

**Methods for Testing and Specification (MTS);
The Testing and Test Control Notation version 3;
Part 6: TTCN-3 Control Interface (TCI);**



Reference

DES/MTS-00063-6[2]

Keywords

TTCN, testing, MTS, TCI

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - NAF 742 C
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° 7803/88

Important notice

Individual copies of the present document can be downloaded from:

<http://www.etsi.org>

The present document may be made available in more than one electronic version or in print. In any case of existing or perceived difference in contents between such versions, the reference version is the Portable Document Format (PDF). In case of dispute, the reference shall be the printing on ETSI printers of the PDF version kept on a specific network drive within ETSI Secretariat.

Users of the present document should be aware that the document may be subject to revision or change of status. Information on the current status of this and other ETSI documents is available at

<http://portal.etsi.org/tb/status/status.asp>

If you find errors in the present document, send your comment to:

editor@etsi.org

Copyright Notification

No part may be reproduced except as authorized by written permission.
The copyright and the foregoing restriction extend to reproduction in all media.

© European Telecommunications Standards Institute 2003.
All rights reserved.

DECT™, **PLUGTESTS™** and **UMTS™** are Trade Marks of ETSI registered for the benefit of its Members.
TIPHON™ and the **TIPHON logo** are Trade Marks currently being registered by ETSI for the benefit of its Members.
3GPP™ is a Trade Mark of ETSI registered for the benefit of its Members and of the 3GPP Organizational Partners.

Contents

Intellectual Property Rights	8
Foreword.....	8
1 Scope	9
2 References	9
3 Definitions and abbreviations.....	9
3.1 Definitions	9
3.2 Abbreviations	10
4 Introduction	11
5 Compliance.....	11
6 General structure of a TTCN-3 test system.....	11
6.1 Entities in a TTCN-3 test system.....	12
6.1.1 Test Management and Control (TMC).....	12
6.1.1.1 Test Management (TM)	12
6.1.1.2 Coding and Decoding (CD)	13
6.1.1.3 Component Handling (CH)	13
6.1.2 TTCN-3 Executable (TE)	14
6.1.3 SUT Adaptor (SA).....	14
6.1.4 Platform Adaptor (PA).....	14
6.2 Execution requirements for a TTCN-3 test system	14
7 TTCN-3 control interface and operations.....	14
7.1 Overview of the TCI.....	14
7.1.1 Correlation between TTCN-3 and TCI operation invocations	15
7.2 TCI data.....	16
7.2.1 General abstract data types	17
7.2.1.1 Management.....	17
7.2.1.2 Communication	18
7.2.2 Abstract TTCN-3 data types and values	18
7.2.2.1 Abstract TTCN-3 data types	19
7.2.2.2 Abstract TTCN-3 values	19
7.2.2.2.1 The abstract data type Value	21
7.2.2.2.2 The abstract data type IntegerValue	21
7.2.2.2.3 The abstract data type FloatValue	21
7.2.2.2.4 The abstract data type BooleanValue	21
7.2.2.2.5 The abstract data type ObjidValue	21
7.2.2.2.6 The abstract data type CharValue.....	22
7.2.2.2.7 The abstract data type UniversalCharValue	22
7.2.2.2.8 The abstract data type CharstringValue	22
7.2.2.2.9 The abstract data type UniversalCharstringValue	23
7.2.2.2.10 The abstract data type BitstringValue.....	23
7.2.2.2.11 The abstract data type OctetstringValue	24
7.2.2.2.12 The abstract data type HexstringValue.....	24
7.2.2.2.13 The abstract data type RecordValue.....	25
7.2.2.2.14 The abstract data type RecordOfValue	25
7.2.2.2.15 The abstract data type UnionValue	26
7.2.2.2.16 The abstract data type EnumeratedValue	27
7.2.2.2.17 The abstract data type VerdictValue	27
7.3 TCI operations.....	27
7.3.1 The TCI-TM interface	28
7.3.1.1 TCI-TM required.....	28
7.3.1.1.1 tciRootModule.....	28
7.3.1.1.2 getImportedModules.....	29

7.3.1.1.3	tciParameters29	29
7.3.1.1.4	tciParameters29	29
7.3.1.1.5	tciParameters29	29
7.3.1.1.6	tciParameters29	29
7.3.1.1.7	tciParameters30	30
7.3.1.1.8	tciParameters30	30
7.3.1.1.9	tciParameters30	30
7.3.1.1.10	tciParameters30	30
7.3.1.2	TCI-TM provided31	31
7.3.1.2.1	tciParameters31	31
7.3.1.2.2	tciParameters31	31
7.3.1.2.3	tciParameters31	31
7.3.1.2.4	tciParameters32	32
7.3.1.2.5	tciParameters32	32
7.3.1.2.6	tciParameters32	32
7.3.2	The TCI-CD interface32	32
7.3.2.1	TCI-CD required33	33
7.3.2.1.1	getInteger33	33
7.3.2.1.2	getInteger33	33
7.3.2.1.3	getInteger33	33
7.3.2.1.4	getInteger34	34
7.3.2.1.5	getInteger34	34
7.3.2.1.6	getInteger34	34
7.3.2.1.7	getInteger34	34
7.3.2.1.8	getInteger34	34
7.3.2.1.9	getInteger34	34
7.3.2.1.10	getInteger34	34
7.3.2.1.11	getInteger34	34
7.3.2.1.12	getInteger35	35
7.3.2.1.13	getInteger35	35
7.3.2.1.14	tciParameters35	35
7.3.2.2	TCI-CD provided35	35
7.3.2.2.1	decode35	35
7.3.2.2.2	encode35	35
7.3.3	The TCI-CH interface36	36
7.3.3.1	TCI-CH required36	36
7.3.3.1.1	tciParameters37	37
7.3.3.1.2	tciParameters37	37
7.3.3.1.3	tciParameters37	37
7.3.3.1.4	tciParameters38	38
7.3.3.1.5	tciParameters38	38
7.3.3.1.6	tciParameters38	38
7.3.3.1.7	tciParameters38	38
7.3.3.1.8	tciParameters39	39
7.3.3.1.9	tciParameters39	39
7.3.3.1.10	tciParameters39	39
7.3.3.1.11	tciParameters39	39
7.3.3.1.12	tciParameters39	39
7.3.3.1.13	tciParameters40	40
7.3.3.1.14	tciParameters40	40
7.3.3.1.15	tciParameters40	40
7.3.3.1.16	tciParameters40	40
7.3.3.1.17	tciParameters41	41
7.3.3.2	TCI-CH provided41	41
7.3.3.2.1	tciParameters41	41
7.3.3.2.2	tciParameters41	41
7.3.3.2.3	tciParameters42	42
7.3.3.2.4	tciParameters42	42
7.3.3.2.5	tciParameters42	42
7.3.3.2.6	tciParameters43	43
7.3.3.2.7	tciParameters43	43
7.3.3.2.8	tciParameters43	43

7.3.3.2.9	tciDisconnectReq	43
7.3.3.2.10	tciMapReq	44
7.3.3.2.11	tciUnmapReq	44
7.3.3.2.12	tciTestComponentTerminatedReq	44
7.3.3.2.13	tciTestComponentRunningReq	44
7.3.3.1.14	tciTestComponentDoneReq	44
7.3.3.2.15	tciGetMTCReq	45
7.3.3.2.16	tciExecuteTestCaseReq	45
7.3.3.2.17	tciResetReq	45
8	Java language mapping	45
8.1	Introduction	45
8.2	Names and scopes	46
8.2.1	Names	46
8.2.2	Scopes	46
8.2	Type mapping	46
8.2.1	Basic type mapping	46
8.2.1.1	boolean	46
8.2.1.2	float	46
8.2.1.3	char	46
8.2.1.4	int	46
8.2.1.5	String	46
8.2.1.6	String[]	47
8.2.1.7	Universal Char	47
8.2.2	Structured type mapping	47
8.2.2.1	TciParameterType	47
8.2.2.2	TciParameterPassingModeType	48
8.2.2.3	TciParameterListType	48
8.2.2.4	TciTypeClassType	49
8.2.2.5	TciTestComponentKindType	49
8.2.2.6	TciSignatureIdType	49
8.2.2.7	TciBehaviourIdType	49
8.2.2.8	TciTestCaseIdType	49
8.2.2.9	TciModuleIdType	50
8.2.2.10	TciModuleParameterIdType	50
8.2.2.11	TciModuleParameterListType	50
8.2.2.12	TciModuleParameterType	50
8.2.2.13	TciParameterTypeListType	51
8.2.2.14	TciModuleIdListType	51
8.2.3	Abstract type mapping	51
8.2.3.1	Type	51
8.2.4	Abstract value mapping	52
8.2.4.1	Value	52
8.2.4.2	IntegerValue	53
8.2.4.3	FloatValue	53
8.2.4.4	BooleanValue	53
8.2.4.5	ObjidValue	53
8.2.4.6	TciObjId	54
8.2.4.7	TciObjIdElement	54
8.2.4.8	CharValue	54
8.2.4.9	UniversalCharValue	55
8.2.4.10	CharstringValue	55
8.2.4.11	BitstringValue	55
8.2.4.12	OctetstringValue	56
8.2.4.13	UniversalCharstringValue	57
8.2.4.14	HexstringValue	57
8.2.4.15	RecordValue	58
8.2.4.16	RecordOfValue	59
8.2.4.17	UnionValue	60
8.2.4.18	EnumeratedValue	60
8.2.4.19	VerdictValue	61
8.3	Constants	61

8.4	Mapping of interfaces.....	62
8.4.1	The TCI-TM interface	63
8.4.1.1	TCI-TM provided.....	63
8.4.1.2	TCI-TM required.....	63
8.4.2	The TCI-CD interface	63
8.4.2.1	TCI-CD provided	63
8.4.2.2	TCI-CD required	64
8.4.3	The TCI-CH interface	64
8.4.3.1	TCI-CH provided	64
8.4.3.2	TCI-CH required	65
8.5	Optional parameters	65
8.6	TCI initialization	66
8.7	Error handling	66
9	ANSI C language mapping.....	66
9.1	Introduction	66
9.2	Value interface	66
9.3	Operation interface	70
9.4	Data	73
9.5	Miscellaneous.....	74
10	Use scenarios.....	75
10.1	Initialization, collecting information, logging	75
10.1.1	Use scenario: initialization.....	75
10.1.1.1	Sequence diagram	75
10.1.1.2	TTCN-3 fragment	75
10.1.2	Use scenario: requesting module parameters	76
10.1.2.1	Sequence diagram	76
10.1.2.2	TTCN-3 fragment	76
10.1.3	Use scenario: logging.....	76
10.1.3.1	Sequence diagram	77
10.1.3.2	TTCN-3 fragment	77
10.2	Execution of test cases and control.....	77
10.2.1	Use scenario: execution of control.....	77
10.2.1.1	Sequence diagram	78
10.2.1.2	TTCN-3 fragment	78
10.2.2	Use scenario: test case execution within control.....	78
10.2.2.1	Sequence diagram	79
10.2.2.2	TTCN-3 fragment	79
10.2.3	Use scenario: direct test case execution.....	79
10.2.3.1	Sequence diagram	80
10.2.3.2	TTCN-3 fragment	80
10.2.4	Use scenario: execute test case to TRI.....	80
10.2.4.1	Sequence diagram	80
10.2.4.2	TTCN-3 fragment	81
10.3	Component handling	81
10.3.1	Use scenario: local control component creation	81
10.3.1.1	Sequence diagram	81
10.3.1.2	TTCN-3 fragment	82
10.3.2	Use scenario: remote control component creation	82
10.3.2.1	Sequence diagram	82
10.3.2.2	TTCN-3 fragment	82
10.3.3	Use scenario: local MTC creation.....	83
10.3.3.1	Sequence diagram	83
10.3.3.2	TTCN-3 fragment	83
10.3.4	Use scenario: remote MTC creation	83
10.3.4.1	Sequence diagram	84
10.3.4.2	TTCN-3 fragment	84
10.3.5	Use scenario: component handling for test case execution within control	84
10.3.5.1	Sequence diagram	85
10.3.5.2	TTCN-3 fragment	85
10.3.6	Use scenario: component handling for direct test case execution.....	86

10.3.6.1	Sequence diagram	86
10.3.6.2	TTCN-3 fragment	87
10.3.7	Use scenario: propagation of map/connect	87
10.3.7.1	Sequence diagram	87
10.3.7.2	TTCN-3 fragment	87
10.3.8	Use scenario: propagation of unmap/disconnect.....	88
10.3.8.1	Sequence diagram	88
10.3.8.2	TTCN-3 fragment	88
10.4	Termination of test cases and control.....	88
10.4.1	Use scenario: stop a test case	88
10.4.1.1	Sequence diagram	89
10.4.1.2	TTCN-3 fragment	89
10.4.2	Use scenario: stop control.....	89
10.4.2.1	Sequence diagram	90
10.4.2.2	TTCN-3 fragment	90
10.4.3	Use scenario: termination of control after error.....	90
10.4.3.1	Sequence diagram	91
10.4.3.2	TTCN-3 fragment	91
10.4.4	Use scenario: termination of a test case after error.....	91
10.4.4.1	Sequence diagram	92
10.4.4.2	TTCN-3 fragment	92
10.4.5	Use scenario: reset	92
10.4.5.1	Sequence diagram	93
10.4.5.2	TTCN-3 fragment	93
10.5	Communication	93
10.5.1	Use scenario: local intercomponent communication	93
10.5.1.1	Sequence diagram	93
10.5.1.2	TTCN-3 fragment	94
10.5.2	Use scenario: internode communication between test components	94
10.5.2.1	Sequence diagram	94
10.5.2.2	TTCN-3 fragment	95
10.5.3	Use scenario: encoding	95
10.5.3.1	Sequence diagram	95
10.5.3.2	TTCN-3 fragment	96
10.5.4	Use scenario: decoding	96
10.5.4.1	Sequence diagram	96
10.5.4.2	TTCN-3 fragment	97
Annex A (normative):	IDL Specification of TCI.....	98
A.1	TCI IDL	98
Annex B (informative):	Bibliography.....	105
History		106

Intellectual Property Rights

IPRs essential or potentially essential to the present document may have been declared to ETSI. The information pertaining to these essential IPRs, if any, is publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (<http://webapp.etsi.org/IPR/home.asp>).

All published ETSI deliverables shall include information which directs the reader to the above source of information.

Foreword

This ETSI Standard (ES) has been produced by ETSI Technical Committee Methods for Testing and Specification (MTS), and is now submitted for the ETSI standards Membership Approval Procedure.

The present document is part 6 of a multi-part deliverable covering the Testing and Test Control Notation version 3, as identified below:

- Part 1: "TTCN-3 Core Language";
- Part 2: "TTCN-3 Tabular presentation Format (TFT)";
- Part 3: "TTCN-3 Graphical presentation Format (GFT)";
- Part 4: "TTCN-3 Operational Semantics".
- Part 5: "TTCN-3 Runtime Interface (TRI)";
- Part 6: "TTCN-3 Control Interfaces (TCI)".**

1 Scope

The present document 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 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 references below). 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.

2 References

The following documents contain provisions which, through reference in this text, constitute provisions of the present document.

- References are either specific (identified by date of publication and/or edition number or version number) or non-specific.
- For a specific reference, subsequent revisions do not apply.
- For a non-specific reference, the latest version applies.

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

- | | |
|-----|--|
| [1] | OMG CORBA v2.2: "The Common Object Request Broker: Architecture and Specification, Section 3, February 1998". |
| [2] | ETSI ES 201 873-5: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 5: TTCN-3 Runtime Interface (TRI)". |
| [3] | ETSI ES 201 873-1: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language". |
| [4] | ETSI ES 201 873-4: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 4: TTCN-3 Operational Semantics". |
| [5] | ISO/IEC 9646-1: " Information technology - Open Systems Interconnection - Conformance testing methodology and framework - Part 1: General concepts". |

3 Definitions and abbreviations

3.1 Definitions

For the purpose of the present document, the terms and definitions given in ISO/IEC 9646-1 [5] and the following apply:

Abstract Test Suite (ATS): test suite composed of abstract test cases

codec: encoder/decoder entity used for encoding and decoding data to be transmitted and received, respectively

Coding/Decoding (CD): entity that administers the value and type handling incl. encoding and decoding in the TTCN-3 test system

Component Handling (CH): entity that administers the handling of test components in the TTCN-3 test system

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

control component: component that executes the behaviour of the control part of a TTCN-3 module

Executable Test Suite (ETS): refer to ISO/IEC 9646-1 [5]

Implementation eXtra Information for Testing (IXIT): refer to ISO/IEC 9646-1 [5]

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.

real test system interface: refer to ISO/IEC 9646-1 [5]

System Under Test (SUT): refer to ISO/IEC 9646-1 [5]

SUT Adaptor (SA): an entity that adapts the TTCN-3 communication operations with the SUT based on an abstract test system interface. It implements the real test system interface

Testing and Test Control Notation (TTCN-3): refer to ISO/IEC 9646-1 [5]

test case: refer to ISO/IEC 9646-1 [5]

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

Test Management (TM): an entity which provides a user interface to as well as the administration of the TTCN-3 test system

Test Management and Control (TMC): a set of three entities providing test management and control; consists of the test management (TM), the component handling (CH) and the coding/decoding (CD)

NOTE: The TMC is an implementation of TCL.

test system: refer to ISO/IEC 9646-1 [5]

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

TTCN-3 Executable (TE): the part of a test system that deals with interpretation or execution of a TTCN-3 ETS

TTCN-3 Control Interfaces (TCI): three interfaces that define the interaction of the TTCN-3 Executable with the test management, the coding and decoding, and the test component handling in a test system

TTCN-3 Runtime Interface (TRI): an interface that defines the interaction of the TTCN-3 Executable with the SUT and platform adaptors in a test system

3.2 Abbreviations

For the purposes of the present document, the following abbreviations apply:

ATS	Abstract Test Suite
CD	Coding/Decoding
CH	Component Handler
ETS	Executable Test Suite
IDL	Interface Definition Language
IXIT	Implementation Extra Information for Testing
MSC	Message Sequence Chart
MTC	Main Test Component
OMG	Object Management Group
PA	Platform Adaptor

PTC	Parallel Test Component
SA	SUT Adaptor
SUT	System Under Test
TC	Test Control
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

4 Introduction

The present document consists of two distinct parts, the first part describing the structure of a TTCN-3 test system implementation and the second part presenting 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), and 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 this 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 ([2]).

5 Compliance

The minimum required for a TCI compliant TTCN-3 test system is to adhere to the interface specification stated in this document. The TTCN-3 semantics in the test system must adhere to the operational semantics defined in [4]. 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.

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.

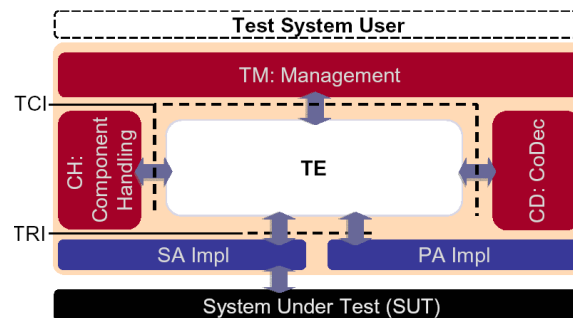


Figure 1: General structure of a TTCN-3 test system

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 [3] 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 [2]. The TCI specification defines the interaction between the TE and the TMC.

6.1.1 Test Management and Control (TMC)

The TMC entity includes functionality related to management of:

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

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. In addition, the TM entity performs test event logging and presentation to the test system user.

6.1.1.2 Coding and Decoding (CD)

The CD entity is responsible for the encoding and decoding of TTCN-3 values into bitstrings suitable to be sent to the System Under Tests. 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 2.

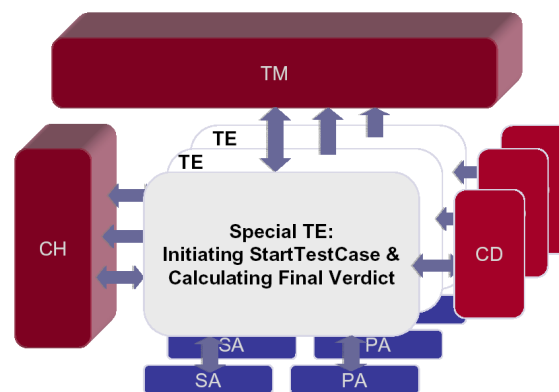


Figure 2: General structure of a distributed TTCN-3 test system

Each node within a test system includes the TE, SA, PA, and CD 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: A test system shall execute at most one test case at a given point in time. That is, 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.

6.1.2 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 [2]. The terminology for TE defined in [2] 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 [2]. The terminology for SA defined in [2] 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 [2]. The terminology for PA defined in [2] 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), and the Coding/Decoding (CD) 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; and
- encode and decode test data.

The TCI consists of three 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.

All interfaces are bi-directional 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, and TCI-CD Provided/ TCI-CD 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.

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 [2].

Table 1: Correlation between TTCN-3 and TCI operation invocations

TTCN-3 Operation Name	TCI Operation Name	TCI Interface Name
send	tcISendConnected	TCI-CH Provided
	tcIEnqueueMsgConnected	TCI-CH Required
call	tcICallConnected	TCI-CH Provided
	tcIEnqueueCallConnected	TCI-CH Required
reply	tcIReplyConnected	TCI-CH Provided
	tcIEnqueueReplyConnected	TCI-CH Required
raise	tcIRaiseConnected	TCI-CH Provided
	tcIEnqueueRaiseConnected	TCI-CH Required
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
connect	tcIConnectReq	TCI-CH Provided
	tcIConnect	TCI-CH Required
disconnect	tcIDisconnectReq	TCI-CH Provided
	tcIDisconnect	TCI-CH Required
map	tcIMapReq	TCI-CH Provided
	tcIMap	TCI-CH Required
unmap	tcIUnmapReq	TCI-CH Provided
	tcIUnmap	TCI-CH Required
running	tcITestComponentRunningReq	TCI-CH Provided
	tcITestComponentRunning	TCI-CH Required
done	tcITestComponentDoneReq	TCI-CH Provided
	tcITestComponentDone	TCI-CH Required
mtc	tcIGetMTCReq	TCI-CH Provided
	tcIGetMTC	TCI-CH Required
execute	tcITestCaseExecuteReq	TCI-CH Provided
	tcITestCaseExecute	TCI-CH Required

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 and 9.

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

TciModuleIdType	A value of TciModuleIdType is the name of a TTCN-3 module as specified in the TTCN-3 ATS. This abstract type is used for module handling
TciModuleParameterIdType	A value of TciModuleIdType 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
TciTestCaseIdType	A value of TciTestCaseIdType is the qualified name of a TTCN-3 testcase as specified in the TTCN-3 ATS. This abstract type is used for testcase handling
TciModuleIdListType	A value of type TciModuleIdListType is a list of TciModuleIdType . This abstract type is used when retrieving the list of modules which are imported by a TTCN-3 module
TciModuleParameterType	A value of type TciModuleParameterType is a structure of TciModuleParameterIdType and Value . This abstract type is used to represent the parameter name and the default value of a module parameter
TciModuleParameterListType	A value of type TciModuleParameterListType is a list of TciModuleParameterType . This abstract type is used when retrieving the module parameters of a TTCN-3 module
TciParameterType	A value of type TciParameterType includes a TTCN-3 Value and a value of TciParameterPassingModeType to represent the parameter passing mode specified for the parameter in the TTCN-3 ATS
TciParameterPassingModeType	A value of type TciParameterPassingModeType is either IN , INOUT , or OUT . This abstract type is used when starting a test case or when the termination of a test case is indicated
TciParameterListType	A value of type TciParameterListType is a list of TciParameterType . This abstract type is used when starting a test case or when the termination of a test case is indicated
TciParameterTypeType	A value of type TciParameterTypeType is a structure of Type and TciParameterPassingModeType . This abstract type is used to represent the type and the parameter passing mode of a test case parameter
TciParameterTypeListType	A value of type TciParameterTypeListType is a list of TciParameterTypeType . This abstract type is used to represent the list of parameters of a test case
TciTestComponentKindType	A value of type TciParameterListType is a literal of the set of kinds of TTCN-3 test components, e.g. MTC , PTC , and CONTROL . This abstract type is used for component handling
TciTypeClassType	A value of type TciTypeClassType is a literal of the set of type classes in TTCN-3 such as boolean , float , record , etc. This abstract type is used for value handling

7.2.1.2 Communication

TciSignatureIdType A value of type **TciSignatureIdType** is the name of a procedure signature as specified in the TTCN-3 ATS. This abstract type is used in procedure based TCI communication operations

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

Additional abstract data types with the prefix **Tri** are taken from [2]: **TriPortIdType**, **TriPortIdListType**, **TriComponentIdType**, 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 mapping for the operations on the abstract data types are given in clause 8 and 9. 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 to
- create and maintain TTCN-3 values.

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

<code>TciModuleIdType</code> <code>getDefiningModule()</code>	Returns the module identifier of the module in which type is defined. Returns the distinct value <code>null</code> if type is a TTCN-3 base type, e.g. boolean, integer, etc.)
<code>TString</code> <code>getName()</code>	Returns the name of the type as defined in the TTCN-3 module
<code>TciTypeClassType</code> <code>getTypeClass()</code>	Returns the type class of the respective type. A value of <code>TciTypeClassType</code> can have one of the following constants: ADDRESS, ANYTYPE, BITSTRING, BOOLEAN, CHAR, CHARSTRING, COMPONENT, ENUMERATED, FLOAT, HEXSTRING, INTEGER, OBJID, OCTETSTRING, RECORD, RECORD_OF, SET, SET_OF, UNION, UNIVERSAL_CHARSTRING, UNIVERSAL_CHAR, VERDICT
<code>Value</code> <code>newInstance()</code>	Returns a freshly created value of the given type. This initial value of the created value is undefined
<code>TString</code> <code>getTypeEncoding()</code>	Returns the type encoding attribute as defined in the TTCN-3 module
<code>TString</code> <code>getTypeEncodingVariant()</code>	This operation returns the value encoding variant attribute as defined in TTCN-3, if any. If no encoding variant attribute is defined the distinct value <code>null</code> is returned

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

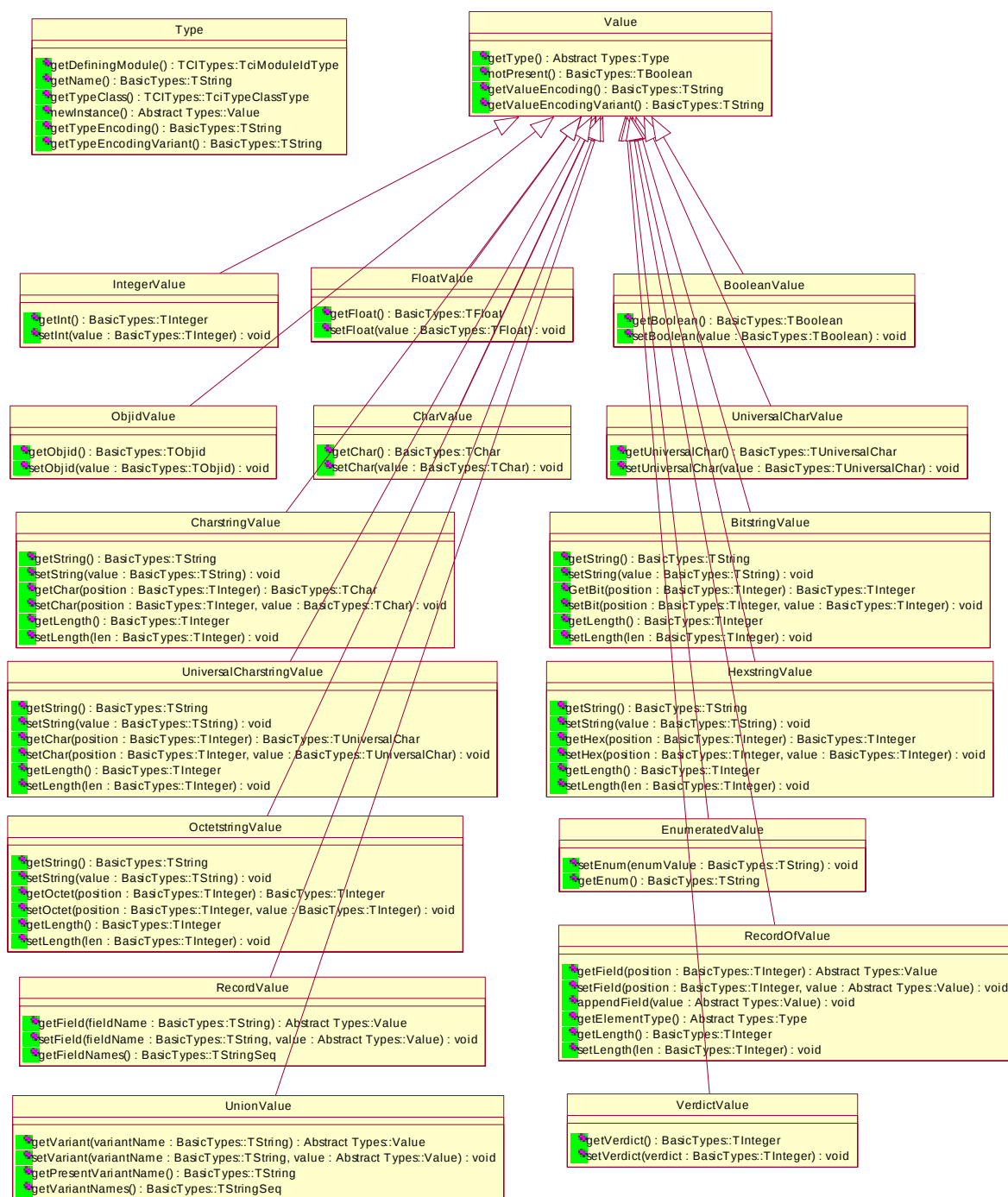


Figure 3: Hierarchy of abstract values

As shown in figure 3, 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 TTCN-3, 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 TTCN-3, 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
<code>void</code> <code>setInt(in TInteger value)</code>	Sets this <code>IntegerValue</code> 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`:

<code>TFloat</code> <code>getFloat()</code>	Returns the float value of this TTCN-3 float
<code>void</code> <code>setFloat(in TFloat value)</code>	Sets this <code>FloatValue</code> 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 `FloatValue`:

<code>TBoolean</code> <code>getBoolean ()</code>	Returns the boolean value of the TTCN-3 boolean
<code>void</code> <code>setBoolean(in TBoolean value)</code>	Sets this boolean value to value

7.2.2.2.5 The abstract data type ObjidValue

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

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

<code>TObjid</code> <code>getObjid()</code>	Returns the object id value of the TTCN-3 <code>objid</code>
<code>void</code> <code>setObjid(in TObjid value)</code>	Sets this <code>ObjidValue</code> to value

7.2.2.2.6 The abstract data type CharValue

The abstract data type CharValue is based on the abstract data type Value. It represents TTCN-3 char values.

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

TChar getChar() Returns the char value of the TTCN-3 char

void setChar(in TChar value) Sets this CharValue to value

7.2.2.2.7 The abstract data type UniversalCharValue

The abstract data type UniversalCharValue is based on the abstract data type Value. It represents TTCN-3 universal char values.

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

TUniversalChar getUniversalChar() Returns the universal char value of the TTCN-3 universal char

void setUniversalChar(in TUniversalChar value)
Sets this universal char value to value

7.2.2.2.8 The abstract data type CharstringValue

The abstract data type CharstringValue is based on the abstract data type Value. It represents TTCN-3 charstring values.

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

7.2.2.2.9 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.

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

<code>TString getString()</code>	Returns the textual representation of this <code>UniversalCharstringValue</code> , 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 position. Position 0 denotes the first <code>TUniversalChar</code> of the TTCN-3 universal charstring. Valid values for <code>position</code> are from 0 to <code>length - 1</code>
<code>void setChar(in TInteger position, in TUniversalChar value)</code>	Sets the universal char at position to <code>value</code> . Valid values for <code>position</code> are from 0 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>

7.2.2.2.10 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 0101 is "0101"B. The textual representation of the empty TTCN-3 bitstring is ""B, 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 0101 if the textual representation in <code>value</code> is "0101"B
<code>TChar getBit(in TInteger position)</code>	Returns the value (0 1) at position of this TTCN-3 bitstring as a character. Position 0 denotes the first bit of the TTCN-3 bitstring. Valid values for <code>position</code> are from 0 to <code>length - 1</code>
<code>void setBit(in TInteger position, in TInteger value)</code>	Sets the bit at position to the value (0 1). Position 0 denotes the first bit in this <code>BitstringValue</code> . Valid values for <code>position</code> are from 0 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>

7.2.2.2.11 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"0</code> . The textual representation of the empty TTCN-3 <code>octetstring</code> is <code>""0</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"0</code>
<code>TChar getOctet(in TInteger position)</code>	Returns the value (<code>0..255</code>) at <code>position</code> of this TTCN-3 <code>octetstring</code> . Position 0 denotes the first octet of the TTCN-3 <code>octetstring</code> . Valid values for <code>position</code> 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 value <code>0..255</code>). Position 0 denotes the first octet in the <code>octetstring</code> . Valid values for <code>position</code> 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 omit
<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>

7.2.2.2.12 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 hexstring. Position 0 denotes the first hex digits of the TTCN-3 hexstring. Valid values for position are from 0 to length - 1
<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
<code>TInteger getLength()</code>	Returns the length of this HexstringValue in octets, zero if the value of this HexstringValue is omit
<code>void setLength(in TInteger len)</code>	setLength first resets this HexstringValue to its initial value and afterwards sets the length of this HexstringValue in hex digits to len

7.2.2.2.13 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. 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`:

<code>Value getField(in TString fieldName)</code>	Returns the value of the field named <code>fieldName</code> . The return value is the common abstract base type <code>Value</code> , as a record field can have any type defined in TTCN-3. If the field can not be obtained from the record the distinct value <code>null</code> is returned
<code>void setField(in TString fieldName, in Value value)</code>	Sets the field named <code>fieldName</code> of the record to <code>value</code> . 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 relied on that subsequent modifications of <code>value</code> will not be considered in the record
<code>TStringSeq getFieldNames()</code>	Returns a sequence of string of field names, the distinct value <code>null</code> , if the record has no fields

7.2.2.2.14 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 `SET_OF`. The distinction between `record of` and `set of` is only relevant at matching time.

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

<code>Value getField(in TInteger position)</code>	Returns the value of the record of at position if position is between zero and length -1, the distinct value <code>null</code> otherwise. The return value is the common abstract base type <code>Value</code> , as a <code>record of</code> can have fields of any type defined in TTCN-3
---	--

<code>void setField(in TInteger position, in Value value)</code>	Sets the field at <code>position</code> to <code>value</code> . If <code>position</code> is greater than <code>(length - 1)</code> the record of is extended to have the length <code>(position + 1)</code> . The record of elements between the original position at <code>length</code> and <code>position - 1</code> is set to <code>omit</code> . 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 relied on that subsequent modifications of <code>value</code> will not be considered in the record of
<code>void appendField(in Value value)</code>	Appends the value at the end of the record of, i.e. at position <code>length</code> . 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 relied on that subsequent modifications of <code>value</code> will not be considered in the record of
<code>Type getElementType()</code>	Returns the <code>Type</code> of the elements of this record of.
<code>TInteger getLength()</code>	Returns the actual length of the record of value, zero if the record of value is <code>omit</code>
<code>void setLength(in TInteger len)</code>	Sets the length of the record of to <code>len</code> . If <code>len</code> is greater than the original length, newly created elements have the value <code>omit</code> . If <code>len</code> is less or equal than the original length this operation is ignored

7.2.2.2.15 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`:

<code>Value getVariant(in TString variantName)</code>	Returns the value of the TTCN-3 union <code>variantName</code> , if <code>variantName</code> equals the result of <code>getPresentVariantName</code> , the distinct value <code>null</code> otherwise. <code>variantName</code> denotes the name of the union variant as defined in TTCN-3
<code>void setVariant(in TString variantName, in Value value)</code>	Sets <code>variantName</code> of the union to <code>value</code> . If <code>variantName</code> is not defined for this union this operation is ignored. If another variant was selected the new variant is selected instead
<code>TString getPresentVariantName()</code>	Returns a <code>String</code> representing the currently selected variant name in the given TTCN-3 union. The distinct value <code>null</code> is returned if no variant is selected
<code>TStringSeq getVariantNames()</code>	Returns a sequence of string of variant names, the distinct value <code>null</code> , if the union has no fields. If the <code>UnionValue</code> represents the TTCN-3 anytype, i.e. the type class of the type obtained by <code>getType()</code> is <code>ANYTYPE</code> , all predefined and user-defined TTCN-3 types is returned

7.2.2.2.16 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`:

<code>TString getEnum()</code>	Returns the string identifier of this <code>EnumeratedValue</code> . This identifier equals the identifier in the TTCN-3 specification
<code>void setEnum(in TString enumValue)</code>	Sets the enum to <code>enumValue</code> . If <code>enumValue</code> is not an allowed value for this enumeration the operation is ignored

7.2.2.2.17 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`:

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

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 clause 8 and 9.

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:

Signature	IDL Signature
In Parameters	Description of data passed as parameters to the operation from the calling entity to the called entity.
Out Parameters	Description of data passed as parameters to the operation from the called entity to the calling entity.
InOut Parameters	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.
Return Value	Description of data returned from the operation to the calling entity.
Constraint	Description of any constraints when the operation can be called.
Effect	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 4).

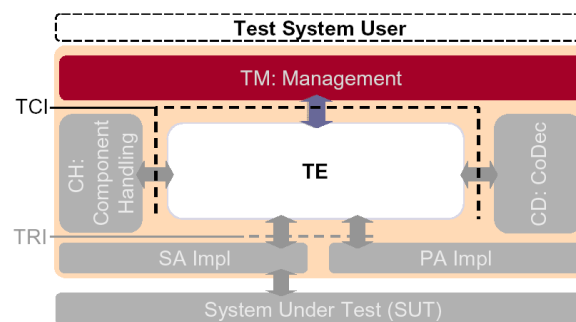


Figure 4: The TCI-TM interface

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, logging and the indication of execution termination.

Clause 10 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`

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

7.3.1.1.2 getImportedModules

Signature	TciModuleIdListType getImportedModules()	
Return Value	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.	
Constraint	Shall be used only if a root module has been set before.	
Effect	The TE provides to the management a list of imported modules of the root module. If no imported module exist, 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

Signature	TciModuleParameterListType tciGetModuleParameters (TciModuleIdType moduleName)	
In Parameters	moduleName	The moduleName denotes the module identifiers for which the module parameters should be retrieved.
Return Value	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.	
Constraint	Shall be used only if a root module has been set before.	
Effect	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

Signature	TciTestCaseIdListType tciGetTestCases ()	
Return Value	A list of all test cases that are either defined in or imported into the root module.	
Constraint	Shall be used only if a root module has been set before.	
Effect	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

Signature	TciParameterTypeListType tciGetTestCaseParameters (in TciTestCaseIdType testCaseId)	
In Parameters	testCaseId	A test case identifier as defined in the TTCN-3 module.
Return Value	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.	
Constraint	Shall be used only if a root module has been set before.	
Effect	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

Signature	TriPortIdListType tciGetTestCaseTSI (in TciTestCaseIdType testCaseId)	
In Parameters	testCaseId	A test case identifier as defined in the TTCN-3 module.
Return Value	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.	
Constraint	Shall be used only if a root module has been set before.	
Effect	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

Signature	void tciStartTestCase(in TciTestCaseIdType testCaseId, in TciParameterListType parameterList)	
In Parameters	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.
Return Value	void	
Constraint	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 can not be started. To start imported test cases the referenced (imported) module must be selected first using the tciRootModule operation.	
Effect	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 <i>in</i> and <i>inout</i> test case parameters in parameterList contain Value. All <i>out</i> 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

Signature	void tciStopTestCase()	
Return Value	void	
Constraint	Shall be called only if a module has been selected before.	
Effect	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 <i>provided</i> 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

Signature	TriComponentId tciStartControl()	
Return Value	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.	
Constraint	Shall be called only if a module has been selected before.	
Effect	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

Signature	void tciStopControl()	
Return Value	void	
Constraint	Shall only be called if a module has been selected before.	
Effect	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

Signature	void tciTestCaseStarted(in TciTestCaseIdType testCaseId, in TciParameterListType parameterList, in TFloat timer)	
In Parameters	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.
Return Value	void	
Constraint	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.	
Effect	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

Signature	void tciTestCaseTerminated (in VerdictValue verdict, in TciParameterListType parameterList)	
In Parameters	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.
Return Value	void	
Constraint	Shall only be called after either the control part of the module or a test cases has been started using the <i>required</i> operations tciStartControl or tciStartTestCase.	
Effect	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

Signature	void tciControlTerminated ()
Return Value	void
Constraint	Shall only be called when the module execution has been started using the tciStartControl operation.
Effect	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

Signature	Value tciGetModulePar (in TciModuleParameterIdType parameterId)	
In Parameters	parameterId	The identifier of the module parameter as defined in TTCN-3.
Return Value	A value.	
Constraint	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.	
Effect	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

Signature	void tciLog (in TriComponentIdType testComponentId, in TString message)	
In Parameters	testComponentId	Identifier of the component that logs the message.
	message	A string value, i.e. the message to be logged.
Return Value	void	
Constraint	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.	
Effect	The TM presents testComponentId and message to the user, how this done is not within the scope of this document.	

7.3.1.2.6 tciError

Signature	void tciError(in TString message)	
In Parameters	message	A string value, i.e. the message error
Return Value	void	
Constraint	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.	
Effect	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 (figure 5).

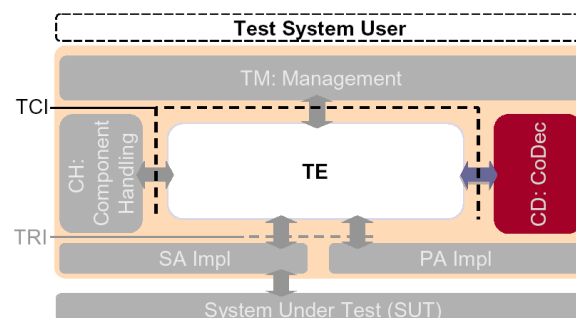


Figure 5: The TCI-CD interface

A codec implementation encodes TTCN-3 values according to the encoding attribute into a bitstring and decodes a bitstring according to decoding hypothesis. To be able to decode a bitstring into a TTCN-3 value the CD requires certain functionality from the TE. In turn the CD provides encoding and decoding functionality to the TTCN-3 Executable.

Clause 10 illustrates the usage and sequential ordering of operation calls by either the TE or the CD.

7.3.2.1 TCI-CD required

This clause specifies the operations the CD requires from the TE. All operations specified in this clause are also required at the TCI-TM and TCI-CH interfaces.

7.3.2.1.1 getTypeForName

Signature	Type getTypeForName(in TString typeName)	
In Parameters	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" "char" "universal char" "boolean" "verdicttype" "objid" typeName has to be the fully qualified type name, i.e. module.typeName
Return Value	A type representing the requested TTCN-3 type.	
Constraint	---	
Effect	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 can not be returned. Note that the anytype and address can not 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 can not be redefined.	

7.3.2.1.2 getInteger

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

7.3.2.1.3 getFloat

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

7.3.2.1.4 getBoolean

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

7.3.2.1.5 getChar

Signature	Type getChar()
Return Value	An instance of Type representing a TTCN-3 char type.
Effect	Constructs and returns a basic TTCN-3 char type.

7.3.2.1.6 getUniversalChar

Signature	Type getUniversalChar()
Return Value	An instance of Type representing a TTCN-3 universal char type.
Effect	Constructs and returns a basic TTCN-3 universal char type.

7.3.2.1.7 getObjid

Signature	Type getObjid()
Return Value	An instance of Type representing a TTCN-3 object id type.
Effect	Constructs and returns a basic TTCN-3 object id type.

7.3.2.1.8 getCharstring

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

7.3.2.1.9 getUniversalCharstring

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

7.3.2.1.10 getHexstring

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

7.3.2.1.11 getBitstring

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

7.3.2.1.12 getOctetstring

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

7.3.2.1.13 getVerdict

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

7.3.2.1.14 tciErrorReq

Signature	void tciErrorReq(in TString message)	
In Parameters	Message	A string value, i.e. the error phrase describing the problem.
Return Value	void	
Constraint	Shall be called whenever an error situation has occurred.	
Effect	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

Signature	Value decode(in TriMessageType message, in Type decodingHypothesis)	
In Parameters	message	The encoded message to be decoded.
	decodingHypothesis	The hypothesis the decoding can be based on.
Return Value	Returns the decoded value, if the value is of a compatible type as the decodingHypothesis, else the distinct value null.	
Constraint	This operation shall be called whenever the TE has to decode an encoded value. The TE might decode immediately after reception of a encoded value, or might for performance considerations postpone the decoding until the actual access of the encoded value.	
Effect	This operations 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

Signature	TriMessageType encode(in Value value)	
In Parameters	value	The value to be encoded.
Return Value	Returns an encoded TriMessage for the specified encoding rule.	
Constraint	This operation shall be called whenever the TE has to encode a Value.	
Effect	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 6).

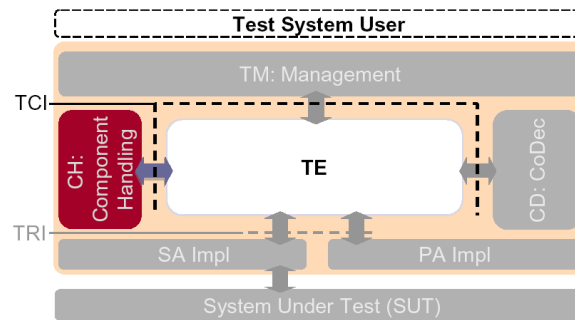


Figure 6: The TCI-CH interface

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.

Clause 10 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* operation of the TCI-CD interface are also required.

7.3.3.1.1 tciEnqueueMsgConnected

Signature	void tciEnqueueMsgConnected (in TriPortIdType sender, in TriComponentIdType receiver, in Value rcvdMessage)	
In Parameters	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.
Return Value	void	
Constraint	This operation shall be called by the CH at the local TE when at remote TE a <i>provided</i> tciSendConnected has been called.	
Effect	The TE enqueues the received value into the local port queue of the indicated receiver component.	

7.3.3.1.2 tciEnqueueCallConnected

Signature	void tciEnqueueCallConnected (in TriPortIdType sender, in TriComponentIdType receiver, in TriSignatureIdType signature, in TciParameterListType parameterList)	
In Parameters	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.
Return Value	void	
Constraint	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.	
Effect	The TE enqueues the calls at the local port queue of the indicated receiver component.	

7.3.3.1.3 tciEnqueueReplyConnected

Signature	void tciEnqueueReplyConnected (in TriPortIdType sender, in TriComponentIdType receiver, in TriSignatureIdType signature, in TciParameterListType parameterList, in Value returnValue)	
In Parameters	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.
Return Value	void	
Constraint	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.	
Effect	The TE enqueues the reply at the local port queue of the indicated receiver component.	

7.3.3.1.4 tciEnqueueRaiseConnected

Signature	void tciEnqueueRaiseConnected (in TriPortIdType sender, in TriComponentIdType receiver, in TriSignatureIdType signature, in Value exception)	
In Parameters	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.
Return Value	void	
Constraint	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.	
Effect	The TE enqueues the exception at the local port queue of the indicated receiver component.	

7.3.3.1.5 tciCreateTestComponent

Signature	TriComponentIdType tciCreateTestComponent (in TciTestComponentKindType kind, in Type componentType)	
In Parameters	kind	The kind of component that shall be created, either MTC, PTC or CONTROL.
	componentType	Identifier of the TTCN-3 component type that shall be created.
Return Value	A TriComponentIdType value for the created component.	
Constraint	This operation shall be called by the CH at the local TE when at a remote TE a <i>provided</i> tciCreateTestComponentReq has been called. componentType shall be set to the distinct value null if a test component of kind control shall be created.	
Effect	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

Signature	void tciStartTestComponent(in TriComponentIdType component, in TciBehaviourIdType behaviour, in TciParameterListType parameterList)	
In Parameters	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.
Return Value	void	
Constraint	This operation shall be called by the CH at the local TE when at a remote TE a <i>provided</i> tciStartTestComponentReq has been called. Since only <i>in</i> parameters are allowed for functions being started [3], parameterList contains only <i>in</i> parameters.	
Effect	The TE shall start the indicated behaviour on the indicated component.	

7.3.3.1.7 tciStopTestComponent

Signature	void tciStopTestComponent(in TriComponentIdType component)	
In Parameters	component	Identifier of the component to be stopped.
Return Value	void	
Constraint	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.	
Effect	The TE shall stop the indicated behaviour on the indicated component.	

7.3.3.1.8 tciConnect

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

7.3.3.1.9 tciDisconnect

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

7.3.3.1.10 tciMap

Signature	void tciMap (in TriPortIdType fromPort, in TriPortIdType toPort)	
In Parameters	fromPort	Identifier of the test component port to be mapped from.
	toPort	Identifier of the test component port to be mapped to.
Return Value	void	
Constraint	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.	
Effect	The TE shall map the indicated ports to one another.	

7.3.3.1.11 tciUnmap

Signature	void tciUnmap (in TriPortIdType fromPort, in TriPortIdType toPort)	
In Parameters	fromPort	Identifier of the test component port to be unmapped.
	toPort	Identifier of the test component port to be unmapped.
Return Value	void	
Constraint	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.	
Effect	The TE shall unmap the indicated ports.	

7.3.3.1.12 tciTestComponentTerminated

Signature	void tciTestComponentTerminated (in TriComponentIdType component, in VerdictValue verdict)	
In Parameters	component	Identifier of the component that has terminated.
	verdict	Verdict after termination of the component.
Return Value	void	
Constraint	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.	
Effect	The local TE is notified of the termination of the indicated test component on a remote TE. Since a function being executed on a test component can only have <i>in</i> parameters [3], the tciTestComponentTerminated operation does not have a parameterList parameter.	

7.3.3.1.13 tciTestComponentRunning

Signature	TBoolean tciTestComponentRunning (in TriComponentIdType component)	
In Parameters	component	Identifier of the component to be checked for running.
Return Value	true if the indicated component is still executing a behaviour, false otherwise.	
Constraint	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.	
Effect	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.	

7.3.3.1.14 tciTestComponentDone

Signature	TBoolean tciTestComponentDone (in TriComponentIdType component)	
In Parameters	component	Identifier of the component to be checked for done.
Return Value	true if the indicated component has completed executing its behaviour, false otherwise	
Constraint	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.	
Effect	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.	

7.3.3.1.15 tciGetMTC

Signature	TriComponentIdType tciGetMTC()	
Return Value	A TriComponentIdType value of the MTC if the MTC executes on the local TE, the distinct value null otherwise.	
Constraint	This operation can be called by the CH at the appropriate local TE when at a remote TE a <i>provided</i> tciGetMTCReq has been called.	
Effect	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 null 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.	

7.3.3.1.16 tciExecuteTestCase

Signature	void tciExecuteTestCase (in TciTestCaseIdType testCaseId, in TriPortIdListType tsiPortList)	
In Parameters	testCaseId	A test case identifier as defined in the TTCN-3 module.
	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.
Return Value	void	
Constraint	This operation shall be called by the CH at the appropriate local TE when at a remote TE a <i>provided</i> tciExecuteTestCaseReq has been called.	
Effect	The local TE determines whether static connections to the SUT and the initialization of communication means for TSI ports should be done.	

7.3.3.1.17 tciReset

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

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

Signature	void tciSendConnected (in TriPortIdType sender, in TriComponentIdType receiver, in Value sendMessage)	
In Parameters	sender	Port identifier at the sending component via which the message is sent.
	receiver	Identifier of the receiving component.
	sendMessage	The message to be send.
Return Value	void	
Constraint	This operation shall be called by the TE when it executes a TTCN-3 send operation on a component port, which has been connected to another component port.	
Effect	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 tciCallConnected

Signature	void tciCallConnected (in TriPortIdType sender, in TriComponentIdType receiver, in TriSignatureIdType signature, in TciParameterListType parameterList)	
In Parameters	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.
Return Value	void	
Constraint	This operation shall be called by the TE when it executes a TTCN-3 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 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.	
Effect	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 the remote TE on which receiver is being executed and enqueues the call in the remote TE.	

7.3.3.2.3 tciReplyConnected

Signature	void tciReplyConnected (in TriPortIdType sender, in TriComponentIdType receiver, in TriSignatureIdType signature, in TciParameterListType parameterList, in Value returnValue)	
In Parameters	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.
Return Value	void	
Constraint	This operation shall be called by the TE when it executes a TTCN-3 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 null shall be passed for the return value.	
Effect	On invocation of this operation the CH can issue the reply to a procedure call corresponding to the signature identifier signature and component identifier receiver. CH transmits the reply to the remote TE on which receiver is being executed and enqueues the reply in the remote TE.	

7.3.3.2.4 tciRaiseConnected

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

7.3.3.2.5 tciCreateTestComponentReq

Signature	TriComponentIdType tciCreateTestComponentReq (in TciTestComponentKindType kind, in Type componentType)	
In Parameters	kind	The kind of component that shall be created, either MTC, PTC or CONTROL.
	componentType	Identifier of the TTCN-3 component type that shall be created.
Return Value	A TriComponentIdType value for the created component.	
Constraint	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.	
Effect	CH transmits the component creation request to the remote TE and calls there the tciCreateTestComponent operation to obtain a component identifier for this component.	

7.3.3.2.6 tciStartTestComponentReq

Signature	void tciStartTestComponentReq(in TriComponentIdType component, in TciBehaviourIdType behaviour, in TciParamaterListType parameterList)	
In Parameters	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.
Return Value	void	
Constraint	This operation shall be called by the TE when it executes the TTCN-3 start operation. Since only <i>in</i> parameters are allowed for functions being started [3], parameterList contains only <i>in</i> parameters.	
Effect	CH transmits the start component request to the remote TE and calls there the tciStartTestComponent operation.	

7.3.3.2.7 tciStopTestComponentReq

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

7.3.3.2.8 tciConnectReq

Signature	void tciConnectReq (in TriPortIdType fromPort, in TriPortIdType toPort)	
In Parameters	fromPort	Identifier of the test component port to be connected from.
	toPort	Identifier of the test component port to be connected to.
Return Value	void	
Constraint	This operation shall be called by the TE when it executes a TTCN-3 connect operation.	
Effect	CH transmits the connection request to the remote TE where it calls the tciConnect 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 tciConnect operation on both remote TEs.	

7.3.3.2.9 tciDisconnectReq

Signature	void tciDisconnectReq (in TriPortIdType fromPort, in TriPortIdType toPort)	
In Parameters	fromPort	Identifier of the test component port to be disconnected.
	toPort	Identifier of the test component port to be disconnected.
Return Value	void	
Constraint	This operation shall be called by the TE when it executes a TTCN-3 disconnect operation.	
Effect	CH transmits the disconnect request to the remote TE where it calls the tciDisconnect 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 tciDisconnect operation on both remote TEs.	

7.3.3.2.10 tciMapReq

Signature	void tciMapReq (in TriPortIdType fromPort, in TriPortIdType toPort)	
In Parameters	fromPort	Identifier of the test component port to be mapped from.
	toPort	Identifier of the test component port to be mapped to.
Return Value	void	
Constraint	This operation shall be called by the TE when it executes a TTCN-3 map operation.	
Effect	CH transmits the map request to the remote TE where it calls the tciMap operation to establish a logical connection between the two indicated ports.	

7.3.3.2.11 tciUnmapReq

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

7.3.3.2.12 tciTestComponentTerminatedReq

Signature	void tciTestComponentTerminatedReq (in TriComponentIdType component, in VerdictValue verdict)	
In Parameters	component	Identifier of the component that has terminated.
	verdict	Verdict after termination of the component.
Return Value	void	
Constraint	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	
Effect	The CH is notified of the termination of the indicated test component. Since a function being executed on a test component can only have <i>in</i> parameters [3], 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.	

7.3.3.2.13 tciTestComponentRunningReq

Signature	TBoolean tciTestComponentRunningReq (in TriComponentIdType component)	
In Parameters	component	Identifier of the component to be checked for running.
Return Value	true if the indicated component is still executing a behaviour, false otherwise.	
Constraint	This operation shall be called by the TE when it executes a TTCN-3 running operation.	
Effect	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.	

7.3.3.1.14 tciTestComponentDoneReq

Signature	TBoolean tciTestComponentDoneReq (in TriComponentIdType component)	
In Parameters	component	Identifier of the component to be checked for done.
Return Value	true if the indicated component has completed executing its behaviour, false otherwise.	
Constraint	This operation shall be called by the TE when it executes a TTCN-3 done operation.	
Effect	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.	

7.3.3.2.15 tciGetMTCReq

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

7.3.3.2.16 tciExecuteTestCaseReq

Signature	void tciExecuteTestCaseReq (in TciTestCaseIdType testCaseId, in TriPortIdListType tsiPortList)	
In Parameters	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.
Return Value	void	
Constraint	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).	
Effect	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.	

7.3.3.2.17 tciResetReq

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

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 to Java language.

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.

8.2 Type mapping

8.2.1 Basic type mapping

Table 2 gives an overview on how the native types **TBoolean**, **TFloat**, **TChar**, **TInteger**, **TString**, **TStringSeq** and **TUniversalChar** are mapped to the Java types.

Table 2: Basic type mappings

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

The native type **TObjId** is defined in the respective section of the **ObjidValue** interface.

8.2.1.1 boolean

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

8.2.1.2 float

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

8.2.1.3 char

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

8.2.1.4 int

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

8.2.1.5 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`.

8.2.1.6 String[]

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

8.2.1.7 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 10646-1 clause 6.2.

8.2.2 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.

8.2.2.1 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 (String name)` Sets the name of this `TciParameter` parameter to `name`;
- `getParameterPassingMode()` Returns the parameter passing mode of this parameter;
- `setParameterPassingMode(TciParameterPassingMode mode)` 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(Value parameter)` 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`.

8.2.2.2 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.

8.2.2.3 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(int index)** Returns the **TciParameter** at the specified position;
- **clear()** Removes all parameters from this **TciParameterList**;
- **add(TciParameter parameter)** Adds parameter to the end of this **TciParameterList**;
- **setParameter(TciParameter[] parameters)** Fills this **TciParameterList** with parameters.

8.2.2.4 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 CHAR              = 4 ;
    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 OBJID            = 11 ;
    public final static int OCTETSTRING      = 12 ;
    public final static int RECORD           = 13 ;
    public final static int RECORD_OF        = 14 ;
    public final static int SET              = 15 ;
    public final static int SET_OF           = 16 ;
    public final static int UNION            = 17 ;
    public final static int UNIVERSAL_CHAR   = 18 ;
    public final static int UNIVERSAL_CHARSTRING = 19 ;
    public final static int VERDICT          = 20 ;
}
```

8.2.2.5 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;
}
```

8.2.2.6 TciSignatureIdType

TciSignatureIdType is mapped to the following interface:

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

8.2.2.7 TciBehaviourIdType

TciBehaviourIdType is mapped to the following interface:

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

8.2.2.8 TciTestCaseIdType

TciTestCaseIdType is mapped to the following interface:

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

8.2.2.9 TciModuleIdType

TciModuleIdType is mapped to the following interface:

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

8.2.2.10 TciModuleParameterIdType

TciModuleParameterIdType is mapped to the following interface:

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

8.2.2.11 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(int index)` Returns the TciModuleParameter at the specified position.

8.2.2.12 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.

8.2.2.13 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(int index)` Returns the `TciParameterType` at the specified position.

8.2.2.14 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 getImportedModules() ;
    public TciModuleId get(int index) ;
}
```

Methods:

- `size()` Returns the number of modules in this list;
- `isEmpty()` Returns `true` if this list contains no modules;
- `getImportedModules()` 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(int index)` Returns the `TciModuleId` at the specified position.

8.2.3 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`, that provides methods to identify types and to retrieve values of a given type.

8.2.3.1 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();
}
```

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`, `CHAR`, `CHARSTRING`, `COMPONENT`, `ENUMERATED`, `FLOAT`, `HEXSTRING`, `INTEGER`, `OBJID`, `OCTETSTRING`, `RECORD`, `RECORD_OF`, `SET`, `SET_OF`, `UNION`, `UNIVERSAL_CHARSTRING`, `UNIVERSAL_CHAR`, `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 TTCN-3, if any. If no encoding variant attribute has been defined the distinct value `null` will be returned.

8.2.4 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.

8.2.4.1 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 TTCN-3, 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.

8.2.4.2 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(int value)` Sets this IntegerValue to the int value value;
- `getInteger()` Returns the int value represented by this IntegerValue.

8.2.4.3 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(float value)` Sets this FloatValue to the float value value;
- `getFloat()` Returns the float value represented by this FloatValue.

8.2.4.4 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(boolean value)` Sets this BooleanValue to the boolean value value;
- `getBoolean()` Returns the boolean value represented by this BooleanValue.

8.2.4.5 ObjidValue

ObjidValue is mapped to the following interface:

```
// TCI IDL ObjidValue
package org.etsi.ttcn.tci;
public interface ObjidValue {
    TciObjId    getObjid ();
    void        setObjid (TciObjId value);
}
```

Methods:

- `getObjid()` Returns the object id value of the TTCN-3 objid;
- `setObjid(TciObjId value)` Sets this ObjidValue to value.

8.2.4.6 TciObjId

TciObjId is mapped to the following interface. The native java representation of a TTCN-3 ObjectId consists of an ordered sequence of TciObjIdElements.

```
package org.etsi.ttcn.tci;
public interface TciObjId {
    public int      size() ;
    public void     setObjElement(TciObjIdElement[] objElements) ;
    public TciObjIdElement getObjElement(int index) ;
}
```

Methods:

- `size()` Returns the size of this Object Id in TciObjIdElements;
- `setObjElement(TciObjIdElement[] objElements)` Sets this ObjId to the list of objElements;
- `getObjElement(int index)` Return the TciObjIdElement at position index.

8.2.4.7 TciObjIdElement

A TciObjIdElement represent a single object element within a TTCN-3 ObjId value. It can be set using different representations like the ASCII representation or as integer.

TciObjIdElement is mapped to the following interface:

```
package org.etsi.ttcn.tci;
public interface TciObjIdElement {
    public void     setElementAsAscii(String element) ;
    public void     setElementAsNumber(int element) ;
    public String   getElementAsAscii() ;
    public int      getElementAsNumber() ;
}
```

Methods:

- `setElementAsAscii(String element)` Sets the internal representation of this ObjIdElement to string value element;
- `setElementAsNumber(int element)` Set this the internal representation of this ObjIdElement to the integer value element;
- `getElementAsAscii()` Returns the internal representation of this ObjIdElement as string;
- `getElementAsNumber()` Returns the internal representation of this ObjIdElement as integer.

8.2.4.8 CharValue

CharValue is mapped to the following interface:

```
// TCI IDL CharValue
package org.etsi.ttcn.tci;
public interface CharValue {
    char    getChar ();
    void    setChar (char value);
}
```

Methods:

- `getChar()` Returns the char value of the TTCN-3 char;
- `setChar(char value)` Sets this CharValue to value.

8.2.4.9 UniversalCharValue

UniversalCharValue is mapped to the following interface:

```
// TCI IDL UniversalCharValue
package org.etsi.ttcn.tci;
public interface UniversalCharValue {
    int    getUniversalChar ();
    void    setUniversalChar (int value);
}
```

Methods:

- `getUniversalChar()` Returns the universal char value of the TTCN-3 universal char as an integer;
- `setUniversalChar(int value)` Sets this `UniversalCharValue` to value.

8.2.4.10 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(String value)` Sets this `CharstringValue` to value;
- `getChar(int 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 0 to length - 1;
- `setChar(int position, char value)` 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(int len)` Sets the length of this `CharstringValue` in chars to len.

8.2.4.11 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(String value)` 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(int position)` 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(int position, int value)` 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(int len)` Sets the length of this `BitstringValue` in bits to `len`.

8.2.4.12 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"0`. The textual representation of the empty TTCN-3 octetstring is `""0`, while its length is zero;
- `setString(String value)` 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"0`;
- `getOctet(int position)` 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(int position, int value)` 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(int len)` Sets the length of this `OctetstringValue` in octets to `len`.

8.2.4.13 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(String value)** Sets the value of this **UniversalCharstringValue** according to the textual representation as defined by **value**;
- **getChar(int position)** 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(int position, char value)** 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(int len)** Sets the length of this **UniversalCharstringValue** in **UniversalChars** to len.

8.2.4.14 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(String value)` 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(int position)` 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(int position, int value)` 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(int len)` Sets the length of this `HexStringValue` in hex digits to `len`.

8.2.4.15 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() ;
}
```

Methods:

- `getField(String 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 can not be obtained from the record the distinct value `null` will be returned;
- `setField(String fieldName, Value value)` 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 relied on that subsequent modifications of `value` will not be considered in the record;
- `getFieldNames()` Returns an array of `String` of field names, the distinct value `null`, if the record has no fields.

8.2.4.16 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) ;
}
```

Methods:

- **getField(String fieldName)** Returns the value of the `record of` at 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(int position, Value value)**
Sets the field at `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 relied on that subsequent modifications of `value` will not be considered in the `record of`;
- **appendField(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 will be copied. Therefore it should be relied on 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(int len)** 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.

8.2.4.17 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(String 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 TTCN-3;
- **setVariant(String variantName, Value value)** 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 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 will be returned.

8.2.4.18 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);
}
```

Methods

- **getEnum()** Returns the string identifier of this **EnumeratedValue**. This identifier equals the identifier in the TTCN-3 specification;
- **setEnum(String enumValue)** Set the enum to *enumValue*. If *enumValue* is not an allowed value for this enumeration the operation will be ignored.

8.2.4.19 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:

- `verdictValue()` Returns the integer value for this `VerdictValue`. The integer is one of the following constants: `ERROR`, `FAIL`, `INCONC`, `NONE`, `PASS`;
- `setVerdict(int 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 `ERROR` and then to `PASS`.

8.3 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 8.3.2.2).

- `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.CHAR;`
- `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.OBJID;`
- `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.UNION;`
- `org.etsi.ttcn.tci.TciTypeClass.UNIVERSAL_CHAR;`
- `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;`

8.4 Mapping of interfaces

The TCI IDL definition defines three interfaces, the **TCI-TM**, the **TCI-CH** and the **TCI-CD** interface. As the operations are defined for different directions within this interface, i.e. some operations can only be called by the TTCN-3 Executable (TE) 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 3: Sub interfaces

Calling/Called	TE	TM	CD	CH
TE	-	TCI-TM Provided	TCI-CD Provided	TCI-CH Provided
TM	TCI-TM Required	-	-	-
CD	TCI-CD Required	-	-	-
CH	TCI-CH Required	-	-	-

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

8.4.1 The TCI-TM interface

The **TCI-TM** interface is divided into two sub interfaces, the **TCI-TM Provided** interface, defining calls from the TE to the TM and the **TCI-TM Required** interface, defining calls from the TM to the TE.

8.4.1.1 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 TciTMPProvided {
    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);
}
```

8.4.1.2 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 TciModuleParameterList tciGetModuleParameters (TciModuleId moduleId);
    public TciTestCaseIdList tciGetTestCases ();
    public TciParameterTypeList tciGetTestCaseParameters (TciTestCaseId TestId);
    public TriPortIdList tciGetTestCaseTSI (TciTestCaseId testId);
    public void tciStartTestCase(String testId, TciParameterList parameterList ) ;
    public void tciStopTestCase () ;
    public TriComponentId tciStartControl () ;
    public void tciStopControl () ;
}
```

8.4.2 The TCI-CD interface

The **TCI-CD** interface is divided into two sub interfaces, the **TCI-CD Provided** interface, defining calls from the TE to the CD and the **TCI-CD Required** interface, defining calls from the CD to the TE.

8.4.2.1 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);
}
```

8.4.2.2 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      getChar ();
    public Type      getUniversalChar ();
    public Type      getObjid ();
    public Type      getCharstring ();
    public Type      getUniversalCharstring ();
    public Type      getHexstring ();
    public Type      getBitstring ();
    public Type      getOctetstring ();
    public Type      getVerdict ();
    public void      tciErrorReq (String message);
}
```

8.4.3 The TCI-CH interface

The **TCI-CH** interface is divided into two sub interfaces, the TCI-CH Provided interface, defining calls from the TE to the CH and the TCI-CH Required interface, defining calls from the CH to the TE.

8.4.3.1 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 tciCallConnected(TriPortId sender,
                                TriComponentId receiver,
                                TriSignatureId signature,
                                TciParameterList parameterList) ;

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

    public void tciRaiseConnected(TriPortId sender,
                                  TriComponentId receiver,
                                  TciSignatureId signature,
                                  Value except) ;

    public TriComponentId tciCreateTestComponentReq(int kind, Type componentType) ;

    public void tciStartTestComponentReq(TriComponentId comp,
                                          TciBehaviourId behavior,
                                          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      tciUnmapReq(TriPortId fromPort, TriPortId toPort);

public void      tciExecuteTestCaseReq(TriComponentId testComponentId,
                                       TriPortIdList tsiPortList);

public void      tciResetReq() ;

public boolean    tciTestComponentDoneReq(TriComponentId testComponentId) ;
}

```

8.4.3.2 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) ;

    public void      tciStartTestComponent(TriComponentId comp,
                                           TciBehaviourId behavior,
                                           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      tciUnmap (TriPortId fromPort, TriPortId toPort);
}

```

8.5 Optional parameters

Clause 7.4 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)`.

8.6 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.

8.7 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 or CD produces an error, this erroneous situation should be communicated to the TE using the procedures as defined in clause 7.3.2.1.14 (`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 TRI ANSI-C language mapping for the TCI data specified in clause 7.2 and for the TCI operations specified in clause 7.3.

9.2 Value interface

TCI IDL Interface	ANSI C representation	Notes and comments
Type		
<code>TciModuleIdType getDefiningModule()</code>	<code>TciModuleIdType getDefiningModule(TciType inst)</code>	
<code>TString getName()</code>	<code>String getName(TciType inst)</code>	String type reused from TRI (OMG recommendation) [1].
<code>TciTypeClassType getTypeClass()</code>	<code>TciTypeClassType tciGetTypeClass (TciType inst)</code>	
<code>Value newInstance()</code>	<code>TciValue tciNewInstance(TciType inst)</code>	
<code>TString getTypeEncoding()</code>	<code>String tciGetTypeEncoding(TciType inst)</code>	
<code>TString getTypeEncodingVariant()</code>	<code>String tciGetTypeEncodingVariant(TciType inst)</code>	
Value		
<code>TString getValueEncoding()</code>	<code>String tciGetValueEncoding(TciValue inst)</code>	
<code>TString getValueEncodingVariant()</code>	<code>String tciGetValueEncodingVariant(TciValue inst)</code>	
<code>Type getType()</code>	<code>TciType tciGetType(TciValue inst)</code>	
<code>TBoolean notPresent()</code>	<code>Boolean tciNotPresent(TciValue inst)</code>	Boolean type reused from TRI (OMG recommendation) [1].

TCI IDL Interface	ANSI C representation	Notes and comments
IntegerValue		
TInteger getInt()	String tciGetIntAbs(TciValue inst)	Returns the (10-base) integer absolute value as an ASCII string.
	unsigned long int tciGetIntNumberOfDigits (TciValue inst)	Returns the number of digits in an integer value.
	Boolean tciGetIntSign(TciValue inst)	Returns true if the number is positive, false otherwise
	char tciGetIntDigit (TciValue 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(TciValue 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 (TciValue inst, unsigned long int dig_num)	Sets the number of digits in an integer value.
	void tciSetIntSign (TciValue inst, Boolean sign)	Sets the sign to + (true) or - (false).
	void tciSetIntDigit (TciValue inst, unsigned long int position, char digit)	Sets the value of the digit at position 'position', where position 0 is the least significant digit.
FloatValue		
TFloat getFloat()	double tciGetFloatValue(TciValue inst)	
void setFloat(in TFloat value)	void tciSetFloatValue(TciValue inst, double value)	
BooleanValue		
TBoolean getBoolean()	Boolean tciGetBooleanValue(TciValue inst)	
void setBoolean (in TBoolean value)	void tciSetBooleanValue (TciValue inst, Boolean value)	
ObjidValue		
TObjid getObjid()	TciObjidValue tciGetTciObjidValue(TciValue inst)	
void setObjid(in TObjid value)	void tciSetObjidValue(TciValue inst, TciObjidValue value)	
CharValue		
TChar getChar()	char tciGetCharValue(TciValue inst)	
void setChar(in TChar value)	void tciSetCharValue(TciValue inst, char value)	
UniversalCharValue		
TUniversalChar getUniversalChar()	TciUCValue tciGetUniversalCharValue (TciValue inst)	
void setUniversalChar (in TUniversalChar value)	void tciSetUniversalCharValue (TciValue inst, TciUCType value)	

TCI IDL Interface	ANSI C representation	Notes and comments
CharstringValue		
TString getString()	TciCharStringValue tciGetCStringValue(TciValue inst)	
void setString(in TString value)	void tciSetCStringValue (TciValue inst, TciCharStringValue value)	
TChar getChar (in TInteger position)	char tciGetCStringCharValue (TciValue inst, long int position)	
void setChar (in TInteger position, in char value)	void tciSetCStringCharValue (TciValue inst, long int position, char value)	
TInteger getLength()	unsigned long int tciGetCStringLength (TciValue inst)	
void setLength(in TInteger len)	void tciSetCStringLength (TciValue inst, unsigned long int len)	
UniversalCharstringValue		
TString getString()	TciUCStringValue tciGetUCStringValue(TciValue inst)	
void setString(in TString value)	void tciSetUCStringValue (TciValue inst, TciUCStringValue value)	
TUniversalChar getChar (in TInteger position)	TciUCValue tciGetUCStringCharValue (TciValue inst, unsigned long int position)	
void setChar (in TInteger position, in TUniversalChar value)	void tciSetUCStringCharValue (TciValue inst, unsigned long int position, TciUCValue value)	
TInteger getLength()	unsigned long int tciGetUCStringLength(TciValue inst)	
void setLength(in TInteger len)	void tciSetUCStringLength (TciValue inst, unsigned long int len)	
BitstringValue		
TString getString()	String tciGetBStringValue(TciValue inst)	
void setString(in TString value)	void tciSetBStringValue (TciValue inst, String value)	
TChar getBit (in integer position)	int tciGetBStringBitValue (TciValue inst, long int position)	
void setBit (in TInteger position, in TInteger value)	void tciSetBStringBitValue (TciValue inst, unsigned long int position, int value)	
TInteger getLength()	unsigned long int tciGetBStringLength(TciValue inst)	
void setLength(in TInteger len)	void tciSetBStringLength (TciValue inst, long int len)	
HexstringValue		
TString getString()	String tciGetHStringValue(TciValue inst)	
void setString(in TString value)	void tciSetHStringValue (TciValue inst, String value)	
TChar getHex (in TInteger position)	int tciGetHStringHexValue (TciValue inst, unsigned long int position)	
void setBit (in TInteger position, in TInteger value)	void tciSetHStringHexValue (TciValue inst, unsigned long int position, int value)	
TInteger getLength()	long int tciGetHStringLength(TciValue inst)	
void setLength(in TInteger len)	void tciSetHStringLength (TciValue inst, unsigned long int len)	

TCI IDL Interface	ANSI C representation	Notes and comments
OctetstringValue		
TString getString()	String tciGet0StringValue(TciValue inst)	
void setString(in TString value)	void tciSet0StringValue (TciValue inst, String value)	
TChar getOctet(in TInteger position)	int tciGet0StringOctetValue (TciValue inst, unsigned long int position)	
void setOctet (in TInteger position, in TInteger value)	void tciSet0StringOctetValue (TciValue inst, unsigned long int position, int value)	
TInteger getLength()	unsigned long int tciGet0StringLength(TciValue inst)	
void setLength(in TInteger len)	void tciSet0StringLength (TciValue inst, unsigned long int len)	
RecordValue		
Value getField(in TString fieldName)	TciValue tciGetRecFieldValue (TciValue inst, String fieldName)	
void setField (in TString fieldName, in Value value)	void tciSetRecFieldValue (TciValue inst, String fieldName, TciValue value)	
TString[] getFieldNames()	char** tciGetRecFieldNames(TciValue inst)	
RecordOfValue		
Value getField(in TInteger position)	TciValue tciGetRecOfFieldValue (TciValue inst, unsigned long int position)	
void setField (in TInteger position, in Value value)	void tciSetRecOfFieldValue (TciValue inst, unsigned long int position, TciValue value)	
void appendField(in Value value)	void appendRecOfFieldValue (TciValue inst, TciValue value)	
Type getElementType()	TciType tciGetRecOfElementType(TciValue inst)	
TInteger getLength()	unsigned long int tciGetRecOfLength(TciValue inst)	
void setLength(in TInteger len)	void tciSetRecOfLength (TciValue inst, unsigned long int len)	
UnionValue		
Value getVariant (in TString variantName)	TciValue tciGetUnionVariant (TciValue inst, String variantName)	
void setVariant (in TString variantName, in Value value)	void tciSetUnionVariant (TciValue inst, String variantName, TciValue value)	
TString getPresentVariantName()	String tciGetUnionPresentVariantName (TciValue inst)	
TString[] getVariantNames()	char** tciGetUnionVariantNames(TciValue inst)	
EnumeratedValue		
TString getEnum()	String tciGetEnumValue(TciValue inst)	
void setEnum(in TString enumValue)	void tciSetEnumValue (TciValue inst, String enumValue)	
VerdictValue		
TInteger getVerdict()	int tciGetVerdictValue(TciValue inst)	
void setVerdict(in TInteger verdict)	void tciSetVerdictValue(TciValue inst, int verdict)	

9.3 Operation interface

TCI-TM required		
void tciRootModule (in TciModuleIdType moduleId)	void tciRootModule(String moduleId)	
TciModuleParameterListType tciGetModuleParameters (in TciModuleIdType moduleName)	TciModuleParameterListType tciGetModuleParameters (TciModuleIdType moduleName)	
TciTestCaseIdListType tciGetTestCases()	TciTestCaseIdListType tciGetTestCases()	
TciParameterTypeListType tciGetTestCaseParameters (in TciTestCaseIdType testCaseId)	TciParameterTypeListType tciGetTestCaseParameters (TciTestCaseIdType testCaseId)	
TriPortIdListType tciGetTestCaseTSI (in TciTestCaseIdType testCaseId)	TriPortIdList tciGetTestCaseTSI (TciTestCaseIdType testCaseId)	
void tciStartTestCase (in TciTestCaseIdType testCaseId, in TciParameterListType parameterlist)	void tciStartTestCase (TciTestCaseIdType testCaseId, TciParameterListType parameterlist)	
void tciStopTestCase()	void tciStopTestCase()	
TriComponentId tciStartControl()	TriComponentId tciStartControl()	
void tciStopControl()	void tciStopControl()	
TCI-TM provided		
void tciTestCaseStarted (in TciTestCaseIdType testCaseId, in TciParameterListType parameterList, in Tfloat timer)	void tciTestCaseStarted (TciTestCaseIdType testCaseId, TciParameterListType parameterList, double timer)	
void tciTestCaseTerminated (in VerdictValue verdict, in TciParameterListType parameterlist)	void tciTestCaseTerminated (TciVerdictValue verdict, TciParameterListType parameterlist)	
void tciControlTerminated()	void tciControlTerminated()	
Value tciGetModulePar (in TciModuleParameterIdType parameterId)	tciValue tciGetModulePar (TciModuleParameterIdType parameterId)	
void tciLog(in String message)	void tciLog(String message)	
void tciError(in String message)	void tciError(String message)	
TCI-CD required		
Type getTypeForName (in String typeName)	TciType TypeForName (String typeName)	
Type getInteger()	TciType tciGetIntegerType()	
Type getFloat()	TciType tciGetFloatType()	
Type getBoolean()	TciType tciGetBooleanType()	
Type getChar()	TciType tciGetCharType()	
Type getUniversalChar()	TciType tciGetUniversalCharType()	
Type getObjid()	TciType tciGetTciObjidType()	
Type getCharstring()	TciType tciGetTciCharstringType()	
Type getUniversalCharstring()	TciType tciGetUniversalCharstringType()	
Type getHexstring()	TciType tciGetHexstringType()	
Type getBitstring()	TciType tciGetBitstringType()	
Type getOctetstring()	TciType tciGetOctetstringType()	
Type getVerdict()	TciType tciGetVerdictType()	
void tciErrorReq(in String message)	void tciErrorReq(String message)	
TCI-CD provided		
Value decode (in TriMessageType message, in Type decodingHypothesis)	TciValue tciDecode (BinaryString message, TciType decHypothesis)	BinaryString type reused from TRI
TriMessageType encode (in Value value)	BinaryString tciEncode(TciValue value)	

TCI-CH required		
void tciEnqueueMsgConnected (in TriPortIdType sender, in TriComponentIdType receiver, in Value rcvdMessage)	void tciEnqueueMsgConnected (TriPortId sender, TriComponentId receiver, TciValue rcvdMessage)	
void tciEnqueueCallConnected (in TriPortIdType sender, in TriComponentIdType receiver, in TriSignatureIdType signature, in TciParameterListType parameterList)	void tciEnqueueCallConnected (TriPortId sender, TriComponentId receiver, TriSignatureIdType signature, TciParameterListType parameterList)	
void tciEnqueueReplyConnected (in TriPortIdType sender, in TriComponentIdType receiver, in TriSignatureIdType signature, in TciParameterListType parameterList, in Value returnValue)	void tciEnqueueReplyConnected (TriPortId sender, TriComponentId receiver, TriSignatureIdType signature, TciParameterListType parameterList, TciValue returnValue)	
void tciEnqueueRaiseConnected (in TriPortIdType sender, in TriComponentIdType receiver, in TriSignatureIdType signature, in Value exception)	void tciEnqueueRaiseConnected (TriPortId sender, TriComponentId receiver, TriSignatureIdType signature, TciValue exception)	
TriComponentIdType tciCreateTestComponent (in TciTestComponentKindType kind in Type componentType)	TriComponentId tciCreateTestComponent (TciTestComponentKindType kind, TciType componentType)	
void tciStartTestComponent (in TriComponentIdType component, in TciBehaviourIdType behavior, in TciParameterListType parameterList)	void tciStartTestComponent (TriComponentId component, TciBehaviourIdType behavior, TciParameterListType parameterList)	
void tciStopTestComponent (in TriComponentIdType component)	void tciStopTestComponent (TriComponentId component)	
void tciConnect (in TriPortIdType fromPort, in TriPortIdType toPort)	void tciConnect (TriPortId fromPort, TriPortId toPort)	
void tciDisconnect (in TriPortIdType fromPort, in TriPortIdType toPort)	void tciDisconnect (TriPortId fromPort, TriPortId toPort)	
void tciMap (in TriPortIdType fromPort, in TriPortIdType toPort)	void tciMap (TriPortIdType fromPort, TriPortIdType toPort)	
void tciUnmap (in TriPortIdType fromPort, in TriPortIdType toPort)	void tciUnmap (TriPortIdType fromPort, TriPortIdType toPort)	
void tciTestComponentTerminated (in TriComponentIdType component, in VerdictValue verdict)	void tciTestComponentTerminated (TriComponentId component, TciVerdictValue verdict)	
boolean tciTestComponentRunning (in TriComponentIdType component)	Boolean tciTestComponentRunning (TriComponentId component)	
boolean tciTestComponentDone (in TriComponentIdType component)	boolean tciTestComponentDone (TriComponentId component)	
TriComponentIdType tciGetMTC()	TriComponentId tciGetMTC()	
void tciExecuteTestCase (in TciTestCaseIdType testCaseId, in TriPortIdListType tsiPortList)	void tciExecuteTestCase (TciTestCaseIdType testCaseId, TriPortIdList tsiPortList)	
void tciReset()	void tciReset()	

TCI-CH provided		
void tciSendConnected (in TriPortIdType sender, in TriComponentIdType receiver, in Value sendMessage)	void tciSendConnected (TriPortId sender, TriComponentId receiver, TciValue sendMessage)	
void tciCallConnected (in TriPortIdType sender, in TriComponentIdType receiver, in TriSignatureIdType signature, in TciParameterListType parameterList)	void tciCallConnected (TriPortId sender, TriComponentId receiver, TriSignatureIdType signature, TciParameterListType parameterList)	
void tciReplyConnected (in TriPortIdType sender, in TriComponentIdType receiver, in TriSignatureIdType signature, in TciParameterListType parameterList, in Value returnValue)	void tciReplyConnected (TriPortId sender, TriComponentId receiver, TriSignatureIdType signature, TciParameterListType parameterList, TciValue returnValue)	
void tciRaiseConnected (in TriPortIdType sender, in TriComponentIdType receiver, in TriSignatureIdType signature, in Value exception)	void tciRaiseConnected (TriPortId sender, TriComponentId receiver, TriSignatureIdType signature, TciValue exception)	
TriComponentIdType tciCreateTestComponentReq (in TciTestComponentKindType kind, in Type componentType)	TriComponentId tciCreateTestComponentReq (TciTestComponentKindType kind, TciType componentType)	
void tciStartTestComponentReq (in TriComponentIdType component, in TciBehaviourIdType behavior, in TciParameterListType parameterList)	void tciStartTestComponentReq (TriComponentId component, TciBehaviourIdType behavior, TciParameterListType parameterList)	
void tciStopTestComponentReq (in TriComponentIdType component)	void tciStopTestComponentReq (TriComponentId component)	
void tciConnectReq (in TriPortIdType fromPort, in TriPortIdType toPort)	void tciConnectReq (TriPortId fromPort, TriPortId toPort)	
void tciDisconnectReq (in TriPortIdType fromPort, in TriPortIdType toPort)	void tciDisconnectReq (TriPortId fromPort, TriPortId toPort)	
void tciMapReq (in TriPortIdType fromPort, in TriPortIdType toPort)	void tciMapReq (TriPortId fromPort, TriPortId toPort)	
void tciUnmapReq (in TriPortIdType fromPort, in TriPortIdType toPort)	void tciUnmapReq (TriPortId fromPort, TriPortId toPort)	
void tciTestComponentTerminatedReq (in TriComponentIdType component, in VerdictValue verdict)	void tciTestComponentTerminatedReq (TriComponentId component, TciVerdictValue verdict)	
boolean tciTestComponentRunningReq (in TriComponentIdType component)	Boolean tciTestComponentRunningReq (TriComponentId component)	
boolean tciTestComponentDoneReq (in TriComponentIdType component)	Boolean tciTestComponentDoneReq (TriComponentId component)	
TriComponentIdType tciGetMTCReq()	TriComponentId tciGetMTCReq()	
void tciExecuteTestCaseReq (in TciTestCaselIdType testCaselId, in TriPortIdListType tsiPortList)	void tciExecuteTestCaseReq (TciTestCaseIdType testCaseId, TriPortIdList tsiPortList)	
void tciResetReq()	void tciResetReq()	

9.4 Data

TCI IDL ADT	ANSI C representation (Type definition)	Notes and comments
TciModuleIdType	QualifiedName	
TciModuleParameterType	typedef struct TciModuleParameterType { String parName; TciValue defaultValue; } TciModuleParameterType;	
TciModuleParameterListType	typedef struct TciModuleParameterListType { long int length; TciModuleParameterType[] modParList; } TciModuleParameterListType;	
TciParameterType	typedef struct TciParameterType { TciParameterPassingModeType parPassMode; TciValue 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; TciType[] parList; } TciParameterTypeListType;	length 0 shall be interpreted as "empty list".
TciTestCaseIdListType	typedef struct TciTestCaseIdListType { long int length; QualifiedName[] idList; } TciTestCaseIdListType;	length 0 shall be interpreted as "empty list".
TciTypeClassType	typedef enum { TCI_ADDRESS_TYPE, TCI_ANYTYPE_TYPE, TCI_BITSTRING_TYPE, TCI_BOOLEAN_TYPE, TCI_CHAR_TYPE, TCI_CHARSTRING_TYPE, TCI_COMPONENT_TYPE, TCI_ENUMERATED_TYPE, TCI_FLOAT_TYPE, TCI_HEXSTRING_TYPE, TCI_INTEGER_TYPE, TCI_OBJID_TYPE, TCI_OCTETSTRING_TYPE, TCI_RECORD_TYPE, TCI_RECORD_OF_TYPE, TCI_SET_TYPE, TCI_SET_OF_TYPE, TCI_UNION_TYPE, TCI_UNIVERSAL_CHAR_TYPE, TCI_UNIVERSAL_CHARSTRING_TYPE, TCI_VERDICT_TYPE } TciTypeClassType;	

TCI IDL ADT	ANSI C representation (Type definition)	Notes and comments
TciTestComponentKindType	typedef enum { TCI_CTRL_COMP, TCI_MTC_COMP, TCI_PTC_COMP, TCI_SYS_COMP } TciTestComponentKindType;	
TciSignatureIdType	QualifiedName	QualifiedName type reused from TRI.
TciBehaviourIdType	QualifiedName	

9.5 Miscellaneous

TCI concept	ANSI C representation	Notes and comments
Verdict representation		
NONE	const int TCI_VERDICT_NONE = 0	Since the TciVerdictValue interface is defined in terms of integers, consensus must be established on which value defines which verdict.
PASS	const int TCI_VERDICT_PASS = 1	
INCONC	const int TCI_VERDICT_INCONC = 2	
FAIL	const int TCI_VERDICT_FAIL = 3	
ERROR	const int TCI_VERDICT_ERROR = 4	
Objid representation		
Objid	typedef struct TciObjidValue { long int length; TciObjidElem[] elements; } TciObjidValue;	Since the Objid value is returned "as is" via the Objid value interface, a representation must be defined.
TciObjidElem	typedef struct TciObjidElemValue { char* elem_as_ascii; long int elem_as_number; void* aux; } TciObjidElemValue;	
CharstringValue representation		
TciCharString	typedef struct TciCharStringValue { unsigned long int length; char* string; } TciCharStringValue	
Universal Character[string] representation		
Universal Char	typedef unsigned char[4] TciUCValue	
Universal Charstring	typedef struct TciUCStringValue { unsigned long int length; TciUCType[] string; } TciUCStringValue;	

10 Use scenarios

This clause 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.

10.1 Initialization, collecting information, logging

10.1.1 Use scenario: initialization

The scenario in figure 7 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.

10.1.1.1 Sequence diagram

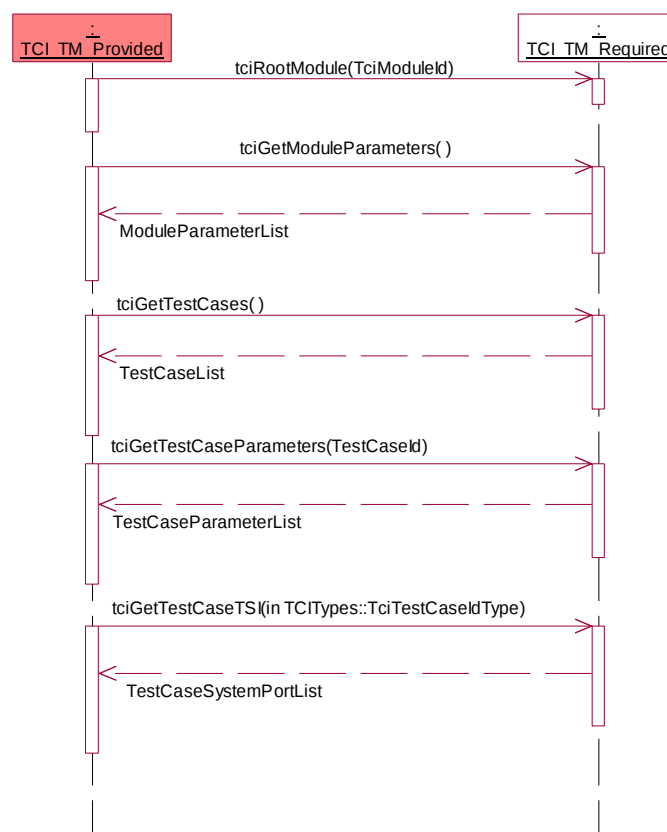


Figure 7: Use scenario - initialization

10.1.1.2 TTCN-3 fragment

The initialization is outside the scope of TTCN-3.

10.1.2 Use scenario: requesting module parameters

The scenario in figure 8 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.

10.1.2.1 Sequence diagram

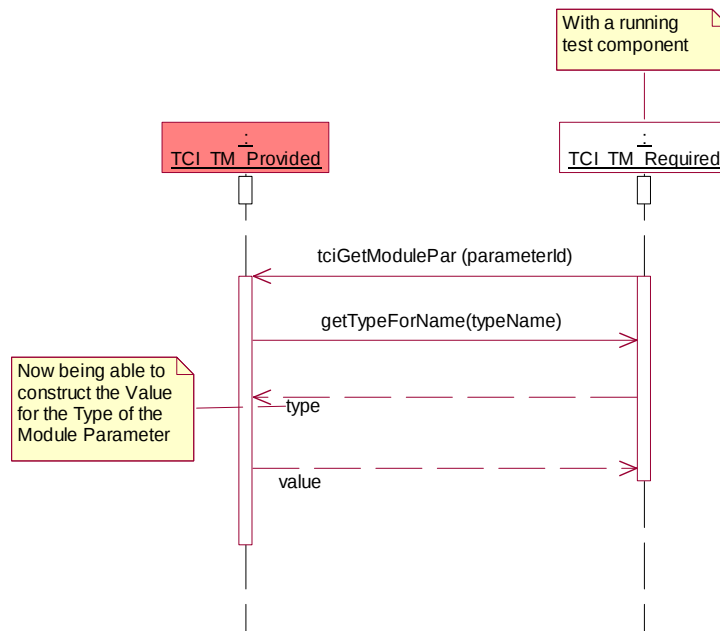


Figure 8: Use scenario - requesting module Pars

10.1.2.2 TTCN-3 fragment

```

module AModule {
  ...
  modulepar {
    integer AModulePar
  }
  ...
  function AFunction (...) ... {
    integer x;
    ...
    x:= 2+AModulePar; // an expression with a module parameter
    ...
  }
  ...
}

```

10.1.3 Use scenario: logging

The scenario in figure 9 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 management.

10.1.3.1 Sequence diagram

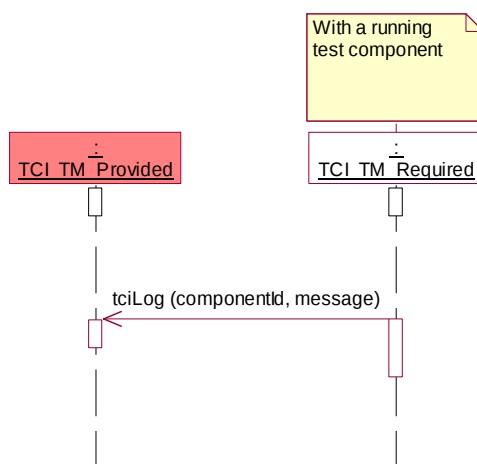


Figure 9: Use scenario - logging

10.1.3.2 TTCN-3 fragment

```

module AModule {
    ...
    function AFunction (...) ... {
        ...
        log('AMessage');
        ...
    }
    ...
}

```

10.2. Execution of test cases and control

10.2.1 Use scenario: execution of control

The scenario in figure 10 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.

10.2.1.1 Sequence diagram

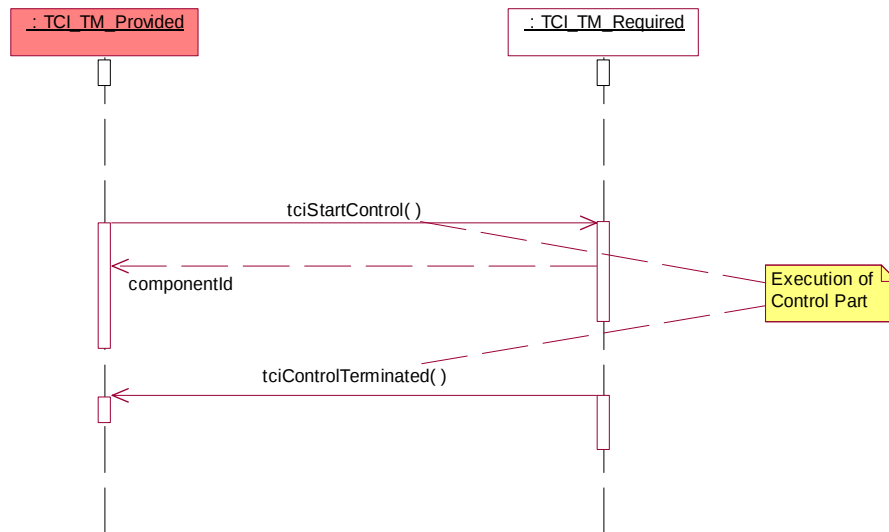


Figure 10: Use scenario - execution of control

10.2.1.2 TTCN-3 fragment

```

module AModule {
  ...
  control {
    ...
  }
  ...
}

```

10.2.2 Use scenario: test case execution within control

The scenario in figure 11 shows how a test case is executed within the control part.

10.2.2.1 Sequence diagram

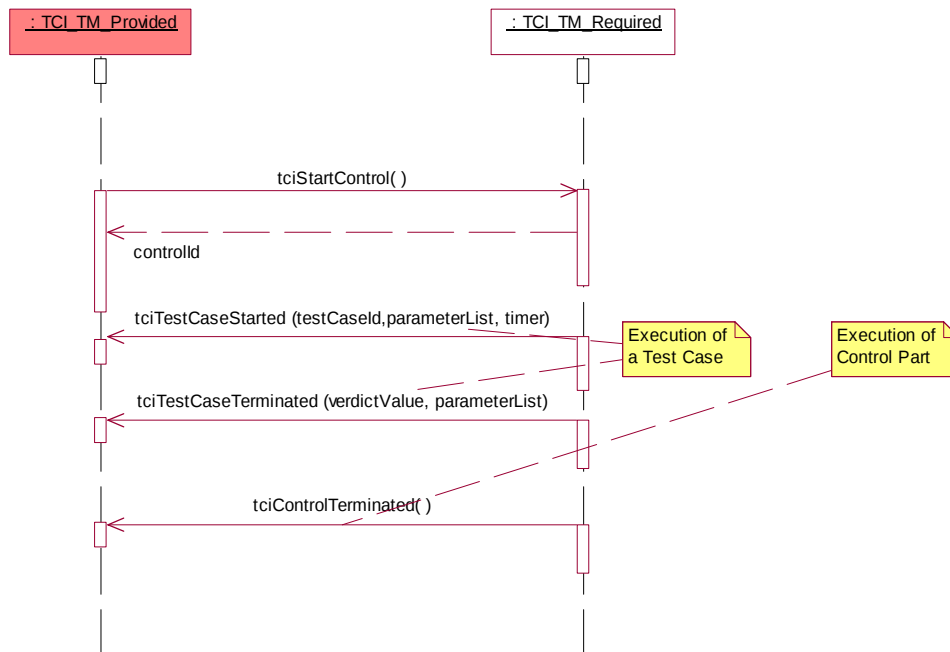


Figure 11: Use scenario - test case execution within control

10.2.2.2 TTCN-3 fragment

```

module AModule {
  ...
  testcase ATestCase(...)... {
    ... //the test case behaviour
  }
  ...
  control {
    ...
    execute(ATestCase(...));
    ...
  }
  ...
}

```

10.2.3 Use scenario: direct test case execution

The scenario in figure 12 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.

10.2.3.1 Sequence diagram

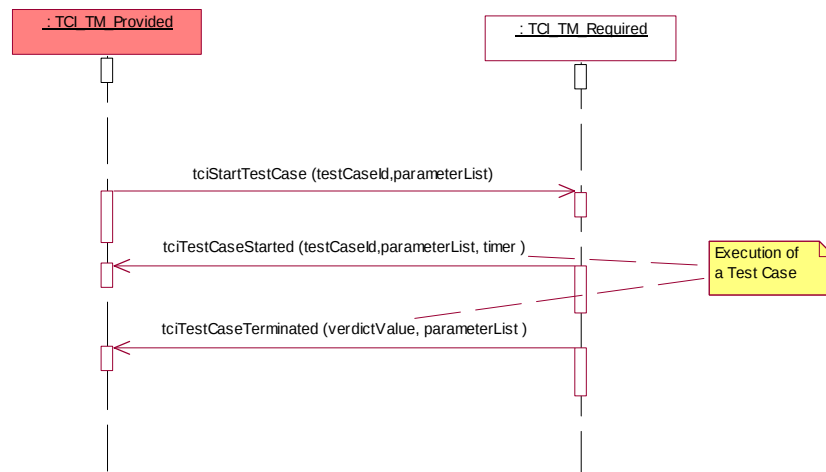


Figure 12: Use scenario - direct test case execution

10.2.3.2 TTCN-3 fragment

The direct execution of a test case is outside the scope of TTCN-3.

10.2.4 Use scenario: execute test case to TRI

The scenario in figure 13 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.

10.2.4.1 Sequence diagram

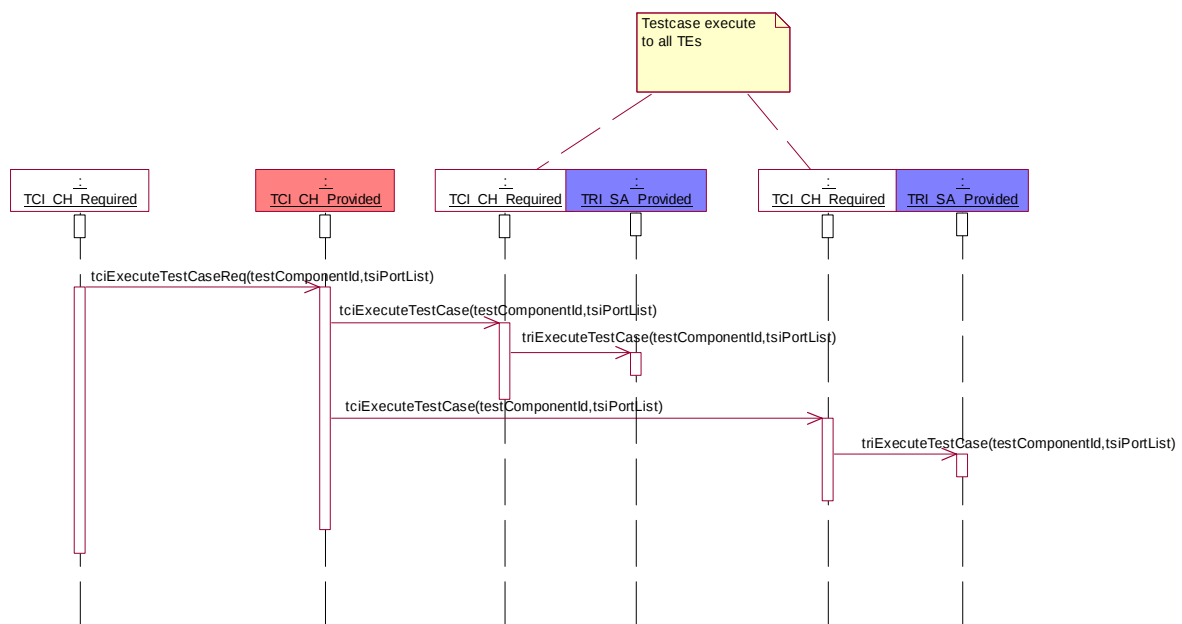


Figure 13: Use scenario - execute test case to TRI

10.2.4.2 TTCN-3 fragment

```

module AModule {
  ...
  testcase ATestCase(...)... {
    ... //the test case behaviour
  }
  ...
  control {
    ...
    execute(ATestCase(...));
    ...
  }
  ...
}

```

10.3 Component handling

10.3.1 Use scenario: local control component creation

The scenario in figure 14 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.

10.3.1.1 Sequence diagram

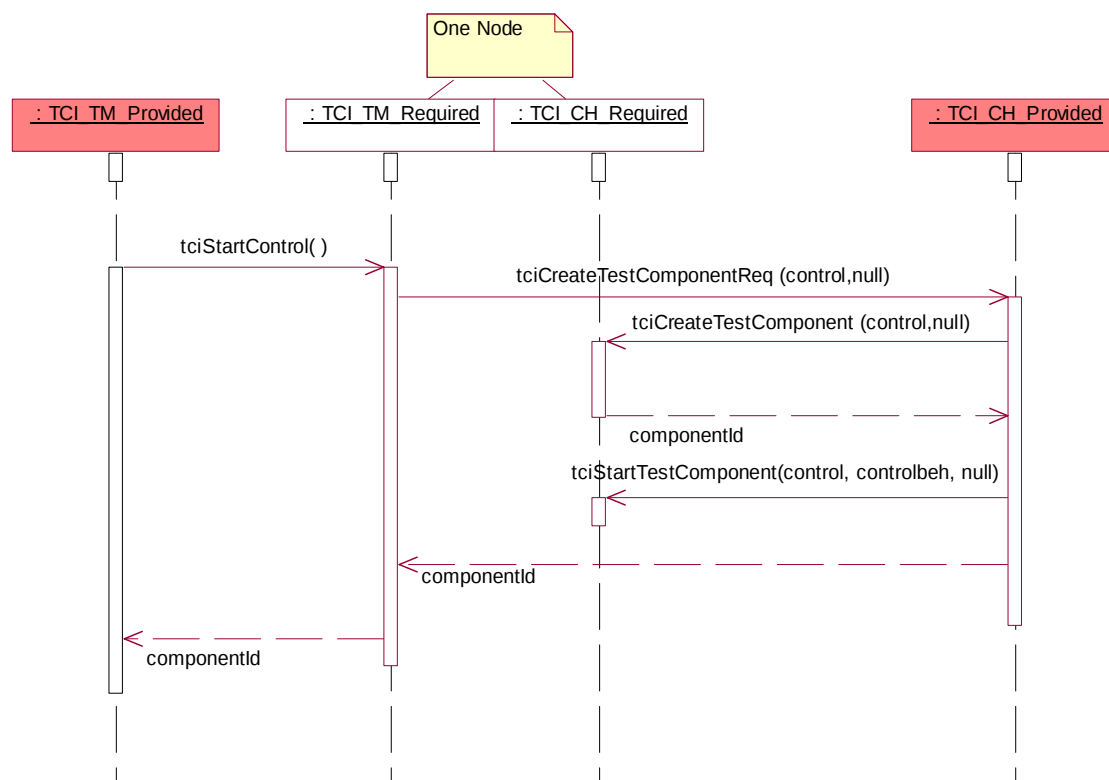


Figure 14: Use scenario - local control component creation

10.3.1.2 TTCN-3 fragment

```

module AModule {
  ...
  control {
    ...
  }
  ...
}

```

10.3.2 Use scenario: remote control component creation

The scenario in figure 15 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.

10.3.2.1 Sequence diagram

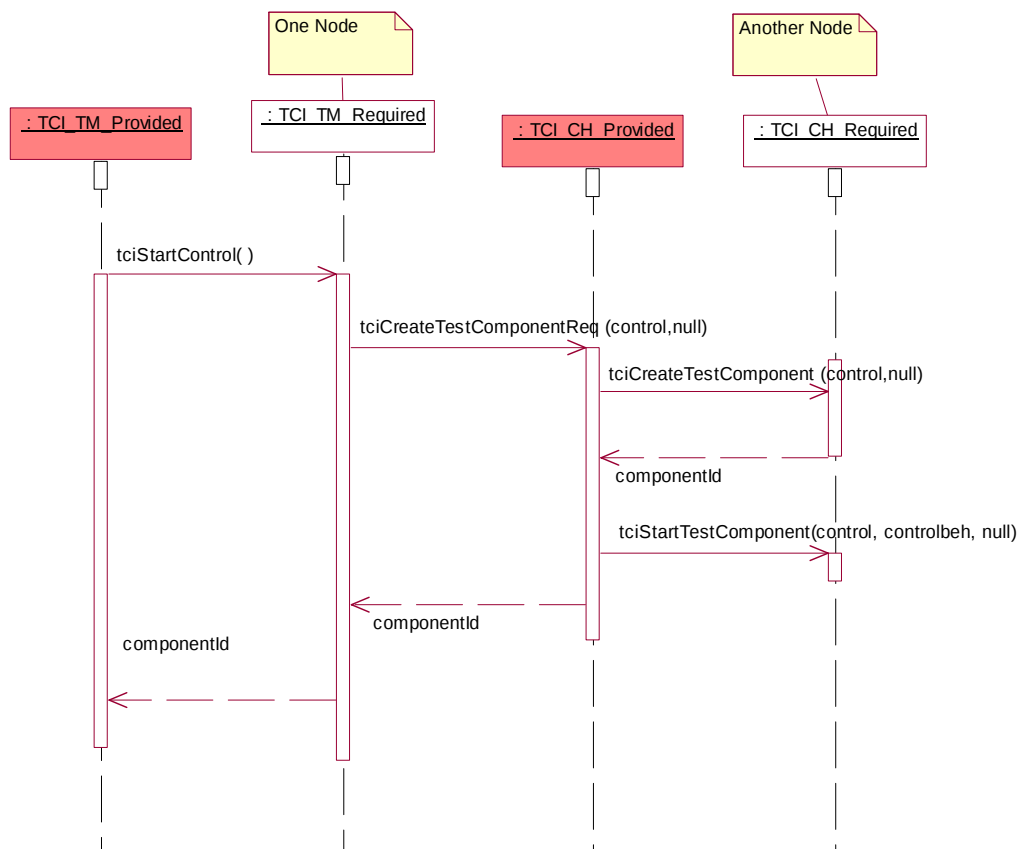


Figure 15: Use scenario - remote control component creation

10.3.2.2 TTCN-3 fragment

```

module AModule {
  ...
  control {
    ...
  }
  ...
}

```

10.3.3 Use scenario: local MTC creation

The scenario in figure 16 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 clause 10.3.6 and clause 10.3.5).

10.3.3.1 Sequence diagram

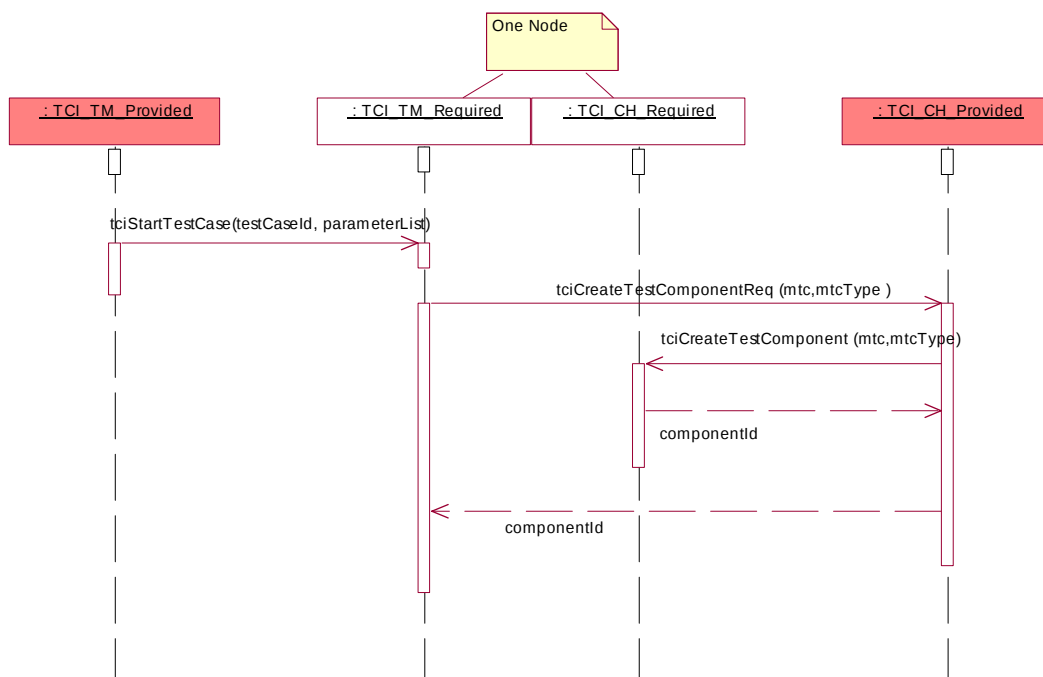


Figure 16: Use scenario - local MTC creation

10.3.3.2 TTCN-3 fragment

```

module AModule {
  ...
  testcase ATestCase (...)runs on MTCType... {
    ... //the test case behaviour
  }
  ...
}
  
```

10.3.4 Use scenario: remote MTC creation

The scenario in figure 17 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 clause 10.3.6 and clause 10.3.5).

10.3.4.1 Sequence diagram

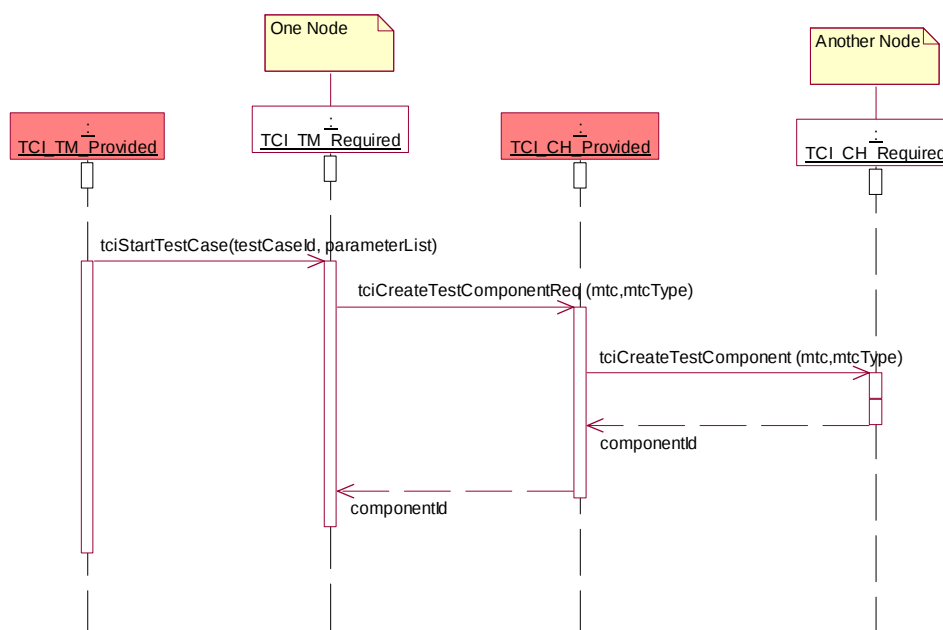


Figure 17: Use scenario - remote MTC creation

10.3.4.2 TTCN-3 fragment

```

module AModule {
  ...
  testcase ATestCase(...)runs on MTCType ... {
    ... //the test case behaviour
  }
  ...
}

```

10.3.5 Use scenario: component handling for test case execution within control

The scenario in figure 18 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.

10.3.5.1 Sequence diagram

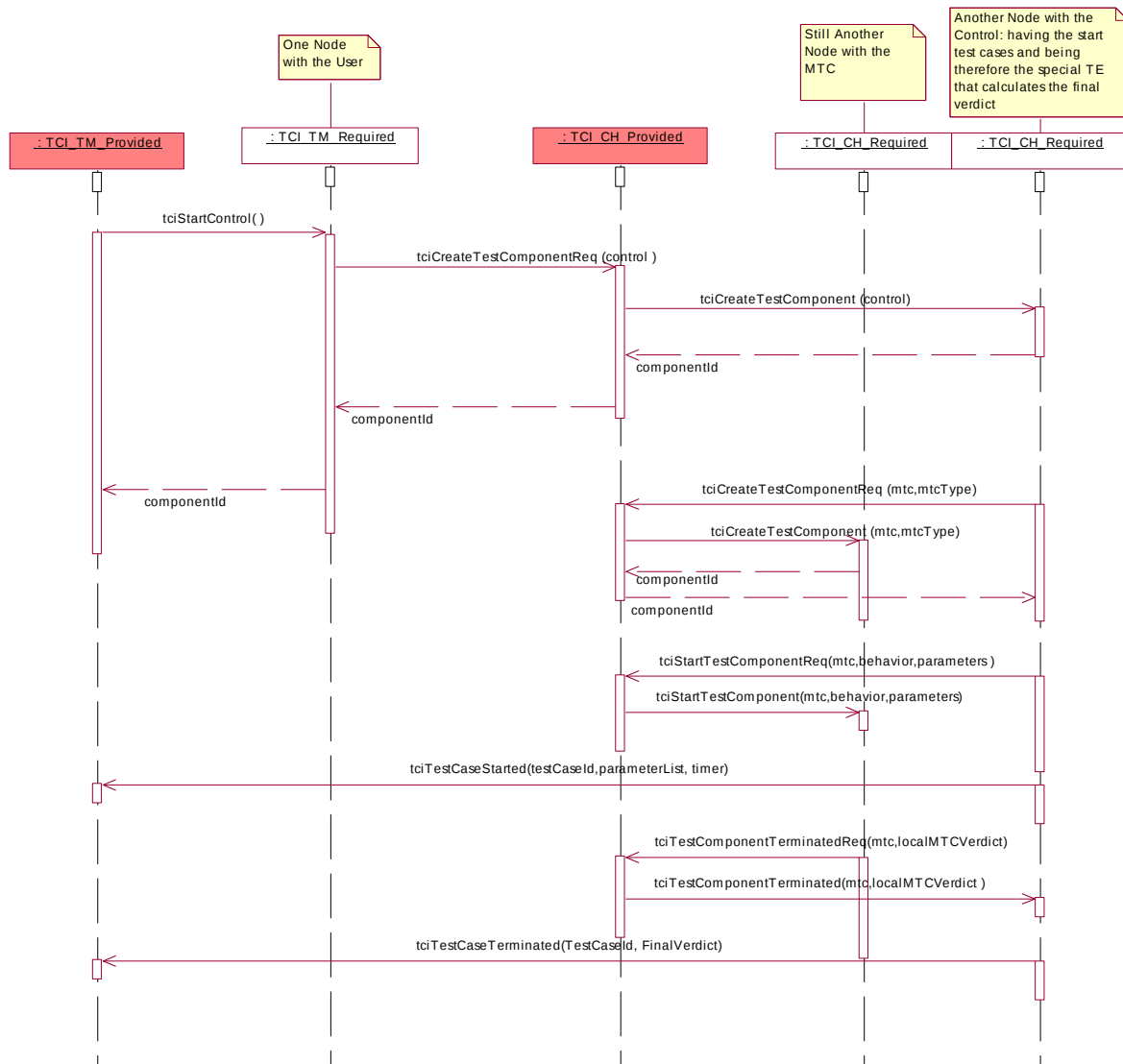


Figure 18. Use scenario: component handling for test case execution within control

10.3.5.2 TTCN-3 fragment

```

module AModule {
  ...
  testcase ATestCase(...)... {
    ... //the test case behaviour
  }
  ...
  control {
    ...
    execute(ATestCase(...));
    ...
  }
  ...
}

```

10.3.6 Use scenario: component handling for direct test case execution

The scenario in figure 19 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 shall 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.

10.3.6.1 Sequence diagram

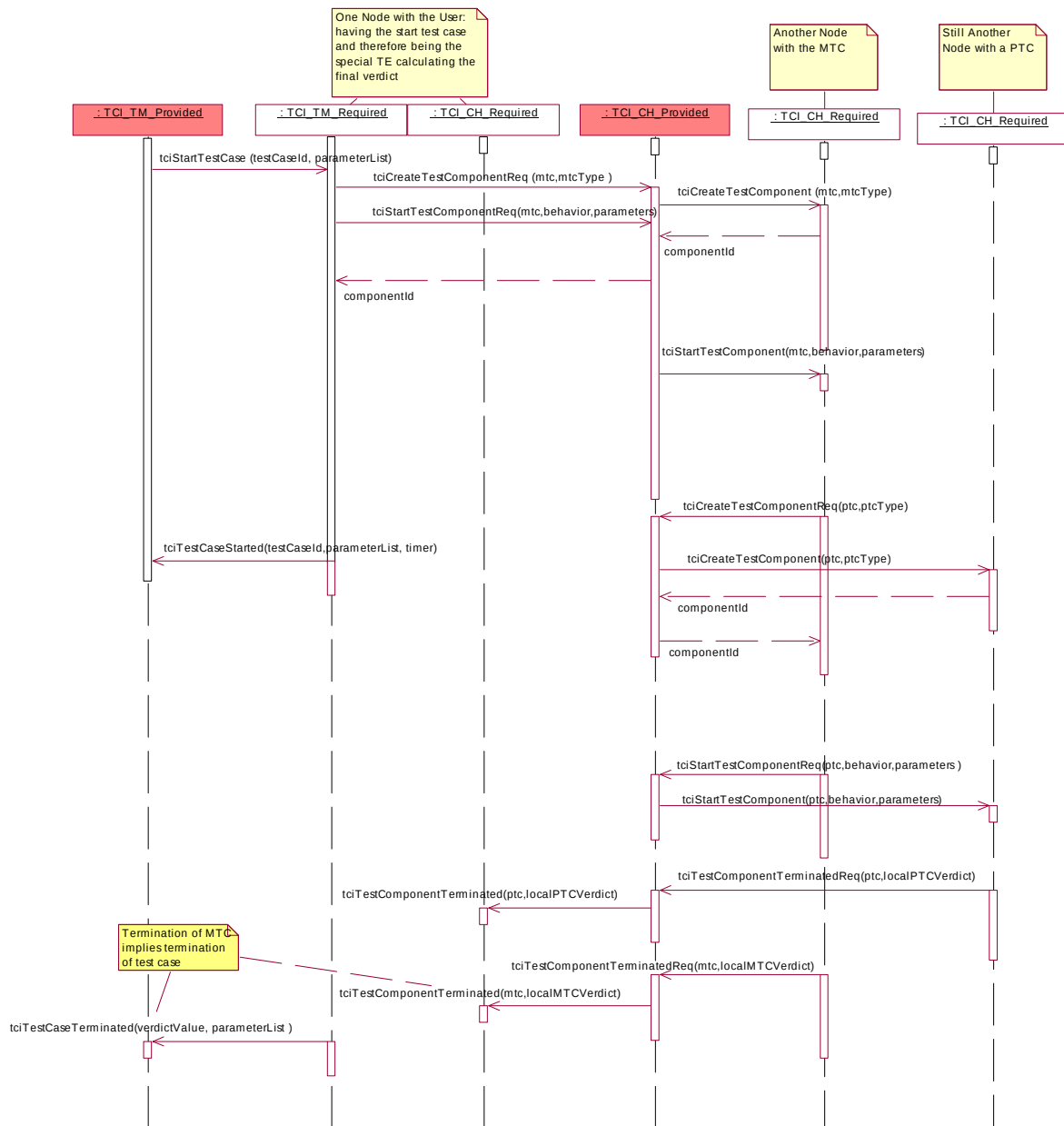


Figure 19: Use scenario: component handling for direct test case execution

10.3.6.2 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(APTCBehaviour(...));
    ...
  }
  ...
}

```

10.3.7 Use scenario: propagation of map/connect

The scenario in figure 20 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.

10.3.7.1 Sequence diagram

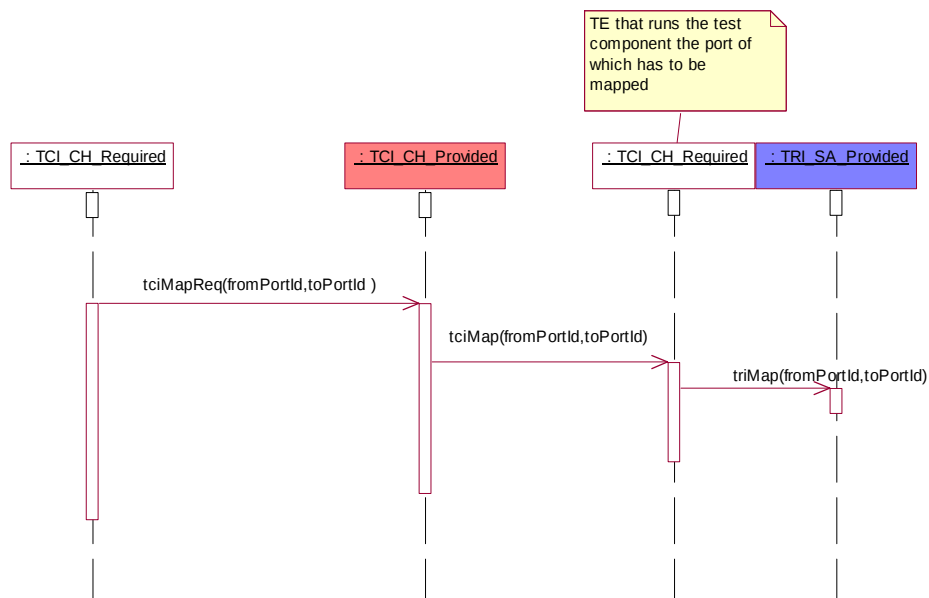


Figure 20: Use scenario: propagation of map

10.3.7.2 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);
    ...
  }
  ...
}

```

10.3.8 Use scenario: propagation of unmap/disconnect

The scenario in figure 21 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.

10.3.8.1 Sequence diagram

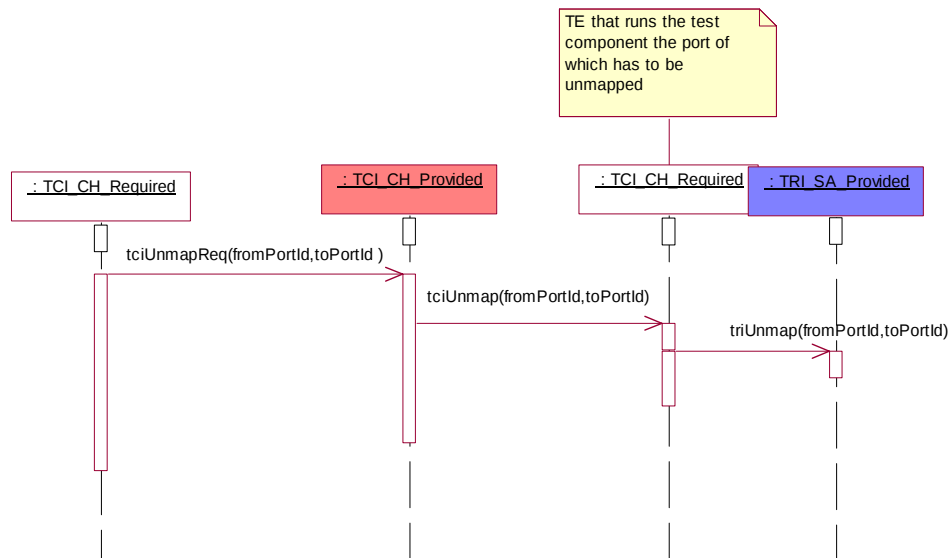


Figure 21: Use scenario - propagation of map

10.3.8.2 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);
    ...
  }
  ...
}

```

10.4 Termination of test cases and control

10.4.1 Use scenario: stop a test case

The scenario in figure 22 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.

10.4.1.1 Sequence diagram

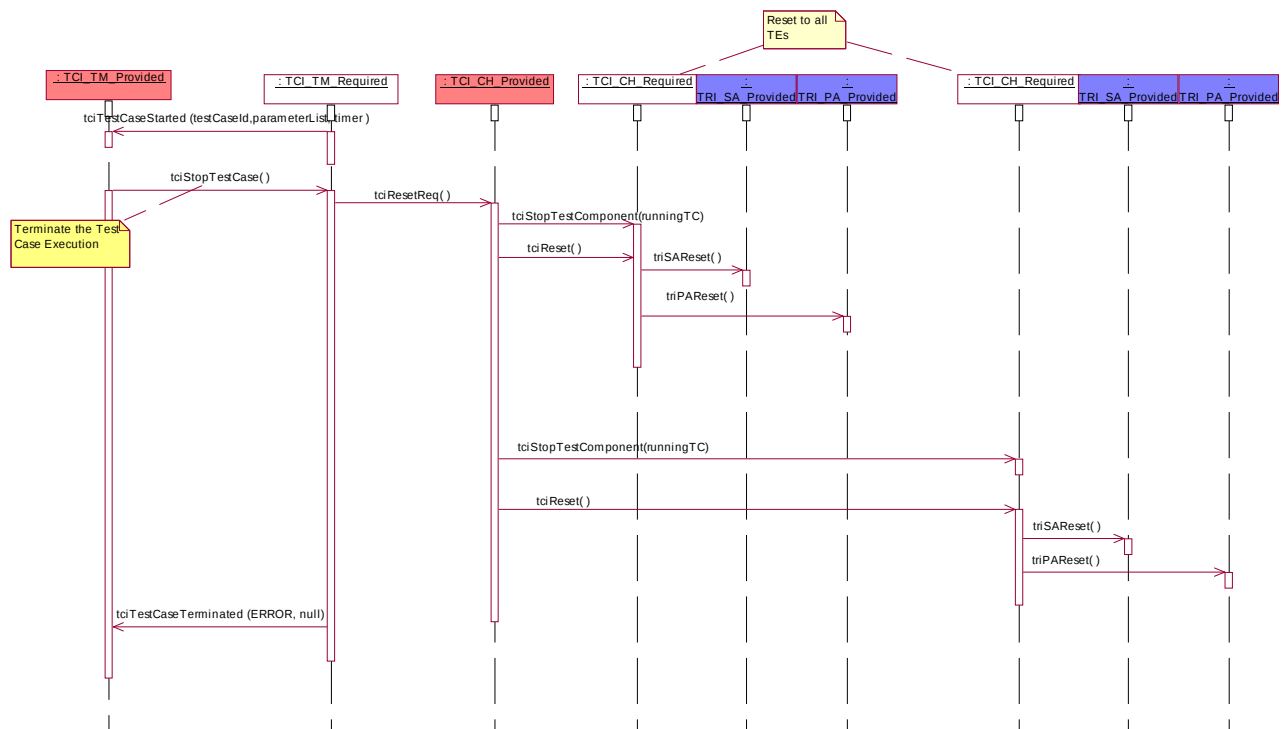


Figure 22: Use scenario: stop a test case

10.4.1.2 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.

10.4.2 Use scenario: stop control

The scenario in figure 23 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 shall be stopped. Furthermore, the test system shall be reset as described in figure 22.

10.4.2.1 Sequence diagram

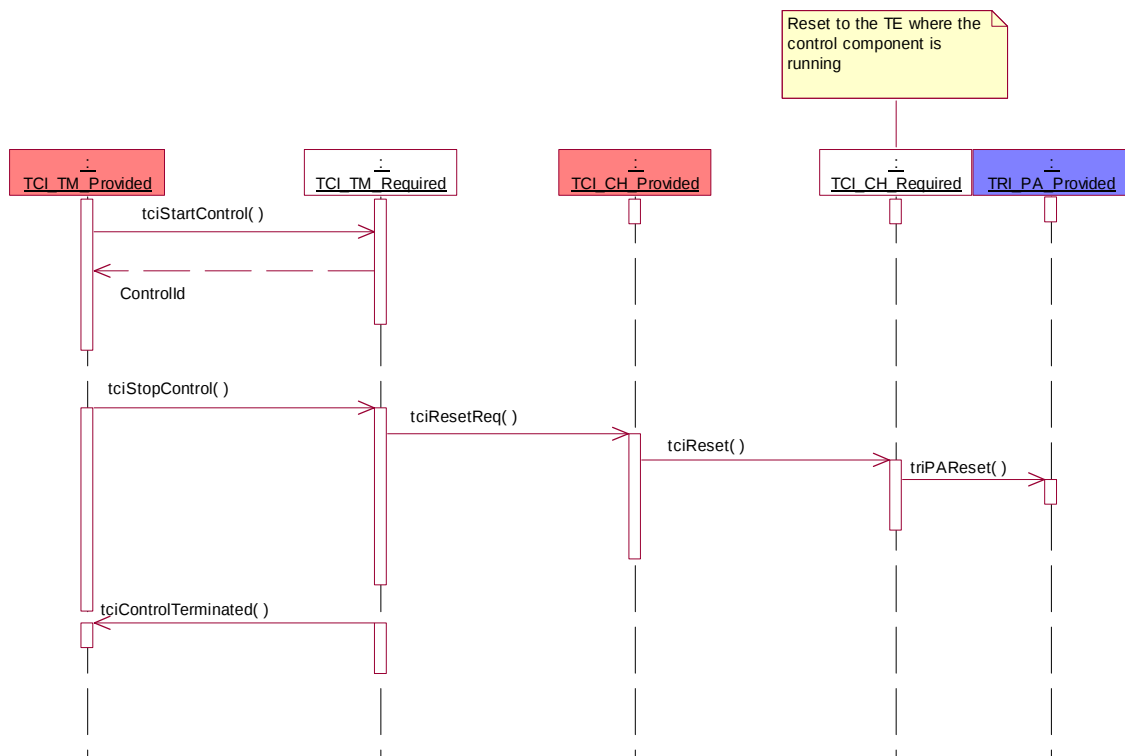


Figure 23: Use scenario - stop control

10.4.2.2 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.

10.4.3 Use scenario: termination of control after error

The scenario in figure 24 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.

10.4.3.1 Sequence diagram

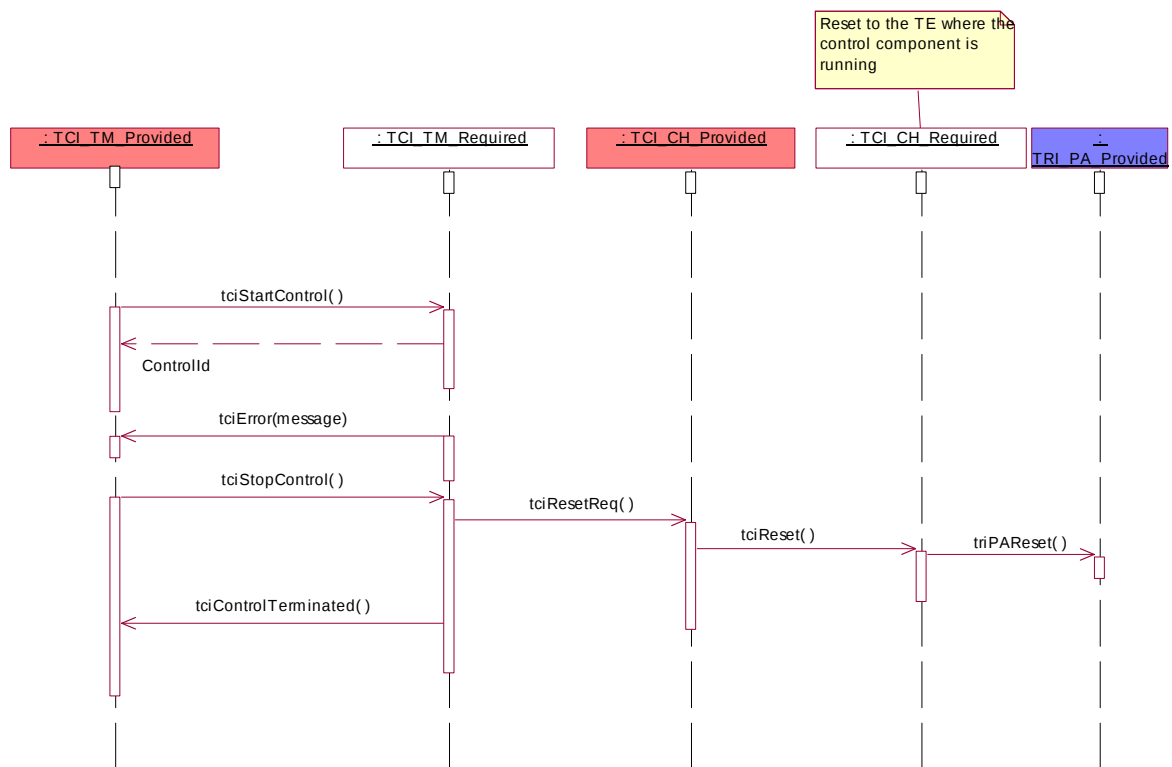


Figure 24: Use scenario - termination of control after error

10.4.3.2 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.

10.4.4 Use scenario: termination of a test case after error

The scenario in figure 25 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 must then explicitly terminate test case execution. Upon stopping a test case, the parallel test components will be stopped and the test system shall be reset.

10.4.4.1 Sequence diagram

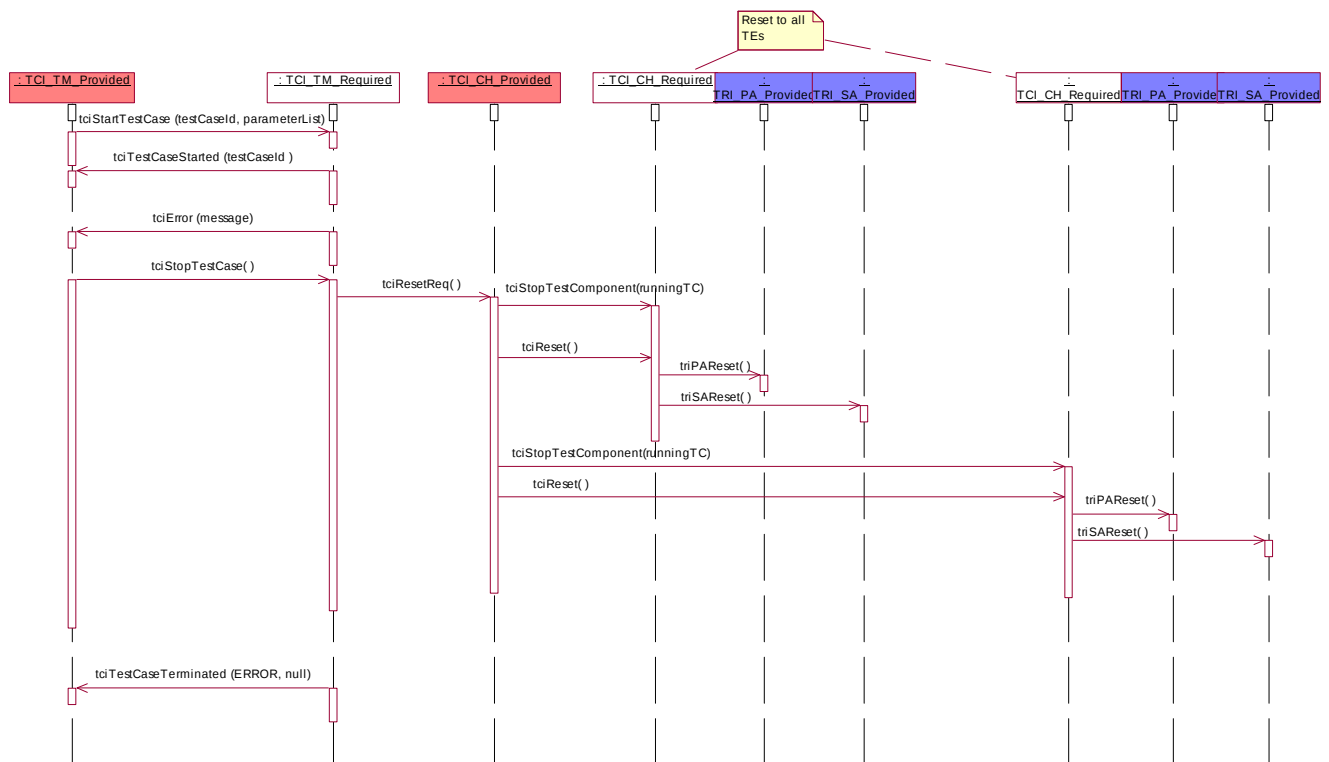


Figure 25: Use scenario - termination of a test case after error

10.4.4.2 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.

10.4.5 Use scenario: reset

The scenario in figure 26 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.

10.4.5.1 Sequence diagram

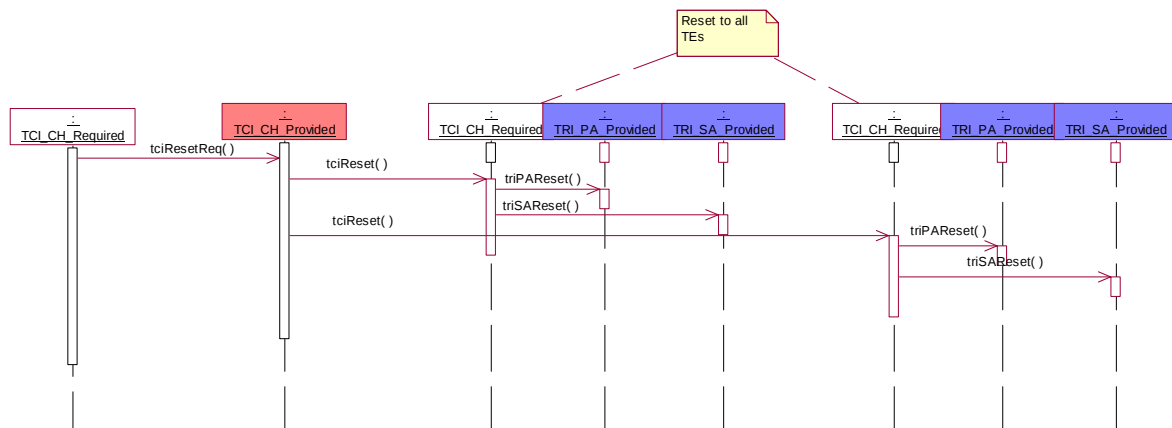


Figure 26: Use scenario - reset

10.4.5.2 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.

10.5 Communication

10.5.1 Use scenario: local intercomponent communication

The scenario in figure 27 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.

10.5.1.1 Sequence diagram

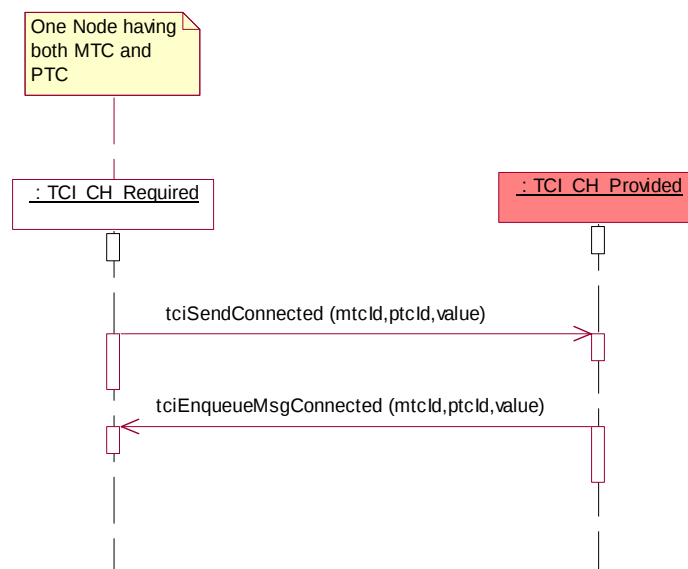


Figure 27: Use scenario - local intercomponent communication

10.5.1.2 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
    ...
  }
  ...
}

```

10.5.2 Use scenario: internode communication between test components

The scenario in figure 28 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.

10.5.2.1 Sequence diagram

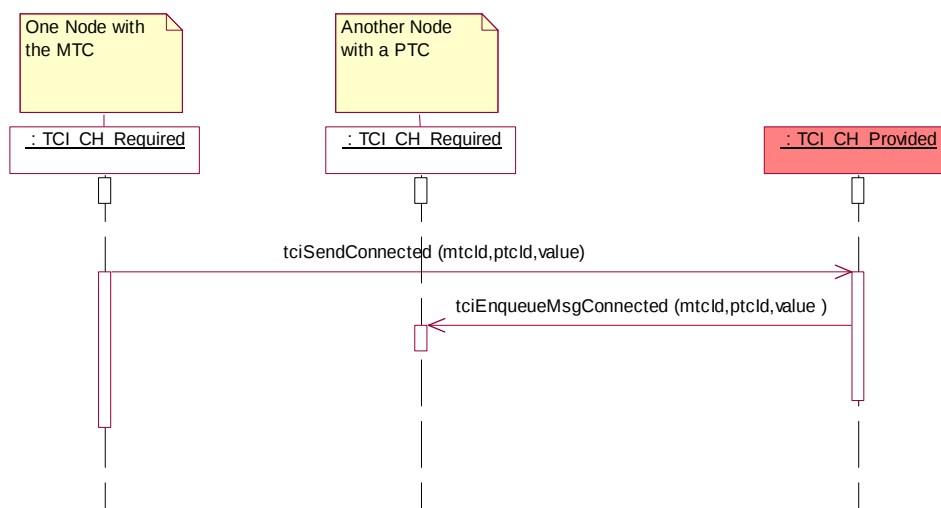


Figure 28: Use scenario - internode communication between test components

10.5.2.2 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
    ...
  }
  ...
}

```

10.5.3 Use scenario: encoding

The scenario in figure 29 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.

10.5.3.1 Sequence diagram

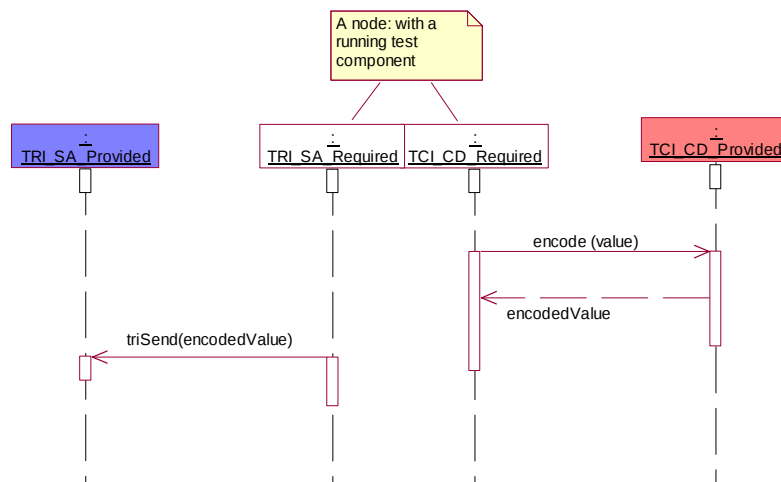


Figure 29: Use scenario - encoding

10.5.3.2 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 = '...' }

```

10.5.4 Use scenario: decoding

The scenario in figure 30 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.

10.5.4.1 Sequence diagram

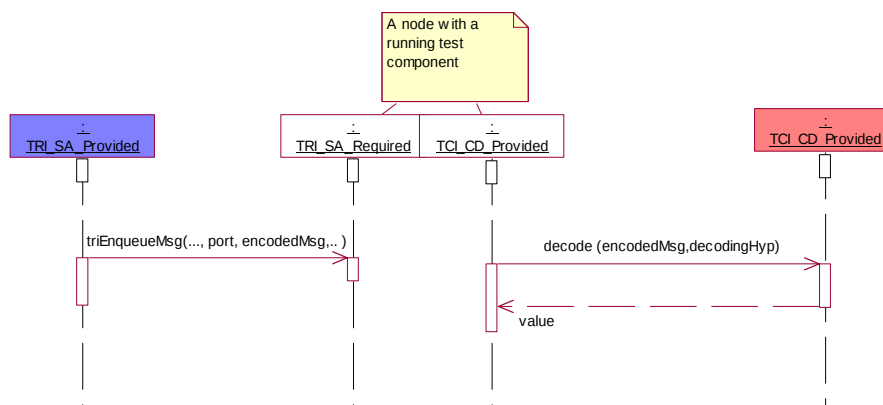


Figure 30: Use scenario - decoding

10.5.4.2 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 = '...' }
```

Annex A (normative): IDL Specification of TCI

A.1 TCI IDL

This annex defines the TTCN-3 Control Interfaces using the Interface Definition Language (IDL 1).

```
// *****
// * 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 ;

    // Communications
    native TriMessageType;

    // *****
    // * General Abstract Data Types
    // *****

    // Basic definitions
    native TBoolean;
    native TFloat;
    native TChar;
    native TInteger;
    native TString;
    native TUniversalChar;
    typedef sequence <TString> TStringSeq;
    native TObjid;

    struct QualifiedName {
        TString moduleName;
        TString baseName;
    };

    // General TCI abstract data types
    typedef QualifiedName TciBehaviourIdType;
    typedef QualifiedName TciSignatureIdType;
    typedef QualifiedName TciModuleIdType;
    typedef QualifiedName TciModuleParameterIdType;
    typedef QualifiedName TciTestCaseIdType;

    enum TciParameterPassingModeType {
        IN_MODE,
        OUT_MODE,
        INOUT_MODE
    };

    struct TciParameterType {
        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 <TciModuleIdType> TciModuleIdListType ;

typedef sequence <TciModuleParameterType> TciModuleParameterListType;

typedef sequence <TciTestCaseIdType> TciTestCaseIdListType;

enum TciTestComponentKindType {
    MTC,
    PTC,
    CONTROL
};

enum TciTypeClassType {
    ADDRESS_CLASS,
    ANYTYPE_CLASS,
    BITSTRING_CLASS,
    BOOLEAN_CLASS,
    CHAR_CLASS,
    CHARSTRING_CLASS,
    COMPONENT_CLASS,
    ENUMERATED_CLASS,
    FLOAT_CLASS,
    HEXSTRING_CLASS,
    INTEGER_CLASS,
    OBJID_CLASS,
    OCTETSTRING_CLASS,
    RECORD_CLASS,
    RECORDOF_CLASS,
    SET_CLASS,
    SETOF_CLASS,
    UNION_CLASS,
    UNIVERSALCHAR_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 ();
};

// 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);
};

interface RecordValue : Value {
    Value    getField (in TString fieldName);
    void     setField (
        in TString fieldName,
        in Value value
    );
    TStringSeq getFieldNames ();
};

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 ObjidValue : Value {
    TObjid  getObjid ();
    void    setObjid (in TObjid value);
};

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 ();
};

// *****
// Coding Decoding Interface
// - Required
// *****

interface TCI_CD_Required {
    Type getTypeForName (in TString typeName);
    Type getInteger ();
    Type getFloat ();
    Type getBoolean ();
    Type getChar ();
    Type getUniversalChar ();
    Type getObjid ();
    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 getImportedModules();
    TciModuleParameterListType tciGetModuleParameters (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 TciSignatureIdType signature,
        in TciParameterListType parameterList
    );
    void tciEnqueueReplyConnected (
        in TriPortIdType sender,
        in TriComponentIdType receiver,
        in TciSignatureIdType signature,
        in TciParameterListType parameterList,
        in Value returnValue
    );
    void tciEnqueueRaiseConnected (
        in TriPortIdType sender,
        in TriComponentIdType receiver,
        in TciSignatureIdType signature,
        in Value except
    );
    TriComponentIdType tciCreateTestComponent (
        in TciTestComponentKindType kind,
        in Type componentType
    );
    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 component
    );
    void tciReset ();
};

// *****
// Component Handling Interface
// - Provided
// *****

```

```

interface TCI_CH_Provided {
    void tciSendConnected (
        in TriPortIdType sender,
        in TriComponentIdType receiver,
        in Value sendMessage
    );
    void tciCallConnected (
        in TriPortIdType sender,
        in TriComponentIdType receiver,
        in TciSignatureIdType signature,
        in TciParameterListType parameterList
    );
    void tciReplyConnected (
        in TriPortIdType sender,
        in TriComponentIdType receiver,
        in TciSignatureIdType signature,
        in TciParameterListType parameterList,
        in Value returnValue
    );
    void tciRaiseConnected (
        in TriPortIdType sender,
        in TriComponentIdType receiver,
        in TciSignatureIdType signature,
        in Value except
    );
    TriComponentIdType tciCreateTestComponentReq (
        in TciTestComponentKindType kind,
        in Type componentType
    );
    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 component
    );
};

};

```

Annex B (informative): Bibliography

- [GCI] INTOOL CGI/NPL038 (V2.2): "Generic Compiler/Interpreter interface; GCI Interface Specification" Infrastructural Tools, December 1996.
- [TTCN-2] ISO/IEC 9646-3: 1997 (E): "Information technology - Open Systems Interconnection - Conformance testing methodology and framework - Part 3: The Tree and Tabular combined Notation (TTCN)".
- [TTCN-2++] ISO/IEC 9646-3 (1998): "Information technology - Open Systems Interconnection - Conformance testing methodology and framework - Part 3: The Tree and Tabular combined Notation (TTCN)".
- ISO/IEC 10646-1: Information technology - Universal Multiple-Octet Coded Character Set (UCS) - Part 1: Architecture and Basic Multilingual Plane

History

Document history		
V1.1.1	May 2003	Membership Approval Procedure MV 20030725: 2003-05-27 to 2003-07-25