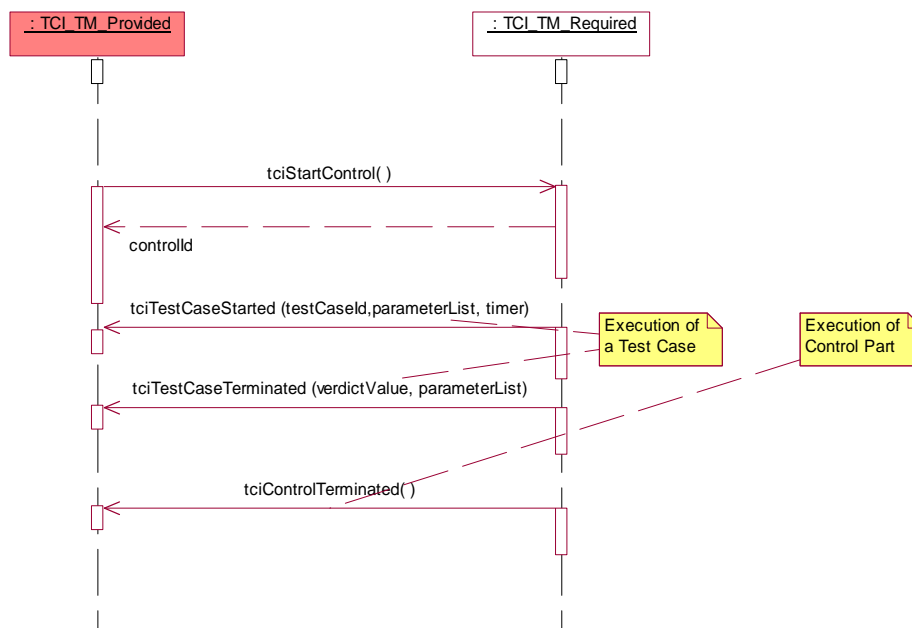


Figure I.5 – Use scenario – test case execution within control

### I.2.2.2 TTCN-3 fragment

```

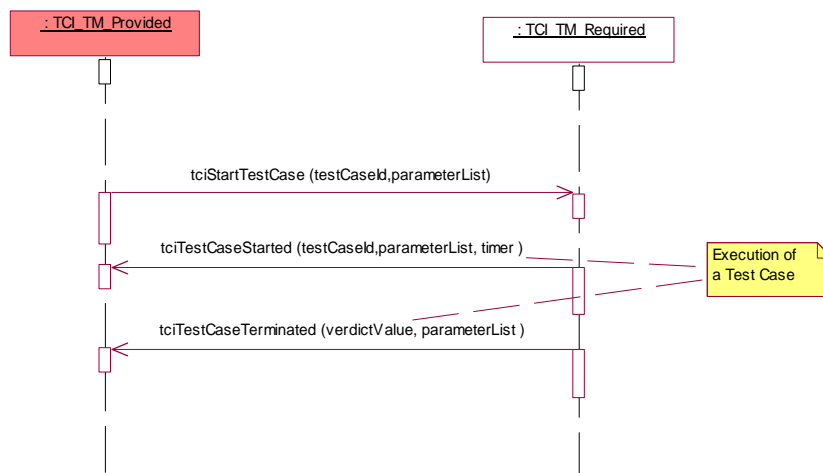
module AModule {
  ...
  testcase ATestCase(...)... {
    ... //the test case behaviour
  }
  ...
  control {
    ...
    execute (ATestCase (...));
    ...
  }
  ...
}

```

## I.2.3 Use scenario: direct test case execution

The scenario in Figure I.6 shows how a test case can be directly executed from the test management outside the control part. After selecting the TTCN-3 module containing the test case to be executed, the start of the test case is requested. When the test case completes its execution, the test management is informed by the TE of the test case termination.

### I.2.3.1 Sequence diagram



**Figure I.6 – Use scenario – direct test case execution**

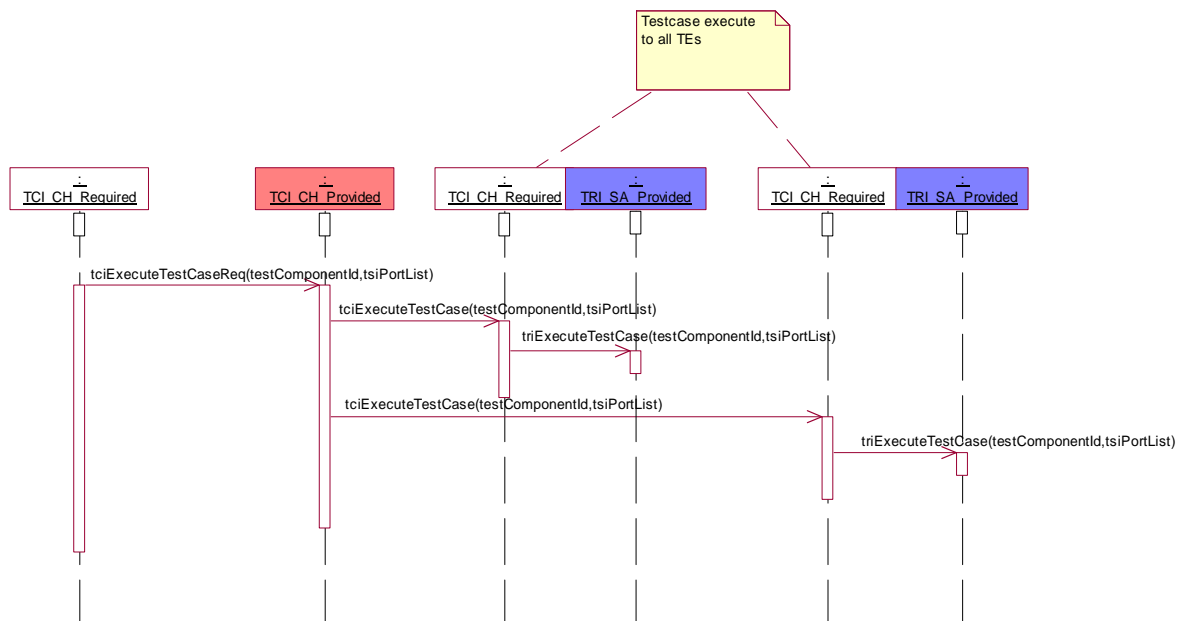
### I.2.3.2 TTCN-3 fragment

The direct execution of a test case is outside the scope of TTCN-3.

### I.2.4 Use scenario: execute test case to TRI

The scenario in Figure I.7 shows how the TRI is informed about the execution of a test case so that it can set up and initialize system ports when needed. The execute test case request has to be issued before the test behaviour on the MTC of the current test case is started.

#### I.2.4.1 Sequence diagram



**Figure I.7 – Use scenario – execute test case to TRI**

### I.2.4.2 TTCN-3 fragment

```

module AModule {
  ...
  testcase ATestCase(...)... {

```

```

... //the test case behaviour
}
...
control {
...
execute(ATestCase (...));
...
}
...
}

```

### I.3 Component handling

#### I.3.1 Use scenario: local control component creation

The scenario in Figure I.8 demonstrates the creation of the control component on the same node where the user interface to the test management TCI-TM resides. A control component is created whenever the control part of a TTCN-3 module is executed. Whenever the test management TCI-TM issues the start of the control part, a create test component request is sent to the TCI-CH, which propagates it to the TE where the control component should be created. In this case it is the TE on the same node. The identifier for the control component is returned and given to the TCI-TM. The identifier is then used to start the behaviour of the control part on the control component.

##### I.3.1.1 Sequence diagram

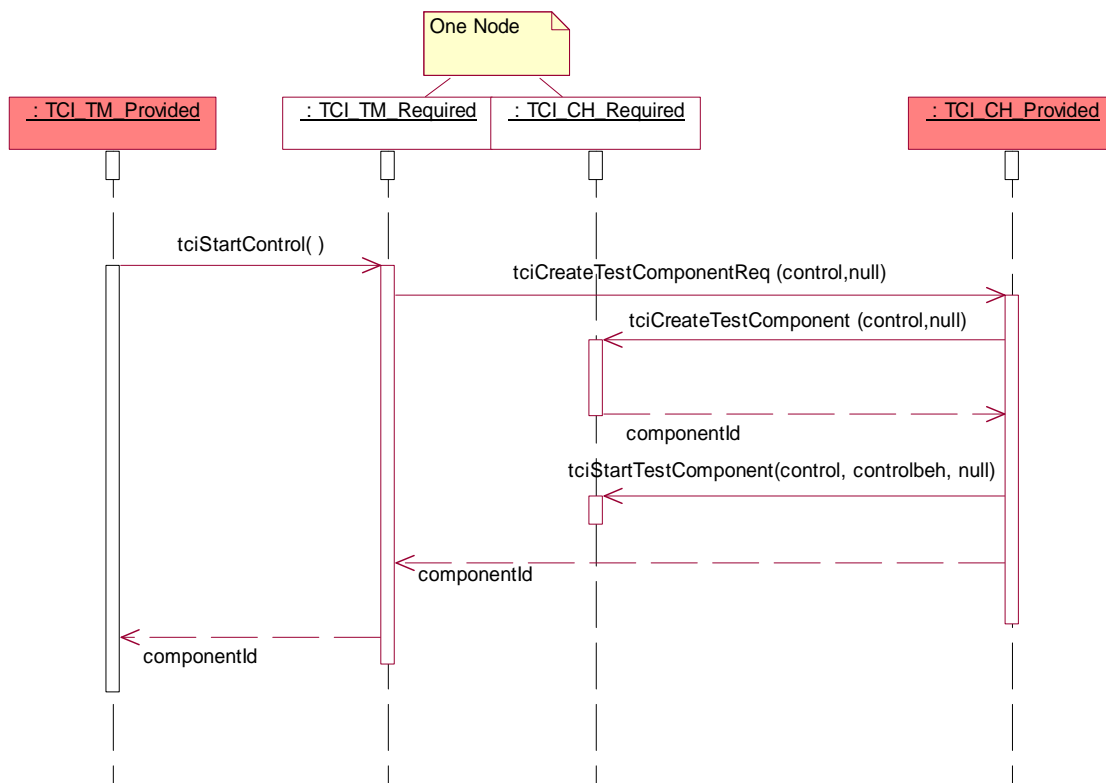


Figure I.8 – Use scenario – local control component creation

##### I.3.1.2 TTCN-3 fragment

```

module AModule {
...
control {
...
}
...
}

```



}

### I.3.2 Use scenario: remote control component creation

The scenario in Figure I.9 demonstrates the creation of the control component on another node than where the user interface to the test management TCI-TM resides. A control component is created whenever the control part of a TTCN-3 module is executed. Whenever the test management TCI-TM issues the start of the control part, a create test component request is sent to the TCI-CH, which propagates it to the TE where the control component should be created. In this case it is the TE on another remote node. The identifier for the control component is returned and given to the TCI-TM. The identifier is then used to start the behaviour of the control part on the control component.

#### I.3.2.1 Sequence diagram

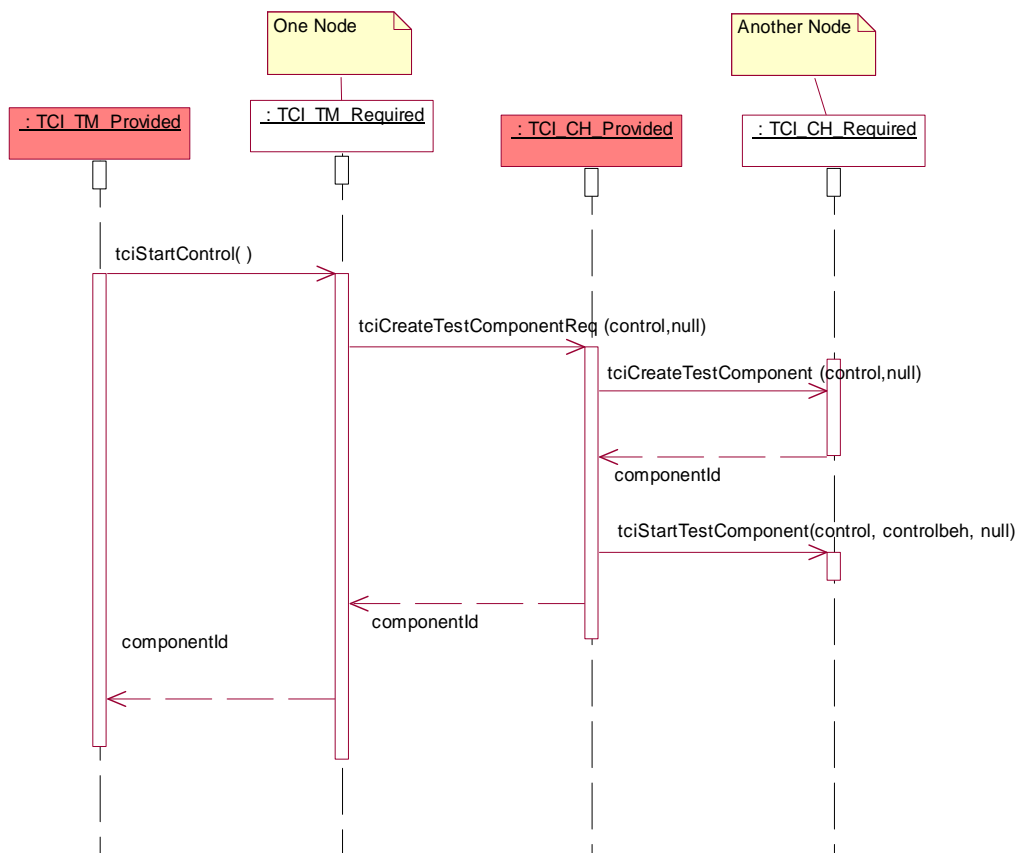


Figure I.9 – Use scenario – remote control component creation

#### I.3.2.2 TTCN-3 fragment

```
module AModule {
    ...
    control {
        ...
    }
    ...
}
```

### I.3.3 Use scenario: local MTC creation

The scenario in Figure I.10 demonstrates the local creation of the main test component. Local is meant for two cases:

- 1) on the same node where the user interface to the test management TCI-TM resides (when a test case is started directly); or
- 2) on the same node where the control component resides (when a test case is executed from a control part).

A main test component is created whenever a test case is executed: a create test component request is sent to the TCI-CH, which propagates it to the TE where the main test component should be created. In this case it is the TE on the same node. The identifier for the main test component is returned and given to the TCI-TM. The identifier is then used to start the test case behaviour on the main test component (this is not shown here, but handled the same way as in the scenarios described in clauses I.2.2 and I.2.3).

### I.3.3.1 Sequence diagram

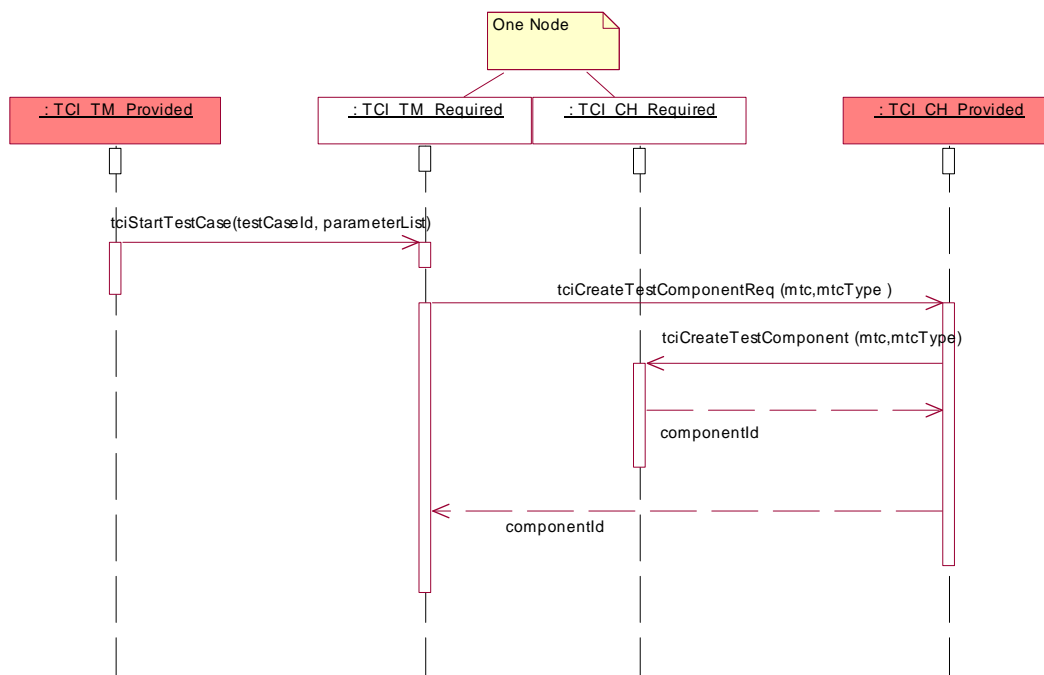


Figure I.10 – Use scenario – local MTC creation

### I.3.3.2 TTCN-3 fragment

```

module AModule {
  ...
  testcase ATestCase (...) runs on MTCType... {
    ... //the test case behaviour
  }
  ...
}
  
```

### I.3.4 Use scenario: remote MTC creation

The scenario in Figure I.11 demonstrates the remote creation of the main test component. Remote is meant for two cases:

- 1) on another node than where the user interface to the test management TCI-TM resides (when a test case is started directly); or
- 2) on another node than where the control component resides (when a test case is executed from a control part).

A main test component is created whenever a test case is executed: a create test component request is sent to the TCI-CH, which propagates it to the TE where the main test component should be created. In this case it is the TE on another node. The identifier for the main test component is returned and given to the TCI-TM. The identifier is then used to start the test case behaviour on the main test component (this is not shown here, but handled the same way as in the scenarios described in clauses I.2.2 and I.2.3).

### I.3.4.1 Sequence diagram

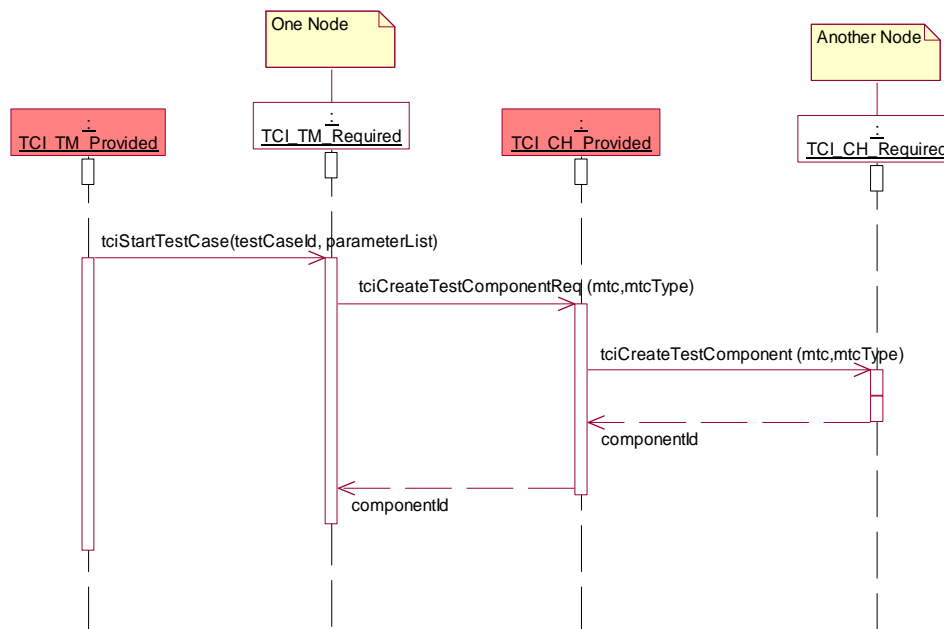


Figure I.11 – Use scenario – remote MTC creation

### I.3.4.2 TTCN-3 fragment

```

module AModule {
  ...
  testcase ATestCase(...) runs on MTCType ... {
    ... //the test case behaviour
  }
  ...
}

```

### I.3.5 Use scenario: component handling for test case execution within control

The scenario in Figure I.12 demonstrates the handling of components for the test case execution within a control part. When the control part is started, a control component is created and its component identifier returned to the test management. For each test case to be executed within the control part, a main test component is created and the component identifier returned to the control component. Afterwards, the test case behaviour is started on the main test component and the test management is informed about the start of the test case. When the main test component terminates, a request for the main test component termination together with the local verdict of the main test component is propagated to enable the derivation of the global test verdict and to enable the information about the test case termination.

### I.3.5.1 Sequence diagram

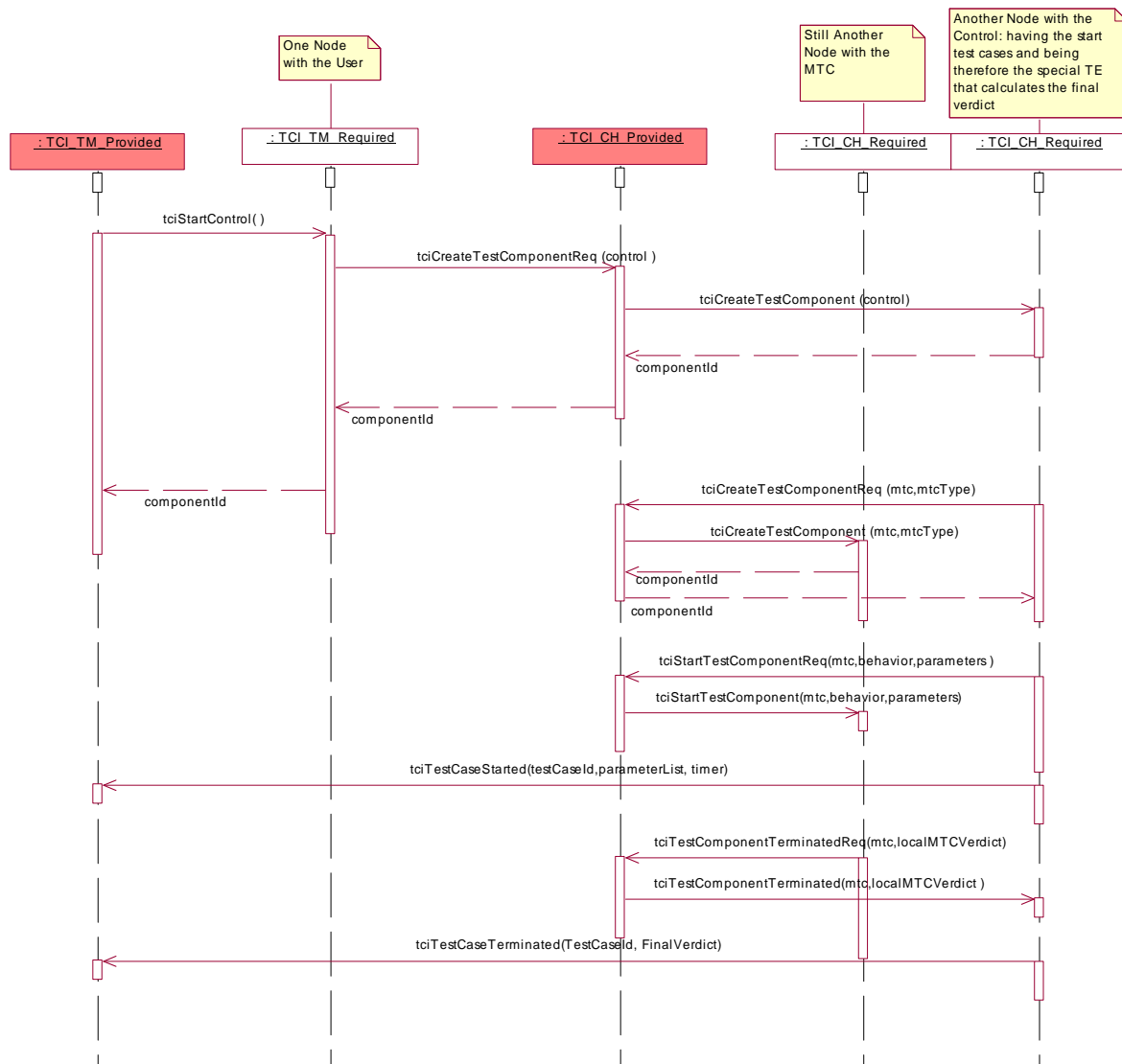


Figure I.12 – Use scenario: component handling for test case execution within control

### I.3.5.2 TTCN-3 fragment

```

module AModule {
  ...
  testcase ATestCase(...) ... {
    ... //the test case behaviour
  }
  ...
  control {
    ...
    execute (ATestCase (...));
    ...
  }
  ...
}

```

### I.3.6 Use scenario: component handling for direct test case execution

The scenario in Figure I.13 shows how test components are handled when a test case is executed directly, i.e., outside a control part. When a test case is started, the main test component is created and the test case behaviour started on this main test component at first. Whenever a parallel test component is used within a test case, it is handled the same: the parallel test component is started

first: giving a test component create request to the TCI-CH entity, which propagates the test component create to the TE in which the parallel test component is to be created. The identifier for the created parallel test component is returned. The identifier is then used to start the PTC behaviour for the start operation. When the PTC terminates its execution, a test component terminate request together with the local test verdict is issued to inform TCI-CH about this termination. The same is done when the main test component terminates. In addition, the termination of the main test component leads to the overall termination of the test case.

### I.3.6.1 Sequence diagram

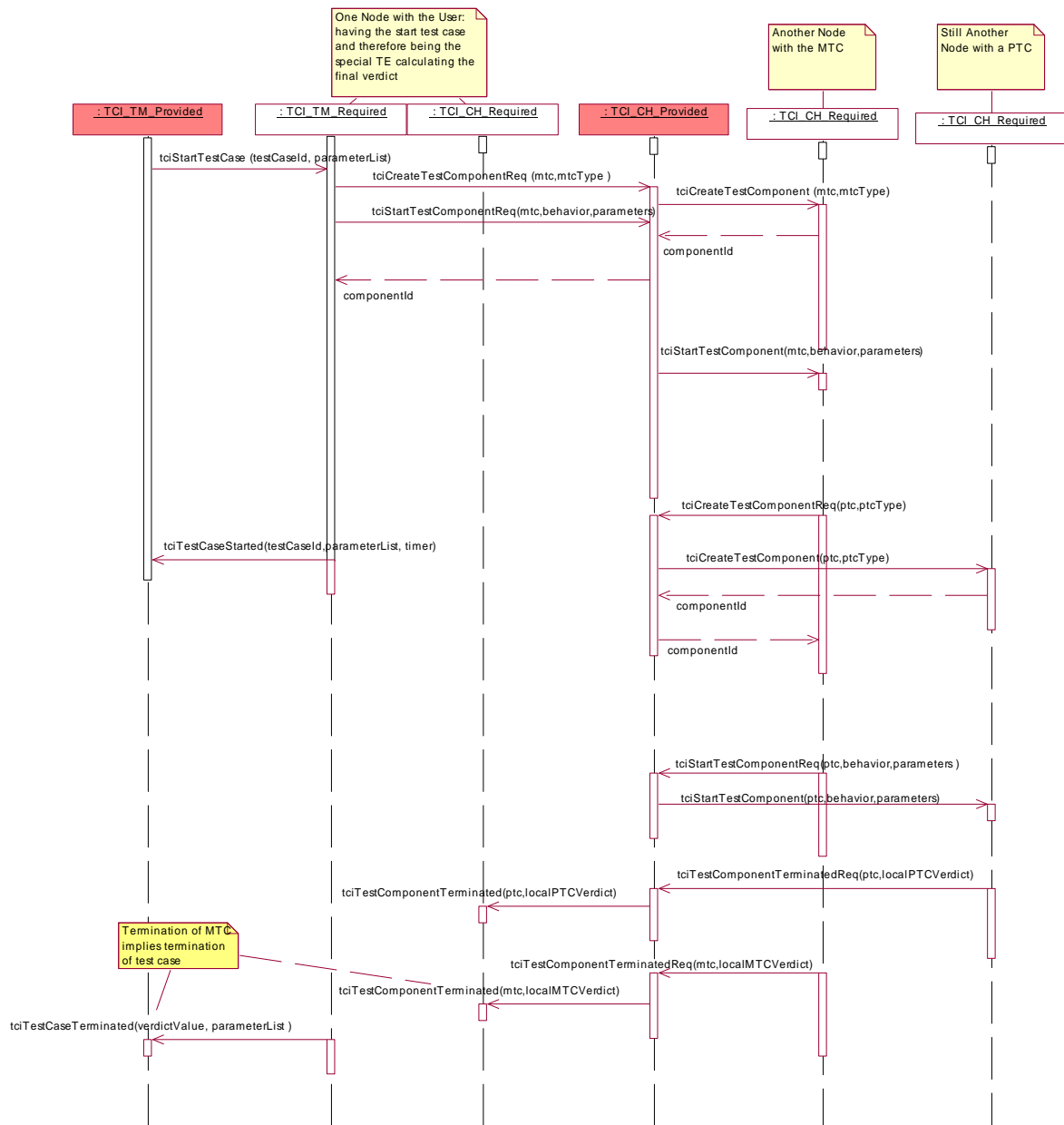


Figure I.13 – Use scenario: component handling for direct test case execution

### I.3.6.2 TTCN-3 fragment

```

module AModule {
  ...
  function APTCBehaviour(...) runs on APTCType {
    ... //the PTC behaviour
  }
  ...
}

```

```

testcase ATestCase(...)... {
  ... //the test case behaviour
  var APTCType PTC:= APTCType.create;
  ...
  PTC.start (APTCCbehaviour (...));
  ...
}
...
}

```

### I.3.7 Use scenario: propagation of map/connect

The scenario in Figure I.14 shows how ports are mapped. The request to map a port is propagated to the TE where the map is finally performed. The propagation of connect requests works analogously.

#### I.3.7.1 Sequence diagram

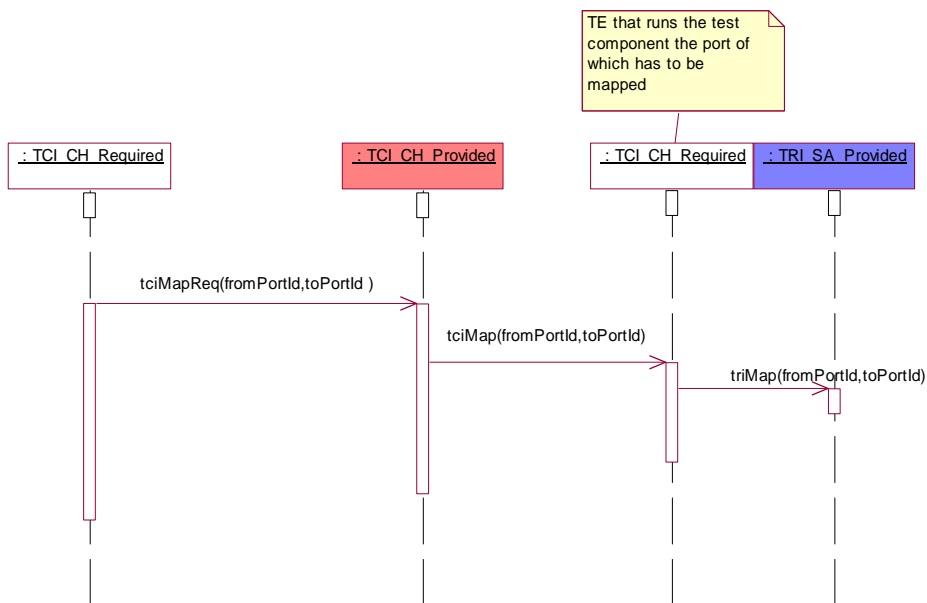


Figure I.14 – Use scenario: propagation of map

#### I.3.7.2 TTCN-3 fragment

```

module AModule {
  ...
  type port A { ... }
  type component CA { port A a }
  type component CB { port A a }
  ...
  testcase ATestCase(...)runs on CA system CB {
    var CA ptc := CA.create;
    ... //the test case behaviour
    map(ptc:a, System:a);
  }
  ...
}

```

### I.3.8 Use scenario: propagation of unmap/disconnect

The scenario in Figure I.15 shows how ports are unmapped. The request to unmap a port is propagated to the TE where the unmap is finally performed. The propagation of disconnect requests works analogously.

### I.3.8.1 Sequence diagram

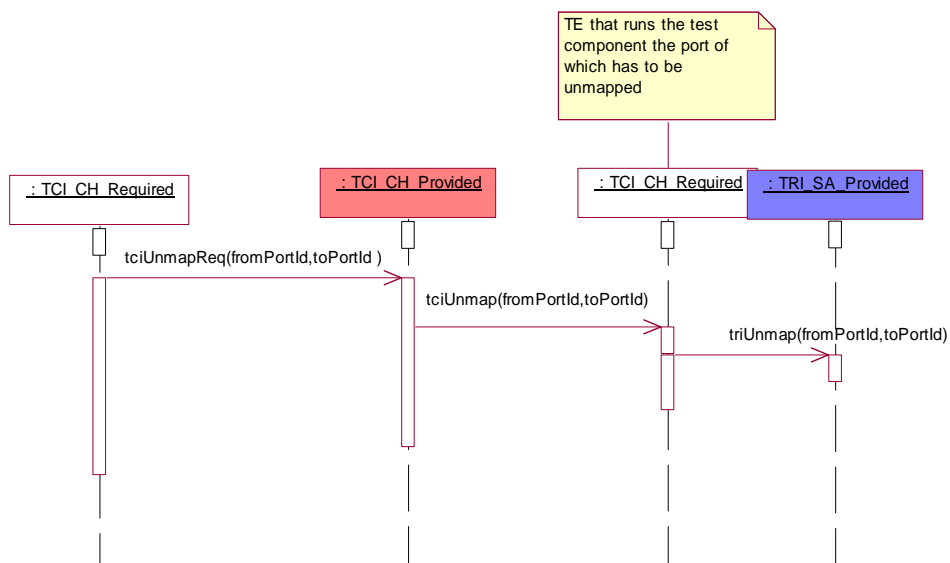


Figure I.15 – Use scenario – propagation of map

### I.3.8.2 TTCN-3 fragment

```

module AModule {
  ...
  type port A { ... }
  type component CA { port A a }
  type component CB { port A a }
  ...
  testcase ATestCase(...) runs on CA system CB {
    var CA ptc := CA.create;
    ... //the test case behaviour
    unmap(ptc:a,system:a);
    ...
  }
  ...
}

```

## I.4 Termination of test cases and control

### I.4.1 Use scenario: stop a test case

The scenario in Figure I.16 shows how a test case is stopped from the test management during test case execution. Once the TM has received information about a started test case, a stop test case can be requested up until receiving the information that the test case has been terminated. Upon stopping a test case, all parallel test components will be stopped and the test system will be reset.

### I.4.1.1 Sequence diagram

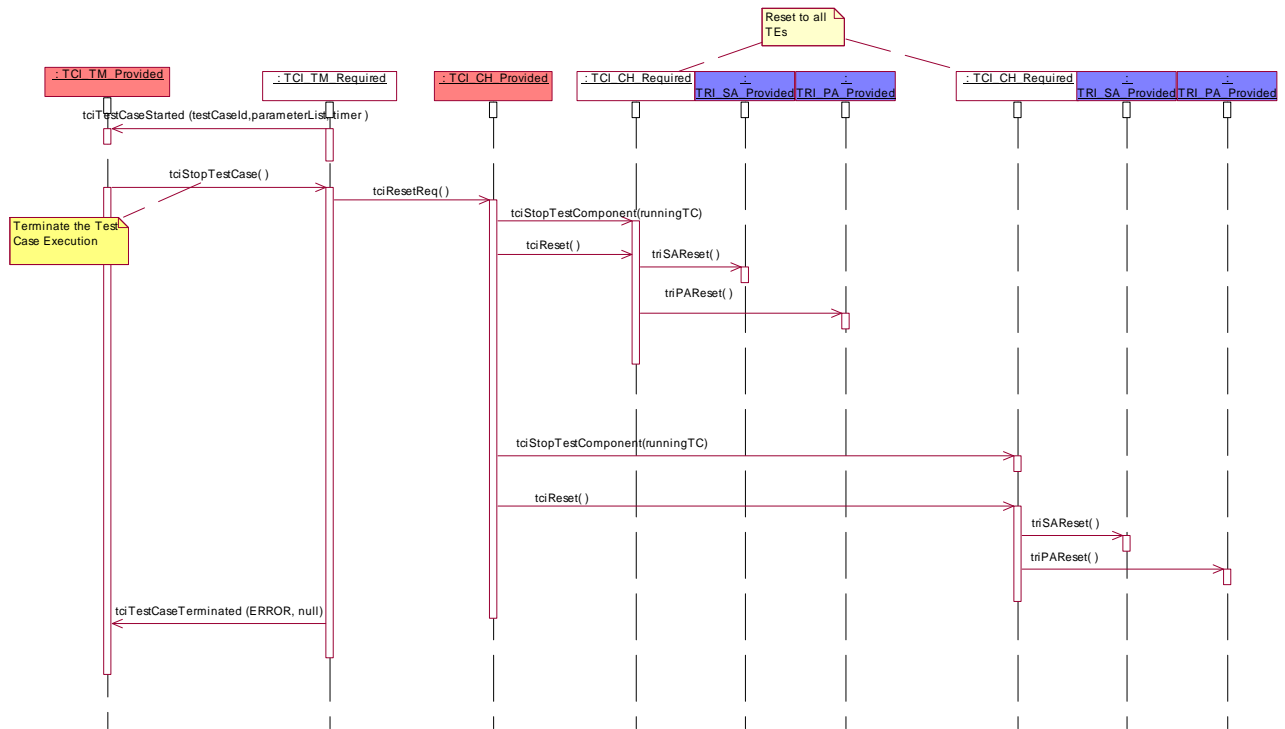


Figure I.16 – Use scenario: stop a test case

### I.4.1.2 TTCN-3 fragment

There is no TTCN-3 code related to how the TM chooses to implement test case termination. This is outside the scope of TTCN-3.

### I.4.2 Use scenario: stop control

The scenario in Figure I.17 shows how a control part is stopped from the test management during control part execution. A control part can be stopped in between starting the control and its termination. If the control part receives a stop test case request while a test case is executing, the executing test case is to be stopped. Furthermore, the test system is to be reset as described in Figure I.16.



### I.4.2.1 Sequence diagram

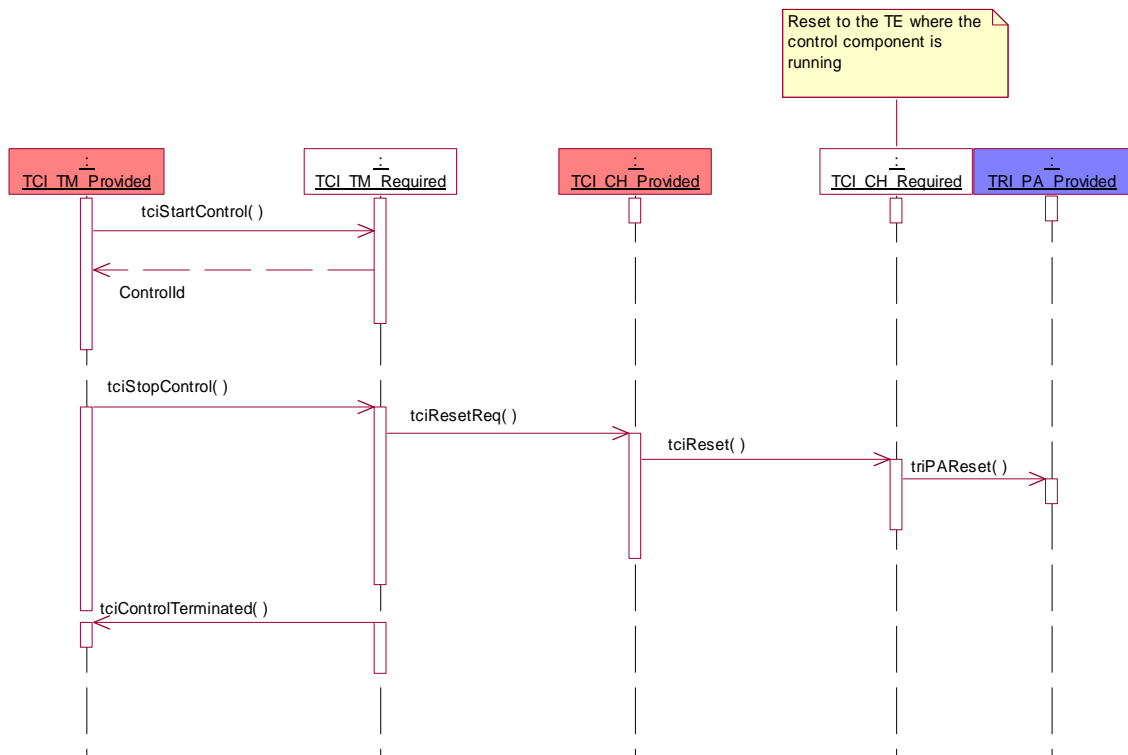


Figure I.17 – Use scenario – stop control

### I.4.2.2 TTCN-3 fragment

Stopping a control part from the test management is outside the scope of TTCN-3 so that no TTCN-3 fragment exists.

### I.4.3 Use scenario: termination of control after error

The scenario in Figure I.18 shows the handling of error situations during the execution of a control part when no test case is being executed. The test management is informed about the error situation and has then to terminate the execution of the control part explicitly. Upon termination of the control part, the test system will be reset.

### I.4.3.1 Sequence diagram

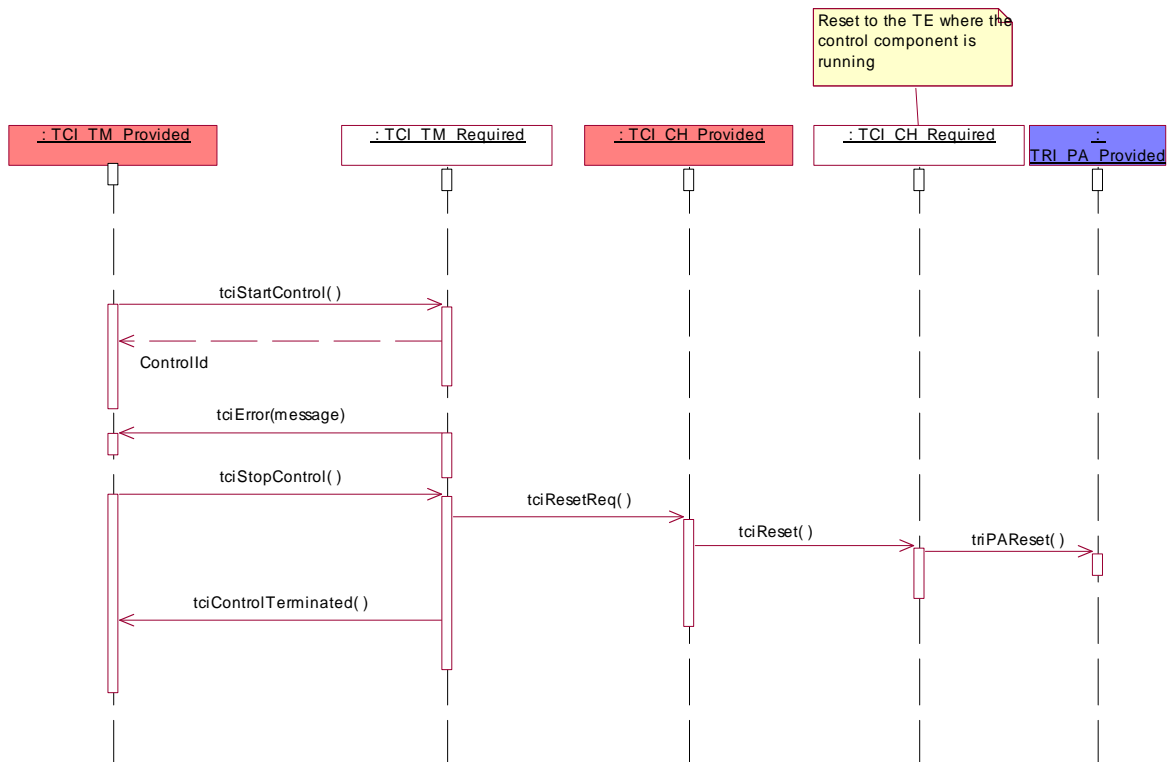


Figure I.18 – Use scenario – termination of control after error

### I.4.3.2 TTCN-3 fragment

There is no TTCN-3 fragment for this scenario since error situations are exceptional cases in a test system and not a TTCN-3 concept as such. Rather, the TTCN-3 semantics describes various potential error situations in a test system.

### I.4.4 Use scenario: termination of a test case after error

The scenario in Figure I.19 shows the handling of error situations during the direct execution of a test case. The test management is informed about the error situation. The TM has then to explicitly terminate test case execution. Upon stopping a test case, the parallel test components will be stopped and the test system is to be reset.

### I.4.4.1 Sequence diagram

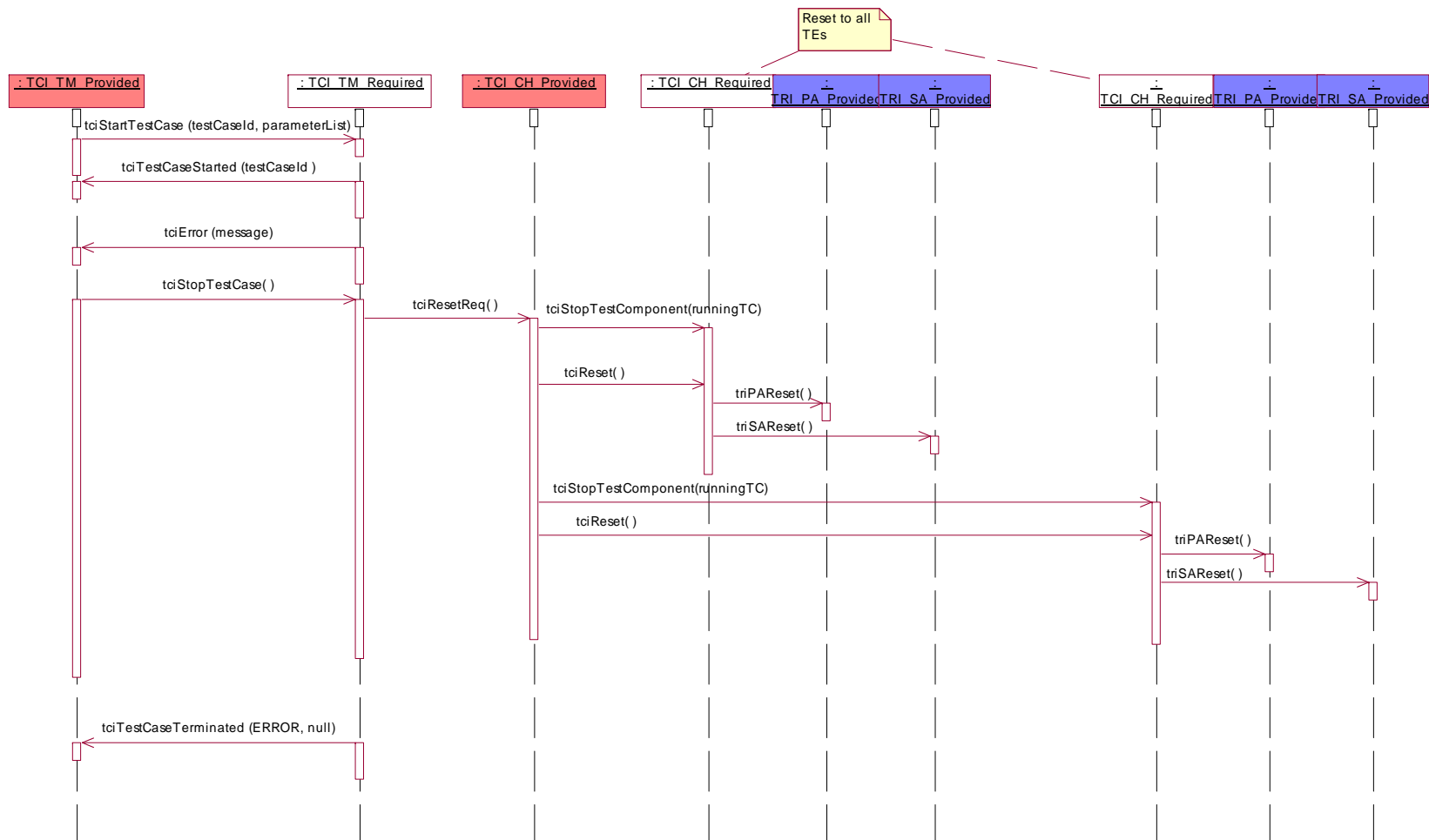


Figure I.19 – Use scenario – termination of a test case after error

#### I.4.4.2 TTCN-3 fragment

There is no TTCN-3 fragment for this scenario since error situations are exceptional cases in a test system and not a TTCN-3 concept as such. Rather, the TTCN-3 semantics describes various potential error situations in a test system.

#### I.4.5 Use scenario: reset

The scenario in Figure I.20 shows the reset of the test system. In that case all involved TEs together with their TRI system adaptors (SA) and platform adaptors (PA) are reset.

##### I.4.5.1 Sequence diagram

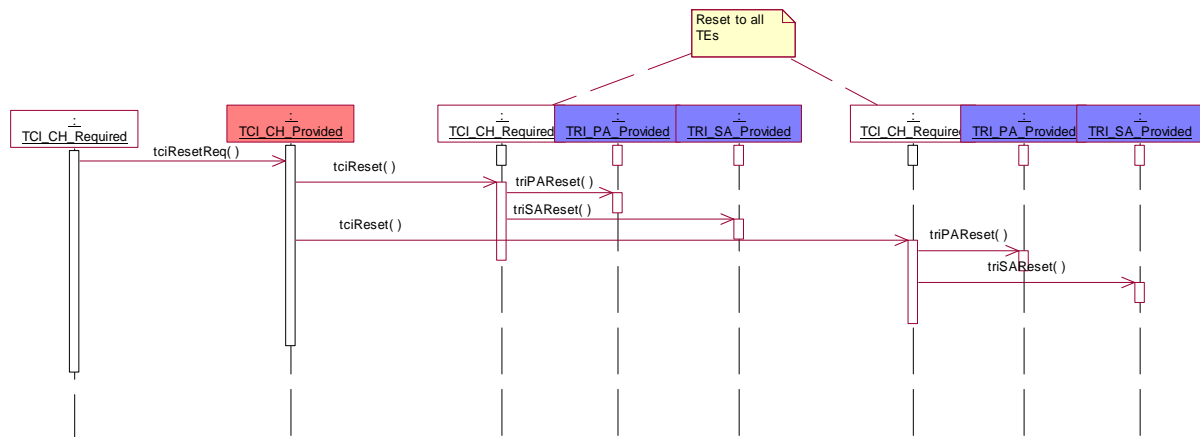


Figure I.20 – Use scenario – reset

#### I.4.5.2 TTCN-3 fragment

There is no TTCN-3 fragment for this scenario since reset as required after error situations are exceptional cases in a test system and not a TTCN-3 concept as such.

### I.5 Communication

#### I.5.1 Use scenario: local intercomponent communication

The scenario in Figure I.21 shows the communication between test components (main test component or parallel test components), which reside on the same node. A communication request is given to the TCI-CH, which then decide where to enqueue this communication template. In this case, the communication is done locally via the TE on the same node. The scenario shows a message-based communication using the send operation – the scenario is the same for call, reply, and raise operations.

### I.5.1.1 Sequence diagram

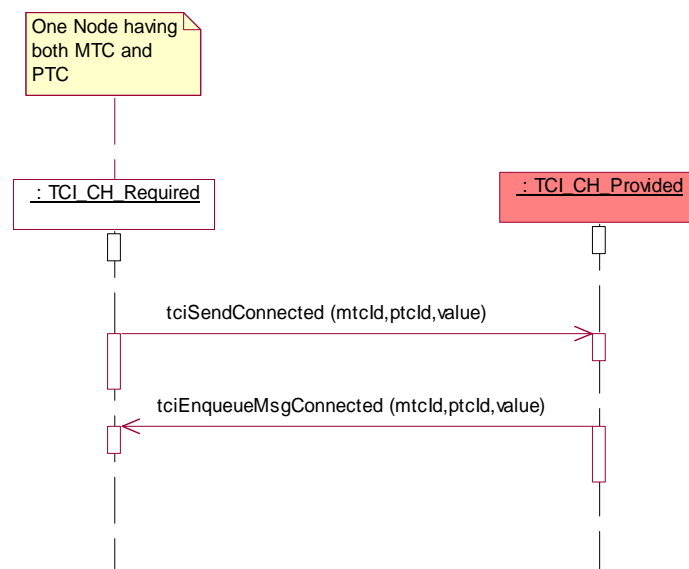


Figure I.21 – Use scenario – local intercomponent communication

### I.5.1.2 TTCN-3 fragment

```

module AModule {
  ...
  type port APortType message { ... }
  ...
  type component ATCType {
    ...
    APortType APort;
    ...
  }
  ...
  template AType AMessageTemplate { ... }
  ...
  function APTCBehaviour(...) runs on APTCType {
    ... //the PTC behaviour
  }
  ...
  testcase ATestCase(...) runs on ATCType... {
    ... //the test case behaviour
    var ATCType PTC1:= ATCType.create;
    connect (PTC1:APort,mtc:APort);
    ...
    PTC1.start(APTCBehaviour(...));
    APort.send(AMessageTemplate); //sending data to a test component
  }
  ...
}
  
```

### I.5.2 Use scenario: internode communication between test components

The scenario in Figure I.22 shows the communication between test components (main test component or parallel test components), which reside on different nodes. A communication request is given to the TCI-CH, which then decides where to enqueue this communication template. In this case, the communication is done remotely via the TE on another node. The scenario shows a message based communication using the send operation – the scenario is the same for call, reply, and raise operations.

### I.5.2.1 Sequence diagram

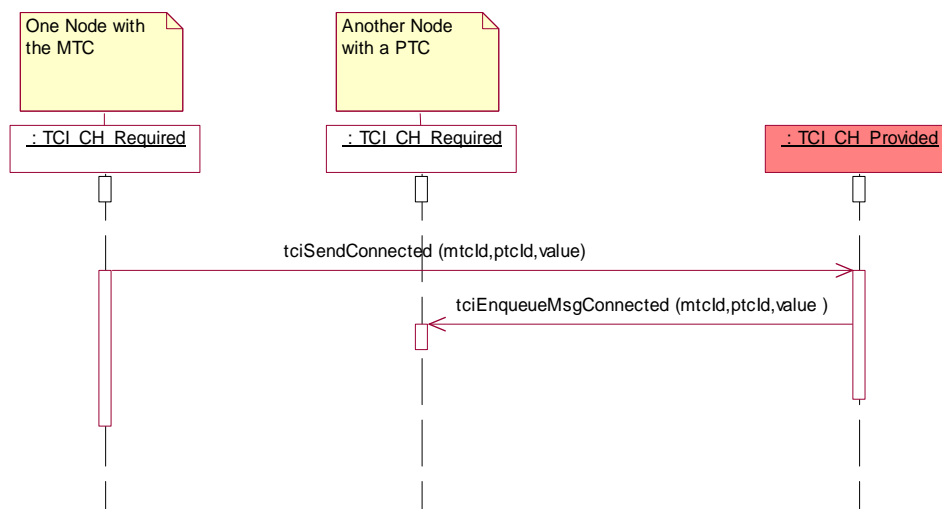


Figure I.22 – Use scenario – internode communication between test components

### I.5.2.2 TTCN-3 fragment

```

module AModule {
  ...
  type port APortType message { ... }
  ...
  type component ATCType {
    ...
    APortType APort;
    ...
  }
  ...
  template AType AMessageTemplate { ... }
  ...
  function APTCBehaviour(...) runs on APTCType {
    ... //the PTC behaviour
  }
  ...
  testcase ATestCase(...) runs on ATCType... {
    ... //the test case behaviour
    var ATCType PTC1:= ATCType.create;
    connect (PTC1:APort,mtc:APort);
    ...
    PTC1.start(APTCBehaviour(...));
    APort.send(AMessageTemplate); //sending data to a test component
  }
  ...
}
  
```

### I.5.3 Use scenario: encoding

The scenario in figure I.23 shows the encoding of data, which is sent to the SUT. The encoded data is received from the coding/decoding entity via the TCI-CD. The encoded value is sent to the SUT via the TRI-SA. The scenario is the same for the call, the reply, and the raise operations.

### I.5.3.1 Sequence diagram

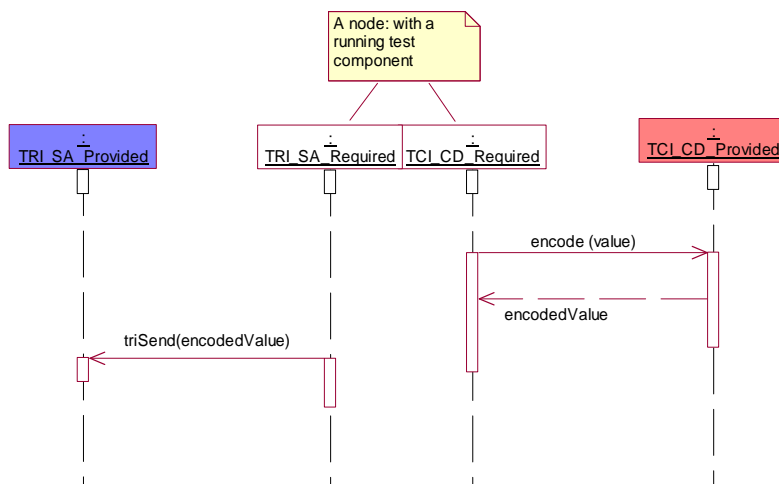


Figure I.23 – Use scenario – encoding

### I.5.3.2 TTCN-3 fragment

```

module AModule {
  ...
  type port APortType message { ... }
  ...
  type component APTCType {
    ...
    APortType APort;
    ...
  }
  ...
  template AType AMessageTemplate { ... }
  ...
  testcase ATestCase(...) runs on APTCType system APTCType {
    ... //the test case behaviour
    map(mtc:APort, system:APort);
    ...
    APort.send(AMessageTemplate); //sending data to the SUT
    ...
  }
  ...
} with { encoding "..."}

```

### I.5.4 Use scenario: decoding

The scenario in Figure I.24 shows the decoding of data, which is received from the SUT via the TRI-SA. The decoded data is received from the coding/decoding entity via the TCI-CD. The scenario is the same for the receive, the getcall, the getreply, the catch, and the check operations.

### I.5.4.1 Sequence diagram

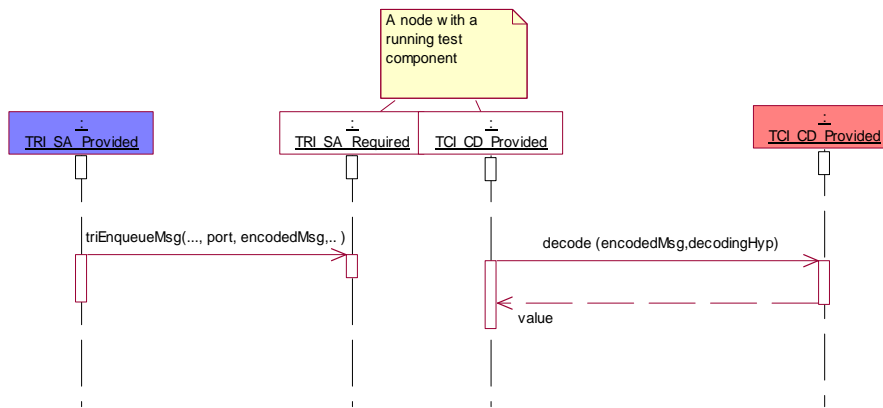


Figure I.24 – Use scenario – decoding

### I.5.4.2 TTCN-3 fragment

```

module AModule {
  ...
  type port APortType message { ... }
  ...
  type component APTCType {
    ...
    APortType APort;
    ...
  }
  ...
  template AType AMessageTemplate { ... }
  ...
  testcase ATestCase(...) runs on APTCType system APTCType {
    ... //the test case behaviour
    map(mtc:APort, system:APort);
    ...
    APort.receive(AMessageTemplate); //receiving data from the SUT
    ...
  }
  ...
} with { encoding "..."}

```



## Bibliography

- [b-ISO/IEC 9646-3] ISO/IEC 9646-3:1998, *Information technology – Open Systems Interconnection – Conformance testing methodology and framework – Part 3: The Tree and Tabular combined Notation (TTCN)*.
- [b-Brady] Brady, F., Baker, R. M. (1996), *Generic Compiler/Interpreter interface; GCI Interface Specification, Infrastructural Tools*, INTOOL CGI/NPL038 (V2.2).





## **SERIES OF ITU-T RECOMMENDATIONS**

Series A	Organization of the work of ITU-T
Series D	General tariff principles
Series E	Overall network operation, telephone service, service operation and human factors
Series F	Non-telephone telecommunication services
Series G	Transmission systems and media, digital systems and networks
Series H	Audiovisual and multimedia systems
Series I	Integrated services digital network
Series J	Cable networks and transmission of television, sound programme and other multimedia signals
Series K	Protection against interference
Series L	Construction, installation and protection of cables and other elements of outside plant
Series M	Telecommunication management, including TMN and network maintenance
Series N	Maintenance: international sound programme and television transmission circuits
Series O	Specifications of measuring equipment
Series P	Terminals and subjective and objective assessment methods
Series Q	Switching and signalling
Series R	Telegraph transmission
Series S	Telegraph services terminal equipment
Series T	Terminals for telematic services
Series U	Telegraph switching
Series V	Data communication over the telephone network
Series X	Data networks, open system communications and security
Series Y	Global information infrastructure, Internet protocol aspects and next-generation networks
<b>Series Z</b>	<b>Languages and general software aspects for telecommunication systems</b>