

**Methods for Testing and Specification (MTS);
The Testing and Test Control Notation version 3;
Part 5: TTCN-3 Runtime Interface (TRI)**



Reference

RES/MTS-00107-5 T3 ed411 TRI

Keywords

interface, MTS, testing, TRI, TTCN

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, please send your comment to one of the following services:

http://portal.etsi.org/chaicor/ETSI_support.asp

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 2009.
All rights reserved.

DECTTM, **PLUGTESTS**TM, **UMTS**TM, **TIPHON**TM, the TIPHON logo and the ETSI logo are Trade Marks of ETSI registered for the benefit of its Members.

3GPPTM is a Trade Mark of ETSI registered for the benefit of its Members and of the 3GPP Organizational Partners.

LTETM is a Trade Mark of ETSI currently being registered

for the benefit of its Members and of the 3GPP Organizational Partners.

GSM[®] and the GSM logo are Trade Marks registered and owned by the GSM Association.

Contents

Intellectual Property Rights	7
Foreword.....	7
Introduction	7
1 Scope	8
1.1 Compliance.....	8
2 References	8
2.1 Normative references	8
2.2 Informative references.....	9
3 Definitions and abbreviations.....	9
3.1 Definitions.....	9
3.2 Abbreviations	10
4 General Structure of a TTCN-3 Test System	10
4.1 Entities in a TTCN-3 Test System.....	11
4.1.1 Test Management and Control (TMC).....	11
4.1.1.1 Test Management (TM)	11
4.1.1.2 Test Logging (TL).....	11
4.1.1.3 Coding and Decoding (CD)	11
4.1.1.4 Component Handling (CH)	11
4.1.2 TTCN-3 Executable (TE)	11
4.1.2.1 Executable Test Suite (ETS)	12
4.1.2.2 TTCN-3 Runtime System (T3RTS)	12
4.1.2.3 Encoding/Decoding System (EDS).....	12
4.1.2.4 Timers in the TTCN-3 Executable	12
4.1.3 SUT Adaptor (SA).....	13
4.1.4 Platform Adaptor (PA).....	13
4.2 Interfaces in a TTCN-3 Test System	13
4.3 Execution requirements for a TTCN-3 test system	13
5 TTCN-3 Runtime Interface and operations.....	14
5.1 Overview of the TRI.....	14
5.1.1 The triCommunication Interface.....	14
5.1.2 The triPlatform Interface	14
5.1.3 Correlation between TTCN-3 and TRI Operation Invocations.....	14
5.2 Error handling	15
5.3 Data interface	15
5.3.1 Connection	16
5.3.2 Communication.....	16
5.3.3 Timer	17
5.3.4 Miscellaneous	17
5.4 Operation descriptions.....	17
5.5 Communication interface operations.....	18
5.5.1 triSAReset (TE → SA)	18
5.5.2 Connection handling operations	18
5.5.2.1 triExecuteTestCase (TE → SA)	18
5.5.2.2 triMap (TE → SA)	18
5.5.2.3 triUnmap (TE → SA).....	19
5.5.2.4 triEndTestCase (TE → SA).....	19
5.5.3 Message based communication operations	19
5.5.3.1 triSend (TE → SA).....	19
5.5.3.2 triSendBC (TE → SA)	20
5.5.3.3 triSendMC (TE → SA)	20
5.5.3.4 triEnqueueMsg (SA → TE).....	21
5.5.4 Procedure based communication operations.....	21

5.5.4.1	triCall (TE → SA).....	21
5.5.4.2	triCallBC (TE → SA).....	22
5.5.4.3	triCallMC (TE → SA).....	23
5.5.4.4	triReply (TE → SA).....	24
5.5.4.5	triReplyBC (TE → SA).....	25
5.5.4.6	triReplyMC (TE → SA).....	26
5.5.4.7	triRaise (TE → SA).....	26
5.5.4.8	triRaiseBC (TE → SA).....	27
5.5.4.9	triRaiseMC (TE → SA).....	27
5.5.4.10	triQueueCall (SA → TE).....	28
5.5.4.11	triQueueReply (SA → TE).....	28
5.5.4.12	triQueueException (SA → TE).....	29
5.5.5	Miscellaneous operations.....	29
5.5.5.1	triSUTactionInformal (TE → SA).....	29
5.5.5.2	triSUTactionTemplate (TE → SA).....	29
5.6	Platform interface operations.....	29
5.6.1	triPAReset (TE → PA).....	29
5.6.2	Timer operations.....	30
5.6.2.1	triStartTimer (TE → PA).....	30
5.6.2.2	triStopTimer (TE → PA).....	30
5.6.2.3	triReadTimer (TE → PA).....	30
5.6.2.4	triTimerRunning (TE → PA).....	31
5.6.2.5	triTimeout (PA → TE).....	31
5.6.3	Miscellaneous operations.....	32
5.6.3.1	triExternalFunction (TE → PA).....	32
6	Java language mapping.....	32
6.1	Introduction.....	32
6.2	Names and scopes.....	32
6.2.1	Names.....	32
6.2.2	Scopes.....	33
6.3	Type mapping.....	33
6.3.1	Basic type mapping.....	33
6.3.1.1	Boolean.....	33
6.3.1.1.1	Methods.....	33
6.3.1.2	String.....	33
6.3.2	Structured type mapping.....	33
6.3.2.1	TriPortIdType.....	34
6.3.2.1.1	Methods.....	34
6.3.2.2	TriPortIdListType.....	34
6.3.2.2.1	Methods.....	34
6.3.2.3	TriComponentIdType.....	35
6.3.2.3.1	Methods.....	35
6.3.2.4	TriComponentIdListType.....	35
6.3.2.4.1	Methods.....	35
6.3.2.5	TriMessageType.....	36
6.3.2.5.1	Methods.....	36
6.3.2.6	TriAddressType.....	36
6.3.2.6.1	Methods.....	36
6.3.2.7	TriAddressListType.....	37
6.3.2.7.1	Methods.....	37
6.3.2.8	TriSignatureIdType.....	37
6.3.2.8.1	Methods.....	37
6.3.2.9	TriParameterType.....	38
6.3.2.9.1	Methods.....	38
6.3.2.10	TriParameterPassingModeType.....	38
6.3.2.10.1	Constants.....	38
6.3.2.11	TriParameterListType.....	39
6.3.2.11.1	Methods.....	39
6.3.2.12	TriExceptionType.....	39
6.3.2.12.1	Methods.....	39

6.3.2.13	TriTimerIdType	40
6.3.2.13.1	Methods	40
6.3.2.14	TriTimerDurationType	40
6.3.2.14.1	Methods	40
6.3.2.15	TriFunctionIdType	40
6.3.2.15.1	Methods	40
6.3.2.16	TriTestCaseIdType	41
6.3.2.16.1	Methods	41
6.3.2.17	TriActionTemplateType	41
6.3.2.18	TriStatusType	41
6.3.2.18.1	Methods	41
6.4	Constants	42
6.5	Mapping of interfaces	42
6.5.1	Out and InOut Parameter Passing Mode	42
6.5.2	triCommunication - Interface	42
6.5.2.1	triCommunicationSA	43
6.5.2.2	triCommunicationTE	44
6.5.3	triPlatform - Interface	44
6.5.3.1	TriPlatformPA	44
6.5.3.2	TriPlatformTE	45
6.6	Optional parameters	45
6.7	TRI initialization	45
6.8	Error handling	45
7	ANSI C language mapping	45
7.1	Introduction	45
7.2	Names and scopes	45
7.2.1	Abstract type mapping	46
7.2.2	ANSI C type definitions	47
7.2.3	IDL type mapping	47
7.2.4	TRI operation mapping	47
7.3	Memory management	49
7.4	Error handling	49
8	C++ language mapping	49
8.1	Introduction	49
8.2	Names and scopes	49
8.3	Memory management	49
8.4	Error handling	50
8.5	Type mapping	50
8.5.1	Encapsulated C++ types	50
8.5.2	Abstract data types	50
8.5.2.1	QualifiedName	50
8.5.2.1.1	Methods	50
8.5.2.2	TriAddress	51
8.5.2.2.1	Methods	51
8.5.2.3	TriAddressList	52
8.5.2.3.1	Methods	52
8.5.2.4	TriComponentId	53
8.5.2.4.1	Methods	53
8.5.2.5	TriComponentIdList	54
8.5.2.5.1	Methods	54
8.5.2.6	TriException	54
8.5.2.6.1	Methods	55
8.5.2.7	TriFunctionId	55
8.5.2.7.1	Methods	55
8.5.2.8	TriMessage	55
8.5.2.8.1	Methods	56
8.5.2.9	TriParameter	56
8.5.2.9.1	Methods	56
8.5.2.10	TriParameterList	57
8.5.2.10.1	Methods	57

8.5.2.11	TriParameterPassingMode	57
8.5.2.11.1	Methods	58
8.5.2.11.2	Constants	58
8.5.2.12	TriPortId.....	58
8.5.2.12.1	Methods	58
8.5.2.13	TriPortIdList.....	59
8.5.2.13.1	Methods	59
8.5.2.14	TriSignatureId	60
8.5.2.14.1	Methods	60
8.5.2.15	TriStatus	60
8.5.2.15.1	Methods	60
8.5.2.15.2	Constants	61
8.5.2.16	TriTestCaseId.....	61
8.5.2.16.1	Methods	61
8.5.2.17	TriTimerDuration.....	61
8.5.2.17.1	Methods	61
8.5.2.18	TriTimerId.....	62
8.5.2.18.1	Methods	62
8.6	Mapping of interfaces.....	62
8.6.1	TriCommunicationSA.....	62
8.6.2	TriCommunicationTE.....	64
8.6.3	TriPlatformPA	64
8.6.4	TriPlatformTE.....	65
9	Use scenarios.....	65
9.1	First scenario	66
9.1.1	TTCN-3 fragment	66
9.1.2	Message sequence chart.....	67
9.2	Second scenario.....	68
9.2.1	TTCN-3 fragment	68
9.2.2	Message sequence chart.....	69
9.3	Third scenario.....	70
9.3.1	TTCN-3 fragment	70
9.3.2	Message sequence chart.....	71
Annex A (normative):	IDL Summary	72
Annex B (informative):	Bibliography.....	75
History		76

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>).

Pursuant to the ETSI IPR Policy, no investigation, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Foreword

This ETSI Standard (ES) has been produced by ETSI Technical Committee Methods for Testing and Specification (MTS).

The present document is part 5 of a multi-part deliverable. Full details of the entire series can be found in part 1 [2].

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 Runtime Interface specification.

The first part introduces the decomposition of a TTCN-3 test system into four main entities: Test Management (TM), TTCN-3 Executable (TE), SUT Adaptor (SA), and Platform Adaptor (PA). In addition, the interaction between these entities, i.e. the corresponding interfaces, is defined.

The second part of the present document specifies the TTCN-3 Runtime Interface (TRI). The interface is defined in terms of operations, which are implemented as part of one entity and called by other entities of the test system. 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 this interface specification only defines interactions between the TSI and the SUT as well as timer operations.

1 Scope

The present document provides the specification of the runtime interface for TTCN-3 test system implementations. The TTCN-3 Runtime Interface provides a standardized adaptation for timing and communication of a test system to a particular processing platform and the system under test, respectively. The present document defines the interface as a set of operations independent of target language.

The interface is defined to be compatible with the TTCN-3 standard (see ES 201 873-1 [2]). The present document uses the CORBA Interface Definition Language (IDL) to specify the TRI completely. Clauses 6, 7, and 8 present language mappings for this abstract specification to the target languages Java, ANSI C, and C++. A summary of the IDL based interface specification is provided in the annex.

1.1 Compliance

The requirement for a TTCN-3 test system to be TRI compliant is to adhere to the interface specification stated in the present document as well as to one of the target language mappings included.

EXAMPLE: If a vendor supports Java, the TRI 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.

2 References

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.
- Non-specific reference may be made only to a complete document or a part thereof and only in the following cases:
 - if it is accepted that it will be possible to use all future changes of the referenced document for the purposes of the referring document;
 - for informative references.

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

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

2.1 Normative references

The following referenced documents are indispensable for the application of the present document. For dated references, only the edition cited applies. For non-specific references, the latest edition of the referenced document (including any amendments) applies.

- [1] ISO/IEC 9646-1: "Information technology - Open Systems Interconnection - Conformance testing methodology and framework - Part 1: General concepts".
- [2] 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".
- [3] 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".

2.2 Informative references

The following referenced documents are not essential to the use of the present document but they assist the user with regard to a particular subject area. For non-specific references, the latest version of the referenced document (including any amendments) applies.

Not applicable.

3 Definitions and abbreviations

3.1 Definitions

For the purposes of the present document, the terms and definitions given in ES 201 873-1 [2] and the following apply:

Abstract Test Suite (ATS): See ISO/IEC 9646-1 [1].

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.

Executable Test Suite (ETS): See ISO/IEC 9646-1 [1].

explicit timer: timer that is declared in a TTCN-3 ATS and that can be accessed through TTCN-3 timer operations

Implementation eXtra Information for Testing (IXIT): See ISO/IEC 9646-1 [1].

implicit timer: system timer that is created by the TTCN-3 Executable to guard a TTCN-3 call or execute operation

NOTE: Implicit timers are not accessible to the TTCN-3 user.

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 external functions as well as explicit and implicit timers.

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

System Under Test (SUT): See ISO/IEC 9646-1 [1].

NOTE: All types are known at compile time, i.e. are statically bound.

test case: See ISO/IEC 9646-1 [1].

test event: either sent or received test data (message or procedure call) on a communication port that is part of the test system interface

Test Management (TM): entity that provides a user interface and administers the TTCN-3 test system

test system: See ISO/IEC 9646-1 [1].

test system interface: 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

Timer IDentification (TID): unique identification for explicit or implicit timer instances that is generated by the TTCN-3 Executable

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

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

TTCN-3 Runtime Interface (TRI): two interfaces that defines the interaction of the TTCN-3 Executable between the SUT and the Platform Adaptor (PA) and the System Adaptor (SA) in a test system

3.2 Abbreviations

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

ATS	Abstract Test Suite
CD	(External) Coding/Decoding
CH	Component Handling
CORBA	Common Object Request Broker Architecture
EDS	(Internal) Encoding/Decoding System
ETS	Executable Test Suite
FIFO	First-In-First-Out (Scheduling Discipline)
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
SA	SUT Adaptor
STL	Standard Template Library of C++
SUT	System Under Test
T3RTS	TTCN-3 RunTime System
TCI	TTCN-3 Control Interfaces
TE	TTCN-3 Executable
TID	Timer IDentification
TL	Test Logging
TM	Test Management
TMC	Test Management and Control
TRI	TTCN-3 Runtime Interfaces
TSI	Test System Interface
TTCN	Testing and Test Control Notation
TTCN-3	Tree and Tabular Combined Notation version 3

4 General Structure of a TTCN-3 Test System

A TTCN-3 test system can be thought of conceptually as a set of interacting entities where each entity corresponds to a particular aspect of functionality in a test system implementation. These entities manage test execution, interpreting or executing compiled TTCN-3 code, realize proper communication with the SUT, implement external functions, and handle timer operations.

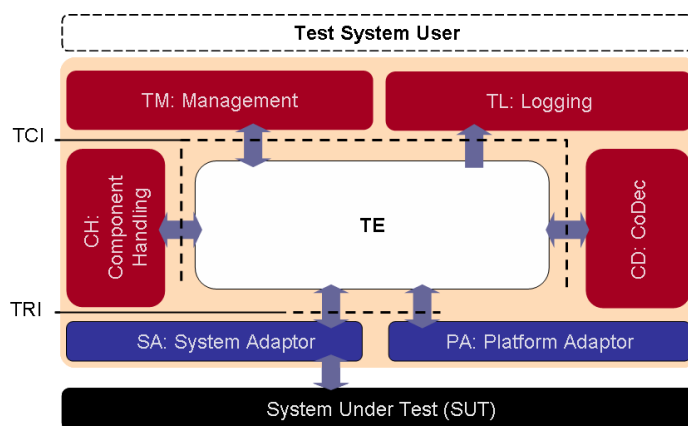


Figure 1: General Structure of a TTCN-3 Test System

4.1 Entities in a TTCN-3 Test System

The structure of a TTCN-3 test system implementation is illustrated in figure 1. It should be noted that the further refinement of TM into smaller entities, as shown in figure 1 and used in the following clauses of the present document, is purely an aid to define TTCN-3 test system interfaces.

The part of the test system that deals with interpretation and execution of TTCN-3 modules, i.e. the Executable Test Suite (ETS), is part of the TTCN-3 Executable (TE). This corresponds either to the executable code produced by a TTCN-3 compiler or a TTCN-3 interpreter in a test system implementation. It is assumed that a test system implementation includes the ETS as derived from a TTCN-3 ATS.

The remaining part of the TTCN-3 test system, which deals with any aspects that cannot be concluded from information being present in the original ATS alone, can be decomposed into Test Management (TM), SUT Adaptor (SA), and Platform Adaptor (PA) entities. In general, these entities cover a test system user interface, test execution control, test event logging, as well as communication with the SUT and timer implementation.

4.1.1 Test Management and Control (TMC)

In the TMC entity, we can distinguish between functionality related to management of test execution, test event logging, external coding and decoding, and component handling.

4.1.1.1 Test Management (TM)

The TM entity is responsible for overall management of the 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 and/or IXIT information to the TE if necessary. Typically, this entity would also implement a test system user interface.

4.1.1.2 Test Logging (TL)

The TL entity is responsible for maintaining the test log. It is explicitly notified to log test events by the TE. The TL entity has a unidirectional interface where any entity part of the TE may post a logging request to the TL entity. A TM internal interface may also be used to record test management information generated by the TE.

4.1.1.3 Coding and Decoding (CD)

The CD entity is optionally responsible for the external encoding and decoding data associated with message based or procedure based communication within the TE. The external codecs can be used in parallel with, or instead of, the built-in codecs associated with the TE. Unlike the built-in codecs the external codecs have a standardized interface which makes them portable between different TTCN-3 systems and tools.

4.1.1.4 Component Handling (CH)

The CH entity is responsible for distributing parallel test components. This distribution might be across one or many physical systems. The CH entity allows the test management to create and control distributed test systems in a manner which is transparent and independent from the TE.

4.1.2 TTCN-3 Executable (TE)

The TE entity is responsible for the interpretation or execution of the TTCN-3 ATS. Conceptually, the TE can be decomposed into three interacting entities: an ETS, TTCN-3 Runtime System (T3RTS), and an optional internal Encoding/Decoding System (EDS) entity. Note that this refinement of the TE into smaller entities is purely a conceptual aid to define TTCN-3 test system interfaces - there is no requirement for this distinction to be reflected in TRI implementations.

The following clauses define the responsibilities of each entity and also discuss the handling of timers in the TRI.

4.1.2.1 Executable Test Suite (ETS)

The ETS entity handles the execution or interpretation of test cases, the sequencing and matching of test events, as defined in the corresponding TTCN-3 modules ES 201 873-1 [2]. It interacts with the T3RTS entity to send, attempt to receive (or match), and log test events during test case execution, to create and remove TTCN-3 test components, as well as to handle external function calls, action operations, and timers. Note that the ETS entity does not directly interact with the SA via the TRI.

4.1.2.2 TTCN-3 Runtime System (T3RTS)

The T3RTS entity interacts with the TM, SA and PA entities via TCI and TRI, and manages ETS and EDS entities. The T3RTS initializes adaptors as well as ETS and EDS entities. This entity performs all the actions necessary to properly start the execution of a test case or function with parameters in the ETS entity. It queries the TM entity for module parameter values required by the ETS and sends logging information to it. It also collects and resolves associated verdicts returned by the ETS entity as defined in ES 201 873-1 [2].

The T3RTS entity implements the creation and removal of TTCN-3 test components, as well as the TTCN-3 semantics of message and procedure based communication, external function calls, action operations and timers. This includes notifying the SUT Adaptor (SA) which message or procedure call is to be sent to the SUT, or the Platform Adaptor (PA) which external function is to be executed or which timers are to be started, stopped, queried, or read. Similarly, the T3RTS notifies the ETS entity of incoming messages or procedure calls from the SUT as well as timeout events.

Prior to sending or receiving messages and procedure calls to or from the SA, or handling function calls and action operations in the PA for the ETS entity, the T3RTS invokes the EDS entity for their encoding or decoding. The T3RTS entity should implement all message and procedure based communication operations between test components, but only the TTCN-3 semantics of procedure based communication with the SUT, i.e. the possible blocking and unblocking of test component execution, guarding with implicit timers, and handling of timeout exceptions as a result of such communication operations. All procedure based communication operations with the SUT are to be realized and identified (in the case of a receiving operation) in the SA as they are most efficiently implemented in a platform specific manner. Note that the timing of any procedure call operation, i.e. implicit timers, is implemented in the Platform Adaptor (PA).

The TTCN-3 Executable is required to maintain its own port queues (distinct from those which may be available in the SA or PA) for input test events to perform snapshots for receiving operations as defined in ES 201 873-1 [2]. Timeout events, which are generated by TTCN-3 timer, call timer, or test case timer implementations, are to be kept in a timeout list as defined in ES 201 873-1 [2]. In figure 2, all of this functionality has been assigned to the T3RTS entity. It is responsible to store events that the SA or PA has notified the TE entity of, but which have yet to be processed.

4.1.2.3 Encoding/Decoding System (EDS)

The EDS entity is responsible for the internal encoding and decoding of test data, which includes data used in communication operations with the SUT, as specified in the executing TTCN-3 module. If no encoding has been specified for a TTCN-3 module the encoding of data values is tool specific. This entity is invoked by and returns to the T3RTS entity. Note that the EDS entity does not directly interact with the SA via the TRI.

4.1.2.4 Timers in the TTCN-3 Executable

Timers that have been declared and named in the TTCN-3 ATS can be conceptually classified as explicit in the TE. Timers that are created by the TE for guarding TTCN-3 procedure calls or execute operations are known in the TE as implicit timers. Explicit as well as implicit timers are both created within the TE but implemented by the Platform Adaptor (PA). This is achieved by generating a unique Timer IDentification (TID) for any timer created in the TE. This unique TID should enable the TE to differentiate between different timers. The TID is to be used by the TE to interact with corresponding timer implementation in the PA.

Note that it is the responsibility of the TE to implement the different TTCN-3 semantics for explicit and implicit timers correctly as defined in ES 201 873-1 [2], e.g. the use of keywords `any` and `all` with timers only applies to explicit timers. In the PA all timers, i.e. implicit and explicit, are treated in the same manner.

4.1.3 SUT Adaptor (SA)

The SA adapts message and procedure based communication of the TTCN-3 test system with the SUT to the particular execution platform of the test system. It is aware of the mapping of the TTCN-3 test component communication ports to test system interface ports and implements the real test system interface as defined in ES 201 873-1 [2]. It is responsible to propagate send requests and SUT action operations from the TTCN-3 Executable (TE) to the SUT, and to notify the TE of any received test events by appending them to the port queues of the TE.

Procedure based communication operations with the SUT are implemented in the SA. The SA is responsible for distinguishing between the different messages within procedure-based communication (i.e. call, reply, and exception) and to propagate them in the appropriate manner either to the SUT or the TE. 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.

The SA has an interface with the TE, which is used to send SUT messages (issued in TTCN-3 SUT action operations) to the SA and to exchange encoded test data between the two entities in communication operations with the SUT.

4.1.4 Platform Adaptor (PA)

The PA implements TTCN-3 external functions and provides a TTCN-3 test system with a single notion of time. In this entity, external functions are to be implemented as well as all timers. Notice that timer instances are created in the TE. A timer in the PA can only be distinguished by its Timer IDentification (TID). Therefore, the PA treats both explicit and implicit timers in the same manner.

The interface with the TE enables the invocation of external functions and the starting, reading, and stopping of timers as well as the inquiring of the status of timers using their timer ID. The PA notifies the TE of expired timers.

4.2 Interfaces in a TTCN-3 Test System

As previously depicted in figure 1, a TTCN-3 test system has two interface sets, the TTCN-3 Control Interface (TCI) and the TTCN-3 Runtime Interface (TRI), which specify the interface between Test Management (TM), Test Logging (TL), Component Handling (CH), Encoding/Decoding (CD) and TTCN-3 Executable (TE) entities, and the TE, SUT Adaptor (SA) and Platform Adaptor (PA) entities, respectively.

The present document defines the TRI. The interaction of the TE with SA and PA are defined in terms of TRI operations.

4.3 Execution requirements for a TTCN-3 test system

Each TRI operation call shall be treated as an atomic operation in the calling entity. The called entity, which implements a TRI 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. Nevertheless, the called entity shall block after the invocation of an external function implementation and wait for its return value. Notice that depending on the test system implementation failure to return from an external function implementation may result in the infinite blocking of test component execution, the TTCN-3 executable, the Platform Adaptor, or even of the entire test system.

The execution requirements stated above can be realized in a tightly integrated test system implementation. Here, the entire TTCN-3 test system is implemented in a single executable or process where each test system entity is assigned at least one thread of execution. TRI operations can be implemented here as procedure calls.

Note that a looser integration of a test system implementation is still possible, e.g. an implementation of a TTCN-3 test system with multiple SUT Adaptors in a distributed computing environment. In this case only a small part of the SUT Adaptor is tightly integrated with the remainder of the TTCN-3 test system whereas actual SA Adaptors may be realized in separate processes. That small part of SA may then only implement a routing of information provided by TRI operations to the desired SUT Adaptor processes, possibly being executed on remote hosts, and vice versa.

5 TTCN-3 Runtime Interface and operations

This clause defines TRI operations in terms of when they are to be used and what their effect is intended to be in a TTCN-3 test system implementation. Also a set of abstract data types is defined which is then used for the definition of TRI operations. This definition also includes a more detailed description of the input parameters required for each TRI operation call and its return value.

5.1 Overview of the TRI

The TRI defines the interaction between the TTCN-3 Executable (TE), SUT Adaptor (SA), and Platform Adaptor (PA) entities within a TTCN-3 test system implementation. Conceptually, it provides a means for the TE to send test data to the SUT or manipulate timers, and similarly to notify the TE of received test data and timeouts.

The TRI can be considered to consist of two sub-interfaces, a triCommunication and a triPlatform interface. The triCommunication interface addresses the communication of a TTCN-3 ETS with the SUT, which is implemented in the SA. The triPlatform interface represents a set of operations, which adapt an ETS to a particular execution platform.

Table 1: Interface Overview

Interface	Direction (calling entity → called entity)	
Name	TE → SA or PA	SA or PA → TE
triCommunication	TE → SA	SA → TE
triPlatform	TE → PA	PA → TE

Both interfaces are bi-directional so that calling and called parts reside in the TE, SA, and PA entities of the test system. Table 1 shows in more detail the caller/callee relationship between the respective entities. Notice that this table only shows interactions visible at the TRI. Internal communication between parts of the same entity is not reflected as the internal structure of the TE, SA, or PA may differ in a TTCN-3 test system implementation.

5.1.1 The triCommunication Interface

This interface consists of operations that are necessary to implement the communication of the TTCN-3 ETS with the SUT. It includes operations to initialize the Test System Interface (TSI), establish connections to the SUT, and handle message and procedure based communication with the SUT. In addition, the triCommunication interface offers an operation to reset the SUT Adaptor (SA).

5.1.2 The triPlatform Interface

This interface includes all operations necessary to adapt the TTCN-3 Executable to a particular execution platform. The triPlatform interface offers means to start, stop, read a timer, enquire its status and to add timeout events to the expired timer list. In addition, it offers operations to call TTCN-3 external functions and to reset the Platform Adaptor (PA). Notice that there is no differentiation between explicit and implicit timers required at the triPlatform Interface. Instead each timer shall be addressed uniformly with its Timer Identifier (TID).

5.1.3 Correlation between TTCN-3 and TRI Operation Invocations

For some TTCN-3 operation invocations there exists a direct correlation to one TRI operation invocation (or possibly two in the case of TTCN-3 `execute` and `call` operations), which is shown in table 2. For all other TRI operation invocations there may be no direct correlation.

The shown correlation for TTCN-3 communication operations (i.e. `send`, `call`, `reply`, and `raise`) only holds if these operations are invoked on a test component port, which is mapped to a TSI port. Nevertheless, this correlation holds for all such operation invocations if no system component has been specified for a test case, i.e. only the MTC test component is created for a test case and no other test components.

Table 2: Correlation between TTCN-3 and TRI Operation Invocations (* = if applicable)

TTCN-3 Operation Name	TRI Operation Name	TRI Interface Name
execute	triExecuteTestCase	TriCommunication
	triStartTimer	TriPlatform
	triEndTestCase	TriCommunication
map	triMap	TriCommunication
unmap	triUnmap	TriCommunication
send	triSend (see note 1)	TriCommunication
	triSendBC (see note 2)	
	triSendMC (see note 3)	
call	triCall (see note 1)	TriCommunication
	triCallBC (see note 2)	
	triCallMC (see note 3)	
	triStartTimer*	TriPlatform
reply	triReply (see note 1)	TriCommunication
	triReplyBC (see note 2)	
	triReplyMC (see note 3)	
raise	triRaise (see note 1)	TriCommunication
	triRaiseBC (see note 2)	
	triRaiseMC (see note 3)	
action	triSUTactionInformal	TriCommunication
start (timer)	triStartTimer	TriPlatform
stop (timer)	triStopTimer	TriPlatform
read (timer)	triReadTimer	TriPlatform
running (timer)	triTimerRunning	TriPlatform
TTCN-3 external function	triExternalFunction	TriPlatform
NOTE 1: For unicast communication.		
NOTE 2: For broadcast communication.		
NOTE 3: For multicast communication.		

Note that all of the TRI operations listed in table 2 are used by the TE and that the TE may implement the invocation of these operations differently when evaluating a TTCN snapshot within the TTCN-3 ETS.

5.2 Error handling

Explicit error handling is specified only for TRI operations called by the TTCN-3 Executable (TE). The SA or PA reports the status of a TRI operation in the return value of a TRI operation. The status value can either indicate the local success (*TRI_OK*) or failure (*TRI_Error*) of the TRI operation. Therefore, the TE may react to an error that occurred either within the SA or PA and issue, e.g. a test case error.

For TRI operations called by the SA or PA no explicit error handling is required since these operations are implemented in the TE. Here, the TE is in control over the test execution in the case that an error occurs in such a TRI operation.

Notice that specific error codes as well as the detection and handling of errors in any of the test system entities are beyond the scope of the current TRI specification.

5.3 Data interface

In the TRI operations only encoded test data shall be passed. The TTCN-3 Executable (TE) is responsible for encoding test data to be sent and decoding received test data in the respective TRI operations since encoding rules can be specified for or within a TTCN-3 module. Notice that the TE is required to encode test data even if no encoding information has been provided in a TTCN-3 ATS. In this case the tool vendor has to define an encoding.

Instead of defining an explicit data interface for TTCN-3 and ASN.1 data types, the TRI standard defines a set of abstract data types. These data types are used in the following definition of TRI operations to indicate which information is to be passed from the calling to the called entity, and vice versa. The concrete representation of these abstract data types as well as the definition of basic data types are defined in the respective language mappings in clauses 6, 7, and 8.

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.

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

5.3.1 Connection

TriComponentIdType A value of type **TriComponentIdType** includes an identifier, a name and the component type. The distinct value of the latter is the component type name as specified in the TTCN-3 ATS. This abstract type is mainly used to resolve TRI communication operations on TSI ports that have mappings to many test component ports. It is also used to resolve TCI component handling.

NOTE 1: For the handling of TCI operations on any or all components, the component type name can also be set to "ANY" or "ALL". In that case, both the identifier and the component name are to be ignored.

TriComponentIdListType A value of type **TriComponentIdListType** is a list of **TriComponentIdType**. This abstract type is used for multicast communication in TCI.

TriPortIdType A value of type **TriPortIdType** includes a value of type **TriComponentIdType** to represent the component to which the port belongs, a port index (if present), and the port and port type name as specified in the TTCN-3 ATS. The **TriPortIdType** type is mainly required to pass information about the TSI and connections to the TSI from the TE to the SA.

NOTE 2: If the port is in an n-dimensional port array of the form "port MyPortType myPort [d1] ... [dn], the port index *ind* of the port *p*[i1] ... [in] is computed by the following formula:

$$ind = (\sum_{k=1..n-1} ik * \prod_{l=k+1..n} dl) + in$$

EXAMPLE:

For a four-dimensional port-array with dimensions [10] [10] [10] [10], the port index *ind1* of the port at index [2] [3] [4] [5] is *ind1* = 2*(10*10*10) + 3*(10*10) + 4*10 + 5 = 2345

For a three-dimensional port-array with dimensions [2] [3] [4], the port index *ind2* of the port at index [1] [2] [2] is *ind2* = 1*(3*4) + 2*4 + 2 = 22.

TriPortIdListType A value of type **TriPortIdListType** is a list of **TriPortIdType**. This abstract type is used for initialization purposes after the invocation of a TTCN-3 test case.

5.3.2 Communication

TriMessageType A value of type **TriMessageType** is encoded test data that either is to be sent to the SUT or has been received from the SUT. The order of bits in a value of type **TriMessagetype** corresponds to the order of bits according to the encoding. If the encoded message consists of the bits *b0* .. *b9*, where *b0* is the first bit, then the value of type **triMessageType** contains "b0 b1 b2 b3 b4 b5 b6 b7 b8 b9" in this order. If padding bits are needed then these are added to the right.

TriAddressType A value of type **TriAddressType** indicates a source or destination address within the SUT. This abstract type can be used in TRI communication operations and is an open type, which is opaque to the TE.

TriAddressListType A value of type **TriAddressListType** is a list of **TriAddressType**. This abstract type is used for multicast communication in TRI.

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

TriParameterType	A value of type TriParameterType includes an encoded parameter and a value of TriParameterPassingModeType to represent the passing mode specified for the parameter in the TTCN-3 ATS.
TriParameterPassingModeType	A value of type TriParameterPassingModeType is either <i>in</i> , <i>inout</i> , or <i>out</i> . This abstract type is used in procedure based TRI communication operations and for external function calls.
TriParameterListType	A value of type TriParameterListType is a list of TriParameterType. This abstract type is used in procedure based TRI communication operations and for external function calls.
TriExceptionType	A value of type TriExceptionType is an encoded type and value of an exception that either is to be sent to the SUT or has been received from the SUT. This abstract type is used in procedure based TRI communication operations.

5.3.3 Timer

TriTimerIdType	A value of type TriTimerIdType specifies an identifier for a timer. This abstract type is required for all TRI timer operations.
TriTimerDurationType	A value of type TriTimerDurationType specifies the duration for a timer in seconds.

5.3.4 Miscellaneous

TriTestCaseIdType	A value of type TriTestCaseIdType is the name of a test case as specified in the TTCN-3 ATS.
TriFunctionIdType	A value of type TriFunctionIdType is the name of an external function as specified in the TTCN-3 ATS.
TriStatusType	A value of type TriStatusType is either <i>TRI_OK</i> or <i>TRI_Error</i> indicating the success or failure of a TRI operation.

5.4 Operation descriptions

All operation definitions are defined using the Interface Definition Language (IDL). Concrete language mappings are defined in clauses 6, 7 and 8.

For every TRI 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. 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 TRI 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 that are defined as optional in the corresponding TRI parameter description. The reserved values for these types are defined in each language mapping and will be subsequently referred to as the `null` value.

All functions in the interface are described using the following template:

F.n.m	Operation Name	calling entity → called entity
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	
InOutParameters	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	
Constraints	Description of any constraints that apply to calling the operation.	
Effect	Behaviour required of the called entity before the operation may return.	

5.5 Communication interface operations

5.5.1 triSAReset (TE → SA)

Signature	TriStatusType triSAReset()
In Parameters	n.a.
Out Parameters	n.a.
Return Value	The return status of the triSAReset operation. The return status indicates the local success (TRI_OK) or failure (TRI_Error) of the operation.
Constraints	This operation can be called by the TE at any time to reset the SA.
Effect	The SA shall reset all communication means that it is maintaining, e.g. reset static connections to the SUT, close dynamic connections to the SUT, discard any pending messages or procedure calls. The triResetSA operation returns TRI_OK in case the operation has been successfully performed, TRI_Error otherwise.

5.5.2 Connection handling operations

5.5.2.1 triExecuteTestCase (TE → SA)

Signature	TriStatusType triExecuteTestCase(in TriTestCaseIdType testCaseId, in TriPortIdListType tsiPortList)
In Parameters	testCaseId identifier of the test case that is going to be executed tsiPortList a list of test system interface ports defined for the test system
Out Parameters	n.a.
Return Value	The return status of the triExecuteTestCase operation. The return status indicates the local success (TRI_OK) or failure (TRI_Error) of the operation.
Constraints	This operation is called by the TE immediately before the execution of any test case. The test case that is going to be executed is indicated by the testCaseId. 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 in the TTCN-3 ATS then the tsiPortList contains all communication ports of the MTC test component. The ports in tsiPortList are ordered as they appear in the respective TTCN-3 component declaration.
Effect	The SA can set up any static connections to the SUT and initialize any communication means for TSI ports. The triExecuteTestCase operation returns TRI_OK in case the operation has been successfully performed, TRI_Error otherwise.

5.5.2.2 triMap (TE → SA)

Signature	TriStatusType triMap(in TriPortIdType compPortId, in TriPortIdType tsiPortId)
In Parameters	compPortId identifier of the test component port to be mapped tsiPortId identifier of the test system interface port to be mapped
Out Parameters	n.a.
Return Value	The return status of the triMap operation. The return status indicates the local success (TRI_OK) or failure (TRI_Error) of the operation.
Constraints	This operation is called by the TE when it executes a TTCN-3 map operation.
Effect	The SA can establish a dynamic connection to the SUT for the referenced TSI port. The triMap operation returns TRI_Error in case a connection could not be established successfully, TRI_OK otherwise. The operation should return TRI_OK in case no dynamic connection needs to be established by the test system.

5.5.2.3 triUnmap (TE → SA)

Signature	TriStatusType triUnmap(in TriPortIdType compPortId, in TriPortIdType tsiPortId)
In Parameters	compPortId identifier of the test component port to be unmapped tsiPortId identifier of the test system interface port to be unmapped
Out Parameters	n.a.
Return Value	The return status of the triUnmap operation. The return status indicates the local success (TRI_OK) or failure (TRI_Error) of the operation.
Constraints	This operation is called by the TE when it executes any TTCN-3 unmap operation.
Effect	The SA shall close a dynamic connection to the SUT for the referenced TSI port. The triUnmap operation returns TRI_Error in case a connection could not be closed successfully or no such connection has been established previously, TRI_OK otherwise. The operation should return TRI_OK in case no dynamic connections have to be established by the test system.

5.5.2.4 triEndTestCase (TE → SA)

Signature	TriStatusType triEndTestCase()
In Parameters	n.a.
Out Parameters	n.a.
Return Value	The return status of the triEndTestCase operation. The return status indicates the local success (TRI_OK) or failure (TRI_Error) of the operation.
Constraints	This operation is called by the TE immediately after the execution of any test case.
Effect	The SA can free resources, cease communication at system ports and to test components. The triEndTestCase operation returns TRI_OK in case the operation has been successfully performed, TRI_Error otherwise.

5.5.3 Message based communication operations

5.5.3.1 triSend (TE → SA)

Signature	TriStatusType triSend(in TriComponentIdType componentId, in TriPortIdType tsiPortId, in TriAddressType SUTaddress, in TriMessageType sendMessage)
In Parameters	componentId identifier of the sending test component tsiPortId identifier of the test system interface port via which the message is sent to the SUT Adaptor SUTaddress (optional) destination address within the SUT sendMessage the encoded message to be sent
Out Parameters	n.a.
Return Value	The return status of the triSend operation. The return status indicates the local success (TRI_OK) or failure (TRI_Error) of the operation.
Constraints	This operation is called by the TE when it executes a TTCN-3 unicast send operation on a component port, which has been mapped to a TSI port. This operation is called by the TE for all TTCN-3 send operations if no system component has been specified for a test case, i.e. only a MTC test component is created for a test case. The encoding of sendMessage has to be done in the TE prior to this TRI operation call.
Effect	The SA can send the message to the SUT. The triSend operation returns TRI_OK in case it has been completed successfully. Otherwise TRI_Error shall be returned. Notice that the return value TRI_OK does not imply that the SUT has received sendMessage.

5.5.3.2 triSendBC (TE → SA)

Signature	TriStatusType triSendBC(in TriComponentIdType componentId, in TriPortIdType tsiPortId, in TriMessageType sendMessage)
In Parameters	componentId identifier of the sending test component tsiPortId identifier of the test system interface port via which the message is sent to the SUT Adaptor sendMessage the encoded message to be sent
Out Parameters	n.a.
Return Value	The return status of the triSendBC operation. The return status indicates the local success (TRI_OK) or failure (TRI_Error) of the operation.
Constraints	This operation is called by the TE when it executes a TTCN-3 broadcast send operation on a component port, which has been mapped to a TSI port. This operation is called by the TE for all TTCN-3 send operations if no system component has been specified for a test case, i.e. only a MTC test component is created for a test case. The encoding of sendMessage has to be done in the TE prior to this TRI operation call.
Effect	The SA can broadcast the message to the SUT. The triSendBC operation returns TRI_OK in case it has been completed successfully. Otherwise TRI_Error shall be returned. Notice that the return value TRI_OK does not imply that the SUT has received sendMessage.

5.5.3.3 triSendMC (TE → SA)

Signature	TriStatusType triSendMC(in TriComponentIdType componentId, in TriPortIdType tsiPortId, in TriAddressListType SUTaddresses, in TriMessageType sendMessage)
In Parameters	componentId identifier of the sending test component tsiPortId identifier of the test system interface port via which the message is sent to the SUT Adaptor SUTaddresses destination addresses within the SUT sendMessage the encoded message to be sent
Out Parameters	n.a.
Return Value	The return status of the triSendMC operation. The return status indicates the local success (TRI_OK) or failure (TRI_Error) of the operation.
Constraints	This operation is called by the TE when it executes a TTCN-3 multicast send operation on a component port, which has been mapped to a TSI port. This operation is called by the TE for all TTCN-3 send operations if no system component has been specified for a test case, i.e. only a MTC test component is created for a test case. The encoding of sendMessage has to be done in the TE prior to this TRI operation call.
Effect	The SA can multicast the message to the SUT. The triSendMC operation returns TRI_OK in case it has been completed successfully. Otherwise TRI_Error shall be returned. Notice that the return value TRI_OK does not imply that the SUT has received sendMessage.

5.5.3.4 triEnqueueMsg (SA → TE)

Signature	void triEnqueueMsg(in TriPortIdType tsiPortId, in TriAddressType SUTaddress, in TriComponentIdType componentId, in TriMessageType receivedMessage)
In Parameters	tsiPortId identifier of the test system interface port via which the message is enqueued by the SUT Adaptor SUTaddress (optional) source address within the SUT componentId identifier of the receiving test component receivedMessage the encoded received message
Out Parameters	n.a.
Return Value	void
Constraints	This operation is called by the SA after it has received a message from the SUT. It can only be used when tsiPortId has been either previously mapped to a port of componentId or has been referenced in the previous triExecuteTestCase statement. In the invocation of a triEnqueueMsg operation receivedMessage shall contain an encoded value.
Effect	This operation shall pass the message to the TE indicating the component componentId to which the TSI port tsiPortId is mapped. The decoding of receivedMessage has to be done in the TE.

5.5.4 Procedure based communication operations

5.5.4.1 triCall (TE → SA)

Signature	TriStatusType triCall(in TriComponentIdType componentId, in TriPortIdType tsiPortId, in TriAddressType SUTaddress, in TriSignatureIdType signatureId, in TriParameterListType parameterList)
In Parameters	componentId identifier of the test component issuing the procedure call tsiPortId identifier of the test system interface port via which the procedure call is sent to the SUT Adaptor SUTaddress (optional) destination address within the SUT signatureId 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
Out Parameters	n.a.
Return Value	The return status of the triCall operation. The return status indicates the local success (TRI_OK) or failure (TRI_Error) of the operation.
Constraints	This operation is called by the TE when it executes a TTCN-3 unicast call operation on a component port, which has been mapped to a TSI port. This operation is called by the TE for all TTCN-3 call operations if no system component has been specified for a test case, i.e. only a MTC test component is created for a test case. All <i>in</i> and <i>inout</i> procedure parameters contain encoded values. The procedure parameters are the parameters specified in the TTCN-3 signature template. Their encoding has to be done in the TE prior to this TRI operation call.
Effect	On invocation of this operation the SA can initiate the procedure call corresponding to the signature identifier signatureId and the TSI port tsiPortId. The triCall operation shall return without waiting for the return of the issued procedure call (see note). This TRI operation returns TRI_OK on successful initiation of the procedure call, TRI_Error otherwise. No error shall be indicated by the SA in case the value of any <i>out</i> parameter is non-null. Notice that the return value of this TRI operation does not make any statement about the success or failure of the procedure call. Note that an optional timeout value, which can be specified in the TTCN-3 ATS for a call operation, is <i>not</i> included in the triCall 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.
NOTE:	This might be achieved for example by spawning a new thread or process. This handling of this procedure call is, however, dependent on implementation of the TE.

5.5.4.2 triCallBC (TE → SA)

Signature	TriStatusType triCallBC(in TriComponentIdType componentId, in TriPortIdType tsiPortId, in TriSignatureIdType signatureId, in TriParameterListType parameterList)
In Parameters	<p>componentId identifier of the test component issuing the procedure call</p> <p>tsiPortId identifier of the test system interface port via which the procedure call is sent to the SUT Adaptor</p> <p>signatureId identifier of the signature of the procedure call</p> <p>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.</p>
Out Parameters	n.a.
Return Value	The return status of the triCallBC operation. The return status indicates the local success (TRI_OK) or failure (TRI_Error) of the operation.
Constraints	<p>This operation is called by the TE when it executes a TTCN-3 broadcast call operation on a component port, which has been mapped to a TSI port. This operation is called by the TE for all TTCN-3 call operations if no system component has been specified for a test case, i.e. only a MTC test component is created for a test case.</p> <p>All <i>in</i> and <i>inout</i> procedure parameters contain encoded values.</p> <p>The procedure parameters are the parameters specified in the TTCN-3 signature template. Their encoding has to be done in the TE prior to this TRI operation call.</p>
Effect	<p>On invocation of this operation the SA can initiate and broadcast the procedure call corresponding to the signature identifier signatureId and the TSI port tsiPortId.</p> <p>The triCallBC operation shall return without waiting for the return of the issued procedure call (see note). This TRI operation returns <i>TRI_OK</i> on successful initiation of the procedure call, <i>TRI_Error</i> otherwise. No error shall be indicated by the SA in case the value of any <i>out</i> parameter is non-null. Notice that the return value of this TRI operation does not make any statement about the success or failure of the procedure call.</p> <p>Note that an optional timeout value, which can be specified in the TTCN-3 ATS for a call operation, is <i>not</i> included in the triCallBC 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.</p>
NOTE:	This might be achieved for example by spawning a new thread or process. This handling of this procedure call is, however, dependent on implementation of the TE.

5.5.4.3 triCallMC (TE → SA)

Signature	TriStatusType triCallMC(in TriComponentIdType componentId, in TriPortIdType tsiPortId, in TriAddressListType SUTaddresses, in TriSignatureIdType signatureId, in TriParameterListType parameterList)
In Parameters	<p>componentId identifier of the test component issuing the procedure call</p> <p>tsiPortId identifier of the test system interface port via which the procedure call is sent to the SUT Adaptor</p> <p>SUTaddresses destination addresses within the SUT</p> <p>signatureId identifier of the signature of the procedure call</p> <p>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.</p>
Out Parameters	n.a.
Return Value	The return status of the triCallMC operation. The return status indicates the local success (TRI_OK) or failure (TRI_Error) of the operation.
Constraints	<p>This operation is called by the TE when it executes a TTCN-3 multicast call operation on a component port, which has been mapped to a TSI port. This operation is called by the TE for all TTCN-3 call operations if no system component has been specified for a test case, i.e. only a MTC test component is created for a test case.</p> <p>All <i>in</i> and <i>inout</i> procedure parameters contain encoded values.</p> <p>The procedure parameters are the parameters specified in the TTCN-3 signature template. Their encoding has to be done in the TE prior to this TRI operation call.</p>
Effect	<p>On invocation of this operation the SA can initiate and multicast the procedure call corresponding to the signature identifier signatureId and the TSI port tsiPortId.</p> <p>The triCallMC operation shall return without waiting for the return of the issued procedure call (see note). This TRI operation returns <i>TRI_OK</i> on successful initiation of the procedure call, <i>TRI_Error</i> otherwise. No error shall be indicated by the SA in case the value of any <i>out</i> parameter is non-null. Notice that the return value of this TRI operation does not make any statement about the success or failure of the procedure call.</p> <p>Note that an optional timeout value, which can be specified in the TTCN-3 ATS for a call operation, is <i>not</i> included in the triCallMC 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.</p>
NOTE:	This might be achieved for example by spawning a new thread or process. This handling of this procedure call is, however, dependent on implementation of the TE.

5.5.4.4 triReply (TE → SA)

Signature	TriStatusType triReply(in TriComponentIdType componentId, in TriPortIdType tsiPortId, in TriAddressType SUTAddress, in TriSignatureIdType signatureId, in TriParameterListType parameterList, in TriParameterType returnValue)
In Parameters	<p>componentId identifier of the replying test component</p> <p>tsiPortId identifier of the test system interface port via which the reply is sent to the SUT Adaptor</p> <p>SUTAddress (optional) destination address within the SUT</p> <p>signatureId identifier of the signature of the procedure call</p> <p>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</p> <p>returnValue (optional) encoded return value of the procedure call</p>
Out Parameters	n.a.
Return Value	The return status of the triReply operation. The return status indicates the local success (TRI_OK) or failure (TRI_Error) of the operation.
Constraints	<p>This operation is called by the TE when it executes a TTCN-3 unicast reply operation on a component port that has been mapped to a TSI port. This operation is called by the TE for all TTCN-3 reply operations if no system component has been specified for a test case, i.e. only a MTC test component is created for a test case.</p> <p>All <i>out</i> and <i>inout</i> procedure parameters and the return value contain encoded values.</p> <p>The parameterList contains procedure call parameters. These parameters are the parameters specified in the TTCN-3 signature template. Their encoding has to be done in the TE prior to this TRI operation call.</p> <p>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.</p>
Effect	<p>On invocation of this operation the SA can issue the reply to a procedure call corresponding to the signature identifier signatureId and the TSI port tsiPortId.</p> <p>The triReply operation will return TRI_OK on successful execution of this operation, TRI_Error otherwise. The SA shall indicate no error in case the value of any <i>in</i> parameter or an undefined return value is different from null.</p>

5.5.4.5 triReplyBC (TE → SA)

Signature	TriStatusType triReplyBC(in TriComponentIdType componentId, in TriPortIdType tsiPortId, in TriSignatureIdType signatureId, in TriParameterListType parameterList, in TriParameterType returnValue)
In Parameters	<p>componentId identifier of the replying test component</p> <p>tsiPortId identifier of the test system interface port via which the reply is sent to the SUT Adaptor</p> <p>signatureId identifier of the signature of the procedure call</p> <p>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</p> <p>returnValue (optional) encoded return value of the procedure call</p>
Out Parameters	n.a.
Return Value	The return status of the triReplyBC operation. The return status indicates the local success (TRI_OK) or failure (TRI_Error) of the operation.
Constraints	<p>This operation is called by the TE when it executes a TTCN-3 broadcast reply operation on a component port that has been mapped to a TSI port. This operation is called by the TE for all TTCN-3 reply operations if no system component has been specified for a test case, i.e. only a MTC test component is created for a test case.</p> <p>All <i>out</i> and <i>inout</i> procedure parameters and the return value contain encoded values.</p> <p>The parameterList contains procedure call parameters. These parameters are the parameters specified in the TTCN-3 signature template. Their encoding has to be done in the TE prior to this TRI operation call.</p> <p>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.</p>
Effect	<p>On invocation of this operation the SA can broadcast the reply to procedure calls corresponding to the signature identifier signatureId and the TSI port tsiPortId.</p> <p>The triReplyBC operation will return TRI_OK on successful execution of this operation, TRI_Error otherwise. The SA shall indicate no error in case the value of any <i>in</i> parameter or an undefined return value is different from null.</p>

5.5.4.6 triReplyMC (TE → SA)

Signature	TriStatusType triReplyMC(in TriComponentIdType componentId, in TriPortIdType tsiPortId, in TriAddressListType SUTaddresses, in TriSignatureIdType signatureId, in TriParameterListType parameterList, in TriParameterType returnValue)
In Parameters	componentId identifier of the replying test component tsiPortId identifier of the test system interface port via which the reply is sent to the SUT Adaptor SUTaddresses destination addresses within the SUT signatureId 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) encoded return value of the procedure call
Out Parameters	n.a.
Return Value	The return status of the triReplyMC operation. The return status indicates the local success (TRI_OK) or failure (TRI_Error) of the operation.
Constraints	This operation is called by the TE when it executes a TTCN-3 multicast reply operation on a component port that has been mapped to a TSI port. This operation is called by the TE for all TTCN-3 reply operations if no system component has been specified for a test case, i.e. only a MTC test component is created for a test case. All <i>out</i> and <i>inout</i> procedure parameters and the return value contain encoded values. The parameterList contains procedure call parameters. These parameters are the parameters specified in the TTCN-3 signature procedure template. Their encoding has to be done in the TE prior to this TRI operation call. 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 SA can multicast the reply to procedure calls corresponding to the signature identifier signatureId and the TSI port tsiPortId. The triReplyMC operation will return TRI_OK on successful execution of this operation, TRI_Error otherwise. The SA shall indicate no error in case the value of any <i>in</i> parameter or an undefined return value is different from null.

5.5.4.7 triRaise (TE → SA)

Signature	TriStatusType triRaise(in TriComponentIdType componentId, in TriPortIdType tsiPortId, in TriAddressType SUTaddress, in TriSignatureIdType signatureId, in TriExceptionType exc)
In Parameters	componentId identifier of the test component raising the exception tsiPortId identifier of the test system interface port via which the exception is sent to the SUT Adaptor SUTaddress (optional) destination address within the SUT signatureId identifier of the signature of the procedure call which the exception is associated with exc the encoded exception
Out Parameters	n.a.
Return Value	The return status of the triRaise operation. The return status indicates the local success (TRI_OK) or failure (TRI_Error) of the operation.
Constraints	This operation is called by the TE when it executes a TTCN-3 unicast raise operation on a component port that has been mapped to a TSI port. This operation is called by the TE for all TTCN-3 raise operations if no system component has been specified for a test case, i.e. only a MTC test component is created for a test case. The encoding of the exception has to be done in the TE prior to this TRI operation call.
Effect	On invocation of this operation the SA can raise an exception to a procedure call corresponding to the signature identifier signatureId and the TSI port tsiPortId. The triRaise operation returns TRI_OK on successful execution of the operation, TRI_Error otherwise.

5.5.4.8 triRaiseBC (TE → SA)

Signature	TriStatusType triRaiseBC(in TriComponentIdType componentId, in TriPortIdType tsiPortId, in TriSignatureIdType signatureId, in TriExceptionType exc)
In Parameters	componentId identifier of the test component raising the exception tsiPortId identifier of the test system interface port via which the exception is sent to the SUT Adaptor signatureId identifier of the signature of the procedure call which the exception is associated with exc the encoded exception
Out Parameters	n.a.
Return Value	The return status of the triRaiseBC operation. The return status indicates the local success (TRI_OK) or failure (TRI_Error) of the operation.
Constraints	This operation is called by the TE when it executes a TTCN-3 broadcast raise operation on a component port that has been mapped to a TSI port. This operation is called by the TE for all TTCN-3 raise operations if no system component has been specified for a test case, i.e. only a MTC test component is created for a test case. The encoding of the exception has to be done in the TE prior to this TRI operation call.
Effect	On invocation of this operation the SA can raise and broadcast an exception to procedure calls corresponding to the signature identifier signatureId and the TSI port tsiPortId. The triRaiseBC operation returns TRI_OK on successful execution of the operation, TRI_Error otherwise.

5.5.4.9 triRaiseMC (TE → SA)

Signature	TriStatusType triRaiseMC(in TriComponentIdType componentId, in TriPortIdType tsiPortId, in TriAddressListType SUTaddresses, in TriSignatureIdType signatureId, in TriExceptionType exc)
In Parameters	componentId identifier of the test component raising the exception tsiPortId identifier of the test system interface port via which the exception is sent to the SUT Adaptor SUTaddresses destination addresses within the SUT signatureId identifier of the signature of the procedure call which the exception is associated with exc the encoded exception
Out Parameters	n.a.
Return Value	The return status of the triRaiseMC operation. The return status indicates the local success (TRI_OK) or failure (TRI_Error) of the operation.
Constraints	This operation is called by the TE when it executes a TTCN-3 multicast raise operation on a component port that has been mapped to a TSI port. This operation is called by the TE for all TTCN-3 raise operations if no system component has been specified for a test case, i.e. only a MTC test component is created for a test case. The encoding of the exception has to be done in the TE prior to this TRI operation call.
Effect	On invocation of this operation the SA can raise and multicast an exception to a procedure calls corresponding to the signature identifier signatureId and the TSI port tsiPortId. The triRaiseMC operation returns TRI_OK on successful execution of the operation, TRI_Error otherwise.

5.5.4.10 triEnqueueCall (SA → TE)

Signature	void triEnqueueCall(in TriPortIdType tsiPortId, in TriAddressType SUTaddress, in TriComponentIdType componentId, in TriSignatureIdType signatureId, in TriParameterListType parameterList)
In Parameters	tsiPortId identifier of the test system interface port via which the procedure call is enqueued by the SUT Adaptor SUTaddress (optional) source address within the SUT componentId identifier of the receiving test component signatureId 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. Description of data passed as parameters to the operation from the calling entity to the called entity
Out Parameters	n.a.
Return Value	void
Constraints	This operation can be called by the SA after it has received a procedure call from the SUT. It can only be used when tsiPortId has been either previously mapped to a port of componentId or referenced in the previous triExecuteTestCase statement. In the invocation of a triEnqueueCall operation all in and inout procedure parameters contain encoded values.
Effect	The TE can enqueue this procedure call with the signature identifier signatureId at the port of the component componentId to which the TSI port tsiPortId is mapped. The decoding of procedure parameters has to be done in the TE. The TE shall indicate no error in case the value of any out parameter is different from null.

5.5.4.11 triEnqueueReply (SA → TE)

Signature	void triEnqueueReply(in TriPortIdType tsiPortId, in TriAddressType SUTaddress, in TriComponentIdType componentId, in TriSignatureIdType signatureId, in TriParameterListType parameterList, in TriParameterType returnValue)
In Parameters	tsiPortId identifier of the test system interface port via which the reply is enqueued by the SUT Adaptor SUTaddress (optional) source address within the SUT componentId identifier of the receiving test component signatureId 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) encoded return value of the procedure call
Out Parameters	n.a.
Return Value	void
Constraints	This operation can be called by the SA after it has received a reply from the SUT. It can only be used when tsiPortId has been either previously mapped to a port of componentId or referenced in the previous triExecuteTestCase statement. In the invocation of a triEnqueueReply operation all out and inout procedure parameters and the return value contain encoded values. If no return type has been defined for the procedure signature in the TTCN-3 ATS, the distinct value null shall be used for the return value.
Effect	The TE can enqueue this reply to the procedure call with the signature identifier signatureId at the port of the component componentId to which the TSI port tsiPortId is mapped. The decoding of the procedure parameters has to be done within the TE. The TE shall indicate no error in case the value of any in parameter or an undefined return value is different from null.

5.5.4.12 triEnqueueException (SA → TE)

Signature	void triEnqueueException(in TriPortIdType tsiPortId, in TriAddressType SUTaddress, in TriComponentIdType componentId, in TriSignatureIdType signatureId, in TriExceptionType exc)
In Parameters	tsiPortId identifier for the test system interface port via which the exception is enqueued by the SUT Adaptor SUTaddress (optional) source address within the SUT componentId identifier of the receiving test component signatureId identifier of the signature of the procedure call which the exception is associated with exc the encoded exception
Out Parameters	n.a.
Return Value	void
Constraints	This operation can be called by the SA after it has received a reply from the SUT. It can only be used when tsiPortId has been either previously mapped to a port of componentId or referenced in the previous triExecuteTestCase statement. In the invocation of a triEnqueueException operation exception shall contain an encoded value.
Effect	The TE can enqueue this exception for the procedure call with the signature identifier signatureId at the port of the component componentId to which the TSI port tsiPortId is mapped. The decoding of the exception has to be done within the TE.

5.5.5 Miscellaneous operations

5.5.5.1 triSUTActionInformal (TE → SA)

Signature	TriStatusType triSUTActionInformal(in string description)
In Parameters	description an informal description of an action to be taken on the SUT
Out Parameters	n.a.
Return Value	The return status of the triSUTActionInformal operation. The return status indicates the local success (TRI_OK) or failure (TRI_Error) of the operation.
Constraints	This operation is called by the TE when it executes a TTCN-3 SUT action operation, which only contains a string.
Effect	On invocation of this operation the SA shall initiate the described actions to be taken on the SUT, e.g. turn on, initialize, or send a message to the SUT. The triSUTActionInformal operation returns TRI_OK on successful execution of the operation, TRI_Error otherwise. Notice that the return value of this TRI operation does not make any statement about the success or failure of the actions to be taken on the SUT.

5.5.5.2 triSUTActionTemplate (TE → SA)

Void.

5.6 Platform interface operations

5.6.1 triPAReset (TE → PA)

Signature	TriStatusType triPAReset()
In Parameters	n.a.
Out Parameters	n.a.
Return Value	The return status of the triPAReset operation. The return status indicates the local success (TRI_OK) or failure (TRI_Error) of the operation.
Constraints	This operation can be called by the TE at any time to reset the PA.
Effect	The PA shall reset all timing activities which it is currently performing, e.g. stop all running timers, discard any pending timeouts of expired timers. The triPAReset operation returns TRI_OK in case the operation has been performed successfully, TRI_Error otherwise.

5.6.2 Timer operations

5.6.2.1 triStartTimer (TE → PA)

Signature	TriStatusType triStartTimer(in TriTimerIdType timerId, in TriTimerDurationType timerDuration)
In Parameters	timerId identifier of the timer instance timerDuration duration of the timer in seconds
Out Parameters	n.a.
Return Value	The return status of the triStartTimer operation. The return status indicates the local success (TRI_OK) or failure (TRI_Error) of the operation.
Constraints	This operation is called by the TE when a timer needs to be started.
Effect	On invocation of this operation the PA shall start the indicated timer with the indicated duration. The timer runs from the value zero (0.0) up to the maximum specified by timerDuration. Should the timer indicated by timerId already be running it is to be restarted. When the timer expires the PA will call the triTimeout() operation with timerId. The triStartTimer operation returns TRI_OK if the timer has been started successfully, TRI_Error otherwise.

5.6.2.2 triStopTimer (TE → PA)

Signature	TriStatusType triStopTimer(in TriTimerIdType timerId)
In Parameters	timerId identifier of the timer instance
Out Parameters	n.a.
Return Value	The return status of the triStopTimer operation. The return status indicates the local success (TRI_OK) or failure (TRI_Error) of the operation.
Constraints	This operation is called by the TE when a timer is to be stopped.
Effect	On invocation of this operation the PA shall use the timerId to stop the indicated timer instance. The stopping of an inactive timer, i.e. a timer which has not been started or has already expired, should have no effect. The triStopTimer operation returns TRI_OK if the operation has been performed successfully, TRI_Error otherwise. Notice that stopping an inactive timer is a valid operation. In this case TRI_OK shall be returned.

5.6.2.3 triReadTimer (TE → PA)

Signature	TriStatusType triReadTimer(in TriTimerIdType timerId, out TriTimerDurationType elapsedTime)
In Parameters	timerId identifier of the timer instance
Out Parameters	elapsedTime value of the time elapsed since the timer has been started in seconds
Return Value	The return status of the triReadTimer operation. The return status indicates the local success (TRI_OK) or failure (TRI_Error) of the operation.
Constraints	This operation may be called by the TE when a TTCN-3 read timer operation is to be executed on the indicated timer.
Effect	On invocation of this operation the PA shall use the timerId to access the time that elapsed since this timer was started. The return value elapsedTime shall be provided in seconds. The reading of an inactive timer, i.e. a timer which has not been started or already expired, shall return an elapsed time value of zero. The triReadTimer operation returns TRI_OK if the operation has been performed successfully, TRI_Error otherwise.

5.6.2.4 triTimerRunning (TE → PA)

Signature	TriStatusType triTimerRunning(in TriTimerIdType timerId, out boolean running)
In Parameters	timerId identifier of the timer instance
Out Parameters	running status of the timer
Return Value	The return status of the triTimerRunning operation. The return status indicates the local success (TRI_OK) or failure (TRI_Error) of the operation.
Constraints	This operation may be called by the TE when a TTCN-3 running timer operation is to be executed on the indicated timer.
Effect	On invocation of this operation the PA shall use the timerId to access the status of the timer. The operation sets running to the boolean value true if and only if the timer is currently running. The triTimerRunning operation returns TRI_OK if the status of the timer has been successfully determined, TRI_Error otherwise.

5.6.2.5 triTimeout (PA → TE)

Signature	void triTimeout(in TriTimerIdType timerId)
In Parameters	timerId identifier of the timer instance
Out Parameters	n.a.
Return Value	void
Constraints	This operation is called by the PA after a timer, which has previously been started using the triStartTimer operation, has expired, i.e. it has reached its maximum duration value.
Effect	The timeout with the timerId can be added to the timeout list in the TE. The implementation of this operation in the TE has to be done in such a manner that it addresses the different TTCN-3 semantics for timers defined in ES 201 873-4 [3].

5.6.3 Miscellaneous operations

5.6.3.1 triExternalFunction (TE → PA)

Signature	TriStatusType triExternalFunction(in TriFunctionIdType functionId, inout TriParameterListType parameterList, out TriParameterType returnValue)
In Parameters	functionId identifier of the external function
Out Parameters	returnValue (optional) encoded return value
InOutParameters	parameterList a list of encoded parameters for the indicated function. The parameters in parameterList are ordered as they appear in the TTCN-3 function declaration.
Return Value	The return status of the triExternalFunction operation. The return status indicates the local success (TRI_OK) or failure (TRI_Error) of the operation.
Constraints	This operation is called by the TE when it executes a function which is defined to be TTCN-3 external (i.e. all non-external functions are implemented within the TE). In the invocation of a triExternalFunction operation by the TE all <i>in</i> and <i>inout</i> function parameters contain encoded values. No error shall be indicated by the PA in case the value of any <i>out</i> parameter is non-null.
Effect	For each external function specified in the TTCN-3 ATS the PA shall implement the behaviour. On invocation of this operation the PA shall invoke the function indicated by the identifier functionId. It shall access the specified <i>in</i> and <i>inout</i> function parameters in parameterList, evaluate the external function using the values of these parameters, and compute values for <i>inout</i> and <i>out</i> parameters in parameterList. The operation shall then return encoded values for all <i>inout</i> and <i>out</i> function parameters and the encoded return value of the external function. If no return type has been defined for this external function in the TTCN-3 ATS, the distinct value null shall be used for the latter. The triExternalFunction operation returns TRI_OK if the PA completes the evaluation of the external function successfully, TRI_Error otherwise. Note that whereas all other TRI operations are considered to be non-blocking, the triExternalFunction operation is considered to be <i>blocking</i> . That means that the operation shall not return before the indicated external function has been fully evaluated. External functions have to be implemented carefully as they could cause deadlock of test component execution or even the entire test system implementation.

6 Java language mapping

6.1 Introduction

This clause introduces the TRI 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 Runtime Interface defines how the IDL definitions described in clause 5 are mapped to the Java language. The language mapping is independent of the used Java version as only basic Java language constructs are used.

6.2 Names and scopes

6.2.1 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 parameter identifiers shall start with a lower case letter, and subsequent part building the parameter identifier start with a capital letter. For example the IDL parameter identifier **SUTaddress** maps to sutAddress in Java.

- Java interfaces or class identifiers are omitting the trailing `Type` used in the IDL definition. For example the IDL type `TriPortIdType` maps to `TriPortId` in Java.

The resulting mapping conforms to the standard Java coding conventions.

6.2.2 Scopes

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

6.3 Type mapping

6.3.1 Basic type mapping

Table 3 gives an overview on how the used basic IDL types are mapped to the Java types.

Table 3: Basic type mappings

IDL Type	Java Type
<code>boolean</code>	<code>org.etsi.ttcn.tri.TriBoolean</code>
<code>string</code>	<code>java.lang.String</code>

Other IDL basic types are not used within the IDL definition.

6.3.1.1 Boolean

The IDL `boolean` type is mapped to the interface `org.etsi.ttcn.tri.TriBoolean`, so that objects implementing this interface can act as holder objects.

The following interface is defined for `org.etsi.ttcn.tri.TriBoolean`:

```
// TriBoolean
package org.etsi.ttcn.tri;
public interface TriBoolean {
    public void setBooleanValue(boolean value);
    public boolean getBooleanValue();
}
```

6.3.1.1.1 Methods

- `setBooleanValue(boolean value)`

Sets this `TriBoolean` to the boolean value `value`.

- `getBooleanValue()`

Returns the boolean value represented by this `TriBoolean`.

6.3.1.2 String

The IDL `string` 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`.

6.3.2 Structured type mapping

The TRI 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.

6.3.2.1 TriPortIdType

TriPortIdType is mapped to the following interface:

```
// TRI IDL TriPortIdType
package org.etsi.ttcn.tri;
public interface TriPortId {
    public String getPortName();
    public String getPortTypeName();
    public TriComponentId getComponent();
    public boolean isArray();
    public int getPortIndex();
}
```

6.3.2.1.1 Methods

- `getPortName()`
Returns the port name as defined in the TTCN-3 specification.
- `getPortTypeName()`
Returns the port type name as defined in the TTCN-3 specification.
- `getComponent()`
Returns the component identifier that this `TriPortId` belongs to as defined in the TTCN-3 specification.
- `isArray()`
Returns `true` if this port is part of a port array, `false` otherwise.
- `getPortIndex()`
Returns the port index if this port is part of a port array starting at zero. If the port is not part of a port array, then `-1` is returned.

6.3.2.2 TriPortIdListType

TriPortIdListType is mapped to the following interface:

```
// TRI IDL TriPortIdListType
package org.etsi.ttcn.tri;
public interface TriPortIdList {
    public int size();
    public boolean isEmpty();
    public java.util.Enumeration getPortIds();
    public TriPortId get(int index);
}
```

6.3.2.2.1 Methods

- `size()`
Returns the number of ports in this list.
- `isEmpty()`
Returns `true` if this list contains no ports.
- `getPortIds()`
Returns an `Enumeration` over the ports in the list. The enumeration provides the ports in the same order as they appear in the list.
- `get(int index)`
Returns the `TriPortId` at the specified position.

6.3.2.3 TriComponentIdType

TriComponentIdType is mapped to the following interface:

```
// TRI IDL TriComponentIdType
package org.etsi.ttcn.tri;
public interface TriComponentId {
    public String getComponentId();
    public String getComponentName();
    public String getComponentTypeName();
    public TriPortIdList getPortList();
    public boolean equals(TriComponentId component);
}
```

6.3.2.3.1 Methods

- `getComponentId()`
Returns a representation of this unique component identifier.
- `getComponentName()`
Returns the component name as defined in the TTCN-3 specification. If no name is provided, an empty string is returned.
- `getComponentTypeName()`
Returns the component type name as defined in the TTCN-3 specification.
- `getPortList()`
Returns the component's port list as defined in the TTCN-3 specification.
- `equals(TriComponentId component)`
Compares `component` with this `TriComponentId` for equality. Returns `true` if and only if both components have the same representation of this unique component identifier, `false` otherwise.

6.3.2.4 TriComponentIdListType

TriComponentIdListType is mapped to the following interface:

```
// TRI IDL TriComponentIdListType
package org.etsi.ttcn.tri;
public interface TriComponentIdListType {
    public int size();
    public boolean isEmpty();
    public java.util.Enumeration getComponents();
    public TriComponentId get(int index);
    public void clear();
    public void add(TriComponentId comp);
}
```

6.3.2.4.1 Methods

- `size()`
Returns the number of components in this list.
- `isEmpty()`
Returns `true` if this list contains no components.
- `getComponents()`
Returns an `Enumeration` over the components in the list. The enumeration provides the components in the same order as they appear in the list.
- `get(int index)`
Returns the `TriComponentId` at the specified position.
- `clear()`
Removes all components from this `TriComponentIdList`.

- `add(TriComponentId comp)`
Adds `comp` to the end of this `TriComponentIdList`.

6.3.2.5 TriMessageType

TriMessageType is mapped to the following interface:

```
// TRI IDL TriMessageType
package org.etsi.ttcn.tri;
public interface TriMessage {
    public byte[] getEncodedMessage();
    public void setEncodedMessage(byte[] message);
    public int getNumberOfBits();
    public void setNumberOfBits(int amount);
    public boolean equals(TriMessage message);
}
```

6.3.2.5.1 Methods

- `getEncodedMessage()`
Returns the message encoded according the coding rules defined in the TTCN-3 specification.
- `setEncodedMessage(byte[] message)`
Sets the encoded message representation of this `TriMessage` to `message`.
- `getNumberOfBits()`
Returns the amount of bits of the message.
- `setNumberOfBits(int amount)`
Sets the amount of bits in the message.
- `equals(TriMessage message)`
Compares `message` with this `TriMessage` for equality. Returns `true` if and only if both messages have the same encoded representation, `false` otherwise.

6.3.2.6 TriAddressType

TriAddressType is mapped to the following interface:

```
// TRI IDL TriAddressType
package org.etsi.ttcn.tri;
public interface TriAddress {
    public byte[] getEncodedAddress();
    public void setEncodedAddress(byte[] address);
    public boolean equals(TriAddress address);
}
```

6.3.2.6.1 Methods

- `getEncodedAddress()`
Returns the encoded address.
- `setEncodedAddress(byte[] address)`
Set the encoded address of this `TriAddress` to `address`.
- `equals(TriAddress address)`
Compares `address` with this `TriAddress` for equality. Returns `true` if and only if both addresses have the same encoded representation, `false` otherwise.

6.3.2.7 TriAddressListType

TriAddressListType is mapped to the following interface:

```
// TRI IDL TriAddressListType
package org.etsi.ttcn.tri;
public interface TriAddressListType {
    public int size();
    public boolean isEmpty();
    public java.util.Enumeration getAddresses();
    public TriAddress get(int index);
    public void clear();
    public void add(TriAddress addr);
}
```

6.3.2.7.1 Methods

- `size()`
Returns the number of components in this list.
- `isEmpty()`
Returns `true` if this list contains no components.
- `getAddresses()`
Returns an `Enumeration` over the components in the list. The enumeration provides the addresses in the same order as they appear in the list.
- `get(int index)`
Returns the `TriAddress` at the specified position.
- `clear()`
Removes all addresses from this `TriAddressList`.
- `add(TriAddress addr)`
Adds `addr` to the end of this `TriAddressList`.

6.3.2.8 TriSignatureIdType

TriSignatureIdType is mapped to the following interface:

```
// TRI IDL TriSignatureIdType
package org.etsi.ttcn.tri;
public interface TriSignatureId {
    public String getSignatureName();
    public void setSignatureName(String sigName);
    public boolean equals(TriSignatureId sig);
}
```

6.3.2.8.1 Methods

- `getSignatureName()`
Returns the signature identifier as defined in the TTCN-3 specification.
- `setSignatureName(String sigName)`
Sets the signature identifier of this `TriSignatureId` to `sigName`.
- `equals(TriSignatureId sig)`
Compares `sig` with this `TriSignatureId` for equality. Returns `true` if and only if both signatures have the same signature identifier, `false` otherwise.

6.3.2.9 TriParameterType

TriParameterType is mapped to the following interface:

```
// TRI IDL TriParameterType
package org.etsi.ttcn.tri;
public interface TriParameter {
    public String getParameterName();
    public void setParameterName(String name);
    public int getParameterPassingMode();
    public void setParameterPassingMode(in mode);
    public byte[] getEncodedParameter();
    public void setEncodedParameter(byte[] parameter);
}
```

6.3.2.9.1 Methods

- `getParameterName()`
Returns the parameter name as defined in the TTCN-3 specification.
- `setParameterName(String name)`
Sets the name of this `TriParameter` parameter to `name`.
- `getParameterPassingMode()`
Returns the parameter passing mode of this parameter.
- `setParameterPassingMode(int mode)`
Sets the parameter mode of this `TriParameter` parameter to `mode`.
- `getEncodedParameter()`
Returns the encoded parameter representation of this `TriParameter`, or the `null` object if the parameter contains the distinct value `null` (see also clause 5.5.4).
- `setEncodedParameter(byte[] parameter)`
Sets the encoded parameter representation of this `TriParameter` 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` (see also clause 5.5.4).

6.3.2.10 TriParameterPassingModeType

TriParameterPassingModeType is mapped to the following interface:

```
// TRI IDL TriParameterPassingModeType
package org.etsi.ttcn.tri;
public interface TriParameterPassingMode {
    public final static int TRI_IN = 0;
    public final static int TRI_INOUT = 1;
    public final static int TRI_OUT = 2;
}
```

6.3.2.10.1 Constants

- `TRI_IN`
Will be used to indicate that a `TriParameter` is an `in` parameter.
- `TRI_INOUT`
Will be used to indicate that a `TriParameter` is an `inout` parameter.
- `TRI_OUT`
Will be used to indicate that a `TriParameter` is an `out` parameter.

6.3.2.11 TriParameterListType

TriParameterListType is mapped to the following interface:

```
// TRI IDL TriParameterListType
package org.etsi.ttcn.tri;
public interface TriParameterList {
    public int size();
    public boolean isEmpty();
    public java.util.Enumeration getParameters();
    public TriParameter get(int index);
    public void clear();
    public void add(TriParameter parameter);
}
```

6.3.2.11.1 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 `TriParameter` at the specified position.
- `clear()`
Removes all parameters from this `TriParameterList`.
- `add(TriParameter parameter)`
Adds `parameter` to the end of this `TriParameterList`.

6.3.2.12 TriExceptionType

TriExceptionType is mapped to the following interface:

```
// TRI IDL TriExceptionType
package org.etsi.ttcn.tri;
public interface TriException {
    public byte[] getEncodedException();
    public void setEncodedException(byte[] message);
    public boolean equals(TriException exc);
}
```

6.3.2.12.1 Methods

- `getEncodedException()`
Returns the exception encoded according to the coding rules defined in the TTCN-3 specification.
- `setEncodedException(byte[] exc)`
Sets the encoded exception representation of this `TriException` to `exc`.
- `equals(TriException exc)`
Compares `exc` with this `TriException` for equality. Returns `true` if and only if both exceptions have the same encoded representation, `false` otherwise.

6.3.2.13 TriTimerIdType

TriTimerIdType is mapped to the following interface:

```
// TRI IDL TriTimerIdType
package org.etsi.ttcn.tri;
public interface TriTimerId {
    public String getTimerName();
    public boolean equals(TriTimerId timer);
}
```

6.3.2.13.1 Methods

- `getTimerName()`
Returns the name of this timer identifier as defined in the TTCN-3 specification. In case of implicit timers the result is implementation dependent (see clause 4.1.2.4).
- `equals(TriTimerId timer)`
Compares `timer` with this `TriTimerId` for equality. Returns `true` if and only if both timers identifiers represent the same timer, `false` otherwise.

6.3.2.14 TriTimerDurationType

TriTimerDurationType is mapped to the following interface:

```
// TRI IDL TriTimerDurationType
package org.etsi.ttcn.tri;
public interface TriTimerDuration {
    public double getDuration();
    public void setDuration(double duration);
    public boolean equals(TriTimerDuration duration);
}
```

6.3.2.14.1 Methods

- `getDuration()`
Returns the duration of a timer as `double`.
- `setDuration(double duration)`
Sets the duration of this `TriTimerDuration` to `duration`.
- `equals(TriTimerDuration duration)`
Compares `duration` with this `TriTimerDuration` for equality. Returns `true` if and only if both have the same duration, `false` otherwise.

6.3.2.15 TriFunctionIdType

TriFunctionIdType is mapped to the following interface:

```
// TRI IDL TriFunctionIdType
package org.etsi.ttcn.tri;
public interface TriFunctionId {
    public String toString();
    public String getFunctionName();
    public boolean equals(TriFunctionId fun);
}
```

6.3.2.15.1 Methods

- `toString()`
Returns the string representation of the function as defined in TTCN-3 specification.
- `getFunctionName()`
Returns the function identifier as defined in the TTCN-3 specification.

- `equals(TriFunctionId fun)`
Compares `fun` with this `TriFunctionId` for equality. Returns `true` if and only if both functions have the same function identifier, `false` otherwise.

6.3.2.16 TriTestCaseIdType

TriTestCaseIdType is mapped to the following interface:

```
// TRI IDL TriTestCaseIdType
package org.etsi.ttcn.tri;
public interface TriTestCaseId {
    public String toString();
    public String getTestCaseName();
    public boolean equals(TriTestCaseId tc);
}
```

6.3.2.16.1 Methods

- `toString()`
Returns the string representation of the test case as defined in TTCN-3 specification.
- `getTestCaseName()`
Returns the test case identifier as defined in the TTCN-3 specification.
- `equals(TriTestCaseId tc)`
Compares `tc` with this `TriTestCaseId` for equality. Returns `true` if and only if both test cases have the same test case identifier, `false` otherwise.

6.3.2.17 TriActionTemplateType

The content of this clause is obsolete.

6.3.2.18 TriStatusType

TriStatusType is mapped to the following interface:

```
// TriStatusType
package org.etsi.ttcn.tri;
public interface TriStatus {
    public final static int TRI_OK = 0;
    public final static int TRI_ERROR = -1;
    public String toString();
    public int getStatus();
    public void setStatus(int status);
    public boolean equals(TriStatus status);
}
```

6.3.2.18.1 Methods

- `toString()`
Returns the string representation of the status.
- `getStatus()`
Returns the status of this `TriStatus`.
- `setStatus(int status)`
Sets the status of this `TriStatus`.
- `equals(TriStatus status)`
Compares `status` with this `TriStatus` for equality. Returns `true` if and only if they have the same status, `false` otherwise.

6.4 Constants

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

The following constants can be used to determine the parameter passing mode of TTCN-3 parameters (see also clause 6.3.2.8):

- `org.etsi.ttcn.tri.TriParameterPassingMode.TRI_IN;`
- `org.etsi.ttcn.tri.TriParameterPassingMode.TRI_INOUT;`
- `org.etsi.ttcn.tri.TriParameterPassingMode.TRI_OUT.`

The values of instances of these constants shall reflect the parameter passing mode defined in the TTCN-3 procedure signatures.

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

The following constants shall be used to indicate the local success of a method (see also clause 6.3.2.18):

- `org.etsi.ttcn.tri.TriStatus.TRI_OK;`
- `org.etsi.ttcn.tri.TriStatus.TRI_ERROR.`

6.5 Mapping of interfaces

The TRI IDL definition defines two interfaces, the `triCommunication` and the `triPlatform` 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 System Adaptor (SA) while others can only be called by the SA on the TE. This is reflected by dividing the TRI IDL interfaces in two sub interfaces, each suffixed by the called entity.

Table 4: Sub Interfaces

Calling/Called	TE	SA	PA
TE	-	<code>TriCommunicationSA</code>	<code>triPlatformPA</code>
SA	<code>TriCommunicationTE</code>	-	-
PA	<code>TriPlatformTE</code>	-	-

All methods defined in this interfaces should behave as defined in clause 5.

6.5.1 Out and InOut Parameter Passing Mode

The following IDL types are used in `out` or `inout` parameter passing mode:

- `TriParameter.`
- `TriParameterList.`
- `TriBoolean.`
- `TriTimerDuration.`

In case they are used in `out` or `inout` parameter passing mode objects of the respective class will be passed with the method call. The called entity can then access methods to set the return values.

6.5.2 `triCommunication` - Interface

The `triCommunication` interface is divided into two sub interfaces, the `triCommunicationSA` interface, defining calls from the TE to the SA and the `triCommunicationTE` interface, defining calls from the SA to the TE.

6.5.2.1 triCommunicationSA

The triCommunicationSA interface is mapped to the following interface:

```
// TriCommunication
// TE -> SA
package org.etsi.ttcn.tri;
public interface TriCommunicationSA {
    // Reset Operation
    // Ref: TRI-Definition 5.5.1
    TriStatus triSAReset();

    // Connection handling operations
    // Ref: TRI-Definition 5.5.2.1
    public TriStatus triExecuteTestCase(TriTestCaseId
        testCaseId, TriPortIdList tsiPorts);
    // Ref: TRI-Definition 5.5.2.2
    public TriStatus triMap(TriPortId compPortId, TriPortId tsiPortId);
    // Ref: TRI-Definition 5.5.2.3
    public TriStatus triUnmap(TriPortId compPortId, TriPortId tsiPortId);
    // Ref: TRI-Definition 5.5.2.4
    public TriStatus triEndTestCase();

    // Message based communication operations
    // Ref: TRI-Definition 5.5.3.1
    public TriStatus triSend(TriComponentId componentId, TriPortId tsiPortId,
        TriAddress sutAddress, TriMessage sendMessage);
    // Ref: TRI-Definition 5.5.3.2
    public TriStatus triSendBC(TriComponentId componentId, TriPortId tsiPortId,
        TriMessage sendMessage);
    // Ref: TRI-Definition 5.5.3.3
    public TriStatus triSendMC(TriComponentId componentId, TriPortId tsiPortId,
        TriAddressList sutAddresses, TriMessage sendMessage);

    // Procedure based communication operations
    // Ref: TRI-Definition 5.5.4.1
    public TriStatus triCall(TriComponentId componentId,
        TriPortId tsiPortId, TriAddress sutAddress,
        TriSignatureId signatureId, TriParameterList parameterList);
    // Ref: TRI-Definition 5.5.4.2
    public TriStatus triCallBC(TriComponentId componentId,
        TriPortId tsiPortId,
        TriSignatureId signatureId, TriParameterList parameterList);
    // Ref: TRI-Definition 5.5.4.3
    public TriStatus triCallMC(TriComponentId componentId,
        TriPortId tsiPortId, TriAddressList sutAddresses,
        TriSignatureId signatureId, TriParameterList parameterList);

    // Ref: TRI-Definition 5.5.4.4
    public TriStatus triReply(TriComponentId componentId,
        TriPortId tsiPortId, TriAddress sutAddress,
        TriSignatureId signatureId, TriParameterList parameterList,
        TriParameter returnValue);
    // Ref: TRI-Definition 5.5.4.5
    public TriStatus triReplyBC(TriComponentId componentId,
        TriPortId tsiPortId,
        TriSignatureId signatureId, TriParameterList parameterList,
        TriParameter returnValue);
    // Ref: TRI-Definition 5.5.4.6
    public TriStatus triReplyMC(TriComponentId componentId,
        TriPortId tsiPortId, TriAddressList sutAddresses,
        TriSignatureId signatureId, TriParameterList parameterList,
        TriParameter returnValue);

    // Ref: TRI-Definition 5.5.4.7
    public TriStatus triRaise(TriComponentId componentId, TriPortId tsitPortId,
        TriAddress sutAddress,
        TriSignatureId signatureId,
        TriException exc);
    // Ref: TRI-Definition 5.5.4.8
    public TriStatus triRaiseBC(TriComponentId componentId,
        TriPortId tsitPortId,
        TriSignatureId signatureId,
        TriException exc);
    // Ref: TRI-Definition 5.5.4.9
    public TriStatus triRaiseMC(TriComponentId componentId, TriPortId tsitPortId,
        TriAddresses sutAddresses,
        TriSignatureId signatureId,
```

```

        TriException exc);

    // Miscellaneous operations
    // Ref: TRI-Definition 5.5.5.1
    public TriStatus triSutActionInformal(String description);
}

```

6.5.2.2 triCommunicationTE

The `triCommunicationTE` interface is mapped to the following interface:

```

// TriCommunication
// SA -> TE
package org.etsi.ttcn.tri;
public interface TriCommunicationTE {
    // Message based communication operations
    // Ref: TRI-Definition 5.5.3.4
    public void triEnqueueMsg(TriPortId tsiPortId,
        TriAddress sutAddress, TriComponentId componentId,
        TriMessage receivedMessage);

    // Procedure based communication operations
    // Ref: TRI-Definition 5.5.4.10
    public void triEnqueueCall(TriPortId tsiPortId,
        TriAddress sutAddress, TriComponentId componentId,
        TriSignatureId signatureId, TriParameterList parameterList );

    // Ref: TRI-Definition 5.5.4.11
    public void triEnqueueReply(TriPortId tsiPortId, TriAddress sutAddress,
        TriComponentId componentId, TriSignatureId signatureId,
        TriParameterList parameterList, TriParameter returnValue);

    // Ref: TRI-Definition 5.5.4.12
    public void triEnqueueException(TriPortId tsiPortId,
        TriAddress sutAddress, TriComponentId componentId,
        TriSignatureId signatureId, TriException exc);
}

```

6.5.3 triPlatform - Interface

The `triPlatform` interface is divided in two sub interfaces, the `triPlatformPA` interface, defining calls from the TE to the PA and the `triPlatformTE` interface, defining calls from the PA to the TE.

6.5.3.1 TriPlatformPA

The `triPlatformPA` interface is mapped to the following interface:

```

// TriPlatform
// TE -> PA
package org.etsi.ttcn.tri;
public interface TriPlatformPA {
    // Ref: TRI-Definition 5.6.1
    public TriStatus triPAReset();

    // Timer handling operations
    // Ref: TRI-Definition 5.6.2.1
    public TriStatus triStartTimer(TriTimerId timerId,
        TriTimerDuration timerDuration);

    // Ref: TRI-Definition 5.6.2.2
    public TriStatus triStopTimer(TriTimerId timerId);

    // Ref: TRI-Definition 5.6.2.3
    public TriStatus triReadTimer(TriTimerId timerId,
        TriTimerDuration elapsedTime);

    // Ref: TRI-Definition 5.6.2.4
    public TriStatus triTimerRunning(TriTimerId timerId,
        TriBoolean running);

    // Miscellaneous operations

```

```
// Ref: TRI-Definition 5.6.3.1
public TriStatus triExternalFunction(TriFunctionId functionId,
    TriParameterList parameterList, TriParameter returnValue);
}
```

6.5.3.2 TriPlatformTE

The `TriPlatformTE` interface is mapped to the following Java interface:

```
// TriPlatform
// PA -> TE
package org.etsi.ttcn.tri;
public interface TriPlatformTE {
    // Ref: TRI-Definition 5.6.2.5
    public void triTimeout(TriTimerId timerId);
}
```

6.6 Optional parameters

Clause 5.4 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 `triSend` operation the address parameter shall be omitted the operation invocation shall be `triSend(componentId, tsiPortId, null, sendMessage)`.

6.7 TRI 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, SA, and PA the mechanism how the TE, the SA, or the PA 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 SA and PA to register the TE, SA, and PA to their respective communication partner.

6.8 Error handling

Beside the error handling as defined in clause 5.2 no additional error handling is defined within this Java language mapping. In particular no exception handling mechanisms are defined.

7 ANSI C language mapping

7.1 Introduction

This clause defines the TRI ANSI-C language mapping for the abstract data types specified in clause 5.35.3. For basic IDL types, the mapping conforms to OMG recommendations.

7.2 Names and scopes

C parameter identifiers shall start with a lower case letter, and subsequent part building the parameter identifier start with a capital letter. For example the IDL parameter `SUTaddress` maps to `sutAddress` in C.

Abstract data type identifiers in C are omitting the trailing `Type` used in the IDL definition. For example the IDL type `TriPortIdType` maps to `TriPortId` in C.

Older C specifications have restricted the identifier uniqueness to the most significant 8 characters. Nevertheless, the recent ANSI-C specifications have moved this limitation to the 31 most significant characters. Aside from this issue, no naming or scope conflicts have been identified in this mapping.

7.2.1 Abstract type mapping

TRI ADT	ANSI C Representation	Notes and comments
TriAddress	BinaryString	
TriAddressList	typedef struct TriAddressList { TriAddress** addrList; long int length; } TriAddressList;	NOTE 1: No special values mark the end of addrList []. The length field shall be used to traverse this array properly.
TriComponentId	typedef struct TriComponentId { BinaryString compInst; String compName; QualifiedName compType; } TriComponentId;	NOTE 2: compInst is for component instance.
TriComponentIdList	typedef struct TriComponentIdList { TriComponentId** compIdList; long int length; } TriComponentIdList;	NOTE 3: No special values mark the end of compIdList []. The length field shall be used to traverse this array properly.
TriException	BinaryString	
TriFunctionId	QualifiedName	
TriMessage	BinaryString	
TriParameterList	typedef struct TriParameterList { TriParameter** parList; long int length; } TriParameterList;	NOTE 4: No special values mark the end of parList. The length field shall be used to traverse this array properly.
TriParameter	typedef struct TriParameter { BinaryString par; TriParameterPassingMode mode; } TriParameter;	
TriParameterPassingMode	typedef enum { TRI_IN = 0, TRI_INOUT = 1, TRI_OUT = 2 } TriParameterPassingMode;	NOTE 5: The values of instances of this type shall reflect the parameter passing mode defined in the corresponding TTCN-3 procedure signatures.
TriPortIdList	typedef struct TriPortIdList { TriPortId** portIdList; long int length; } TriPortIdList;	NOTE 6: No special values mark the end of portIdList []. The length field shall be used to traverse this array properly.
TriPortId	typedef struct TriPortId { TriComponentId compInst; char* portName; long int portIndex; QualifiedName portType; void* aux; } TriPortId;	NOTE 7: compInst is for component instance. NOTE 8: For a singular (non-array) declaration, the portIndex value should be -1. NOTE 9: The aux field is for future extensibility of TRI functionality.
TriSignatureId	QualifiedName	
TriStatus	long int #define TRI_ERROR -1 #define TRI_OK 0	NOTE 10: All negative values are reserved for future extension of TRI functionality.

TRI ADT	ANSI C Representation	Notes and comments
TriTestCaseId	QualifiedName	
TriTimerDuration	Double	
TriTimerId	BinaryString	NOTE 11: Pending ETSI statement on timer and snapshot semantics may influence future representation!

7.2.2 ANSI C type definitions

C ADT	Type definition	Notes and comments
BinaryString	<pre>typedef struct BinaryString { unsigned char* data; long int bits; void* aux; } BinaryString;</pre>	<p>NOTE 1: data is a non-null-terminated string.</p> <p>NOTE 2: bits is the number of bits used in data. bits value -1 is used to denote omitted value.</p> <p>NOTE 3: The aux field is for future extensibility of TRI functionality.</p>
QualifiedName	<pre>typedef struct QualifiedName { char* moduleName; char* objectName; void* aux; } QualifiedName;</pre>	<p>NOTE 4: The moduleName and objectName fields are the TTCN-3 identifiers literally.</p> <p>NOTE 5: The aux field is for future extensibility of TRI functionality.</p>

7.2.3 IDL type mapping

IDL type	ANSI C Representation	Notes and comments
Boolean	unsigned char	From OMG IDL to C++ mapping
String	char*	From OMG IDL to C++ mapping

7.2.4 TRI operation mapping

IDL Representation	ANSI C Representation
TriStatusType triSAReset()	TriStatus triSAReset()
TriStatusType triExecuteTestCase (in TriTestCaseIdType testCaseId, in TriPortIdListType tsiPortList)	TriStatus triExecuteTestCase (const TriTestCaseId* testCaseId, const TriPortIdList* tsiPortList)
TriStatusType triMap (in TriPortIdType compPortId, in TriPortIdType tsiPortId)	TriStatus triMap (const TriPortId* compPortId, const TriPortId* tsiPortId)
TriStatusType triUnmap (in TriPortIdType compPortId, in TriPortIdType tsiPortId)	TriStatus triUnmap (const TriPortId* compPortId, const TriPortId* tsiPortId)
TriStatusType triEndTestCase()	TriStatus triEndTestCase()
TriStatusType triSend (in TriComponentIdType componentId, in TriPortIdType tsiPortId, in TriAddressType sutAddress, in TriMessageType sendMessage)	TriStatus triSend (const TriComponentId* componentId, const TriPortId* tsiPortId, const TriAddress* sutAddress, const TriMessage* sendMessage)
TriStatusType triSendBC (in TriComponentIdType componentId, in TriPortIdType tsiPortId, in TriMessageType sendMessage)	TriStatus triSendBC (const TriComponentId* componentId, const TriPortId* tsiPortId, const TriMessage* sendMessage)
TriStatusType triSendMC (in TriComponentIdType componentId, in TriPortIdType tsiPortId, in TriAddressListType sutAddresses, in TriMessageType sendMessage)	TriStatus triSendMC (const TriComponentId* componentId, const TriPortId* tsiPortId, const TriAddressList* sutAddresses, const TriMessage* sendMessage)

IDL Representation	ANSI C Representation
void triEnqueueMsg (in TriPortIdType tsiPortId, in TriAddressType SUTAddress, in TriComponentIdType componentId, in TriMessageType receivedMessage)	void triEnqueueMsg (const TriPortId* tsiPortId, const TriAddress* sutAddress, const TriComponentId* componentId, const TriMessage* receivedMessage)
TriStatusType triCall (in TriComponentIdType componentId, in TriPortIdType tsiPortId, in TriAddressType SUTAddress, in TriSignatureIdType signatureId, in TriParameterListType parameterList)	TriStatus triCall (const TriComponentId* componentId, const TriPortId* tsiPortId, const TriAddress* sutAddress, const TriSignatureId* signatureId, const TriParameterList* parameterList)
TriStatusType triCallBC (in TriComponentIdType componentId, in TriPortIdType tsiPortId, in TriSignatureIdType signatureId, in TriParameterListType parameterList)	TriStatus triCallBC (const TriComponentId* componentId, const TriPortId* tsiPortId, const TriSignatureId* signatureId, const TriParameterList* parameterList)
TriStatusType triCallMC (in TriComponentIdType componentId, in TriPortIdType tsiPortId, in TriAddressListType SUTAddresses, in TriSignatureIdType signatureId, in TriParameterListType parameterList)	TriStatus triCallMC (const TriComponentId* componentId, const TriPortId* tsiPortId, const TriAddressList* sutAddresses, const TriSignatureId* signatureId, const TriParameterList* parameterList)
TriStatusType triReply (in TriComponentIdType componentId, in TriPortIdType tsiPortId, in TriAddressType SUTAddress, in TriSignatureIdType signatureId, in TriParameterListType parameterList, in TriParameterType returnValue)	TriStatus triReply (const TriComponentId* componentId, const TriPortId* tsiPortId, const TriAddress* sutAddress, const TriSignatureId* signatureId, const TriParameterList* parameterList, const TriParameter* returnValue)
TriStatusType triReplyBC (in TriComponentIdType componentId, in TriPortIdType tsiPortId, in TriSignatureIdType signatureId, in TriParameterListType parameterList, in TriParameterType returnValue)	TriStatus triReplyBC (const TriComponentId* componentId, const TriPortId* tsiPortId, const TriSignatureId* signatureId, const TriParameterList* parameterList, const TriParameter* returnValue)
TriStatusType triReplyMC (in TriComponentIdType componentId, in TriPortIdType tsiPortId, in TriAddressListType SUTAddresses, in TriSignatureIdType signatureId, in TriParameterListType parameterList, in TriParameterType returnValue)	TriStatus triReplyMC (const TriComponentId* componentId, const TriPortId* tsiPortId, const TriAddressList* sutAddresses, const TriSignatureId* signatureId, const TriParameterList* parameterList, const TriParameter* returnValue)
TriStatusType triRaise (in TriComponentIdType componentId, in TriPortIdType tsiPortId, in TriAddressType SUTAddress, in TriSignatureIdType signatureId, in TriExceptionType exc)	TriStatus triRaise (const TriComponentId* componentId, const TriPortId* tsiPortId, const TriAddress* sutAddress, const TriSignatureId* signatureId, const TriException* exception)
TriStatusType triRaiseBC (in TriComponentIdType componentId, in TriPortIdType tsiPortId, in TriSignatureIdType signatureId, in TriExceptionType exc)	TriStatus triRaiseBC (const TriComponentId* componentId, const TriPortId* tsiPortId, const TriSignatureId* signatureId, const TriException* exception)
TriStatusType triRaiseMC (in TriComponentIdType componentId, in TriPortIdType tsiPortId, in TriAddressListType SUTAddresses, in TriSignatureIdType signatureId, in TriExceptionType exc)	TriStatus triRaiseMC (const TriComponentId* componentId, const TriPortId* tsiPortId, const TriAddressList* sutAddresses, const TriSignatureId* signatureId, const TriException* exception)
void triEnqueueCall (in TriPortIdType tsiPortId, in TriAddressType SUTAddress, in TriComponentId componentId, in TriSignatureIdType signatureId, in TriParameterListType parameterList)	void triEnqueueCall (const TriPortId* tsiPortId, const TriAddress* sutAddress, const TriComponentId* componentId, const TriSignatureId* signatureId, const TriParameterList* parameterList)
void triEnqueueReply (in TriPortIdType tsiPortId, in TriAddressType SUTAddress, in TriComponentIdType componentId, in TriSignatureIdType signatureId, in TriParameterListType parameterList, in TriParameterType returnValue)	void triEnqueueReply (const TriPortId* tsiPortId, const TriAddress* sutAddress, const TriComponentId* componentId, const TriSignatureId* signatureId, const TriParameterList* parameterList, const TriParameter* returnValue)

IDL Representation	ANSI C Representation
void triEnqueueException (in TriPortIdType tsiPortId, in TriAddressType sutAddress, in TriComponentIdType componentId, in TriSignatureIdType signatureId, in TriExceptionType exc)	void triEnqueueException (const TriPortId* tsiPortId, const TriAddress* sutAddress, const TriComponentId* componentId, const TriSignatureId* signatureId, const TriException* exception)
TriStatusType triSUTActionInformal (in string description)	TriStatus triSUTActionInformal (const char* description)
TriStatusType triPAReset()	TriStatus triPAReset()
TriStatusType triStartTimer (in TriTimerIdType timerId, in TriTimerDurationType timerDuration)	TriStatus triStartTimer (const TriTimerId* timerId, TriTimerDuration timerDuration)
TriStatusType triStopTimer (in TriTimerIdType timerId)	TriStatus triStopTimer (const TriTimerId* timerId)
TriStatusType triReadTimer (in TriTimerIdType timerId, out TriTimerDurationType elapsedTime)	TriStatus triReadTimer (const TriTimerId* timerId, TriTimerDuration* elapsedTime)
TriStatusType triTimerRunning (in TriTimerIdType timerId, out boolean running)	TriStatus triTimerRunning (const TriTimerId* timerId, unsigned char* running)
void triTimeout (in TriTimerIdType timerId)	void triTimeout (const TriTimerId* timerId)
TriStatusType triExternalFunction (in TriFunctionIdType functionId, inout TriParameterListType parameterList, out TriParameterType returnValue)	TriStatus triExternalFunction (const TriFunctionId* functionId, TriParameterList* parameterList, TriParameter* returnValue)

7.3 Memory management

The content of this clause is obsolete.

7.4 Error handling

No error handling has been defined for this mapping.

8 C++ language mapping

8.1 Introduction

This clause introduces the TRI C++ language mapping for the definitions given in clause 5.

8.2 Names and scopes

The namespace `ORG_ETSI_TTCN3_TRI` has been defined for the TRI C++ mapping, in order to avoid conflicts with the different names used, for example, in the C mapping.

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

8.3 Memory management

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

8.4 Error handling

No additional error handling has been defined for this mapping the one defined in clause 5.2.

8.5 Type mapping

This clause introduces the TRI C++ language mapping for the abstract types defined in 5.3. The following concepts have been used:

- Pure classes have been used, following the concept of an interface.
- Basic data types have both set and get methods, so that they can be handled in a general way.
- The Standard Template Library (STL) has been used as it is a standardised way of using container classes, and iterators, such as lists. All classes define the operator "<" for easy insertion in STL containers.
- C++ types have been encapsulated under abstract definitions, like Tfloat or Tinteger.

8.5.1 Encapsulated C++ types

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

- Boolean type definition: `typedef bool Tboolean.`
- Integer type definition: `typedef long int Tinteger.`
- Size type definition: `typedef unsigned long int Tsize.`
- Float type definition: `typedef double Tfloat.`
- Byte type definition: `typedef unsigned char Tbyte.`
- String type definition: `typedef std::string Tstring.`

8.5.2 Abstract data types

8.5.2.1 QualifiedName

This class defines a TTCN-3 identifier: moduleName + objectName. It is mapped to the following pure class:

```
class QualifiedName {
public:
~QualifiedName ();
const Tstring & getModuleName ();
void setModuleName (const Tstring &mName);
const Tstring & getObjectName ();
void setObjectName (const Tstring &oName);
Tstring toString ();
Tboolean equals (const QualifiedName &qn);
QualifiedName * cloneQualifiedName ();
Tboolean operator< (const QualifiedName &qn);
}
```

8.5.2.1.1 Methods

- [~QualifiedName](#) ()
Destructor.
- [Tstring](#) & [getModuleName](#) ()
Gets the module name as string value.

- [setModuleName](#) (const [Tstring](#) &mName)
Sets the module name from string parameter.
- [getObjectName](#) ()
Gets the object name as string value.
- [setObjectName](#) (const [Tstring](#) &oName)
Sets the object name from string parameter.
- [toString](#) ()
Returns the string representation of the [QualifiedName](#).
- [equals](#) (const [QualifiedName](#) &qn)
Returns true if both objects are equal.
- [QualifiedName](#) * [cloneQualifiedName](#) ()
Returns a copy of the [QualifiedName](#).
- [operator<](#) (const [QualifiedName](#) &qn)
Operator < overload.

8.5.2.2 TriAddress

A value of type [TriAddress](#) indicates a source or destination address within the SUT. The [TriAddress](#) class contains a C++ template: [TAddress](#). It is mapped to the following pure virtual class:

```
template <typename TAddress>
class TriAddress {
public:
    virtual ~TriAddress ()
    virtual TAddress & getAddress() const = 0;
    virtual void setAddress (const TAddress &T) = 0;
    virtual Tboolean equals (const TriAddress &add) const =0;
    virtual TriAddress * cloneAddress () const =0;
    virtual Tboolean operator< (const TriAddress &add) const =0;
}
```

8.5.2.2.1 Methods

- [~TriAddress](#) ()
Destructor.
- [getAddress](#) ()
Get the address value
- [setAddress](#) (const [TAddress](#) &T)
Set the address value
- [equals](#) (const [TriAddress](#) &add)
Returns true if both [TriAddress](#) objects are equal.
- [cloneAddress](#) ()
Returns a copy of the [TriAddress](#).
- [operator<](#) (const [TriAddress](#) &add)

Operator < overload.

8.5.2.3 TriAddressList

The value of this type is a list of TriAddress elements. This abstract type is used for multicast communication in TRI. It is mapped to the following pure virtual class:

```
class TriAddressList {
public:
    virtual ~TriAddressList ();
    virtual Tsize size () const =0;
    virtual Tboolean isEmpty () const =0;
    virtual const std::vector< TriAddress * > & getAddresses () const =0;
    virtual const TriAddress & get (Tsize index) const =0;
    virtual void clear ()=0;
    virtual void add (const TriAddress &elem)=0;
    virtual Tboolean equals (const TriAddressList &add1) const =0;
    virtual TriAddressList * cloneAddressList () const =0;
    virtual Tboolean operator< (const TriAddressList &add1) const =0;
}

```

8.5.2.3.1 Methods

- [~TriAddressList](#) ()
Destructor.
- [size](#) ()
Returns the number of addresses in the list.
- [isEmpty](#) ()
Returns true if address list is empty.
- [get](#) ([Tsize](#) index)
Gets TriAddress element at specified position from the address list.
- [clear](#) ()
Removes all TriAddress elements from the list.
- [add](#) (const [TriAddress](#) &elem)
Adds a TriAddress element to the list.
- [equals](#) (const [TriAddressList](#) &add1)
Returns true if both TriAddressList objects are equal.
- [cloneAddressList](#) ()
Returns a copy of the [TriAddressList](#).
- [operator<](#) (const [TriAddressList](#) &add1)
Operator < overload.

8.5.2.4 TriComponentId

A value of type `TriComponentId` includes an identifier, a name and the component type. This abstract type is mainly used to resolve TRI communication operations on TSI ports that have mappings to many test component ports. It is mapped to the following pure virtual class:

```
class TriComponentId {
public:
    virtual ~TriComponentId ();
    virtual const QualifiedName & getComponentTypeName () const =0;
    virtual void setComponentTypeName (const QualifiedName &tName)=0;
    virtual Tstring & getComponentName () const =0;
    virtual void setComponentName (const Tstring &sName)=0;
    virtual const Tinteger & getComponentId () const =0;
    virtual void setComponentId (const Tinteger &id)=0;
    virtual Tstring toString () const =0;
    virtual Tboolean equals (const TriComponentId &cmp) const =0;
    virtual TriComponentId * cloneComponentId () const =0;
    virtual Tboolean operator< (const TriComponentId &cmp) const =0;
}
```

8.5.2.4.1 Methods

- [~TriComponentId](#) ()
Destructor.
- [getComponentTypeName](#) ()
Returns a const reference to the component type name.
- [setComponentTypeName](#) (const [QualifiedName](#) &tName)
Sets the component type name.
- [getComponentName](#) ()
Gets the component name.
- [setComponentName](#) (const [Tstring](#) &sName)
Sets the component name.
- [getComponentId](#) ()
Returns the component identifier.
- [setComponentId](#) (const [BinaryString](#) &id)
Sets the component identifier.
- [toString](#) ()
Returns the string representation of the [TriComponentId](#).
- [equals](#) (const [TriComponentId](#) &cmp)
Returns true if both [TriComponentId](#) objects are equal.
- [cloneComponentId](#) ()
Returns a copy of the [TriComponentId](#).
- [operator<](#) (const [TriComponentId](#) &cmp)

Operator < overload.

8.5.2.5 TriComponentIdList

This abstract type defines a list of TriComponentId elements. It is mapped to the following pure virtual class:

```
class TriComponentIdList {
public:
    virtual ~TriComponentIdList ();
    virtual Tsize size () const =0;
    virtual Tboolean isEmpty () const =0;
    virtual const std::vector< TriComponentId * > & getComponents () const =0;
    virtual const TriComponentId & get (Tsize index) const=0;
    virtual void clear ()=0;
    virtual void add (const TriComponentId &comp)=0;
    virtual Tboolean equals (const TriComponentIdList &cmp1) const =0;
    virtual TriComponentIdList * cloneComponentIdList () const =0;
    virtual Tboolean operator< (const TriComponentIdList &cmp1) const =0;
}

```

8.5.2.5.1 Methods

- [~TriComponentIdList \(\)](#)
Destructor.
- [size \(\)](#)
Returns the number of components in the list.
- [isEmpty \(\)](#)
Returns true if this list contains no components.
- [get \(Tsize index\)](#)
Returns the component at the specified position.
- [clear \(\)](#)
Removes all the components from this list.
- [add \(const TriComponentId &comp\)](#)
Adds a component to the end of this list.
- [equals \(const TriComponentIdList &cmp1\)](#)
Returns true if both TriComponentIdList are equal.
- [cloneComponentIdList \(\)](#)
Returns a copy of the [TriComponentIdList](#).
- [operator< \(const TriComponentIdList &cmp1\)](#)
Operator < overload.

8.5.2.6 TriException

A value of type [TriException](#) is an encoded type and value of an exception that either is to be sent to the SUT or has been received from the SUT. This abstract type is used in procedure based TRI communication operations. It is mapped to the following pure virtual class:

```
class TriException {
public:
    virtual ~TriException ();
    virtual const Tbyte *getData() const=0;
    virtual void setData(const Tbyte *str, Tsize bitLen)=0;
    virtual Tsize getBitsDataLen() const=0;
    virtual Tstring toString() const=0;
    virtual Tboolean equals (const TriException &exc) const=0;
    virtual TriException * cloneException () const =0;
    virtual Tboolean operator< (const TriException &exc) const =0;
}

```

8.5.2.6.1 Methods

- [~TriException](#) ()
Destructor.
- `Tbyte *getData()`
Gets binary string data (array of characters).
- `setData(const Tbyte *str, Tsize bitLen)`
Sets the binary string data (array of characters).
- `getBitsDataLen()`
Gets data length.
- `toString()`
Returns the string representation of the object.
- [equals](#) (const [TriException](#) &exc)
Returns true if both [TriException](#) objects are equal.
- [cloneException](#) ()
Returns a copy of the [TriException](#).
- [operator<](#) (const [TriException](#) &exc)
Operator < overload.

8.5.2.7 TriFunctionId

A value of type [TriFunctionId](#) is the name of a function as specified in the TTCN-3 ATS. It is a derived class from [QualifiedName](#), mapped to the following pure virtual class:

```
class TriFunctionId : QualifiedName {
public:
    virtual ~TriFunctionId ();
    virtual Tboolean equals (const TriFunctionId &fid) const =0;
    virtual TriFunctionId \* cloneFunctionId () const =0;
    virtual Tboolean operator< (const TriFunctionId &fid) const =0;
}
```

8.5.2.7.1 Methods

- [~TriFunctionId](#) ()
Destructor.
- [equals](#) (const [TriFunctionId](#) &fid)
Returns true if both [TriFunctionId](#) objects are equal.
- [cloneFunctionId](#) ()
Returns a copy of the [TriFunctionId](#).
- [operator<](#) (const [TriFunctionId](#) &fid)
Operator < overload.

8.5.2.8 TriMessage

A value of type [TriMessage](#) is encoded test data that either is to be sent to the SUT or has been received from the SUT. It is mapped to following pure virtual class:

```
class TriMessage {
public:
    virtual ~TriMessage ();
    virtual const Tbyte \*getData()const=0;
    virtual void setData(const Tbyte *str, Tsize bitLen)=0;
    virtual Tsize getBitsDataLen()const=0;
    virtual Tstring toString()const=0;
    virtual Tboolean equals (const TriMessage &msg) const =0;
    virtual TriMessage \* cloneMessage () const =0;
    virtual Tboolean operator< (const TriMessage &msg) const =0;
}
```

8.5.2.8.1 Methods

- [~TriMessage](#) ()
Destructor.
- [getData](#)()
Gets binary string data (array of characters).
- [setData](#)(const Tbyte *str, Tsize bitLen)
Sets the binary string data (array of characters).
- [getBitsDataLen](#)()
Gets data length
- [toString](#)()
Returns the string representation of the object
- [equals](#) (const [TriMessage](#) &msg)
Returns true if both [TriMessage](#) objects are equal.
- [cloneMessage](#) ()
Returns a copy of the [TriMessage](#).
- [operator<](#) (const [TriMessage](#) &msg)
Operator < overload.

8.5.2.9 TriParameter

A value of type [TriParameter](#) includes an encoded parameter and a value of [TriParameterPassingMode](#) to represent the passing mode specified for the parameter in the TTCN-3 ATS. . It is mapped to the following pure virtual class:

```
class TriParameter {
public:
    virtual ~TriParameter ();
    virtual const Tstring & getParameterName () const =0;
    virtual void setParameterName (const Tstring &name)=0;
    virtual const TriParameterPassingMode & getParameterPassingMode () const =0;
    virtual void setParameterPassingMode (const TriParameterPassingMode &mode)=0;
    virtual const BinaryString & getEncodedParameter () const =0;
    virtual void setEncodedParameter (const BinaryString &par)=0;
    virtual Tboolean equals (const TriParameter &par) const =0;
    virtual TriParameter * cloneParameter () const =0;
    virtual Tboolean operator< (const TriParameter &par) const =0;
}
```

8.5.2.9.1 Methods

- [~TriParameter](#) ()
Destructor.
- [Tstring](#) & [getParameterName](#) ()
Returns the parameter name as defined in the TTCN-3 specification.
- [setParameterName](#) (const [Tstring](#) &name)
Sets the parameter name.
- [TriParameterPassingMode](#) & [getParameterPassingMode](#) ()
Returns the parameter passing mode of this parameter.
- [setParameterPassingMode](#) (const [TriParameterPassingMode](#) &mode)
Sets the parameter passing mode.
- [BinaryString](#) & [getEncodedParameter](#) ()
Returns the encoded parameter representation of this TriParameter.
- [setEncodedParameter](#) (const [BinaryString](#) &par)
Sets the encoded parameter representation of this TriParameter to parameter.

- `equals` (const `TriParameter` &par)
Returns true if both `TriParameter` objects are equal.
- `cloneParameter` ()
Returns a copy of the `TriParameter`.
- `operator<` (const `TriParameter` &par)
Operator < overload.

8.5.2.10 TriParameterList

A value of type `TriParameterList` is a list of `TriParameterType`. This abstract type is used in procedure based TRI communication operations and for external function calls. It is mapped to the following pure virtual class:

```
class TriParameterList {
public:
    virtual ~TriParameterList ();
    virtual Tsize size () const =0;
    virtual Tboolean isEmpty () const =0;
    virtual const std::vector< const TriParameter * > &getParameters () const =0;
    virtual const TriParameter &get (Tsize index) const =0;
    virtual void clear ()=0;
    virtual void add (const TriParameter &parameter)=0;
    virtual Tboolean equals (const TriParameterList &pml) const =0;
    virtual TriParameterList * cloneParameterList () const =0;
    virtual Tboolean operator< (const TriParameterList &pml) const =0;
}

```

8.5.2.10.1 Methods

- `~TriParameterList` ()
Destructor.
- `size` ()
Returns the number of parameters in this list.
- `isEmpty` ()
Returns true if this list contains no parameters.
- `getParameters` ()
Returns the list of elements
- `get` (Tsize index)
Returns the `TriParameter` at the specified position.
- `clear` ()
Removes all parameters from this `TriParameterList`.
- `add` (const `TriParameter` ¶meter)
Adds parameter to the end of this `TriParameterList`.
- `equals` (const `TriParameterList` &pml)
Returns true if both `TriParameterList` objects are equal.
- `cloneParameterList` ()
Returns a copy of the `TriParameterList`.
- `operator<` (const `TriParameterList` &pml)
Operator < overload.

8.5.2.11 TriParameterPassingMode

A value of type `TriParameterPassingMode` is either in, inout, or out. This abstract type is used in procedure based TRI communication operations and for external function calls. The type is presented as a type safe enumeration. It is mapped to the following class:

```
class TriParameterPassingMode {
public:

```

```

static const TriParameterPassingMode IN;
static const TriParameterPassingMode OUT;
static const TriParameterPassingMode INOUT;
virtual ~TriParameterPassingMode ();
TriParameterPassingMode (const TriParameterPassingMode &);
Tinteger getParameterPassingMode () const;
const Tboolean equals (const TriParameterPassingMode &parmode) const;
}

```

8.5.2.11.1 Methods

- [~TriParameterPassingMode](#) ()
Destructor.
- [TriParameterPassingMode](#) (const [TriParameterPassingMode](#) &)
Copy constructor.
- [getParameterPassingMode](#) ()
Returns the parameterPassingMode value.
- [equals](#) (const [TriParameterPassingMode](#) &parmode)
Returns true if both [TriParameterPassingMode](#) objects are equal.

8.5.2.11.2 Constants

- [IN](#)
Is used to indicate that a [TriParameter](#) is an in parameter.
- [OUT](#)
Is used to indicate that a [TriParameter](#) is an inout parameter.
- [INOUT](#)
Is used to indicate that a [TriParameter](#) is an out parameter.

8.5.2.12 TriPortId

A value of type [TriPortId](#) includes a value of type [TriComponentIdType](#) to represent the component to which the port belongs, a port index (if present), and the port name as specified in the TTCN-3 ATS. The [TriPortId](#) type is mainly required to pass information about the TSI and connections to the TSI from the TE to the SA. It is mapped to the following pure virtual class:

```

class TriPortId {
public:
    virtual ~TriPortId (void)
    virtual const Tstring & getPortName () const =0
    virtual void setPortName (const Tstring &pName)=0
    virtual const TriComponentId & getComponent () const =0
    virtual void setComponent (const TriComponentId &comp)=0
    virtual Tsize getPortIndex () const =0
    virtual void setPortIndex (Tsize index)=0
    virtual const QualifiedName & getPortType () const =0
    virtual void setPortType (const QualifiedName &pType)=0
    virtual Tboolean isArray () const =0
    virtual Tboolean equals (const TriPortId &pPort) const =0
    virtual TriPortId * clonePortId () const =0
    virtual Tboolean operator< (const TriPortId &pPort) const =0
}

```

8.5.2.12.1 Methods

- [~TriPortId](#) (void)
Destructor.
- [getPortName](#) ()
Returns the port name.
- [setPortName](#) (const [Tstring](#) &pName)
Sets the port name.

- [GetComponent](#) ()
Returns the component identifier that this TRI port belongs to.
- [setComponent](#) (const [TriComponentId](#) &comp)
Sets the component identifier that this port belongs to.
- [getPortIndex](#) ()
Gets the port index (0..N). Returns -1 if it isn't part of an array.
- [setPortIndex](#) ([Tsize](#) index)
Sets the port index.
- [getPortType](#) ()
Gets the port type.
- [setPortType](#) (const [QualifiedName](#) &pType)
Sets the port type.
- [isArray](#) ()
Returns true if port is defined as part of an array.
- [equals](#) (const [TriPortId](#) &prt)
Returns true if both [TriPortId](#) objects are equal.
- [clonePortId](#) ()
Returns a copy of the [TriPortId](#).
- [operator<](#) (const [TriPortId](#) &prt)
Operator < overload.

8.5.2.13 TriPortIdList

The value of this type is a list of TriPortIdType elements. It is mapped to the following pure virtual class:

```
class TriPortIdList {
public:
    virtual ~TriPortIdList ();
    virtual Tsize size () const =0;
    virtual bool isEmpty () const =0;
    virtual const std::vector< const TriPortId * > & getPortIds () const =0;
    virtual const TriPortId & get (Tsize index) const =0;
    virtual void clear ()=0;
    virtual void add (const TriPortId &elem)=0;
    virtual Tboolean equals (const TriPortIdList &prtl) const =0;
    virtual TriPortIdList * clonePortIdList () const =0;
    virtual Tboolean operator< (const TriPortIdList &prtl) const =0;
}
```

8.5.2.13.1 Methods

- [~TriPortIdList](#) ()
Destructor.
- [size](#) ()
Returns the number of ports in this list.
- [isEmpty](#) ()
Returns true if port list is empty.
- [getPortIds](#) ()
Returns the ports list.
- [get](#) ([Tsize](#) index)
Gets TriPortIdType element at position index from the port list.

- [clear](#) ()
Removes all `TriPortIdType` elements from the list.
- [add](#) (const [TriPortId](#) &elem)
Adds a `TriPortIdType` element to the list.
- [equals](#) (const [TriPortIdList](#) &prtl)
Returns true if both `TriPortIdList` objects are equal.
- [clonePortIdList](#) ()
Returns a copy of the [TriPortIdList](#).
- [operator<](#) (const [TriPortIdList](#) &prtl)
Operator < overload.

8.5.2.14 TriSignatureId

A value of type `TriSignatureIdType` is the name of a procedure signature as specified in the TTCN-3 ATS. It is a derived class from `QualifiedName`, mapped to the following pure virtual class;

```
class TriSignatureId : QualifiedName {
public:
    virtual ~TriSignatureId ();
    virtual Tboolean equals (const TriSignatureId &sid) const =0;
    virtual TriSignatureId \* cloneSignatureId () const =0;
    virtual Tboolean operator< (const TriSignatureId &sid) const =0;
}
```

8.5.2.14.1 Methods

- [~TriSignatureId](#) ()
Destructor.
- [equals](#) (const [TriSignatureId](#) &sid)
Returns true if both [TriSignatureId](#) objects are equal.
- [cloneSignatureId](#) ()
Returns a copy of the [TriSignatureId](#).
- [operator<](#) (const [TriSignatureId](#) &sid)
Operator < overload.

8.5.2.15 TriStatus

A value of type [TriStatus](#) is either `TRI_OK` or `TRI_ERROR`. This value represents the success or failure of a TRI operation. The type is presented as a type safe enumeration. It is mapped to the following class:

```
class TriStatus {
public:
    static const TriStatus TRI\_OK;
    static const TriStatus TRI\_ERROR;
    virtual ~TriStatus ();
    TriStatus (const TriStatus &);
    Tstring toString () const;
    Tboolean equals (const TriStatus &tStatus) const;
    Tinteger getStatus () const;
}
```

8.5.2.15.1 Methods

- [~TriStatus](#) ()
Destructor.
- [TriStatus](#) (const [TriStatus](#) &)
Copy constructor.
- [toString](#) () const
Returns the status as string value.

- [equals](#) (const [TriStatus](#) &tStatus) const
Returns true if equals.
- [getStatus](#) () const
Returns the status value.

8.5.2.15.2 Constants

- [TRI_OK](#)
Indicates the success of an operation.
- [TRI_ERROR](#)
Indicates the failure of an operation.

8.5.2.16 TriTestCaseId

A value of type [TriTestCaseId](#) is the name of a test case as specified in the TTCN-3 ATS. It is a derived class from [QualifiedName](#), mapped to the following pure virtual class:

```
class TriTestCaseId : QualifiedName {
public:
    virtual ~TriTestCaseId ();
    virtual Tboolean equals (const TriTestCaseId &tcid) const =0;
    virtual TriTestCaseId * cloneTestCaseId () const =0;
    virtual Tboolean operator< (const TriTestCaseId &tcid) const =0;
}
```

8.5.2.16.1 Methods

- [~TriTestCaseId](#) ()
Destructor.
- [equals](#) (const [TriTestCaseId](#) &tcid)
Returns true if both [TriTestCaseId](#) objects are equal.
- [cloneTestCaseId](#) ()
Returns a copy of the [TriTestCaseId](#).
- [operator<](#) (const [TriTestCaseId](#) &tcid)
Operator < overload.

8.5.2.17 TriTimerDuration

A value of type [TriTimerDuration](#) specifies the duration for a timer in seconds. It is mapped to the following pure virtual class:

```
class TriTimerDuration {
public:
    virtual ~TriTimerDuration ()
    virtual Tfloat getDuration () const =0
    virtual void setDuration (Tfloat duration)=0
    virtual Tboolean equals (const TriTimerDuration &timd) const =0
    virtual TriTimerDuration * cloneTimerDuration () const =0
    virtual Tboolean operator< (const TriTimerDuration &timd) const =0
}
```

8.5.2.17.1 Methods

- [~TriTimerDuration](#) ()
Destructor.
- [getDuration](#) ()
Gets the timer duration time.
- [setDuration](#) (Tfloat duration)
Sets the timer duration time from "duration" value.

- [equals](#) (const [TriTimerDuration](#) &timd)
Returns true if both [TriTimerDuration](#) objects are equal.
- [cloneTimerDuration](#) ()
Returns a copy of the [TriTimerDuration](#).
- [operator<](#) (const [TriTimerDuration](#) &timd)
Operator < overload.

8.5.2.18 TriTimerId

A value of type [TriTimerId](#) specifies an identifier for a timer. This type is required for all TRI timer operations. It is mapped to the following pure virtual class:

```
class TriTimerId {
public:
    virtual ~TriTimerId ();
    virtual const Tstring & getTimerName () const =0;
    virtual void setTimerName (const Tstring &tName)=0;
    virtual const Tinteger getTid() const =0;
    virtual Tboolean equals (const TriTimerId &tmid) const =0;
    virtual TriTimerId * cloneTimerId () const =0;
    virtual Tboolean operator< (const TriTimerId &tmid) const =0;
}
```

8.5.2.18.1 Methods

- [~TriTimerId](#) ()
Destructor.
- [getTimerName](#) ()
Returns the timer id name (string value).
- [setTimerName](#) ([Tstring](#) &tName)
Sets the timer id name.
- [equals](#) (const [TriTimerId](#) &tmid)
Returns true if both [TriTimerId](#) objects are equal.
- [cloneTimerId](#) ()
Returns a copy of the [TriTimerId](#).
- [operator<](#) (const [TriTimerId](#) &tmid)
Operator < overload.

8.6 Mapping of interfaces

8.6.1 TriCommunicationSA

This interface consists of operations that are necessary to implement the communication of the TTCN-3 ETS with the SUT. It is mapped to the following pure virtual class:

```
class TriCommunicationSA {
public:
    //Destructor.
    virtual ~TriCommunicationSA ();
    //To reset the System Adaptor
    virtual TriStatus triSAReset ()=0;

    //To execute a test case.
    virtual TriStatus triExecuteTestCase (const TriTestCaseId *testCaseId, const TriPortIdList
    *tsiPortList)=0;

    //To establish a dynamic connection between two ports.
    virtual TriStatus triMap (const TriPortId *comPortId, const TriPortId *tsiPortId)=0;
```

```

//To close a dynamic connection to the SUT for the referenced TSI port.
virtual TriStatus triUnmap (const TriPortId *comPortId, const TriPortId *tsiPortId)=0;

//To end a test case.
virtual TriStatus triEndTestCase ()=0;

//Send operation on a component which has been mapped to a TSI port.
virtual TriStatus triSend (const TriComponentId *componentId, const TriPortId *tsiPortId, const
TriAddress *SUTAddress, const TriMessage *sendMessage)=0;

//Send (broadcast) operation on a component which has been mapped to a TSI port.

virtual TriStatus triSendBC (const TriComponentId *componentId, const TriPortId *tsiPortId, const
TriMessage *sendMessage)=0;

//Send (multicast) operation on a component which has been mapped to a TSI port.
virtual TriStatus triSendMC (const TriComponentId *componentId, const TriPortId *tsiPortId, const
TriAddressList *SUTAddresses, const TriMessage *sendMessage)=0;

//Initiate the procedure call.
virtual TriStatus triCall (const TriComponentId *componentId, const TriPortId *tsiPortId, const
TriAddress *sutAddress, const TriSignatureId *signatureId, const TriParameterList
*parameterList)=0;

//Initiate and broadcast the procedure call.
virtual TriStatus triCallBC (const TriComponentId *componentId, const TriPortId *tsiPortId, const
TriSignatureId *signatureId, const TriParameterList *parameterList)=0;

//Initiate and multicast the procedure call.
virtual TriStatus triCallMC (const TriComponentId *componentId, const TriPortId *tsiPortId, const
TriAddressList *sutAddresses, const TriSignatureId *signatureId, const TriParameterList
*parameterList)=0;

//Issue the reply to a procedure call.
virtual TriStatus triReply (const TriComponentId *componentId, const TriPortId *tsiPortId, const
TriAddress *sutAddress, const TriSignatureId *signatureId, const TriParameterList *parameterList,
const TriParameter *returnValue)=0;

//Broadcast the reply to a procedure call.
virtual TriStatus triReplyBC (const TriComponentId *componentId, const TriPortId *tsiPortId,
const TriSignatureId *signatureId, const TriParameterList *parameterList, const TriParameter
*returnValue)=0;

//Multicast the reply to a procedure call.
virtual TriStatus triReplyMC (const TriComponentId *componentId, const TriPortId *tsiPortId,
const TriAddressList *sutAddresses, const TriSignatureId *signatureId, const TriParameterList
*parameterList, const TriParameter *returnValue)=0;

//Raise an exception to a procedure call.
virtual TriStatus triRaise (const TriComponentId *componentId, const TriPortId *tsiPortId, const
TriAddress *sutAddress, const TriSignatureId *signatureId, const TriException *exc)=0;

//Raise an broadcast an exception to a procedure call.
virtual TriStatus triRaiseBC (const TriComponentId *componentId, const TriPortId *tsiPortId,
const TriSignatureId *signatureId, const TriException *exc)=0;

//Raise an multicast an exception to a procedure call.
virtual TriStatus triRaiseMC (const TriComponentId *componentId, const TriPortId *tsiPortId,
const TriAddressList *sutAddresses, const TriSignatureId *signatureId, const TriException
*exc)=0;

//Initiate the described actions to be taken on the SUT.
virtual TriStatus triSUTactionInformal (const Tstring *description)=0;
}

```

8.6.2 TriCommunicationTE

This interface consists of operations that are necessary to implement the communication of the SUT with the TTCN-3 ETS. It is mapped to the following pure virtual class:

```
class TriCommunicationTE {
public:

    //Destructor.
    virtual ~TriCommunicationTE ();

    //Called by SA after it has received a message from the SUT.
    virtual void triEnqueueMsg (const TriPortId *tsiPortId, const TriAddress *SUTaddress, const
    TriComponentId *componentId, const TriMessage *receivedMessage)=0;

    //Called by SA after it has received a procedure call from the SUT.
    virtual void triEnqueueCall (const TriPortId *tsiPortId, const TriAddress *SUTaddress, const
    TriComponentId *componentId, const TriSignatureId *signatureId, const TriParameterList
    *parameterList)=0;

    //Called by SA after it has received a reply from the SUT.
    virtual void triEnqueueReply (const TriPortId *tsiPortId, const TriAddress *SUTaddress, const
    TriComponentId *componentId, const TriSignatureId *signatureId, const TriParameterList
    *parameterList, const TriParameter *returnValue)=0;

    //Called by SA after it has received an exception from the SUT.
    virtual void triEnqueueException (const TriPortId *tsiPortId, const TriAddress *SUTaddress, const
    TriComponentId *componentId, const TriSignatureId *signatureId, const TriException *exc)=0;
}

```

8.6.3 TriPlatformPA

This interface consists of operations that are necessary to implement the communication of the TTCN-3 ETS with the platform, in which the testcase is running. It is mapped to the following pure virtual class:

```
class TriPlatformPA {
public:

    //Destructor.
    virtual ~TriPlatformPA ();

    //Reset all timing activities which it is currently performing.
    virtual TriStatus triPAReset ()=0;

    //Start the indicated timer with the indicated duration.
    virtual TriStatus triStartTimer (const TriTimerId *timerId, const TriTimerDuration
    *timerDuration)=0;

    //Start the indicated timer with the indicated duration.
    virtual TriStatus triStopTimer (const TriTimerId *timerId)=0;

    //Use the timerId to access the time that elapsed since this timer was started.
    virtual TriStatus triReadTimer (const TriTimerId *timerId, TriTimerDuration *elapsedTime)=0;

    //Use the timerId to access the status of the timer.
    virtual TriStatus triTimerRunning (const TriTimerId *timerId, Tboolean running)=0;

    //For each external function specified in the TTCN-3 ATS implement the behaviour.
    virtual TriStatus triExternalFunction (const TriFunctionId *functionId, TriParameterList
    *parameterList, TriParameter *returnValue)=0;
}

```


8.6.4 TriPlatformTE

This interface consists of operations that are necessary to implement the communication of the platform, in which the testcase is running, with the TTCN-3 ETS. It is mapped to the following pure virtual class:

```
class TriPlatformTE {
public:
    //Destructor.
    virtual ~TriPlatformTE ();

    //Start the indicated timer with the indicated duration.
    virtual void triTimeout (const TriTimerId *timerId);
}

```

9 Use scenarios

This clause contains use scenarios that should help users of the TRI and tool vendors providing the TRI understand the semantics of the operations defined within the present document.

Three scenarios are defined in terms of Message Sequence Charts (MSC). A scenario consists of a TTCN-3 code fragment that uses TTCN-3 communication functions to the SUT as well as timer handling functions. The MSC shows the interactions between the TE, SA, and PA entities together with the SUT.

Please note that the TTCN-3 fragments are not complete, as the main objective of the fragments is the usage of dynamic behaviour. All of the presented scenarios use a common preamble sequence of TRI operations shown in figure 2.

Notice that the MSCs presented in this clause use message pairs to model each TRI operation. The MSC message triMap followed by triMapOK denotes, for example, that the TRI operation triMap has been invoked by the TE and it returns successfully from the SA. TRI operation calls are shown using abstract types and values, and are intended to serve only for illustration purposes. The concrete representation of these parameters in a particular target language is defined in the respective language mappings.

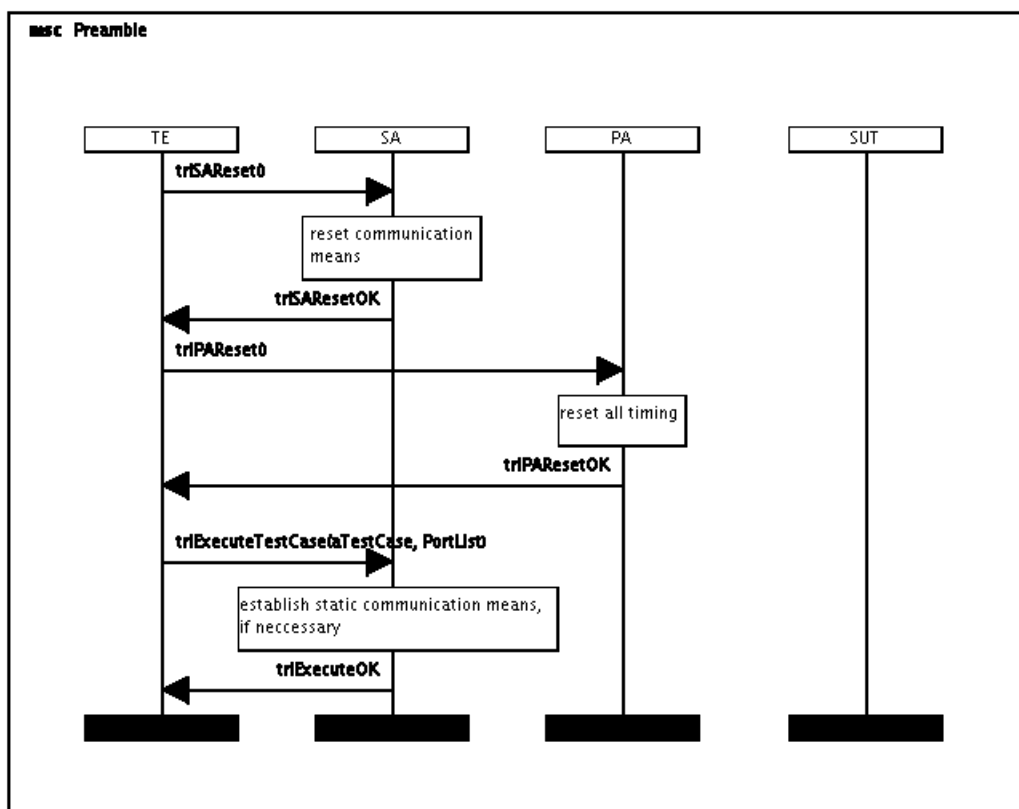


Figure 2: Common MSC Preamble

9.1 First scenario

The first scenario shows some TTCN-3 timer operations, i.e. start and timer running, message based communication operations, i.e. send and receive, as well as connection handling operations, i.e. map and unmap.

9.1.1 TTCN-3 fragment

```
module triScenario1
{
  external function MyFunction();

  type port PortTypeMsg message { inout integer }

  type component MyComponent {
    port PortTypeMsg MyPort;
    timer MyTimer
  }

  type component MyTSI {
    port PortTypeMsg PC01;
  }

  testcase scenario1() runs on MyComponent system MyTSI
  {
    MyPort.clear;
    MyPort.start;
    MyTimer.start(2);

    map(MyComponent: MyPort, system: PC01);
    MyPort.send(integer : 5);
    if (MyTimer.running)
    {
      MyPort.receive(integer:7);
    }
    else
    {
      MyFunction();
    }
    unmap(MyComponent: MyPort, system:PC01);
    MyPort.stop;
  }

  control {
    execute( scenario1() );
  }
}
```

9.1.2 Message sequence chart

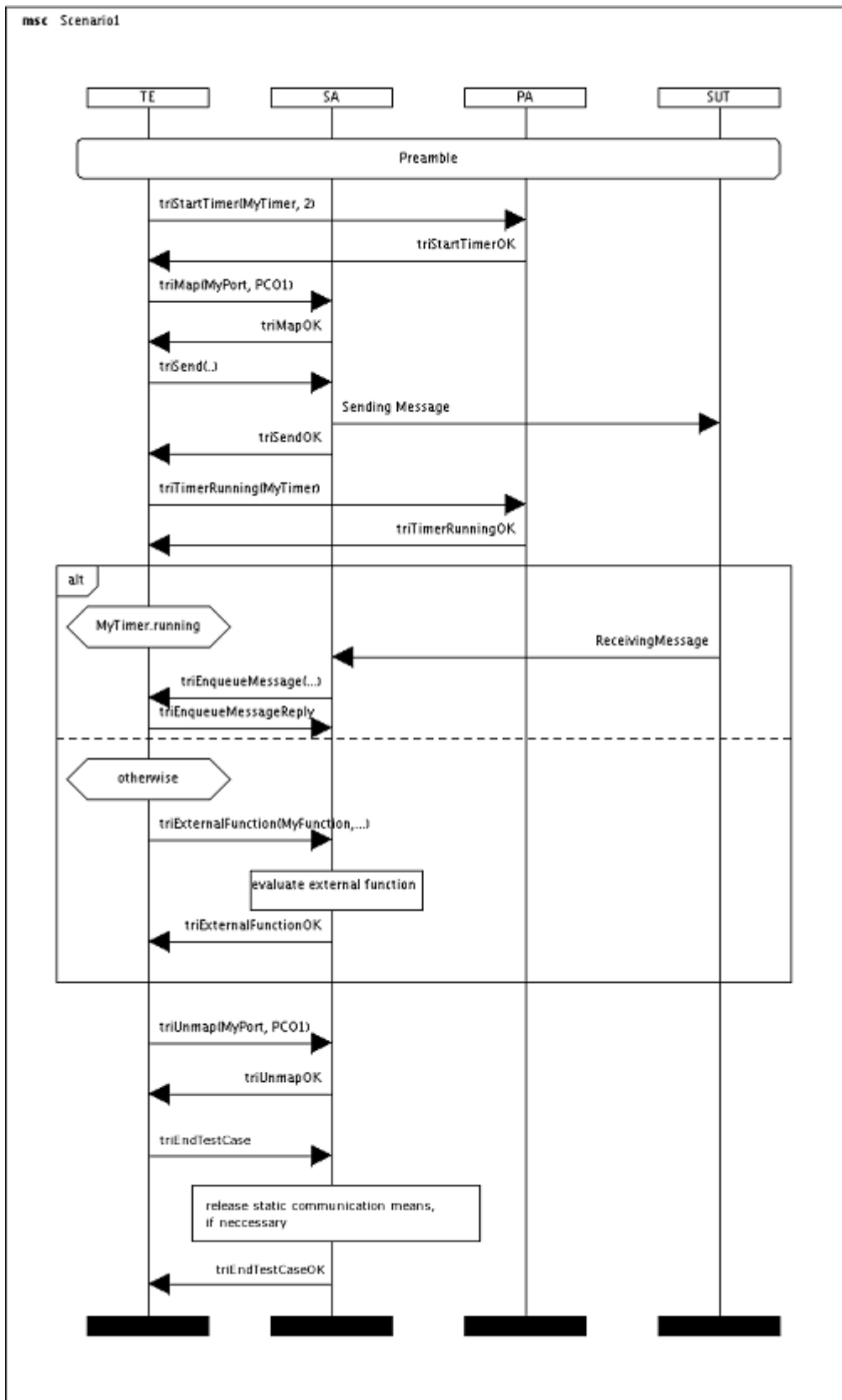


Figure 3: Use Scenario 1

9.2 Second scenario

The second example shows a similar scenario which also uses timed procedure based communication operations which are initiated by the test component `MyComponent`. In this example `MyComponent` is assumed to run as the MTC.

9.2.1 TTCN-3 fragment

```

module triScenario2
{
  signature MyProc ( in float par1, inout float par2)
    exception(MyExceptionType);

  type record MyExceptionType { FieldType1 par1, FieldType2 par2 }

  type port PortTypeProc procedure { out MyProc }

  type component MyComponent {
    port PortTypeProc MyPort;
    timer MyTimer = 7
  }

  testcase scenario2() runs on MyComponent
  {
    var float MyVar;

    MyPort.clear;
    MyPort.start;
    MyTimer.start;

    MyVar := MyTimer.read;

    if (MyVar>5.0) {
      MyPort.call (MyProc:{MyVar, 5.7}, 5);
      alt {
        [] MyPort.getreply(MyProc:{-,MyVar*5}) {}
        [] MyPort.catch (MyProc, MyExceptionType:* ) {}
        [] MyPort.catch (timeout) {}
      }
    }
    MyTimer.stop;
    MyPort.stop;
  }

  control {
    execute( scenario2() );
  }
}

```

9.2.2 Message sequence chart

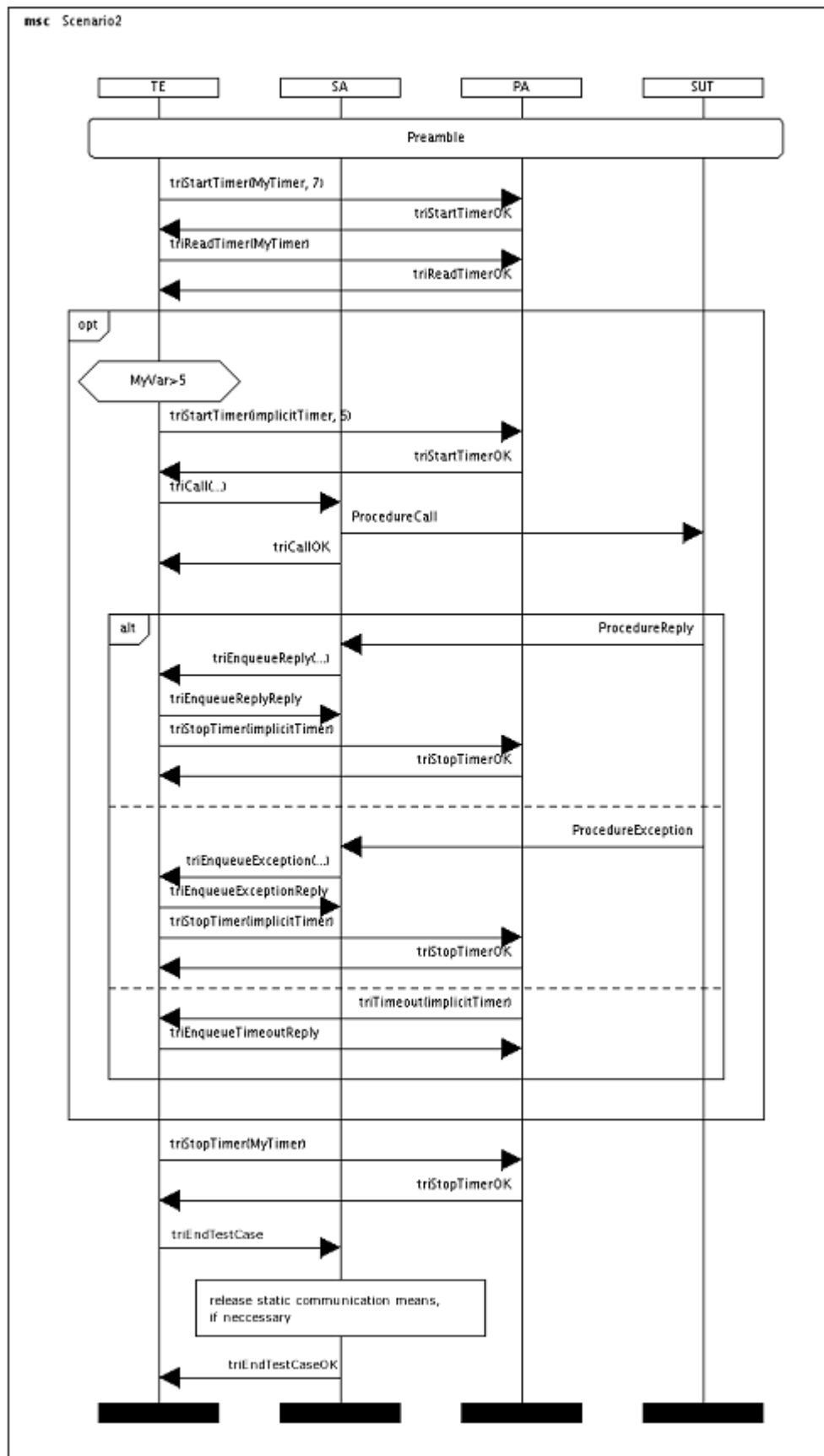


Figure 4: Use Scenario 2

9.3 Third scenario

Use scenario 3 shows the reception of a procedure call as well as a reply and raising of an exception based on this received call. Again `MyComponent` is assumed to run as the MTC. `FieldType1`, `FieldType2`, `p1`, and `p2` are assumed to be defined elsewhere.

9.3.1 TTCN-3 fragment

```

module triScenario3
{
  signature MyProc ( in float par1, inout float par2)
    exception(MyExceptionType);

  type record MyExceptionType { FieldType1 par1, FieldType2 par2 }

  type port PortTypeProc procedure { in MyProc }

  type component MyComponent {
    port PortTypeProc MyPort;
    timer MyTimer = 3
  }

  testcase scenario3(integer x) runs on MyComponent
  {
    MyPort.start;
    MyTimer.start;
    alt
    {
      [] MyPort.getcall(MyProc:{5.0, 6.0})
      {
        MyTimer.stop;
      }
      [x>5] MyTimer.timeout
      {
        MyPort.reply(MyProc:{-, 30.0});
      }
      [x<=5] MyTimer.timeout
      {
        MyPort.raise(MyProc, MyExceptionType:{p1, p2} );
      }
    }
    MyPort.stop;
  }

  control {
    execute( scenario3(4) );
  }
}

```

9.3.2 Message sequence chart

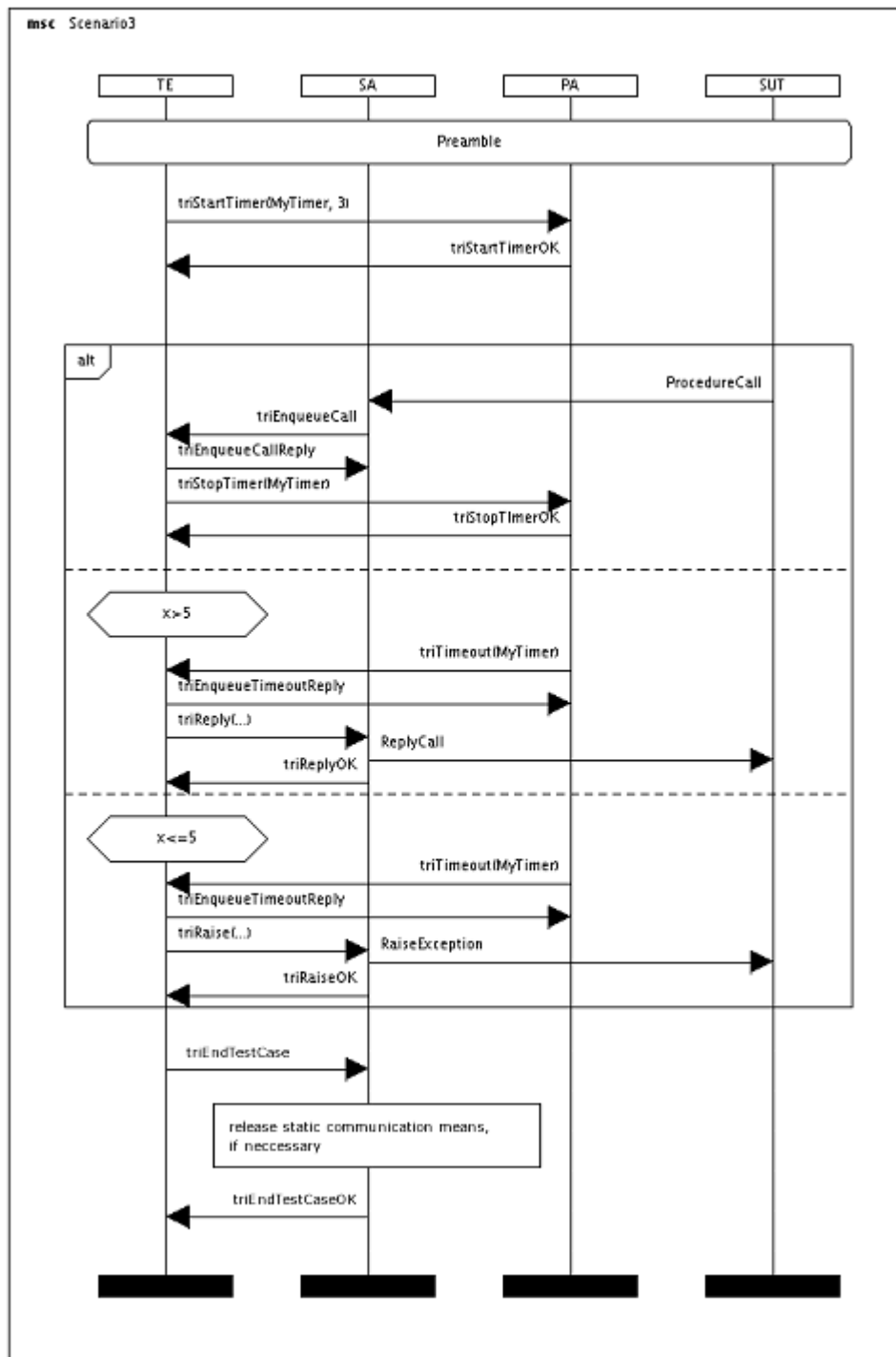


Figure 5: Use Scenario 3

Annex A (normative): IDL Summary

This clause summarizes the IDL definition of TRI operations as defined in clause 5.

```
// *****
// Interface definition for the TTCN-3 Runtime Interface
// *****

module triInterface
{
    //
    // *****
    // Types
    // *****
    //

    // Connection
    native TriPortIdType;
    typedef sequence<TriPortIdType> TriPortIdListType;
    native TriComponentIdType;
    typedef sequence<TriComponentIdType> TriComponentIdListType;

    // Communication
    native TriMessageType;
    native TriAddressType;
    typedef sequence<TriAddressType> TriAddressListType;
    native TriSignatureIdType;
    native TriParameterType;
    typedef sequence<TriParameterType> TriParameterListType;
    native TriExceptionType;

    // Timing
    native TriTimerIdType;
    native TriTimerDurationType;

    // Miscellaneous
    native TriFunctionIdType;
    native TriTestCaseIdType;
    native TriStatusType;

    //
    // *****
    // Interfaces
    // *****
    //

    //
    // *****
    // The communication interface (Ref: TRI-Definition: 5.5)
    // *****
    //
    interface triCommunication
    {
        // Reset operation

        // Ref: TRI-Definition 5.5.1
        TriStatusType triSAReset();

        // Connection handling operations

        // Ref: TRI-Definition 5.5.2.1
        TriStatusType triExecuteTestCase(in TriTestCaseIdType testCaseId,
            in TriPortIdListType tsiPortList);

        // Ref: TRI-Definition 5.5.2.2
        TriStatusType triMap(in TriPortIdType compPortId, in TriPortIdType tsiPortId);

        // Ref: TRI-Definition 5.5.2.3
        TriStatusType triUnmap(in TriPortIdType compPortId, in TriPortIdType tsiPortId);
    }
}
```



```

// Ref: TRI-Definition 5.5.2.4
TriStatusType triEndTestCase();

// Message based communication operations

// Ref: TRI-Definition 5.5.3.1
TriStatusType triSend(in TriComponentIdType componentId, in TriPortIdType tsiPortId,
in TriAddressType SUTAddress, in TriMessageType sendMessage);
// Ref: TRI-Definition 5.5.3.2
TriStatusType triSendBC(in TriComponentIdType componentId, in TriPortIdType tsiPortId,
in TriMessageType sendMessage);
// Ref: TRI-Definition 5.5.3.3
TriStatusType triSendMC(in TriComponentIdType componentId, in TriPortIdType tsiPortId,
in TriAddressListType SUTAddresses, in TriMessageType sendMessage);

// Ref: TRI-Definition 5.5.3.4
void triEnqueueMsg(in TriPortIdType tsiPortId , in TriAddressType SUTAddress,
in TriComponentIdType componentId, in TriMessageType receivedMessage);

// Procedure based communication operations

// Ref: TRI-Definition 5.5.4.1
TriStatusType triCall(in TriComponentIdType componentId, in TriPortIdType tsiPortId,
in TriAddressType SUTAddress, in TriSignatureIdType signatureId,
in TriParameterListType parameterList);
// Ref: TRI-Definition 5.5.4.2
TriStatusType triCallBC(in TriComponentIdType componentId, in TriPortIdType tsiPortId,
in TriSignatureIdType signatureId,
in TriParameterListType parameterList);
// Ref: TRI-Definition 5.5.4.3
TriStatusType triCallMC(in TriComponentIdType componentId, in TriPortIdType tsiPortId,
in TriAddressListType SUTAddresses, in TriSignatureIdType signatureId,
in TriParameterListType parameterList);

// Ref: TRI-Definition 5.5.4.4
TriStatusType triReply(in TriComponentIdType componentId, in TriPortIdType tsiPortId,
in TriAddressType SUTAddress, in TriSignatureIdType signatureId,
in TriParameterListType parameterList, in TriParameterType returnValue );
// Ref: TRI-Definition 5.5.4.5
TriStatusType triReplyBC(in TriComponentIdType componentId, in TriPortIdType tsiPortId,
in TriSignatureIdType signatureId,
in TriParameterListType parameterList, in TriParameterType returnValue );
// Ref: TRI-Definition 5.5.4.6
TriStatusType triReplyMC(in TriComponentIdType componentId, in TriPortIdType tsiPortId,
in TriAddressListType SUTAddresses, in TriSignatureIdType signatureId,
in TriParameterListType parameterList, in TriParameterType returnValue );

// Ref: TRI-Definition 5.5.4.7
TriStatusType triRaise(in TriComponentIdType componentId, in TriPortIdType tsiPortId,
in TriAddressType SUTAddress, in TriSignatureIdType signatureId,
in TriExceptionType exc);
// Ref: TRI-Definition 5.5.4.8
TriStatusType triRaiseBC(in TriComponentIdType componentId, in TriPortIdType tsiPortId,
in TriSignatureIdType signatureId,
in TriExceptionType exc);
// Ref: TRI-Definition 5.5.4.9
TriStatusType triRaiseMC(in TriComponentIdType componentId, in TriPortIdType tsiPortId,
in TriAddressListType SUTAddresses, in TriSignatureIdType signatureId,
in TriExceptionType exc);

// Ref: TRI-Definition 5.5.4.10
void triEnqueueCall(in TriPortIdType tsiPortId, in TriAddressType SUTAddress,
in TriComponentIdType componentId, in TriSignatureIdType signatureId,
in TriParameterListType parameterList );

// Ref: TRI-Definition 5.5.4.11
void triEnqueueReply(in TriPortIdType tsiPortId, in TriAddressType SUTAddress,
in TriComponentIdType componentId, in TriSignatureIdType signatureId,
in TriParameterListType parameterList, in TriParameterType returnValue );

// Ref: TRI-Definition 5.5.4.12
void triEnqueueException(in TriPortIdType tsiPortId, in TriAddressType SUTAddress,
in TriComponentIdType componentId, in TriSignatureIdType signatureId,
in TriExceptionType exc);

```

```
// Miscellaneous operations

// Ref: TRI-Definition 5.5.5.1
TriStateType triSUTactionInformal(in string description);
};

//
// *****
// The platform interface (Ref: TRI-Definition: 5.6)
// *****
//
interface triPlatform
{

    // Reset Operation

    // Ref: TRI-Definition 5.6.1
    TriStatusType triPAReset();

    // Timer handling operations

    // Ref: TRI-Definition 5.6.2.1
    TriStatusType triStartTimer(in TriTimerIdType timerId,
    in TriTimerDurationType timerDuration);

    // Ref: TRI-Definition 5.6.2.2
    TriStatusType triStopTimer(in TriTimerIdType timerId);

    // Ref: TRI-Definition 5.6.2.3
    TriStatusType triReadTimer(in TriTimerIdType timerId,
    out TriTimerDurationType elapsedTime);

    // Ref: TRI-Definition 5.6.2.4
    TriStatusType triTimerRunning(in TriTimerIdType timerId, out boolean running);

    // Ref: TRI-Definition 5.6.2.5
    void triTimeout(in TriTimerIdType timerId);

    // Miscellaneous operations

    // Ref: TRI-Definition 5.6.3.1
    TriStatusType triExternalFunction(in TriFunctionIdType functionId,
    inout TriParameterListType parameterList,
    out TriParameterType returnValue);
};
};
```

Annex B (informative): Bibliography

- CORBA 3.0: "The Common Object Request Broker - Architecture and Specification", OMG Formal Document.
- INTOOL CGI/NPL038 (V2.2): "Generic Compiler/Interpreter interface; GCI Interface Specification", Infrastructural Tools, December 1996.

History

Document history		
V1.1.1	February 2003	Publication
V3.1.1	June 2005	Publication
V3.2.1	February 2007	Publication
V3.3.1	April 2008	Publication
V4.1.1	April 2009	Membership Approval Procedure MV 20090605: 2009-04-07 to 2009-06-05
V4.1.1	June 2009	Publication