International Telecommunication Union

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

Z.165.1

(05/2012)

SERIES Z: LANGUAGES AND GENERAL SOFTWARE ASPECTS FOR TELECOMMUNICATION SYSTEMS

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

Testing and Test Control Notation version 3: TTCN-3 extension package: Extended TRI

Recommendation ITU-T Z.165.1

ITU-T Z-SERIES RECOMMENDATIONS

LANGUAGES AND GENERAL SOFTWARE ASPECTS FOR TELECOMMUNICATION SYSTEMS

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

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

Recommendation ITU-T Z.165.1

Testing and Test Control Notation version 3: TTCN-3 extension package: Extended TRI

Summary

Recommendation ITU-T Z.165.1 defines the extended TRI package of TTCN-3. TTCN-3 can be used for the specification of all types of reactive system tests over a variety of communication ports. Typical areas of application are protocol testing (including mobile and Internet protocols), service testing (including supplementary services), module testing, testing of CORBA based platforms, APIs, etc. TTCN-3 is not restricted to conformance testing and can be used for many other kinds of testing including interoperability, robustness, regression, system and integration testing. The specification of test suites for physical layer protocols is outside the scope of this Recommendation.

This Recommendation is technically aligned with ETSI ES 202 789 V1.1.1 (2012).

History

Edition	Recommendation	Approval	Study Group
1.0	ITU-T Z.165.1	2012-05-29	17

FOREWORD

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

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

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

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

NOTE

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

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

INTELLECTUAL PROPERTY RIGHTS

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

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

Table of Contents

Recommendation ITU-T Z.165.1

Testing and Test Control Notation version 3: TTCN-3 extension package: Extended TRI

1 Scope

This Recommendation defines the Extended TRI package of TTCN-3. TTCN-3 can be used for the specification of all types of reactive system tests over a variety of communication ports. Typical areas of application are protocol testing (including mobile and Internet protocols), service testing (including supplementary services), module testing, testing of CORBA based platforms, APIs, etc. TTCN-3 is not restricted to conformance testing and can be used for many other kinds of testing including interoperability, robustness, regression, system and integration testing. The specification of test suites for physical layer protocols is outside the scope of the present document.

TTCN-3 packages are intended to define additional TTCN-3 concepts, which are not mandatory as concepts in the TTCN-3 core language or in its interfaces TRI and TCI, but which are optional as part of a package which is suited for dedicated applications and/or usages of TTCN-3.

This package defines a more efficient handling of software values by a version of TRI, that does not use binary encoded messages for the communication with the SUT, but uses the values as they are; meaning, e.g., that software objects or serialized data can be passed directly between the SUT and the TE.

While the design of TTCN-3 package has taken into account the consistency of a combined usage of the core language with a number of packages, the concrete usages of and guidelines for this package in combination with other packages is outside the scope of 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 specific references, only the cited version applies. For non-specific references, the latest version of the reference document (including any amendments) applies.

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

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

2.1 Normative references

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

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

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

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

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

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

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

[4] Recommendation ITU-T Z.166 (2012), *Testing and Test Control Notation version 3: TTCN-3 control interface (TCI).*

ETSI ES 201 873-6 (2012), *Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 6: TTCN-3 Control Interface (TCI).*
<http://webapp.etsi.org/workprogram/Report_WorkItem.asp?WKI_ID=35097>

[5] Recommendation ITU-T X.290 (1995), *OSI conformance testing methodology and framework for protocol Recommendations for ITU-T applications – General concepts.*

NOTE – The corresponding ISO/IEC standard is ISO/IEC 9646-1:1994, *Information technology – Open Systems Interconnection – Conformance testing methodology and framework – Part 1: General concepts.*

<http://webstore.iec.ch/webstore/webstore.nsf/ArtNum_PK/39613?OpenDocument>

2.2 Informative references

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

[i.1] Recommendation ITU-T Z.162 (2012), *Testing and Test Control Notation version 3: TTCN-3 tabular presentation format (TFT).*

ETSI ES 201 873-2 (2007), *Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 2: TTCN-3 Tabular presentation Format (TFT).*
<http://webapp.etsi.org/workprogram/Report_WorkItem.asp?WKI_ID=25471>

[i.2] Recommendation ITU-T Z.163 (2012), *Testing and Test Control Notation version 3: TTCN-3 graphical presentation format (GFT).*

ETSI ES 201 873-3 (2007), *Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 3: TTCN-3 Graphical presentation Format (GFT).*
<http://webapp.etsi.org/workprogram/Report_WorkItem.asp?WKI_ID=25472>

[i.3] Recommendation ITU-T Z.167 (2012), *Testing and Test Control Notation version 3: TTCN-3 mapping from ASN.1*.

ETSI ES 201 873-7 (2012), *Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 7: Using ASN.1 with TTCN-3.*
<http://webapp.etsi.org/workprogram/Report_WorkItem.asp?WKI_ID=35098>

[i.4] Recommendation ITU-T Z.168 (2012), *Testing and Test Control Notation version 3: TTCN-3 mapping from CORBA IDL*.

ETSI ES 201 873-8 (2012), *Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 8: The IDL to TTCN-3 Mapping*.
<http://webapp.etsi.org/workprogram/Report_WorkItem.asp?WKI_ID=35099>

[i.5] Recommendation ITU-T Z.169 (2012), *Testing and Test Control Notation version 3: TTCN-3 mapping from XML data definition*.

ETSI ES 201 873-9 (2012), *Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 9: Using XML schema with TTCN-3*.
<http://webapp.etsi.org/workprogram/Report_WorkItem.asp?WKI_ID=35100>

[i.6] Recommendation ITU-T Z.170 (2012), *Testing and Test Control Notation version 3: TTCN-3 documentation comment specification.*

ETSI ES 201 873-10 (2012), *Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 10: TTCN-3 Documentation Comment Specification*.
<http://webapp.etsi.org/workprogram/Report_WorkItem.asp?WKI_ID=35101>

[i.7] ETSI ES 202 781 (2010), *Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; TTCN-3 Language Extensions: Configuration and Deployment Support*.
<http://pda.etsi.org/pda/home.asp?wki_id=YONHi.nwr3GIPIMLJ.Y6I>

[i.8] ETSI ES 202 784 (2011), *Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; TTCN-3 Language Extensions: Advanced Parameterization*.
<http://pda.etsi.org/pda/home.asp?wki_id=u-e'6-ZnV9697F68knFt->

[i.9] ETSI ES 202 785 (2011), *Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; TTCN-3 Language Extensions: Behaviour Types*.
<http://pda.etsi.org/pda/home.asp?wki_id=cL74ubdUfkwzx-wx,-1lZ>

[i.10] ETSI ES 202 782 (2010), *Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; TTCN-3 Language Extensions: TTCN-3 Performance and Real Time Testing*.
<http://pda.etsi.org/pda/home.asp?wki_id=3rxD8efqMX'-6-33q3YBK>

3 Definitions and abbreviations

3.1 Definitions

For the purposes of this Recommendation, the terms and definitions given in [1], [2], [3], [4] and [5] apply.

3.2 Abbreviations and acronyms

For the purposes of this Recommendation, the abbreviations and acronyms given in [1], [2], [3], [4], [5] and the following apply:

XTRI Extended TRI

4 Package conformance and compatibility

The package has no package tag as the choice to use TRI and/or XTRI affects the test adaptor only, but not the test specifications in TTCN-3.

For an implementation claiming to conform to this package version, all features specified in the present document shall be implemented consistently with the requirements given in the present document and in [1] and [2].

The package presented in the present document is compatible to:

[1]

[2]

[4]

[i.3]

[i.4]

[i.5]

[i.6]

If later versions of those parts are available and should be used instead, the compatibility of the package defined in the present document has to be checked individually.

The package defined in the present document is also compatible to, and can be used together with, the following packages:

ES 202 781 [i.7] (V1.1.1)

ES 202 782 [i.10] (V1.1.1)

ES 202 784 [i.8] (V1.2.1)

ES 202 785 [i.9] (V1.2.1)

If later versions of those packages are available and should be used instead, the compatibility to the package defined in the present document has to be checked individually.

5 Package concepts for the core language

Not applicable.

6 Package semantics

Not applicable.

7 TRI extensions for the package

Historically, TTCN has been used to test communication protocols which typically use encoded messages. This has been reflected in the TRI SA and TCI CD design of TTCN-3 by encoding and decoding messages to and from bitstrings. However, TTCN-3 also supports signature-based communication for which the transformation of objects into bitstrings and vice versa is cumbersome. Furthermore, some protocols use also structured messages for which the bitstring encoding is not helpful.

Therefore, an alternative API is being defined in this extension package of TTCN-3 along which TTCN-3 values can be directly passed to/from the SUT. It is defined by redefining the operations in TRI SA and PA as follows.

7.1 Changes to clause 5.5.2 of [3], Connection handling operations

5.5.2.3 `triMapParam` → `xtriMapParam`

Signature	`TriStatusType xtriMap(in TriPortIdType compPortId,`
	`in TriPortIdType tsiPortId,`
	`in TciParameterListType paramList)`

In Parameters	`compPortId` identifier of the test component port to be mapped `tsiPortId` identifier of the test system interface port to be mapped `paramList` parameters of the parameterized map
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.

5.5.2.5 `triUnmapParam` → `xtriUnmapParam`

Signature	`TriStatusType xtriUnmap(in TriPortIdType compPortId,`
	`in TriPortIdType tsiPortId,`
	`in TciParameterListType paramList)`

In Parameters	`compPortId` identifier of the test component port to be unmapped `tsiPortId` identifier of the test system interface port to be unmapped `paramList` parameters of the parameterized map
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 closed by the test system.

7.2 Changes to clause 5.5.3 of [3], Message based communication operations

5.5.3.1 `triSend` → `xtriSend`

Signature	`TriStatusType xtriSend(in TriComponentIdType componentId,`
	`in TriPortIdType tsiPortId,`
	`in Value SUTaddress,`
	`in Value 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 value within the SUT `sendMessage` the value 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` → `xtriSendBC`

Signature	`TriStatusType xtriSendBC(in TriComponentIdType componentId,`
	`in TriPortIdType tsiPortId,`
	`in Value 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 value 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` → `xtriSendMC`

Signature	`TriStatusType xtriSendMC(in TriComponentIdType componentId,`
	`in TriPortIdType tsiPortId,`
	`in TciValueList SUTaddresses,`
	`in Value 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 address values within the SUT `sendMessage` the values 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` → `xtriEnqueueMsg`

Signature	`void xtriEnqueueMsg(in TriPortIdType tsiPortId, in any SUTaddress, in TriComponentIdType componentId, in any 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 value within the SUT `componentId` identifier of the receiving test component `receivedMessage` the received value
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.

7.3 Addition to clause 5.5.3 of [3], Message based communication operations

In order to interpret unknown values along a type hypothesis, an additional xtriConvert operation is defined. It can be used in all cases where the type of the incoming value is not known. Please note that typically the value type is known in procedure-based communication and sometimes in message-based communication.

5.5.3.5 `xtriConvert`

Signature	`Value xtriConvert(in any value, in Type typeHypothesis)`
In Parameters	`value` the value to be converted `typeHypothesis` the type hypothesis
Out Parameters	n.a.
Return Value	Returns the converted value, if the value is of a compatible type as the `typeHypothesis`, else the distinct value `null`.
Constraints	This operation shall be called whenever the TE has to convert a value. The TE might convert immediately after reception of the value, or might for performance considerations postpone the conversion until the actual access to the value.
Effect	This operation converts a `value` and returns a value according to the type hypothesis if it matches. The `typeHypothesis` determines whether the value can be converted. If not, the distinct `null` value shall be returned.

7.4 Changes to clause 5.5.4 of [3], Procedure based communication operations

5.5.4.1 `triCall` → `xtriCall`

Signature	`TriStatusType xtriCall(in TriComponentIdType componentId,`
	`in TriPortIdType tsiPortId,`
	`in Value SUTaddress,`
	`in TriSignatureIdType signatureId,`
	`in TciParameterListType 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 *in* and *inout* 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 *out* 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 *not* 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` → `xtriCallBC`

Signature	`TriStatusType xtriCallBC(in TriComponentIdType componentId,`
	`in TriPortIdType tsiPortId,`
	`in TriSignatureIdType signatureId,`
	`in TciParameterListType 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 `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 `triCallBC` 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 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 *in* and *inout* 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 and broadcast the procedure call corresponding to the signature identifier `signatureId` and the TSI port `tsiPortId`. The `triCallBC` 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 *out* 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 *not* 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`.
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` → `xtriCallMC`

Signature	`TriStatusType xtriCallMC(in TriComponentIdType componentId,`
	`in TriPortIdType tsiPortId,`
	`in TciValueList SUTaddresses,`
	`in TriSignatureIdType signatureId,`
	`in TciParameterListType 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 `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.

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	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. ~~All *in* and *inout* 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 and multicast the procedure call corresponding to the signature identifier `signatureId` and the TSI port `tsiPortId`. The `triCallMC` 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 *out* 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 *not* 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`.
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` → `xtriReply`

Signature	`TriStatusType xtriReply(in TriComponentIdType componentId,`
	`in TriPortIdType tsiPortId,`
	`in Value SUTaddress,`
	`in TriSignatureIdType signatureId,`
	`in TciParameterListType parameterList,`
	`in Value 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 `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 `returnValue` (optional) ~~encoded~~ return value of the procedure call
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	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. ~~All *out* and *inout* 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 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 issue the reply to a procedure call corresponding to the signature identifier `signatureId` and the TSI port `tsiPortId`. 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 *in* parameter or an undefined return value is different from null.

5.5.4.5 `triReplyBC` → `xtriReplyBC`

Signature	`TriStatusType xtriReplyBC(in TriComponentIdType componentId,`
	`in TriPortIdType tsiPortId,`
	`in TriSignatureIdType signatureId,`
	`in TciParameterListType parameterList,`
	`in Value 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 `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 `triReplyBC` 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 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 *out* and *inout* 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 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 broadcast the reply to procedure calls corresponding to the signature identifier `signatureId` and the TSI port `tsiPortId`. 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 *in* parameter or an undefined return value is different from null.

5.5.4.6 `triReplyMC` → `xtriReplyMC`

Signature	`TriStatusType xtriReplyMC(in TriComponentIdType componentId,`
	`in TriPortIdType tsiPortId,`
	`in TciValueList SUTaddresses,`
	`in TriSignatureIdType signatureId,`
	`in TciParameterListType parameterList,`
	`in Value 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 *out* and *inout* 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 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 *in* parameter or an undefined return value is different from null.

5.5.4.7 `triRaise` → `xtriRaise`

Signature	`TriStatusType xtriRaise(in TriComponentIdType componentId,`
	`in TriPortIdType tsiPortId,`
	`in Value SUTaddress,`
	`in TriSignatureIdType signatureId,`
	`in Value 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` → `xtriRaiseBC`

Signature	`TriStatusType xtriRaiseBC(in TriComponentIdType componentId,`
	`in TriPortIdType tsiPortId,`
	`in TriSignatureIdType signatureId,`
	`in Value 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` → `xtriRaiseMC`

Signature	`TriStatusType xtriRaiseMC(in TriComponentIdType componentId,`
	`in TriPortIdType tsiPortId,`
	`in TciValueList SUTaddresses,`
	`in TriSignatureIdType signatureId,`
	`in Value 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` → `xtriEnqueueCall`

Signature	`void xtriEnqueueCall(in TriPortIdType tsiPortId,`
	`in any SUTaddress,`
	`in TriComponentIdType componentId,`
	`in TriSignatureIdType signatureId,`
	`in TciParameterListType 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` → `xtriEnqueueReply`

Signature	`void xtriEnqueueReply(in TriPortIdType tsiPortId,`
	`in any SUTaddress,`
	`in TriComponentIdType componentId,`
	`in TriSignatureIdType signatureId,`
	`in TciParameterListType parameterList,`
	`in Value 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` → `xtriEnqueueException`

Signature	`void xtriEnqueueException(in TriPortIdType tsiPortId,`
	`in any SUTaddress,`
	`in TriComponentIdType componentId,`
	`in TriSignatureIdType signatureId,`
	`in any 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.

7.5 Changes to clause 5.6.3 of [3], Miscellaneous operations

5.6.3.1 `triExternalFunction` → **`xtriExternalFunction`**

Signature	`TriStatusType xtriExternalFunction(`
	`  in TriFunctionIdType functionId,`
	`  inout TciParameterListType parameterList,`
	`  out Value 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 *in* and *inout* function parameters contain encoded values.~~ No error shall be indicated by the PA in case the value of any *out* 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 *in* and *inout* function parameters in `parameterList`, evaluate the external function using the values of these parameters, and compute values for *inout* and *out* parameters in `parameterList`. The operation shall then return ~~encoded~~ values for all *inout* and *out* 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 *blocking.* 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.

7.6 Changes to clause 6 of [3], Java language mapping

Addition of the following clause in clause 6.3 of [3], Type mapping.

6.3.3 Any type mapping

The IDL any type is represented by Java java.lang.Object.

6.5.2.1 Changes to triCommunicationSA

The extensions to the `triCommunicationSA` interface is mapped to the following interface:

```
// TriCommunication
// TE -> SA
package org.etsi.ttcn.xtri;
public interface xTriCommunicationSA {
    public TriStatus xtriMapParam(TriPortId compPortId, TriPortId tsiPortId,
            in TciParameterListType paramList);
    // Ref: TRI-Definition 5.5.2.3
    public TriStatus xtriUnmapParam(TriPortId compPortId, TriPortId tsiPortId,
            in TciParameterListType paramList);
    // Ref: TRI-Definition 5.5.2.4

    // Message based communication operations
    // Ref: TRI-Definition 5.5.3.1
    public TriStatus xtriSend(TriComponentId componentId, TriPortId tsiPortId,
            Value sutAddress, Value sendMessage);
    // Ref: TRI-Definition 5.5.3.2
    public TriStatus xtriSendBC(TriComponentId componentId, TriPortId tsiPortId,
            Value sendMessage);
    // Ref: TRI-Definition 5.5.3.3
    public TriStatus xtriSendMC(TriComponentId componentId, TriPortId tsiPortId,
            TciValueList sutAddresses, Value sendMessage);

    // Procedure based communication operations
    // Ref: TRI-Definition 5.5.4.1
    public TriStatus xtriCall(TriComponentId componentId,
            TriPortId tsiPortId, Value sutAddress,
            TriSignatureId signatureId, TciParameterList parameterList);
    // Ref: TRI-Definition 5.5.4.2
    public TriStatus xtriCallBC(TriComponentId componentId,
            TriPortId tsiPortId,
            TriSignatureId signatureId, TciParameterList parameterList);
    // Ref: TRI-Definition 5.5.4.3
    public TriStatus xtriCallMC(TriComponentId componentId,
            TriPortId tsiPortId, TciValueList sutAddresses,
            TriSignatureId signatureId, TciParameterList parameterList);

    // Ref: TRI-Definition 5.5.4.4
    public TriStatus xtriReply(TriComponentId componentId,
            TriPortId tsiPortId, Value sutAddress,
            TriSignatureId signatureId, TciParameterList parameterList,
            Value returnValue);
    // Ref: TRI-Definition 5.5.4.5
    public TriStatus xtriReplyBC(TriComponentId componentId,
            TriPortId tsiPortId,
            TriSignatureId signatureId, TciParameterList parameterList,
            Value returnValue);
    // Ref: TRI-Definition 5.5.4.6
    public TriStatus xtriReplyMC(TriComponentId componentId,
            TriPortId tsiPortId, TciValueList sutAddresses,
            TriSignatureId signatureId, TciParameterList parameterList,
            Value returnValue);

    // Ref: TRI-Definition 5.5.4.7
    public TriStatus xtriRaise(TriComponentId componentId, TriPortId tsitPortId,
            Value sutAddress,
            TriSignatureId signatureId,
            Value exc);
    // Ref: TRI-Definition 5.5.4.8
    public TriStatus xtriRaiseBC(TriComponentId componentId,
            TriPortId tsitPortId,
            TriSignatureId signatureId,
            Value exc);
    // Ref: TRI-Definition 5.5.4.9
    public TriStatus xtriRaiseMC(TriComponentId componentId, TriPortId tsitPortId,
            TciValueList sutAddresses,
            TriSignatureId signatureId,
```

```
            Value exc);

}
```

6.5.2.2 Changes to triCommunicationTE

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

```
// TriCommunication
// SA -> TE
package org.etsi.ttcn.xtri;
public interface xTriCommunicationTE {
    // Message based communication operations
    // Ref: TRI-Definition 5.5.3.4
    public void xtriEnqueueMsg(TriPortId tsiPortId,
            Value sutAddress, TriComponentId componentId,
            Object receivedMessage);

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

    // Ref: TRI-Definition 5.5.4.11
    public void xtriEnqueueReply(TriPortId tsiPortId, Object sutAddress,
            TriComponentId componentId, TriSignatureId signatureId,
            TciParameterList parameterList, Value returnValue);

    // Ref: TRI-Definition 5.5.4.12
    public void xtriEnqueueException(TriPortId tsiPortId,
            Object sutAddress, TriComponentId componentId,
            TriSignatureId signatureId, Object exc);

    // Miscellaneous operations
    // Ref: TRI-Definition 5.5.3.5
    public Value xtriConvert(in Object value, in Type typeHypothesis);

}
```

6.5.3.1 Changes to TriPlatformPA

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

```
// TriPlatform
// TE -> PA
package org.etsi.ttcn.xtri;
public interface xTriPlatformPA {
    // Ref: TRI-Definition 5.6.3.1
    public TriStatus xtriExternalFunction(TriFunctionId functionId,
     TciParameterList parameterList, Value returnValue);
}
```

7.7 Changes to clause 7 of [3], C language mapping

7.2.1 Changes to Abstract type mapping

TRI ADT	ANSI C Representation	Notes and comments

any	typedef enumerated {
	  e_char = 1, // character
	  e_unsigned_char = 2, // unsigned char
	  e_signed_char = 3, // signed char
	
	  e_short = 4, // short signed integer
	  e_short_int = 5, // short signed integer
	  e_signed_short = 6, // short signed integer
	  e_signed_short_int = 7, // short signed integer
	  e_unsigned_short = 8, // unsigned short
	  e_unsigned_short_int = 9, // unsigned short integer	

TRI ADT	ANSI C Representation	Notes and comments

	e_int = 10, // integer
	e_signed_int = 11, // signed integer
	e_unsigned = 12, // unsigned
	e_unsigned_int = 13, // unsigned integer
	
	e_long = 14, // long integer
	e_long_int = 15, // long integer
	e_signed_long = 16, // signed long integer
	e_signed_long_int = 17, // signed long integer
	e_unsigned_long = 18, // unsigned long integer
	e_unsigned_long_int = 19, // unsigned long integer
	
	e_long_long = 20, // long long integer
	e_long_long_int = 21, // long long integer
	e_signed_long_long = 22, // signed long long integer
	e_signed_long_long_int = 23, // signed long long integer
	e_unsigned_long_long = 24, // unsigned long long integer
	e_unsigned_long_long_int = 25, // unsigned long long integer
	
	e_float = 26, // float
	e_double = 27, // double
	e_long_double = 28, // long double
	
	e_ptr = 29 // void *
	} type_kind;
	
	typedef void *value;
	
	typedef struct {
	type_kind tag,
	value val
	} Object;	

7.2.4 Changes to TRI operation mapping

```
TriStatus xtriMapParam
 (const TriPortId* compPortId,
 const TriPortId* tsiPortId,
 const TciParameterList* parameterList)
TriStatus xtriUnmapParam
(const TriPortId* compPortId,
 const TriPortId* tsiPortId,
 const TciParameterList* parameterList)
TriStatus xtriSend
(const TriComponentId* componentId,
 const TriPortId* tsiPortId,
 const Value* sutAddress,
 const Value* sendMessage)
TriStatus xtriSendBC
(const TriComponentId* componentId,
 const TriPortId* tsiPortId,
 const Value* sendMessage)
TriStatus xtriSendMC
(const TriComponentId* componentId,
 const TriPortId* tsiPortId,
 const TciValueList* sutAddresses,
 const Value* sendMessage)
void xtriEnqueueMsg
(const TriPortId* tsiPortId,
 const Object* sutAddress,
 const TriComponentId* componentId,
 const Object* receivedMessage)
TriStatus xtriCall
(const TriComponentId* componentId,
 const TriPortId* tsiPortId,
 const Value* sutAddress,
```

```
 const TriSignatureId* signatureId,
 const TciParameterList* parameterList)
TriStatus xtriCallBC
(const TriComponentId* componentId,
 const TriPortId* tsiPortId,
 const TriSignatureId* signatureId,
 const TciParameterList* parameterList)
TriStatus xtriCallMC
(const TriComponentId* componentId,
 const TriPortId* tsiPortId,
 const TciValueList* sutAddresses,
 const TriSignatureId* signatureId,
 const TciParameterList* parameterList)
TriStatus xtriReply
 (const TriComponentId* componentId,
 const TriPortId* tsiPortId,
 const Value* sutAddress,
 const TriSignatureId* signatureId,
 const TciParameterList* parameterList,
 const Value* returnValue)
TriStatus xtriReplyBC
 (const TriComponentId* componentId,
 const TriPortId* tsiPortId,
 const TriSignatureId* signatureId,
 const TciParameterList* parameterList,
 const Value* returnValue)
TriStatus xtriReplyMC
 (const TriComponentId* componentId,
 const TriPortId* tsiPortId,
 const TciValueList* sutAddresses,
 const TriSignatureId* signatureId,
 const TciParameterList* parameterList,
 const Value* returnValue)
TriStatus xtriRaise
 (const TriComponentId* componentId,
 const TriPortId* tsiPortId,
 const Value* sutAddress,
 const TriSignatureId* signatureId,
 const Value* exception)
TriStatus xtriRaiseBC
 (const TriComponentId* componentId,
 const TriPortId* tsiPortId,
 const TriSignatureId* signatureId,
 const Value* exception)
TriStatus xtriRaiseMC
 (const TriComponentId* componentId,
 const TriPortId* tsiPortId,
 const TciValueList* sutAddresses,
 const TriSignatureId* signatureId,
 const Value* exception)
void xtriEnqueueCall
 (const TriPortId* tsiPortId,
 const Object* sutAddress,
 const TriComponentId* componentId,
 const TriSignatureId* signatureId,
 const TciParameterList* parameterList)
void xtriEnqueueReply
 (const TriPortId* tsiPortId,
 const Object* sutAddress,
 const TriComponentId* componentId,
 const TriSignatureId* signatureId,
 const TciParameterList* parameterList,
 const Value* returnValue)
void xtriEnqueueException
 (const TriPortId* tsiPortId,
 const Object* sutAddress,
 const TriComponentId* componentId,
 const TriSignatureId* signatureId,
 const Object* exception)
TriStatus xtriExternalFunction
 (const TriFunctionId* functionId,
```

```
 TciParameterList* parameterList,
 Value* returnValue)
Value xtriConvert
 (Object* value,
 Type* typeHypothesis)
```

7.8 Changes to clause 8 of [3], C++ language mapping

Addition of the following clause in clause 8.5 of [3], Type mapping.

8.5.3 Any type mapping

The IDL any type is represented by struct type of type tag and value:

```
typedef enumerated {
  e_char = 1,                       // character
  e_unsigned_char = 2,              // unsigned char
  e_signed_char = 3,                // signed char

  e_short = 4,                      // short signed integer
  e_short_int = 5,                  // short signed integer
  e_signed_short = 6,               // short signed integer
  e_signed_short_int = 7,           // short signed integer
  e_unsigned_short = 8,             // unsigned short
  e_unsigned_short_int = 9,         // unsigned short integer

  e_int = 10,                       // integer
  e_signed_int = 11,                // signed integer
  e_unsigned = 12,                  // unsigned
  e_unsigned_int = 13,              // unsigned integer

  e_long = 14,                      // long integer
  e_long_int = 15,                  // long integer
  e_signed_long = 16,               // signed long integer
  e_signed_long_int = 17,           // signed long integer
  e_unsigned_long = 18,             // unsigned long integer
  e_unsigned_long_int = 19,         // unsigned long integer

  e_long_long = 20,                 // long long integer
  e_long_long_int = 21,             // long long integer
  e_signed_long_long = 22,          // signed long long integer
  e_signed_long_long_int = 23,      // signed long long integer
  e_unsigned_long_long = 24,        // unsigned long long integer
  e_unsigned_long_long_int = 25,    // unsigned long long integer

  e_float = 26,                     // float
  e_double = 27,                    // double
  e_long_double = 28,               // long double

  e_ptr = 29                        // void *
} type_kind;

typedef void *value;

typedef struct {
  type_kind tag,
  value val
} Object;
```

8.6.1 Changes to TriCommunicationSA

The extensions to the `triCommunicationSA` interface is mapped to the following interface:

```
class xTriCommunicationSA {
public:

    //Destructor.
    virtual ~xTriCommunicationSA ();
    //To reset the System Adaptor
    virtual xTriStatus triSAReset ()=0;

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

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

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

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

    //Send (multicast) operation on a component which has been mapped to a TSI port.
    virtual TriStatus xtriSendMC (const TriComponentId *componentId, const TriPortId *tsiPortId,
    const TciValueList *SUTaddresses, const TciValue *sendMessage)=0;

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

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

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

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

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

    //Multicast the reply to a procedure call.
    virtual TriStatus xtriReplyMC (const TriComponentId *componentId, const TriPortId *tsiPortId,
    const TciValueList *sutAddresses, const TriSignatureId *signatureId, const TciParameterList
    *parameterList, const TciValue *returnValue)=0;

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

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

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

}
```

8.6.2 Changes to TriCommunicationTE

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

```
class xTriCommunicationTE {
public:

    //Destructor.
    virtual ~xTriCommunicationTE ();

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

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

```
    //Called by SA after it has received a reply from the SUT.
    virtual void xtriEnqueueReply (const TriPortId *tsiPortId, const Object *SUTaddress, const
    TriComponentId *componentId, const TriSignatureId *signatureId, const TciParameterList
    *parameterList, const TciValue *returnValue)=0;

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

    // Miscellaneous operations
    virtual TciValue xtriConvert(const Object *value, const TciType *typeHypothesis)=0;

}
```

8.6.3 Changes to TriPlatformPA

The extensions to the `TriPlatformPA` interface is mapped to the following interface:

```
class xTriPlatformPA {
public:

    //Destructor.
    virtual ~TriPlatformPA ();

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

7.9 Changes to clause 9 of [3], C# language mapping

Addition of the following clause in clause 9.4 of [3], Type mapping.

9.4.3 Any type mapping

The IDL any type is represented by C# Object.

9.5.2.1 Changes to ITriCommunicationSA

The extensions to the **`ITriCommunicationSA`** interface are defined as follows:

```
public interface IXTriCommunicationSA {
    // Reset operation
    // Ref: TRI-Definition 5.5.1
    TriStatus XTriMapParam(ITriPortId compPortId, ITriPortId tsiPortId,
        ITciValueList parameterList);
    // Ref: TRI-Definition 5.5.2.3
    TriStatus XTriUnmapParam(ITriPortId compPortId, ITriPortId tsiPortId,
        ITciValueList parameterList);
    // Ref: TRI-Definition 5.5.2.4

    // Message based communication operations
    // Ref: TRI-Definition 5.5.3.1
    TriStatus XTriSend(ITriComponentId componentId, ITriPortId tsiPortId,
        ITciValue address, ITciValue sentMessage);
    // Ref: TRI-Definition 5.5.3.2
    TriStatus XTriSendBC(ITriComponentId componentId, ITriPortId tsiPortId,
        ITciValue sentMessage);
    // Ref: TRI-Definition 5.5.3.3
    TriStatus XTriSendMC(ITriComponentId componentId, ITriPortId tsiPortId,
        ITciValueList addresses, ITciValue sentMessage);

    // Procedure based communication operations
    // Ref: TRI-Definition 5.5.4.1
    TriStatus XTriCall(ITriComponentId componentId, ITriPortId tsiPortId,
        ITciValue sutAddress, ITriSignatureId signatureId,
        ITciValueList parameterList);
    // Ref: TRI-Definition 5.5.4.2
    TriStatus XTriCallBC(ITriComponentId componentId, ITriPortId tsiPortId,
        ITriSignatureId signatureId, ITciValueList parameterList);
    // Ref: TRI-Definition 5.5.4.3
    TriStatus XTriCallMC(ITriComponentId componentId, ITriPortId tsiPortId,
        ITciValueList sutAddresses, ITriSignatureId signatureId,
        ITciValueList parameterList);
    // Ref: TRI-Definition 5.5.4.4
```

```
    TriStatus XTriReply(ITriComponentId componentId, ITriPortId tsiPortId,
        ITciValue sutAddress, ITriSignatureId signatureId,
        ITciValueList parameterList, ITciValue returnValue);
    // Ref: TRI-Definition 5.5.4.5
    TriStatus XTriReplyBC(ITriComponentId componentId, ITriPortId tsiPortId,
        ITriSignatureId signatureId, ITciValueList parameterList,
        ITciValue returnValue);
    // Ref: TRI-Definition 5.5.4.6
    TriStatus XTriReplyMC(ITriComponentId componentId, ITriPortId tsiPortId,
        ITciValueList sutAddresses, ITriSignatureId signatureId,
        ITciValueList parameterList, ITciValue returnValue);
    // Ref: TRI-Definition 5.5.4.7
    TriStatus XTriRaise(ITriComponentId componentId, ITriPortId tsiPortId,
        ITciValue sutAddress, ITriSignatureId signatureId,
        ITciValue exc);
    // Ref: TRI-Definition 5.5.4.8
    TriStatus XTriRaiseBC(ITriComponentId componentId, ITriPortId tsiPortId,
        ITriSignatureId signatureId, ITciValue exc);
    // Ref: TRI-Definition 5.5.4.9
    TriStatus XTriRaiseMC(ITriComponentId componentId, ITriPortId tsiPortId,
        ITciValueList sutAddresses, ITriSignatureId signatureId,
        ITciValue exc);
}
```

9.5.2.2 Changes to ITriCommunicationTE

The extensions to the **ITriCommunicationTE** interface are defined as follows:

```
public interface IXTriCommunicationTE {
    // Message based communication operations
    // Ref: TRI-Definition 5.5.3.4
    void XTriEnqueueMessage(ITriPortId tsiPortId, Object sutAddress,
        ITriComponentId componentId, Object msg);

    // Procedure based communication operations
    // Ref: TRI-Definition 5.5.4.10
    void XTriEnqueueCall(ITriPortId tsiPortId, Object sutAddress,
        ITriComponentId componentId, ITriSignatureId signatureId,
        ITciValueList parameterList);
    // Ref: TRI-Definition 5.5.4.10
    void XTriEnqueueReply(ITriPortId tsiPortId, Object sutAddress,
        ITriComponentId componentId, ITriSignatureId signatureId,
        ITciValueList parameterList, ITciValue returnValue);
    // Ref: TRI-Definition 5.5.4.11
    void XTriEnqueueException(ITriPortId tsiPortId, Object sutAddress,
        ITriComponentId componentId, ITriSignatureId signatureId,
        Object exc);
    // Ref: TRI-Definition 5.5.3.5
    ITciValue XTriConvert(Object value, ITciType typeHypothesis);

}
```

9.5.2.3 Changes to ITriPlatformPA

The extensions to the **ITriPlatformPA** interface are defined as follows:

```
public interface IXTriPlatformPA {
    // Ref: TRI-Definition 5.6.1    // Miscellaneous operations
    // Ref: TRI-Definition 5.6.3.1
    TriStatus XTriExternalFunction(ITriFunctionId functionId,
        ITciValueList parameterList, ITciValue returnValue);
}
```

8 TCI extensions for the package

Not applicable.

SERIES OF ITU-T RECOMMENDATIONS

Series A	Organization of the work of ITU-T
Series D	General tariff principles
Series E	Overall network operation, telephone service, service operation and human factors
Series F	Non-telephone telecommunication services
Series G	Transmission systems and media, digital systems and networks
Series H	Audiovisual and multimedia systems
Series I	Integrated services digital network
Series J	Cable networks and transmission of television, sound programme and other multimedia signals
Series K	Protection against interference
Series L	Construction, installation and protection of cables and other elements of outside plant
Series M	Telecommunication management, including TMN and network maintenance
Series N	Maintenance: international sound programme and television transmission circuits
Series O	Specifications of measuring equipment
Series P	Terminals and subjective and objective assessment methods
Series Q	Switching and signalling
Series R	Telegraph transmission
Series S	Telegraph services terminal equipment
Series T	Terminals for telematic services
Series U	Telegraph switching
Series V	Data communication over the telephone network
Series X	Data networks, open system communications and security
Series Y	Global information infrastructure, Internet protocol aspects and next-generation networks
Series Z	**Languages and general software aspects for telecommunication systems**

Printed in Switzerland
Geneva, 2012