Final draft ETSI ES 202 789 V1.5.1 (2020-02)

Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; TTCN-3 Language Extensions: Extended TRI

Reference
RES/MTS-202789ed151

Keywords
interface, testing, TTCN-3

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

Contents

Intellectual Property Rights

Essential patents

IPRs essential or potentially essential to normative deliverables 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 (https://ipr.etsi.org/).

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.

Trademarks

The present document may include trademarks and/or tradenames which are asserted and/or registered by their owners. ETSI claims no ownership of these except for any which are indicated as being the property of ETSI, and conveys no right to use or reproduce any trademark and/or tradename. Mention of those trademarks in the present document does not constitute an endorsement by ETSI of products, services or organizations associated with those trademarks.

Foreword

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

The use of strike through (deleted text) highlights the differences between base document and extended documents.

The present document relates to the multi-part standard ETSI ES 201 873 covering the Testing and Test Control Notation version 3, as identified in ETSI ES 201 873-1 [1].

Modal verbs terminology

In the present document "**shall**", "**shall not**", "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the ETSI Drafting Rules (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

1 Scope

The present document 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

2.1 Normative 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 referenced document (including any amendments) applies.

Referenced documents which are not found to be publicly available in the expected location might be found at https://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.

The following referenced documents are necessary for the application of the present document.

[1] 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".

[2] 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".

[3] ETSI ES 201 873-5: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 5: TTCN-3 Runtime Interface (TRI)".

[4] ETSI ES 201 873-6: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 6: TTCN-3 Control Interface (TCI)".

[5] Recommendation ITU-T X.290: "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: "Information technology -- Open Systems Interconnection -- Conformance testing methodology and framework -- Part 1: General concepts".

2.2 Informative 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 referenced document (including any amendments) applies.

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

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] Void.

[i.2] Void.

[i.3] ETSI ES 201 873-7: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 7: Using ASN.1 with TTCN-3".

[i.4] ETSI ES 201 873-8: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 8: The IDL to TTCN-3 Mapping".

[i.5] ETSI ES 201 873-9: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 9: Using XML schema with TTCN-3".

[i.6] ETSI ES 201 873-10: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 10: TTCN-3 Documentation Comment Specification".

[i.7] ETSI ES 202 781: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; TTCN-3 Language Extensions: Configuration and Deployment Support".

[i.8] ETSI ES 202 784: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; TTCN-3 Language Extensions: Advanced Parameterization".

[i.9] ETSI ES 202 785: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; TTCN-3 Language Extensions: Behaviour Types".

[i.10] ETSI ES 202 782: "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".

[i.11] ETSI ES 202 786: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; TTCN-3 Language Extensions: Support of interfaces with continuous signals".

3 Definition of terms, symbols and abbreviations

3.1 Terms

For the purposes of the present document, the terms given in ETSI ES 201 873-1 [1], ETSI ES 201 873-4 [2], ETSI ES 201 873-5 [3], ETSI ES 201 873-6 [4] and Recommendation ITU-T X.290 [5] apply.

3.2 Symbols

Void.

3.3 Abbreviations

For the purposes of the present document, the abbreviations given in ETSI ES 201 873-1 [1], ETSI ES 201 873-4 [2], ETSI ES 201 873-5 [3], ETSI ES 201 873-6 [4], Recommendation ITU-T X.290 [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, ETSI ES 201 873-1 [1] and ETSI ES 201 873-4 [2].

The package presented in the present document is compatible to:

ETSI ES 201 873-1 [1] (V4.5.1)

ETSI ES 201 873-4 [2] (V4.4.1)

ETSI ES 201 873-6 [4] (V4.5.1)

ETSI ES 201 873-7 [i.3] (V4.5.1)

ETSI ES 201 873-8 [i.4] (V4.5.1)

ETSI ES 201 873-9 [i.5] (V4.5.1)

ETSI ES 201 873-10 [i.6] (V4.5.1)

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:

ETSI ES 202 784 [i.8] (V1.3.1)

ETSI ES 202 781 [i.7] (V1.2.1)

ETSI ES 202 782 [i.10] (V1.2.1)

ETSI ES 202 785 [i.9] (V1.3.1)

ETSI ES 202 786 [i.11] (V1.2.1)

and can be used together with those packages.

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

7.0 Introduction

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 5.2 of ETSI ES 201 873-5, Error handling

The SA or PA can in addition provide notifications about unrecoverable error situations by use of the operations xtriSAErrorReq and xtriPAErrorReq, respectively.

5.2.1 triSAErrorReq → xtriSAErrorReq

Signature	`void xtriSAErrorReq(in string message, in any cause)`	
In Parameters	`message`	A string value, i.e. the error phrase describing the problem.
	`cause`	(Optional) cause of the problem.
Return Value	`void`	
Constraint	Shall be called whenever an error situation has occurred in the SA with the exception of errors occurring when processing SA calls initiated by the TE. These errors are reported in the operation return. The optional cause parameter can be used to provide information in addition to the error phrase in message.	
Effect	The TE will be notified about an unrecoverable error situation within the SA and may forward the error indication to the test management.	

5.2.2 triPAErrorReq → xtriPAErrorReq

Signature	`void xtriPAErrorReq(in string message, in any cause)`	
In Parameters	`message`	A string value, i.e. the error phrase describing the problem.
	`cause`	(Optional) cause of the problem.
Return Value	`Void`	
Constraint	Shall be called whenever an error situation has occurred in the PA with the exception of errors occurring when processing PA calls initiated by the TE. These errors are reported in the operation return. The optional cause parameter can be used to provide information in addition to the error phrase in message.	
Effect	The TE will be notified about an unrecoverable error situation within the PA and may forward the error indication to the test management.	

7.2 Changes to clause 5.5.2 of ETSI ES 201 873-5, 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.

7.3 Changes to clause 5.5.3 of ETSI ES 201 873-5, 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.
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.
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.
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.
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.

7.4 Addition to clause 5.5.3 of ETSI ES 201 873-5, 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.

7.5 Changes to clause 5.5.4 of ETSI ES 201 873-5, 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 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. The procedure parameters are the parameters specified in the TTCN-3 signature template.	
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 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. The procedure parameters are the parameters specified in the TTCN-3 signature template.	
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 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. The procedure parameters are the parameters specified in the TTCN-3 signature template.
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 parameters which are part of the indicated signature. The parameters in `parameterList` are ordered as they appear in the TTCN-3 signature declaration `returnValue` (optional) return value of the procedure call
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. The `parameterList` contains procedure call parameters. These parameters are the parameters specified in the TTCN-3 signature template. If no return type has been defined for the procedure signature in the TTCN-3 ATS, the distinct value `null` shall be passed for the return value.
Effect	On invocation of this operation the 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 parameters which are part of the indicated signature. The parameters in `parameterList` are ordered as they appear in the TTCN-3 signature declaration
	`returnValue`	(optional) return value of the procedure call
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. The `parameterList` contains procedure call parameters. These parameters are the parameters specified in the TTCN-3 signature template. If no return type has been defined for the procedure signature in the TTCN-3 ATS, the distinct value `null` shall be passed for the return value.	
Effect	On invocation of this operation the 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 parameters which are part of the indicated signature. The parameters in parameterList are ordered as they appear in the TTCN-3 signature declaration
	returnValue	(optional) return value of the procedure call
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. The parameterList contains procedure call parameters. These parameters are the parameters specified in the TTCN-3 signature template. If no return type has been defined for the procedure signature in the TTCN-3 ATS, the distinct value null shall be passed for the return value.	
Effect	On invocation of this operation the 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 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.	
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 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.
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 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.
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 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.	
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 parameters which are part of the indicated signature. The parameters in `parameterList` are ordered as they appear in the TTCN-3 signature declaration
	`returnValue`	(optional) return value of the procedure call
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. 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 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.	
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.	

7.5A Addition to clause 5.5.5 of ETSI ES 201 873-5, Miscellaneous operations

In order to properly log message or procedure communication via TCI TL, an optional xtriDisplay function is used for conversion of not encoded XTRI data structures into a format suitable for TCI TL operations.

5.5.5.2 xtriDisplay

Signature	`TriMessageType xtriDisplay(in TriComponentIdType componentId,`
	`in TriPortIdType tsiPortId,`
	`in Value dataToDisplay)`	

In Parameters	`componentId`	identifier of the sending or receiving test component
	`tsiPortId`	identifier of the test system interface port via which the message was sent to the SUT adaptor or received from it
	`dataToDisplay`	data to be logged
Out Parameters	n.a.	
Return Value	Data in the format suitable for TCI logging or `null` if no logging format is available.	
Constraints	This optional TRI operation is called by the TE when it logs message or procedure communication and a TCI TL operation requires data in its encoded form (TriMessageType for messages; TriParameterListType, TriParameterType and TriExceptionType for procedures; TriAddressType for addresses).	
Effect	The operation converts the supplied value to a TRI data structure and returns it. The converted structure typically contains the data in a form that can be used in the logs for increased readability. The conversion might be context-specific, depending on the provided component and TSI port participating in the logged communication operation. If the conversion is not supported, the distinct `null` value shall be returned.	

7.6 Changes to clause 5.6.3 of ETSI ES 201 873-5, 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). 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 values for all *inout* and *out* function parameters and the 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.

5.6.3.3 triRnd → xtriRnd

Signature	`FloatValue xtriRnd(in TriComponentIdType componentId, in FloatValue seed)`
In Parameters	`componentId` identifier of the component for which to generate the random number
	`seed` the seed to be used for generation of the random number or null
Out Parameters	n.a.
Return Value	The generated float random number.
Constraints	This operation is called by the PA to generate a random number in the context of an external function.
Effect	A random number is generated in the scope of the component identified by the given component Id using the given seed (if any) according to the specification of the predefined rnd function defined in ETSI ES 201 873-1 [1].

7.7 Changes to clause 6 of ETSI ES 201 873-5, Java language mapping

Addition of the following clause in clause 6.3 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 extension 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,
            TciParameterList paramList);
    // Ref: TRI-Definition 5.5.2.3
    public TriStatus xtriUnmapParam(TriPortId compPortId, TriPortId tsiPortId,
            TciParameterList 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 tsiPortId,
            Value sutAddress,
            TriSignatureId signatureId,
            Value exc);
    // Ref: TRI-Definition 5.5.4.8
    public TriStatus xtriRaiseBC(TriComponentId componentId,
            TriPortId tsiPortId,
            TriSignatureId signatureId,
            Value exc);
    // Ref: TRI-Definition 5.5.4.9
    public TriStatus xtriRaiseMC(TriComponentId componentId, TriPortId tsiPortId,
            TciValueList sutAddresses,
            TriSignatureId signatureId,
            Value exc);

    // Miscellaneous operations
    // Ref: TRI-Definition 5.5.3.5
    public Value xtriConvert(Object value, Type typeHypothesis);
    // Ref: TRI-Definition 5.5.5.2
    public TriMessage xtriDisplay(TriComponentId componentId, TriPortId tsiPortId,
            Value dataToDisplay);


}
```

6.5.2.2 Changes to triCommunicationTE

The extension 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,
            Object 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);

    // Error handling
    // Ref: TRI-Definition 5.2.1
    public void xtriSAErrorReq (String message, Object cause);
}
```

6.5.3.1 Changes to TriPlatformPA

The extension 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);
}
```

6.5.3.2 Changes to TriPlatformTE

The extension to the `triPlatformTE` interface is mapped to the following interface:

```
// TriPlatform
// PA -> TE
package org.etsi.ttcn.xtri;
public interface xTriPlatformTE {
    // Error handling
    // Ref: TRI-Definition 5.2.2
    public void xtriPAErrorReq (String message, Object cause);
    // Ref: TRI-Definition 5.6.3.3
    public FloatValue xtriRnd(TriComponentId componentId, FloatValue seed);}
```

7.8 Changes to clause 7 of ETSI ES 201 873-5, C language mapping

7.2.1 Changes to Abstract type mapping

TRI ADT	ANSI C Representation	Notes and comments

any	typedef enum {
	  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 *
	
	  e_char_string = 30, // char *
	  e_wchar_string = 31 // wchar_t *
	} 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 TciParameterListType* parameterList)
TriStatus xtriUnmapParam
(const TriPortId* compPortId,
 const TriPortId* tsiPortId,
 const TciParameterListType* 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 TciParameterListType* parameterList)
TriStatus xtriCallBC
(const TriComponentId* componentId,
 const TriPortId* tsiPortId,
 const TriSignatureId* signatureId,
 const TciParameterListType* parameterList)
TriStatus xtriCallMC
(const TriComponentId* componentId,
 const TriPortId* tsiPortId,
 const TciValueList* sutAddresses,
 const TriSignatureId* signatureId,
 const TciParameterListType* parameterList)
TriStatus xtriReply
 (const TriComponentId* componentId,
 const TriPortId* tsiPortId,
 const Value* sutAddress,
 const TriSignatureId* signatureId,
 const TciParameterListType* parameterList,
 const Value* returnValue)
TriStatus xtriReplyBC
 (const TriComponentId* componentId,
 const TriPortId* tsiPortId,
 const TriSignatureId* signatureId,
 const TciParameterListType* parameterList,
 const Value* returnValue)
TriStatus xtriReplyMC
 (const TriComponentId* componentId,
 const TriPortId* tsiPortId,
 const TciValueList* sutAddresses,
 const TriSignatureId* signatureId,
 const TciParameterListType* 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 TciParameterListType* parameterList)
void xtriEnqueueReply
 (const TriPortId* tsiPortId,
 const Object* sutAddress,
 const TriComponentId* componentId,
 const TriSignatureId* signatureId,
 const TciParameterListType* 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,
 TciParameterListType* parameterList,
 Value* returnValue)
Value xtriConvert
 (const Object* value,
 const Type* typeHypothesis)
TFloat xtriRnd(TriComponentId *componentId, TFloat* seed)
void xtriPAErrorReq
 (const char* message,
  const Object* cause)
void xtriSAErrorReq
 (const char* message,
  const Object* cause)
TriMessage * xtriDisplay(TriComponentId * componentId, TriPortId * tsiPortId,
    Value dataToDisplay)
```

7.9 Changes to clause 8 of ETSI ES 201 873-5, C++ language mapping

Addition of the following clause in clause 8.5 Type mapping.

8.5.3 Any type mapping

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

```
   typedef enum {
     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 *

     e_char_string = 30,            // char *
     e_wchar_string = 31            // wchar_t *
   } type_kind;
```

```
typedef void *value;

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

8.6.1 Changes to TriCommunicationSA

The extension to the `TriCommunicationSA` class is mapped to the following class:

```
class xTriCommunicationSA {
public:

    //Destructor.
    virtual ~xTriCommunicationSA ();

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

```
    Conversion of received data
    virtual TciValue *xtriConvert(const Object *value, const TciType *typeHypothesis)=0;

    // Conversion to the logging format
    virtual TriMessage * xtriDisplay(const TriComponentId * componentId, const TriPortId *
    tsiPortId, const TciValue dataToDisplay)=0;


}
```

8.6.2 Changes to TriCommunicationTE

The extension to the `TriCommunicationTE` class is mapped to the following class:

```
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;

    // Error handling
    virtual void xtriSAErrorReq (const String message, const Object *cause)=0;
}
```

8.6.3 Changes to TriPlatformPA

The extension to the `TriPlatformPA` class is mapped to the following class:

```
class xTriPlatformPA {
public:

    //Destructor.
    virtual ~xTriPlatformPA ();

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

8.6.4 Changes to TriPlatformTE

The extension to the `TriPlatformTE` class is mapped to the following interface:

```
class TriPlatformTE {
public:

    //Destructor.
    virtual ~xTriPlatformTE ();

    //Called by PA in unrecoverable error situations.
    virtual void xtriPAError (const Tstring &message, const Object *cause)=0;

    //Generate random number.
    virtual FloatValue* xtriRnd (const TriComponentId *componentId, const FloatValue *seed)=0;
}
```

7.10 Changes to clause 9 of ETSI ES 201 873-5, C# language mapping

Addition of the following clause in clause 9.4 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 extension to the **`ITriCommunicationSA`** interface is defined as follows:

```
public interface IXTriCommunicationSA {
    // Reset operation
    // Ref: TRI-Definition 5.5.1
    TriStatus XTriMapParam(ITriPortId compPortId, ITriPortId tsiPortId,
        ITciParameterList parameterList);
    // Ref: TRI-Definition 5.5.2.3
    TriStatus XTriUnmapParam(ITriPortId compPortId, ITriPortId tsiPortId,
        ITciParameterList 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,
        ITciParameterList parameterList);
    // Ref: TRI-Definition 5.5.4.2
    TriStatus XTriCallBC(ITriComponentId componentId, ITriPortId tsiPortId,
        ITriSignatureId signatureId, ITciParameterList parameterList);
    // Ref: TRI-Definition 5.5.4.3
    TriStatus XTriCallMC(ITriComponentId componentId, ITriPortId tsiPortId,
        ITciValueList sutAddresses, ITriSignatureId signatureId,
        ITciParameterList parameterList);
    // Ref: TRI-Definition 5.5.4.4
    TriStatus XTriReply(ITriComponentId componentId, ITriPortId tsiPortId,
        ITciValue sutAddress, ITriSignatureId signatureId,
        ITciParameterList parameterList, ITciValue returnValue);
    // Ref: TRI-Definition 5.5.4.5
    TriStatus XTriReplyBC(ITriComponentId componentId, ITriPortId tsiPortId,
        ITriSignatureId signatureId, ITciParameterList parameterList,
        ITciValue returnValue);
    // Ref: TRI-Definition 5.5.4.6
    TriStatus XTriReplyMC(ITriComponentId componentId, ITriPortId tsiPortId,
        ITciValueList sutAddresses, ITriSignatureId signatureId,
        ITciParameterList 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);
    // Ref: TRI-Definition 5.5.3.5
    ITciValue XTriConvert(object value, ITciType typeHypothesis);
    // Ref: TRI-Definition 5.5.5.2
    ITriMessage XTriDisplay(ITriComponentId componentId, ITriPortId tsiPortId,
        ITciValue dataToDisplay);
```

```
}
```

9.5.2.2 Changes to ITriCommunicationTE

The extension to the **ITriCommunicationTE** interface is 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,
        ITciParameterList parameterList);
    // Ref: TRI-Definition 5.5.4.10
    void XTriEnqueueReply(ITriPortId tsiPortId, object sutAddress,
        ITriComponentId componentId, ITriSignatureId signatureId,
        ITciParameterList 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 5.2.1
    void XTriSAErrorReq (string message, object cause);
}
```

9.5.2.3 Changes to ITriPlatformPA

The extension to the **ITriPlatformPA** interface is 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,
        ITciParameterList parameterList, ITciValue returnValue);
}
```

9.5.2.4 Changes to ITriPlatformTE

The extension to the **ITriPlatformTE** interface is defined as follows:

```
public interface ITriPlatformTE {
    // Ref: TRI Definition 5.2.2
    void XTriPAErrorReq (string message, object cause);
    // Ref: TRI Definition clause 5.6.3.3
    FloatValue XTriRnd(ITriComponentId componentId, FloatValue seed);
}
```

8 TCI extensions for the package

Not applicable.

Annex A (informative): Bibliography

- ETSI ES 201 873-3: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 3: TTCN-3 Graphical presentation Format (GFT)".

History

Document history		
V1.1.1	April 2012	Publication
V1.2.1	April 2013	Publication
V1.3.1	June 2014	Publication
V1.4.1	June 2015	Publication
V1.5.1	February 2020	Membership Approval Procedure MV 20200428: 2020-02-28 to 2020-04-28