ETSI ES 201 873-6 V4.12.2 (2021-12)

Methods for Testing and Specification (MTS);

The Testing and Test Control Notation version 3;

Part 6: TTCN‑3 Control Interface (TCI)

**ETSI Standard**

Reference

RES/MTS-201873-6v4.12.1

Keywords

control, interface, methodology, TCI, 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

***Important notice***

The present document can be downloaded from:  
<http://www.etsi.org/standards-search>

The present document may be made available in electronic versions and/or in print. The content of any electronic and/or print versions of the present document shall not be modified without the prior written authorization of ETSI. In case of any existing or perceived difference in contents between such versions and/or in print, the prevailing version of an ETSI deliverable is the one made publicly available in PDF format at [www.etsi.org/deliver](http://www.etsi.org/deliver).

Users of the present document should be aware that the document may be subject to revision or change of status. Information on the current status of this and other ETSI documents is available at <https://portal.etsi.org/TB/ETSIDeliverableStatus.aspx>

If you find errors in the present document, please send your comment to one of the following services:  
<https://portal.etsi.org/People/CommiteeSupportStaff.aspx>

***Copyright Notification***

No part may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm except as authorized by written permission of ETSI.  
The content of the PDF version shall not be modified without the written authorization of ETSI.  
The copyright and the foregoing restriction extend to reproduction in all media.

© ETSI 2020.

All rights reserved.

**DECT™**, **PLUGTESTS™**, **UMTS™** and the ETSI logo are trademarks of ETSI registered for the benefit of its Members.  
**3GPP™**and **LTE™** are trademarks of ETSI registered for the benefit of its Members and  
of the 3GPP Organizational Partners.  
**oneM2M™** logo is a trademark of ETSI registered for the benefit of its Members and  
of the oneM2M Partners.  
**GSM®** and the GSM logo are trademarks registered and owned by the GSM Association.

Contents

Intellectual Property Rights 16

Foreword 16

Modal verbs terminology 16

1 Scope 17

2 References 17

2.1 Normative references 17

2.2 Informative references 18

3 Definition of terms, symbols and abbreviations 18

3.1 Terms 18

3.2 Symbols 19

3.3 Abbreviations 19

4 Introduction 20

5 Compliance 20

6 General structure of a TTCN‑3 test system 21

6.1 Entities in a TTCN‑3 test system 21

6.1.0 Types of entities 21

6.1.1 Test Management and Control (TMC) 22

6.1.1.0 Test Management and Control Entities 22

6.1.1.1 Test Management (TM) 22

6.1.1.2 Coding and Decoding (CD) 23

6.1.1.3 Component Handling (CH) 23

6.1.1.4 Test Logging (TL) 24

6.1.2 TTCN‑3 Executable (TE) 24

6.1.3 SUT Adaptor (SA) 24

6.1.4 Platform Adaptor (PA) 24

6.2 Execution requirements for a TTCN‑3 test system 24

7 TTCN‑3 control interface and operations 25

7.1 Overview of the TCI 25

7.1.0 TCI role in a TTCN-3 test system 25

7.1.1 Correlation between TTCN‑3 and TCI operation invocations 25

7.1.1.0 Mapping of TTCN-3 operations to TCI operations 25

7.1.1.1 TTCN-3 operations with TCI operation equivalent 26

7.1.1.2 TTCN-3 operations with TCI operation pair equivalent 26

7.1.1.3 TTCN-3 operations without direct TCI operation equivalent 27

7.1.1.3.0 Mapping of TTCN-3 operations to series of TCI operations 27

7.1.1.3.1 Test case stop operation 27

7.2 TCI data 27

7.2.0 Abstract data types 27

7.2.1 General abstract data types 28

7.2.1.0 Use of general abstract data types 28

7.2.1.1 Management 28

7.2.1.2 Communication 29

7.2.2 Abstract TTCN‑3 data types and values 29

7.2.2.0 Definition and scope of use 29

7.2.2.1 Abstract TTCN‑3 data types 29

7.2.2.2 Abstract TTCN‑3 values 31

7.2.2.2.0 Basic rules 31

7.2.2.2.1 The abstract data type Value 32

7.2.2.2.2 The abstract data type IntegerValue 35

7.2.2.2.3 The abstract data type FloatValue 35

7.2.2.2.4 The abstract data type BooleanValue 35

7.2.2.2.5 The abstract data type CharstringValue 35

7.2.2.2.6 The abstract data type UniversalCharstringValue 36

7.2.2.2.7 The abstract data type BitstringValue 36

7.2.2.2.8 The abstract data type OctetstringValue 37

7.2.2.2.9 The abstract data type HexstringValue 38

7.2.2.2.10 The abstract data type RecordValue 39

7.2.2.2.11 The abstract data type RecordOfValue 39

7.2.2.2.12 The abstract data type UnionValue 41

7.2.2.2.13 The abstract data type EnumeratedValue 41

7.2.2.2.14 The abstract data type VerdictValue 41

7.2.2.2.15 The abstract data type AddressValue 42

7.2.2.3 Abstract TTCN‑3 matching mechanisms 42

7.2.2.3.1 The abstract data type MatchingMechanism 42

7.2.2.3.2 The abstract data type MatchingList 42

7.2.2.3.3 The abstract data type ValueRange 43

7.2.2.3.4 The abstract data type CharacterPattern 43

7.2.2.3.5 The abstract data type MatchDecodedContent 44

7.2.2.4 Data types for complex TTCN‑3 properties 44

7.2.2.4.0 Scope of use of TTCN-3 properties 44

7.2.2.4.1 The abstract data type LengthRestriction 44

7.2.2.4.2 The abstract data type Permutation 44

7.2.2.4.3 The abstract data type RangeBoundary 45

7.2.3 Abstract logging types 45

7.2.3.1 The abstract data type TciValueTemplate 45

7.2.3.2 The abstract data type TciNonValueTemplate 45

7.2.3.3 The Value List and Mismatch Types 46

7.2.3.4 The Status Types 46

7.3 TCI operations 47

7.3.0 The TCI interfaces 47

7.3.1 The TCI‑TM interface 48

7.3.1.0 Scope of use 48

7.3.1.1 TCI‑TM required 48

7.3.1.1.0 Scope of use 48

7.3.1.1.1 tciRootModule 48

7.3.1.1.2 tciGetImportedModules 49

7.3.1.1.3 tciGetModuleParameters 49

7.3.1.1.4 tciGetTestCases 49

7.3.1.1.5 tciGetTestCaseParameters 49

7.3.1.1.6 tciGetTestCaseTSI 49

7.3.1.1.7 tciStartTestCase 50

7.3.1.1.8 tciStopTestCase 50

7.3.1.1.9 tciStartControl 50

7.3.1.1.10 tciStopControl 50

7.3.1.1.11 tciGetControlParameters 51

7.3.1.1.12 tciStartControlWithParameters 51

7.3.1.2 TCI‑TM provided 51

7.3.1.2.0 Scope of use 51

7.3.1.2.1 tciTestCaseStarted 51

7.3.1.2.2 tciTestCaseTerminated 51

7.3.1.2.3 tciControlTerminated 52

7.3.1.2.4 tciGetModulePar 52

7.3.1.2.5 tciLog 52

7.3.1.2.6 tciError 52

7.3.1.2.7 tciControlTerminatedWithResult 53

7.3.2 The TCI‑CD interface 53

7.3.2.0 Scope of use 53

7.3.2.1 TCI‑CD required 54

7.3.2.1.0 Scope of use 54

7.3.2.1.1 getTypeForName 54

7.3.2.1.2 getInteger 54

7.3.2.1.3 getFloat 54

7.3.2.1.4 getBoolean 54

7.3.2.1.5 Void 54

7.3.2.1.6 getCharstring 55

7.3.2.1.7 getUniversalCharstring 55

7.3.2.1.8 getHexstring 55

7.3.2.1.9 getBitstring 55

7.3.2.1.10 getOctetstring 55

7.3.2.1.11 getVerdict 55

7.3.2.1.12 tciErrorReq 55

7.3.2.2 TCI‑CD provided 55

7.3.2.2.0 Scope of use 55

7.3.2.2.1 decode 56

7.3.2.2.2 encode 56

7.3.2.2.3 decodeValue 56

7.3.2.2.4 encodeValue 57

7.3.3 The TCI‑CH interface 57

7.3.3.0 Scope of use 57

7.3.3.1 TCI‑CH required 58

7.3.3.1.0 Scope of use 58

7.3.3.1.1 tciEnqueueMsgConnected 58

7.3.3.1.2 tciEnqueueCallConnected 58

7.3.3.1.3 tciEnqueueReplyConnected 59

7.3.3.1.4 tciEnqueueRaiseConnected 59

7.3.3.1.5 tciCreateTestComponent 59

7.3.3.1.6 tciStartTestComponent 60

7.3.3.1.7 tciStopTestComponent 60

7.3.3.1.8 tciConnect 60

7.3.3.1.9 tciDisconnect 60

7.3.3.1.10 tciMap 61

7.3.3.1.11 tciMapParam 61

7.3.3.1.12 tciUnmap 61

7.3.3.1.13 tciUnmapParam 61

7.3.3.1.14 tciTestComponentTerminated 62

7.3.3.1.15 tciTestComponentRunning 62

7.3.3.1.16 tciTestComponentDone 62

7.3.3.1.17 tciGetMTC 62

7.3.3.1.18 tciExecuteTestCase 63

7.3.3.1.19 tciReset 63

7.3.3.1.20 tciKillTestComponent 63

7.3.3.1.21 tciTestComponentAlive 63

7.3.3.1.22 tciTestComponentKilled 64

7.3.3.1.23 tciCallTestComponent 64

7.3.3.1.24 tciTestComponentCallTerminated 64

7.3.3.2 TCI‑CH provided 65

7.3.3.2.0 Scope of use 65

7.3.3.2.1 tciSendConnected 65

7.3.3.2.2 tciSendConnectedBC 65

7.3.3.2.3 tciSendConnectedMC 65

7.3.3.2.4 tciCallConnected 66

7.3.3.2.5 tciCallConnectedBC 66

7.3.3.2.6 tciCallConnectedMC 67

7.3.3.2.7 tciReplyConnected 67

7.3.3.2.8 tciReplyConnectedBC 68

7.3.3.2.9 tciReplyConnectedMC 68

7.3.3.2.10 tciRaiseConnected 69

7.3.3.2.11 tciRaiseConnectedBC 69

7.3.3.2.12 tciRaiseConnectedMC 69

7.3.3.2.13 tciCreateTestComponentReq 70

7.3.3.2.14 tciStartTestComponentReq 70

7.3.3.2.15 tciStopTestComponentReq 70

7.3.3.2.16 tciConnectReq 70

7.3.3.2.17 tciDisconnectReq 71

7.3.3.2.18 tciMapReq 71

7.3.3.2.19 tciMapParamReq 71

7.3.3.2.20 tciUnmapReq 71

7.3.3.2.21 tciUnmapParamReq 72

7.3.3.2.22 tciTestComponentTerminatedReq 72

7.3.3.2.23 tciTestComponentRunningReq 72

7.3.3.2.24 tciTestComponentDoneReq 72

7.3.3.2.25 tciGetMTCReq 72

7.3.3.2.26 tciExecuteTestCaseReq 73

7.3.3.2.27 tciResetReq 73

7.3.3.2.28 tciKillTestComponentReq 73

7.3.3.2.29 tciTestComponentAliveReq 73

7.3.3.2.30 tciTestComponentKilledReq 73

7.3.3.2.31 tciCallTestComponentReq 74

7.3.3.2.32 tciTestComponentCallTerminatedReq 74

7.3.4 The TCI‑TL interface 75

7.3.4.0 Scope of use 75

7.3.4.1 TCI‑TL provided 75

7.3.4.1.0 Scope of use 75

7.3.4.1.1 tliTcExecute 75

7.3.4.1.2 tliTcStart 76

7.3.4.1.3 tliTcStop 76

7.3.4.1.4 tliTcStarted 76

7.3.4.1.5 tliTcTerminated 77

7.3.4.1.6 tliCtrlStart 77

7.3.4.1.7 tliCtrlStop 77

7.3.4.1.8 tliCtrlTerminated 78

7.3.4.1.9 tliMSend\_m 78

7.3.4.1.10 tliMSend\_m\_BC 79

7.3.4.1.11 tliMSend\_m\_MC 79

7.3.4.1.12 tliMSend\_c 80

7.3.4.1.13 tliMSend\_c\_BC 80

7.3.4.1.14 tliMSend\_c\_MC 80

7.3.4.1.15 tliMDetected\_m 81

7.3.4.1.16 tliMDetected\_c 81

7.3.4.1.17 tliMMismatch\_m 81

7.3.4.1.18 tliMMismatch\_c 82

7.3.4.1.19 tliMReceive\_m 82

7.3.4.1.20 tliMReceive\_c 83

7.3.4.1.21 tliPrCall\_m 83

7.3.4.1.22 tliPrCall\_m\_BC 84

7.3.4.1.23 tliPrCall\_m\_MC 84

7.3.4.1.24 tliPrCall\_c 85

7.3.4.1.25 tliPrCall\_c\_BC 85

7.3.4.1.26 tliPrCall\_c\_MC 86

7.3.4.1.27 tliPrGetCallDetected\_m 86

7.3.4.1.28 tliPrGetCallDetected\_c 87

7.3.4.1.29 tliPrGetCallMismatch\_m 87

7.3.4.1.30 tliPrGetCallMismatch\_c 87

7.3.4.1.31 tliPrGetCall\_m 88

7.3.4.1.32 tliPrGetCall\_c 88

7.3.4.1.33 tliPrReply\_m 89

7.3.4.1.34 tliPrReply\_m\_BC 89

7.3.4.1.35 tliPrReply\_m\_MC 90

7.3.4.1.36 tliPrReply\_c 90

7.3.4.1.37 tliPrReply\_c\_BC 91

7.3.4.1.38 tliPrReply\_c\_MC 91

7.3.4.1.39 tliPrGetReplyDetected\_m 92

7.3.4.1.40 tliPrGetReplyDetected\_c 92

7.3.4.1.41 tliPrGetReplyMismatch\_m 93

7.3.4.1.42 tliPrGetReplyMismatch\_c 93

7.3.4.1.43 tliPrGetReply\_m 94

7.3.4.1.44 tliPrGetReply\_c 94

7.3.4.1.45 tliPrRaise\_m 95

7.3.4.1.46 tliPrRaise\_m\_BC 95

7.3.4.1.47 tliPrRaise\_m\_MC 96

7.3.4.1.48 tliPrRaise\_c 96

7.3.4.1.49 tliPrRaise\_c\_BC 97

7.3.4.1.50 tliPrRaise\_c\_MC 97

7.3.4.1.51 tliPrCatchDetected\_m 98

7.3.4.1.52 tliPrCatchDetected\_c 98

7.3.4.1.53 tliPrCatchMismatch\_m 99

7.3.4.1.54 tliPrCatchMismatch\_c 99

7.3.4.1.55 tliPrCatch\_m 100

7.3.4.1.56 tliPrCatch\_c 100

7.3.4.1.57 tliPrCatchTimeoutDetected 101

7.3.4.1.58 tliPrCatchTimeout 101

7.3.4.1.59 tliCCreate 101

7.3.4.1.60 tliCStart 102

7.3.4.1.61 tliCRunning 102

7.3.4.1.62 tliCAlive 102

7.3.4.1.63 tliCStop 103

7.3.4.1.64 tliCKill 103

7.3.4.1.65 tliCDoneMismatch 103

7.3.4.1.66 tliCDone 104

7.3.4.1.67 tliCKilledMismatch 104

7.3.4.1.68 tliCKilled 104

7.3.4.1.69 tliCTerminated 105

7.3.4.1.70 tliPConnect 105

7.3.4.1.71 tliPDisconnect 105

7.3.4.1.72 tliPMap 106

7.3.4.1.73 tliPMapParam 106

7.3.4.1.74 tliPUnmap 106

7.3.4.1.75 tliPUnmapParam 107

7.3.4.1.76 tliPClear 107

7.3.4.1.77 tliPStart 107

7.3.4.1.78 tliPStop 108

7.3.4.1.79 tliPHalt 108

7.3.4.1.80 tliEncode 108

7.3.4.1.81 tliDecode 109

7.3.4.1.82 tliTTimeoutDetected 109

7.3.4.1.83 tliTTimeoutMismatch 109

7.3.4.1.84 tliTTimeout 110

7.3.4.1.85 tliTStart 110

7.3.4.1.86 tliTStop 110

7.3.4.1.87 tliTRead 111

7.3.4.1.88 tliTRunning 111

7.3.4.1.89 tliSEnter 111

7.3.4.1.90 tliSLeave 112

7.3.4.1.91 tliVar 112

7.3.4.1.92 tliModulePar 112

7.3.4.1.93 tliGetVerdict 113

7.3.4.1.94 tliSetVerdict 113

7.3.4.1.95 tliLog 113

7.3.4.1.96 tliAEnter 114

7.3.4.1.97 tliALeave 114

7.3.4.1.98 tliANomatch 114

7.3.4.1.99 tliARepeat 114

7.3.4.1.100 tliADefaults 115

7.3.4.1.101 tliAActivate 115

7.3.4.1.102 tliADeactivate 115

7.3.4.1.103 tliAWait 116

7.3.4.1.104 tliAction 116

7.3.4.1.105 tliMatch 116

7.3.4.1.106 tliMatchMismatch 116

7.3.4.1.107 tliInfo 117

7.3.4.1.108 tliMChecked\_m 117

7.3.4.1.109 tliMChecked\_c 117

7.3.4.1.110 tliPrGetCallChecked\_m 118

7.3.4.1.111 tliPrGetCallChecked\_c 118

7.3.4.1.112 tliPrGetReplyChecked\_m 119

7.3.4.1.113 tliPrGetReplyChecked\_c 120

7.3.4.1.114 tliPrCatchChecked\_m 120

7.3.4.1.115 tliPrCatchChecked\_c 121

7.3.4.1.116 tliCheckedAny\_m 121

7.3.4.1.117 tliCheckedAny\_c 121

7.3.4.1.118 tliCheckAnyMismatch\_m 122

7.3.4.1.119 tliCheckAnyMismatch\_c 122

7.3.4.1.120 tliRnd 122

7.3.4.1.121 tliEvaluate 123

7.3.4.1.122 tliCCall 123

7.3.4.1.123 tliCCallTerminated 123

7.3.4.1.124 tliCtrlStartWithParameters 124

7.3.4.1.125 tliCtrlTerminatedWithResult 124

8 JavaTM language mapping 124

8.1 Introduction 124

8.2 Names and scopes 124

8.2.1 Names 124

8.2.2 Scopes 125

8.3 Type mapping 125

8.3.1 Basic type mapping 125

8.3.2 Structured type mapping 126

8.3.2.0 General principles 126

8.3.2.1 TciParameterType 126

8.3.2.2 TciParameterPassingModeType 126

8.3.2.3 TciParameterListType 127

8.3.2.4 TciTypeClassType 127

8.3.2.5 TciTestComponentKindType 128

8.3.2.6 TciBehaviourIdType 128

8.3.2.7 TciTestCaseIdType 128

8.3.2.8 TciModuleIdType 128

8.3.2.9 TciModuleParameterIdType 128

8.3.2.10 TciModuleParameterListType 128

8.3.2.11 TciModuleParameterType 129

8.3.2.12 TciParameterTypeListType 129

8.3.2.13 TciParameterTypeType 129

8.3.2.14 TciModuleIdListType 130

8.3.2.15 TciTestCaseIdListType 130

8.3.2.16 TciDecodingResult 130

8.3.2.17 TciMatchingTypeType 131

8.3.2.18 LengthRestriction 131

8.3.2.19 Permutation 131

8.3.2.20 RangeBoundary 132

8.3.3 Abstract type mapping 132

8.3.3.0 General principles 132

8.3.3.1 Type 132

8.3.4 Abstract value mapping 134

8.3.4.0 General principles 134

8.3.4.1 Value 134

8.3.4.2 IntegerValue 135

8.3.4.3 FloatValue 135

8.3.4.4 BooleanValue 136

8.3.4.5 CharstringValue 136

8.3.4.6 BitstringValue 137

8.3.4.7 OctetstringValue 138

8.3.4.8 UniversalCharstringValue 139

8.3.4.9 HexstringValue 139

8.3.4.10 RecordValue 140

8.3.4.11 RecordOfValue 141

8.3.4.12 UnionValue 142

8.3.4.13 EnumeratedValue 142

8.3.4.14 VerdictValue 143

8.3.4.15 AddressValue 143

8.3.5 Abstract template mapping 143

8.3.5.0 General principles 143

8.3.5.1 MatchingMechanism 144

8.3.5.2 MatchingList 144

8.3.5.3 ValueRange 144

8.3.5.4 CharacterPattern 145

8.3.5.5 MatchDecodedContent 145

8.3.6 Abstract logging types mapping 145

8.3.6.0 General principles 145

8.3.6.1 TciValueTemplate 145

8.3.6.2 TciNonValueTemplate 146

8.3.6.3 TciValueList 146

8.3.6.4 TciValueDifference 146

8.3.6.5 TciValueDifferenceList 147

8.3.6.6 ComponentStatus 147

8.3.6.7 TimerStatus 147

8.3.6.8 TciStatus 147

8.4 Constants 148

8.5 Mapping of interfaces 149

8.5.0 Calling rules 149

8.5.1 The TCI‑TM interface 150

8.5.1.1 TCI‑TM provided 150

8.5.1.2 TCI‑TM required 150

8.5.2 The TCI‑CD interface 150

8.5.2.1 TCI‑CD provided 150

8.5.2.2 TCI‑CD required 150

8.5.3 The TCI‑CH interface 151

8.5.3.1 TCI‑CH provided 151

8.5.3.2 TCI‑CH required 152

8.5.4 The TCI‑TL interface 153

8.5.4.1 TCI‑TL provided 153

8.6 Optional parameters 158

8.7 TCI initialization 158

8.8 Error handling 158

9 ANSI C language mapping 158

9.1 Introduction 158

9.2 Value interfaces 158

9.3 Logging interface 164

9.4 Operation interfaces 164

9.4.1 The TCI‑TM interface 164

9.4.1.1 TCI‑TM provided 164

9.4.1.2 TCI‑TM required 165

9.4.2 The TCI‑CD interface 165

9.4.2.1 TCI‑CD provided 165

9.4.2.2 TCI‑CD required 165

9.4.3 The TCI‑CH interface 165

9.4.3.1 TCI‑CH provided 165

9.4.3.2 TCI‑CH required 166

9.4.4 The TCI‑TL interface 167

9.4.4.1 TCI‑TL provided 167

9.5 Data 172

9.6 Miscellaneous 174

9.7 Optional parameters 174

10 C++ language mapping 175

10.1 Introduction 175

10.2 Names and scopes 175

10.3 Memory management 175

10.4 Error handling 175

10.5 Type mapping 175

10.5.0 Basic concepts 175

10.5.1 Encapsulated C++ types 175

10.5.2 General abstract data types 176

10.5.2.1 TciBehaviourId 176

10.5.2.2 TciModuleId 176

10.5.2.3 TciModuleParameterId 176

10.5.2.4 TciTestCaseId 177

10.5.2.5 TciModuleIdList 177

10.5.2.6 TciModuleParameter 178

10.5.2.7 TciModuleParameterList 178

10.5.2.8 TciParameterPassingMode 179

10.5.2.9 TciParameter 179

10.5.2.10 TciParameterList 179

10.5.2.11 TciParameterType 180

10.5.2.12 TciParameterTypeList 180

10.5.2.13 TciTestComponentKind 181

10.5.2.14 TciTypeClass 181

10.5.2.15 TciTestCaseIdList 182

10.5.2.16 TciMatchingTypeType 182

10.5.2.17 LengthRestriction 182

10.5.2.18 Permutation 183

10.5.2.19 RangeBoundary 184

10.5.3 Abstract TTCN-3 data types and values 184

10.5.3.1 TciType 184

10.5.3.2 TciValue 185

10.5.3.3 IntegerValue 187

10.5.3.4 FloatValue 187

10.5.3.5 BooleanValue 187

10.5.3.6 CharstringValue 188

10.5.3.7 UniversalCharstringValue 188

10.5.3.8 BitstringValue 189

10.5.3.9 OctetstringValue 190

10.5.3.10 HexstringValue 191

10.5.3.11 RecordValue 192

10.5.3.12 RecordOfValue 192

10.5.3.13 UnionValue 193

10.5.3.14 EnumeratedValue 193

10.5.3.15 VerdictValue 194

10.5.3.16 VerdictValueEnum 194

10.5.3.17 AddressValue 195

10.5.3.18 MatchingMechanism 195

10.5.3.19 MatchingList 195

10.5.3.20 ValueRange 196

10.5.3.21 CharacterPattern 196

10.5.3.22 MatchDecodedContent 197

10.5.4 Abstract logging types 197

10.5.4.1 TciValueTemplate 197

10.5.4.2 TciNonValueTemplate 198

10.5.4.3 TciValueList 198

10.5.4.4 TciValueDifference 199

10.5.4.5 TciValueDifferenceList 199

10.5.4.6 ComponentStatus 200

10.5.4.7 TimerStatus 200

10.5.4.8 TciStatus 200

10.6 Operations mapping 201

10.6.1 TCI-TM 201

10.6.1.1 TciTmRequired 201

10.6.1.2 TciTmProvided 201

10.6.2 TCI-CD 202

10.6.2.1 TciCdRequired 202

10.6.2.2 TciCdProvided 202

10.6.3 TCI-CH 203

10.6.3.1 TciChRequired 203

10.6.3.2 TciChProvided 204

10.6.4 TCI-TL 206

10.6.4.1 TciTlProvided 206

11 W3C XML mapping 214

11.1 Introduction 214

11.2 Scopes 214

11.3 Type mapping 215

11.3.1 Mapping of simple types 215

11.3.1.1 TBoolean 215

11.3.1.2 TString 215

11.3.1.3 TInteger 215

11.3.1.4 TriTimerDurationType 215

11.3.1.5 TciParameterPassingModeType 215

11.3.1.6 TriStatusType 215

11.3.1.7 TciStatusType 215

11.3.1.8 ComponentStatusType 215

11.3.1.9 TimerStatusType 215

11.3.1.10 PortStatusType 216

11.3.2 Complex type mapping 216

11.3.2.1 TriPortIdType 216

11.3.2.2 TriComponentIdType 216

11.3.2.3 TriComponentIdListType 216

11.3.2.4 Port 217

11.3.2.5 Id 217

11.3.2.6 TriMessageType 217

11.3.2.7 TriParameterType 218

11.3.2.8 TriParameterListType 218

11.3.2.9 TriAddressType 218

11.3.2.10 TriAddressListType 219

11.3.2.11 TriExceptionType 219

11.3.2.12 TriSignatureIdType 219

11.3.2.13 TriTimerIdType 220

11.3.2.14 TriTimerDurationType 220

11.3.2.15 QualifiedName 220

11.3.2.16 TciBehaviourIdType 220

11.3.2.17 TciTestCaseIdType 221

11.3.2.18 TciParameterType 221

11.3.2.19 TciParameterListType 221

11.3.2.20 TriPortIdListType 221

11.3.3 Abstract value mapping 222

11.3.3.1 Value 222

11.3.3.2 IntegerValue 224

11.3.3.3 FloatValue 224

11.3.3.4 BooleanValue 224

11.3.3.5 Void 225

11.3.3.6 VerdictValue 225

11.3.3.7 BitstringValue 225

11.3.3.8 HexstringValue 225

11.3.3.9 OctetstringValue 225

11.3.3.10 CharstringValue 226

11.3.3.11 UniversalCharstringValue 226

11.3.3.12 RecordValue 226

11.3.3.13 RecordOfValue 227

11.3.3.14 ArrayValue 228

11.3.3.15 SetValue 229

11.3.3.16 SetOfValue 229

11.3.3.17 EnumeratedValue 229

11.3.3.18 UnionValue 230

11.3.3.19 AnytypeValue 230

11.3.3.20 AddressValue 231

11.3.3.21 ComponentValue 231

11.3.3.22 PortValue 231

11.3.3.23 DefaultValue 232

11.3.3.24 TimerValue 232

11.3.3.25 MatchingMechanism 232

11.3.3.26 MatchingList 233

11.3.3.27 ValueRange 233

11.3.3.28 CharacterPattern 234

11.3.3.29 MatchDecodedContent 234

11.3.4 Abstract logging types mapping 234

11.3.4.1 TciValueTemplate 234

11.3.4.2 TciNonValueTemplate 236

11.3.4.3 TciValueList 236

11.3.4.4 TciValueDifference 237

11.3.4.5 TciValueDifferenceList 237

11.4 Mapping of the operations on the logging interface 237

11.4.0 Mapping rules 237

11.4.1 Event 237

11.4.2 The TCI‑TL interface 238

11.4.2.1 TCI‑TL provided 238

12 C# mapping 260

12.1 Introduction 260

12.2 Names and scopes 260

12.2.1 Names 260

12.2.2 Scopes 261

12.3 Null value mapping 261

12.4 Type mapping 261

12.4.1 Basic type mapping 261

12.4.1.0 Mapped types 261

12.4.1.1 TciVerdict 262

12.4.2 Structured type mapping 262

12.4.2.0 Mapping rules 262

12.4.2.1 TciParameterPassingModeType 262

12.4.2.2 TciParameterType 262

12.4.2.3 TciParameterListType 262

12.4.2.4 TciTypeClassType 263

12.4.2.5 TciTestComponentKindType 263

12.4.2.6 TciBehaviourIdType 263

12.4.2.7 TciTestCaseIdType 264

12.4.2.8 TciTestCaseIdListType 264

12.4.2.9 TciModuleIdType 264

12.4.2.10 TciModuleIdListType 264

12.4.2.11 TciModuleParameterIdType 265

12.4.2.12 TciModuleParameterType 265

12.4.2.13 TciModuleParameterListType 265

12.4.2.14 TciParameterTypeType 265

12.4.2.15 TciParameterTypeListType 266

12.4.2.16 TciMatchingTypeType 266

12.4.2.17 LengthRestriction 266

12.4.2.18 Permutation 267

12.4.2.19 RangeBoundary 267

12.4.3 Abstract type mapping 267

12.4.3.0 Mapping rules 267

12.4.3.1 Type 267

12.4.4 Abstract value mapping 269

12.4.4.0 Mapping rules 269

12.4.4.1 Value 269

12.4.4.2 IntegerValue 270

12.4.4.3 FloatValue 271

12.4.4.4 BooleanValue 271

12.4.4.5 CharstringValue 271

12.4.4.6 BitstringValue 271

12.4.4.7 OctetstringValue 272

12.4.4.8 UniversalCharstringValue 273

12.4.4.9 HexstringValue 274

12.4.4.10 RecordValue 275

12.4.4.11 RecordOfValue 275

12.4.4.12 UnionValue 276

12.4.4.13 EnumeratedValue 277

12.4.4.14 VerdictValue 277

12.4.4.15 AddressValue 277

12.4.5 Abstract template mapping 278

12.4.5.0 Mapping rules 278

12.4.5.1 MatchingMechanism 278

12.4.5.2 MatchingList 278

12.4.5.3 ValueRange 278

12.4.5.4 CharacterPattern 279

12.4.5.5 MatchDecodedContent 279

12.4.6 Abstract logging types mapping 279

12.4.6.0 Mapping rules 279

12.4.6.1 TciValueTemplate 279

12.4.6.2 TciNonValueTemplate 280

12.4.6.3 TciValueList 280

12.4.6.4 TciValueDifference 280

12.4.6.5 TciValueDifferenceList 281

12.4.6.6 TciStatusType 281

12.4.6.7 ComponentStatusType 281

12.4.6.8 TimerStatusType 281

12.5 Mapping of interfaces 282

12.5.0 Calling rules 282

12.5.1 TCI-TM interface 282

12.5.1.1 TCI-TM provided 282

12.5.1.2 TCI-TM required 282

12.5.2 TCI-CD interface 283

12.5.2.1 TCI-CD provided 283

12.5.2.2 TCI-CD required 283

12.5.3 TCI-CH interface 283

12.5.3.1 TCI-CH provided 283

12.5.3.2 TCI-CH required 284

12.5.4 TCI-TL interface 285

12.5.4.1 TCI-TL provided 285

12.6 Optional parameters 292

12.7 Error Handling 292

Annex A (normative): IDL Specification of TCI 293

Annex B (normative): XML Mapping for TCI TL Provided 311

B.0 Introduction 311

B.1 TCI‑TL XML Schema for Simple Types 311

B.2 TCI‑TL XML Schema for Types 312

B.3 TCI‑TL XML Schema for Values 314

B.4 TCI‑TL XML Schema for Templates 318

B.5 TCI‑TL XML Schema for Events 321

B.6 TCI‑TL XML Schema for a Log 344

Annex C (informative): Use scenarios 348

C.0 Introduction 348

C.1 Initialization, collecting information, logging 348

C.1.1 Use scenario: initialization 348

C.1.1.0 Scenario description 348

C.1.1.1 Sequence diagram 349

C.1.1.2 TTCN‑3 fragment 349

C.1.2 Use scenario: requesting module parameters 349

C.1.2.0 Scenario description 349

C.1.2.1 Sequence diagram 350

C.1.2.2 TTCN‑3 fragment 350

C.1.3 Use scenario: logging 350

C.1.3.0 Scenario description 350

C.1.3.1 Sequence diagram 351

C.1.3.2 TTCN‑3 fragment 351

C.2 Execution of test cases and control 351

C.2.1 Use scenario: execution of control 351

C.2.1.0 Scenario description 351

C.2.1.1 Sequence diagram 352

C.2.1.2 TTCN‑3 fragment 352

C.2.2 Use scenario: test case execution within control 352

C.2.2.0 Scenario description 352

C.2.2.1 Sequence diagram 353

C.2.2.2 TTCN‑3 fragment 353

C.2.3 Use scenario: direct test case execution 353

C.2.3.0 Scenario description 353

C.2.3.1 Sequence diagram 354

C.2.3.2 TTCN‑3 fragment 354

C.2.4 Use scenario: execute test case to TRI 354

C.2.4.0 Scenario description 354

C.2.4.1 Sequence diagram 355

C.2.4.2 TTCN‑3 fragment 355

C.3 Component handling 355

C.3.1 Use scenario: local control component creation 355

C.3.1.0 Scenario description 355

C.3.1.1 Sequence diagram 356

C.3.1.2 TTCN‑3 fragment 356

C.3.2 Use scenario: remote control component creation 356

C.3.2.0 Scenario description 356

C.3.2.1 Sequence diagram 357

C.3.2.2 TTCN‑3 fragment 357

C.3.3 Use scenario: local MTC creation 357

C.3.3.0 Scenario description 357

C.3.3.1 Sequence diagram 358

C.3.3.2 TTCN‑3 fragment 358

C.3.4 Use scenario: remote MTC creation 358

C.3.4.0 Scenario description 358

C.3.4.1 Sequence diagram 359

C.3.4.2 TTCN‑3 fragment 359

C.3.5 Use scenario: component handling for test case execution within control 359

C.3.5.0 Scenario description 359

C.3.5.1 Sequence diagram 360

C.3.5.2 TTCN‑3 fragment 360

C.3.6 Use scenario: component handling for direct test case execution 361

C.3.6.0 Scenario description 361

C.3.6.1 Sequence diagram 361

C.3.6.2 TTCN‑3 fragment 362

C.3.7 Use scenario: propagation of map/connect 362

C.3.7.0 Scenario description 362

C.3.7.1 Sequence diagram 362

C.3.7.2 TTCN‑3 fragment 362

C.3.8 Use scenario: propagation of unmap/disconnect 363

C.3.8.0 Scenario description 363

C.3.8.1 Sequence diagram 363

C.3.8.2 TTCN‑3 fragment 363

C.4 Termination of test cases and control 363

C.4.1 Use scenario: stop a test case 363

C.4.1.0 Scenario description 363

C.4.1.1 Sequence diagram 364

C.4.1.2 TTCN‑3 fragment 364

C.4.2 Use scenario: stop control 364

C.4.2.0 Scenario description 364

C.4.2.1 Sequence diagram 365

C.4.2.2 TTCN‑3 fragment 365

C.4.3 Use scenario: termination of control after error 365

C.4.3.0 Scenario description 365

C.4.3.1 Sequence diagram 366

C.4.3.2 TTCN‑3 fragment 366

C.4.4 Use scenario: termination of a test case after error 366

C.4.4.0 Scenario description 366

C.4.4.1 Sequence diagram 367

C.4.4.2 TTCN‑3 fragment 368

C.4.5 Use scenario: reset 368

C.4.5.0 Scenario description 368

C.4.5.1 Sequence diagram 368

C.4.5.2 TTCN‑3 fragment 368

C.5 Communication 368

C.5.1 Use scenario: local intercomponent communication 368

C.5.1.0 Scenario description 368

C.5.1.1 Sequence diagram 369

C.5.1.2 TTCN‑3 fragment 369

C.5.2 Use scenario: internode communication between test components 370

C.5.2.0 Scenario description 370

C.5.2.1 Sequence diagram 370

C.5.2.2 TTCN‑3 fragment 370

C.5.3 Use scenario: encoding 371

C.5.3.0 Scenario description 371

C.5.3.1 Sequence diagram 371

C.5.3.2 TTCN‑3 fragment 371

C.5.4 Use scenario: decoding 371

C.5.4.0 Scenario description 371

C.5.4.1 Sequence diagram 372

C.5.4.2 TTCN‑3 fragment 372

Annex D (informative): Bibliography 373

History 374

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

The present document is part 6 of a multi-part deliverable. Full details of the entire series can be found in part 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](https://portal.etsi.org/Services/editHelp!/Howtostart/ETSIDraftingRules.aspx) (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 specifies the control interfaces for TTCN‑3 test system implementations. The TTCN‑3 Control Interfaces provide a standardized adaptation for management, test component handling and encoding/decoding of a test system to a particular test platform. The present document defines the interfaces as a set of operations independent of a target language.

The interfaces are defined to be compatible with the TTCN‑3 standard (see clause 2). The interface definition uses the CORBA Interface Definition Language (IDL) to specify the TCI completely. Clauses 8, 9, 10, 11 and 12 present language mappings for this abstract specification to the target languages Java™, ANSI C, C++, XML and C#. A summary of the IDL‑based interface specification is provided in annex A.

NOTE: Java™ is the trade name of a programming language developed by Oracle Corporation. This information is given for the convenience of users of the present document and does not constitute an endorsement by ETSI of the programming language named. Equivalent programming languages may be used if they can be shown to lead to the same results.

# 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] 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".

[5] ISO/IEC 10646: "Information technology -- Universal Coded Character Set (UCS)".

[6] CORBA 3.0: "Common Object Request Broker Architecture: Core Specification", OMG Formal Document (specifies IDL).

[7] Sun Microsystems: "The Java™ Language Specification".

NOTE: See at <https://docs.oracle.com/javase/specs/>.

[8] ISO/IEC 9899: "Information technology - Programming languages - C".

[9] ISO/IEC 14882:2017: " Programming languages - C++".

[10] W3C Recommendation: "XML Schema Part 0: Primer".

NOTE: See at <http://www.w3.org/TR/xmlschema-0/>.

[11] W3C Recommendation: "XML Schema Part 1: Structures".

NOTE: See at <http://www.w3.org/TR/xmlschema-1/>.

[12] W3C Recommendation: "XML Schema Part 2: Datatypes".

NOTE: See at <http://www.w3.org/TR/xmlschema-2/>.

[13] ECMA-334: "C# Language Specification".

NOTE: See at <http://www.ecma-international.org/publications/standards/Ecma-334.htm>.

## 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.

Not applicable.

# 3 Definition of terms, symbols and abbreviations

## 3.1 Terms

For the purposes of the present document, the terms given in Recommendation ITU‑T X.290 [4] and the following apply:

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

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

**Coding/Decoding (CD):** entity that administers the value and type handling including encoding and decoding in the TTCN‑3 test system

**communication port:** abstract mechanism facilitating communication between test components

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

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

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

**Executable Test Suite (ETS):** Refer to Recommendation ITU‑T X.290 [4].

**Implementation eXtra Information for Testing (IXIT):** Refer to Recommendation ITU‑T X.290 [4].

**Platform Adaptor (PA):** entity that adapts the TTCN‑3 Executable to a particular execution platform

NOTE: The Platform Adaptor creates a single notion of time for a TTCN‑3 test system, and implements both, explicit and implicit, timers as well as external functions.

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

**SUT Adaptor (SA):** entity that adapts the TTCN‑3 communication operations with the SUT based on an abstract test system interface

NOTE: It implements the real test system interface.

**System Under Test (SUT):** Refer to Recommendation ITU‑T X.290 [4].

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

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

**Test Logging (TL):** entity which provides logging information about test execution (including also the information provided by the TTCN‑3 log statement)

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

**Test Management and Control (TMC):** set of entities providing test management and control; consists of the Test Management (TM), the Component Handling (CH), the Test Logging (TL) and the Coding/Decoding (CD)

NOTE: The TMC is an implementation of TCI.

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

**Test System Interface (TSI):** test component that provides a mapping of the ports available in the (abstract) TTCN‑3 test system to those offered by a real test system

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

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

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

**TTCN‑3 Runtime Interface (TRI):** two interfaces that define the interaction of the TTCN‑3 Executable between the SUT and the Platform Adapter (PA) and the System Adapter (SA) in a test system

## 3.2 Symbols

Void.

## 3.3 Abbreviations

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

ADT Abstract Data Type

ANSI American National Standards Institute

ASCII American Standard Code for Information Interchange

ATS Abstract Test Suite

CD (External) Coding/Decoding

CH Component Handler

CORBA Common Object Request Broker Architecture

CR Carriage Return

ETS Executable Test Suite

FIFO First In First Out

IDL Interface Definition Language

IXIT Implementation eXtra Information for Testing

LF Line Feed

MTC Main Test Component

OMG Object Management Group

PA Platform Adaptor

PTC Parallel Test Component

SA SUT Adaptor

SUT System Under Test

TCI TTCN‑3 Control Interfaces

TE TTCN‑3 Executable

TL Test Logging

TLI Test Logging Interface

TM Test Management

TMC Test Management and Control

TRI TTCN‑3 Runtime Interface

TSI Test System Interface

TTCN‑3 Testing and Test Control Notation Version 3

UML Unified Modelling Language

W3C World Wide Web Consortium

XML eXtensible Markup Language

# 4 Introduction

The present document consists of two distinct parts, the first part describing the structure of a TTCN‑3 test system implementation and the second part presenting the TTCN‑3 Control Interfaces specification.

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

* Test Management and Control (TMC).
* TTCN‑3 Executable (TE).
* SUT Adaptor (SA).
* Platform Adaptor (PA).

The TMC consists itself of three entities: Test Management (TM), Coder/Decoder (CD), and Test Component Handler (CH). In addition, the interaction between these entities, i.e. the corresponding interfaces, is defined.

The second part of the present document specifies the TTCN‑3 Control Interfaces (TCI). The interfaces are defined in terms of operations implemented as part of one entity and called by other test system entities. For each operation, the interface specification defines associated data structures, the intended effect on the test system and any constraints on the usage of the operation. Note that these interface specifications only define interactions between the TE and TM, TE and CD, and TE and CH. For interactions between the TE and SA and the TE and PA please refer to the TTCN‑3 Runtime Interface specification (ETSI ES 201 873‑5 [3]).

# 5 Compliance

The minimum required for a TCI compliant TTCN‑3 test system is to adhere to the interface specification stated in the present document. The TTCN‑3 semantics in the test system shall adhere to the operational semantics defined in ETSI ES 201 873‑4 [2]. In addition, one language mapping shall be supported. For example, if a vendor supports Java™, the TCI operation calls and implementations, which are part of the TTCN‑3 executable, shall comply with the IDL to Java™ mapping specified in the present document. For the logging interface, the XML mapping can be used instead of the Java™ or the C mapping.

# 6 General structure of a TTCN‑3 test system

## 6.1 Entities in a TTCN‑3 test system

### 6.1.0 Types of entities

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

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

The structure of a TTCN‑3 test system implementation is illustrated in figure 1.



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

As shown in figure 1, the TTCN‑3 Executable (TE), also referred to as the Executable Test Suite (ETS), interprets and executes TTCN‑3 modules. Various TE structural elements can be identified: control, behaviour, components, types, values and queues. The structural elements within the TE represent functionality that is defined within a TTCN‑3 module or by the TTCN‑3 standard (ETSI ES 201 873‑1 [1]) itself. For example, the structural element "Control" represents the control part within a TTCN‑3 module, while the structural element "Queues" represents the requirement on a TTCN‑3 Executable that each port of a test component maintains its own port queue. While the first is specified within a TTCN‑3 module, the latter is required by the TTCN‑3 specification.

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

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

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

The SA and PA and their interaction with the TE are defined in ETSI ES 201 873‑5 [3]. The TCI specification defines the interaction between the TE and the TMC.

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



Figure : Detailed View on TL

### 6.1.1 Test Management and Control (TMC)

#### 6.1.1.0 Test Management and Control Entities

The TMC entity includes functionality related to management of:

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

#### 6.1.1.1 Test Management (TM)

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

#### 6.1.1.2 Coding and Decoding (CD)

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

#### 6.1.1.3 Component Handling (CH)

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

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

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



Figure 3: General structure of a distributed TTCN‑3 test system

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

NOTE 2: As stated in ETSI ES 201 873-4 [2], a test system executes at most one test case at a given point in time, i.e. a TTCN‑3 module cannot execute multiple test cases at the same time.

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

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

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

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

Additional communication is needed to implement the distribution of test components onto several nodes. Component management communication includes the indication of the creation of test components, the starting of execution of a test component, verdict distribution, as well as component termination indication. The CH does not implement the TTCN‑3 component behaviour. Rather, it implements the communication between several components implemented by a TE.

#### 6.1.1.4 Test Logging (TL)

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

### 6.1.2 TTCN‑3 Executable (TE)

The TE entity executes or interprets a TTCN‑3 module. Conceptually, the TE can be decomposed into six interacting entities: a Control, Behaviour, Component, Type, Value and Queue entity. This structural decomposition of the TE is defined in ETSI ES 201 873‑5 [3]. The terminology for TE defined in ETSI ES 201 873‑5 [3] is used within the present document.

### 6.1.3 SUT Adaptor (SA)

The SA is the implementation of the System under Test Adaptor (SA) as defined in ETSI ES 201 873‑5 [3]. The terminology for SA defined in ETSI ES 201 873‑5 [3] is used within the present document.

### 6.1.4 Platform Adaptor (PA)

The PA is the implementation of the Platform Adaptor (PA) as defined in ETSI ES 201 873‑5 [3]. The terminology for PA defined in ETSI ES 201 873‑5 [3] is used within the present document.

## 6.2 Execution requirements for a TTCN‑3 test system

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

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

A TCI implementation shall fulfil the above mentioned requirements.

# 7 TTCN‑3 control interface and operations

## 7.1 Overview of the TCI

### 7.1.0 TCI role in a TTCN-3 test system

The clause 7 defines a set of abstract data types used to represent data communicated between the TE and the TMC. In addition, it defines TCI operations in terms of their signatures, when they are to be used and what their effects on the TTCN‑3 test system are.

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

The TCI defines the interaction between the TTCN‑3 Executable (TE), Component Handling (CH), the Test Management (TM), the Coding/Decoding (CD), the Test Logging (TL) entities within a TTCN‑3 test system. It provides means for the TE to:

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

The TCI consists of four sub‑interfaces:

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

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

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

#### 7.1.1.0 Mapping of TTCN-3 operations to TCI operations

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

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

The correlation shown for TTCN‑3 communication operations (i.e. send, call, reply, and raise) only holds if these operations are invoked on a test component port connected to another test component port. The correlation for communication operations that are invoked on test component ports that are mapped to test system interface ports is defined in ETSI ES 201 873‑5 [3].

Table 1: Correlation between TTCN‑3 communication operations and  
TCI operation invocations

| TTCN‑3 Operation Name | TCI Operation Name | TCI Interface Name |
| --- | --- | --- |
| send | tciSendConnected (see note 1) | TCI‑CH Provided |
| tciSendConnectedBC (see note 2) |
| tciSendConnectedMC (see note 3) |
| tciEnqueueMsgConnected | TCI‑CH Required |
| call | tciCallConnected (see note 1) | TCI‑CH Provided |
| tciCallConnectedBC (see note 2) |
| tciCallConnectedMC (see note 3) |
| tciEnqueueCallConnected | TCI‑CH Required |
| reply | tciReplyConnected (see note 1) | TCI‑CH Provided |
| tciReplyConnectedBC (see note 2) |  |
| tciReplyConnectedMC (see note 3) |  |
| tciEnqueueReplyConnected | TCI‑CH Required |
| raise | tciRaiseConnected (see note 1) | TCI‑CH Provided |
| tciRaiseConnectedBC (see note 2) |
| tciRaiseConnectedMC (see note 3) |
| tciEnqueueRaiseConnected | TCI‑CH Required |
| log | tliLog | TCI‑TL Provided |
| NOTE 1: For unicast communication.  NOTE 2: For broadcast communication.  NOTE 3: For multicast communication. | | |

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

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

Table : Correlation between TTCN‑3 test case, test component and  
port operations and TCI operation invocations

| TTCN‑3 Operation Name | TCI Operation Name | TCI Interface Name |
| --- | --- | --- |
| create | tciCreateTestComponentReq | TCI‑CH Provided |
| tciCreateTestComponent | TCI‑CH Required |
| start (a component) | tciStartTestComponentReq | TCI‑CH Provided |
| tciStartTestComponent | TCI‑CH Required |
| stop (a component) | tciStopTestComponentReq | TCI‑CH Provided |
| tciStopTestComponent | TCI‑CH Required |
| kill | tciKillTestComponentReq | TCI‑CH Provided |
| tciKillTestComponent | TCI‑CH Required |
| connect | tciConnectReq | TCI‑CH Provided |
| tciConnect | TCI‑CH Required |
| disconnect | tciDisconnectReq | TCI‑CH Provided |
| tciDisconnect | TCI‑CH Required |
| map | tciMapReq (see note 1) | TCI‑CH Provided |
| tciMapParamReq (see note 2) |  |
| tciMap (see note 1) | TCI‑CH Required |
| tciMapParam (see note 2) |  |
| unmap | tciUnmapReq (see note 1) | TCI‑CH Provided |
| tciUnmapParamReq (see note 2) |  |
| tciUnmap (see note 1) | TCI‑CH Required |
| tciUnmapParam (see note 2) |  |
| running | tciTestComponentRunningReq | TCI‑CH Provided |
| tciTestComponentRunning | TCI‑CH Required |
| alive | tciTestComponentAliveReq | TCI‑CH Provided |
| tciTestComponentAlive | TCI‑CH Required |
| done | tciTestComponentDoneReq | TCI‑CH Provided |
| tciTestComponentDone | TCI‑CH Required |
| killed | tciTestComponentKilledReq | TCI‑CH Provided |
| tciTestComponentKilled | TCI‑CH Required |
| mtc | tciGetMTCReq | TCI‑CH Provided |
| tciGetMTC | TCI‑CH Required |
| execute | tciTestCaseExecuteReq | TCI‑CH Provided |
| tciTestCaseExecute | TCI‑CH Required |
| NOTE 1: For statement without configuration parameter.  note 2: For statement with configuration parameter. | | |

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

##### 7.1.1.3.0 Mapping of TTCN-3 operations to series of TCI operations

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

##### 7.1.1.3.1 Test case stop operation

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

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

## 7.2 TCI data

### 7.2.0 Abstract data types

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

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

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

The concrete representation of these abstract data types as well as the definition of basic data types like string and boolean are defined in the respective language mappings in clauses 8, 9, 10, 11 and 12.

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

### 7.2.1 General abstract data types

#### 7.2.1.0 Use of general abstract data types

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

#### 7.2.1.1 Management

TciModuleIdType A value of TciModuleIdType is the name of a TTCN‑3 module as specified in the TTCN‑3 ATS. This abstract type is used for module handling.

TciModuleParameterIdType A value of TciModuleParameterIdType is the qualified name of a TTCN‑3 module parameter as specified in the TTCN‑3 ATS. This abstract type is used for module parameter handling.

TciTestCaseIdType A value of TciTestCaseIdType is the qualified name of a TTCN‑3 testcase as specified in the TTCN‑3 ATS. This abstract type is used for testcase handling.

TciTestCaseIdListType A value of TciTestCaseIdListType is a list of TciTestCaseIdListType. This abstract type is used when retrieving the list of test cases in a TTCN‑3 module.

TciModuleIdListType A value of type TciModuleIdListType is a list of TciModuleIdType. This abstract type is used when retrieving the list of modules which are imported by a TTCN‑3 module.

TciModuleParameterType A value of type TciModuleParameterType is a structure of TciModuleParameterIdType and Value. This abstract type is used to represent the parameter name and the default value of a module parameter.

TciModuleParameterListType A value of type TciModuleParameterListType is a list of TciModuleParameterType. This abstract type is used when retrieving the module parameters of a TTCN‑3 module.

TciParameterType A value of type TciParameterType includes a TTCN‑3 Value, which can be absent, and a value of TciParameterPassingModeType to represent the name, the value and parameter passing mode specified for the parameter in the TTCN‑3 ATS.

TciParameterPassingModeType A value of type TciParameterPassingModeType is either IN, INOUT, or OUT. This abstract type is used when starting a test case or when the termination of a test case is indicated.

TciParameterListType A value of type TciParameterListType is a list of TciParameterType. This abstract type is used when starting a test case or when the termination of a test case is indicated.

TciParameterTypeType A value of type TciParameterTypeType is a structure of Type and TciParameterPassingModeType. This abstract type is used to represent the type, the name and the parameter passing mode of a test case parameter.

TciParameterTypeListType A value of type TciParameterTypeListType is a list of TciParameterTypeType. This abstract type is used to represent the list of parameters of a test case.

TciTestComponentKindType A value of type TciTestComponentKindType is a literal of the set of kinds of TTCN‑3 test components, i.e. CONTROL, MTC, PTC, SYSTEM, and PTC\_ALIVE. This abstract type is used for component handling.

TciTypeClassType A value of type TciTypeClassType is a literal of the set of type classes in TTCN‑3 such as boolean, float, record, etc. This abstract type is used for value handling.

TciMatchingTypeType A value of type TciMatchingTypeType is a literal of the set of matching mechanism types in TTCN‑3 such as template list, range, AnyValue etc. This abstract type is used for template handling.

#### 7.2.1.2 Communication

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

Additional abstract data types with the prefix Tri are taken from ETSI ES 201 873‑5 [3]: TriPortIdType, TriPortIdListType, TriComponentIdType, TriComponentIdListType, TriAddressType, TriAddressListType, TriTimerIdType and TriMessageType.

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

#### 7.2.2.0 Definition and scope of use

The clause 7.2.2 defines the set of abstract data types that build up the TTCN‑3 type, value and template representation. Functionality of each data type is defined by an accompanying set of operations. Operations on or using this abstract data type return either a value of this abstract type or a basic type like boolean.

All operations have been defined using the Interface Description Language (IDL). Concrete language mappings for the operations on the abstract data types are given in clauses 8, 9, 10, 11 and 12. In certain languages, the application of an operation on an abstract data type is represented by passing (either by‑value or by-reference, depending on the mapping) the concrete value as a parameter to the operation. Other languages might choose other referencing method to the concrete value, e.g. by considering the value as an object on which a method corresponding to the operation is invoked. To indicate the inability to perform a certain task or to indicate the absence of an optional parameter in the following, the distinct value null is used. It can be considered as being a reserved value indicating a special value. The language mappings will define a concrete representation of this distinct value null.

The abstract TTCN‑3 type, value and template representation consists of the following parts:

* an abstract data type Type that represents all TTCN‑3 types in a TTCN‑3 module;
* different abstract data types that represent TTCN‑3 values, i.e. TTCN‑3 values of a given TTCN‑3 type. This can be either values of TTCN‑3 predefined types or of TTCN‑3 user‑defined types;
* different abstract data types that represent matching mechanisms that can occur in TTCN-3 templates;
* other abstract data types that represent complex value properties such a permutations or length restrictions.

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

#### 7.2.2.1 Abstract TTCN‑3 data types

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

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

* reference predefined and user‑defined TTCN‑3 data types; and
* create and maintain TTCN‑3 values and templates.

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

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

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

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

NOTE 2: The naming for a nested type without explicit name can follow the rules defined in clauses 6.2.1.1 and 6.2.3.2 of ETSI ES 201 873‑1 [1], e.g. TypeIdOrExpression.ElementId and TypeId[-], respectively.

NOTE 3: There might be several instances of the abstract data type Type representing the same TTCN-3 type, e.g. in a situation when the type has different attributes in different context.

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

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

NOTE 4: Newly created instances of empty record types are considered to be initialized.

TString getTypeEncoding() Returns the current type encode attribute as defined in the TTCN‑3 module and possibly dynamically restricted by the setencode operation or dynamic\_encoding parameter of codec operations. If no encode attribute is defined, the distinct value null is returned. If the type has more than one encode attributes associated with it, all encode attributes are concatenated to a single string using LF (line feed char(0, 0, 0, 10)), CR (carriage return char(0, 0, 0, 13)) or their combination as a separator.

TString getTypeEncodingVariant() This operation returns the current value encoding variant attribute as defined in the TTCN-3 module and possibly dynamically restricted by the setencode operation or dynamic\_encoding parameter of codec operations, if any. If no encoding variant attribute is defined, the distinct value null is returned. If the type has more than one variant attributes associated with it, all variant attributes are concatenated to a single string using LF (line feed char(0, 0, 0, 10)), CR (carriage return char(0, 0, 0, 13)) or their combination as a separator. In case the type supports multiple encodings, the function returns variants for all encodings. Each variant is prefixed with the associated encode attribute and followed by a full stop character in this case.

TStringSeq getEncodingAttributes() Returns all current encode attributes of the type as defined in the TTCN‑3 module. If no encode attribute is defined the distinct value null is returned.

TStringSeq getVariantAttributes(in TString encoding)  
This operation returns all variant attributes of the type as defined in the TTCN-3 module and possibly dynamically restricted by the setencode operation or dynamic\_encoding parameter of codec operations. If no variant attribute is defined, the distinct value null is returned. The parameter is used to specify encoding the variant attributes are related to. It is required when the type has multiple encodings associated with it. If the type uses a single encoding, the parameter can be set to the special value null. The operation returns the special value null, if the parameter specifies a non-existent encoding or if it contains null and the type contains multiple encodings.

TStringSeq getTypeExtension() Returns the type extension attribute as defined in the TTCN‑3 module.

Value parseValue(in TString val) This operation creates a new value of the given type from a string provided in the parameter. The input string shall use valid TTCN‑3 syntax for values or templates of this type. The only references allowed in the input string are type references. If the input string contains an error, the distinct value null is returned. The operation is an optional part of the TCI and tool vendors are not required to support value parsing. If not supported, parseValue will always return the distinct value null.

NOTE 5: The parseValue operation can be used for defining matching symbols to enable the representation of templates in TCI.

MatchingMechanism newTemplate(TciMatchingType matchingType)  
Returns a freshly created matching mechanism of this type. The matchingType parameter determines what kind of matching mechanism will be created (see clause 7.2.2.3.1 for more details). If the created matching mechanism contains additional data properties, these properties are uninitialized in the created matching mechanism.

RangeBoundary getLowerTypeBoundary()  
Returns the lower range boundary of the type if it has a range-restriction attached to it. Otherwise, the distinct value null is returned.

RangeBoundary getUpperTypeBoundary()  
Returns the lower range boundary of the type if it has a range-restriction attached to it. Otherwise, the distinct value null is returned.

LengthRestriction getTypeLengthRestriction()  
Returns the length restriction of the type if it has a length restriction attached to it. Otherwise, the distinct value null is returned.

MatchingMechanism getTypeMatchingMechanism()  
Returns the matching mechanism of the type if it is restricted by a matching mechanism. Otherwise, the distinct value null is returned.

#### 7.2.2.2 Abstract TTCN‑3 values

##### 7.2.2.2.0 Basic rules

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

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

Type

Value

MatchingMechanism

IntegerValue

FloatValue

BooleanValue

CharstringValue

UniversalCharstringValue

BitstringValue

OctetstringValue

HexstringValue

RecordOfValue

RecordValue

UnionValue

EnumeratedValue

VerdictValue

AddressValue

MatchingList

ValueRange

CharacterPattern

MatchDecodedContent

Figure 4: Hierarchy of abstract values

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

In addition, Value can be used to represent matching mechanisms, which are used instead or inside values e.g. in template parameters or for template variables. Two additional operations: isMatchingSymbol (returns true for matching symbols) and valueToString (for printing value content in the same way as the any2unistr predefined function; can be used for displaying value content) are defined. These operations are not mandatory - it is up to a tool vendor to support them or not.

Values using @lazy and @fuzzy modifiers are represented by the Value data type too. However, it is not possible to use the Value data type to perform evaluation of these values; evaluation can be performed by the TE only. If a @lazy or @fuzzy value has been assigned, but it does not contain result of the evaluation, any data access operations shall result in an error.

##### 7.2.2.2.1 The abstract data type Value

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

Type getType() Returns the type of the specified value.

TBoolean notPresent() Returns true if the specified value is omit, false otherwise.

TString getValueEncoding() Returns the current value encode attribute as defined in the TTCN-3 module and possibly dynamically restricted by the setencode operation or dynamic\_encoding parameter of codec operations, if any. If no encoding attribute is defined, the distinct value null is returned. If the value has more than one encode attributes associated with it, all encode attributes are concatenated to a single string using LF (line feed char(0, 0, 0, 10)), CR (carriage return char(0, 0, 0, 13)) or their combination as a separator.

TString getValueEncodingVariant() Returns the current value encoding variant attribute as defined in the TTCN-3 module and possibly dynamically restricted by the setencode operation or dynamic\_encoding parameter of codec operations, if any. If no encoding variant attribute is defined, the distinct value null is returned. If the value has more than one variant attributes associated with it, all variant attributes are concatenated to a single string using LF (line feed char(0, 0, 0, 10)), CR (carriage return char(0, 0, 0, 13)) or their combination as a separator. In case the value supports multiple encodings, the function returns variants for all encodings. Each variant will be prefixed with the associated encode attribute and followed by a full stop character in this case.

TStringSeq getEncodingAttributes() Returns all current encode attributes of the value as defined in the TTCN‑3 module and possibly dynamically restricted by the setencode operation or dynamic\_encoding parameter of codec operations. If no encode attribute is defined, the distinct value null is returned.

TStringSeq getVariantAttributes(in TString encoding)  
This operation returns all current variant attributes of the value as defined in the TTCN-3 module and possibly dynamically restricted by the setencode operation or dynamic\_encoding parameter of codec operations. If no variant attribute is defined, the distinct value null is returned. The parameter is used to specify encoding the variant attributes are related to. It is required when the value has multiple encodings associated with it. If the type uses a single encoding, the parameter can be set to the special value null. The operation returns the distinct value null, if the parameter specifies a non-existent encoding or if it contains null and the value contains multiple encodings.

TBoolean isMatchingSymbol() Returns true if the instance is of the MatchingMechanism abstract data type (or any other abstract data type derived from the MatchingMechanism data type) and false in all other cases.

NOTE: This method can be used for detecting the exact abstract data type of the instance. If the method returns false, it is safe to assume that the instance is one of the abstract value data types defined in clause 7.2.2.2. If the method returns true, the instance is one of the matching mechanism defined in clause 7.2.2.3.

TString valueToString() Returns the same string as produced by the any2unistr predefined function (specified in clause C.1.33 of ETSI ES 201 873‑1 [1]) with the specified value as the invalue parameter and the format parameter equal to "canonical".

TBoolean isFuzzy () Returns true if the specified value has the @fuzzy modifier, false otherwise.

TBoolean isLazy () Returns true if the specified value has the @lazy modifier, false otherwise.

TBoolean isEvaluated () Returns true if the value has been evaluated and its data content is available, false otherwise. In case of uninitialized values, false is always returned. The method is typically used for @lazy values, and it returns false for values that have been assigned, but not evaluated yet and true if the value contains the evaluation result. The method returns false for @fuzzy values, as the result of evaluation is never stored by the TE. For all other values, the method returns true.

LengthRestriction getLengthRestriction()  
Returns a length restriction matching attribute (specified in clause B.1.4.1 of ETSI ES 201 873‑1 [1]) in case it is attached to the value or the distinct value null if no such matching attribute is present.

void setLengthRestriction(LengthRestriction restriction)  
Adds a length restriction matching attribute (specified in clause B.1.4.1 of ETSI ES 201 873‑1 [1]) to the value or modifies an existing one. The distinct value null can be used as a parameter to disable an existing length restriction.

TBoolean isIfPresentEnabled() Returns true if the ifpresent indicator (specified in clause B.1.4.2 of ETSI ES 201 873‑1 [1]) is attached to the value and false otherwise.

void setIfPresentEnabled(TBoolean enabled)  
Sets the whether the ifpresent indicator (specified in clause B.1.4.2 of ETSI ES 201 873‑1 [1]) is attached to the value or not.

RangeBoundary getLowerTypeBoundary()  
Returns the lower range boundary of the value's type if it has a range-restriction attached to it. Otherwise, the distinct value null is returned.  
This is only applicable for values with types of typeclass INTEGER and FLOAT.

RangeBoundary getUpperTypeBoundary()  
Returns the lower range boundary of the value's type if it has a range-restriction attached to it. Otherwise, the distinct value null is returned.  
This is only applicable for values with types of typeclass INTEGER or FLOAT.

LengthRestriction getTypeLengthRestriction()  
Returns a length restriction matching attribute (specified in clause B.1.4.1 of ETSI ES 201 873‑1 [1]) in case it is attached to the value's type or the distinct value null if no such matching attribute is present.  
This is only applicable for values with types of typeclass CHARSTRING, UNIVERSAL\_CHARSTRING, BITSTRING, HEXSTRING, OCTETSTRING, RECORD\_OF, SET\_OF or ARRAY.

MatchingMechanism getTypeMatchingMechanism()  
Returns the matching mechanism (see clause 7.2.2.3.1) of the value's type if it is restricted by a subtype specification attribute (specified in clauses 6.1.2 and 6.2.13 of ETSI ES 201 873‑1 [1]). Otherwise, the distinct value null is returned.

TBoolean isOptional() Returns true if and only if the value is either an optional field or a template without value or present template restriction.

When working with length restriction data using the getLengthRestriction and setLengthRestriction, methods, no assumption shall be made on how the data are stored in a value. An internal implementation might choose to use a reference to the data or copy the data. It is safe to assume that the data are copied. Therefore, it should be assumed that subsequent modifications of the length restriction data will not be considered in the value object.

##### 7.2.2.2.2 The abstract data type IntegerValue

The abstract data type IntegerValue is based on the abstract data type Value. It represents TTCN‑3 integer values.

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

TInteger getInt() Returns the integer value of this TTCN‑3 integer.

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

##### 7.2.2.2.3 The abstract data type FloatValue

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

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

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

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

##### 7.2.2.2.4 The abstract data type BooleanValue

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

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

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

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

##### 7.2.2.2.5 The abstract data type CharstringValue

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

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

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

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

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

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

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

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

##### 7.2.2.2.6 The abstract data type UniversalCharstringValue

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

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

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

void setString(in TString value) Sets the value of this UniversalCharstringValue according to the textual representation as defined by value.

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

void setChar(in TInteger position, in TUniversalChar value)  
Sets the universal char at position to value. Valid values for position are from 0 to length - 1.

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

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

##### 7.2.2.2.7 The abstract data type BitstringValue

The abstract data type BitstringValue is based on the abstract data type Value. It represents TTCN‑3 bitstring values. This abstract data type uses a parameter position in some of its operations for addressing individual bits or embedded matching mechanisms. The following rules are valid in this case:

* Position 0 denotes the first bit or matching mechanism of the TTCN‑3 bitstring
* Valid values for position are from 0 to length – 1, where length is the total number of bits and individual matching mechanisms in the value

Each individual matching mechanism takes exactly one position regardless of how many bits it can match.

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

TString getString() Returns the textual representation of this BitstringValue, as defined in TTCN‑3. E.g. the textual representation of 0101 is '0101'B. The textual representation of the empty TTCN‑3 bitstring is ''B, while its length is zero.

void setString(in TString value) Sets the value of this BitstringValue according to the textual representation as defined by value. E.g. the value of this BitstringValue is 0101 if the textual representation in value is '0101'B. The parameter may contain allowed matching symbols such as AnyElement or AnyElementsOrNone.

TChar getBit(in TInteger position) Returns the value (0 | 1) at position of this TTCN‑3 bitstring as a character. An error will occur when the specified position contains a matching mechanism.

void setBit(in TInteger position, in TInteger value)  
Sets the bit at position to the value (0 | 1).

TInteger getLength() Returns the length of this BitstringValue. The length is equal to the total number of bits and individual matching mechanisms in the value. If the value of this BitstringValue is omit, the length is equal to 0. Each individual matching mechanism is considered to have the length of one bit regardless of how many bits it can match.

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

TBoolean isMatchingAt(in TInteger position)  
Returns true if the item at position of this TTCN‑3 bitstring is a matching mechanism inside a value (AnyElement, AnyElementsOrNone) and false otherwise.

MatchingMechanism getMatchingAt(in TInteger position)  
If the position of this TTCN‑3 bitstring contains a matching mechanism inside a value (AnyElement, AnyElementsOrNone), the method returns it. Otherwise the distinct value null is returned.

void setMatching(in TInteger position, in MatchingMechanism template)  
Sets a matching mechanism at position. Only two matching mechanisms are allowed: AnyElement and AnyElementsOrNone.

##### 7.2.2.2.8 The abstract data type OctetstringValue

The abstract data type OctetstringValue is based on the abstract data type Value. It represents TTCN‑3 octetstring values. This abstract data type uses a parameter position in some of its operations for addressing individual octets or embedded matching mechanisms. The following rules are valid in this case:

* Position 0 denotes the first octet or matching mechanism of the TTCN‑3 octetstring
* Valid values for position are from 0 to length – 1, where length is the total number of octets and individual matching mechanisms in the value
* Each individual matching mechanism takes exactly one position regardless of how many octets it can match

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

TString getString() Returns the textual representation of this OctetstringValue, as defined in TTCN‑3. E.g. the textual representation of 0xCAFFEE is 'CAFFEE'O. The textual representation of the empty TTCN‑3 octetstring is ''O, while its length is zero.

void setString(in TString value) Sets the value of this OctetstringValue according to the textual representation as defined by value. E.g. The value of this OctetstringValue is 0xCAFFEE if the textual representation in value is 'CAFFEE'O. The parameter may contain allowed matching symbols such as AnyElement or AnyElementsOrNone.

TChar getOctet(in TInteger position)  
Returns the value (0..255) at position of this TTCN‑3 octetstring. An error will occur when the specified position contains a matching mechanism.

void setOctet(in TInteger position, in TInteger value)  
Sets the octet at position to value (0..255).

TInteger getLength() Returns the length of this OctetstringValue. The length is equal to the total number of octets and individual matching mechanisms in the value. If the value of this OctetstringValue is omit, the length is equal to 0. Each individual matching mechanism is considered to have the length of one octet regardless of how many octets it can match.

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

TBoolean isMatchingAt(in TInteger position)  
Returns true if the item at position of this TTCN‑3 octetstring is a matching mechanism inside a value (AnyElement, AnyElementsOrNone) and false otherwise.

MatchingMechanism getMatchingAt(in TInteger position)  
If the position of this TTCN‑3 octetstring contains a matching mechanism inside a value (AnyElement, AnyElementsOrNone), the method returns it. Otherwise the distinct value null is returned.

void setMatching(in TInteger position, in MatchingMechanism template)  
Sets a matching mechanism at position. Only two matching mechanisms are allowed: AnyElement and AnyElementsOrNone.

##### 7.2.2.2.9 The abstract data type HexstringValue

The abstract data type HexstringValue is based on the abstract data type Value. It represents TTCN‑3 hexstring values. This abstract data type uses a parameter position in some of its operations for addressing individual hex digits or embedded matching mechanisms. The following rules are valid in this case:

* Position 0 denotes the first hex digit or matching mechanism of the TTCN‑3 hexstring
* Valid values for position are from 0 to length – 1, where length is the total number of hex digits and individual matching mechanisms in the value

Each individual matching mechanism takes exactly one position regardless of how many hex digits it can match.

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

TString getString() Returns the textual representation of this HexstringValue, as defined in TTCN‑3. E.g. the textual representation of 0xAFFEE is 'AFFEE'H. The textual representation of the empty TTCN‑3 hexstring is ''H, while its length is zero.

void setString(in TString value) Sets the value of this HexstringValue according to the textual representation as defined by value. E.g. The value of this HexstringValue is 0xAFFEE if the textual representation in value is 'AFFEE'H. The parameter may contain allowed matching symbols such as AnyElement or AnyElementsOrNone.

TChar getHex(in TInteger position) Returns the value (0..15) at position of this TTCN‑3 hexstring. An error will occur when the specified position contains a matching mechanism.

void setHex(in TInteger position, in TInteger value)  
Sets the hex digit at position to value (0..15).

TInteger getLength() Returns the length of this HexstringValue. The length is equal to the total number of hex digits and individual matching mechanisms in the value. If the value of this HexstringValue is omit, the length is equal to 0. Each individual matching mechanism is considered to have the length of one hex digit regardless of how many hex digits it can match.

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

TBoolean isMatchingAt(in TInteger position)  
Returns true if the item at position of this TTCN‑3 hexstring is a matching mechanism inside a value (AnyElement, AnyElementsOrNone) and false otherwise.

MatchingMechanism getMatchingAt(in TInteger position)  
If the position of this TTCN‑3 hexstring contains a matching mechanism inside a value (AnyElement, AnyElementsOrNone) the method returns it. Otherwise the distinct value null is returned.

void setMatching(in TInteger position, in MatchingMechanism template)  
Sets a matching mechanism at position. Only two matching mechanisms are allowed: AnyElement and AnyElementsOrNone.

##### 7.2.2.2.10 The abstract data type RecordValue

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

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

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

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

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

void setField(in TString fieldName, in Value value)  
Sets the field named fieldName of the record to value. No assumption shall be made on how a field is stored in a record. An internal implementation might choose to store a reference to this value or to copy the value. It is safe to assume that the value is copied. Therefore it should be assumed that subsequent modifications of value will not be considered in the record. Using a MatchingMechanism of the OMIT\_TEMPLATE type as the value parameter has the same effect as calling the setFieldOmitted with the fieldName as a parameter.

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

void setFieldOmitted(in TString fieldName)  
Mark the referenced field of the record as being omitted.

##### 7.2.2.2.11 The abstract data type RecordOfValue

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

This abstract data type uses a parameter position in some of its operations for addressing individual elements or embedded matching mechanisms. The following rules are valid in this case:

* Position 0 denotes the first element or matching mechanism of the record of, set of or array value.
* Valid values for position are from 0 to length – 1, where length is the total number of elements and individual matching mechanisms in the record of, set of and or array value.
* In case of array, element indices start from 0, independent of the lower index bound.
* Each individual matching mechanism with exception permutations takes exactly one position regardless of how many elements it can match.
* Permutation takes exactly as many positions as many items it contains: each value or matching mechanism inside it takes exactly one position.

When working with instances received from the get methods (getField, getPermutation), and passed to the set methods (setField, appendField, setPermutation), no assumption shall be made on how the data are stored in a record of. An internal implementation might choose to use a reference to the data or to copy the data. It is safe to assume that the data are copied. Therefore, it should be assumed that subsequent modifications of the field or permutation data will not be considered in the record of.

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

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

void setField(in TInteger position, in Value value)  
Sets the field at position to value. If position is greater than (length - 1) the record of is extended to have the length (position + 1). The record of elements between the original position at length and position - 1 is set to omit.

void appendField(in Value value) Appends the value at the end of the record of, i.e. at position length.

Type getElementType() Returns the Type of the elements of this record of.

TInteger getLength() Returns the actual length of the record of value, zero if the record of value is omit.

void setLength(in TInteger len) Sets the length of the record of to len. If len is greater than the original length, newly created elements have the value omit. If len is less or equal than the original length this operation is ignored.

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

TInteger getPermutationCount() Returns the number of permutations in the record of or array value.

Permutation getPermutation(TInteger index)  
Returns the permutation at the specified index. The allowed index range is from 0 to (getPermutationCount() – 1).

void definePermutation(Permutation permutation)  
Creates permutation from existing elements of a record of value. The permutation parameter shall not include a direct or indirect reference to elements that are already a part of other existing permutations attached to the same record of. In particular, no element with an index greater or equal to permutation.getStartPosition() and less than permutation.getStartPosition() + permutation.getLength() may be a part of a different permutation. No elements are added to the record of by this operation.

void removePermutation(TInteger index)  
Removes the permutation at the specified index. The allowed index range is from 0 to (getPermutationCount() – 1). No elements are removed from the record of by this operation. When the operation completes, the existing elements at positions specified by the removed permutation do not belong to any permutation.

void clearPermutations() Removes all permutations from the value. The elements that belonged to the removed permutation are not removed.

##### 7.2.2.2.12 The abstract data type UnionValue

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

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

Value getVariant(in TString variantName)  
Returns the value of the TTCN‑3 union variantName, if variantName equals the result of getPresentVariantName, the distinct value null otherwise. variantName denotes the name of the union variant as defined in the TTCN-3 module.

void setVariant(in TString variantName, in Value value)  
Sets variantName of the union to value. If variantName is not defined for this union this operation is ignored. If another variant was selected the new variant is selected instead.

TString getPresentVariantName() Returns a String representing the currently selected variant name in the given TTCN‑3 union. The distinct value null is returned if no variant is selected.

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

##### 7.2.2.2.13 The abstract data type EnumeratedValue

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

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

TString getEnum() Returns the string identifier of this EnumeratedValue. This identifier equals the identifier in the TTCN‑3 specification.

void setEnum(in TString enumValue) Sets the enum to enumValue. If enumValue is not an allowed value for this enumeration the operation is ignored.

TInteger getInt() Returns the integer value of this EnumeratedValue. This integer equals the user-assigned integer value in the TTCN‑3 specification or the automatically assigned integer value.

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

##### 7.2.2.2.14 The abstract data type VerdictValue

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

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

TInteger getVerdict() Returns the integer value for this VerdictValue. The integer is one of the following constants: ERROR, FAIL, INCONC, NONE, PASS, USER\_ERROR.

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

##### 7.2.2.2.15 The abstract data type AddressValue

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

Value getAddress() Returns the address value, which will no longer be of type class ADDRESS but rather of the actual type used for address.

void setAddress(in Value value) Sets this address value to value.

#### 7.2.2.3 Abstract TTCN‑3 matching mechanisms

##### 7.2.2.3.1 The abstract data type MatchingMechanism

In case a template can occur in a TCI operation in the same place as a value (e.g. in a function call).

TTCN‑3 matching mechanisms are represented at the TCI interfaces via numerous abstract data types. All matching mechanisms share the same abstract data type MatchingMechanism. This data type is based on the abstract data type Value and inherits all its operations. While simpler matching mechanisms that do not have any additional properties are represented by the MatchingMechanism data type, more complex ones have their own dedicated data types. These data types are derived from the MatchingMechanism data type implicitly inherit all its operations.

The abstract data type MatchingMechanism defines operations that are common for all TCI matching mechanisms. It is also used to represent simple matching mechanisms with no additional properties, in particular: AnyValue, AnyValueOrNone, AnyElement, AnyElementsOrNone, omit.

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

TciMatchingTypeType getMatchingType()  
Returns the matching mechanism type. A value of TciMatchingTypeType can have one of the following constants: TEMPLATE\_LIST, COMPLEMENTED\_LIST, ANY\_VALUE, ANY\_VALUE\_OR\_NONE, VALUE\_RANGE, SUBSET, SUPERSET, ANY\_ELEMENT, ANY\_ELEMENTS\_OR\_NONE, PATTERN, MATCH\_DECODED\_CONTENT, OMIT\_TEMPLATE.

NOTE: OMIT\_TEMPLATE is used to represent an omit matching symbol assigned to a top-level template. Omitted fields of record templates are represented in a different way (see clause 7.2.2.2.10 for more details).

##### 7.2.2.3.2 The abstract data type MatchingList

The abstract data type MatchingList is used to represent matching mechanisms that contain a list of items of the same type: template list, complemented template list, SubSet and SuperSet.

This abstract data type uses a parameter position in some of its operations for addressing individual values or templates inside this matching mechanism. The following rules are valid in this case:

* Position 0 denotes the first value or template in the TTCN‑3 matching mechanism
* Valid values for position are from 0 to (size() – 1)

When working with instances received from the get method and passed to the set methods (add, insert), no assumption shall be made on how the data are stored in a matching mechanism. An internal implementation might choose to use a reference to the data or to copy the data. It is safe to assume that the data are copied. Therefore, it should be assumed that subsequent modifications of the data will not be considered in the matching mechanism.

The following operations are defined on the base abstract data type MatchingList:

TInteger size() Returns the number of items in the matching mechanism.

Value get(TInteger position) Returns the value or template at the specified position inside the matching mechanism.

void add(Value item) Adds a new item to the end of the matching list, increasing its size.

void remove(TInteger position) Removes the value or template at the specified position inside the matching mechanism.

void clear() Removes all values and templates from the matching mechanism.

##### 7.2.2.3.3 The abstract data type ValueRange

The abstract data type ValueRange is used to represent the value range matching mechanism.

When working with instances received from the get methods (getLowerBoundary, getUpperBoundary) and passed to the set methods (setLowerBoundary, setUpperBoundary), no assumption shall be made on how the data are stored in a matching mechanism. An internal implementation might choose to use a reference to the data or to copy the data. It is safe to assume that the data are copied. Therefore, it should be assumed that subsequent modifications of the data will not be considered in the ValueRange instance.

The following operations are defined on the base abstract data type ValueRange:

RangeBoundary getLowerBoundary() Returns the lower boundary of the range.

RangeBoundary getUpperBoundary() Returns the upper boundary of the range.

void setLowerBoundary(RangeBoundary lowerBoundary)  
Sets the lower boundary of the range.

void setUpperBoundary(RangeBoundary upperBoundary)  
Sets the upper boundary of the range.

##### 7.2.2.3.4 The abstract data type CharacterPattern

The abstract data type CharacterPattern is used to represent the character pattern matching mechanism.

When working with instances received from the getPatternString method and passed to the setPatternString method, no assumption shall be made on how the data are stored in a matching mechanism. An internal implementation might choose to use a reference to the data or to copy the data. It is safe to assume that the data are copied. Therefore, it should be assumed that subsequent modifications of the data will not be considered in the CharacterPattern instance.

The following operations are defined on the base abstract data type CharacterPattern:

Value getPatternString () Returns the character pattern definition of this pattern (either a CharstringValue or UniversalCharstringValue).

void setPatternString(Value characterPattern)  
Sets the character pattern definition of this pattern. The characterPattern parameter shall contain either a CharstringValue or UniversalCharstringValue.

##### 7.2.2.3.5 The abstract data type MatchDecodedContent

The abstract data type MatchDecodedContent is used to represent the decmatch matching mechanism.

When working with instances received from the getContent method and passed to the setContent method, no assumption shall be made on how the data are stored in a matching mechanism. An internal implementation might choose to use a reference to the data or to copy the data. It is safe to assume that the data are copied. Therefore, it should be assumed that subsequent modifications of the data will not be considered in the DecodedMatch instance.

The following operations are defined on the base abstract data type DecodedMatch:

Value getContent() Returns the value or matching mechanism used as an argument of the decmatch matching mechanism.

void setContent(Value content) Sets the value or matching mechanism that is used as an argument of the decmatch matching mechanism.

#### 7.2.2.4 Data types for complex TTCN‑3 properties

##### 7.2.2.4.0 Scope of use of TTCN-3 properties

The abstract data types described in the clause 7.2.2.4 are used to describe complex properties of TTCN-3 values and matching mechanisms.

##### 7.2.2.4.1 The abstract data type LengthRestriction

The abstract data type LengthRestiction is used to represent the length restriction matching attribute.

The following operations are defined on the base abstract data type LengthRestriction:

TInteger getLowerBoundary() Returns the lower boundary of the length restriction.

TInteger getUpperBoundary() Returns the upper boundary of the length restriction.

void setLowerBoundary(TInteger boundary)  
Sets the lower boundary value.

void setUpperBoundary(TInteger boundary)  
Sets the upper boundary value.

TBoolean isUpperBoundaryInfinity()  
Returns true if the upper boundary contains infinity and false otherwise.

void setInfiniteUpperBoundary() Sets the upper boundary to infinity.

##### 7.2.2.4.2 The abstract data type Permutation

The abstract data type Permutation is used to describe properties of a permutation matching mechanism embedded in a RecordOfValue instance.

The following operations are defined on the base abstract data type Permutation:

TInteger getStartPosition() Returns the position of the first item of the permutation in the RecordOfValue.

setStartPosition(TInteger position)  
Sets the position of the first item of the permutation in the RecordOfValue.

TInteger getLength() Returns the number of elements or matching mechanisms of the RecordOfValue that are included in the permutation.

void setLength(TInteger length) Sets the number of elements or matching mechanisms of the RecordOfValue that are included in the permutation.

##### 7.2.2.4.3 The abstract data type RangeBoundary

The abstract data type RangeBoundary is used to describe properties of lower and upper bound of ValueRange instances.

The following operations are defined on the base abstract data type RangeBoundary:

Value getBoundary() Returns the boundary value. Dependent on the type of the value range, the return value can be either an IntegerValue, FloatValue, CharstringValue or UniversalCharstringValue. If the boundary is undefined or it cannot be represented by a Value instance (infinity in case of integer values), the distinct value null is returned.

TBoolean isInclusive() Returns true if the boundary value is a part of the allowed range and false otherwise.

void setBoundary(Value boundary, TBoolean inclusive)  
Sets the boundary value. Dependent on the type of the value range, the boundary parameter can contain either an IntegerValue, FloatValue, UniversalCharstringValue or CharstringValue. The inclusive parameter determines whether the boundary value is a part of the range (true) or not (false).

TBoolean isInfinity() Returns true if the boundary is equal to infinity or -infinity and false otherwise. The infinity setting is context-dependent. If the RangeBoundary instance is used as a lower boundary, it will be interpreted as -infinity and if it is used as an upper boundary, it will be interpreted as infinity.

void setToInfinity() Sets the boundary to infinity.

### 7.2.3 Abstract logging types

#### 7.2.3.1 The abstract data type TciValueTemplate

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

TBoolean isOmit() Returns true if the template is an omit template.

TBoolean isAny() Returns true if the template is an any template.

TBoolean isAnyOrOmit() Returns true if the template is an any or omit template.

TString getTemplateDef() Returns the definition of that template.

#### 7.2.3.2 The abstract data type TciNonValueTemplate

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

TBoolean isAny() Returns true if the template is an any template.

TBoolean isAll() Returns true if the template is an all template.

TString getTemplateDef() Returns the definition of that template.

#### 7.2.3.3 The Value List and Mismatch Types

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

TciValueList A value of TciValueList is a list of values.

TciValueDifference A value of TciValueDifference is a structure containing a value, a template, and a description for the reason of this difference.

TciValueDifferenceList A value of TciValueDifferenceList is a sequence of value differences.

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

TInteger size() Returns the number of values in this list.

TBoolean isEmpty() Returns true if this list contains no values.

Value get(in TInteger index) Returns the value at the specified position.

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

Value getValue() Returns the value of the TciValueDifference.

TciValueTemplate getTciValueTemplate()  
Returns the template of the TciValueDifference.

String getDescription() Returns the description of the mismatch.

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

TInteger size() Returns the number of values in the list.

TBoolean isEmpty() Returns true if the list contains no values.

TciValueDifference get(in TInteger index)  
Returns the TciValueDifference at the specified position.

#### 7.2.3.4 The Status Types

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

ComponentStatusType A value of ComponentStatusType is either "inactiveC", "runningC", "stoppedC", "killedC", or "nullC".

TimerStatusType A value of TimerStatusType is either "runningT", "inactiveT", "expiredT", or "nullC".

PortStatusType A value of PortStatusType is either "startedP", "haltedP", or   
"stoppedP".

## 7.3 TCI operations

### 7.3.0 The TCI interfaces

The clause 7.3 specifies the operations that a TTCN‑3 Executable shall provide to a test system (*required operations*) and which functionality shall be provided by the test system to the TTCN‑3 Executable (*provided operations*).

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

All operation definitions in this clause are defined using the Interface Definition Language (IDL). Concrete language mappings are defined in clauses 8, 9, 10, 11 and 12. Annex B provides for the logging interface an alternative mapping to XML.

For every TCI operation call all *in*, *inout*, and *out* parameters listed in the particular operation definition are mandatory. The value of an *in* parameter is specified by the calling entity. Calling entity refers to the direction of the call. For operations on a *required* interface the calling entity is the test system while the called entity is the TTCN‑3 Executable. For operations on a *provided* interface the calling entity is the TTCN‑3 Executable while the test system is the called entity.

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

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

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

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

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

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

### 7.3.1 The TCI‑TM interface

#### 7.3.1.0 Scope of use

The TCI Test Management Interface (TCI‑TM) describes the operations a TTCN‑3 Executable is required to implement and the operations a test management implementation shall provide to the TE (figure 5).



Figure 5: The TCI‑TM interface

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

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

#### 7.3.1.1 TCI‑TM required

##### 7.3.1.1.0 Scope of use

Clause 7.3.1.1 specifies the operations the TM requires from the TE. In addition to the operations specified in this clause, a test management requires the operations as required at the TCI‑CD interface.

##### 7.3.1.1.1 tciRootModule

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

##### 7.3.1.1.2 tciGetImportedModules

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

##### 7.3.1.1.3 tciGetModuleParameters

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

##### 7.3.1.1.4 tciGetTestCases

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

##### 7.3.1.1.5 tciGetTestCaseParameters

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

##### 7.3.1.1.6 tciGetTestCaseTSI

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

##### 7.3.1.1.7 tciStartTestCase

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

##### 7.3.1.1.8 tciStopTestCase

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

##### 7.3.1.1.9 tciStartControl

|  |  |
| --- | --- |
| Signature | TriComponentId tciStartControl() |
| Return Value | A TriComponentId that represents the test component the module control part is executed on. If the TE cannot start control part of the selected module the distinct value null will be returned. |
| Constraint | Shall be called only if a module has been selected before. |
| Effect | Starts the module control function of the selected module. The module control function started this way shall not contain any parameters and it shall not return a value. The module control function will start TTCN‑3 test cases as described in TTCN‑3. While executing the control function the TE will call the *provided* operation tciTestCaseStarted and tciTestCaseTerminated for every test case that has been started and that has terminated. After termination of the control part the TE will call the *provided* operation tciControlPartTerminated. |

##### 7.3.1.1.10 tciStopControl

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

##### 7.3.1.1.11 tciGetControlParameters

|  |  |
| --- | --- |
| Signature | TciParameterTypeListType tciGetControlParameters () |
| Return Value | A list of all parameter types of the module control function of the root module. The parameter types are ordered as they appear in the TTCN‑3 signature of the module control function. |
| Constraint | Shall be used only if a root module has been set before. |
| Effect | The TE provides to the management a list of parameter types of the module control function of the root module. If no module control function parameters exist, an empty parameter type list is returned. If the TE cannot provide a list, the distinct null value shall be returned. |

##### 7.3.1.1.12 tciStartControlWithParameters

|  |  |  |
| --- | --- | --- |
| Signature | void tciStartControlWithParameters(in TciParameterListType parameterList) | |
| In Parameters | parameterList | A list of Values where each value defines a parameter from the parameter list as defined in the TTCN‑3 module control function definition. The parameters in parameterList are ordered as they appear in the TTCN‑3 signature of the module control function. If no parameters have to be passed either the null value or an empty parameterList, i.e. a list of length zero shall be passed. |
| Return Value | void | |
| Constraint | Shall be called only if a module has been selected before. | |
| Effect | tciStartControlWithParameters starts the module control function of the currently selected module with the given parameters. A tciError will be issued by the TE if the module control function does not exist.  All *in* and *inout* parameters in parameterList shall contain a Value. All *out* parameters in parameterList shall contain the distinct value of null since they are only of relevance when the module control function terminates. | |

#### 7.3.1.2 TCI‑TM provided

##### 7.3.1.2.0 Scope of use

Clause 7.3.1.2 specifies the operations the TM has to provide to the TE.

##### 7.3.1.2.1 tciTestCaseStarted

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

##### 7.3.1.2.2 tciTestCaseTerminated

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

##### 7.3.1.2.3 tciControlTerminated

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

##### 7.3.1.2.4 tciGetModulePar

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

##### 7.3.1.2.5 tciLog

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

##### 7.3.1.2.6 tciError

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

##### 7.3.1.2.7 tciControlTerminatedWithResult

|  |  |  |
| --- | --- | --- |
| Signature | void tciControlTerminatedWithResult (in Value result,  in TciParameterListType parameterList) | |
| In Parameters | result | The return value of the module control function or the distinct value null if the function does not return any value. |
| parameterList | A list of values that are part of the module control function signature. The parameters in parameterList are ordered as they appear in the TTCN‑3 module control function declaration. |
| Return Value | Void | |
| Constraint | Shall only be called when the module execution has been started using the tciStartControlWithParameters operation. | |
| Effect | This operation will be called by the TE to indicate the test management that the control part of the selected module has just terminated execution. On the invocation of a tciControlTerminatedWithResult operation all *out* and *inout* parameters contain Values. All *in* parameters contain the distinct value of null. | |

### 7.3.2 The TCI‑CD interface

#### 7.3.2.0 Scope of use

The TCI Codec Interface (TCI‑CD) describes the operations a TTCN‑3 Executable is required to implement and the operations a codec implementation for a certain encoding scheme shall provide to the TE (see figure 6).



Figure 6: The TCI‑CD interface

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

Annex C illustrates the usage and sequential ordering of operation calls by either the TE or the CD.

#### 7.3.2.1 TCI‑CD required

##### 7.3.2.1.0 Scope of use

The clause 7.3.2.1 specifies the operations the CD requires from the TE. All operations specified in this clause are also required at the TCI‑TM and TCI‑CH interfaces.

##### 7.3.2.1.1 getTypeForName

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

##### 7.3.2.1.2 getInteger

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

##### 7.3.2.1.3 getFloat

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

##### 7.3.2.1.4 getBoolean

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

##### 7.3.2.1.5 Void

##### 7.3.2.1.6 getCharstring

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

##### 7.3.2.1.7 getUniversalCharstring

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

##### 7.3.2.1.8 getHexstring

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

##### 7.3.2.1.9 getBitstring

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

##### 7.3.2.1.10 getOctetstring

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

##### 7.3.2.1.11 getVerdict

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

##### 7.3.2.1.12 tciErrorReq

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

#### 7.3.2.2 TCI‑CD provided

##### 7.3.2.2.0 Scope of use

Clause 7.3.2.2 specifies the operations the TM shall provide to the TE.

##### 7.3.2.2.1 decode

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

##### 7.3.2.2.2 encode

|  |  |  |
| --- | --- | --- |
| Signature | TriMessageType encode(in Value value) | |
| In Parameters | value | The value to be encoded. |
| Return Value | Returns an encoded TriMessage for the specified encoding rule. | |
| Constraint | This operation shall be called whenever the TE has to implicitly encode a value (e.g. when performing a port operation such as send, call, reply and raise or calling an external function). | |
| Effect | Returns an encoded TriMessage according to the encoding rules. | |

##### 7.3.2.2.3 decodeValue

|  |  |  |
| --- | --- | --- |
| Signature | TInteger decodeValue(inout TriMessageType encodedValue,  in Type decodingHypothesis,  in TString encodingInfo,  out Value decodedValue) | |
| In Parameters | message | The encoded message to be decoded. |
| decodingHypothesis | The hypothesis the decoding can be based on. |
| decodingInfo | Dynamic decoding parameters. |
| Out Parameters | decodedValue | The decoded value, if the value is of a compatible type as the decodingHypothesis, else the distinct value null. |
| Return Value | An integer value indicating success of the operation: 0 in case of success, 1 in case of an unspecified decoding error and 2 if decoding could not be completed because encodedValue did not contain enough bits. | |
| Constraint | This operation shall be called whenever the TE invokes the predefined functions decvalue or decvalue\_unichar. | |
| Effect | This operation decodes message according to the encoding rules and returns the result of the decoding operation. The decodingHypothesis shall be used to determine whether the encoded value can be decoded. If an encoding rule is not self‑sufficient, i.e. if the encoded message does not inherently contain its type decodingHypothesis shall be used. In case of success, the used bits are removed from the encodedValue parameter and the decoded TTCN‑3 value is passed to the caller in the decodedValue parameter. In case of a failure, the TE shall ignore the content of the encodedValue and decodedValue parameter and shall act as if the former one were unchanged and the latter one contained the distinct value null. | |

##### 7.3.2.2.4 encodeValue

|  |  |  |
| --- | --- | --- |
| Signature | TriMessageType encode(in Value value, in TString encodingInfo) | |
| In Parameters | value | The value to be encoded. |
| encodingInfo | Dynamic encoding parameters. |
| Return Value | Returns an encoded TriMessage for the specified encoding rule. | |
| Constraint | This operation shall be called whenever the TE invokes the predefined functions encvalue or encvalue\_unichar. | |
| Effect | Returns an encoded TriMessage according to the encoding rules. | |

### 7.3.3 The TCI‑CH interface

#### 7.3.3.0 Scope of use

The TCI Component Handling Interface (TCI‑CH) describes the operations a TTCN‑3 Executable is required to implement and the operations a component handling implementation shall provide to the TE (figure 7).



Figure 7: The TCI‑CH interface

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

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

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

Annex C illustrates the usage and sequential ordering of operation calls by either the TE or the CH.

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

#### 7.3.3.1 TCI‑CH required

##### 7.3.3.1.0 Scope of use

Clause 7.3.3.1 specifies the operations the CH requires from the TE. In addition to the operations specified in this clause, all *required* operations of the TCI‑CD interface are also required.

##### 7.3.3.1.1 tciEnqueueMsgConnected

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

##### 7.3.3.1.2 tciEnqueueCallConnected

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

##### 7.3.3.1.3 tciEnqueueReplyConnected

|  |  |  |
| --- | --- | --- |
| Signature | void tciEnqueueReplyConnected (in TriPortIdType sender,  in TriComponentIdType receiver,  in TriSignatureIdType signature,  in TciParameterListType parameterList,  in Value returnValue) | |
| In Parameters | sender | Identifier of the port sending the reply. |
| receiver | Identifier of the component receiving the reply. |
| signature | Identifier of the signature of the procedure call. |
| parameterList | A list of value parameters which are part of the indicated signature. The parameters in parameterList are ordered as they appear in the TTCN‑3 signature declaration. |
| returnValue | (Optional) return value of the procedure call. |
| Return Value | void | |
| Constraint | This operation shall be called by the CH at the local TE when at a remote TE a *provided* tciReplyConnected has been called. All *out* and *inout* procedure parameters and the return value contain values. All *in* procedure parameters shall contain the distinct value of nullsince they are only of relevance to the procedure call but not in the reply to the call. The parameterListcontains 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 nullshall be passed for the returnValue. | |
| Effect | The TE enqueues the reply at the local port queue of the indicated receiver component. | |

##### 7.3.3.1.4 tciEnqueueRaiseConnected

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

##### 7.3.3.1.5 tciCreateTestComponent

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

##### 7.3.3.1.6 tciStartTestComponent

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

##### 7.3.3.1.7 tciStopTestComponent

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

##### 7.3.3.1.8 tciConnect

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

##### 7.3.3.1.9 tciDisconnect

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

##### 7.3.3.1.10 tciMap

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

##### 7.3.3.1.11 tciMapParam

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

##### 7.3.3.1.12 tciUnmap

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

##### 7.3.3.1.13 tciUnmapParam

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

##### 7.3.3.1.14 tciTestComponentTerminated

|  |  |  |
| --- | --- | --- |
| Signature | void tciTestComponentTerminated (in TriComponentIdType component,  in VerdictValue verdict) | |
| In Parameters | component | Identifier of the component that has terminated. |
| verdict | Verdict after termination of the component. |
| Return Value | void | |
| Constraint | This operation shall be called by the CH at the local TE when at a remote TE a *provided* tciTestComponentTerminatedReq has been called. | |
| Effect | The local TE is notified of the termination of the indicated test component on a remote TE. Because the out values of *inout* and *out* parameters of a function being executed on a test component have no effect on that test component (ETSI ES 201 873‑1 [1]), the tciTestComponentTerminated operation does not have a parameterList parameter. | |

##### 7.3.3.1.15 tciTestComponentRunning

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

##### 7.3.3.1.16 tciTestComponentDone

|  |  |  |
| --- | --- | --- |
| Signature | TBoolean tciTestComponentDone (in TriComponentIdType comp, out TInteger verdict) | |
| In Parameters | comp | Identifier of the component to be checked for done. |
| Out Parameters | verdict | If the component has completed executing its behaviour, the parameter will contain numeric representation of the final component verdict (one of the following constants: ERROR, FAIL, INCONC, NONE, PASS, USER\_ERROR). Otherwise, the parameter will contain the ERROR constant. |
| Return Value | true if the indicated component has completed executing its behaviour, false otherwise. | |
| Constraint | This operation shall be called by the CH at the local TE when at a remote TE a *provided* tciTestComponentDoneReq has been called. | |
| Effect | The local TE determines whether the indicated component has completed executing its test behaviour. If the component has completed its behaviour true will be returned. In any other case, e.g. test component has not been started, or test component is still executing, false will be returned. After the operation returns, the CH will communicate the value back to the remote TE. | |

##### 7.3.3.1.17 tciGetMTC

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

##### 7.3.3.1.18 tciExecuteTestCase

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

##### 7.3.3.1.19 tciReset

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

##### 7.3.3.1.20 tciKillTestComponent

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

##### 7.3.3.1.21 tciTestComponentAlive

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

##### 7.3.3.1.22 tciTestComponentKilled

|  |  |  |
| --- | --- | --- |
| Signature | TBoolean tciTestComponentKilled (in TriComponentIdType comp, out TInteger verdict) | |
| In Parameters | comp | Identifier of the component to be checked for being killed. |
| Out Parameters | verdict | If the component has been killed, the parameter will contain numeric representation of the final component verdict (one of the following constants: ERROR, FAIL, INCONC, NONE, PASS, USER\_ERROR). Otherwise, the parameter will contain the ERROR constant. |
| Return Value | true if the indicated component has been killed, false otherwise. | |
| Constraint | This operation shall be called by the CH at the local TE when at a remote TE a *provided* tciTestComponentKilledReq has been called. | |
| Effect | The local TE determines whether the indicated component is in the killed state. If it is, true will be returned. In any other case, false will be returned. After the operation returns, the CH will communicate the value back to the remote TE. | |

##### 7.3.3.1.23 tciCallTestComponent

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

##### 7.3.3.1.24 tciTestComponentCallTerminated

|  |  |  |
| --- | --- | --- |
| Signature | void tciTestComponentCallTerminated (in TriComponentIdType component,  in VerdictValue verdict,  in TciParameterListType parameterList,  in Value returnValue) | |
| In Parameters | component | Identifier of the called component that has finished execution. |
| verdict | Verdict after finishing component call. |
| parameterList | A list of Values where each value defines a parameter from the parameter list as defined in the TTCN‑3 function declaration of the function being started. The parameters in parameterList are ordered as they appear in the TTCN‑3 definition of the called. If no parameters have to be passed either the null value or an empty parameterList, i.e. a list of length zero shall be passed. |
| returnValue | The return value of the called function or the null value if no return value is defined. |
| Return Value | Void | |
| Constraint | This operation shall be called by the CH at the local TE when at a remote TE a *provided* tciTestComponentCallTerminatedReq has been called. | |
| Effect | The local TE is notified that the call to the indicated test component on a remote TE has been finished and receives the result of the call. | |

#### 7.3.3.2 TCI‑CH provided

##### 7.3.3.2.0 Scope of use

The clause 7.3.3.2 specifies the operations the CH shall provide to the TE.

##### 7.3.3.2.1 tciSendConnected

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

##### 7.3.3.2.2 tciSendConnectedBC

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

##### 7.3.3.2.3 tciSendConnectedMC

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

##### 7.3.3.2.4 tciCallConnected

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

##### 7.3.3.2.5 tciCallConnectedBC

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

##### 7.3.3.2.6 tciCallConnectedMC

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

##### 7.3.3.2.7 tciReplyConnected

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

##### 7.3.3.2.8 tciReplyConnectedBC

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

##### 7.3.3.2.9 tciReplyConnectedMC

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

##### 7.3.3.2.10 tciRaiseConnected

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

##### 7.3.3.2.11 tciRaiseConnectedBC

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

##### 7.3.3.2.12 tciRaiseConnectedMC

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

##### 7.3.3.2.13 tciCreateTestComponentReq

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

##### 7.3.3.2.14 tciStartTestComponentReq

|  |  |  |
| --- | --- | --- |
| Signature | void tciStartTestComponentReq(in TriComponentIdType component,  in TciBehaviourIdType behaviour,  in TciParameterListType parameterList) | |
| In Parameters | component | Identifier of the component to be started. |
| behaviour | Identifier of the behaviour to be started on the component. |
| parameterList | A list of Values where each value defines a parameter from the parameter list as defined in the TTCN‑3 function declaration of the function being started. The parameters in parameterList are ordered as they appear in the TTCN‑3 signature of the test case. If no parameters have to be passed either the null value or an empty parameterList, i.e. a list of length zero shall be passed. |
| Return Value | void | |
| Constraint | This operation shall be called by the TE when it executes the TTCN‑3 start operation. | |
| Effect | CH transmits the start component request to the remote TE and calls there the tciStartTestComponent operation. | |

##### 7.3.3.2.15 tciStopTestComponentReq

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

##### 7.3.3.2.16 tciConnectReq

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

##### 7.3.3.2.17 tciDisconnectReq

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

##### 7.3.3.2.18 tciMapReq

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

##### 7.3.3.2.19 tciMapParamReq

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

##### 7.3.3.2.20 tciUnmapReq

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

##### 7.3.3.2.21 tciUnmapParamReq

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

##### 7.3.3.2.22 tciTestComponentTerminatedReq

|  |  |  |
| --- | --- | --- |
| Signature | void tciTestComponentTerminatedReq (in TriComponentIdType component,  in VerdictValue verdict) | |
| In Parameters | component | Identifier of the component that has terminated. |
| verdict | Verdict after termination of the component. |
| Return Value | void | |
| Constraint | This operation shall be called by the TE when a test component terminates execution, either explicitly with the TTCN‑3 stop operation or implicitly, if it has reached the last statement. | |
| Effect | The CH is notified of the termination of the indicated test component. Because the out values of *inout* and *out* parameters of a function being executed on a test component have no effect on that test component (ETSI ES 201 873‑1 [1]), the tciTestComponentTerminateReq operation does not have a parameterList parameter. CH communicates the termination of the indicated component to all participating TEs and to the special TE\*, which keeps track of the overall verdict. | |

##### 7.3.3.2.23 tciTestComponentRunningReq

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

##### 7.3.3.2.24 tciTestComponentDoneReq

|  |  |  |
| --- | --- | --- |
| Signature | TBoolean tciTestComponentDoneReq (in TriComponentIdType comp, out TInteger verdict) | |
| In Parameters | comp | Identifier of the component to be checked for done. |
| Out Parameters | verdict | If the component has completed executing its behaviour, the parameter will contain numeric representation of the final component verdict (one of the following constants: ERROR, FAIL, INCONC, NONE, PASS, USER\_ERROR). Otherwise, the parameter will contain the ERROR constant. |
| Return Value | true if the indicated component has completed executing its behaviour, false otherwise. | |
| Constraint | This operation shall be called by the TE when it executes a TTCN‑3 done operation. | |
| Effect | CH transmits the done request to the remote TE having the test component to be checked, where it calls the tciTestComponentDone operation to check the status of the indicated test component. | |

##### 7.3.3.2.25 tciGetMTCReq

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

##### 7.3.3.2.26 tciExecuteTestCaseReq

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

##### 7.3.3.2.27 tciResetReq

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

##### 7.3.3.2.28 tciKillTestComponentReq

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

##### 7.3.3.2.29 tciTestComponentAliveReq

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

##### 7.3.3.2.30 tciTestComponentKilledReq

|  |  |  |
| --- | --- | --- |
| Signature | TBoolean tciTestComponentKilledReq (in TriComponentIdType comp, out TInteger verdict) | |
| In Parameters | comp | Identifier of the component to be checked for being killed. |
| Out Parameters | verdict | If the component has been killed, the parameter will contain numeric representation of the final component verdict (one of the following constants: ERROR, FAIL, INCONC, NONE, PASS, USER\_ERROR). Otherwise, the parameter will contain the ERROR constant. |
| Return Value | true if the indicated component has been killed, false otherwise. | |
| Constraint | This operation shall be called by the TE when it executes the TTCN‑3 killed operation. | |
| Effect | CH transmits the request to the remote TE that created the test component in question, where it calls the tciTestComponentKilled operation to check the status of the indicated test component. | |

##### 7.3.3.2.31 tciCallTestComponentReq

|  |  |  |
| --- | --- | --- |
| Signature | void tciCallTestComponentReq (in TriComponentIdType component,  in TciBehaviourIdType behaviour,  in TciParameterListType parameterList) | |
| In Parameters | component | Identifier of the component to be called. |
| behaviour | Identifier of the behaviour to be called on the component. |
| parameterList | A list of Values where each value defines a parameter from the parameter list as defined in the TTCN‑3 function declaration of the function being started. The parameters in parameterList are ordered as they appear in the TTCN‑3 definition of the called function. If no parameters have to be passed either the null value or an empty parameterList, i.e. a list of length zero shall be passed. |
| Return Value | void | |
| Constraint | This operation shall be called by the TE when it executes the TTCN‑3 call component operation. | |
| Effect | CH transmits the call component request to the remote TE and calls there the tciCallTestComponent operation. | |

##### 7.3.3.2.32 tciTestComponentCallTerminatedReq

|  |  |  |
| --- | --- | --- |
| Signature | void tciTestComponentCallTerminatedReq (in TriComponentIdType component,  in VerdictValue verdict,  in TciParameterListType parameterList,  in Value returnValue) | |
| In Parameters | component | Identifier of the called component that has finished execution. |
| verdict | Verdict after finishing component call. |
| parameterList | A list of Values where each value defines a parameter from the parameter list as defined in the TTCN‑3 function declaration of the function being started. The parameters in parameterList are ordered as they appear in the TTCN‑3 definition of the called. If no parameters have to be passed either the null value or an empty parameterList, i.e. a list of length zero shall be passed. |
| returnValue | The return value of the called function or the null value if no return value is defined. |
| Return Value | void | |
| Constraint | This operation shall be called by the TE when a test component terminates execution of a function called on the component implicitly, if it has reached the return statement or executed the last statement of the called function. | |
| Effect | The CH is notified of the termination of the called test component. CH communicates the termination of the indicated component to the component that initiated the call operation and notifies all other participating TEs and the special TE\* which keeps track of the overall verdict that the component has been terminated. This notification is done by using the tciTestComponentTerminated function as all other participating TEs and the special TE do not need the return value and values of the out parameters. | |

### 7.3.4 The TCI‑TL interface

#### 7.3.4.0 Scope of use

The TCI Test Logging Interface (TCI‑TL) describes the operations a TTCN‑3 Executable is required to implement and the operations a test logging implementation shall provide to the TE (figure 8).



Figure 8: The TCI‑TL interface

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

#### 7.3.4.1 TCI‑TL provided

##### 7.3.4.1.0 Scope of use

Clause 7.3.4.1.0 specifies the operations the TL shall provide to the TE.

NOTE: A logging event is timestamped. The timestamps are specific to the language mapping and to the tool used. For example, C# mapping uses System. DataTime objects and C++ mapping uses the timeval structure. Java™, C and XML use integer instead, so that a tool has to choose its timestamp handling. For example, for Java™ System.currentTimeMillis() could be used or for C time() could be used.

##### 7.3.4.1.1 tliTcExecute

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

##### 7.3.4.1.2 tliTcStart

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

##### 7.3.4.1.3 tliTcStop

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

##### 7.3.4.1.4 tliTcStarted

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

##### 7.3.4.1.5 tliTcTerminated

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

##### 7.3.4.1.6 tliCtrlStart

|  |  |  |
| --- | --- | --- |
| Signature | void tliCtrlStart(in TString am, in TInteger ts,  in TString src, in TInteger line,  in TriComponentIdType c) | |
| In Parameters | am | An additional message. |
| ts | The time when the event is produced. |
| src | The source file of the test specification. |
| line | The line number where the request is performed. |
| c | The component which produces this event. |
| Return Value | void | |
| Constraint | Shall be called by TE to log the start of the module control function in cases when the module control function does not define any parameters and does not return a value. This event occurs before the control is started. | |
| Effect | The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document. | |

##### 7.3.4.1.7 tliCtrlStop

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

##### 7.3.4.1.8 tliCtrlTerminated

|  |  |  |
| --- | --- | --- |
| Signature | void tliCtrlTerminated (in TString am, in TInteger ts,  in TString src, in TInteger line,  in TriComponentIdType c) | |
| In Parameters | am | An additional message. |
| ts | The time when the event is produced. |
| src | The source file of the test specification. |
| line | The line number where the request is performed. |
| c | The component which produces this event. |
| Return Value | void | |
| Constraint | Shall be called by TM or TE to log the termination of the module control function in cases when the module control function does not define any parameters and does not return a value. This event occurs after the control has terminated. | |
| Effect | The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document. | |

##### 7.3.4.1.9 tliMSend\_m

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

##### 7.3.4.1.10 tliMSend\_m\_BC

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

##### 7.3.4.1.11 tliMSend\_m\_MC

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

##### 7.3.4.1.12 tliMSend\_c

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

##### 7.3.4.1.13 tliMSend\_c\_BC

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

##### 7.3.4.1.14 tliMSend\_c\_MC

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

##### 7.3.4.1.15 tliMDetected\_m

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

##### 7.3.4.1.16 tliMDetected\_c

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

##### 7.3.4.1.17 tliMMismatch\_m

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

##### 7.3.4.1.18 tliMMismatch\_c

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

##### 7.3.4.1.19 tliMReceive\_m

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

##### 7.3.4.1.20 tliMReceive\_c

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

##### 7.3.4.1.21 tliPrCall\_m

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

##### 7.3.4.1.22 tliPrCall\_m\_BC

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

##### 7.3.4.1.23 tliPrCall\_m\_MC

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

##### 7.3.4.1.24 tliPrCall\_c

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

##### 7.3.4.1.25 tliPrCall\_c\_BC

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

##### 7.3.4.1.26 tliPrCall\_c\_MC

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

##### 7.3.4.1.27 tliPrGetCallDetected\_m

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

##### 7.3.4.1.28 tliPrGetCallDetected\_c

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

##### 7.3.4.1.29 tliPrGetCallMismatch\_m

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

##### 7.3.4.1.30 tliPrGetCallMismatch\_c

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

##### 7.3.4.1.31 tliPrGetCall\_m

|  |  |  |
| --- | --- | --- |
| Signature | void tliPrGetCall\_m(in TString am, in TInteger ts, in TString src,  in TInteger line, in TriComponentIdType c,  in TriPortIdType at, in TriSignatureIdType signature,  in TciParameterListType tciPars,  in TciValueTemplate parsTmpl,  in Value addrValue,  in TciValueTemplate addressTmpl) | |
| In Parameters | am | An additional message. |
| ts | The time when the event is produced. |
| src | The source file of the test specification. |
| line | The line number where the request is performed. |
| c | The component which produces this event. |
| at | The port via which the call is received. |
| signature | The signature of the detected call. |
| tciPars | The parameters of detected call. |
| parsTmpl | The template used to check the parameter match. |
| addrValue | The address value of the source within the SUT. |
| addressTmpl | The expected address of the source within the SUT. |
| Return Value | void | |
| Constraint | Shall be called by TE to log getting a call. This event occurs after getcall has matched against a template. This event is used for logging the communication with the SUT. | |
| Effect | The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document. | |

##### 7.3.4.1.32 tliPrGetCall\_c

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

##### 7.3.4.1.33 tliPrReply\_m

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

##### 7.3.4.1.34 tliPrReply\_m\_BC

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

##### 7.3.4.1.35 tliPrReply\_m\_MC

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

##### 7.3.4.1.36 tliPrReply\_c

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

##### 7.3.4.1.37 tliPrReply\_c\_BC

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

##### 7.3.4.1.38 tliPrReply\_c\_MC

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

##### 7.3.4.1.39 tliPrGetReplyDetected\_m

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

##### 7.3.4.1.40 tliPrGetReplyDetected\_c

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

##### 7.3.4.1.41 tliPrGetReplyMismatch\_m

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

##### 7.3.4.1.42 tliPrGetReplyMismatch\_c

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

##### 7.3.4.1.43 tliPrGetReply\_m

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

##### 7.3.4.1.44 tliPrGetReply\_c

|  |  |  |
| --- | --- | --- |
| Signature | void tliPrGetReply\_c(in TString am, in TInteger ts, in TString src,  in TInteger line, in TriComponentIdType c,  in TriPortIdType at, in TriSignatureIdType signature,  in TciParameterListType tciPars,  in TciValueTemplate parsTmpl,  in Value replValue, in TciValueTemplate replyTmpl,  in TriComponentIdType from,  in TciNonValueTemplate fromTmpl) | |
| In Parameters | am | An additional message. |
| ts | The time when the event is produced. |
| src | The source file of the test specification. |
| line | The line number where the request is performed. |
| c | The component which produces this event. |
| at | The port via which the reply is received. |
| signature | The signature relating to the reply. |
| tciPars | The signature parameters relating to the reply. |
| parsTmpl | The template used to check the parameter match. |
| replValue | The received reply. |
| replyTmpl | The template used to check the reply match. |
| from | The component which sent the reply. |
| fromTmpl | The expected replying component. |
| Return Value | void | |
| Constraint | Shall be called by TE to log getting a reply. This event occurs after getreply is checked against a template. This event is used for logging the intercomponent communication. | |
| Effect | The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document. | |

##### 7.3.4.1.45 tliPrRaise\_m

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

##### 7.3.4.1.46 tliPrRaise\_m\_BC

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

##### 7.3.4.1.47 tliPrRaise\_m\_MC

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

##### 7.3.4.1.48 tliPrRaise\_c

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

##### 7.3.4.1.49 tliPrRaise\_c\_BC

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

##### 7.3.4.1.50 tliPrRaise\_c\_MC

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

##### 7.3.4.1.51 tliPrCatchDetected\_m

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

##### 7.3.4.1.52 tliPrCatchDetected\_c

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

##### 7.3.4.1.53 tliPrCatchMismatch\_m

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

##### 7.3.4.1.54 tliPrCatchMismatch\_c

|  |  |  |
| --- | --- | --- |
| Signature | void tliPrCatchMismatch\_c(in TString am, in TInteger ts,  in TString src,  in TInteger line, in TriComponentIdType c,  in TriPortIdType at, in TriSignatureIdType signature,  in Value excValue, in TciValueTemplate excTmpl,  in TciValueDifferenceList diffs,  in TriComponentIdType from,  in TciNonValueTemplate fromTmpl) | |
| In Parameters | Am | An additional message. |
| Ts | The time when the event is produced. |
| Src | The source file of the test specification. |
| Line | The line number where the request is performed. |
| C | The component which produces this event. |
| At | The port via which the exception is received. |
| Signature | The signature relating to the exception. |
| excValue | The received exception. |
| excTmpl | The template used to check the exception match. |
| Diffs | The difference/the mismatch between exception and template. |
| From | The component which sent the reply. |
| fromTmpl | The expected replying component. |
| Return Value | Void | |
| Constraint | Shall be called by TE to log the mismatch of a catch operation. This event occurs after catch is checked against a template. This event is used for logging the intercomponent communication. | |
| Effect | The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document. | |

##### 7.3.4.1.55 tliPrCatch\_m

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

##### 7.3.4.1.56 tliPrCatch\_c

|  |  |  |
| --- | --- | --- |
| Signature | void tliPrCatch\_c(in TString am, in TInteger ts, in TString src,  in TInteger line, in TriComponentIdType c,  in TriPortIdType at, in TriSignatureIdType signature,  in Value excValue, in TciValueTemplate excTmpl,  in TriComponentIdType from,  in TciNonValueTemplate fromTmpl) | |
| In Parameters | Am | An additional message. |
| Ts | The time when the event is produced. |
| Src | The source file of the test specification. |
| Line | The line number where the request is performed. |
| C | The component which produces this event. |
| At | The port via which the exception is received. |
| Signature | The signature relating to the exception. |
| excValue | The received exception. |
| excTmpl | The template used to check the exception match. |
| From | The component which sent the reply. |
| fromTmpl | The expected replying component. |
| Return Value | Void | |
| Constraint | Shall be called by CH or TE to log catching an exception. This event occurs after catch is checked against a template. This event is used for logging the intercomponent communication. | |
| Effect | The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document. | |

##### 7.3.4.1.57 tliPrCatchTimeoutDetected

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

##### 7.3.4.1.58 tliPrCatchTimeout

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

##### 7.3.4.1.59 tliCCreate

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

##### 7.3.4.1.60 tliCStart

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

##### 7.3.4.1.61 tliCRunning

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

##### 7.3.4.1.62 tliCAlive

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

##### 7.3.4.1.63 tliCStop

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

##### 7.3.4.1.64 tliCKill

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

##### 7.3.4.1.65 tliCDoneMismatch

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

##### 7.3.4.1.66 tliCDone

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

##### 7.3.4.1.67 tliCKilledMismatch

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

##### 7.3.4.1.68 tliCKilled

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

##### 7.3.4.1.69 tliCTerminated

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

##### 7.3.4.1.70 tliPConnect

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

##### 7.3.4.1.71 tliPDisconnect

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

##### 7.3.4.1.72 tliPMap

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

##### 7.3.4.1.73 tliPMapParam

|  |  |  |
| --- | --- | --- |
| Signature | void tliPMapParam(in TString am, in TInteger ts, in TString src,  in TInteger line, in TriComponentIdType c,  in TriPortIdType port1, in TriPortIdType port2,  in TciParameterListType tciPars,  in TciStatusType encoderFailure,  in TriParameterListType triPars) | |
| In Parameters | am | An additional message. |
| ts | The time when the event is produced. |
| src | The source file of the test specification. |
| line | The line number where the request is performed. |
| c | The component which produces this event. |
| port1 | The first port to be mapped. |
| port2 | The second port to be mapped. |
| tciPars | The configuration parameter list. |
| encoderFailure | The failure message which might occur at encoding. |
| triPars | The encoded configuration parameter list. |
| Return Value | void | |
| Constraint | Shall be called by SA or TE to log the map operation. This event occurs after the map operation including parameters. | |
| Effect | The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document. | |

##### 7.3.4.1.74 tliPUnmap

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

##### 7.3.4.1.75 tliPUnmapParam

|  |  |  |
| --- | --- | --- |
| Signature | void tliPUnmapParam(in TString am, in TInteger ts, in TString src,  in TInteger line, in TriComponentIdType c,  in TriPortIdType port1, in TriPortIdType port2,  in TciParameterListType tciPars,  in TciStatusType encoderFailure,  in TriParameterListType triPars) | |
| In Parameters | am | An additional message. |
| ts | The time when the event is produced. |
| src | The source file of the test specification. |
| line | The line number where the request is performed. |
| c | The component which produces this event. |
| port1 | The first port to be unmapped. |
| port2 | The second port to be unmapped. |
| tciPars | The configuration parameter list. |
| encoderFailure | The failure message which might occur at encoding. |
| triPars | The encoded configuration parameter list. |
| Return Value | void | |
| Constraint | Shall be called by SA or TE to log the unmap operation. This event occurs after the unmap operation including parameters. | |
| Effect | The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document. | |

##### 7.3.4.1.76 tliPClear

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

##### 7.3.4.1.77 tliPStart

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

##### 7.3.4.1.78 tliPStop

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

##### 7.3.4.1.79 tliPHalt

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

##### 7.3.4.1.80 tliEncode

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

##### 7.3.4.1.81 tliDecode

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

##### 7.3.4.1.82 tliTTimeoutDetected

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

##### 7.3.4.1.83 tliTTimeoutMismatch

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

##### 7.3.4.1.84 tliTTimeout

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

##### 7.3.4.1.85 tliTStart

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

##### 7.3.4.1.86 tliTStop

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

##### 7.3.4.1.87 tliTRead

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

##### 7.3.4.1.88 tliTRunning

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

##### 7.3.4.1.89 tliSEnter

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

##### 7.3.4.1.90 tliSLeave

|  |  |  |
| --- | --- | --- |
| Signature | void tliSLeave(in TString am, in TInteger ts, in TString src,  in TInteger line, in TriComponentIdType c,  in QualifiedName name, in TciParameterListType tciPars,  in Value returnValue, in TString kind) | |
| In Parameters | am | An additional message. |
| ts | The time when the event is produced. |
| src | The source file of the test specification. |
| line | The line number where the request is performed. |
| c | The component which produces this event. |
| name | The name of the scope. |
| tciPars | The parameters of the scope. |
| returnValue | The return value of the scope. |
| kind | The kind of the scope. If the scope contains modifiers, they prefix the scope string. |
| Return Value | void | |
| Constraint | Shall be called by TE to log the leaving of a scope. This event occurs after the scoped has been left. | |
| Effect | The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document. | |

##### 7.3.4.1.91 tliVar

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

##### 7.3.4.1.92 tliModulePar

|  |  |  |
| --- | --- | --- |
| Signature | void tliModulePar(in TString am, in TInteger ts, in TString src,  in TInteger line, in TriComponentIdType c,  in QualifiedName name, in Value parValue) | |
| In Parameters | am | An additional message. |
| ts | The time when the event is produced. |
| src | The source file of the test specification. |
| line | The line number where the request is performed. |
| c | The component which produces this event. |
| name | The name of the module parameter. |
| parValue | The value of the module parameter. |
| Return Value | void | |
| Constraint | Shall be called by TE to log the value of a module parameter. This event occurs after the access to the value of a module parameter and only if the value of the module parameter was accessed for the first time. | |
| Effect | The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document. | |

##### 7.3.4.1.93 tliGetVerdict

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

##### 7.3.4.1.94 tliSetVerdict

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

##### 7.3.4.1.95 tliLog

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

##### 7.3.4.1.96 tliAEnter

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

##### 7.3.4.1.97 tliALeave

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

##### 7.3.4.1.98 tliANomatch

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

##### 7.3.4.1.99 tliARepeat

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

##### 7.3.4.1.100 tliADefaults

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

##### 7.3.4.1.101 tliAActivate

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

##### 7.3.4.1.102 tliADeactivate

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

##### 7.3.4.1.103 tliAWait

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

##### 7.3.4.1.104 tliAction

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

##### 7.3.4.1.105 tliMatch

|  |  |  |
| --- | --- | --- |
| Signature | void tliMatch(in TString am, in TInteger ts, in TString src, in TInteger line,  in TriComponentIdType c, in Value expr, in TciValueTemplate tmpl) | |
| In Parameters | am | An additional message. |
| ts | The time when the event is produced. |
| src | The source file of the test specification. |
| line | The line number where the request is performed. |
| c | The component which produces this event. |
| expr | The expression to be matched with tmpl. |
| tmpl | The template to be matched with expr. |
| Return Value | void | |
| Constraint | Shall be called by TE to log that the component successfully executed a match operation. | |
| Effect | The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document. | |

##### 7.3.4.1.106 tliMatchMismatch

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

##### 7.3.4.1.107 tliInfo

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

##### 7.3.4.1.108 tliMChecked\_m

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

##### 7.3.4.1.109 tliMChecked\_c

|  |  |  |
| --- | --- | --- |
| **Signature** | void tliMChecked\_c(in TString am, in TInteger ts,  in TString src, in TInteger line, in TriComponentIdType c,  in TriPortIdType at,  in Value msgValue,  in TciValueTemplate msgTmpl,  in TriComponentIdType from,  in TciNonValueTemplate fromTmpl) | |
| **In Parameters** | am | An additional message. |
| ts | The time when the event is produced. |
| src | The source file of the test specification. |
| line | The line number where the request is performed. |
| c | The component which produces this event. |
| msgValue | The received message. |
| msgTmpl | The template used to check the message match. |
| from | The component which sent the message. |
| fromTmpl | The expected sender component. |
| **Return Value** | Void | |
| **Constraint** | Shall be called by CH or TE to log the checking of a message. This event occurs after the message is checked successfully. This event is used for logging the intercomponent communication. | |
| **Effect** | The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document. | |

##### 7.3.4.1.110 tliPrGetCallChecked\_m

|  |  |  |
| --- | --- | --- |
| **Signature** | void tliPrGetCallChecked\_m(in TString am, in TInteger ts,  in TString src,  in TInteger line, in TriComponentIdType c,  in TriPortIdType at,  in TriSignatureIdType signature,  in TciParameterListType tciPars,  in TciValueTemplate parsTmpl,  in Value addrValue,  in TciValueTemplate addressTmpl) | |
| **In Parameters** | am | An additional message. |
| ts | The time when the event is produced. |
| src | The source file of the test specification. |
| line | The line number where the request is performed. |
| c | The component which produces this event. |
| at | The port via which the call is received. |
| signature | The signature of the checked call. |
| tciPars | The parameters of detected call. |
| parsTmpl | The template used to check the parameter match. |
| addrValue | The address value of the source within the SUT. |
| addressTmpl | The expected address of the source within the SUT. |
| **Return Value** | Void | |
| **Constraint** | Shall be called by SA or TE to log the getcall check operation. This event occurs after call is checked successfully. This event is used for logging the communication with the SUT. | |
| **Effect** | The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document. | |

##### 7.3.4.1.111 tliPrGetCallChecked\_c

|  |  |  |
| --- | --- | --- |
| **Signature** | void tliPrGetCallChecked\_c(in TString am, in TInteger ts,  in TString src,  in TInteger line, in TriComponentIdType c,  in TriPortIdType at,  in TriSignatureIdType signature,  in TciParameterListType tciPars,  in TciValueTemplate parsTmpl,  in TriComponentIdType from,  in TciNonValueTemplate fromTmpl) | |
| **In Parameters** | am | An additional message. |
| ts | The time when the event is produced. |
| src | The source file of the test specification. |
| line | The line number where the request is performed. |
| c | The component which produces this event. |
| at | The port via which the call is received. |
| signature | The signature of the called operation. |
| tciPars | The encoded parameters of checked call. |
| parsTmpl | The template used to check the parameter match. |
| from | The component which called the operation. |
| fromTmpl | The expected calling component. |
| **Return Value** | Void | |
| **Constraint** | Shall be called by CH or TE to log the getcall check operation. This event occurs after call is checked successfully. This event is used for logging the intercomponent communication. | |
| **Effect** | The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document. | |

##### 7.3.4.1.112 tliPrGetReplyChecked\_m

|  |  |  |
| --- | --- | --- |
| **Signature** | void tliPrGetReplyChecked\_m(in TString am, in TInteger ts,  in TString src,  in TInteger line, in TriComponentIdType c,  in TriPortIdType at,  in TriSignatureIdType signature,  in TciParameterListType tciPars,  in TciValueTemplate parsTmpl,  in Value replValue, in TciValueTemplate replyTmpl,  in Value addrValue,  in TciValueTemplate addressTmpl) | |
| **In Parameters** | am | An additional message. |
| ts | The time when the event is produced. |
| src | The source file of the test specification. |
| line | The line number where the request is performed. |
| c | The component which produces this event. |
| at | The port via which the reply is received. |
| signature | The signature relating to the reply. |
| tciPars | The signature parameters relating to the reply. |
| parsTmpl | The template used to check the parameter match. |
| replValue | The received reply. |
| replyTmpl | The template used to check the reply match. |
| addrValue | The address value of the source within the SUT. |
| addressTmpl | The expected address of the source within the SUT. |
| **Return Value** | Void | |
| **Constraint** | Shall be called by SA or TE to log the getreply check operation. This event occurs after getreply is checked successfully. This event is used for logging the communication with the SUT. | |
| **Effect** | The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document. | |

##### 7.3.4.1.113 tliPrGetReplyChecked\_c

|  |  |  |
| --- | --- | --- |
| **Signature** | void tliPrGetReplyChecked\_c(in TString am, in TInteger ts,  in TString src,  in TInteger line, in TriComponentIdType c,  in TriPortIdType at,  in TriSignatureIdType signature,  in TciParameterListType tciPars,  in TciValueTemplate parsTmpl,  in Value replValue, in TciValueTemplate replyTmpl,  in TriComponentIdType from,  in TciNonValueTemplate fromTmpl) | |
| **In Parameters** | am | An additional message. |
| ts | The time when the event is produced. |
| src | The source file of the test specification. |
| line | The line number where the request is performed. |
| c | The component which produces this event. |
| at | The port via which the reply is received. |
| signature | The signature relating to the reply. |
| tciPars | The encoded parameters of checked reply. |
| parsTmpl | The template used to check the parameter match. |
| replValue | The received reply. |
| replyTmpl | The template used to check the reply match. |
| from | The component which sent the reply. |
| fromTmpl | The expected replying component. |
| **Return Value** | Void | |
| **Constraint** | Shall be called by CH or TE to log the getreply check operation. This event occurs after getreply is checked successfully. This event is used for logging the intercomponent communication. | |
| **Effect** | The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document. | |

##### 7.3.4.1.114 tliPrCatchChecked\_m

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

##### 7.3.4.1.115 tliPrCatchChecked\_c

|  |  |  |
| --- | --- | --- |
| **Signature** | void tliPrCatchChecked\_c(in TString am, in TInteger ts,  in TString src,  in TInteger line, in TriComponentIdType c,  in TriPortIdType at,  in TriSignatureIdType signature,  in Value excValue, in TciValueTemplate excTmpl,  in TriComponentIdType from,  in TciNonValueTemplate fromTmpl) | |
| **In Parameters** | am | An additional message. |
| ts | The time when the event is produced. |
| src | The source file of the test specification. |
| line | The line number where the request is performed. |
| c | The component which produces this event. |
| at | The port via which the exception is received. |
| signature | The signature relating to the exception. |
| excValue | The catched exception. |
| excTmpl | The template used to check the exception match. |
| from | The component which sent the reply. |
| fromTmpl | The expected replying component. |
| **Return Value** | Void | |
| **Constraint** | Shall be called by CH or TE to log the catch check operation. This event occurs after catch is checked successfully. This event is used for logging the intercomponent communication. | |
| **Effect** | The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document. | |

##### 7.3.4.1.116 tliCheckedAny\_m

|  |  |  |
| --- | --- | --- |
| **Signature** | void tliCheckedAny\_m(in TString am, in TInteger ts,  in TString src, in TInteger line, in TriComponentIdType c,  in TriPortIdType at, in Value addrValue,  in TciValueTemplate addressTmpl) | |
| **In Parameters** | am | An additional message. |
| ts | The time when the event is produced. |
| src | The source file of the test specification. |
| line | The line number where the request is performed. |
| c | The component which produces this event. |
| at | The port via which the message is received. |
| addrValue | The address value of the source within the SUT. |
| addressTmpl | The expected address of the source within the SUT. |
| **Return Value** | Void | |
| **Constraint** | Shall be called by SA or TE to log a check any operation. This event occurs after the check is successfully performed. This event is used for logging the communication with the SUT. | |
| **Effect** | The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document. | |

##### 7.3.4.1.117 tliCheckedAny\_c

|  |  |  |
| --- | --- | --- |
| **Signature** | void tliCheckedAny\_c(in TString am, in TInteger ts,  in TString src, in TInteger line, in TriComponentIdType c,  in TriPortIdType at, in TriComponentIdType from,  in TciNonValueTemplate fromTmpl) | |
| **In Parameters** | am | An additional message. |
| ts | The time when the event is produced. |
| src | The source file of the test specification. |
| line | The line number where the request is performed. |
| c | The component which produces this event. |
| at | The port via which the message is received. |
| from | The component which sent the message. |
| fromTmpl | The expected sender component. |
| **Return Value** | Void | |
| **Constraint** | Shall be called by CH or TE to log a check any operation. This event occurs after the check is successfully performed. This event is used for logging the intercomponent communication. | |
| **Effect** | The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document. | |

##### 7.3.4.1.118 tliCheckAnyMismatch\_m

|  |  |  |
| --- | --- | --- |
| **Signature** | void tliCheckAnyMismatch\_m(in TString am, in TInteger ts,  in TString src, in TInteger line, in TriComponentIdType c,  in TriPortIdType at,  in Value addrValue,  in TciValueTemplate addressTmpl) | |
| **In Parameters** | am | An additional message. |
| ts | The time when the event is produced. |
| src | The source file of the test specification. |
| line | The line number where the request is performed. |
| c | The component which produces this event. |
| at | The port via which the message is received. |
| addrValue | The address value of the source within the SUT. |
| addressTmpl | The expected address of the source within the SUT. |
| **Return Value** | Void | |
| **Constraint** | Shall be called by TE to log a mismatch that occurred during a check any operation. This event occurs after checking an address match. This event is used for logging the communication with the SUT. | |
| **Effect** | The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document. | |

##### 7.3.4.1.119 tliCheckAnyMismatch\_c

|  |  |  |
| --- | --- | --- |
| **Signature** | void tliCheckAnyMismatch\_c(in TString am, in TInteger ts,  in TString src, in TInteger line, in TriComponentIdType c,  in TriPortIdType at,  in TriComponentIdType from,  in TciNonValueTemplate fromTmpl) | |
| **In Parameters** | am | An additional message. |
| ts | The time when the event is produced. |
| src | The source file of the test specification. |
| line | The line number where the request is performed. |
| c | The component which produces this event. |
| at | The port via which the message is received. |
| from | The component which sent the message. |
| fromTmpl | The expected sender component. |
| **Return Value** | Void | |
| **Constraint** | Shall be called by TE to log a mismatch that occurred during a check any operation. This event occurs after checking a component match. This event is used for logging the intercomponent communication. | |
| **Effect** | The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document. | |

##### 7.3.4.1.120 tliRnd

|  |  |  |
| --- | --- | --- |
| **Signature** | void tliRnd(in TString am, in TInteger ts,  in TString src, in TInteger line, in TriComponentIdType c,  in FloatValue val,  in FloatValue seed) | |
| **In Parameters** | am | An additional message. |
| ts | The time when the event is produced. |
| src | The source file of the test specification. |
| line | The line number where the request is performed. |
| c | The component which produces this event. |
| val | The value returned by the random number generator. |
| seed | The value used by the random number generator as seed. |
| **Return Value** | Void | |
| **Constraint** | Shall be called by TE to log a call to the random number generator function rnd. This event occurs after the random number has been generated. | |
| **Effect** | The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document. | |

##### 7.3.4.1.121 tliEvaluate

|  |  |  |
| --- | --- | --- |
| Signature | void tliEvaluate (in TString am, in TInteger ts, in TString src,  in TInteger line, in TriComponentIdType c,  in QualifiedName name, in Value evalResult) | |
| In Parameters | am | An additional message. |
| ts | The time when the event is produced. |
| src | The source file of the test specification. |
| line | The line number where the request is performed. |
| c | The component which produces this event. |
| name | The name of the template or variable. |
| evalResult | The result of evaluation. |
| Return Value | void | |
| Constraint | Shall be called by TE to log the result of evaluation of a @lazy or @fuzzy template or variable. This event occurs after the template or variable has been evaluated. | |
| Effect | The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document. | |

##### 7.3.4.1.122 tliCCall

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

##### 7.3.4.1.123 tliCCallTerminated

|  |  |  |
| --- | --- | --- |
| Signature | void tliCCallTerminated(in TString am, in TInteger ts, in TString src,  in TInteger line, in TriComponentIdType c,  in VerdictValue verdict, in TString reason,  in TciParameterListType tciPars, in Value returnValue) | |
| In Parameters | am | An additional message. |
| ts | The time when the event is produced. |
| src | The source file of the test specification. |
| line | The line number where the request is performed. |
| c | The component which produces this event. |
| verdict | The verdict of the component. |
| reason | The optional reason of the setverdict statement. |
| tciPars | The parameters of the called behaviour. |
| reason | The return value of the called behaviour. |
| Return Value | void | |
| Constraint | Shall be called by TE to log implicit termination of a called component. This event occurs after the termination of the component. | |
| Effect | The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document. | |

##### 7.3.4.1.124 tliCtrlStartWithParameters

|  |  |  |
| --- | --- | --- |
| Signature | void tliCtrlStartWithParameters(in TString am, in TInteger ts,  in TString src, in TInteger line,  in TriComponentIdType c, in TciParameterListType tciPars) | |
| In Parameters | am | An additional message. |
| ts | The time when the event is produced. |
| src | The source file of the test specification. |
| line | The line number where the request is performed. |
| c | The component which produces this event. |
| tciPars | The list of parameters passed to the module control function. |
| Return Value | void | |
| Constraint | Shall be called by TE to log the start of the module control function in cases when the module control function defines parameters or returns a value. This event occurs before the control is started. | |
| Effect | The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document. | |

##### 7.3.4.1.125 tliCtrlTerminatedWithResult

|  |  |  |
| --- | --- | --- |
| Signature | void tliCtrlTerminated (in TString am, in TInteger ts,  in TString src, in TInteger line,  in TriComponentIdType c,  in Value returnValue, in TciParameterListType tciPars) | |
| In Parameters | am | An additional message. |
| ts | The time when the event is produced. |
| src | The source file of the test specification. |
| line | The line number where the request is performed. |
| c | The component which produces this event. |
| returnValue | The value returned by the module control function. |
| tciPars | The list of parameters passed from the module control function. |
| Return Value | void | |
| Constraint | Shall be called by TM or TE to log the termination of the module control function in cases when the module control function defines parameters or returns a value. This event occurs after the control has terminated. | |
| Effect | The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document. | |

# 8 JavaTM language mapping

## 8.1 Introduction

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

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

## 8.2 Names and scopes

### 8.2.1 Names

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

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

EXAMPLE: The IDL type **TciTestCaseIdType** maps to TciTestCaseId in Java™.

The resulting mapping conforms to the standard Java™ coding conventions.

### 8.2.2 Scopes

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

## 8.3 Type mapping

### 8.3.1 Basic type mapping

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

Table : Basic type mappings

|  |  |
| --- | --- |
| IDL Type | Java™ Type |
| Tboolean | boolean |
| Tfloat | float |
| Tinteger | int |
| Tstring | java.lang.String |
| TstringSeq | java.lang.String[] |

boolean

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

float

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

char

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

int

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

String

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

String[]

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

Universal Char

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

### 8.3.2 Structured type mapping

#### 8.3.2.0 General principles

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

#### 8.3.2.1 TciParameterType

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

// TCI IDL TciParameterType

package org.etsi.ttcn.tci;

public interface TciParameter {

public String getParameterName();

public void setParameterName(String name);

public int getParameterPassingMode();

public void setParameterPassingMode(TciParameterPassingMode mode);

public Value getParameter();

public void setParameter(Value parameter);

}

**Methods:**

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

#### 8.3.2.2 TciParameterPassingModeType

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

// TCI IDL TciParameterPassingModeType

package org.etsi.ttcn.tci;

public interface TciParameterPassingMode {

public final static int TCI\_IN = 0;

public final static int TCI\_INOUT = 1;

public final static int TCI\_OUT = 2;

}

**Constants:**

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

#### 8.3.2.3 TciParameterListType

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

// TCI IDL TciParameterListType

package org.etsi.ttcn.tci;

public interface TciParameterList {

public int size() ;

public boolean isEmpty() ;

public java.util.Enumeration getParameters() ;

public TciParameter get(int index) ;

public void clear() ;

public void add(TciParameter parameter) ;

public void setParameters(TciParameter[] parameters) ;

}

**Methods:**

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

#### 8.3.2.4 TciTypeClassType

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

// TCI IDL TciTypeClassType

package org.etsi.ttcn.tci;

public interface TciTypeClass {

public final static int ADDRESS = 0 ;

public final static int ANYTYPE = 1 ;

public final static int BITSTRING = 2 ;

public final static int BOOLEAN = 3 ;

public final static int CHARSTRING = 5 ;

public final static int COMPONENT = 6 ;

public final static int ENUMERATED = 7 ;

public final static int FLOAT = 8 ;

public final static int HEXSTRING = 9 ;

public final static int INTEGER = 10 ;

public final static int OCTETSTRING = 12 ;

public final static int RECORD = 13 ;

public final static int RECORD\_OF = 14 ;

public final static int ARRAY = 15 ;

public final static int SET = 16 ;

public final static int SET\_OF = 17 ;

public final static int UNION = 18 ;

public final static int UNIVERSAL\_CHARSTRING = 20 ;

public final static int VERDICT = 21 ;

public final static int DEFAULT = 22 ;

public final static int PORT = 23 ;

public final static int TIMER = 24 ;

}

#### 8.3.2.5 TciTestComponentKindType

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

// TCI IDL TciTestComponentKindType

public interface TciTestComponentKind {

public final static int TCI\_CTRL\_COMP = 0;

public final static int TCI\_MTC\_COMP = 1;

public final static int TCI\_PTC\_COMP = 2;

public final static int TCI\_SYSTEM\_COMP = 3;

public final static int TCI\_ALIVE\_COMP = 4;

}

#### 8.3.2.6 TciBehaviourIdType

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

// TCI IDL TciBehaviourIdType

package org.etsi.ttcn.tci;

public interface TciBehaviourId extends QualifiedName {

}

#### 8.3.2.7 TciTestCaseIdType

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

// TCI IDL TciTestCaseIdType

package org.etsi.ttcn.tci;

public interface TciTestCaseId extends QualifiedName {

}

#### 8.3.2.8 TciModuleIdType

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

// TCI IDL TciModuleIdType

package org.etsi.ttcn.tci;

public interface TciModuleId extends QualifiedName {

}

#### 8.3.2.9 TciModuleParameterIdType

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

// TCI IDL TciModuleParameterIdType

package org.etsi.ttcn.tci;

public interface TciModuleParameterId extends QualifiedName {

}

#### 8.3.2.10 TciModuleParameterListType

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

// TCI IDL TciModuleParameterListType

package org.etsi.ttcn.tci;

public interface TciModuleParameterList {

public int size() ;

public boolean isEmpty() ;

public java.util.Enumeration getModuleParameters() ;

public TciModuleParameter get(int index) ;

}

**Methods:**

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

#### 8.3.2.11 TciModuleParameterType

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

// TCI IDL TciModuleParameterType

package org.etsi.ttcn.tci;

public interface TciModuleParameter {

public TciModuleParameterId getModuleParameterName();

public Value getDefaultValue();

}

**Methods:**

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

#### 8.3.2.12 TciParameterTypeListType

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

// TCI IDL TciParameterTypeListType

package org.etsi.ttcn.tci;

public interface TciParameterTypeList {

public int size() ;

public boolean isEmpty() ;

public java.util.Enumeration getParameterTypes() ;

public TciParameterType get(int index) ;

}

**Methods:**

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

#### 8.3.2.13 TciParameterTypeType

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

// TCI IDL TciParameterTypeType

package org.etsi.ttcn.tci;

public interface TciParameterType {

public String getParameterName();

public Type getParameterType() ;

public int getParameterPassingMode();

}

**Methods:**

* getParameterName Returns the parameter name as defined in the TTCN‑3 specification.
* getParameterType Returns the Type of the parameter.
* getParameterPassingMode Returns the parameter passing mode of this parameter.

#### 8.3.2.14 TciModuleIdListType

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

// TCI IDL TciModuleIdListType

package org.etsi.ttcn.tci;

public interface TciModuleIdList {

public int size() ;

public boolean isEmpty() ;

public java.util.Enumeration tciGetImportedModules() ;

public TciModuleId get(int index) ;

}

**Methods:**

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

#### 8.3.2.15 TciTestCaseIdListType

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

// TCI IDL TciTestCaseIdListType

package org.etsi.ttcn.tci;

public interface TciTestCaseIdList {

public int size() ;

public boolean isEmpty() ;

public java.util.Enumeration tciGetTestCases() ;

public TciTestCaseId get(int index) ;

}

**Methods:**

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

#### 8.3.2.16 TciDecodingResult

**TciDecodingResult** is used as a return type of the TciCDProvided.decodeValue operation and it is defined as follows:

package org.etsi.ttcn.tci;

public interface TciDecodingResult {

public int getResult ();

public Value getDecodedValue ();

}

**Methods:**

* getResult Returns the numeric result of the TciCDProvided.decodeValue operation.
* getDecodedValue Returns the decoded value or the distinct value null.

#### 8.3.2.17 TciMatchingTypeType

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

// TCI IDL TciTypeClassType

package org.etsi.ttcn.tci;

public interface TciMatchingType {

public final static int TEMPLATE\_LIST = 0 ;

public final static int COMPLEMENTED\_LIST = 1 ;

public final static int ANY\_VALUE = 2 ;

public final static int ANY\_VALUE\_OR\_NONE = 3 ;

public final static int VALUE\_RANGE = 4 ;

public final static int SUBSET = 5 ;

public final static int SUPERSET = 6 ;

public final static int ANY\_ELEMENT = 7 ;

public final static int ANY\_ELEMENTS\_OR\_NONE = 8 ;

public final static int PATTERN = 9 ;

public final static int MATCH\_DECODED\_CONTENT = 10 ;

public final static int OMIT\_TEMPLATE = 11 ;

}

#### 8.3.2.18 LengthRestriction

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

// TCI IDL LengthRestriction

package org.etsi.ttcn.tci;

public interface LengthRestriction {

public int getLowerBoundary ();

public int getUpperBoundary ();

public void setLowerBoundary (int boundary);

public void setUpperBoundary (int boundary);

public boolean isUpperBoundaryInfinity ();

public void setInfiniteUpperBoundary ();

}

**Methods:**

* getLowerBoundary Returns the lower boundary of the length restriction.
* getUpperBoundary Returns the upper boundary of the length restriction.
* setLowerBoundary Sets the lower boundary value.
* setUpperBoundary Sets the upper boundary value.
* isUpperBoundaryInfinity Returns true if the upper boundary contains infinity and false otherwise.
* setInfiniteUpperBoundary Sets the upper boundary to infinity.

#### 8.3.2.19 Permutation

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

// TCI IDL Permutation

package org.etsi.ttcn.tci;

public interface Permutation {

public int getStartPosition ();

public void setStartPosition (int position);

public int getLength ();

public void setLength (int length);

}

**Methods:**

* getStartPosition Returns the position of the first item of the permutation in the RecordOfValue.
* setStartPosition Sets the position of the first item of the permutation in the RecordOfValue.
* getLength Returns the number of elements or matching mechanisms of the RecordOfValue that are included in the permutation.
* setLength Sets the number of elements or matching mechanisms of the RecordOfValue that are included in the permutation.

#### 8.3.2.20 RangeBoundary

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

// TCI IDL RangeBoundary

package org.etsi.ttcn.tci;

public interface RangeBoundary {

public Value getBoundary ();

public boolean isInclusive ();

public void setBoundary (Value boundary, boolean isInclusive);

public boolean isInfinity ();

public void setToInfinity ();

}

**Methods:**

* getBoundary Returns the boundary value. Dependent on the type of the value range, the return value can be either an IntegerValue, FloatValue, CharstringValue or UniversalCharstringValue. If the boundary is undefined or it cannot be represented by a Value instance (infinity in case of integer values), the distinct value null is returned.
* isInclusive Returns true if the boundary value is a part of the allowed range and false otherwise.
* setBoundary Sets the boundary value. Dependent on the type of the value range, the boundary parameter can contain either an IntegerValue, FloatValue, UniversalCharstringValue or CharstringValue. The inclusive parameter determines whether the boundary value is a part of the range (true) or not (false).
* isInfinity Returns true if the boundary is equal to infinity and false otherwise.
* setToInfinity Sets the boundary to infinity (if the instance is used for the upper range boundary) or –infinity (if the instance is used for the lower range boundary).

### 8.3.3 Abstract type mapping

#### 8.3.3.0 General principles

The TTCN‑3 data types are modelled in Java™ using the abstract type mapping as defined in the clause 8.3.3. The Type interface defines only operations used to retrieve in TTCN‑3 defined types. No TTCN‑3 types can be constructed using the Type interface. Types are modelled using the single interface Type, that provides methods to identify types and to retrieve values of a given type.

#### 8.3.3.1 Type

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

// TCI IDL Type

package org.etsi.ttcn.tci;

public interface Type {

public TciModuleId getDefiningModule ();

public String getName ();

public int getTypeClass ();

public Value newInstance ();

public String getTypeEncoding ();

public String getTypeEncodingVariant();

public String[] getEncodeAttributes();

public String[] getVariantAttributes(String encoding);

public String[] getTypeExtension();

public Value parseValue (String val);

public MatchingMechanism newTemplate (int matchingType);

public RangeBoundary getLowerTypeBoundary();

public RangeBoundary getUpperTypeBoundary();

public LengthRestriction getTypeLengthRestriction();

public MatchingMechanism getTypeMatchingMechanism();

}

**Methods:**

* getDefiningModule Returns the module identifier of the module the type has been defined in. If the type represents a TTCN‑3 base type the distinct value null will be returned.
* getName Returns name of the type as defined in the TTCN‑3 module.
* getTypeClass Returns the type class of the respective type. A value of TciTypeClassType can have on of the following constants: ADDRESS, ANYTYPE, BITSTRING, BOOLEAN, CHARSTRING, COMPONENT, ENUMERATED, FLOAT, HEXSTRING, INTEGER, OCTETSTRING, RECORD, RECORD\_OF, ARRAY, SET, SET\_OF, UNION, UNIVERSAL\_CHARSTRING, VERDICT, DEFAULT, PORT, TIMER.
* newInstance Returns a freshly created value of the given type. This initial value of the created value is undefined.
* getTypeEncoding Returns the type encoding attribute as defined in the TTCN‑3 module.
* getTypeEncodingVariant This operation returns the value encoding variant attribute as defined in the TTCN-3 module, if any. If no encoding variant attribute has been defined the distinct value null will be returned.
* getEncodeAttributes Returns all encode attributes of the type.
* getVariantAttributes Returns all variant attributes of the type for the specified encoding.
* getTypeExtension Returns the type extension attribute as defined in the TTCN‑3 module.
* parseValue Parses the value provided in the parameter and in case of successful parsing returns a Value object representing the parsed value. In case of an error or if value parsing is not supported by the tool, the method returns null.
* newTemplate Returns a freshly created matching mechanism of this type. The matchingType parameter determines what kind of matching mechanism will be created and it shall be one of the following constants: TEMPLATE\_LIST, COMPLEMENTED\_LIST, ANY\_VALUE, ANY\_VALUE\_OR\_NONE, VALUE\_RANGE, SUBSET, SUPERSET, ANY\_ELEMENT, ANY\_ELEMENTS\_OR\_NONE, PATTERN, DECODED\_MATCH. If the created matching mechanism contains additional data properties, these properties are uninitialized in the created matching mechanism.
* getLowerTypeBoundary Return the lower boundary of the type restriction or null.
* getUpperTypeBoundary Return the upper boundary of the type restriction or null.
* getTypeLengthRestriction Return the length restriction of the type or null.
* getTypeMatchingMechanism Return the matching mechanism restriction of the type or null.

### 8.3.4 Abstract value mapping

#### 8.3.4.0 General principles

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

#### 8.3.4.1 Value

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

// TCI IDL Value

package org.etsi.ttcn.tci;

public interface Value {

public Type getType();

public boolean notPresent();

public String getValueEncoding();

public String getValueEncodingVariant();

public String[] getEncodeAttributes();

public String[] getVariantAttributes(String encoding);

public boolean isMatchingSymbol();

public String valueToString ();

public boolean isLazy();

public boolean isFuzzy();

public boolean isEvaluated();

public LengthRestriction getLengthRestriction ();

public LengthRestriction newLengthRestriction ();

public void setLengthRestriction (LengthRestriction restriction);

public boolean isIfPresentEnabled ();

public void setIfPresentEnabled (boolean enabled);

public RangeBoundary getLowerTypeBoundary();

public RangeBoundary getUpperTypeBoundary();

public LengthRestriction getTypeLengthRestriction();

public MatchingMechanism getTypeMatchingMechanism();

public boolean isOptional();

}

public LengthRestriction getTypeLengthRestriction();

**Methods:**

* getType Returns the type of the specified value.
* notPresent Returns true if the specified value is omit, false otherwise.
* getValueEncoding This operation returns the value encoding attribute as defined in the TTCN‑3module, if any. If no encoding attribute has been defined the distinct valuenull will be returned.
* getValueEncodingVariant This operation returns the value encoding variant attribute as defined in TTCN‑3, if any. If no encoding variant attribute has been defined the distinct value null will be returned.
* getEncodeAttributes Returns all encode attributes of the value.
* getVariantAttributes Returns all variant attributes of the value for the specified encoding.
* isMatchingSymbol Returns true if the specified value is a matching symbol (see clause 7.2.2.2.1 for more details), false otherwise.
* valueToString Returns the same string as produced by the any2unistr predefined function with the specified value as its parameter.
* isLazy Returns true if the specified value is @lazy, false otherwise.
* isFuzzy Returns true if the specified value is @fuzzy, false otherwise.
* isEvaluated Returns true if the specified value contains an evaluation result, false otherwise (see clause 7.2.2.2.1 for more details).
* getLengthRestriction Returns a length restriction matching attribute in case it is attached to the value or the distinct value null if no such matching attribute is present.
* newLengthRestriction Creates a new instance of the LengthRestriction interface.
* setLengthRestriction Adds a length restriction matching to the value or modifies an existing one. The distinct value null can be used as a parameter to disable an existing length restriction.
* isIfPresentEnabled Returns true if the ifpresent is attached to the value and false otherwise.
* setIfPresentEnabled Sets the whether the ifpresent indicator is attached to the value or not.
* getLowerTypeBoundary Return the lower boundary of the value's type restriction or null.
* getUpperTypeBoundary Return the upper boundary of the value's type restriction or null.
* getTypeLengthRestriction Return the value's type length restriction or null.
* getTypeMatchingMechanism Return the value's type restriction.
* isOptional Returns true if and only if the value is either an optional field or a template without value or present template restriction.

#### 8.3.4.2 IntegerValue

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

// IntegerValue

package org.etsi.ttcn.tci;

public interface IntegerValue {

public void setInteger(int value);

public int getInteger();

}

**Methods:**

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

#### 8.3.4.3 FloatValue

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

// FloatValue

package org.etsi.ttcn.tci;

public interface FloatValue {

public void setFloat(float value);

public float getFloat();

public void setDouble(double value);

public double getDouble();

public void setBigDecimal(java.math.BigDecimal value);

public java.math.BigDecimal getBigDecimal();

}

**Methods:**

* setFloat Sets this FloatValue to the float value value.
* getFloat Returns the float value represented by this FloatValue.
* setDouble Sets this FloatValue to the float value with double precision.
* getDouble Returns the float value represented by this FloatValue with double precision.
* setBigDecimal Sets this FloatValue to the float value with user-defined precision. If loss of precision is probable when using float or double, this method should be used.
* getBigDecimal Returns the float value represented by this FloatValue with user-defined precision. If loss of precision is probable when using float or double, this method should be used.

#### 8.3.4.4 BooleanValue

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

// BooleanValue

package org.etsi.ttcn.tci;

public interface BooleanValue {

public void setBoolean(boolean value);

public boolean getBoolean();

}

**Methods:**

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

#### 8.3.4.5 CharstringValue

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

// TCI IDL CharstringValue

package org.etsi.ttcn.tci;

public interface CharstringValue {

String getString ();

void setString (String value);

char getChar (int position);

void setChar (int position, char value);

int getLength ();

void setLength (int len);

}

**Methods:**

* getString Returns the string of the TTCN‑3 charstring. The textual representation of the empty TTCN‑3 charstring is "", while its length is zero.
* setString Sets this CharstringValue to value.
* getChar Returns the char value of the TTCN‑3 charstring at position. position 0 denotes the first char of the TTCN‑3 charstring. Valid values for position are 0 to length - 1.
* setChar Set the char at position to value. Valid values for position are 0 to length - 1.
* getLength Returns the length of this CharstringValue in chars, zero if the value of this CharstringValue is omit.
* setLength Sets the length of this CharstringValue in chars to len.

#### 8.3.4.6 BitstringValue

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

// TCI IDL BitstringValue

package org.etsi.ttcn.tci;

public interface BitstringValue {

String getString ();

void setString (String value);

int getBit (int position);

void setBit (int position, int value);

int getLength ();

void setLength (int len);

public java.io.InputStream getInputStream();

public void setInputStream(java.io.InputStream stream, int numberOfBits);

boolean isMatchingAt (int position);

MatchingMechanism getMatchingAt (int position);

void setMatchingAt (int position, MatchingMechanism template);

}

**Methods:**

* getString Returns the textual representation of this BitstringValue, as defined in TTCN‑3. E.g. the textual representation of 0101 is '0101'B. The textual representation of the empty TTCN‑3 bitstring is ''B, while its length is zero. In some cases, the message data are available in the form of a stream and cannot be converted into a string by TCI (e.g. because of memory restrictions). In such cases, this method returns null and the getInputStream method shall be used for reading the data.
* setString Sets the value of this BitstringValue according to the textual representation as defined by value. E.g. The value of this BitstringValue will be 0101 if the textual representation in value is '0101'B.
* getBit Returns the value (0 | 1) at position of this TTCN‑3 bitstring. position 0 denotes the first bit of the TTCN‑3 bitstring. Valid values for position are 0 to length - 1.
* setBit Set the bit at position to value (0 | 1). position 0 denotes the first bit in this BitstringValue. Valid values for position are 0 to length - 1.
* getLength Returns the length of this BitstringValue in bits, zero if the value of this BitstringValue is omit.
* setLength Sets the length of this BitstringValue in bits to len.
* getInputStream Returns the bits in the form of an input stream. Repeated calls to the same method return different stream instances. The method returns null if the bitstring contains matching symbols.
* setInputStream Sets the value of this BitstringValue by providing a source stream that is used by the BitstringValue object to read the value content. Values set this way do not contain matching symbols.
* isMatchingAt Returns true if the item at position of this TTCN‑3 bitstring is a matching mechanism inside a value (AnyElement, AnyElementsOrNone) and false otherwise.
* getMatchingAt If the position of this TTCN‑3 bitstring contains a matching mechanism inside a value (AnyElement, AnyElementsOrNone), the method returns it. Otherwise the distinct value null is returned.
* setMatching Sets a matching mechanism at position. Only two matching mechanisms are allowed: AnyElement and AnyElementsOrNone.

#### 8.3.4.7 OctetstringValue

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

// TCI IDL OctetstringValue

package org.etsi.ttcn.tci;

public interface OctetstringValue {

String getString ();

void setString (String value);

int getOctet (int position);

void setOctet (int position, int value);

int getLength ();

void setLength (int len);

public java.io.InputStream getInputStream();

public void setInputStream(java.io.InputStream stream);

boolean isMatchingAt (int position);

MatchingMechanism getMatchingAt (int position);

void setMatchingAt (int position, MatchingMechanism template);

}

**Methods:**

* getString Returns the textual representation of this OctetstringValue, as defined in TTCN‑3. E.g. the textual representation of 0xCAFFEE is 'CAFFEE'O. The textual representation of the empty TTCN‑3 octetstring is ''O, while its length is zero. In some cases, the message data are available in the form of a byte stream and cannot be converted into a string by TCI (e.g. because of memory restrictions). In such cases, this method returns null and the getInputStream method shall be used for reading the data.
* setString Sets the value of this OctetstringValue according to the textual representation as defined by value. E.g. the value of this OctetstringValue will be 0xCAFFEE if the textual representation in value is 'CAFFEE'O.
* getOctet Returns the value (0..255) at position of this TTCN‑3 octetstring. position 0 denotes the first octet of the TTCN‑3 octetstring. Valid values for position are 0 to length - 1.
* setOctet Set the octet at position to value (0..255). position 0 denotes the first octet in the octetstring. Valid values for position are 0 to length - 1.
* getLength Returns the length of this OctetstringValue in octets, zero if the value of this OctetstringValue is omit.
* setLength Sets the length of this OctetstringValue in octets to len.
* getInputStream Returns the octets in the form of an input stream. Repeated calls to the same method return different stream instances. The method returns null if the octetstring contains matching symbols.
* setInputStream Sets the value of this OctetstringValue by providing a source stream that is used by the OctetstringValue object to read the value content. Values set this way do not contain matching symbols.
* isMatchingAt Returns true if the item at position of this TTCN‑3 octetstring is a matching mechanism inside a value (AnyElement, AnyElementsOrNone) and false otherwise.
* getMatchingAt If the position of this TTCN‑3 octetstring contains a matching mechanism inside a value (AnyElement, AnyElementsOrNone), the method returns it. Otherwise the distinct value null is returned.
* setMatching Sets a matching mechanism at position. Only two matching mechanisms are allowed: AnyElement and AnyElementsOrNone.

#### 8.3.4.8 UniversalCharstringValue

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

// TCI IDL UniversalCharstringValue

package org.etsi.ttcn.tci;

public interface UniversalCharstringValue {

String getString ();

void setString (String value);

int getChar (int position);

void setChar (int position, int value);

int getLength ();

void setLength (int len);

}

**Methods:**

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

#### 8.3.4.9 HexstringValue

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

// TCI IDL HexstringValue

package org.etsi.ttcn.tci;

public interface HexstringValue {

String getString ();

void setString (String value);

int getHex (int position);

void setHex (int position, int value);

int getLength ();

void setLength (int len);

public java.io.InputStream getInputStream();

public void setInputStream(java.io.InputStream stream, int numberOfItems);

boolean isMatchingAt (int position);

MatchingMechanism getMatchingAt (int position);

void setMatchingAt (int position, MatchingMechanism template);

}

**Methods:**

* getString Returns the textual representation of this HextstringValue, as defined in TTCN‑3. E.g. the textual representation of 0xAFFEE is 'AFFEE'H. The textual representation of the empty TTCN‑3 hexstring is ''H, while its length is zero. In some cases, the message data are available in the form of a byte stream and cannot be converted into a string by TCI (e.g. because of memory restrictions). In such cases, this method returns null and the getInputStream method shall be used for reading the data.
* setString Sets the value of this HexstringValue according to the textual representation as defined by value. E.g. the value of this HexstringValue will be 0xAFFEE if the textual representation in value is 'AFFEE'H.
* getHex Returns the value (0...15) at position of this TTCN‑3 hexstring. position 0 denotes the first hex digits of the TTCN‑3 hexstring. Valid values for position are 0 to length - 1.
* setHex Set the hex digit at position to value (0...16). position 0 denotes the first octet in the hexstring. Valid values for position are 0 to length - 1.
* getLength Returns the length of this HexstringValue in octets, zero if the value of this HexstringValue is omit.
* setLength Sets the length of this HexstringValue in hex digits to len.
* getInputStream Returns the content in the form of an input stream. Repeated calls to the same method return different stream instances. The method returns null if the hexstring contains matching symbols.
* setInputStream Sets the value of this HexstringValue by providing a source stream that is used by the HexstringValue object to read the value content. Values set this way do not contain matching symbols.
* isMatchingAt Returns true if the item at position of this TTCN‑3 hexstring is a matching mechanism inside a value (AnyElement, AnyElementsOrNone) and false otherwise.
* getMatchingAt If the position of this TTCN‑3 hexstring contains a matching mechanism inside a value (AnyElement, AnyElementsOrNone), the method returns it. Otherwise the distinct value null is returned.
* setMatching Sets a matching mechanism at position. Only two matching mechanisms are allowed: AnyElement and AnyElementsOrNone.

#### 8.3.4.10 RecordValue

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

// TCI IDL RecordValue

package org.etsi.ttcn.tci;

public interface RecordValue {

public Value getField(String fieldName) ;

public void setField(String fieldName, Value value) ;

public String[] getFieldNames() ;

public void setFieldOmitted(String fieldName);

}

**Methods:**

* getField Returns the value of the field named fieldName. The return value is the common abstract base type Value, as a record field can have any type defined in TTCN‑3.If the field cannot be obtained from the record the distinct value null will be returned.
* setField Set the field named fieldName of the record to value. No assumption shall be made on how a field is stored in a record. An internal implementation might choose to store a reference to this value or to copy the value. It is safe to assume that the value will be copied. Therefore it should be assumed that subsequent modifications of value will not be considered in the record.
* getFieldNames Returns an array of String of field names, the empty sequence, if the record has no fields.
* setFieldOmitted Set the field named fieldName of the record to omit.

#### 8.3.4.11 RecordOfValue

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

// TCI IDL RecordOfValue

package org.etsi.ttcn.tci;

public interface RecordOfValue {

public Value getField(String fieldName) ;

public void setField(int position, Value value) ;

public void appendField(Value value) ;

public Type getElementType() ;

public int getLength() ;

public void setLength(int len) ;

public int getOffset() ;

public int getPermutationCount ();

public Permutation getPermutation (int index);

public Permutation newPermutation ();

public void definePermutation (Permutation permutation);

public void removePermutation (int index);

public void clearPermutations ();

}

**Methods:**

* getField Returns the value of the record of at position if position is between zero and length ‑ 1, the distinct value null otherwise. The return value is the common abstract base type Value, as a record of can have fields of any type defined in TTCN‑3.
* setField Sets the field at position to value. If position is greater than (length ‑ 1) the record of will be extended to have the length   
  (position + 1). The record of elements between the original position at length and position ‑ 1 will be set to OMIT. No assumption shall be made on how a field is stored in a record of. An internal implementation might choose to store a reference to this value or to copy the value. It is safe to assume that the value will be copied. Therefore it should be assumed that subsequent modifications of value will not be considered in the record of.
* appendField Appends the value at the end of the record of, i.e. at position length. No assumption shall be made on how a field is stored in a record of. An internal implementation might choose to store a reference to this value or to copy the value. It is safe to assume that the value will be copied. Therefore it should be assumed that subsequent modifications of value will not be considered in the record of.
* getElementType This operation will return the Type of the elements of this record of.
* getLength Returns the actual length of the record of value, zero if the record of value is OMIT.
* setLength Set the length of the record of to len. If len is greater than the original length, newly created elements have the value OMIT. If len is less or equal than the original length this operation will be ignored.
* getOffset Returns the lowest possible index. For a record of or set of value this is always 0. For an array value, this is the lower index bound used in the type definition.
* getPermutationCount Returns the number of permutations in the record of or array value.
* getPermutation Returns the permutation at the specified index. The allowed index range is from 0 to (getPermutationCount() – 1).
* newPermutation Creates a new instance of the Permutation interface.
* definePermutation Creates permutation from existing elements of a record of value. The permutation parameter shall not include elements that are already a part of other existing permutations attached to the same record of. No elements are added to the record of by this operation.
* removePermutation Removes the permutation at the specified index. The allowed index range is from 0 to (getPermutationCount() – 1). No elements are removed from the record of by this operation. When the operation completes, the existing elements at positions specified by the removed permutation do not belong to any permutation.
* clearPermutations Removes all permutations from the value. The elements that belonged to the removed permutation are not removed.

#### 8.3.4.12 UnionValue

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

// TCI IDL UnionValue

package org.etsi.ttcn.tci;

public interface UnionValue {

Value getVariant (String variantName);

void setVariant (String variantName, Value value);

String getPresentVariantName ();

String[] getVariantNames ();

}

**Methods:**

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

#### 8.3.4.13 EnumeratedValue

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

// TCI IDL EnumeratedValue

package org.etsi.ttcn.tci;

public interface EnumeratedValue {

String getEnum ();

void setEnum (String enumValue);

int getInt();

void setInt(int intValue);

}

**Methods:**

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

#### 8.3.4.14 VerdictValue

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

// TCI IDL VerdictValue

package org.etsi.ttcn.tci;

public interface VerdictValue {

public static final int NONE = 0;

public static final int PASS = 1;

public static final int INCONC = 2;

public static final int FAIL = 3;

public static final int ERROR = 4;

public int getVerdict() ;

public void setVerdict(int verdict) ;

}

**Methods:**

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

#### 8.3.4.15 AddressValue

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

// TCI IDL Address\_Value

package org.etsi.ttcn.tci;

public interface AddressValue {

public int getAddress() ;

public void setAddress(Value value) ;

}

**Methods:**

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

### 8.3.5 Abstract template mapping

#### 8.3.5.0 General principles

TTCN‑3 matching mechanisms can be retrieved from the TE and constructed using the MatchingMechanism interface. The template mapping interface is constructed hierarchically with MatchingMechanism as the basic interface. Specialized interfaces for different types of matching mechanisms have been defined.

#### 8.3.5.1 MatchingMechanism

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

// TCI IDL MatchingMechanism

package org.etsi.ttcn.tci;

public interface MatchingMechanism {

public int getMatchingType ();

}

**Methods:**

* getMatchingType Returns the matching mechanism type.

#### 8.3.5.2 MatchingList

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

// TCI IDL MatchingList

package org.etsi.ttcn.tci;

public interface MatchingList {

public int size ();

public Value get (int position);

public void add (Value item);

public void remove (int position);

public void clear ();

}

**Methods:**

* size Returns the number of items in the matching mechanism.
* get Returns a value or template at the specified position.
* add Adds a value or template to the matching mechanism.
* remove Removes a value or template from the specified position.
* clear Removes all values and templates from the matching mechanism.

#### 8.3.5.3 ValueRange

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

// TCI IDL ValueRange

package org.etsi.ttcn.tci;

public interface ValueRange {

public RangeBoundary getLowerBoundary ();

public RangeBoundary getUpperBoundary ();

public void setLowerBoundary (RangeBoundary boundary);

public void setUpperBoundary (RangeBoundary boundary);

}

**Methods:**

* getLowerBoundary Returns the lower boundary of the range.
* getUpperBoundary Returns the upper boundary of the range.
* setLowerBoundary Sets the lower boundary of the range.
* setUpperBoundary Sets the upper boundary of the range.

#### 8.3.5.4 CharacterPattern

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

// TCI IDL CharacterPattern

package org.etsi.ttcn.tci;

public interface CharacterPattern {

public Value getPatternString ();

public void setPatternString (Value pattern);

}

**Methods:**

* getPatternString Returns the character pattern definition of this pattern (either a CharstringValue or UniversalCharstringValue).
* setPatternString Sets the character pattern definition of this pattern. The characterPattern parameter shall contain either a CharstringValue or UniversalCharstringValue.

#### 8.3.5.5 MatchDecodedContent

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

// TCI IDL MatchDecodedContent

package org.etsi.ttcn.tci;

public interface MatchDecodedContent {

public Value getContent ();

public void setContent (Value pattern);

}

**Methods:**

* getContent Returns the value or matching mechanism used as an argument of the decmatch matching mechanism.
* setContent Sets the value or matching mechanism that is used as an argument of the decmatch matching mechanism.

### 8.3.6 Abstract logging types mapping

#### 8.3.6.0 General principles

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

#### 8.3.6.1 TciValueTemplate

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

// TCI IDL TciValueTemplate

package org.etsi.ttcn.tci;

public interface TciValueTemplate {

public boolean isOmit();

public boolean isAny();

public boolean isAnyOrOmit();

public String getTemplateDef();

}

**Methods:**

* isOmit Returns true if the template is omit, false otherwise.
* isAny Returns true if the template is any, false otherwise.
* isAnyOrOmit Returns true if the template is anyoromit, false otherwise.
* getTemplateDef This operation returns the template definition.

#### 8.3.6.2 TciNonValueTemplate

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

// TCI IDL TciNonValueTemplate

package org.etsi.ttcn.tci;

public interface TciNonValueTemplate {

public boolean isAny();

public boolean isAll();

public String getTemplateDef();

}

**Methods:**

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

#### 8.3.6.3 TciValueList

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

// TCI IDL TciValueList

package org.etsi.ttcn.tci;

public interface TciValueList{

public int size() ;

public boolean isEmpty() ;

public Value get(int index) ;

}

**Methods:**

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

#### 8.3.6.4 TciValueDifference

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

// TCI IDL TciValueDifference

package org.etsi.ttcn.tci;

public interface TciValueDifference {

public Value getValue();

public TciValueTemplate getTciValueTemplate();

public String getDescription();

}

**Methods:**

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

#### 8.3.6.5 TciValueDifferenceList

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

// TCI IDL TciValueDifferenceList

package org.etsi.ttcn.tci;

public interface TciValueDifferenceList{

public int size() ;

public boolean isEmpty() ;

public TciValueDifference get(int index) ;

}

**Methods:**

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

#### 8.3.6.6 ComponentStatus

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

// TCI IDL ComponentStatus

package org.etsi.ttcn.tci;

public interface ComponentStatus {

public static final int INACTIVE\_C = 0;

public static final int RUNNING\_C = 1;

public static final int STOPPED\_C = 2;

public static final int KILLED\_C = 3;

public static final int NULL\_C = 4;

public int getComponentStatus() ;

public void setComponentStatus (int componentStatus) ;

}

#### 8.3.6.7 TimerStatus

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

// TCI IDL TimerStatus

package org.etsi.ttcn.tci;

public interface TimerStatus {

public static final int RUNNING\_T = 0;

public static final int INACTIVE\_T = 1;

public static final int EXPIRED\_T = 2;

public static final int NULL\_T = 3;

public int getTimerStatus() ;

public void setTimerStatus (int timerStatus) ;

}

#### 8.3.6.8 TciStatus

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

// TCI IDL TciStatus

package org.etsi.ttcn.tci;

public interface TciStatus {

public static final int TCI\_OK = 0;

public static final int TCI\_ERROR = -1;

public int getTciStatus() ;

public void setTciStatus (int tciStatus) ;

}

## 8.4 Constants

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

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

* org.etsi.ttcn.tci.TciParameterPassingMode.TCI\_IN;
* org.etsi.ttcn.tci.TciParameterPassingMode.TCI\_INOUT;
* org.etsi.ttcn.tri.TciParameterPassingMode.TCI\_OUT.

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

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

* org.etsi.ttcn.tci.TciTypeClass.ADDRESS;
* org.etsi.ttcn.tci.TciTypeClass.ANYTYPE;
* org.etsi.ttcn.tci.TciTypeClass.BITSTRING;
* org.etsi.ttcn.tci.TciTypeClass.BOOLEAN;
* org.etsi.ttcn.tci.TciTypeClass.CHARSTRING;
* org.etsi.ttcn.tci.TciTypeClass.COMPONENT;
* org.etsi.ttcn.tci.TciTypeClass.ENUMERATED;
* org.etsi.ttcn.tci.TciTypeClass.FLOAT;
* org.etsi.ttcn.tci.TciTypeClass.HEXSTRING;
* org.etsi.ttcn.tci.TciTypeClass.INTEGER;
* org.etsi.ttcn.tci.TciTypeClass.OCTETSTRING;
* org.etsi.ttcn.tci.TciTypeClass.RECORD;
* org.etsi.ttcn.tci.TciTypeClass.RECORD\_OF;
* org.etsi.ttcn.tci.TciTypeClass.SET;
* org.etsi.ttcn.tci.TciTypeClass.SET\_OF;
* org.etsi.ttcn.tci.TciTypeClass.ARRAY;
* org.etsi.ttcn.tci.TciTypeClass.UNION;
* org.etsi.ttcn.tci.TciTypeClass.UNIVERSAL\_CHARSTRING;
* org.etsi.ttcn.tci.TciTypeClass.VERDICT.

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

* org.etsi.ttcn.tci.TciTestComponentKind.TCI\_CTRL\_COMP;
* org.etsi.ttcn.tci.TciTestComponentKind.TCI\_MTC\_COMP;
* org.etsi.ttcn.tci.TciTestComponentKind.TCI\_PTC\_COMP;
* org.etsi.ttcn.tci.TciTestComponentKind.TCI\_SYSTEM\_COMP;
* org.etsi.ttcn.tci.TciTestComponentKind.TCI\_ALIVE\_COMP.

The following constants shall be used for component status (see also clause 8.3.6.6).

* org.etsi.ttcn.tci.ComponentStatus.INACTIVE\_C;
* org.etsi.ttcn.tci.ComponentStatus.RUNNING\_C;
* org.etsi.ttcn.tci.ComponentStatus.STOPPED\_C;
* org.etsi.ttcn.tci.ComponentStatus.KILLED\_C;
* org.etsi.ttcn.tci.ComponentStatus.NULL\_C.

The following constants shall be used for timer status (see also clause 8.3.6.7).

* org.etsi.ttcn.tci.TimerStatus.RUNNING\_T;
* org.etsi.ttcn.tci.TimerStatus.INACTIVE\_T;
* org.etsi.ttcn.tci.TimerStatus.EXPIRED\_T;
* org.etsi.ttcn.tci.TimerStatus.NULL\_T.

The following constants shall be used for TCI status (see also clause 8.3.6.8).

* org.etsi.ttcn.tci.TciStatus.TCI\_OK;
* org.etsi.ttcn.tci.TciStatus.TCI\_ERROR.

## 8.5 Mapping of interfaces

### 8.5.0 Calling rules

The TCI IDL definition defines four interfaces, the **TCI‑TM,** the **TCI‑CH**, the **TCI‑CD**, and the **TCI‑TL** interface. The operations are defined for different directions within this interface, i.e. some operations can only be called by the TTCN‑3 Executable (TE), the System Adaptor (SA) or the Platform Adaptor (PA) on the Test Management and Control (TMC) while others can only be called by the TMC on the TE. This is reflected by dividing the TCI IDL interfaces in two sub interfaces, each suffixed by Required or Provided.

Table : Sub interfaces

| Calling/Called | TE | TM | CD | CH | SA | PA | TL |
| --- | --- | --- | --- | --- | --- | --- | --- |
| TE | ‑ | TCI‑TM Provided | TCI‑CD Provided | TCI‑CH Provided | See ETSI ES 201 873‑5 [3] | See ETSI ES 201 873‑5 [3] | TCI‑TL Provided |
| TM | TCI‑TM Required | ‑ | ‑ | ‑ | ‑ | ‑ | TCI‑TL Provided |
| CD | TCI‑CD Required | ‑ | ‑ | ‑ | ‑ | ‑ | TCI‑TL Provided |
| CH | TCI‑CH Required | ‑ | ‑ | ‑ | ‑ | ‑ | TCI‑TL Provided |
| SA | See ETSI ES 201 873-5 [3] | - | - | - | ‑ | ‑ | TCI‑TL Provided |
| PA | See ETSI ES 201 873-5 [3] | - | - | - | ‑ | ‑ | TCI‑TL Provided |
| TL | ‑ | ‑ | ‑ | ‑ | ‑ | ‑ | ‑ |

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

### 8.5.1 The TCI‑TM interface

#### 8.5.1.1 TCI‑TM provided

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

// TCI‑TM

// TE ‑> TM

package org.etsi.ttcn.tci;

public interface TciTMProvided {

public void tciTestCaseStarted (TciTestCaseId testCaseId, TciParameterList parameterList, Float timer);

public void tciTestCaseTerminated (VerdictValue verdict, TciParameterList parameterList);

public void tciControlTerminated ();

public Value tciGetModulePar (TciModuleParameterId parameterId);

public void tciLog (TriComponentId testComponentId, String message);

public void tciError (String message);

public void tciControlTerminatedWithResult (Value result, TciParameterList parameterList);

}

#### 8.5.1.2 TCI‑TM required

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

// TCI‑TM

// TM ‑> TE

package org.etsi.ttcn.tci;

public interface TciTMRequired {

public void tciRootModule (TciModuleId moduleName);

public TciModuleIdList tciGetImportedModules ();

public TciModuleParameterList tciGetModuleParameters (TciModuleId moduleId);

public TciTestCaseIdList tciGetTestCases ();

public TciParameterTypeList tciGetTestCaseParameters (TciTestCaseId TestCaseId);

public TriPortIdList tciGetTestCaseTSI (TciTestCaseId testCaseId);

public void tciStartTestCase

(String testCaseId, TciParameterList parameterList );

public void tciStopTestCase ();

public TriComponentId tciStartControl ();

public void tciStopControl ();

public TciParameterTypeList tciGetControlParameters ();

public TriComponentId tciStartControlWithParameters (TciParameterList parameterList);

}

### 8.5.2 The TCI‑CD interface

#### 8.5.2.1 TCI‑CD provided

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

// TCI‑CD

// TE ‑> CD

package org.etsi.ttcn.tci;

public interface TciCDProvided {

public Value decode (TriMessage message, Type decodingHypothesis );

public TriMessage encode (Value value);

public TciDecodingResult decodeValue (TriMessage message, Type decodingHypothesis,  
 String decodingInfo );

public TriMessage encodeValue (Value value, String encodingInfo);

}

#### 8.5.2.2 TCI‑CD required

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

// TCI‑CD

// CD ‑> TE

package org.etsi.ttcn.tci;

public interface TciCDRequired {

public Type getTypeForName (String typeName);

public Type getInteger ();

public Type getFloat ();

public Type getBoolean ();

public Type getCharstring ();

public Type getUniversalCharstring ();

public Type getHexstring ();

public Type getBitstring ();

public Type getOctetstring ();

public Type getVerdict ();

public void tciErrorReq (String message);

}

### 8.5.3 The TCI‑CH interface

#### 8.5.3.1 TCI‑CH provided

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

// TciCHProvided

// TE ‑> CH

package org.etsi.ttcn.tci;

public interface TciCHProvided {

public void tciSendConnected (TriPortId sender, TriComponentId receiver, Value sendMessage);

public void tciSendConnectedBC (TriPortId sender, Value sendMessage);

public void tciSendConnectedMC (TriPortId sender, TriComponentIdList receivers,

Value sendMessage);

public void tciCallConnected(TriPortId sender,

TriComponentId receiver,

TriSignatureId signature,

TciParameterList parameterList) ;

public void tciCallConnectedBC(TriPortId sender,

TriSignatureId signature,

TciParameterList parameterList) ;

public void tciCallConnectedMC(TriPortId sender,

TriComponentIdList receivers,

TriSignatureId signature,

TciParameterList parameterList) ;

public void tciReplyConnected(TriPortId sender,

TriComponentId receiver,

TriSignatureId signature,

TciParameterList parameterList,

Value returnValue) ;

public void tciReplyConnectedBC(TriPortId sender,

TriSignatureId signature,

TciParameterList parameterList,

Value returnValue) ;

public void tciReplyConnectedMC(TriPortId sender,

TriComponentIdList receivers,

TriSignatureId signature,

TciParameterList parameterList,

Value returnValue) ;

public void tciRaiseConnected(TriPortId sender,

TriComponentId receiver,

TriSignatureId signature,

Value except) ;

public void tciRaiseConnectedBC(TriPortId sender,

TriSignatureId signature,

Value except) ;

public void tciRaiseConnectedMC(TriPortId sender,

TriComponentIdList receivers,

TriSignatureId signature,

Value except) ;

public TriComponentId tciCreateTestComponentReq(int kind,

Type componentType,

String name,

Value hostId) ;

public void tciStartTestComponentReq(TriComponentId comp,

TciBehaviourId behaviour,

TciParameterList parameterList) ;

public void tciStopTestComponentReq(TriComponentId comp) ;

public void tciConnectReq(TriPortId fromPort, TriPortId toPort) ;

public void tciDisconnectReq(TriPortId fromPort, TriPortId toPort) ;

public void tciTestComponentTerminatedReq(TriComponentId comp, VerdictValue verdict) ;

public boolean tciTestComponentRunningReq(TriComponentId comp) ;

public TriComponentId tciGetMTCReq() ;

public void tciMapReq(TriPortId fromPort, TriPortId toPort);

public void tciMapParamReq(TriPortId fromPort, TriPortId toPort,

TciParameterList parameterList);

public void tciUnmapReq(TriPortId fromPort, TriPortId toPort);

public void tciUnmapParamReq(TriPortId fromPort, TriPortId toPort,

TciParameterList parameterList);

public void tciExecuteTestCaseReq(TriComponentId testComponentId,

TriPortIdList tsiPortList);

public void tciResetReq() ;

public boolean tciTestComponentDoneReq(TriComponentId testComponentId, VerdictValue verdict) ;

public void tciKillTestComponentReq(TriComponentId component);

public boolean tciTestComponentAliveReq (TriComponentId component);

public boolean tciTestComponentKilledReq (TriComponentId component, VerdictValue verdict);

public void tciCallTestComponentReq (TriComponentId comp,

TciBehaviourId behaviour,

TciParameterList parameterList) ;

public void tciTestComponentCallTerminatedReq (TriComponentId comp,

VerdictValue verdict,

TciParameterList parameterList,

Value returnValue) ;

}

#### 8.5.3.2 TCI‑CH required

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

// TciCHRequired

// CH ‑> TE

package org.etsi.ttcn.tci;

public interface TciCHRequired extends TciCDRequired {

public void tciEnqueueMsgConnected(TriPortId sender,

TriComponentId receiver,

Value receivedMessage) ;

public void tciEnqueueCallConnected(TriPortId sender,

TriComponentId receiver,

TriSignatureId signature,

TciParameterList parameterList) ;

public void tciEnqueueReplyConnected(TriPortId sender,

TriComponentId receiver,

TriSignatureId signature,

TciParameterList parameterList,

Value returnValue) ;

public void tciEnqueueRaiseConnected(TriPortId sender,

TriComponentId receiver,

TriSignatureId signature,

Value except) ;

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

public void tciStartTestComponent(TriComponentId comp,

TciBehaviourId behaviour,

TciParameterList parameterList) ;

public void tciStopTestComponent(TriComponentId comp) ;

public void tciConnect(TriPortId fromPort, TriPortId toPort) ;

public void tciDisconnect(TriPortId fromPort, TriPortId toPort) ;

public void tciTestComponentTerminated(TriComponentId comp, VerdictValue verdict);

public boolean tciTestComponentRunning(TriComponentId comp);

public boolean tciTestComponentDone(TriComponentId comp, VerdictValue verdict);

public TriComponentId tciGetMTC ();

public void tciExecuteTestCase (TciTestCaseId TestCaseId, TriPortIdList tsiPortList);

public void tciReset ();

public void tciMap (TriPortId fromPort, TriPortId toPort);

public void tciMapParam (TriPortId fromPort, TriPortId toPort,

TciParameterList parameterList);

public void tciUnmap (TriPortId fromPort, TriPortId toPort);

public void tciUnmapParam (TriPortId fromPort, TriPortId toPort,

TciParameterList parameterList);

public void tciKillTestComponent(TriComponentId component);

public boolean tciTestComponentAlive (TriComponentId component);

public boolean tciTestComponentKilled (TriComponentId component, VerdictValue verdict);

public void tciCallTestComponent (TriComponentId comp,

TciBehaviourId behaviour,

TciParameterList parameterList) ;

public void tciTestComponentCallTerminated (TriComponentId comp,

VerdictValue verdict,

TciParameterList parameterList,

Value returnValue) ;

}

### 8.5.4 The TCI‑TL interface

#### 8.5.4.1 TCI‑TL provided

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

// TCI‑TL

// TE, TM,CH,CD, SA,PA ‑> TL

package org.etsi.ttcn.tci;

public interface TciTLProvided {

public void tliTcExecute(String am, int ts, String src, int line, TriComponentId c,

TciTestCaseId tcId, TciParameterList tciPars, TriTimerDuration dur);

public void tliTcStart(String am, int ts, String src, int line, TriComponentId c,

TciTestCaseId tcId, TciParameterList tciPars, TriTimerDuration dur);

public void tliTcStop(String am, int ts, String src, int line, TriComponentId c, String reason);

public void tliTcStarted(String am, int ts, String src, int line, TriComponentId c,

TciTestCaseId tcId, TciParameterList tciPars, TriTimerDuration dur);

public void tliTcTerminated(String am, int ts, String src, int line, TriComponentId c,

TciTestCaseId tcId, TciParameterList tciPars, VerdictValue verdict, String reason);

public void tliCtrlStart(String am, int ts, String src, int line, TriComponentId c);

public void tliCtrlStop(String am, int ts, String src, int line, TriComponentId c);

public void tliCtrlTerminated(String am, int ts, String src, int line, TriComponentId c);

public void tliMSend\_m(String am, int ts, String src, int line, TriComponentId c,

TriPortId at, TriPortId to, Value msgValue, Value addrValue,

TciStatus encoderFailure, TriMessage msg, TriAddress address,  
 TriStatus transmissionFailure);

public void tliMSend\_m\_BC(String am, int ts, String src, int line, TriComponentId c,

TriPortId at, TriPortId to, Value msgValue,

TciStatus encoderFailure, TriMessage msg, TriStatus transmissionFailure) ;

public void tliMSend\_m\_MC(String am, int ts, String src, int line, TriComponentId c,

TriPortId at, TriPortId to, Value msgValue, TciValueList addrValues,

TciStatus encoderFailure, TriMessage msg, TriAddressList addresses,

TriStatus transmissionFailure);

public void tliMSend\_c(String am, int ts, String src, int line, TriComponentId c,

TriPortId at, TriPortId to, Value msgValue, TriStatus transmissionFailure);

public void tliMSend\_c\_BC(String am, int ts, String src, int line, TriComponentId c,

TriPortId at, TriPortIdList to, Value msgValue, TriStatus transmissionFailure);

public void tliMSend\_c\_MC(String am, int ts, String src, int line, TriComponentId c,

TriPortId at, TriPortIdList to, Value msgValue, TriStatus transmissionFailure);

public void tliMDetected\_m(String am, int ts, String src, int line, TriComponentId c,

TriPortId at, TriPortId from, TriMessage msg, TriAddress address);

public void tliMDetected\_c(String am, int ts, String src, int line, TriComponentId c,

TriPortId at, TriPortId from, Value msgValue );

public void tliMMismatch\_m(String am, int ts, String src, int line, TriComponentId c,

TriPortId at, Value msgValue, TciValueTemplate msgTmpl, TciValueDifferenceList diffs,

Value addrValue, TciValueTemplate addressTmpl);

public void tliMMismatch\_c(String am, int ts, String src, int line, TriComponentId c,

TriPortId at, Value msgValue, TciValueTemplate msgTmpl, TciValueDifferenceList diffs,

TriComponentId from, TciNonValueTemplate fromTmpl);

public void tliMReceive\_m(String am, int ts, String src, int line, TriComponentId c,

TriPortId at, Value msgValue, TciValueTemplate msgTmpl,

Value addrValue, TciValueTemplate addressTmpl);

public void tliMReceive\_c(String am, int ts, String src, int line, TriComponentId c,

TriPortId at, Value msgValue, TciValueTemplate msgTmpl,

TriComponentId from, TciNonValueTemplate fromTmpl);

public void tliPrCall\_m(String am, int ts, String src, int line, TriComponentId c,

TriPortId at, TriPortId to, TriSignatureId signature, TciParameterList tciPars,

Value addrValue, TciStatus encoderFailure, TriParameterList triPars,

TriAddress address, TriStatus transmissionFailure);

public void tliPrCall\_m\_BC(String am, int ts, String src, int line, TriComponentId c,

TriPortId at, TriPortId to, TriSignatureId signature, TciParameterList tciPars,

TciStatus encoderFailure, TriParameterList triPars,

TriStatus transmissionFailure);

public void tliPrCall\_m\_MC(String am, int ts, String src, int line, TriComponentId c,

TriPortId at, TriPortId to, TriSignatureId signature, TciParameterList tciPars,

TciValueList addrValues, TciStatus encoderFailure, TriParameterList triPars,

TriAddressList addresses, TriStatus transmissionFailure);

public void tliPrCall\_c(String am, int ts, String src, int line, TriComponentId c,

TriPortId at, TriPortId to, TriSignatureId signature, TciParameterList tciPars,

TriStatus transmissionFailure);

public void tliPrCall\_c\_BC(String am, int ts, String src, int line, TriComponentId c,

TriPortId at, TriPortIdList to, TriSignatureId signature, TciParameterList tciPars,

TriStatus transmissionFailure);

public void tliPrCall\_c\_MC(String am, int ts, String src, int line, TriComponentId c,

TriPortId at, TriPortIdList to, TriSignatureId signature, TciParameterList tciPars,

TriStatus transmissionFailure);

public void tliPrGetCallDetected\_m(String am, int ts, String src, int line, TriComponentId c,  
 TriPortId at, TriPortId from, TriSignatureId signature, TriParameterList triPars,   
 TriAddress address);

public void tliPrGetCallDetected\_c(String am, int ts, String src, int line, TriComponentId c,

TriPortId at, TriPortId from, TriSignatureId signature, TciParameterList tciPars );

public void tliPrGetCallMismatch\_m(String am, int ts, String src, int line, TriComponentId c,

TriPortId at, TriSignatureId signature,

TciParameterList tciPars, TciValueTemplate parsTmpl, TciValueDifferenceList diffs,

Value addrValue, TciValueTemplate addressTmpl);

public void tliPrGetCallMismatch\_c(String am, int ts, String src, int line, TriComponentId c,

TriPortId at, TriSignatureId signature,

TciParameterList tciPars, TciValueTemplate parsTmpl, TciValueDifferenceList diffs,

TriComponentId from, TciNonValueTemplate fromTmpl);

public void tliPrGetCall\_m(String am, int ts, String src, int line, TriComponentId c,

TriPortId at, TriSignatureId signature,

TciParameterList tciPars, TciValueTemplate parsTmpl,

Value addrValue, TciValueTemplate addressTmpl);

public void tliPrGetCall\_c(String am, int ts, String src, int line, TriComponentId c,

TriPortId at, TriSignatureId signature,

TciParameterList tciPars, TciValueTemplate parsTmpl,

TriComponentId from, TciNonValueTemplate fromTmpl);

public void tliPrReply\_m(String am, int ts, String src, int line, TriComponentId c,

TriPortId at, TriPortId to, TriSignatureId signature, TciParameterList tciPars,

Value replValue, Value addrValue,

TciStatus encoderFailure, TriParameterList triPars,

TriParameter repl, TriAddress address, TriStatus transmissionFailure);

public void tliPrReply\_m\_BC(String am, int ts, String src, int line, TriComponentId c,

TriPortId at, TriPortId to, TriSignatureId signature, TciParameterList tciPars,

Value replValue, TciStatus encoderFailure,

TriParameterList triPars, TriParameter repl, TriStatus transmissionFailure);

public void tliPrReply\_m\_MC(String am, int ts, String src, int line, TriComponentId c,

TriPortId at, TriPortId to, TriSignatureId signature, TciParameterList tciPars,

Value replValue, TciValueList addrValues,

TciStatus encoderFailure, TriParameterList triPars,

TriParameter repl, TriAddressList addresses, TriStatus transmissionFailure);

public void tliPrReply\_c(String am, int ts, String src, int line, TriComponentId c,

TriPortId at, TriPortId to, TriSignatureId signature,

TciParameterList tciPars, Value replValue,

TriStatus transmissionFailure);

public void tliPrReply\_c\_BC(String am, int ts, String src, int line, TriComponentId c,

TriPortId at, TriPortIdList to, TriSignatureId signature, TciParameterList tciPars,

Value replValue, TriStatus transmissionFailure);

public void tliPrReply\_c\_MC(String am, int ts, String src, int line, TriComponentId c,

TriPortId at, TriPortIdList to, TriSignatureId signature, TciParameterList tciPars,

Value replValue, TriStatus transmissionFailure);

public void tliPrGetReplyDetected\_m(String am, int ts, String src, int line, TriComponentId c,

TriPortId at, TriPortId from, TriSignatureId signature, TriParameterListType triPars,

TriParameter repl, TriAddress address);

public void tliPrGetReplyDetected\_c(String am, int ts, String src, int line, TriComponentId c,

TriPortId at, TriPortId from, TriSignatureId signature, TciParameterList tciPars,

Value replValue);

public void tliPrGetReplyMismatch\_m(String am, int ts, String src, int line, TriComponentId c,

TriPortId at, TriSignatureId signature,

TciParameterList tciPars, TciValueTemplate parsTmpl,

Value replValue, TciValueTemplate replyTmpl, TciValueDifferenceList diffs,

Value addrValue, TciValueTemplate addressTmpl);

public void tliPrGetReplyMismatch\_c(String am, int ts, String src, int line, TriComponentId c,

TriPortId at, TriSignatureId signature,

TciParameterList tciPars, TciValueTemplate parsTmpl,

Value replValue, TciValueTemplate replyTmpl, TciValueDifferenceList diffs,

TriComponentId from, TciNonValueTemplate fromTmpl);

public void tliPrGetReply\_m(String am, int ts, String src, int line, TriComponentId c,

TriPortId at, TriSignatureId signature,

TciParameterList tciPars, TciValueTemplate parsTmpl,

Value replValue, TciValueTemplate replyTmpl,

Value addrValue, TciValueTemplate addressTmpl);

public void tliPrGetReply\_c(String am, int ts, String src, int line, TriComponentId c,

TriPortId at, TriSignatureId signature,

TciParameterList tciPars, TciValueTemplate parsTmpl,

Value replValue, TciValueTemplate replyTmpl,

TriComponentId from, TciNonValueTemplate fromTmpl);

public void tliPrRaise\_m(String am, int ts, String src, int line, TriComponentId c,

TriPortId at, TriPortId to,

TriSignatureId signature, TciParameterList tciPars, Value excValue,

Value addrValue, TciStatus encoderFailure, TriException exc,

TriAddress address, TriStatus transmissionFailure);

public void tliPrRaise\_m\_BC(String am, int ts, String src, int line, TriComponentId c,

TriPortId at, TriPortId to,

TriSignatureId signature, TciParameterList tciPars, Value excValue,

TciStatus encoderFailure, TriException exc, TriStatus transmissionFailure) ;

public void tliPrRaise\_m\_MC(String am, int ts, String src, int line, TriComponentId c,

TriPortId at, TriPortId to,

TriSignatureId signature, TciParameterList tciPars, Value excValue,

TciValueList addrValues, TciStatus encoderFailure, TriException exc,

TriAddressList addresses, TriStatus transmissionFailure);

public void tliPrRaise\_c(String am, int ts, String src, int line, TriComponentId c,

TriPortId at, TriPortId to, TriSignatureId signature,

TciParameterList tciPars, Value excValue, TriStatus transmissionFailure);

public void tliPrRaise\_c\_BC(String am, int ts, String src, int line, TriComponentId c,

TriPortId at, TriPortIdList to, TriSignatureId signature, TciParameterList tciPars,

Value excValue, TriStatus transmissionFailure);

public void tliPrRaise\_c\_MC(String am, int ts, String src, int line, TriComponentId c,

TriPortId at, TriPortIdList to, TriSignatureId signature, TciParameterList tciPars,

Value excValue, TriStatus transmissionFailure);

public void tliPrCatchDetected\_m(String am, int ts, String src, int line, TriComponentId c,

TriPortId at, TriPortId from, TriSignatureId signature,

TriException exc, TriAddress address);

public void tliPrCatchDetected\_c(String am, int ts, String src, int line, TriComponentId c,

TriPortId at, TriPortId from, TriSignatureId signature,

Value excValue);

public void tliPrCatchMismatch\_m(String am, int ts, String src, int line, TriComponentId c,

TriPortId at, TriSignatureId signature,

Value excValue, TciValueTemplate excTmpl, TciValueDifferenceList diffs,

Value addrValue, TciValueTemplate addressTmpl);

public void tliPrCatchMismatch\_c(String am, int ts, String src, int line, TriComponentId c,

TriPortId at, TriSignatureId signature,

Value excValue, TciValueTemplate excTmpl, TciValueDifferenceList diffs,

TriComponentId from, TciNonValueTemplate fromTmpl);

public void tliPrCatch\_m(String am, int ts, String src, int line, TriComponentId c,

TriPortId at, TriSignatureId signature,

Value excValue, TciValueTemplate excTmpl,

Value addrValue, TciValueTemplate addressTmpl);

public void tliPrCatch\_c(String am, int ts, String src, int line, TriComponentId c,

TriPortId at, TriSignatureId signature,

Value excValue, TciValueTemplate excTmpl,

TriComponentId from, TciNonValueTemplate fromTmpl);

public void tliPrCatchTimeoutDetected(String am, int ts, String src, int line, TriComponentId c,

TriPortId at, TriSignatureId signature);

public void tliPrCatchTimeout(String am, int ts, String src, int line, TriComponentId c,

TriPortId at, TriSignatureId signature);

public void tliCCreate(String am, int ts, String src, int line, TriComponentId c,

TriComponentId comp, String name, Boolean alive);

public void tliCStart(String am, int ts, String src, int line, TriComponentId c,

TriComponentId comp, TciBehaviourId name, TciParameterList tciPars);

public void tliCRunning(String am, int ts, String src, int line, TriComponentId c,

TriComponentId comp, ComponentStatus status);

public void tliCAlive(String am, int ts, String src, int line, TriComponentId c,

TriComponentId comp, ComponentStatus status);

public void tliCStop(String am, int ts, String src, int line, TriComponentId c,

TriComponentId comp);

public void tliCKill(String am, int ts, String src, int line, TriComponentId c,

TriComponentId comp);

public void tliCDoneMismatch(String am, int ts, String src, int line, TriComponentId c,

TriComponentId comp, TciNonValueTemplate compTmpl);

public void tliCDone(String am, int ts, String src, int line, TriComponentId c,

TciNonValueTemplate compTmpl, VerdictValue verdict);

public void tliCKilledMismatch(String am, int ts, String src, int line, TriComponentId c,

TriComponentId comp, TciNonValueTemplate compTmpl);

public void tliCKilled(String am, int ts, String src, int line, TriComponentId c,

TciNonValueTemplate compTmpl, VerdictValue verdict);

public void tliCTerminated(String am, int ts, String src, int line, TriComponentId c,

VerdictValue verdict, String reason);

public void tliPConnect(String am, int ts, String src, int line, TriComponentId c,

TriPortId port1, TriPortId port2);

public void tliPDisconnect(String am, int ts, String src, int line, TriComponentId c,

TriPortId port1, TriPortId port2);

public void tliPMap(String am, int ts, String src, int line, TriComponentId c,

TriPortId port1, TriPortId port2);

public void tliPMapParam(String am, int ts, String src, int line, TriComponentId c,

TriPortId port1, TriPortId port2, TciParameterList tciPars,

TciStatus encoderFailure, TriParameterList triPars);

public void tliPUnmap(String am, int ts, String src, int line, TriComponentId c,

TriPortId port1, TriPortId port2);

public void tliPUnmapParam(String am, int ts, String src, int line, TriComponentId c,

TriPortId port1, TriPortId port2, TciParameterList tciPars,

TciStatus encoderFailure, TriParameterList triPars);

public void tliPClear(String am, int ts, String src, int line, TriComponentId c,

TriPortId port);

public void tliPStart(String am, int ts, String src, int line, TriComponentId c,

TriPortId port);

public void tliPStop(String am, int ts, String src, int line, TriComponentId c,

TriPortId port);

public void tliPHalt(String am, int ts, String src, int line, TriComponentId c,

TriPortId port);

public void tliEncode(String am, int ts, String src, int line, TriComponentId c,

Value val, TciStatus encoderFailure, TriMessage msg, String codec);

public void tliDecode(String am, int ts, String src, int line, TriComponentId c,

TriMessage msg, TciStatus decoderFailure, Value val, String codec);

public void tliTTimeoutDetected(String am, int ts, String src, int line, TriComponentId c,

TriTimerId timer);

public void tliTTimeoutMismatch(String am, int ts, String src, int line, TriComponentId c,

TriTimerId timer, TciNonValueTemplate timerTmpl);

public void tliTTimeout(String am, int ts, String src, int line, TriComponentId c,

TriTimerId timer, TciNonValueTemplate timerTmpl);

public void tliTStart(String am, int ts, String src, int line, TriComponentId c,

TriTimerId timer, TriTimerDuration dur);

public void tliTStop(String am, int ts, String src, int line, TriComponentId c,

TriTimerId timer, TriTimerDuration dur);

public void tliTRead(String am, int ts, String src, int line, TriComponentId c,

TriTimerId timer, TriTimerDuration elapsed);

public void tliTRunning(String am, int ts, String src, int line, TriComponentId c,

TriTimerId timer, TimerStatus status);

public void tliSEnter(String am, int ts, String src, int line, TriComponentId c,

QualifiedName name, TciParameterList tciPars, String kind);

public void tliSLeave(String am, int ts, String src, int line, TriComponentId c,

QualifiedName name, TciParameterList tciPars, Value returnValue, String kind);

public void tliVar(String am, int ts, String src, int line, TriComponentId c,

QualifiedName name, Value varValue);

public void tliModulePar(String am, int ts, String src, int line, TriComponentId c,

QualifiedName name, Value parValue);

public void tliGetVerdict(String am, int ts, String src, int line, TriComponentId c,

VerdictValue verdict);

public void tliSetVerdict(String am, int ts, String src, int line, TriComponentId c,

VerdictValue verdict, String reason);

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

String log);

public void tliAEnter(String am, int ts, String src, int line, TriComponentId c);

public void tliALeave(String am, int ts, String src, int line, TriComponentId c);

public void tliADefaults(String am, int ts, String src, int line, TriComponentId c);

public void tliAActivate(String am, int ts, String src, int line, TriComponentId c,

QualifiedName name, TciParameterList tciPars, Value ref);

public void tliADeactivate(String am, int ts, String src, int line, TriComponentId c,

Value ref);

public void tliANomatch(String am, int ts, String src, int line, TriComponentId c);

public void tliARepeat(String am, int ts, String src, int line, TriComponentId c);

public void tliAWait(String am, int ts, String src, int line, TriComponentId c);

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

public void tliMatch(String am, int ts, String src, int line, TriComponentId c, Value expr,

TciValueTemplate tmpl);

public void tliMatchMismatch(String am, int ts, String src, int line, TriComponentId c,

Value expr, TciValueTemplate tmpl, TciValueDifferenceList diffs);

public void tliInfo (String am, int ts, String src, int line, TriComponentId c,

int level, String info)

public void tliMChecked\_m(String am, int ts, String src, int line, TriComponentId c,

TriPortId at, Value msgValue, TciValueTemplate msgTmpl,

Value addrValue, TciValueTemplate addressTmpl);

public void tliMChecked\_c(String am, int ts, String src, int line, TriComponentId c,

TriPortId at, Value msgValue, TciValueTemplate msgTmpl,

TriComponentId from, TciNonValueTemplate fromTmpl);

public void tliPrGetCallChecked\_m(String am, int ts, String src, int line, TriComponentId c,

TriPortId at, TriSignatureId signature, TciParameterList tciPars,

TciValueTemplate parsTmpl, Value addrValue, TciValueTemplate addressTmpl);

public void tliPrGetCallChecked\_c(String am, int ts, String src, int line, TriComponentId c,

TriPortId at, TriSignatureId signature, TciParameterList tciPars,

TciValueTemplate parsTmpl, TriComponentId from, TciNonValueTemplate fromTmpl);

public void tliPrGetReplyChecked\_m(String am, int ts, String src, int line, TriComponentId c,

TriPortId at, TriSignatureId signature,

TciParameterList tciPars, TciValueTemplate parsTmpl,

Value replValue, TciValueTemplate replyTmpl,

Value addrValue, TciValueTemplate addressTmpl);

public void tliPrGetReplyChecked\_c(String am, int ts, String src, int line, TriComponentId c,

TriPortId at, TriSignatureId signature,

TciParameterList tciPars, TciValueTemplate parsTmpl,

Value replValue, TciValueTemplate replyTmpl,

TriComponentId from, TciNonValueTemplate fromTmpl);

public void tliPrCatchChecked\_m(String am, int ts, String src, int line, TriComponentId c,

TriPortId at, TriSignatureId signature,

Value excValue, TciValueTemplate excTmpl,

Value addrValue, TciValueTemplate addressTmpl);

public void tliPrCatchChecked\_c(String am, int ts, String src, int line, TriComponentId c,

TriPortId at, TriSignatureId signature,

Value excValue, TciValueTemplate excTmpl,

TriComponentId from, TciNonValueTemplate fromTmpl);

public void tliCheckedAny\_m(String am, int ts, String src, int line, TriComponentId c,

TriPortId at, Value addrValue, TciValueTemplate addressTmpl);

public void tliCheckedAny\_c(String am, int ts, String src, int line, TriComponentId c,

TriPortId at, TriComponentId from, TciNonValueTemplate fromTmpl);

public void tliCheckAnyMismatch\_m(String am, int ts, String src, int line,

TriComponentId c, TriPortId at, Value addrValue, TciValueTemplate addressTmpl);

public void tliCheckAnyMismatch\_c(String am, int ts, String src, int line,

TriComponentId c, TriPortId at, TriComponentId from, TciNonValueTemplate fromTmpl);

public void tliRnd(String am, int ts, String src, int line,

TriComponentId c, FloatValue val, FloatValue seed);

public void tliEvaluate(String am, int ts, String src, int line, TriComponentId c,

QualifiedName name, Value evalResult);

public void tliCCall (String am, int ts, String src, int line, TriComponentId c,

TriComponentId comp, TciBehaviourId name, TciParameterList tciPars);

public void tliCCallTerminated (String am, int ts, String src, int line, TriComponentId c,

VerdictValue verdict, String reason, TciParameterList tciPars, Value returnValue);

public void tliCtrlStartWithParameters (String am, int ts, String src, int line,   
 TriComponentId c, TciParameterList tciPars);

public void tliCtrlTerminatedWithResult (String am, int ts, String src, int line,  
 TriComponentId c, Value result, TciParameterList tciPars);

}

## 8.6 Optional parameters

Clause 7.2.2 defines that a reserved value shall be used to indicate the absence of an optional parameter. For the Java™ language mapping the Java™ null value shall be used to indicate the absence of an optional value. For example, if in the tciReplyConnected operation the value parameter shall be omitted the operation invocation shall be tciReplyConnected (sender, receiver, signature, parameterList, null).

## 8.7 TCI initialization

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

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

## 8.8 Error handling

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

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

# 9 ANSI C language mapping

## 9.1 Introduction

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

## 9.2 Value interfaces

Table

| TCI IDL Interface | ANSI C representation | Notes and comments |
| --- | --- | --- |
| Type | | |
| TciModuleIdType getDefiningModule() | TciModuleIdType  tciGetDefiningModule(Type inst) |  |
| Tstring getName() | String tciGetName(Type inst) | String type reused from IDL (OMG recommendation) |
| TciTypeClassType getTypeClass() | TciTypeClassType tciGetTypeClass  (Type inst) |  |
| Value newInstance() | Value tciNewInstance(Type inst) |  |
| Value newTemplate (TciMatchingType matchingType) | Value tciNewTemplate(Type inst, TciMatchingTypeType matchingType) |  |
| TString getTypeEncoding() | String tciGetTypeEncoding(Type inst) |  |
| TStringSeq getTypeExtension() | String\* tciGetTypeExtension(Type inst) | Returns null pointer or a null‑pointer terminated array |
| TString getTypeEncodingVariant() | String tciGetTypeEncodingVariant(Type inst) |  |
| TString getEncodeAttributes () | String\* tciGetTypeEncodeAttributes(Type inst) | Returns null pointer or a null‑pointer terminated array |
| TString getVariantAttributes (TString encoding) | String\* tciGetTypeVariantAttributes(Type inst, String encoding) | Returns null pointer or a null‑pointer terminated array |
| Value parseValue(TString val) | Value tciParseValue(Type inst, String val) |  |
| LengthRestriction getTypeLengthRestriction() | int tciGetTypeLengthRestriction  (Type inst, TciLengthRestriction \* restriction) | Returns 0 for no restriction and -1 for restriction. The restriction is returned in the second parameter |
| RangeBoundary getLowerTypeBoundary() | int tciGetLowerTypeBoundary  (Type inst, TciRangeBoundary \* boundary) | Returns 0 for no boundary and -1 if boundary is present. The boundary is returned in the second parameter |
| RangeBoundary geUpperTypeBoundary() | int tciGetUpperTypeBoundary  (Type inst, TciRangeBoundary \* boundary) |
| MatchingMechanism getTypeMatchingMechanism() | Value tciGetTypeMatchingMechanism  (Type inst) | Get the restriction of type of value |
| Value | | |
| TString getValueEncoding() | String tciGetValueEncoding(Value inst) |  |
| TString getValueEncodingVariant() | String tciGetValueEncodingVariant(Value inst) |  |
| TString getEncodeAttributes () | String\* tciGetValueEncodeAttributes(Type inst) | Returns null pointer or a null‑pointer terminated array |
| TString getVariantAttributes (TString encoding) | String\* tciGetValueVariantAttributes(Type inst, String encoding) | Returns null pointer or a null‑pointer terminated array |
| Type getType() | Type tciGetType(Value inst) |  |
| Tboolean notPresent() | Boolean tciNotPresent(Value inst) | Boolean type reused from IDL (OMG recommendation) |
|  | void tciSetNull(Value inst) | For optional parameters of operations, see clause 9.7 |
|  | Boolean tciIsNull(Value inst) | For optional parameters of operations, see clause 9.7.  Boolean type reused from IDL (OMG recommendation) |
| Tboolean isMatchingSymbol() | Boolean tciIsMatchingSymbol(Value inst) |  |
| TString valueToString() | String tciValueToString(Value inst) |  |
| Tboolean isLazy () | Boolean tciIsLazy(Value inst) |  |
| Tboolean isFuzzy () | Boolean tciIsFuzzy(Value inst) |  |
| Tboolean isEvaluated() | Boolean tciIsEvaluated(Value inst) |  |
|  | Boolean tciHasLengthRestriction(Value inst) | Returns true if length restriction is present, false otherwise |
| LengthRestriction getLengthRestriction() | TciLengthRestriction tciGetLengthRestriction(Value inst) |  |
|  | void tciRemoveLengthRestriction(Value inst) | Removes length restriction |
| void setLengthRestriction (LengthRestriction restriction) | void tciSetLengthRestriction(Value inst, TciLengthRestriction restriction) |  |
| TBoolean isIfPresentEnabled() | Boolean tciIsIfPresentEnabled(Value inst) |  |
| void setIfPresentEnabled(TBoolean enabled) | void tciSetIfPresentEnabled(Value inst, Boolean enabled) |  |
| LengthRestriction getTypeLengthRestriction() | int  tciGetValueLengthRestriction (Value inst, TciLengthRestriction \* restriction) | Returns 0 for no restriction and -1 for restriction. The restriction is returned in the second parameter |
| RangeBoundary getLowerTypeBoundary() | int  tciGetLowerValueBoundary  (Value inst, TciRangeBoundary \* boundary) | Returns 0 for no boundary and -1 if boundary is present. The boundary is returned in the second parameter |
| RangeBoundary geUpperTypeBoundary() | int tciGetUpperValueBoundary  (Value inst, TciRangeBoundary \* boundary) |
| MatchingMechanism getTypeMatchingMechanism() | Value tciGetValueMatchingMechanism (Value inst) | Get the restriction of type of value |
| Tboolean isOptional() | Boolean tciIsOptional(Value inst) |  |
| IntegerValue | | |
| Tinteger getInt() | String tciGetIntAbs(Value inst) | Returns the (10‑base) integer absolute value as an ASCII string |
|  | unsigned long int tciGetIntNumberOfDigits  (Value inst) | Returns the number of digits in an integer value |
|  | Boolean tciGetIntSign(Value inst) | Returns true if the number is positive, false otherwise |
|  | char tciGetIntDigit  (Value inst,  unsigned long int position) | Returns the value of the digit at position 'position', where position 0 is the least significant digit |
|  | long long tciGetInt(Value inst) | Alternative getInt realization for integers not exceeding the boundaries of signed 64-bit |
| void setInt(in Tinteger value) | void tciSetIntAbs(Value inst, String value) | Sets the (10‑base) absolute value of the integer via an ASCII string. The first digit shall not be 0 unless the value is 0 |
|  | Void tciSetIntNumberOfDigits  (Value inst,  unsigned long int dig\_num) | Sets the number of digits in an integer value |
|  | void tciSetIntSign  (Value inst,  Boolean sign) | Sets the sign to + (true) or - (false) |
|  | void tciSetIntDigit  (Value inst,  unsigned long int position,  char digit) | Sets the value of the digit at position 'position', where position 0 is the least significant digit |
|  | void tciSetInt(Value inst, long long value) | Alternative setInt realization for integers not exceeding the boundaries of signed 64-bit |
| FloatValue | | |
| Tfloat getFloat() | double tciGetFloatValue(Value inst) |  |
| void setFloat(in Tfloat value) | void tciSetFloatValue(Value inst, double value) |  |
| BooleanValue | | |
| Tboolean getBoolean() | Boolean tciGetBooleanValue(Value inst) |  |
| void setBoolean (in Tboolean value) | void tciSetBooleanValue  (Value inst, Boolean value) |  |
| CharstringValue | | |
| Tstring getString() | TciCharStringValue tciGetCStringValue(Value inst) |  |
| void setString(in Tstring value) | void tciSetCStringValue  (Value inst, TciCharStringValue value) |  |
| Tchar getChar (in Tinteger position) | char tciGetCStringCharValue  (Value inst, long int position) |  |
| void setChar  (in Tinteger position, in Tchar value) | void tciSetCStringCharValue  (Value inst,  long int position,  char value) |  |
| Tinteger getLength() | unsigned long int tciGetCStringLength (Value inst) |  |
| void setLength(in Tinteger len) | void tciSetCStringLength  (Value inst,  unsigned long int len) |  |
| UniversalCharstringValue | | |
| Tstring getString() | TciUCStringValue tciGetUCStringValue(Value inst) |  |
| void setString(in Tstring value) | void tciSetUCStringValue  (Value inst, TciUCStringValue value) |  |
| TuniversalChar getChar  (in Tinteger position) | void tciGetUCStringCharValue  (Value inst, unsigned long int position,  TciUCValue result) |  |
| void setChar  (in Tinteger position,  in TuniversalChar value) | void tciSetUCStringCharValue  (Value inst,  unsigned long int position,  TciUCValue value) |  |
| Tinteger getLength() | unsigned long int tciGetUCStringLength(Value inst) |  |
| void setLength(in Tinteger len) | void tciSetUCStringLength  (Value inst,  unsigned long int len) |  |
| BitstringValue | | |
| Tstring getString() | String tciGetBStringValue(Value inst) |  |
| void setString(in Tstring value) | void tciSetBStringValue  (Value inst, String value) |  |
| Tchar getBit (in integer position) | int tciGetBStringBitValue  (Value inst, long int position) |  |
| void setBit  (in Tinteger position,  in Tinteger value) | void tciSetBStringBitValue  (Value inst,  unsigned long int position,  int value) |  |
| Tinteger getLength() | unsigned long int tciGetBStringLength(Value inst) |  |
| void setLength(in Tinteger len) | void tciSetBStringLength  (Value inst,  long int len) |  |
| TBoolean isMatchingAt(in TInteger position) | Boolean tciIsBStringMatchingAt(Value inst, unsigned long int position) |  |
| MatchingMechanism getMatchingAt(in TInteger position) | Value tciGetBStringMatchingAt(Value inst, unsigned long int position) |  |
| void setMatching(in TInteger position, in MatchingMechanism template) | void tciSetBStringMatchingAt(Value inst, unsigned long int position, Value template) |  |
| HexstringValue | | |
| Tstring getString() | String tciGetHStringValue(Value inst) |  |
| void setString(in Tstring value) | void tciSetHStringValue  (Value inst, String value) |  |
| Tchar getHex (in Tinteger position) | int tciGetHStringHexValue  (Value inst, unsigned long int position) |  |
| void setBit  (in Tinteger position,  in Tinteger value) | void tciSetHStringHexValue  (Value inst,  unsigned long int position,  int value) |  |
| Tinteger getLength() | long int tciGetHStringLength(Value inst) |  |
| void setLength(in Tinteger len) | void tciSetHStringLength  (Value inst,  unsigned long int len) |  |
| TBoolean isMatchingAt(in TInteger position) | Boolean tciIsHStringMatchingAt(Value inst, unsigned long int position) |  |
| MatchingMechanism getMatchingAt(in TInteger position) | Value tciGetHStringMatchingAt(Value inst, unsigned long int position) |  |
| void setMatching(in TInteger position, in MatchingMechanism template) | void tciSetHStringMatchingAt(Value inst, unsigned long int position, Value template) |  |
| OctetstringValue | | |
| Tstring getString() | String tciGetOStringValue(Value inst) |  |
| void setString(in Tstring value) | void tciSetOStringValue  (Value inst, String value) |  |
| Tchar getOctet(in Tinteger position) | int tciGetOStringOctetValue  (Value inst, unsigned long int position) |  |
| void setOctet  (in Tinteger position,  in Tinteger value) | void tciSetOStringOctetValue  (Value inst,  unsigned long int position,  int value) |  |
| Tinteger getLength() | unsigned long int tciGetOStringLength(Value inst) |  |
| void setLength(in Tinteger len) | void tciSetOStringLength  (Value inst,  unsigned long int len) |  |
| TBoolean isMatchingAt(in TInteger position) | Boolean tciIsOStringMatchingAt(Value inst, unsigned long int position) |  |
| MatchingMechanism getMatchingAt(in TInteger position) | Value tciGetOStringMatchingAt(Value inst, unsigned long int position) |  |
| void setMatching(in TInteger position, in MatchingMechanism template) | void tciSetOStringMatchingAt(Value inst, unsigned long int position, Value template) |  |
| RecordValue | | |
| Value getField(in Tstring fieldName int) | Value tciGetRecFieldValue  (Value inst,  String fieldName int) |  |
| void setField  (in Tstring fieldName int,  in Value value) | void tciSetRecFieldValue  (Value inst,  String fieldName int,  Value value) |  |
| Tstring[] getFieldNames() | char\*\* tciGetRecFieldNames(Value inst) | Returns a NULL‑terminated array of the field names |
| void setFieldOmitted  (in Tstring fieldName int) | void setFieldOmitted  (Value inst,  String fieldName int) |  |
| RecordOfValue | | |
| Value getField(in Tinteger position) | Value tciGetRecOfFieldValue  (Value inst,  unsigned long int position) |  |
| void setField  (in Tinteger position,  in Value value) | void tciSetRecOfFieldValue  (Value inst,  unsigned long int position,  Value value) |  |
| void appendField(in Value value) | void tciAppendRecOfFieldValue  (Value inst,  Value value) |  |
| Type getElementType() | Type tciGetRecOfElementType(Value inst) |  |
| Tinteger getLength() | unsigned long int tciGetRecOfLength(Value inst) |  |
| void setLength(in Tinteger len) | void tciSetRecOfLength  (Value inst,  unsigned long int len) |  |
| Tinteger getOffset() | unsigned long int tciGetOffset(Value inst) |  |
| TInteger getPermutationCount() | unsigned long int tciGetPermutationCount(Value inst) |  |
| Permutation getPermutation(TInteger index) | TciPermutation tciGetPermutation(Value inst, unsigned long int index) |  |
| void definePermutation(Permutation permutation) | void tciDefinePermutation(Value inst, TciPermutation permutation) |  |
| void removePermutation(TInteger index) | void tciRemovePermutation(Value intst, unsigned long int index) |  |
| void clearPermutations() | void tciClearPermutations(Value inst) |  |
| UnionValue | | |
| Value getVariant  (in Tstring variantName) | Value tciGetUnionVariant  (Value inst,  String variantName) |  |
| void setVariant  (in Tstring variantName,  in Value value) | void tciSetUnionVariant  (Value inst,  String variantName,  Value value) |  |
| Tstring getPresentVariantName() | String tciGetUnionPresentVariantName  (Value inst) |  |
| Tstring[] getVariantNames() | char\*\* tciGetUnionVariantNames(Value inst) | Returns a NULL‑terminated array of the field names |
| EnumeratedValue | | |
| Tstring getEnum() | String tciGetEnumValue(Value inst) |  |
| void setEnum(in Tstring enumValue) | void tciSetEnumValue  (Value inst,  String enumValue) |  |
| Tinteger getInt() | long tciGetEnumInt(Value inst); |  |
| setInt(in Tinteger intValue) | void tciSetEnumInt(Value inst, long intValue); |  |
| VerdictValue | | |
| Tinteger getVerdict() | int tciGetVerdictValue(Value inst) |  |
| void setVerdict(in Tinteger verdict) | void tciSetVerdictValue(Value inst, int verdict) |  |
| AddressValue | | |
| Value getAddress() | Value tciGetAddressValue(Value inst) |  |
| void setAddress(in Value value) | void tciSetAddressValue(Value inst, Value value) |  |
| **MatchingMechanism** | | |
| TciMatchingTypeType getMatchingType() | TciMatchingTypeType tciGetMatchingType(Value inst) |  |
| **MatchingList** | | |
| TInteger size() | unsigned long int tciGetMatchingListSize(Value inst) |  |
| Value get(TInteger position) | Value tciGetMatchingListItem(Value inst, unsigned long int position) |  |
| void add(Value item) | void tciAddMatchingListItem(Value inst, Value item) |  |
| void remove(TInteger position) | void tciRemoveMatchingListItem(Value inst, unsigned long int position) |  |
| void clear() | void tciClearMatchingList(Value inst) |  |
| **ValueRange** | | |
| RangeBoundary getLowerBoundary() | TciRangeBoundary tciGetLowerRangeBoundary(Value inst) |  |
| RangeBoundary getUpperBoundary() | TciRangeBoundary tciGetUpperRangeBoundary(Value inst) |  |
| void setLowerBoundary (RangeBoundary lowerBoundary) | void tciSetLowerRangeBoundary(Value inst, TciRangeBoundary lowerBoundary) |  |
| void setUpperBoundary (RangeBoundary upperBoundary) | void tciSetUpperRangeBoundary(Value inst, TciRangeBoundary upperBoundary) |  |
| **CharacterPattern** | | |
| Value getPatternString () | Value tciGetPatternString(Value inst) |  |
| void setPatternString(Value characterPattern) | void tciSetPatternString(Value inst, Value characterPattern) |  |
| **MatchDecodedContent** | | |
| Value getContent() | Value tciGetDecodedMatchContent(Value inst) |  |
| void setContent(Value content) | void tciSetDecodedMatchContent(Value inst, Value content) |  |

## 9.3 Logging interface

Table

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

## 9.4 Operation interfaces

### 9.4.1 The TCI‑TM interface

#### 9.4.1.1 TCI‑TM provided

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

void tciTestCaseStarted

(TciTestCaseIdType testCaseId, TciParameterListType parameterList, double timer)

void tciTestCaseTerminated (VerdictValue verdict, TciParameterListType parameterlist)

void tciControlTerminated()

Value tciGetModulePar (TciModuleParameterIdType parameterId)

void tciLog(String message)

void tciError(String message)

void tciControlTerminatedWithResult (Value result, TciParameterListType parameterlist)

#### 9.4.1.2 TCI‑TM required

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

void tciRootModule(String moduleId)

TciModuleIdListType tciGetImportedModules()

TciModuleParameterListType tciGetModuleParameters(TciModuleIdType moduleName)

TciTestCaseIdListType tciGetTestCases()

TciParameterTypeListType tciGetTestCaseParameters (TciTestCaseIdType testCaseId)

TriPortIdList tciGetTestCaseTSI (TciTestCaseIdType testCaseId)

void tciStartTestCase (TciTestCaseIdType testCaseId, TciParameterListType parameterlist)

void tciStopTestCase()

TriComponentId tciStartControl()

void tciStopControl()

TciParameterTypeListType tciGetControlParameters ()

TriComponentId tciStartControlWithParameters (TciParameterListType parameterlist)

### 9.4.2 The TCI‑CD interface

#### 9.4.2.1 TCI‑CD provided

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

Value tciDecode (BinaryString message, Type decHypothesis)

BinaryString tciEncode(Value value)

int tciDecodeValue (BinaryString \* message, Type decHypothesis, String decodingInfo, Value \* decodedValue)

BinaryString tciEncodeValue(Value value, string encodingInfo)

NOTE: BinaryString type reused from TRI.

#### 9.4.2.2 TCI‑CD required

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

Type tciGetTypeForName(String typeName)

Type tciGetIntegerType()

Type tciGetFloatType()

Type tciGetBooleanType()

Type tciGetCharType()

Type tciGetUniversalCharType()

Type tciGetTciCharstringType()

Type tciGetUniversalCharstringType()

Type tciGetHexstringType()

Type tciGetBitstringType()

Type tciGetOctetstringType()

Type tciGetVerdictType()

void tciErrorReq(String message)

### 9.4.3 The TCI‑CH interface

#### 9.4.3.1 TCI‑CH provided

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

void tciSendConnected(TriPortId sender, TriComponentId receiver, Value sendMessage)

void tciSendConnectedBC(TriPortId sender, Value sendMessage)

void tciSendConnectedMC

(TriPortId sender, TriComponentIdList receivers, Value sendMessage)

void tciCallConnected

(TriPortId sender, TriComponentId receiver, TriSignatureId signature,

TciParameterListType parameterList)

void tciCallConnectedBC

(TriPortId sender, TriSignatureId signature, TciParameterListType parameterList)

void tciCallConnectedMC

(TriPortId sender, TriComponentIdList receivers, TriSignatureId signature,

TciParameterListType parameterList)

void tciReplyConnected

(TriPortId sender, TriComponentId receiver, TriSignatureId signature,

TciParameterListType parameterList, Value returnValue)

void tciReplyConnectedBC

(TriPortId sender, TriSignatureId signature, TciParameterListType parameterList,

Value returnValue)

void tciReplyConnectedMC

(TriPortId sender, TriComponentIdList receivers, TriSignatureId signature,

TciParameterListType parameterList, Value returnValue)

void tciRaiseConnected

(TriPortId sender, TriComponentId receiver, TriSignatureId signature, Value exception)

void tciRaiseConnectedBC

(TriPortId sender, TriSignatureId signature, Value exception)

void tciRaiseConnectedMC

(TriPortId sender, TriComponentIdList receivers, TriSignatureId signature, Value exception)

TriComponentId tciCreateTestComponentReq

(TciTestComponentKindType kind, Type componentType, String name, Value hostId)

void tciStartTestComponentReq

(TriComponentId component, TciBehaviourIdType behaviour, TciParameterListType parameterList)

void tciStopTestComponentReq (TriComponentId component)

void tciConnectReq(TriPortId fromPort, TriPortId toPort)

void tciDisconnectReq(TriPortId fromPort, TriPortId toPort)

void tciMapReq(TriPortId fromPort, TriPortId toPort)

void tciMapParamReq

(TriPortId fromPort, TriPortId toPort, TciParameterListType parameterList)

void tciUnmapReq(TriPortId fromPort, TriPortId toPort)

void tciUnmapParamReq

(TriPortId fromPort, TriPortId toPort, TciParameterListType parameterList)

void tciTestComponentTerminatedReq(TriComponentId component, VerdictValue verdict)

Boolean tciTestComponentRunningReq(TriComponentId component)

Boolean tciTestComponentDoneReq(TriComponentId component, int \* verdict)

TriComponentId tciGetMTCReq()

void tciExecuteTestCaseReq(TciTestCaseIdType testCaseId, TriPortIdList tsiPortList)

void tciResetReq()

void tciKillTestComponentReq(TriComponentId component)

Boolean tciTestComponentAliveReq (TriComponentId component)

Boolean tciTestComponentKilledReq (TriComponentId component, int \* verdict)

void tciCallTestComponentReq

(TriComponentId component, TciBehaviourIdType behaviour, TciParameterListType parameterList)

void tciTestComponentCallTerminatedReq

(TriComponentId component, VerdictValue verdict, TciParameterListType parameterList,

Value returnValue)

#### 9.4.3.2 TCI‑CH required

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

void tciEnqueueMsgConnected

(TriPortId sender, TriComponentId receiver, Value rcvdMessage)

void tciEnqueueCallConnected

(TriPortId sender, TriComponentId receiver, TriSignatureId signature,

TciParameterListType parameterList)

void tciEnqueueReplyConnected

(TriPortId sender, TriComponentId receiver, TriSignatureId signature,

TciParameterListType parameterList, Value returnValue)

void tciEnqueueRaiseConnected

(TriPortId sender, TriComponentId receiver, TriSignatureId signature, Value exception)

TriComponentId tciCreateTestComponent

(TciTestComponentKindType kind, Type componentType, String name)

void tciStartTestComponent

(TriComponentId component, TciBehaviourIdType behaviour, TciParameterListType parameterList)

void tciStopTestComponent(TriComponentId component)

void tciConnect(TriPortId fromPort, TriPortId toPort)

void tciDisconnect(TriPortId fromPort, TriPortId toPort)

void tciMap (TriPortId fromPort, TriPortId toPort)

void tciMapParam

(TriPortId fromPort, TriPortId toPort, TciParameterListType parameterList)

void tciUnmap(TriPortId fromPort, TriPortId toPort)

void tciUnmapParam

(TriPortId fromPort, TriPortId toPort, TciParameterListType parameterList)

void tciTestComponentTerminated(TriComponentId component, VerdictValue verdict)

Boolean tciTestComponentRunning(TriComponentId component)

Boolean tciTestComponentDone(TriComponentId component, int \* verdict)

TriComponentId tciGetMTC()

void tciExecuteTestCase(TciTestCaseIdType testCaseId, TriPortIdList tsiPortList)

void tciReset()

void tciKillTestComponent(TriComponentId component)

Boolean tciTestComponentAlive(TriComponentId component)

Boolean tciTestComponentKilled(TriComponentId component, int \* verdict)

void tciCallTestComponent

(TriComponentId component, TciBehaviourIdType behaviour, TciParameterListType parameterList)

void tciTestComponentCallTerminated

(TriComponentId component, VerdictValue verdict, TciParameterListType parameterList,

Value returnValue)

### 9.4.4 The TCI‑TL interface

#### 9.4.4.1 TCI‑TL provided

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

void tliTcExecute

(String am, int ts, String src, int line, TriComponentId c, TciTestCaseIdType tcId,

TciParameterListType tciPars, TriTimerDuration dur)

void tliTcStart

(String am, int ts, String src, int line, TriComponentId c, TciTestCaseIdType tcId,

TciParameterListType tciPars, TriTimerDuration dur)

void tliTcStop

(String am, int ts, String src, int line, TriComponentId c, String reason)

void tliTcStarted

(String am, int ts, String src, int line, TriComponentId c, TciTestCaseIdType tcId,

TciParameterListType tciPars, TriTimerDuration dur)

void tliTcTerminated

(String am, int ts, String src, int line, TriComponentId c, TciTestCaseIdType tcId,

TciParameterListType tciPars, VerdictValue verdict, String reason)

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

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

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

void tliMSend\_m

(String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriPortId to,

Value msgValue, Value addrValue, TciStatus encoderFailure, TriMessage msg,

TriAddress address, TriStatus transmissionFailure)

void tliMSend\_m\_BC

(String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriPortId to,

Value msgValue, TciStatus encoderFailure, TriMessage msg, TriStatus transmissionFailure)

void tliMSend\_m\_MC

(String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriPortId to,

Value msgValue, TciValueList addrValues, TciStatus encoderFailure, TriMessage msg,

TriAddressList addresses, TriStatus transmissionFailure)

void tliMSend\_c

(String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriPortId to,

Value msgValue, TriStatus transmissionFailure)

void tliMSend\_c\_BC

(String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriPortIdList to,

Value msgValue, TriStatus transmissionFailure)

void tliMSend\_c\_MC

(String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriPortIdList to,

Value msgValue, TriStatus transmissionFailure)

void tliMDetected\_m

(String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriPortId from,

TriMessage msg, TriAddress address)

void tliMDetected\_c

(String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriPortId from,

Value msgValue)

void tliMMismatch\_m

(String am, int ts, String src, int line, TriComponentId c, TriPortId at, Value msgValue,

TciValueTemplate msgTmpl, TciValueDifferenceList diffs, Value addrValue,

TciValueTemplate addressTmpl)

void tliMMismatch\_c

(String am, int ts, String src, int line, TriComponentId c, TriPortId at, Value msgValue,

TciValueTemplate msgTmpl, TciValueDifferenceList diffs, TriComponentId from,

TciNonValueTemplate fromTmpl)

void tliMReceive\_m

(String am, int ts, String src, int line, TriComponentId c, TriPortId at, Value msgValue,

TciValueTemplate msgTmpl, Value addrValue, TciValueTemplate addressTmpl)

void tliMReceive\_c

(String am, int ts, String src, int line, TriComponentId c, TriPortId at, Value msgValue,

TciValueTemplate msgTmpl, TriComponentId from, TciNonValueTemplate fromTmpl)

void tliPrCall\_m

(String am, int ts, String src, int line, TriComponentId c, TriPortId at,

TriPortId to, TriSignatureId signature,

TciParameterListType tciPars, Value addrValue, TciStatus encoderFailure,

TriParameterList triPars, TriAddress address, TriStatus transmissionFailure)

void tliPrCall\_m\_BC

(String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriPortId to,

TriSignatureId signature, TciParameterListType tciPars, TciStatus encoderFailure,

TriParameterList triPars, TriStatus transmissionFailure)

void tliPrCall\_m\_MC

(String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriPortId to,

TriSignatureId signature, TciParameterListType tciPars, TciValueList addrValues,

TciStatus encoderFailure, TriParameterList triPars, TriAddressList addresses,

TriStatus transmissionFailure)

void tliPrCall\_c

(String am, int ts, String src, int line, TriComponentId c, TriPortId at,

TriPortId to, TriSignatureId signature,

TciParameterListType tciPars, TriStatus transmissionFailure)

void tliPrCall\_c\_BC

(String am, int ts, String srcint line, TriComponentId c, TriPortId at, TriPortIdList to,

TriSignatureId signature, TciParameterListType tciPars, TriStatus transmissionFailure)

void tliPrCall\_c\_MC

(String am, int ts, String srcint line, TriComponentId c, TriPortId at, TriPortIdList to,

TriSignatureId signature, TciParameterListType tciPars, TriStatus transmissionFailure)

void tliPrGetCallDetected\_m

(String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriPortId from,

TriSignatureId signature, TriParameterList triPars, TriAddress address)

void tliPrGetCallDetected\_c

(String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriPortId from,

TriSignatureId signature, TciParameterListType tciPars)

void tliPrGetCallMismatch\_m

(String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriSignatureId signature,

TciParameterListType tciPars, TciValueTemplate parsTmpl, TciValueDifferenceList diffs,

Value addrValue, TciValueTemplate addressTmpl)

void tliPrGetCallMismatch\_c

(String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriSignatureId signature,

TciParameterListType tciPars, TciValueTemplate parsTmpl, TciValueDifferenceList diffs,

TriComponentId from, TciNonValueTemplate fromTmpl)

void tliPrGetCall\_m

(String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriSignatureId signature,

TciParameterListType tciPars, TciValueTemplate parsTmpl, Value addrValue,

TciValueTemplate addressTmpl)

void tliPrGetCall\_c

(String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriSignatureId signature,

TciParameterListType tciPars, TciValueTemplate parsTmpl, TriComponentId from,

TciNonValueTemplate fromTmpl)

void tliPrReply\_m

(String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriPortId to,

TriSignatureId signature, TciParameterListType tciPars, Value replValue,

Value addrValue, TciStatus encoderFailure, TriParameterList triPars,

TriParameter repl, TriAddress address, TriStatus transmissionFailure)

void tliPrReply\_m\_BC

(String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriPortId to,

TriSignatureId signature, TciParameterListType tciPars, Value replValue,

TciStatus encoderFailure, TriParameterList triPars, TriParameter repl,

TriStatus transmissionFailure)

void tliPrReply\_m\_MC

(String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriPortId to,

TriSignatureId signature, TciParameterListType tciPars, Value replValue,

TciValueList addrValues, TriStatus encoderFailure, TriParameterList triPars,

TriParameter repl, TriAddressList addresses, TciStatus transmissionFailure)

void tliPrReply\_c

(String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriPortId to,

TriSignatureId signature, TciParameterListType tciPars, Value replValue,

TriStatus transmissionFailure)

void tliPrReply\_c\_BC

(String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriPortIdList to,

TriSignatureId signature, TciParameterListType tciPars, Value replValue,

TriStatus transmissionFailure)

void tliPrReply\_c\_MC

(String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriPortIdList to,

TriSignatureId signature, TciParameterListType tciPars, Value replValue,

TriStatus transmissionFailure)

void tliPrGetReplyDetected\_m

(String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriPortId from,

TriSignatureId signature, TriParameterList triPars, TriParameter repl, TriAddress address)

void tliPrGetReplyDetected\_c

(String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriPortId from,

TriSignatureId signature, TciParameterListType tciPars, Value replValue)

void tliPrGetReplyMismatch\_m

(String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriSignatureId signature,

TciParameterListType tciPars, TciValueTemplate parsTmpl, Value replValue,

TciValueTemplate replyTmpl, TciValueDifferenceList diffs, Value addrValue,

TciValueTemplate addressTmpl)

void tliPrGetReplyMismatch\_c

(String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriSignatureId signature,

TciParameterListType tciPars, TciValueTemplate parsTmpl, Value replValue,

TciValueTemplate replyTmpl, TciValueDifferenceList diffs, TriComponentId from,

TciNonValueTemplate fromTmpl)

void tliPrGetReply\_m

(String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriSignatureId signature,

TciParameterListType tciPars, TciValueTemplate parsTmpl, Value replValue,

TciValueTemplate replyTmpl, Value addrValue, TciValueTemplate addressTmpl)

void tliPrGetReply\_c

(String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriSignatureId signature,

TciParameterListType tciPars, TciValueTemplate parsTmpl, Value replValue,

TciValueTemplate replyTmpl, TriComponentId from, TciNonValueTemplate fromTmpl)

void tliPrRaise\_m

(String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriPortId to,

TriSignatureId signature, TciParameterListType tciPars, Value excValue, Value addrValue,

TciStatus encoderFailure, TriException exc, TriAddress address, TriStatus transmissionFailure)

void tliPrRaise\_m\_BC

(String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriPortId to,

TriSignatureId signature, TciParameterListType tciPars, Value excValue,

TciStatus encoderFailure, TriException exc, TriStatus transmissionFailure)

void tliPrRaise\_m\_MC

(String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriPortId to,

TriSignatureId signature, TciParameterListType tciPars, Value excValue,

TciValueList addrValues, TciStatus encoderFailure, TriException exc,

TriAddressList addresses, TriStatus transmissionFailure)

void tliPrRaise\_c

(String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriPortId to,

TriSignatureId signature, TciParameterListType tciPars, Value excValue,

TriStatus transmissionFailure)

void tliPrRaise\_c\_BC

(String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriPortIdList to,

TriSignatureId signature, TciParameterListType tciPars, Value excValue,

TriStatus transmissionFailure)

void tliPrRaise\_c\_MC

(String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriPortIdList to,

TriSignatureId signature, TciParameterListType tciPars, Value excValue,

TriStatus transmissionFailure)

void tliPrCatchDetected\_m

(String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriPortId from,

TriSignatureId signature, TriException exc, TriAddress address)

void tliPrCatchDetected\_c

(String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriPortId from,

TriSignatureId signature, Value excValue)

void tliPrCatchMismatch\_m

(String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriSignatureId signature,

Value excValue, TciValueTemplate excTmpl, TciValueDifferenceList diffs, Value addrValue,

TciValueTemplate addressTmpl)

void tliPrCatchMismatch\_c

(String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriSignatureId signature,

Value excValue, TciValueTemplate excTmpl, TciValueDifferenceList diffs, TriComponentId from,

TciNonValueTemplate fromTmpl)

void tliPrCatch\_m

(String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriSignatureId signature,

Value excValue, TciValueTemplate excTmpl, Value addrValue, TciValueTemplate addressTmpl)

void tliPrCatch\_c

(String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriSignatureId signature,

Value excValue, TciValueTemplate excTmpl, TriComponentId from, TciNonValueTemplate fromTmpl)

void tliPrCatchTimeoutDetected

(String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriSignatureId signature)

void tliPrCatchTimeout

(String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriSignatureId signature)

void tliCCreate

(String am, int ts, String src, int line, TriComponentId c, TriComponentId comp, String name,

Boolean alive)

void tliCStart

(String am, int ts, String src, int line, TriComponentId c, TriComponentId comp,

TciBehaviourIdType name, TciParameterListType tciPars)

void tliCRunning

(String am, int ts, String src, int line, TriComponentId c, TriComponentId comp,

ComponentStatus status)

void tliCAlive

(String am, int ts, String src, int line, TriComponentId c, TriComponentId comp,

ComponentStatus status)

void tliCStop

(String am, int ts, String src, int line, TriComponentId c, TriComponentId comp)

void tliCKill  
 (String am, int ts, String src, int line, TriComponentId c, TriComponentId comp)

void tliCDoneMismatch

(String am, int ts, String src, int line, TriComponentId c, TriComponentId comp,

TciNonValueTemplate compTmpl)

void tliCDone

(String am, int ts, String src, int line, TriComponentId c, TciNonValueTemplate compTmpl,  
 VerdictValue verdict)

void tliCTerminated

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

void tliCKilledMismatch

(String am, int ts, String src, int line, TriComponentId c, TriComponentId comp,

TciNonValueTemplate compTmpl)

void tliCKilled

(String am, int ts, String src, int line, TriComponentId c, TciNonValueTemplate compTmpl,  
 VerdictValue verdict)

void tliPConnect

(String am, int ts, String src, int line, TriComponentId c, TriPortId port1, TriPortId port2)

void tliPDisconnect

(String am, int ts, String src, int line, TriComponentId c, TriPortId port1,

TriPortId port2)

void tliPMap

(String am, int ts, String src, int line, TriComponentId c, TriPortId port1, TriPortId port2)

void tliPMapParam

(String am, int ts, String src, int line, TriComponentId c, TriPortId port1, TriPortId port2,

TciParameterListType tciPars, TciStatus encoderFailure, TriParameterList triPars)

void tliPUnmap

(String am, int ts, String src, int line, TriComponentId c, TriPortId port1,

TriPortId port2)

void tliPUnmapParam

(String am, int ts, String src, int line, TriComponentId c, TriPortId port1,

TriPortId port2, TciParameterListType tciPars, TciStatus encoderFailure, TriParameterList triPars)

void tliPClear

(String am, int ts, String src, int line, TriComponentId c, TriPortId port)

void tliPStart

(String am, int ts, String src, int line, TriComponentId c, TriPortId port)

void tliPStop

(String am, int ts, String src, int line, TriComponentId c, TriPortId port)

void tliPHalt

(String am, int ts, String src, int line, TriComponentId c, TriPortId port)

void tliEncode

(String am, int ts, String src, int line, TriComponentId c, Value val, TciStatus encoderFailure,

TriMessage msg, String codec)

void tliDecode

(String am, int ts, String src, int line, TriComponentId c, TriMessage msg,

TciStatus decoderFailure, Value val, String codec)

void tliTTimeoutDetected

(String am, int ts, String src, int line, TriComponentId c, TriTimerId timer)

void tliTTimeoutMismatch

(String am, int ts, String src, int line, TriComponentId c, TriTimerId timer,

TciNonValueTemplate timerTmpl)

void tliTTimeout

(String am, int ts, String src, int line, TriComponentId c, TriTimerId timer,

TciNonValueTemplate timerTmpl)

void tliTStart

(String am, int ts, String src, int line, TriComponentId c, TriTimerId timer, TriTimerDuration dur)

void tliTStop

(String am, int ts, String src, int line, TriComponentId c, TriTimerId timer, TriTimerDuration dur)

void tliTRead

(String am, int ts, String src, int line, TriComponentId c, TriTimerId timer,

TriTimerDuration elapsed)

void tliTRunning

(String am, int ts, String src, int line, TriComponentId c, TriTimerId timer, TimerStatus status)

void tliSEnter

(String am, int ts, String src, int line, TriComponentId c, QualifiedName name,

TciParameterListType tciPars, String kind)

void tliSLeave

(String am, int ts, String src, int line, TriComponentId c, QualifiedName name,

TciParameterListType tciPars, Value returnValue, String kind)

void tliVar

(String am, int ts, String src, int line, TriComponentId c, QualifiedName name, Value varValue)

void tliModulePar

(String am, int ts, String src, int line, TriComponentId c, QualifiedName name, Value parValue)

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

void tliSetVerdict

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

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

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

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

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

void tliAActivate

(String am, int ts, String src, int line, TriComponentId c, QualifiedName name,

TciParameterListType tciPars, Value ref)

void tliADeactivate(String am, int ts, String src, int line, TriComponentId c, Value ref)

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

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

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

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

void tliMatch

(String am, int ts, String src, int line, TriComponentId c, Value expr, TciValueTemplate tmpl)

void tliMatchMismatch

(String am, int ts, String src, int line, TriComponentId c, Value expr, TciValueTemplate tmpl,

TciValueDifferenceList diffs);

void tliInfo

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

void tliMChecked\_m

(String am, int ts, String src, int line, TriComponentId c, TriPortId at, Value msgValue,

TciValueTemplate msgTmpl, Value addrValue, TciValueTemplate addressTmpl)

void tliMChecked\_c

(String am, int ts, String src, int line, TriComponentId c, TriPortId at, Value msgValue,

TciValueTemplate msgTmpl, TriComponentId from, TciNonValueTemplate fromTmpl)

void tliPrGetCallChecked\_m

(String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriSignatureId signature,

TciParameterListType tciPars, TciValueTemplate parsTmpl,

Value addrValue, TciValueTemplate addressTmpl)

void tliPrGetCallChecked\_c

(String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriSignatureId signature,

TciParameterListType tciPars, TciValueTemplate parsTmpl,

TriComponentId from, TciNonValueTemplate fromTmpl)

void tliPrGetReplyChecked\_m

(String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriSignatureId signature,

TciParameterListType tciPars, TciValueTemplate parsTmpl,

Value replValue, TciValueTemplate replyTmpl, Value addrValue, TciValueTemplate addressTmpl)

void tliPrGetReplyChecked\_c

(String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriSignatureId signature,

TciParameterListType tciPars, TciValueTemplate parsTmpl,

Value replValue, TciValueTemplate replyTmpl, TriComponentId from, TciNonValueTemplate fromTmpl)

void tliPrCatchChecked\_m

(String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriSignatureId signature,

Value excValue, TciValueTemplate excTmpl, Value addrValue, TciValueTemplate addressTmpl)

void tliPrCatchChecked\_c

(String am, int ts, String src, int line, TriComponentId c, TriPortId at, TriSignatureId signature,

Value excValue, TciValueTemplate excTmpl, TriComponentId from, TciNonValueTemplate fromTmpl)

void tliCheckedAny\_m

(String am, int ts, String src, int line, TriComponentId c, TriPortId at,

Value addrValue, TciValueTemplate addressTmpl)

void tliCheckedAny\_c

(String am, int ts, String src, int line, TriComponentId c, TriPortId at,

TriComponentId from, TciNonValueTemplate fromTmpl)

void tliCheckAnyMismatch\_m

(String am, int ts, String src, int line, TriComponentId c, TriPortId at,

Value addrValue, TciValueTemplate addressTmpl)

void tliCheckAnyMismatch\_c

(String am, int ts, String src, int line, TriComponentId c, TriPortId at,

TriComponentId from, TciNonValueTemplate fromTmpl)

void tliRnd

(String am, int ts, String src, int line, TriComponentId c, Value val, Value seed)

void tliEvaluate

(String am, int ts, String src, int line, TriComponentId c, QualifiedName name, Value evalResult)

void tliCCall

(String am, int ts, String src, int line, TriComponentId c, TriComponentId comp,

TciBehaviourIdType name, TciParameterListType tciPars)

void tliCCallTerminated

(String am, int ts, String src, int line, TriComponentId c, VerdictValue verdict, String reason,  
 TciParameterListType tciPars, Value returnValue)

void tliCtrlStartWithParameters   
 (String am, int ts, String src, int line, TriComponentId c, TciParameterListType tciPars)

void tliCtrlTerminatedWithResult  
 (String am, int ts, String src, int line, TriComponentId c, Value result,   
 TciParameterListType tciPars)

## 9.5 Data

Table

| TCI IDL ADT | ANSI C representation (Type definition) | Notes and comments |
| --- | --- | --- |
| TciModuleIdType | QualifiedName |  |
| TciModuleParameterType | typedef struct TciModuleParameterType  {  QualifiedName parName;  Value defaultValue;  } TciModuleParameterType; |  |
| TciModuleParameterListType | typedef struct TciModuleParameterListType  {  long int length;  TciModuleParameterType \*modParList;  } TciModuleParameterListType; |  |
| TciParameterType | typedef struct TciParameterType  {  String parName;  TciParameterPassingModeType parPassMode;  Value parValue;  } TciParameterType; |  |
| TciParameterPassingModeType | typedef enum  {  TCI\_IN\_PAR = 0,  TCI\_INOUT\_PAR = 1,  TCI\_OUT\_PAR = 2  } TciParameterPassingModeType; |  |
| TciParameterListType | typedef struct TciParameterListType  {  long int length;  TciParameterType \*parList;  } TciParameterListType; | length 0 shall be interpreted as "empty list". |
| TciParameterTypeListType | typedef struct TciParameterTypeListType  {  long int length;  TciParameterTypeType \*parList;  } TciParameterTypeListType; | length 0 shall be interpreted as "empty list". |
| TciParameterTypeType | typedef struct TciParameterTypeType {  String parName;  Type parameterType;  TciParameterPassingModeType mode;  } TciParameterTypeType; |  |
| TciTestCaseIdListType | typedef struct TciTestCaseIdListType  {  long int length;  QualifiedName \*idList;  } TciTestCaseIdListType; | length 0 shall be interpreted as "empty list". |
| TciTypeClassType | typedef enum  {  TCI\_ADDRESS\_TYPE = 0,  TCI\_ANYTYPE\_TYPE = 1,  TCI\_BITSTRING\_TYPE = 2,  TCI\_BOOLEAN\_TYPE = 3,  TCI\_CHARSTRING\_TYPE = 5,  TCI\_COMPONENT\_TYPE = 6,  TCI\_ENUMERATED\_TYPE = 7,  TCI\_FLOAT\_TYPE = 8,  TCI\_HEXSTRING\_TYPE = 9,  TCI\_INTEGER\_TYPE = 10,  TCI\_OCTETSTRING\_TYPE = 12,  TCI\_RECORD\_TYPE = 13,  TCI\_RECORD\_OF\_TYPE = 14,  TCI\_ARRAY\_TYPE = 15,  TCI\_SET\_TYPE = 16,  TCI\_SET\_OF\_TYPE = 17,  TCI\_UNION\_TYPE = 18,  TCI\_UNIVERSAL\_CHARSTRING\_TYPE = 20,  TCI\_VERDICT\_TYPE = 21  TCI\_DEFAULT\_TYPE = 22,  TCI\_PORT\_TYPE = 23,  TCI\_TIMER\_TYPE = 24  } TciTypeClassType; |  |
| TciTestComponentKindType | typedef enum  {  TCI\_CTRL\_COMP,  TCI\_MTC\_COMP,  TCI\_PTC\_COMP,  TCI\_SYS\_COMP,  TCI\_ALIVE\_COMP  } TciTestComponentKindType; |  |
| TciBehaviourIdType | QualifiedName |  |
| TciValueDifference | typedef struct TciValueDifference  {  Value val;  TciValueTemplate tmpl;  String desc;  } TciValueDifference; |  |
| TciValueDifferenceList | typedef struct TciValueDifferenceList  {  long int length;  TciValueDifference\* diffList;  } TciValueDifferenceList; | length 0 shall be interpreted as "empty list". |
| TciMatchingTypeType | typedef enum  {  TCI\_TEMPLATE\_LIST = 0,  TCI\_COMPLEMENTED\_LIST = 1,  TCI\_ANY\_VALUE = 2,  TCI\_ANY\_VALUE\_OR\_NONE = 3,  TCI\_VALUE\_RANGE = 4,  TCI\_SUBSET = 5,  TCI\_SUPERSET = 6,  TCI\_ANY\_ELEMENT = 7,  TCI\_ANY\_ELEMENTS\_OR\_NONE = 8,  TCI\_PATTERN = 9,  TCI\_MATCH\_DECODED\_CONTENT = 10,  TCI\_OMIT\_TEMPLATE = 11  } TciMatchingTypeType; |  |
| LengthRestriction | typedef struct TciLengthRestriction  {  unsigned long int lowerBoundary;  unsigned long int upperBoundary;  Boolean isUpperBoundaryInfinity;  } TciLengthRestriction; |  |
| Permutation | typedef struct TciPermutation  {  unsigned long int startPosition;  unsigned long int length; } TciPermutation; |  |
| RangeBoundary | typedef struct TciRangeBoundary  {  Value boundary;  Boolean isInclusive;  Boolean isInfinity;  } TciRangeBoundary; |  |

## 9.6 Miscellaneous

Table

| TCI concept | ANSI C representation | Notes and comments |
| --- | --- | --- |
| Verdict representation | | |
| NONE | const int TCI\_VERDICT\_NONE = 0 | Since the VerdictValue interface is defined in terms of integers, consensus shall be established on which value defines which verdict. |
| PASS | const int TCI\_VERDICT\_PASS = 1 |  |
| INCONC | const int TCI\_VERDICT\_INCONC = 2 |  |
| FAIL | const int TCI\_VERDICT\_FAIL = 3 |  |
| ERROR | const int TCI\_VERDICT\_ERROR = 4 |  |
| USER\_ERROR | const int TCI\_VERDICT\_USER\_ERROR = 5 |  |
| ComponentStatus | | |
| INACTIVE\_C | const int TCI\_INACTIVE\_C = 0 |  |
| RUNNING\_C | const int TCI\_RUNNING\_C = 1 |  |
| STOPPED\_C | const int TCI\_STOPPED\_C = 2 |  |
| KILLED\_C | const int TCI\_KILLED\_C = 3 |  |
| NULL\_C | const int TCI\_NULL\_C = 4 |  |
| TimerStatus | | |
| RUNNING\_T | const int TCI\_RUNNING\_T = 0 |  |
| INACTIVE\_T | const int TCI\_INACTIVE\_T = 1 |  |
| EXPIRED\_T | const int TCI\_ EXPIRED\_T = 2 |  |
| NULL\_T | const int TCI\_NULL\_T = 3 |  |
| TciStatus | | |
| TCI\_OK | const int TCI\_OK = 0 |  |
| TCI\_ERROR | const int TCI\_ERROR = -1 |  |
| CharstringValue representation | | |
| TciCharString | typedef struct TciCharStringValue  {  unsigned long int length;  char\* string;  } TciCharStringValue |  |
| Universal Character[string] representation | | |
| Universal Char | typedef unsigned char TciUCValue[4] |  |
| Universal Charstring | typedef struct TciUCStringValue  {  unsigned long int length;  TciUCValue \*string;  } TciUCStringValue; |  |

## 9.7 Optional parameters

Clause 7.2.2 defines that a reserved value shall be used to indicate the absence of an optional parameter. For the C language mapping an explicit null shall be used. The function tciSetNull can be used to set a value to null and tciIsNull can be used to check whether a value represents null. tciIsNull returns true if the value is null, false otherwise.

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

tciSetNull(reply);  
tciReplyConnected (sender, receiver, signature, parameterList, reply).

# 10 C++ language mapping

## 10.1 Introduction

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

## 10.2 Names and scopes

The namespace ORG\_ETSI\_TTCN3\_TCI has been defined for the TCI C++ mapping, in order to avoid conflicts with the different names used, for example, in the C mapping.

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

## 10.3 Memory management

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

## 10.4 Error handling

No additional error handling has been defined for this mapping.

## 10.5 Type mapping

### 10.5.0 Basic concepts

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

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

### 10.5.1 Encapsulated C++ types

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

Boolean type definition: typedef bool Tboolean

Integer type definition: typedef long int Tinteger

Size type definition: typedef unsigned long int Tsize

Index type definition: typedef unsigned long int Tindex

Float type definition: typedef double Tfloat

String type definition: typedef std::string Tstring

Universal string type definition: typedef std::wstring TuniversalString

Bit type definition: typedef unsigned char Tbit

Char type definition: typedef unsigned char Tchar

### 10.5.2 General abstract data types

#### 10.5.2.1 TciBehaviourId

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

class TciBehaviourId: public ORG\_ETSI\_TTCN3\_TRI::QualifiedName {

public:

virtual ~TciBehaviourId ();

virtual Tboolean operator== (const TciBehaviourId &bid) const =0;

virtual TciBehaviourId \* clone () const =0;

virtual Tboolean operator< (const TciBehaviourId &bid) const =0;

}

**Methods:**

~TciBehaviourId

Destructor

operator==

Returns true if both objects are equal

clone

Return a copy of the TciBehaviourId

operator<

Operator < overload

#### 10.5.2.2 TciModuleId

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

class TciModuleId {

public:

virtual ~TciModuleId ()

virtual const Tstring & getObjectName() const = 0;

virtual void setObjectName (const Tstring &p\_name)=0;

virtual Tboolean operator== (const TciModuleId &mid) const =0;

virtual TciModuleId \* clone () const =0;

virtual Tboolean operator< (const TciModuleId &mid) const =0;

}

**Methods:**

~TciModuleId

Destructor

getObjectName

Get the moduleId name

setObjectName

Set the moduleId name

operator==

Returns true if both objects are equal

clone

Return a copy of the TciModuleId

operator<

Operator < overload

#### 10.5.2.3 TciModuleParameterId

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

class TciModuleParameterId : public ORG\_ETSI\_TTCN3\_TRI::QualifiedName {

public:

virtual ~TciModuleParameterId ();

virtual Tboolean operator== (const TciModuleParameterId &mparId) const =0;

virtual TciModuleParameterId \* clone () const =0;

virtual Tboolean operator< (const TciModuleParameterId &mparId) const =0;

}

**Methods:**

~TciModuleParameterId

Destructor

operator==

Returns true if both objects are equal

clone

Return a copy of the TciModuleParameterId

operator<

Operator < overload

#### 10.5.2.4 TciTestCaseId

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

class TciTestCaseId : public TciBehaviourId {

public:

virtual ~TciTestCaseId();

virtual Tboolean operator== (const TciTestCaseId &tcid) const =0;

virtual TciTestCaseId \* clone () const =0;

virtual Tboolean operator< (const TciTestCaseId &tcid) const =0;

}

**Methods:**

~TciTestCaseId

Destructor

operator==

Returns true if both objects are equal

clone

Return a copy of the TciTestCaseId

operator<

Operator < overload

#### 10.5.2.5 TciModuleIdList

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

class TciModuleIdList {

public:

virtual ~TciModuleIdList();

virtual Tsize size () const =0;

virtual Tboolean empty () const =0;

virtual const TciModuleId \*get (Tsize p\_index) const =0;

virtual void clear ()=0;

virtual void push\_back (const TciModuleId &comp)=0;

virtual Tboolean operator== (const TciModuleIdList &midList) const =0;

virtual TciModuleIdList \* clone () const =0;

virtual Tboolean operator< (const TciModuleIdList &midList) const =0;

}

**Methods:**

~TciModuleIdList

Destructor

size

Return the size of the list

empty

Return true if the list is empty

get

Return the requested element

clear

Remove all the components from this list

push\_back

Add a component to the end of this list

operator==

Returns true if both objects are equal

clone

Return a copy of the TciModuleId

operator<

Operator < overload

#### 10.5.2.6 TciModuleParameter

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

class TciModuleParameter {

public:

virtual ~TciModuleParameter ();

virtual const TciValue & getDefaultValue () const =0;

virtual const Tstring & getModuleParameterName () const =0;

virtual const TciModuleParameterId & getTciModuleParameterId () const =0;

virtual Tboolean operator== (const TciModuleParameter &mpar) const =0;

virtual TciModuleParameter \* clone () const =0;

virtual Tboolean operator< (const TciModuleParameter &mpar) const =0;

}

**Methods:**

~TciModuleParameter

Destructor

getDefaultValue

Return default value of the parameter

getModuleParameterName

Return parameter name

getTciModuleParameterId

Get the name of the module parameter as defined in the TTCN-3 module

operator==

Returns true if both objects are equal

clone

Return a copy of the TciModuleParameter

operator<

Operator < overload

#### 10.5.2.7 TciModuleParameterList

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

class TciModuleParameterList {

public:

virtual ~TciModuleParameterList ();

virtual Tsize size () const =0;

virtual Tboolean empty () const =0;

virtual const TciModuleParameter \*get (Tindex p\_index) const =0;

virtual void clear ()=0;

virtual void push\_back (const TciModuleParameter &comp)=0;

virtual Tboolean operator== (const TciModuleParameterList &mparList) const =0;

virtual TciModuleParameterList \* clone () const =0;

virtual Tboolean operator< (const TciModuleParameterList &mparList) const =0;

}

**Methods:**

~TciModuleParameterList

Destructor

size

Return the size of the list

empty

Return true if the list is empty

get

Retrieve the specified element

clear

Remove all components from this list

push\_back

Add a component to the end of this list

operator==

Returns true if both objects are equal

clone

Return a copy of the TciModuleParameterList

operator<

Operator < overload

#### 10.5.2.8 TciParameterPassingMode

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

typedef enum

{

IN = 0,

OUT = 1,

INOUT = 2

} TciParameterPassingMode;

#### 10.5.2.9 TciParameter

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

class TciParameter {

public:

virtual ~TciParameter ();

virtual TciValue & getValue ()=0;

virtual void setValue (TciValue &value)=0;

virtual const TciParameterPassingMode &getParameterPassingMode () const =0;

virtual void setParameterPassingMode (const TciParameterPassingMode &mode)=0;

virtual const Tstring & getParameterName () const =0;

virtual void setParameterName (const Tstring &name)=0;

virtual Tboolean operator== (const TciParameter &param) const =0;

virtual TciParameter \* clone () const =0;

virtual Tboolean operator< (const TciParameter &param) const =0;

}

**Methods:**

~TciParameter

Destructor

getValue

Retrieve the TTCN-3 value

setValue

Set the TTCN-3 value

getParameterPassingMode

Return the parameter passing mode

setParameterPassingMode

Set the parameter passing mode

getParameterName

Return the name of the parameter

setParameterName

Set the name of the parameter

operator==

Returns true if both objects are equal

clone

Return a copy of the TciParameter

operator<

Operator < overload

#### 10.5.2.10 TciParameterList

Defines a list of TciParameter elements. It is mapped to the following pure virtual class:

class TciParameterList {

public:

virtual ~TciParameterList ();

virtual Tsize size () const =0;

virtual Tboolean empty () const =0;

virtual TciParameter \*get (Tindex p\_index) =0;

virtual void clear ()=0;

virtual void push\_back (const TciParameter &comp)=0;

virtual Tboolean operator== (const TciParameterList &param) const =0;

virtual TciParameterList \* clone () const =0;

virtual Tboolean operator< (const TciParameterList &param) const =0;

}

**Methods:**

~TciParameterList

Destructor

size

Return the size of the list

empty

Return true if the list is empty

get

Get the specified element

clear

Remove all the components from this list

push\_back

Add a component to the end of this list

operator==

Returns true if both objects are equal

clone

Return a copy of the TciParameterList

operator<

Operator < overload

#### 10.5.2.11 TciParameterType

Includes a TTCN-3 type, parameter name and a TciParameterPassingMode. It is mapped to the following pure virtual class:

class TciParameterType {

public:

virtual ~TciParameterType ();

virtual const Tstring & getParameterName () const =0;

virtual const TciType & getType () const =0;

virtual const TciParameterPassingMode & getParameterPassingMode () const =0;

virtual Tboolean operator== (const TciParameterType &parType) const =0;

virtual TciParameterType \* clone () const =0;

virtual Tboolean operator< (const TciParameterType &parType) const =0;

}

**Methods:**

~TciParameterType

Destructor

getType

Return the TTCN-3 Type

getParameterName

Return the name of the parameter

getParameterPassingMode

Get the parameter passing mode

operator==

Returns true if both objects are equal

clone

Return a copy of the TciParameterType

operator<

Operator < overload

#### 10.5.2.12 TciParameterTypeList

Specifies a list of TciParameterType elements. It is mapped to the following pure virtual class:

class TciParameterTypeList {

public:

virtual ~TciParameterTypeList ();

virtual Tsize size () const =0;

virtual Tboolean empty () const =0;

virtual const TciParameterType \*get (Tindex p\_position) const =0;

virtual void clear ()=0;

virtual void push\_back (const TciParameterType &comp)=0;

virtual Tboolean operator== (const TciParameterTypeList &ptypeList) const =0;

virtual TciParameterTypeList \* clone () const =0;

virtual Tboolean operator< (const TciParameterTypeList &ptypeList) const =0;

}

**Methods:**

~TciParameterTypeList

Destructor

size

Return the size of the list

empty

Returns true if the list is empty

get

Return the requested element

clear

Remove all the components from this list

push\_back

Add a component to the end of this list

operator==

Returns true if both objects are equal

clone ()

Returns a copy of the TciParameterTypeList

operator< (const TciParameterTypeList &ptypeList)

Operator < overload

#### 10.5.2.13 TciTestComponentKind

Defines the test component kind. It is mapped to an enumeration:

typedef enum

{

SYSTEM\_COMP = 0,

PTC\_COMP = 1,

PTC\_ALIVE\_COMP = 2,

MTC\_COMP = 3,

CTRL\_COMP = 4

} TciTestComponentKind;

#### 10.5.2.14 TciTypeClass

Defines the type class. It is mapped to an enumeration:

typedef enum

{

TCI\_ADDRESS = 0,

TCI\_ANYTYPE = 1,

TCI\_BITSTRING = 2,

TCI\_BOOLEAN = 3,

TCI\_CHARSTRING = 5,

TCI\_COMPONENT = 6,

TCI\_ENUMERATED = 7,

TCI\_FLOAT = 8,

TCI\_HEXSTRING = 9,

TCI\_INTEGER = 10,

TCI\_OCTETSTRING = 12,

TCI\_RECORD = 13,

TCI\_RECORD\_OF = 14,

TCI\_ARRAY = 15,

TCI\_SET = 16,

TCI\_SET\_OF = 17,

TCI\_UNION = 18,

TCI\_UNIVERSAL\_CHARSTRING = 20,

TCI\_VERDICT = 21

TCI\_DEFAULT = 22,

TCI\_PORT = 23,

TCI\_TIMER = 24

} TciTypeClass;

#### 10.5.2.15 TciTestCaseIdList

Specifies a list of TciTestCaseId elements. It is mapped to the following pure virtual class:

class TciTestCaseIdList {

public:

virtual ~ TciTestCaseIdList ();

virtual Tsize size () const =0;

virtual Tboolean empty () const =0;

virtual const TciTestCaseId \*get (Tindex p\_position) const =0;

virtual void clear ()=0;

virtual void push\_back (const TciTestCaseId &comp)=0;

virtual Tboolean operator== (const TciTestCaseIdList &ptypeList) const =0;

virtual TciTestCaseIdList \* clone () const =0;

virtual Tboolean operator< (const TciTestCaseIdList &ptypeList) const =0;

}

**Methods:**

~TciTestCaseIdList

Destructor

size

Return the size of the list

empty

Returns true if the list is empty

get

Return the requested element

clear

Remove all the components from this list

push\_back

Add a component to the end of this list

operator==

Returns true if both objects are equal

clone ()

Returns a copy of the TciTestCaseIdList

operator< (const TciTestCaseIdList &ptypeList)

Operator < overload

#### 10.5.2.16 TciMatchingTypeType

Defines the matching template type. It is mapped to an enumeration:

typedef enum

{

TCI\_TEMPLATE\_LIST = 0,

TCI\_COMPLEMENTED\_LIST = 1,

TCI\_ANY\_VALUE = 2,

TCI\_ANY\_VALUE\_OR\_NONE = 3,

TCI\_VALUE\_RANGE = 4,

TCI\_SUBSET = 5,

TCI\_SUPERSET = 6,

TCI\_ANY\_ELEMENT = 7,

TCI\_ANY\_ELEMENTS\_OR\_NONE = 8,

TCI\_PATTERN = 9,

TCI\_MATCH\_DECODED\_CONTENT = 10,

TCI\_OMIT\_TEMPLATE = 11

} TciMatchingType;

#### 10.5.2.17 LengthRestriction

Specifies a length restriction. It is mapped to the following pure virtual class:

class LengthRestriction {

public:

virtual ~LengthRestriction ();

virtual Tinteger getLowerBoundary () const =0;

virtual Tinteger getUpperBoundary () const =0;

virtual void setLowerBoundary (Tinteger p\_boundary) =0;

virtual void setUpperBoundary (Tinteger p\_boundary) =0;

virtual Tboolean isUpperBoundaryInfinity () const =0;

virtual void setInfiniteUpperBoundary () =0;

virtual Tboolean operator== (const LengthRestriction &p\_lenRestriction) const =0;

virtual LengthRestriction \* clone () const =0;

virtual Tboolean operator< (const LengthRestriction &p\_lenRestriction) const =0;

}

**Methods:**

~LengthRestriction

Destructor

getLowerBoundary

Returns the lower boundary of the length restriction

getUpperBoundary

Returns the upper boundary of the length restriction

setLowerBoundary

Sets the lower boundary value

setUpperBoundary

Sets the upper boundary value

isUpperBoundaryInfinity

Returns true if the upper boundary contains infinity and false otherwise

setInfiniteUpperBoundary

Sets the upper boundary to infinity

operator==

Returns true if both objects are equal

clone ()

Returns a copy of the LengthRestriction

operator<

Operator < overload

#### 10.5.2.18 Permutation

Specifies a permutation. It is mapped to the following pure virtual class:

class Permutation {

public:

virtual ~Permutation ();

virtual Tindex getStartPosition () const =0;

virtual void setStartPosition (Tindex p\_position) =0;

virtual Tsize getLength () const =0;

virtual void setLength (Tsize p\_length) =0;

virtual Tboolean operator== (const Permutation &p\_permutation) const =0;

virtual Permutation \* clone () const =0;

virtual Tboolean operator< (const Permutation &p\_permutation) const =0;

}

**Methods:**

~Permutation

Destructor

getStartPosition

Returns the position of the first item of the permutation in the RecordOfValue

setStartPosition

Sets the position of the first item of the permutation in the RecordOfValue

getLength

Returns the number of elements or matching mechanisms of the RecordOfValue that are included in the permutation

setLength

Sets the number of elements or matching mechanisms of the RecordOfValue that are included in the permutation

operator==

Returns true if both objects are equal

clone ()

Returns a copy of the Permutation

operator<

Operator < overload

#### 10.5.2.19 RangeBoundary

Specifies a lower or upper boundary of a ValueRange. It is mapped to the following pure virtual class:

class RangeBoundary {

public:

virtual ~RangeBoundary ();

virtual TciValue& getBoundary () const =0;

virtual Tboolean isInclusive () const =0;

virtual void setBoundary (TciValue &p\_value, Tboolean p\_isInclusive) =0;

virtual Tboolean isInfinity () const =0;

virtual void setToInfinity () =0;

virtual Tboolean operator== (const RangeBoundary &p\_boundary) const =0;

virtual RangeBoundary \* clone () const =0;

virtual Tboolean operator< (const RangeBoundary &p\_boundary) const =0;

}

**Methods:**

~RangeBoundary

Destructor

getBoundary

Returns the boundary value

isInclusive

Returns true if the boundary value is a part of the allowed range and false otherwise

setBoundary

Sets the boundary value

isInfinity

Returns true if the boundary is equal to infinity or -infinity and false otherwise

setToInfinity

Sets the boundary to infinity

operator==

Returns true if both objects are equal

clone ()

Returns a copy of the LengthRestriction

operator<

Operator < overload

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

#### 10.5.3.1 TciType

A value of TciType represents one of the TTCN-3 types in a TTCN-3 module. It is mapped to the following pure virtual class:

class TciType {

public:

virtual ~TciType ();

virtual const TciModuleId & getDefiningModule () const =0;

virtual const Tstring & getName () const =0;

virtual const TciTypeClass & getTypeClass () const =0;

virtual const Tstring & getTypeEncoding () const =0;

virtual const Tstring & getTypeEncodingVariant () const =0;

virtual const std::vector<Tstring\*> & getEncodeAttributes () const =0;

virtual const std::vector<Tstring\*> & getVariantAttributes (const Tstring \* encoding) const =0;

virtual const std::vector<Tstring\*> & getTypeExtension() const =0;

virtual TciValue \* newInstance () const =0;

virtual MatchingMechanism \* newTemplate (TciMatchingType matchingType) const =0;

virtual TciValue \* parseValue (const Tstring & val) const =0;

virtual const RangeBoundary \* getLowerTypeBoundary() const = 0;

virtual const RangeBoundary \* getUpperTypeBoundary() const = 0;

virtual const LengthRestriction \* getTypeLengthRestriction() const = 0;

virtual const MatchingMechanism \* getTypeMatchingMechanism() const = 0;

virtual Tboolean operator== (const TciType &typ) const =0;

virtual TciType \* clone () const =0;

virtual Tboolean operator< (const TciType &typ) const =0;

}

**Methods:**

~TciType

Destructor

getDefiningModule

Returns the defining module as defined in the TTCN-3 module

getName

Returns type name as defined in the TTCN-3 module

getTypeClass

Returns this type class

getTypeEncoding

Returns type encoding as defined in the TTCN-3 module

getTypeEncodingVariant

Returns encoding variant as defined in the TTCN-3 module

getEncodeAttributes

Returns all encode attributes of the type as defined in the TTCN-3 module. The distinct value null is mapped to an empty vector

getVariantAttributes

Returns all variant attributes of the type as defined in the TTCN-3 module. The distinct value null is mapped to an empty vector

getTypeExtension

Returns type extension as defined in the TTCN-3 module

newInstance

Returns a new Value instance of this type

newTemplate

Returns a freshly created matching mechanism of this type. The matchingType parameter determines what kind of matching mechanism will be created. If the created matching mechanism contains additional data properties, these properties are uninitialized in the created matching mechanism

parseValue

Returns a new TciValue instance in case of successful parsing or null pointer in case of parsing error or if value parsing is not supported by the tool

getLowerTypeBoundary

Returns the lower boundary of the type restriction or null

getUpperTypeBoundary

Returns the upper boundary of the type restriction or null

getTypeLengthRestriction

Returns the type length restriction or null

getTypeMatchingMechanism

Returns the matching mechanism restriction of the type or null

operator==

Returns true if the types are equal

clone

Returns a copy of the TciType

operator<

Operator < overload

#### 10.5.3.2 TciValue

A value of TciValue represents TTCN-3 values for a given type. It is mapped to the following pure virtual class:

class TciValue {

public:

virtual ~TciValue ();

virtual const TciType & getType () const =0;

virtual const Tstring & getValueEncoding () const =0;

virtual const Tstring & getValueEncodingVariant () const =0;

virtual const std::vector<Tstring\*> & getEncodeAttributes () const =0;

virtual const std::vector<Tstring\*> & getVariantAttributes (const Tstring \* encoding) const =0;

virtual Tboolean notPresent () const =0;

virtual Tboolean isMatchingSymbol () const =0;

virtual const Tstring & valueToString () const =0;

virtual Tboolean isLazy () const =0;

virtual Tboolean isFuzzy () const =0;

virtual Tboolean isEvaluated () const =0;

virtual LengthRestriction \* getLengthRestriction () const = 0;

virtual LengthRestriction \* newLengthRestriction () const = 0;

virtual void setLengthRestriction (const LengthRestriction \* p\_restriction) =0;

virtual Tboolean isIfPresentEnabled () const =0;

virtual void setIfPresentEnabled (Tboolean p\_enabled) =0;

virtual Tboolean isOptional () const =0;

virtual RangeBoundary \* getLowerTypeBoundary() const = 0;

virtual RangeBoundary \* getUpperTypeBoundary() const = 0;

virtual LengthRestriction \* getTypeLengthRestriction() const = 0;

virtual MatchingMechanism \* getTypeMatchingMechanism() const = 0;

virtual Tboolean operator== (const TciValue &p\_val) const =0;

virtual TciValue \* clone () const =0;

virtual Tboolean operator< (const TciValue &p\_val) const =0;

}

**Methods:**

~TciValue

Destructor

getType

Returns the type of the specified value

getValueEncoding

Returns the value encoding attribute as defined in the TTCN-3 module

getValueEncodingVariant

Returns the value encoding variant attribute as defined in the TTCN-3 module

getEncodeAttributes

Returns all encode attributes of the value as defined in the TTCN-3 module. The distinct value null is mapped to an empty vector

getVariantAttributes

Returns all variant attributes of the value as defined in the TTCN-3 module. The distinct value null is mapped to an empty vector

notPresent

Returns true if the specified value is omit

isMatchingSymbol

Returns true if the specified value is a matching symbol (see clause 7.2.2.2.1 for more details), false otherwise

valueToString

Returns the same string as produced by the any2unistr predefined function

isLazy

Returns true if the specified value is @lazy, false otherwise

isFuzzy

Returns true if the specified value is @fuzzy, false otherwise

isEvaluated

Returns true if the specified value contains an evaluation result, false otherwise (see clause 7.2.2.2.1 for more details)

getLengthRestriction

Returns a length restriction matching attribute or null if no restriction is present

newLengthRestriction

Creates a new instance of the LengthRestriction class

setLengthRestriction

Adds a length restriction matching to the value or modifies an existing one. Null pointer can be used to remove an existing length restriction

isIfPresentEnabled

Returns true if the ifpresent matching attribute is attached to the value and false   
otherwise

setIfPresentEnabled

Sets the whether the ifpresent matching attribute is attached to the value or not

isOptional

Returns whether the value is an optional field or a template without the present or value restriction

getLowerTypeBoundary

Returns the lower boundary of the value's type restriction or null

getUpperTypeBoundary

Returns the upper boundary of the value's type restriction or null

getTypeLengthRestriction

Returns the value's type length restriction or null

getTypeMatchingMechanism

Returns the matching mechanism type restriction of the value's type or null

operator==

Returns true if both objects are equal

clone

Return a copy of the TciValue

operator<

Operator < overload

#### 10.5.3.3 IntegerValue

TTCN-3 integer value support. It is mapped to the following pure virtual class:

class IntegerValue : public virtual TciValue {

public:

virtual ~IntegerValue ();

virtual Tinteger getInt () const =0;

virtual void setInt (Tinteger p\_value)=0;

virtual Tboolean operator== (const IntegerValue &p\_intVal) const =0;

virtual IntegerValue \* clone () const =0;

virtual Tboolean operator< (const IntegerValue & p\_intVal) const =0;

}

**Methods:**

~IntegerValue

Destructor

getInt

Return integer value

setInt

Set integer value

operator==

Returns true if both objects are equal

clone

Return a copy of the IntegerValue

operator<

Operator < overload

#### 10.5.3.4 FloatValue

TTCN-3 float value support. It is mapped to the following pure virtual class:

class FloatValue : public virtual TciValue {

public:

virtual ~FloatValue ();

virtual Tfloat getFloat () const =0;

virtual void setFloat (Tfloat p\_floatValue)=0;

virtual Tboolean operator== (const FloatValue & p\_floatVal) const =0;

virtual FloatValue \* clone () const =0;

virtual Tboolean operator< (const FloatValue & p\_floatVal) const =0;

}

**Methods:**

~FloatValue

Destructor

getFloat

Return the float value

setFloat

Set float value

operator==

Returns true if both objects are equal

clone

Return a copy of the FloatValue

operator<

Operator < overload

#### 10.5.3.5 BooleanValue

TTCN-3 boolean values support. It is mapped to the following pure virtual class:

class BooleanValue : public virtual TciValue {

public:

virtual ~BooleanValue ();

virtual Tboolean getBoolean () const =0;

virtual void setBoolean (Tboolean p\_booleanValue)=0;

virtual Tboolean operator== (const BooleanValue & p\_booleanVal) const =0;

virtual BooleanValue \* clone () const =0;

virtual Tboolean operator< (const BooleanValue & p\_booleanVal) const =0;

}

**Methods:**

~BooleanValue

Destructor

getBoolean

Return the boolean value

setBoolean

Set the variable to booleanValue

operator==

Returns true if both objects are equal

clone

Return a copy of the BooleanValue

operator<

Operator < overload

#### 10.5.3.6 CharstringValue

TTCN-3 charstring value support. It is mapped to the following pure virtual class:

class CharstringValue : public virtual TciValue {

public:

virtual ~CharstringValue ();

virtual char getChar (Tindex p\_position) const =0;

virtual Tsize getLength () const =0;

virtual const Tstring & getString () const =0;

virtual void setChar (Tsize p\_position, char p\_char)=0;

virtual void setLength (Tsize p\_length)=0;

virtual void setString (const Tstring &p\_charValue)=0;

virtual Tboolean operator== (const CharstringValue & p\_charStr) const =0;

virtual CharstringValue \* clone () const =0;

virtual Tboolean operator< (const CharstringValue & p\_charStr) const =0;

}

**Methods:**

~CharstringValue

Destructor

getChar

Return the char at the specified position

getLength

Return length of the string

getString

Return the value of the string

setChar

Set the char at the specified position

setLength

Set length of the string

setString

Set the value of the string

operator==

Returns true if both objects are equal

clone

Return a copy of the CharstringValue

operator<

Operator < overload

#### 10.5.3.7 UniversalCharstringValue

TTCN-3 universal charstring value support. It is mapped to the following pure virtual class:

class UniversalCharstringValue : public virtual TciValue {

public:

virtual ~UniversalCharstringValue ();

virtual wchar\_t getChar (Tindex p\_position) const =0;

virtual Tsize getLength () const =0;

virtual const TuniversalString & getString () const =0;

virtual void setChar (Tindex p\_position, const wchar\_t p\_ucValue)=0;

virtual void setLength (Tsize p\_length)=0;

virtual void setString (const TuniversalString &p\_ucsValue)=0;

virtual Tboolean operator== (const UniversalCharstringValue & p\_uniCharstr) const =0;

virtual UniversalCharstringValue \* clone () const =0;

virtual Tboolean operator< (const UniversalCharstringValue & p\_uniCharstr) const =0;

}

**Methods:**

~UniversalCharstringValue

Destructor

getChar

Return the requested element

getLength

Return the length of the universal charstring

getString

Return the textual representation of the string

setChar

Set the char at the specified position

setLength

Set the length of the string

setString

Set the value of the string

operator==

Returns true if both objects are equal

clone

Return a copy of the UniversalCharstringValue

operator<

Operator < overload

#### 10.5.3.8 BitstringValue

TTCN-3 bitstring value support. It is mapped to the following pure virtual class:

class BitstringValue : public virtual TciValue {

public:

virtual ~BitstringValue ();

virtual Tbit getBit (Tindex p\_position) const =0;

virtual Tsize getLength () const =0;

virtual const Tstring & getString () const =0;

virtual void setBit (Tindex p\_position, Tbit p\_bsValue)=0;

virtual void setLength (Tindex p\_new\_length)=0;

virtual void setString (const Tstring &p\_bsValue)=0;

virtual std:istream \* getInputStream()=0;

virtual void setInputStream(std:istream \* stream, Tsize bitLen)=0;

virtual Tboolean isMatchingAt (Tindex p\_position) const =0;

virtual MatchingMechanism & getMatchingAt (Tindex p\_position) const =0;

virtual void setMatchingAt (Tindex p\_position, MatchingMechanism &p\_template) = 0;

virtual Tboolean operator== (const BitstringValue &p\_bitStr) const =0;

virtual BitstringValue \* clone () const =0;

virtual Tboolean operator< (const BitstringValue &p\_bitStr) const =0;

}

**Methods:**

~BitstringValue

Destructor

getBit

Returns the bit at the specified position

getLength

Returns the length of the string

getString

Returns the value of the string. In some cases, the message data are available in the form of a stream and cannot be converted into a string by TCI (e.g. because of memory restrictions). In such cases, this method returns null and the getInputStream method shall be used for reading the data

setBit

Sets the bit value at the specified position

setLength

Sets the length of the string

setString

Sets the string value

getInputStream  
Returns the bits in the form of an input stream. Repeated calls to the same method return different stream instances. The method returns null if the bitstring contains matching symbols

setInputStream  
Sets the value of this BitstringValue by providing a source stream that is used by the BitstringValue object to read the value content. Values set this way do not contain matching symbols

isMatchingAt

Returns true if the item at the specified position is a matching mechanism inside a value

getMatchingAt

Returns a matching mechanism at the specified position

setMatchingAt

Sets the matching mechanism at the specified position

operator==

Returns true if both objects are equal

clone

Returns a copy of the BitstringValue

operator<

Operator < overload

#### 10.5.3.9 OctetstringValue

TTCN-3 octetstring value support. It is mapped to the following pure virtual class:

class OctetstringValue : public virtual TciValue {

public:

virtual ~OctetstringValue ();

virtual Tsize getLength () const =0;

virtual const Tchar getOctet (Tindex p\_position) const =0;

virtual const Tstring & getString () const =0;

virtual void setLength (Tsize p\_length)=0;

virtual void setOctet (Tindex p\_position, Tchar p\_ochar)=0;

virtual void setString (const Tstring &p\_osValue)=0;

virtual std:istream \* getInputStream()=0;

virtual void setInputStream(std:istream \* stream)=0;

virtual Tboolean isMatchingAt (Tindex p\_position) const =0;

virtual MatchingMechanism & getMatchingAt (Tindex p\_position) const =0;

virtual void setMatchingAt (Tindex p\_position, MatchingMechanism &p\_template) = 0;

virtual Tboolean operator== (const OctetstringValue & p\_octStr) const =0;

virtual OctetstringValue \* clone () const =0;

virtual Tboolean operator< (const OctetstringValue & p\_octStr) const =0;

}

**Methods:**

~OctetstringValue

Destructor

getLength

Returns the length of the string

getOctet

Returns the textual representation of the octetchar at the specified position

getString

Returns the string value. In some cases, the message data are available in the form of a byte stream and cannot be converted into a string by TCI (e.g. because of memory restrictions). In such cases, this method returns null and the getInputStream method shall be used for reading the data

setLength

Sets the length of the string

setOctet

Sets the char at specified position

setString

Sets the value of the string

getInputStream  
Returns the octets in the form of an input stream. Repeated calls to the same method return different stream instances. The method returns null if the octetstring contains matching symbols

setInputStream  
Sets the value of this OctetstringValue by providing a source stream that is used by the OctetstringValue object to read the value content. Values set this way do not contain matching symbols

isMatchingAt

Returns true if the item at the specified position is a matching mechanism inside a value

getMatchingAt

Returns a matching mechanism at the specified position

setMatchingAt

Sets the matching mechanism at the specified position

operator==

Returns true if both objects are equal

clone

Returns a copy of the OctetstringValue

operator<

Operator < overload

#### 10.5.3.10 HexstringValue

TTCN-3 hexstring value support. It is mapped to the following pure virtual class:

class HexstringValue : public virtual TciValue {

public:

virtual ~HexstringValue ();

virtual Tchar getHex (Tindex p\_position) const =0;

virtual Tsize getLength () const =0;

virtual const Tstring & getString () const =0;

virtual void setHex (Tindex p\_position, Tchar p\_hcValue)=0;

virtual void setLength (Tsize p\_length)=0;

virtual void setString (const Tstring &p\_hsValue)=0;

virtual std:istream \* getInputStream()=0;

virtual void setInputStream(std:istream \* stream, Tsize numberOfItems)=0;

virtual Tboolean isMatchingAt (Tindex p\_position) const =0;

virtual MatchingMechanism & getMatchingAt (Tindex p\_position) const =0;

virtual void setMatchingAt (Tindex p\_position, MatchingMechanism &p\_template) = 0;

virtual Tboolean operator== (const HexstringValue & p\_hexStr) const =0;

virtual HexstringValue \* clone () const =0;

virtual Tboolean operator< (const HexstringValue & p\_hexStr) const =0;

}

**Methods:**

~HexstringValue

Destructor

getHex

Returns the element at the specified position

getLength

Returns the length of the string

getString

Returns the string value. In some cases, the message data are available in the form of a stream and cannot be converted into a string by TCI (e.g. because of memory restrictions). In such cases, this method returns null and the getInputStream method shall be used for reading the data

setHex

Sets the hex value at the specified position

setLength

Sets the length of the string

setString

Sets the value of the string

getInputStream  
Returns the data in the form of an input stream. Repeated calls to the same method return different stream instances. The method returns null if the hexstring contains matching symbols

setInputStream  
Sets the value of this HexstringValue by providing a source stream that is used by the HexstringValue object to read the value content. Values set this way do not contain matching symbols

isMatchingAt

Returns true if the item at the specified position is a matching mechanism inside a value

getMatchingAt

Returns a matching mechanism at the specified position

setMatchingAt

Sets the matching mechanism at the specified position

operator==

Returns true if both objects are equal

clone

Returns a copy of the HexstringValue

operator<

Operator < overload

#### 10.5.3.11 RecordValue

TTCN-3 record value support. It is mapped to the following pure virtual class:

class RecordValue : public virtual TciValue {

public:

virtual ~RecordValue ();

virtual TciValue &getField (const Tstring &p\_field\_name) =0;

virtual void setField (const Tstring &p\_field\_name,const TciValue &p\_new\_value)=0;

virtual const std::vector< Tstring \*> & getFieldNames () const =0;

virtual void setFieldOmitted (const Tstring & p\_fieldName)=0;

virtual Tboolean operator== (const RecordValue & p\_rec) const =0;

virtual RecordValue \* clone () const =0;

virtual Tboolean operator< (const RecordValue & p\_rec) const =0;

}

**Methods:**

~RecordValue

Destructor

getField

Return a reference to the field name

setField

Set the value of a field

getFieldNames

Return a list which containing the names of all the fields

setFieldOmitted

Set omit in one field

operator==

Returns true if both objects are equal

clone

Return a copy of the RecordValue

operator<

Operator < overload

#### 10.5.3.12 RecordOfValue

TTCN-3 record of value support. It is mapped to the following pure virtual class:

class RecordOfValue : public virtual TciValue {

public:

virtual ~RecordOfValue ();

virtual TciValue & getField (Tindex p\_position)=0;

virtual void setField (Tindex p\_position, const TciValue &p\_value)=0;

virtual void appendField (const TciValue &p\_value)=0;

virtual const TciType & getElementType () const =0;

virtual Tsize getLength () const =0;

virtual void setLength (Tsize p\_length)=0;

virtual Tindex getOffset() const =0;

virtual Tsize getPermutationCount () const =0;

virtual Permutation & getPermutation (Tindex p\_position) const =0;

virtual Permutation \* newPermutation () =0;

virtual void definePermutation (const Permutation & permutation) =0;

virtual void removePermutation (Tindex p\_position) =0;

virtual void clearPermutations () =0;

virtual Tboolean operator== (const RecordOfValue &p\_recOf) const =0;

virtual RecordOfValue \* clone () const =0;

virtual Tboolean operator< (const RecordOfValue &p\_recOf) const =0;

}

**Methods:**

~RecordOfValue

Destructor

getField

Return the field at the specified position

setField

Set the value at the specified position

appendField

Add a value at the end of the record of

getElementType

Return the type of the elements of this record of

getLength

Return the length of the object

setLength

Set length of the record of

getOffset

For arrays, return the lower index bound used in the type definition of arrays. Return 0 for record of and set of

getPermutationCount

Returns the number of permutations in the record of or array value

getPermutation

Returns the permutation at the specified index

getPermutation

Creates a new Permutation class instance

definePermutations

Creates permutation from existing elements of a record of value

removePermutation

Removes the permutation at the specified index

clearPermutations

Removes all permutations from the value

operator==

Returns true if both objects are equal

clone

Return a copy of the RecordOfValue

operator<

Operator < overload

#### 10.5.3.13 UnionValue

TTCN-3 union value support. It is mapped to the following pure virtual class:

class UnionValue : public virtual TciValue {

public:

virtual ~UnionValue ()

virtual void setVariant (const Tstring &p\_variantName, const TciValue &p\_value)=0;

virtual TciValue & getVariant (const Tstring &p\_variantName) =0;

virtual const Tstring & getPresentVariantName () const =0;

virtual const std::set< Tstring \*> & getVariantNames () const =0;

virtual Tboolean operator== (const UnionValue & p\_unionVal) const =0;

virtual UnionValue \* clone () const =0;

virtual Tboolean operator< (const UnionValue & p\_unionVal) const =0;

}

**Methods:**

~UnionValue

Destructor

setVariant

Set the variant name to a value

getVariant

Return the value of the variant if exits

getPresentVariantName

Return the name of the current variant value. null if no initialized

getVariantNames

Return a list which contains the variant names as defined in the TTCN-3 module

operator==

Returns true if both objects are equal

clone

Return a copy of the UnionValue

operator<

Operator < overload

#### 10.5.3.14 EnumeratedValue

TTCN-3 enumerated value support. It is mapped to the following pure virtual class:

class EnumeratedValue : public virtual TciValue {

public:

virtual ~EnumeratedValue ();

virtual const Tstring & getEnum () const =0;

virtual void setEnum (const Tstring &p\_value)=0;

virtual Tinteger getInt() const =0;

virtual void setInt(Tinteger p\_int);

virtual Tboolean operator== (const EnumeratedValue & p\_enumVal) const =0;

virtual EnumeratedValue \* clone () const =0;

virtual Tboolean operator< (const EnumeratedValue & p\_enumVal) const =0;

}

**Methods:**

~EnumeratedValue

Destructor

getEnum

Return current value

setEnum

Set the enumeration value

getInt

Return current integer value

setInt

Set the integer value

operator==

Returns true if both objects are equal

clone

Return a copy of the EnumeratedValue

operator<

Operator < overload

#### 10.5.3.15 VerdictValue

TTCN-3 verdict value support. It is mapped to the following pure virtual class:

class VerdictValue : public virtual TciValue {

public:

virtual ~VerdictValue ();

virtual const VerdictValueEnum & getVerdict () const =0;

virtual void setVerdict (const VerdictValueEnum & p\_enum)=0;

virtual Tboolean operator== (const VerdictValue & p\_verdictVal) const =0;

virtual VerdictValue \* clone () const =0;

virtual Tboolean operator< (const VerdictValue & p\_verdictVal) const =0;

}

**Methods:**

~VerdictValue

Destructor

getVerdict

Return the value of the verdict

setVerdict

Set the value of the verdict

operator==

Returns true if both objects are equal

clone

Return a copy of the VerdictValue

operator<

Operator < overload

#### 10.5.3.16 VerdictValueEnum

Defines verdict values as an enumeration:

typedef enum

{

NONE = 0,

PASS = 1,

FAIL = 2,

INCONC = 3,

ERROR = 4,

USER\_ERROR = 5

} VerdictValueEnum;

#### 10.5.3.17 AddressValue

TTCN-3 address value support. It is mapped to the following pure virtual class:

class AddressValue {

public:

virtual ~AddressValue ();

virtual TciValue & getAddress ()=0;

virtual void setAddress ( const TciValue& T)=0;

virtual Tboolean operator== (const AddressValue & p\_addr) const =0;

virtual AddressValue \* cloneAddressValue () const =0;

virtual Tboolean operator< (const AddressValue & p\_addr) const =0;

}

**Methods:**

~AddressValue

Destructor

getAddress

Return the value of the address

setAddress

Set the value of the address

operator==

Returns true if both objects are equal

clone

Return a copy of the AddressValue

operator<

Operator < overload

#### 10.5.3.18 MatchingMechanism

Represents a TTCN-3 matching mechanism. It is mapped to the following pure virtual class:

class MatchingMechanism : public virtual TciValue {

public:

virtual ~MatchingMechanism ();

virtual TciMatchingType getMatchingType () const =0;

virtual Tboolean operator== (const MatchingMechanism &p\_template) const =0;

virtual MatchingMechanism \* clone () const =0;

virtual Tboolean operator< (const MatchingMechanism &p\_template) const =0;

}

**Methods:**

~MatchingMechanism

Destructor

getMatchingType

Returns the matching mechanism type

operator==

Returns true if both objects are equal

clone

Return a copy of the MatchingMechanism

operator<

Operator < overload

#### 10.5.3.19 MatchingList

Represents the following TTCN-3 matching mechanisms: template list, complemented template list, subset, superset. It is mapped to the following pure virtual class:

class MatchingList : public virtual MatchingMechanism {

public:

virtual ~MatchingList ();

virtual Tsize size () const =0;

virtual TciValue& get (Tindex p\_position) const =0;

virtual void add (TciValue& p\_val) =0;

virtual void remove (Tindex p\_position) =0;

virtual void clear () =0;

virtual Tboolean operator== (const MatchingList &p\_list) const =0;

virtual MatchingList \* clone () const =0;

virtual Tboolean operator< (const MatchingList &p\_list) const =0;

}

**Methods:**

~MatchingList

Destructor

size

Returns the number of items in the matching list

get

Returns the value or template at the specified position inside the matching list

add

Adds a new item to the end of the matching list, increasing its size

remove

Removes the value or template at the specified position inside the matching list

clear

Removes all values and templates from the matching list

operator==

Returns true if both objects are equal

clone

Return a copy of the MatchingList

operator<

Operator < overload

#### 10.5.3.20 ValueRange

TTCN-3 value range support. It is mapped to the following pure virtual class:

class ValueRange : public virtual MatchingMechanism {

public:

virtual ~ValueRange ();

virtual RangeBoundary & getLowerBoundary () =0;

virtual RangeBoundary & getUpperBoundary () =0;

virtual void setLowerBoundary (const RangeBoundary & p\_boundary) =0;

virtual void setUpperBoundary (const RangeBoundary & p\_boundary) =0;

virtual Tboolean operator== (const ValueRange &p\_range) const =0;

virtual ValueRange \* clone () const =0;

virtual Tboolean operator< (const ValueRange &p\_range) const =0;

}

**Methods:**

~ValueRange

Destructor

getLowerBoundary

Returns the lower boundary of the range

getUpperBoundary

Returns the upper boundary of the range

setLowerBoundary

Sets the lower boundary of the range

setUpperBoundary

Sets the upper boundary of the range

operator==

Returns true if both objects are equal

clone

Return a copy of the ValueRange

operator<

Operator < overload

#### 10.5.3.21 CharacterPattern

TTCN-3 pattern support. It is mapped to the following pure virtual class:

class CharacterPattern : public virtual MatchingMechanism {

public:

virtual ~CharacterPattern ();

virtual TciValue & getPatternString () =0;

virtual void setPatternString (const TciValue & p\_string) =0;

virtual Tboolean operator== (const CharacterPattern &p\_pattern) const =0;

virtual CharacterPattern \* clone () const =0;

virtual Tboolean operator< (const CharacterPattern &p\_pattern) const =0;

}

**Methods:**

~CharacterPattern

Destructor

getPatternString

Returns the character pattern definition of this pattern

setPatternString

Sets the character pattern definition of this pattern

operator==

Returns true if both objects are equal

clone

Return a copy of the CharacterPattern

operator<

Operator < overload

#### 10.5.3.22 MatchDecodedContent

TTCN-3 MatchDecodedContent support. It is mapped to the following pure virtual class:

class MatchDecodedContent : public virtual MatchingMechanism {

public:

virtual ~MatchDecodedContent ();

virtual TciValue & getContent () =0;

virtual void setContent (const TciValue & p\_content) =0;

virtual Tboolean operator== (const MatchDecodedContent &p\_content) const =0;

virtual MatchDecodedContent \* clone () const =0;

virtual Tboolean operator< (const MatchDecodedContent &p\_content) const =0;

}

**Methods:**

~MatchDecodedContent

Destructor

getContent

Returns the value or matching mechanism used as an argument of the decmatch matching mechanism

setContent

Sets the value or matching mechanism used as an argument of the decmatch matching mechanism

operator==

Returns true if both objects are equal

clone

Return a copy of the MatchDecodedContent template

operator<

Operator < overload

### 10.5.4 Abstract logging types

#### 10.5.4.1 TciValueTemplate

Interface that defines the concrete operations of the TTCN-3 template. It is mapped to the following pure virtual class:

class TciValueTemplate {

public:

virtual ~TciValueTemplate ();

virtual Tboolean isOmit () const =0;

virtual Tboolean isAny () const =0;

virtual Tboolean isAnyOrOmit () const =0;

virtual const Tstring & getTemplateDef () const =0;

virtual Tboolean operator== (const TciValueTemplate &vtempl) const =0;

virtual TciValueTemplate \* clone () const =0;

virtual Tboolean operator< (const TciValueTemplate &vtempl) const =0;

}

**Methods:**

~TciValueTemplate ()

Destructor

isOmit ()

Return true if value of template is omit

isAny ()

Return true if value of template is any

isAnyOrOmit ()

Return true value of template if any or omit

getTemplateDef ()

Return the template definition as defined in the TTCN-3 module

operator== (const TciValueTemplate &vtempl)

Returns true if both objects are equal

clone ()

Return a copy of the TciValueTemplate

operator< (const TciValueTemplate &vtempl)

Operator < overload

#### 10.5.4.2 TciNonValueTemplate

Support *all* and *any* matching mechanisms over TTCN-3 components and timers. It is mapped to the following pure virtual class:

class TciNonValueTemplate {

public:

virtual ~TciNonValueTemplate ();

virtual Tboolean isAny () const =0;

virtual Tboolean isAll () const =0;

virtual const Tstring & getTemplateDef () const =0;

virtual Tboolean operator== (const TciNonValueTemplate &nvtempl) const =0;

virtual TciNonValueTemplate \* clone () const =0;

virtual Tboolean operator< (const TciNonValueTemplate &nvtempl) const =0;

}

**Methods:**

~TciNonValueTemplate ()

Destructor

isAny ()

Return true if value is any

isAll ()

Return true if is value all

getTemplateDef ()

Return template definition as defined in the TTCN-3 module

operator== (const TciNonValueTemplate &nvtempl)

Returns true if both objects are equal

clone ()

Return a copy of the TciNonValueTemplate

operator< (const TciNonValueTemplate &nvtempl)

Operator < overload

#### 10.5.4.3 TciValueList

A list of TciValues. It is mapped to the following pure virtual class:

class TciValueList {

public:

virtual ~TciValueList (void);

virtual Tsize size () const =0;

virtual Tboolean empty () const =0;

virtual const TciValue \*get (Tindex index) const =0;

virtual void clear ()=0;

virtual void add (const TciValue &comp)=0;

virtual Tboolean operator== (const TciValueList &valList) const =0;

virtual TciValueList \* clone () const =0;

virtual Tboolean operator< (const TciValueList &valList) const =0;

}

**Methods:**

~TciValueList ()

Destructor

size ()

Return the size of the list

empty ()

Return true if the list is empty

get (Tindex index)

Return the value at the specified position

clear ()

Remove all the elements from this list

add (const TciValue &comp)

Add an element to the end of this list

operator== (const TciValueList &valList)

Returns true if both objects are equal

clone ()

Return a copy of the TciValueList

operator< (const TciValueList &valList)

Operator < overload

#### 10.5.4.4 TciValueDifference

Represents the differences during a match operation. It is mapped to the following pure virtual class:

class TciValueDifference {

public:

virtual ~TciValueDifference ();

virtual const TciValue & getValue () const =0;

virtual void setValue (TciValue &val)=0;

virtual const TciValueTemplate & getTciValueTemplate () const =0;

virtual void setTciValueTemplate (TciValueTemplate &valT)=0;

virtual const Tstring & getDescription () const =0;

virtual void setDescription (const Tstring &descr)=0;

virtual Tboolean operator== (const TciValueDifference &vdiff) const =0;

virtual TciValueDifference \* clone () const =0;

virtual Tboolean operator< (const TciValueDifference &vdiff) const =0;

}

**Methods:**

~TciValueDifference ()

Destructor

getValue ()

Return the value definition

setValue (TciValue &val)

Set the value definition

getTciValueTemplate ()

Return the template definition

setTciValueTemplate (TciValueTemplate &valT)

Set the template definition

getDescription ()

Return a string which describes the difference

setDescription (const Tstring &descr)

Set description

operator== (const TciValueDifference &vdiff)

Returns true if both objects are equal

clone ()

Return a copy of the TciValueDifference

operator< (const TciValueDifference &vdiff)

Operator < overload

#### 10.5.4.5 TciValueDifferenceList

Collection of TciValueDifferences. It is mapped to the following pure virtual class:

class TciValueDifferenceList {

public:

virtual ~TciValueDifferenceList ();

virtual Tsize size () const =0;

virtual Tboolean empty () const =0;

virtual const TciValueDifference \*get (Tindex p\_position) const =0;

virtual void clear ()=0;

virtual void add (const TciValueDifference &comp)=0;

virtual Tboolean operator== (const TciValueDifferenceList &vdList) const =0;

virtual TciValueDifferenceList \* clone () const =0;

virtual Tboolean operator< (const TciValueDifferenceList &vdList) const =0;

}

**Methods:**

~TciValueDifferenceList ()

Destructor

size ()

Return the size of the list

empty ()

Return true if this list contains no elements

get (Tindex p\_position)

Return the requested difference

clear ()

Remove all the components from this list

add (const TciValueDifference &comp)

Add a component to the end of the list

operator== (const TciValueDifferenceList &vdList)

Returns true if both objects are equal

clone ()

Return a copy of the TciValueDifferenceList

operator< (const TciValueDifferenceList &vdList)

Operator < overload

#### 10.5.4.6 ComponentStatus

Defines component status as an enumeration:

typedef enum

{

INACTIVE\_C = 0,

RUNNING\_C = 1,

STOPPED\_C = 2,

KILLED\_C = 3

NULL\_C = 4

} ComponentStatus;

#### 10.5.4.7 TimerStatus

Defines timer status as an enumeration:

typedef enum

{

RUNNING\_T = 0,

INACTIVE\_T = 1,

EXPIRED\_T = 2

NULL\_T = 3

} TimerStatus;

#### 10.5.4.8 TciStatus

Defines TCI status as an enumeration:

typedef enum

{

TCI\_OK = 0,

TCI\_ERROR = -1

} TciStatus;

## 10.6 Operations mapping

### 10.6.1 TCI-TM

#### 10.6.1.1 TciTmRequired

Specifies the operations the TM requires from TE. It is mapped to the following interface:

//Destructor

virtual ~TciTmRequired ();

//Selects the indicated module for execution

virtual void tciRootModule (const TciModuleId \*moduleName)=0;

//The TE provides to the management a list of imported modules of the root module

virtual const TciModuleIdList \* getImportedModules () const =0;

//The TE provides to the management a list of module parameters of the identified module

virtual const TciModuleParameterList \* tciGetModuleParameters (const TciModuleId \*moduleName)=0;

//The TE provides to the management a list of test cases

virtual const TciTestCaseIdList \* tciGetTestCases () const =0;

//The TE provides to the management a list of parameter types of the given test case

virtual const TciParameterTypeList \* tciGetTestCaseParameters (const TciTestCaseId \*testCaseId) const =0;

//The TE provides to the management a list of system ports of the given test case

virtual const TriPortIdList \* tciGetTestCaseTSI (const TciTestCaseId &testCaseId) const =0;

//Starts a testcase in the currently selected module with the given parameters

virtual void tciStartTestCase (const TciTestCaseId \*testCaseId, const TciParameterList \*parameterList)=0;

//Stops the testcase currently being executed

virtual void tciStopTestCase ()=0;

//Starts the module control function of the selected module

virtual const TriComponentId \* tciStartControl ()=0;

//Stops execution of the control part

virtual void tciStopControl ()=0;

//The TE provides to the management a list of parameter types of the module control function

virtual const TciParameterTypeList \* tciGetControlParameters () const =0;

//Starts the module control function of the selected module with additional parameters

virtual const TriComponentId \* tciStartControlWithParameters ()=0;

#### 10.6.1.2 TciTmProvided

Specifies the operation the TM has to provide to the TE. It is mapped to the following interface:

//Destructor

virtual ~TciTmProvided ();

//Indicates to the TM that a test case with testCaseId has been started

virtual void tciTestCaseStarted (const TciTestCaseId &testCaseId, const TciParameterList &parameterList, const Tfloat &timer)=0;

//Called to indicate that the test case has terminated execution

virtual void tciTestCaseTerminated (const VerdictValue &verdict, const TciParameterList &parameterList)=0;

//Called to indicate that the control function of the selected module has just terminated execution

virtual void tciControlTerminated ()=0;

//The management provides to the TE a Value for the indicated parameterId

virtual TciValue \* tciGetModulePar (const TciModuleParameterId &parameterId)=0;

//Indicates the occurrence of an unrecoverable error situation

virtual void tciError (const Tstring &message)=0;

//The TE indicates a message of a test component

virtual void tciLog (const TriComponentId &testComponentId, const Tstring &message)=0;

//Called to pass the result of the module control function execution to the TM

virtual void tciControlTerminatedWithResult (const TciValue &result, const TciParameterList &parameterList)=0;

### 10.6.2 TCI-CD

#### 10.6.2.1 TciCdRequired

This class defines the TCI\_CD required interface. It is mapped to the following interface:

//Destructor

virtual ~TciCdRequired ();

//Returns a type representing a ttcn type

virtual const TciType \* getTypeForName (const Tstring typeName) const =0;

//Constructs and returns a basic TTCN-3 integer type

virtual const TciType & getInteger () const =0;

//Constructs and returns a basic TTCN-3 float type

virtual const TciType & getFloat () const =0;

//Constructs and returns a basic TTCN-3 boolean type

virtual const TciType & getBoolean () const =0;

//Constructs and returns a basic TTCN-3 charstring type

virtual const TciType & getCharstring () const =0;

//Constructs and returns a basic TTCN-3 universal charstring type

virtual const TciType & getUniversalCharstring () const =0;

//Constructs and returns a basic TTCN-3 hexstring type

virtual const TciType & getHexstring () const =0;

//Constructs and returns a basic TTCN-3 bitstring type

virtual const TciType & getBitstring () const =0;

//Constructs and returns a basic TTCN-3 octetstring type

virtual const TciType & getOctetstring () const =0;

//Constructs and returns a basic TTCN-3 verdict type

virtual const TciType & getVerdict () const =0;

//The TE will be notified about an unrecoverable error situation within the CD

virtual void tciErrorReq (const Tstring message)=0;

#### 10.6.2.2 TciCdProvided

This class defines the TCI\_CD provided interface. It is mapped to the following interface:

//Destructor

virtual ~TciCdProvided ();

//This operation is called whenever the TE has to implicitly decode a value

virtual TciValue \* decode (const TriMessage \*p\_message, const TciType \*p\_decodingHypothesis)=0;

//This operation is called whenever the TE has to implicitly encode a value

virtual TriMessage \* encode (const TciValue \*p\_value)=0;

//This operation is called whenever the TE invokes decvalue

virtual Tinteger decodeValue (TriMessage \*p\_message, const TciType \*p\_decodingHypothesis,

const TuniversalString & decodingInfo, TciValue \*\* decodedValue )=0;

//This operation is called whenever the TE invokes encvalue

virtual TriMessage \* encodeValue (const TciValue \*p\_value, const TuniversalString & encodingInfo)=0;

### 10.6.3 TCI-CH

#### 10.6.3.1 TciChRequired

This class defines the TCI\_CH required interface. It is mapped to the following interface:

//Default destructor

virtual ~TciChRequired ();

//This operation is called by the CH at the local TE when at a remote TE a provided //tciSendConnected has been called

virtual void tciEnqueueMsgConnected (const TriPortId \*sender, const TriComponentId \*receiver,  
 const TciValue \*rcvdMessage)=0;

//This operation is called by the CH at the local TE when at a remote TE a provided //tciCallConnected has been called

virtual void tciEnqueueCallConnected (const TriPortId \*sender, const TriComponentId \*receiver,  
 const TriSignatureId \*signature, const TciParameterList \*parameterList)=0;

//This operation is called by the CH at the local TE when at a remote TE a provided //tciReplyConnected has been called

virtual void tciEnqueueReplyConnected (const TriPortId \*sender, const TriComponentId \*receiver,   
 const TriSignatureId \*signature, const TciParameterList \*parameterList,  
 const TciValue \*returnValue)=0;

//This operation is called by the CH at the local TE when at a remote TE a provided //tciRaiseConnected has been called

virtual void tciEnqueueRaiseConnected (const TriPortId \*sender, const TriComponentId \*receiver,  
 const TriSignatureId \*signature, const TciValue \*exception)=0;

//This operation is called by the CH at the local TE when at a remote TE a provided //tciCreateTestComponentReq has been called

virtual const TriComponentId \* tciCreateTestComponent (const TciTestComponentKind \*kind,  
 const TciType \*componentType, const Tstring \*name)=0;

//This operation is called by the CH at the local TE when at a remote TE a provided //tciStartTestComponentReq has been called

virtual void tciStartTestComponent (const TriComponentId \*component,  
 const TciBehaviourId \*behaviour, const TciParameterList \*parameterList)=0;

//This operation is called by the CH at the local TE when at a remote TE a provided //tciStopTestComponentReq has been called

virtual void tciStopTestComponent (const TriComponentId \*component)=0;

//This operation is called by the CH at the local TE when at a remote TE a provided tciConnect

//has been called

virtual void tciConnect (const TriPortId \*fromPort, const TriPortId \*toPort)

//This operation is called by the CH at the local TE when at a remote TE a provided

//tciDisconnect has been called

virtual void tciDisconnect (const TriPortId \*fromPort, const TriPortId \*toPort)=0;

//This operation is called by the CH at the local TE when at a remote TE a provided tciMapReq

//has been called

virtual void tciMap (const TriPortId \*fromPort, const TriPortId \*toPort)=0;

//This operation is called by the CH at the local TE when at a remote TE a provided

//tciMapParamReq has been called

virtual void tciMapParam (const TriPortId \*fromPort, const TriPortId \*toPort,  
 const TciParameterList \*parameterList)=0;

//This operation is called by the CH at the local TE when at a remote TE a provided tciUnmapReq //has been called

virtual void tciUnmap (const TriPortId \*fromPort, const TriPortId \*toPort)=0;

//This operation is called by the CH at the local TE when at a remote TE a provided

//tciUnmapParamReq has been called

virtual void tciUnmapParam (const TriPortId \*fromPort, const TriPortId \*toPort,  
 const TciParameterList \*parameterList)=0;

//This operation is called by the CH at the local TE when at a remote TE a provided   
//tciTestComponentTerminatedReq has been called

virtual void tciTestComponentTerminated (const TriComponentId \*component,  
 const VerdictValue \*verdict) const =0;

//This operation is called by the CH at the local TE when at a remote TE a provided //tciTestComponentRunningReq has been called

virtual Tboolean tciTestComponentRunning (const TriComponentId \*component) const =0;

//This operation is called by the CH at the local TE when at a remote TE a provided //tciTestComponentDoneReq has been called

virtual Tboolean tciTestComponentDone (const TriComponentId \*comp, VerdictValueEnum \* verdict)

const =0;

//This operation can be called by the CH at the appropriate local TE when at a remote TE a //provided tciGetMTCReq has been called

virtual const TriComponentId \* tciGetMTC () const =0;

//This operation is called by the CH at the appropriate local TE when at a remote TE a provided //tciExecuteTestCaseReq has been called

virtual void tciExecuteTestCase (const TciTestCaseId \*testCaseId,  
 const TriPortIdList \*tsiPortList)=0;

//This operation is called by the CH at appropriate local TEs when at a remote TE a provided //tciResetReq has been called

virtual void tciReset ()=0;

//This operation is called by the CH at the local TE when at a remote TE a provided //tciKillTestComponentReq has been called

virtual void tciKillTestComponent (const TriComponentId \*comp, VerdictValueEnum \* verdict)=0;

//This operation is called by the CH at the local TE when at a remote TE a provided //tciTestComponentAliveReq has been called

virtual Tboolean tciTestComponentAlive (const TriComponentId \*comp) const =0;

//This operation is called by the CH at the local TE when at a remote TE a provided //tciTestComponentKilledReq has been called

virtual Tboolean tciTestComponentKilled (const TriComponentId \*comp) const =0;

//This operation is called by the CH at the local TE when at a remote TE a provided //tciCallTestComponentReq has been called

virtual void tciCallTestComponent (const TriComponentId \*component,  
 const TciBehaviourId \*behaviour, const TciParameterList \*parameterList)=0;

//This operation is called by the CH at the local TE when at a remote TE a provided  
//tciTestComponentCallTerminatedReq has been called

virtual void tciTestComponentCallTerminated (const TriComponentId \*component,  
 const VerdictValue \*verdict, const TciParameterList \*parameterList,   
 const TciValue \* returnValue)=0;

#### 10.6.3.2 TciChProvided

This class defines the TCI\_CH provided interface. It is mapped to the following interface:

//Destructor

virtual ~TciChProvided ();

//Called by the TE when it executes a TTCN-3 unicast send operation on a component port

virtual void tciSendConnected (const TriPortId \*sender, const TriComponentId \*receiver,  
 const TciValue \*sendMessage)=0;

//Called by the TE when it executes a TTCN-3 broadcast send operation on a component port

virtual void tciSendConnectedBC (const TriPortId \*sender, const TciValue \*sendMessage)=0;

//Called by the TE when it executes a TTCN-3 multicast send operation on a component port

virtual void tciSendConnectedMC (const TriPortId \*sender, const TriComponentIdList \*receivers,  
 const TciValue \*sendMessage)=0;

//Called by the TE when it executes a TTCN-3 unicast call operation on a component port

virtual void tciCallConnected (const TriPortId \*sender, const TriComponentId \*receiver,  
 const TriSignatureId \*signature, const TciParameterList \*parameterList)=0;

//Called by the TE when it executes a TTCN-3 broadcast call operation on a component port

virtual void tciCallConnectedBC (const TriPortId \*sender, const TriSignatureId \*signature,  
 const TciParameterList \*parameterList)=0;

//Called by the TE when it executes a TTCN-3 multicast call operation on a component port

virtual void tciCallConnectedMC (const TriPortId \*sender, const TriComponentIdList \*receivers,  
 const TriSignatureId \*signature, const TciParameterList \*parameterList)=0;

//Called by the TE when it executes a TTCN-3 unicast reply operation on a component port

virtual void tciReplyConnected (const TriPortId \*sender, const TriComponentId \*receiver,  
 const TriSignatureId \*signature, const TciParameterList \*parameterList,  
 const TciValue \*returnValue)=0;

//Called by the TE when it executes a TTCN-3 broadcast reply operation on a component port

virtual void tciReplyConnectedBC (const TriPortId \*sender, const TriSignatureId \*signature,  
 const TciParameterList \*parameterList, const TciValue \*returnValue)=0;

//Called by the TE when it executes a TTCN-3 multicast reply operation on a component

virtual void tciReplyConnectedMC (const TriPortId \*sender, const TriComponentIdList \*receivers,  
 const TriSignatureId \*signature, const TciParameterList \*parameterList,  
 const TciValue \*returnValue)=0;

//Called by the TE when it executes a TTCN-3 unicast raise operation on a component port

virtual void tciRaiseConnected (const TriPortId \*sender, const TriComponentId \*receiver,  
 const TriSignatureId \*signature, const TciValue \*exception)=0;

//Called by the TE when it executes a TTCN-3 broadcast raise operation on a component portvirtual void tciRaiseConnectedBC (const TriPortId \*sender, const TriSignatureId \*signature,  
 const TciValue \*exception)=0;

//Called by the TE when it executes a TTCN-3 multicast raise operation on a component

virtual void tciRaiseConnectedMC (const TriPortId \*sender, const TriComponentIdList \*receiver,  
 const TriSignatureId \*signature, const TciValue \*exception)=0;

//Called from the TE when a component has to be created

virtual const TriComponentId \* tciCreateTestComponentReq (const TciTestComponentKind \*kind,  
 const QualifiedName \*componentType, const Tstring &name, const TciValue \*hostId)=0;

//Called by the TE when it executes the TTCN-3 start operation

virtual void tciStartTestComponentReq (const TriComponentId \*component,  
 const TciBehaviourId \*behaviour, const TciParameterList \*parameterList)=0;

//Called by the TE when it executes the TTCN-3 stop operation

virtual void tciStopTestComponentReq (const TriComponentId \*component)=0;

//Called by the TE when it executes a TTCN-3 connect operation

virtual void tciConnectReq (const TriPortId \*fromPort, const TriPortId \*toPort)=0;

//Called by the TE when it executes a TTCN-3 disconnect operation

virtual void tciDisconnectReq (const TriPortId \*fromPort, const TriPortId \*toPort)=0;

//Called by the TE when it executes a TTCN-3 map operation

virtual void tciMapReq (const TriPortId \*fromPort, const TriPortId \*toPort)=0;

//Called by the TE when it executes a TTCN-3 map operation including parameters

virtual void tciMapParamReq (const TriPortId \*fromPort, const TriPortId \*toPort,   
 const TciParameterList \*parameterList)=0;

//Called by the TE when it executes a TTCN-3 unmap operation

virtual void tciUnmapReq (const TriPortId \*fromPort, const TriPortId \*toPort)=0;

//Called by the TE when it executes a TTCN-3 unmap operation including parameters

virtual void tciUnmapParamReq (const TriPortId \*fromPort, const TriPortId \*toPort,  
 const TciParameterList \*parameterList)=0;

//Called by the TE when a test component terminates execution

virtual void tciTestComponentTerminatedReq (const TriComponentId \*component,  
 const VerdictValue \*verdict)=0;

//Called by the TE when it executes a TTCN-3 running operation

virtual Tboolean tciTestComponentRunningReq (const TriComponentId \*component) const =0;

//Called by the TE when it executes a TTCN-3 done operation

virtual Tboolean tciTestComponentDoneReq (const TriComponentId \*comp, VerdictValueEnum \* verdict)

const =0;

//Called by the TE when it executes a TTCN-3 mtc operation

virtual const TriComponentId \* tciGetMTCReq () const =0;

//Called by the TE immediately before it starts the test case behaviour on the MTC

virtual void tciExecuteTestCaseReq (const TciTestCaseId \*testCaseId,  
 const TriPortIdList \*tsiPortList)=0;

//Called by the TE at any time to reset the test system

virtual void tciResetReq ()=0;

//Called by the TE when it executes the TTCN-3 kill operation

virtual void tciKillTestComponentReq (const TriComponentId \*comp)=0;

//Called by the TE when it executes the TTCN-3 alive operation

virtual Tboolean tciTestComponentAliveReq (const TriComponentId \*comp) const =0;

//Called by the TE when it executes the TTCN-3 killed operation

virtual Tboolean tciTestComponentKilledReq (const TriComponentId \*comp, VerdictValueEnum \* verdict) const =0;

//Called by the TE when it starts executing the TTCN-3 call component operation

virtual void tciCallTestComponentReq (const TriComponentId \*component,  
 const TciBehaviourId \*behaviour, const TciParameterList \*parameterList)=0;

//Called by the TE when an execution of a test component call operation is finished

virtual void tciTestComponentCallTerminatedReq (const TriComponentId \*component,  
 const VerdictValue \*verdict, const TciParameterList \*parameterList,   
 const TciValue \* returnValue)=0;

### 10.6.4 TCI-TL

#### 10.6.4.1 TciTlProvided

This class defines the TCI\_TL provided Tinterface:

//Default constructor

TciTlProvided ();

// Destructor

virtual ~TciTlProvided ();

//Called by TE to log the execute test case request

virtual void tliTcExecute (const Tstring &am, const timeval ts, const Tstring src, const Tinteger line, const TriComponentId \*c, const TciTestCaseId \*tcId, const TciParameterList \*tciPars, const TriTimerDuration \*dur)=0;

//Called by TE to log the start of a testcase. This event occurs before the testcase is started

virtual void tliTcStart (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c, const TciTestCaseId \*tcId, const TciParameterList \*tciPars, const TriTimerDuration \*dur)=0;

//Called by TE to log the stop of a testcase

virtual void tliTcStop (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c, const TString &reason)=0;

//Called by TE to log the start of a testcase

virtual void tliTcStarted (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c, const TciTestCaseId \*tcId, const TciParameterList \*tciPars, const TriTimerDuration \*dur)=0;

//Called by TE to log the termination of a testcase

virtual void tliTcTerminated (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c, const TciTestCaseId \*tcId, const TciParameterList \*tciPars, const VerdictValue \*verdict, const TString &reason)=0;

//Called by TE to log the start of the control part

virtual void tliCtrlStart (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c)=0;

//Called by TE to log the stop of the control part. This event occurs after the control has //stopped. If the control is not represented by TRI component, c is null

virtual void tliCtrlStop (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c)=0;

//Called by TE to log the termination of the control part

virtual void tliCtrlTerminated (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c)=0;

//Called by TE to log a unicast send operation

virtual void tliMSend\_m (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c, const TriPortId \*at, const TriPortId \*to, const TciValue \*msgValue, const TriAddress \*address, const TciStatus \*encoderFailure, const TriMessage \*msg, const TriStatus \*transmissionFailure)=0;

//Called by TE to log a broadcast send operation

virtual void tliMSend\_m\_BC (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c, const TriPortId \*at, const TriPortId \*to, const TciValue \*msgValue, const TciStatus \*encoderFailure, const TriMessage \*msg, const TriStatus \*transmissionFailure)=0;

//Called by TE to log a multicast send operation

virtual void tliMSend\_m\_MC (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c, const TriPortId \*at, const TriPortId \*to, const TciValue \*msgValue, const TriAddressList \*addresses, const TciStatus \*encoderFailure, const TriMessage \*msg, const TriStatus \*transmissionFailure)=0;

//Called by TE to log a unicast send operation

virtual void tliMSend\_c (const Tstring &am, const timeval ts, const Tstring src, const Tinteger line, const TriComponentId \*c, const TriPortId \*at, const TriPortId \*to, const TciValue \*msgValue, const TriStatus \*transmissionFailure)=0;

//Called by TE to log a broadcast send operation

virtual void tliMSend\_c\_BC (const Tstring &am, const timeval ts, const Tstring src, const Tinteger line, const TriComponentId \*c, const TriPortId \*at, const TriPortIdList \*to, const TciValue \*msgValue, const TriStatus \*transmissionFailure)=0;

//Called by TE to log a multicast send operation

virtual void tliMSend\_c\_MC (const Tstring &am, const timeval ts, const Tstring src, const Tinteger line, const TriComponentId \*c, const TriPortId \*at, const TriPortIdList \*to, const TciValue \*msgValue, const TriStatus \*transmissionFailure)=0;

//Called by TE to log the enqueuing of a message

virtual void tliMDetected\_m (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c, const TriPortId \*at, const TriPortId \*from, const TriMessage \*msg, const TriAddress \*address)=0;

//Called by CH to log the enqueuing of a message

virtual void tliMDetected\_c (const Tstring &am, const timeval ts, const Tstring src, const Tinteger line, const TriComponentId \*c, const TriPortId \*at, const TriPortId \*from, const TciValue \*msgValue)=0;

//Called by TE to log the mismatch of a template

virtual void tliMMismatch\_m (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c, const TriPortId \*at, const TciValue \*msgValue, const TciValueTemplate \*msgTmpl, const TciValueDifferenceList \*diffs, const TciValue \*addrValue, const TciValueTemplate \*addressTmpl)=0;

//Called by TE to log the mismatch of a template

virtual void tliMMismatch\_c (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c, const TriPortId \*at, const TciValue \*msgValue, const TciValueTemplate \*msgTmpl, const TciValueDifferenceList \*diffs, const TriComponentId \*from, const TciNonValueTemplate \*fromTmpl)=0;

// Called by TE to log the receiving of a message

virtual void tliMReceive\_m (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c, const TriPortId \*at, const TciValue \*msgValue, const TciValueTemplate \*msgTmpl, const TciValue \*addrValue, const TciValueTemplate \*addressTmpl)=0;

//Called by TE to log the mismatch of a template

virtual void tliMReceive\_c (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c, const TriPortId \*at, const TciValue \*msgValue, const TciValueTemplate \*msgTmpl, const TriComponentId \*fromComp, const TciNonValueTemplate \*fromTmpl)=0;

//Called by TE to log a unicast call operation

virtual void tliPrCall\_m (const Tstring &am, const timeval ts, const Tstring src, const Tinteger line, const TriComponentId \*c, const TriPortId \*at, const TriPortId \*to, const TriSignatureId \*signature, const TciParameterList \*tciPars, const TriAddress \*address, const TciStatus \*encoderFailure, const TriParameterList \*triPars, const TriStatus \*transmissionFailure)=0;

//Called by TE to log a broadcast call operation

virtual void tliPrCall\_m\_BC (const Tstring &am, const timeval ts, const Tstring src, const Tinteger line, const TriComponentId \*c, const TriPortId \*at, const TriPortId \*to, const TriSignatureId \*signature, const TciParameterList \*tciPars, const TciStatus \*encoderFailure, const TriParameterList \*triPars, const TriStatus \*transmissionFailure)=0;

//Called by TE to log a multicast call operation

virtual void tliPrCall\_m\_MC (const Tstring &am, const timeval ts, const Tstring src, const Tinteger line, const TriComponentId \*c, const TriPortId \*at, const TriPortId \*to, const TriSignatureId \*signature, const TciParameterList \*tciPars, const TriAddressList \*addresses, const TciStatus \*encoderFailure, const TriParameterList \*triPars, const TriStatus \*transmissionFailure)=0;

//Called by TE to log a unicast call operation

virtual void tliPrCall\_c (const Tstring &am, const timeval ts, const Tstring src, const Tinteger line, const TriComponentId \*c, const TriPortId \*at, const TriPortId \*to, const TriSignatureId \*signature, const TciParameterList \*tciPars, const TriStatus \*transmissionFailure)=0;

//Called by TE to log a broadcast call operation

virtual void tliPrCall\_c\_BC (const Tstring &am, const timeval ts, const Tstring src, const Tinteger line, const TriComponentId \*c, const TriPortId \*at, const TriPortIdList \*to, const TriSignatureId \*signature, const TciParameterList \*tciPars, const TriStatus \*transmissionFailure)=0;

//Called by TE to log a multicast call operation

virtual void tliPrCall\_c\_MC (const Tstring &am, const timeval ts, const Tstring src, const Tinteger line, const TriComponentId \*c, const TriPortId \*at, const TriPortIdList \*to, const TriSignatureId \*signature, const TciParameterList \*tciPars, const TriStatus \*transmissionFailure)=0;

//Called by TE to log the getcall enqueue operation

virtual void tliPrGetCallDetected\_m (const Tstring &am, const timeval ts, const Tstring src, const Tinteger line, const TriComponentId \*c, const TriPortId \*at, const TriPortId \*from, const TriSignatureId \*signature, const TriParameterList \*triPars, const TriAddress \*address)=0;

//Called by TE to log the getcall enqueue operation

virtual void tliPrGetCallDetected\_c (const Tstring &am, const timeval ts, const Tstring src, const Tinteger line, const TriComponentId \*c, const TriPortId \*at, const TriPortId \*from, const TriSignatureId \*signature, const TciParameterList \*tciPars)=0;

//Called by TE to log the mismatch of a getcall

virtual void tliPrGetCallMismatch\_m (const Tstring &am, const timeval ts, const Tstring src, const Tinteger line, const TriComponentId \*c, const TriPortId \*at, const TriSignatureId \*signature, const TciParameterList \*tciPars, const TciValueTemplate \*parsTmpl, const TciValueDifferenceList \*diffs, const TciValue \*addrValue, const TciValueTemplate \*addressTmpl)=0;

//Called by TE to log the mismatch of a getcall

virtual void tliPrGetCallMismatch\_c (const Tstring &am, const timeval ts, const Tstring src, const Tinteger line, const TriComponentId \*c, const TriPortId \*at, const TriSignatureId \*signature, const TciParameterList \*tciPars, const TciValueTemplate \*parsTmpl, const TciValueDifferenceList \*diffs, const TriComponentId \*from, const TciValueTemplate \*fromTmpl)=0;

//Called by TE to log getting a call

virtual void tliPrGetCall\_m (const Tstring &am, const timeval ts, const Tstring src, const Tinteger line, const TriComponentId \*c, const TriPortId \*at, const TriSignatureId \*signature, const TciParameterList \*tciPars, const TciValueTemplate \*parsTmpl, const TciValue \*addrValue, const TciValueTemplate \*addressTmpl)=0;

//Called by TE to log getting a call

virtual void tliPrGetCall\_c (const Tstring &am, const timeval ts, const Tstring src, const Tinteger line, const TriComponentId \*c, const TriPortId \*at, const TriSignatureId \*signature, const TciParameterList \*tciPars, const TciValueTemplate \*parsTmpl, const TriComponentId \*from, const TciNonValueTemplate \*fromTmpl)=0;

//Called by TE to log a unicast reply operation

virtual void tliPrReply\_m (const Tstring &am, const timeval ts, const Tstring src, const Tinteger line, const TriComponentId \*c, const TriPortId \*at, const TriPortId \*to, const TriSignatureId \*signature, const TciParameterList \*tciPars, const TciValue \*replValue, const TriAddress \*address, const TciStatus \*encoderFailure, const TriParameterList \*triPars, const TriParameter \*repl, const TriStatus \*transmissionFailure)=0;

//Called by TE to log a broadcast reply operation

virtual void tliPrReply\_m\_BC (const Tstring &am, const timeval ts, const Tstring src, const Tinteger line, const TriComponentId \*c, const TriPortId \*at, const TriPortId \*to, const TriSignatureId \*signature, const TciParameterList \*tciPars, const TciValue \*replValue, const TciStatus \*encoderFailure, const TriParameterList \*triPars, const TriParameter \*repl, const TriStatus \*transmissionFailure)=0;

//Called by TE to log a multicast reply operation

virtual void tliPrReply\_m\_MC (const Tstring &am, const timeval ts, const Tstring src, const Tinteger line, const TriComponentId \*c, const TriPortId \*at, const TriPortId \*to, const TriSignatureId \*signature, const TciParameterList \*tciPars, const TciValue \*replValue, const TriAddressList \*addresses, const TciStatus \*encoderFailure, const TriParameterList \*triPars, const TriParameter \*repl, const TriStatus \*transmissionFailure)=0;

//Called by TE to log a unicast reply operation

virtual void tliPrReply\_c (const Tstring &am, const timeval ts, const Tstring src, const Tinteger line, const TriComponentId \*c, const TriPortId \*at, const TriPortId \*to, const TriSignatureId \*signature, const TciValue \*parsValue, const TciValue \*replValue, const TriStatus \*transmissionFailure)=0;

//Called by TE to log a broadcast reply operation

virtual void tliPrReply\_c\_BC (const Tstring &am, const timeval ts, const Tstring src, const Tinteger line, const TriComponentId \*c, const TriPortId \*at, const TriPortIdList \*to, const TriSignatureId \*signature, const TciValue \*parsValue, const TciValue \*replValue, const TriStatus \*transmissionFailure)=0;

//Called by TE to log a multicast reply operation

virtual void tliPrReply\_c\_MC (const Tstring &am, const timeval ts, const Tstring src, const Tinteger line, const TriComponentId \*c, const TriPortId \*at, const TriPortIdList \*to, const TriSignatureId \*signature, const TciValue \*parsValue, const TciValue \*replValue, const TriStatus \*transmissionFailure)=0;

//Called by TE to log the getreply enqueue operation

virtual void tliPrGetReplyDetected\_m (const Tstring &am, const timeval ts, const Tstring src, const Tinteger line, const TriComponentId \*c, const TriPortId \*at, const TriPortId \*from, const TriSignatureId \*signature, const TriParameterList \*triPars, const TriParameter \*repl, const TriAddress \*address)=0;

//Called by CH to log the getreply enqueue operation

virtual void tliPrGetReplyDetected\_c (const Tstring &am, const timeval ts, const Tstring src, const Tinteger line, const TriComponentId \*c, const TriPortId \*at, const TriPortId \*from, const TriSignatureId \*signature, const TciParameterList \*tciPars, const TciValue \*replValue)=0;

//Called by TE to log the mismatch of a getreply operation

virtual void tliPrGetReplyMismatch\_m (const Tstring &am, const timeval ts, const Tstring src, const Tinteger line, const TriComponentId \*c, const TriPortId \*at, const TriSignatureId \*signature, const TciParameterList \*tciPars, const TciValueTemplate \*parsTmpl, const TciValue \*replValue, const TciValueTemplate \*replyTmpl, const TciValueDifferenceList \*diffs, const TciValue \*addrValue, const TciValueTemplate \*addressTmpl)=0;

//Called by TE to log the mismatch of a getreply operation

virtual void tliPrGetReplyMismatch\_c (const Tstring &am, const timeval ts, const Tstring src, const Tinteger line, const TriComponentId \*c, const TriPortId \*at, const TriSignatureId \*signature, const TciParameterList \*tciPars, const TciValueTemplate \*parsTmpl, const TciValue \*replValue, const TciValueTemplate \*replyTmpl, const TciValueDifferenceList \*diffs, const TriComponentId \*from, const TciNonValueTemplate \*fromTmpl)=0;

//Called by TE to log getting a reply

virtual void tliPrGetReply\_m (const Tstring &am, const timeval ts, const Tstring src, const Tinteger line, const TriComponentId \*c, const TriPortId \*at, const TriSignatureId \*signature, const TciParameterList \*tciPars, const TciValueTemplate \*parsTmpl, const TciValue \*replValue, const TciValueTemplate \*replyTmpl, const TciValue \*addrValue, const TciValueTemplate \*addressTmpl)=0;

//Called by TE to log getting a reply

virtual void tliPrGetReply\_c (const Tstring &am, const timeval ts, const Tstring src, const Tinteger line, const TriComponentId \*c, const TriPortId \*at, const TriSignatureId \*signature, const TciParameterList \*tciPars, const TciValueTemplate \*parsTmpl, const TciValue \*replValue, const TciValueTemplate \*replyTmpl, const TriComponentId \*from, const TciNonValueTemplate \*fromTmpl)=0;

//Called by TE to log a unicast raise operation

virtual void tliPrRaise\_m (const Tstring &am, const timeval ts, const Tstring src, const Tinteger line, const TriComponentId \*c, const TriPortId \*at, const TriPortId \*to, const TriSignatureId \*signature, const TciParameterList \*tciPars, const TciValue \*excValue, const TriAddress \*address, const TriStatus \*encoderFailure, const TriException \*exc, const TriStatus \*transmissionFailure)=0;

//Called by TE to log a broadcast raise operation

virtual void tliPrRaise\_m\_BC (const Tstring &am, const timeval ts, const Tstring src, const Tinteger line, const TriComponentId \*c, const TriPortId \*at, const TriPortId \*to, const TriSignatureId \*signature, const TciParameterList \*tciPars, const TciValue \*excValue, const TriStatus \*encoderFailure, const TriException \*exc, const TriStatus \*transmissionFailure)=0;

//Called by TE to log a multicast raise operation

virtual void tliPrRaise\_m\_MC (const Tstring &am, const timeval ts, const Tstring src, const Tinteger line, const TriComponentId \*c, const TriPortId \*at, const TriPortId \*to, const TriSignatureId \*signature, const TciParameterList \*tciPars, const TciValue \*excValue, const TriAddressList \*addresses, const TriStatus \*encoderFailure, const TriException \*exc, const TriStatus \*transmissionFailure)=0;

//Called by TE to log a unicast raise operation

virtual void tliPrRaise\_c (const Tstring &am, const timeval ts, const Tstring src, const Tinteger line, const TriComponentId \*c, const TriPortId \*at, const TriPortId \*to, const TriSignatureId \*signature, const TciParameterList \*tciPars, const TciValue \*excValue, const TriStatus \*transmissionFailure)=0;

//Called by TE to log a broadcast raise operation

virtual void tliPrRaise\_c\_BC (const Tstring &am, const timeval ts, const Tstring src, const Tinteger line, const TriComponentId \*c, const TriPortId \*at, const TriPortIdList \*to, const TriSignatureId \*signature, const TciParameterList \*tciPars, const TciValue \*excValue, const TriStatus \*transmissionFailure)=0;

//Called by TE to log a multicast raise operation

virtual void tliPrRaise\_c\_MC (const Tstring &am, const timeval ts, const Tstring src, const Tinteger line, const TriComponentId \*c, const TriPortId \*at, const TriPortIdList \*to, const TriSignatureId \*signature, const TciParameterList \*tciPars, const TciValue \*excValue, const TriStatus \*transmissionFailure)=0;

//Called by TE to log the catch enqueue operation

virtual void tliPrCatchDetected\_m (const Tstring &am, const timeval ts, const Tstring src, const Tinteger line, const TriComponentId \*c, const TriPortId \*at, const TriPortId \*from, const TriSignatureId \*signature, const TriException \*exc, const TriAddress \*address)=0;

//Called by TE to log the catch enqueue operation

virtual void tliPrCatchDetected\_c (const Tstring &am, const timeval ts, const Tstring src, const Tinteger line, const TriComponentId \*c, const TriPortId \*at, const TriPortId \*from, const TriSignatureId \*signature, const TciValue \*excValue)=0;

//Called by TE to log the mismatch of a catch operation

virtual void tliPrCatchMismatch\_m (const Tstring &am, const timeval ts, const Tstring src, const Tinteger line, const TriComponentId \*c, const TriPortId \*at, const TriSignatureId \*signature, const TciValue \*excValue, const TciValueTemplate \*excTmpl, const TciValueDifferenceList \*diffs, const TciValue \*addrValue, const TciValueTemplate \*addressTmpl)=0;

//Called by TE to log the mismatch of a catch operation

virtual void tliPrCatchMismatch\_c (const Tstring &am, const timeval ts, const Tstring src, const Tinteger line, const TriComponentId \*c, const TriPortId \*at, const TriSignatureId \*signature, const TciValue \*excValue, const TciValueTemplate \*excTmpl, const TciValueDifferenceList \*diffs, const TriComponentId \*from, const TciNonValueTemplate \*fromTmpl)=0;

//Called by TE to log catching an exception

virtual void tliPrCatch\_m (const Tstring &am, const timeval ts, const Tstring src, const Tinteger line, const TriComponentId \*c, const TriPortId \*at, const TriSignatureId \*signature, const TciValue \*excValue, const TciValueTemplate \*excTmpl, const TciValue \*addrValue, const TciValueTemplate \*addressTmpl)=0;

//Called by TE to log catching an exception

virtual void tliPrCatch\_c (const Tstring &am, const timeval ts, const Tstring src, const Tinteger line, const TriComponentId \*c, const TriPortId \*at, const TriSignatureId \*signature, const TciValue \*excValue, const TciValueTemplate \*excTmpl, const TriComponentId \*from, const TciNonValueTemplate \*fromTmpl)=0;

//Called by TE to log the detection of a catch timeout

virtual void tliPrCatchTimeoutDetected (const Tstring &am, const timeval ts, const Tstring src, const Tinteger line, const TriComponentId \*c, const TriPortId \*at, const TriSignatureId \*signature)=0;

//Called by TE to log catching a timeout

virtual void tliPrCatchTimeout (const Tstring &am, const timeval ts, const Tstring src, const Tinteger line, const TriComponentId \*c, const TriPortId \*at, const TriSignatureId \*signature)=0;

//Called by TE to log the create component operation

virtual void tliCCreate (const Tstring &am, const timeval ts, const Tstring src, const Tinteger line, const TriComponentId \*c, const TriComponentId \*comp, const Tstring &name, const Tboolean alive)=0;

//Called by TE to log the start component operation

virtual void tliCStart (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c, const TriComponentId \*comp, const TciBehaviourId \*beh, const TciParameterList \*tciPars)=0;

//Called by TE to log the running component operation

virtual void tliCRunning (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c, const TriComponentId \*comp, const ComponentStatus status)=0;

//Called by TE to log the alive component operation

virtual void tliCAlive (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c, const TriComponentId \*comp, const ComponentStatus status)=0;

//Called by TE to log the stop component operation

virtual void tliCStop (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c, const TriComponentId \*comp)=0;

//Called by TE to log the kill component operation

virtual void tliCKill (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c, const TriComponentId \*comp)=0;

//Called by TE to log the mismatch of a done component operation

virtual void tliCDoneMismatch (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c, const TriComponentId \*comp, const TciNonValueTemplate \*compTmpl)=0;

//Called by TE to log the done component operation

virtual void tliCDone (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c, const TciNonValueTemplate \*compTmpl, const VerdictValue \* verdict)=0;

//Called by TE to log the mismatch of a killed component operation

virtual void tliCKilledMismatch (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c, const TciNonValueTemplate \*compTmpl)=0;

//Called by TE to log the killed component operation

virtual void tliCKilled (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c, const TciNonValueTemplate \*compTmpl, const VerdictValue \* verdict)=0;

//Called by TE to log the termination of a component

virtual void tliCTerminated (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c, const VerdictValue \*verdict, const TString &reason)=0;

//Called by TE to log the connect operation

virtual void tliPConnect (const Tstring &am, const timeval ts, const Tstring src, const Tinteger line, const TriComponentId \*c, const TriPortId \*port1, const TriPortId \*port2)=0;

//Called by TE to log the connect operation

virtual void tliPDisconnect (const Tstring &am, const timeval ts, const Tstring src, const Tinteger line, const TriComponentId \*c, const TriPortId \*port1, const TriPortId \*port2)=0;

//Called by TE to log the map operation

virtual void tliPMap (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c, const TriPortId \*port1, const TriPortId \*port2)=0;

//Called by TE to log the map operation including param

virtual void tliPMapParam (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c, const TriPortId \*port1, const TriPortId \*port2, const TciParameterList \*tciPars, const TriStatus \*encoderFailure,   
const TriParameterList \*triPars)=0

//Called by TE to log the unmap operation

virtual void tliPUnmap (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c, const TriPortId \*port1, const TriPortId \*port2)=0;

//Called by TE to log the unmap operation including param

virtual void tliPUnmapParam (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c, const TriPortId \*port1, const TriPortId \*port2, const TciParameterList \*tciPars, const TriStatus \*encoderFailure,   
const TriParameterList \*triPars)=0

//Called by TE to log the port clear operation

virtual void tliPClear (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c, const TriPortId \*port)=0;

//Called by TE to log the port start operation

virtual void tliPStart (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c, const TriPortId \*port)=0;

//Called by TE to log the port stop operation

virtual void tliPStop (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c, const TriPortId \*port)=0;

//Called by TE to log the port stop operation

virtual void tliPHalt (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c, const TriPortId \*port)=0;

//Called by TE to log the encode operation

virtual void tliEncode (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c, const TciValue \*val, const TciStatus \*encoderFailure, const TriMessage \*msg, const Tstring &codec)=0;

//Called by TE to log the decode operation

virtual void tliDecode (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c, const TriMessage \*msg, const TciStatus \*decoderFailure, const TciValue \*val, const Tstring &codec)=0;

//Called by TE to log the detection of a timeout

virtual void tliTTimeoutDetected (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c, const TriTimerId \*timer)=0;

//Called by TE to log a timeout mismatch

virtual void tliTTimeoutMismatch (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c, const TriTimerId \*timer, const TciNonValueTemplate \*timerTmpl)=0;

//Called by TE to log a timeout match

virtual void tliTTimeout (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c, const TriTimerId \*timer, const TciNonValueTemplate \*timerTmpl)=0;

//Called by TE to log the start of a timer

virtual void tliTStart (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c, const TriTimerId \*timer, const TriTimerDuration \*dur)=0;

//Called by TE to log the stop of a timer

virtual void tliTStop (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c, const TriTimerId \*timer, const TriTimerDuration \*dur)=0;

//Called by TE to log the reading of a timer

virtual void tliTRead (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c, const TriTimerId \*timer, const TriTimerDuration \*elapsed)=0;

//Called by TE to log the running timer operation

virtual void tliTRunning (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c, const TriTimerId \*timer, const TimerStatus status)=0;

//Called by TE to log the entering of a scope

virtual void tliSEnter (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c, const QualifiedName &name, const TciParameterList \*tciPars, const Tstring &kind)=0;

//Called by TE to log the leaving of a scope

virtual void tliSLeave (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c, const QualifiedName &name, const TciParameterList \*tciPars, const TciValue \*returnValue, const Tstring &kind)=0;

//Called by TE to log the modification of the value of a variable

virtual void tliVar (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c, const QualifiedName &name, const TciValue \*varValue)=0;

//Called by TE to log the value of a module parameter

virtual void tliModulePar (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c, const QualifiedName &name, const TciValue \*parValue)=0;

//Called by TE to log the value of a module parameter

virtual void tliGetVerdict (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c, const VerdictValue \*verdict)=0;

//Called by TE to log the setverdict operation

virtual void tliSetVerdict (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c, const VerdictValue \*verdict, const TString &reason)=0;

//Called by TE to log the TTCN-3 statement log

virtual void tliLog (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c, const Tstring \*log)=0;

//Called by TE to log entering an alt

virtual void tliAEnter (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c)=0;

//Called by TE to log leaving an alt

virtual void tliALeave (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c)=0;

//Called by TE to log the nomatch of an alt

virtual void tliANomatch (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c)=0;

//Called by TE to log repeating an alt

virtual void tliARepeat (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c)=0;

//Called by TE to log entering the default section

virtual void tliADefaults (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c)=0;

//Called by TE to log the activation of a default

virtual void tliAActivate (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c, const QualifiedName &name, const TciParameterList \*tciPars, const TciValue \*ref)=0;

//Called by TE to log the deactivation of a default

virtual void tliADeactivate (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c, const TciValue \*ref)=0;

//Called by TE to log entering an alt

virtual void tliAWait (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c)=0;

//Called by TE to log that the component executed an SUT action

virtual void tliAction (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c, const Tstring &action)=0;

//Called by TE to log that the component successfully executed a match operation

virtual void tliMatch (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c, const TciValue &expr, const TciValueTemplate &tmpl)=0;

//Called by TE to log that the component executed a match operation, and a mismatch occurred

virtual void tliMatchMismatch (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c, const TciValue &expr, const TciValueTemplate &tmpl, const TciValueDifferenceList &diffs)=0;

//Can be called by the TE to log additional information during test execution

virtual void tliInfo (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c, const Tinteger level, const Tstring &info)=0;

//Called by TE to log the checking of a message

virtual void tliMChecked\_m (const Tstring &am, const timeval ts, const Tstring &src,

const Tinteger line, const TriComponentId \*c, const TriPortId \*at,

const TciValue \*msgValue, const TciValueTemplate \*msgTmpl,

const TciValue \*address, const TciValueTemplate \*addressTmpl)=0;

//Called by CH to log the checking of a message

virtual void tliMChecked\_c (const Tstring &am, const timeval ts, const Tstring src,

const Tinteger line, const TriComponentId \*c, const TriPortId \*at,

const TciValue \*msgValue, const TciValueTemplate \*msgTmpl,

const TriComponentId \*from, const TciNonValueTemplate \*fromTmpl)=0;

//Called by TE to log checking of the getcall operation

virtual void tliPrGetCallChecked\_m (const Tstring &am, const timeval ts, const Tstring src,

const Tinteger line, const TriComponentId \*c, const TriPortId \*at, const TriSignatureId \*signature,

const TciParameterList \*tciPars, const TciValueTemplate \*parsTmpl,

const TciValue \*address, const TciValueTemplate \*addressTmpl)=0;

//Called by TE to log checking of the getcall operation

virtual void tliPrGetCallChecked\_c (const Tstring &am, const timeval ts, const Tstring src,

const Tinteger line, const TriComponentId \*c, const TriPortId \*at, const TriSignatureId \*signature,

const TciParameterList \*tciPars, const TciValueTemplate \*parsTmpl,

const TriComponentId \*from, const TciNonValueTemplate \*fromTmpl)=0;

//Called by TE to log checking of the getreply operation

virtual void tliPrGetReplyChecked\_m (const Tstring &am, const timeval ts, const Tstring src,

const Tinteger line, const TriComponentId \*c, const TriPortId \*at, const TriSignatureId \*signature,

const TciParameterList \*tciPars, const TciValueTemplate \*parsTmpl,

const TciValue \*replValue, const TciValueTemplate \*replyTmpl,

const TciValue \*address, const TciValueTemplate \*addressTmpl)=0;

//Called by CH to log checking of the getreply operation

virtual void tliPrGetReplyChecked\_c (const Tstring &am, const timeval ts, const Tstring src,

const Tinteger line, const TriComponentId \*c, const TriPortId \*at, const TriSignatureId \*signature,

const TciParameterList \*tciPars, const TciValueTemplate \*parsTmpl,

const TciValue \*replValue, const TciValueTemplate \*replyTmpl,

const TriComponentId \*from, const TciNonValueTemplate \*fromTmpl)=0;

//Called by TE to log checking of the catch operation

virtual void tliPrCatchChecked\_m (const Tstring &am, const timeval ts, const Tstring src,

const Tinteger line, const TriComponentId \*c, const TriPortId \*at, const TriSignatureId \*signature,

const TciValue \*excValue, const TciValueTemplate \*excTmpl,

const TciValue \*address, const TciValueTemplate \*addressTmpl)=0;

//Called by TE to log checking of the catch operation

virtual void tliPrCatchChecked\_c (const Tstring &am, const timeval ts, const Tstring src,

const Tinteger line, const TriComponentId \*c, const TriPortId \*at, const TriSignatureId \*signature,

const TciValue \*excValue, const TciValueTemplate \*excTmpl,

const TriComponentId \*from, const TciNonValueTemplate \*fromTmpl)=0;

//Called by TE to log the check any operation

virtual void tliCheckedAny\_m (const Tstring &am, const timeval ts, const Tstring &src,

const Tinteger line, const TriComponentId \*c, const TriPortId \*at,

const TciValue \*address, const TciValueTemplate \*addressTmpl)=0;

//Called by CH to log the check any operation

virtual void tliCheckedAny\_c (const Tstring &am, const timeval ts, const Tstring src,

const Tinteger line, const TriComponentId \*c, const TriPortId \*at,

const TriComponentId \*from, const TciNonValueTemplate \*fromTmpl)=0;

//Called by TE to log the mismatch in a check any operation

virtual void tliCheckAnyMismatch\_m (const Tstring &am, const timeval ts, const Tstring &src,

const Tinteger line, const TriComponentId \*c, const TriPortId \*at,

const TciValue \*addrValue, const TciValueTemplate \*addressTmpl)=0;

//Called by CH to log the mismatch in a check any operation

virtual void tliCheckAnyMismatch\_c (const Tstring &am, const timeval ts, const Tstring src,

const Tinteger line, const TriComponentId \*c, const TriPortId \*at,

const TriComponentId \*from, const TciNonValueTemplate \*fromTmpl)=0;

//Called by TE to log the generation of a random number

virtual void tliRnd (const Tstring &am, const timeval ts, const Tstring src, const Tinteger line, const TriComponentId \*c, const FloatValue \*val, const FloatValue \*seed)=0;

//Called by TE to log evaluation of a @fuzzy or @lazy template or variable

virtual void tliEvaluate (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c, const QualifiedName &name, const TciValue \*evalResult)=0;

//Called by TE to log the component call operation

virtual void tliCCall (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c, const TriComponentId \*comp, const TciBehaviourId \*beh, const TciParameterList \*tciPars)=0;

//Called by TE to log the end of a component call

virtual void tliCCallTerminated (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c, const VerdictValue \*verdict, const TString &reason,  
TciParameterList \*tciPars, const TciValue \*returnValue)=0;

//Called by TE to log the start of a parameterized module control function

virtual void tliCtrlStartWithParameters (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c, const TciParameterList \*tciPars)=0;

//Called by TE to log the termination of a parameterized module control function

virtual void tliCtrlTerminatedWithResult (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c, const TciValue \*val, const TciParameterList \*tciPars)=0;

# 11 W3C XML mapping

## 11.1 Introduction

This clause introduces the TCI XML mapping [10], [11] and [12] for the logging interface of TCI. The XML mapping for the logging interface defines how the IDL definitions for TCI-TL described in clause 7 are mapped to XML. The complete schema definitions for this mapping are given in annex B.

## 11.2 Scopes

The IDL module **tciInterface** is mapped to an XML schema with the name space http://uri.etsi.org/ttcn‑3/tci/TLI\_v4\_10\_1.xsd.

This schema uses further schemas:

* http://uri.etsi.org/ttcn‑3/tci/SimpleTypes\_v4\_10\_1.xsd  
   for the mapping of simple types to XML.
* http://uri.etsi.org/ttcn‑3/tci/Types\_v4\_10\_1.xsd   
   for the mapping of structured types to XML.
* http://uri.etsi.org/ttcn‑3/tci/Values\_v4\_10\_1.xsd  
   for the mapping of values to XML.
* http://uri.etsi.org/ttcn‑3/tci/Templates\_v4\_10\_1.xsd  
   for the mapping of templates to XML.
* http://uri.etsi.org/ttcn‑3/tci/Events\_v4\_10\_1.xsd  
   for the mapping of logging events to XML.

## 11.3 Type mapping

### 11.3.1 Mapping of simple types

#### 11.3.1.1 TBoolean

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

#### 11.3.1.2 TString

The IDL **TString** type is mapped to the xsd basic type string.

#### 11.3.1.3 TInteger

The IDL **TInteger** type is mapped to the xsd basic type integer.

#### 11.3.1.4 TriTimerDurationType

The IDL **TriTimerDurationType** type is mapped to the xsd basic type float.

#### 11.3.1.5 TciParameterPassingModeType

The IDL **TciParameterPassingModeType** type is mapped to the xsd basic type string with enumeration values "in", "out" and "inout".

#### 11.3.1.6 TriStatusType

The IDL **TriStatusType** type is mapped to the xsd basic type string with enumeration values "TRI\_Ok" and "TRI\_Error".

#### 11.3.1.7 TciStatusType

The IDL **TriStatusType** type is mapped to the xsd basic type string with enumeration values "TCI\_Ok" and "TCI\_Error".

#### 11.3.1.8 ComponentStatusType

The IDL **ComponentStatusType** type is mapped to the xsd basic type string with enumeration values "inactiveC", "runningC", "stoppedC", "killedC" and "nullC".

#### 11.3.1.9 TimerStatusType

The IDL **TimerStatusType** type is mapped to the xsd basic type string with enumeration values "runningT", "inactiveT", "expiredT", and "nullT".

#### 11.3.1.10 PortStatusType

The IDL **PortStatusType** type is mapped to the xsd basic type string with enumeration values "startedP", "haltedP" and "stoppedP".

### 11.3.2 Complex type mapping

#### 11.3.2.1 TriPortIdType

**TriPortIdType** is mapped to the following complex type:

<xsd:complexType name="TriPortIdType">

<xsd:sequence>

<xsd:element name="comp" type="Types:TriComponentIdType" />

<xsd:element name="port" type="Types:Port"/>

</xsd:sequence>

</xsd:complexType>

**Elements:**

* comp The TRI component identifier.
* port The identification of the port.

**Attributes:**

* none.

#### 11.3.2.2 TriComponentIdType

**TriComponentIdType** is mapped to the following complex type:

<xsd:complexType name="TriComponentIdType">

<xsd:sequence>

<xsd:choice>

<xsd:element name="null" type="Templates:null"/>

<xsd:element name="id" type="Types:Id"/>

</xsd:choice>

</xsd:sequence>

</xsd:complexType>

**Elements:**

* id The identifier of the TRI component.
* null The null identifier. To be used if there is no TRI component identifier.

**Attributes:**

* none.

#### 11.3.2.3 TriComponentIdListType

**TriComponentIdListType** is mapped to the following complex type:

<xsd:complexType name="TriComponentIdListType">

<xsd:sequence>

<xsd:element name="comp" type="Types:TriComponentIdType" minOccurs="0"

maxOccurs="unbounded"/>

</xsd:sequence>

</xsd:complexType>

**Elements:**

* comp The identifiers of TRI components in that list.

**Attributes:**

* none.

#### 11.3.2.4 Port

**Port** is mapped to the following complex type:

<xsd:complexType name="Port">

<xsd:sequence>

<xsd:element name="id" type="Types:Id"/>

<xsd:element name="index" type="xsd:int"/>

</xsd:sequence>

</xsd:complexType>

**Elements:**

* id The port identifier.
* port The port index.

**Attributes:**

* none.

#### 11.3.2.5 Id

**Id** is used as identification for components, ports and timers and is mapped to the following complex type:

<xsd:complexType name="Id">

<xsd:sequence>

<xsd:element name="name" type="SimpleTypes:TString"/>

<xsd:element name="id" type="SimpleTypes:TString" minOccurs="0"/>

<xsd:element name="type" type="SimpleTypes:TString" minOccurs="0"/>

</xsd:sequence>

</xsd:complexType>

**Elements:**

* name The name of the component, port or timer.
* id The internal representation of the component, port or timer.
* type The type of the component, port or timer.

**Attributes:**

* none.

#### 11.3.2.6 TriMessageType

**TriMessageType** is mapped to the following complex type:

<xsd:complexType name="TriMessageType">

<xsd:attribute name="val" type="xsd:hexBinary"/>

<xsd:attribute name="paddingBits" type="xsd:integer" use="optional" default="0"/>

</xsd:complexType>

NOTE: paddingBits is optional with a default value of 0 and should only take values between 0 and 7.   
The relation between paddingBits and numberOfBits is:   
numberOfBits == (length(val-attribute)/2)\*8-paddingBits  
In the byte-aligned case which is the typical one, the paddingBits attribute can be left out.

**Elements:**

* none.

**Attributes:**

* val The encoded message.
* paddingBits The padding bits of the encoded message.

#### 11.3.2.7 TriParameterType

**TriParameterType** is mapped to the following complex type:

<xsd:complexType name="TriParameterType">

<xsd:attribute name="val" type="xsd:hexBinary"/>

<xsd:attribute name="paddingBits" type="xsd:integer" use="optional" default="0"/>

<xsd:attribute name="name" type="SimpleTypes:TString"/>

<xsd:attribute name="mode" type="SimpleTypes:TciParameterPassingModeType"/>

</xsd:complexType>

NOTE: paddingBits is optional with a default value of 0 and should only take values between 0 and 7.   
The relation between paddingBits and numberOfBits is:   
numberOfBits == (length(val-attribute)/2)\*8-paddingBits  
In the byte-aligned case which is the typical one, the paddingBits attribute can be left out.

**Elements:**

* none.

**Attributes:**

* val The encoded parameter.
* paddingBits The padding bits of the encoded parameter.
* name The parameter name.
* mode The parameter passing mode.

#### 11.3.2.8 TriParameterListType

**TriParameterListType** is mapped to the following complex type:

<xsd:complexType name="TriParameterListType">

<xsd:sequence>

<xsd:element name="par" type="Types:TriParameterType" minOccurs="0"

maxOccurs="unbounded"/>

</xsd:sequence>

</xsd:complexType>

**Sequence of Elements:**

* par The parameters in that list.

**Attributes:**

* none.

#### 11.3.2.9 TriAddressType

**TriAddressType** is mapped to the following complex type:

<xsd:complexType name="TriAddressType">

<xsd:attribute name="val" type="xsd:hexBinary"/>

<xsd:attribute name="paddingBits" type="xsd:integer" use="optional" default="0"/>

</xsd:complexType>

NOTE: paddingBits is optional with a default value of 0 and should only take values between 0 and 7.   
The relation between paddingBits and numberOfBits is:   
numberOfBits == (length(val-attribute)/2)\*8-paddingBits  
In the byte-aligned case which is the typical one, the paddingBits attribute can be left out.

**Elements:**

* none.

**Attributes:**

* val The address value.
* paddingBits The padding bits of the encoded address.

#### 11.3.2.10 TriAddressListType

**TriAddressListType** is mapped to the following complex type:

<xsd:complexType name="TriAddressListType">

<xsd:sequence>

<xsd:element name="addr" type="Types:TriAddressType" minOccurs="0"

maxOccurs="unbounded"/>

</xsd:sequence>

</xsd:complexType>

**Elements:**

* addr The addresses in that list.

**Attributes:**

* none.

#### 11.3.2.11 TriExceptionType

**TriExceptionType** is mapped to the following complex type:

<xsd:complexType name="TriExceptionType">

<xsd:attribute name="val" type="xsd:hexBinary"/>

<xsd:attribute name="paddingBits" type="xsd:integer" use="optional" default="0"/>

</xsd:complexType>

NOTE: paddingBits is optional with a default value of 0 and should only take values between 0 and 7.   
The relation between paddingBits and numberOfBits is:   
numberOfBits == (length(val-attribute)/2)\*8-paddingBits  
In the byte-aligned case which is the typical one, the paddingBits attribute can be left out.

**Elements:**

* val The exception.

**Attributes:**

* none.

#### 11.3.2.12 TriSignatureIdType

**TriSignatureIdType** is mapped to the following complex type:

<xsd:complexType name="TriSignatureIdType">

<xsd:attribute name="val" type="SimpleTypes:TString" use="required"/>

</xsd:complexType>

**Elements:**

* val The signature.

**Attributes:**

* none.

#### 11.3.2.13 TriTimerIdType

**TriTimerIdType** is mapped to the following complex type:

<xsd:complexType name="TriTimerIdType">

<xsd:sequence>

<xsd:element name="id" type="Types:Id"/>

</xsd:sequence>

</xsd:complexType>

**Elements:**

* id The identification of the timer.

**Attributes:**

* none.

#### 11.3.2.14 TriTimerDurationType

**TriTimerDurationType** is mapped to the following simple type:

<xsd:simpleType name="TriTimerDurationType">

<xsd:restriction base="xsd:float"/>

</xsd:simpleType>

#### 11.3.2.15 QualifiedName

**QualifiedName** is used to fully qualify module parameters, variables, etc and is mapped to the following complex type:

<xsd:complexType name="QualifiedName">

<xsd:attribute name="moduleName" type="SimpleTypes:TString" use="required"/>

<xsd:attribute name="baseName" type="SimpleTypes:TString" use="required"/>

</xsd:complexType>

**Elements:**

* moduleName The module name of the TTCN‑3 module.
* baseName The name of the object that is fully qualified.

**Attributes:**

* none.

#### 11.3.2.16 TciBehaviourIdType

**TciBehaviourIdType** is mapped to the following complex type:

<xsd:complexType name="TciBehaviourIdType">

<xsd:sequence>

<xsd:element name="name" type="Types:QualifiedName"/>

</xsd:sequence>

</xsd:complexType>

**Elements:**

* name The qualified name of the behaviour.

**Attributes:**

* none.

#### 11.3.2.17 TciTestCaseIdType

**TciTestCaseIdType** is mapped to the following complex type:

<xsd:complexType name="TciTestCaseIdType">

<xsd:sequence>

<xsd:element name="name" type="Types:QualifiedName"/>

</xsd:sequence>

</xsd:complexType>

**Elements:**

* name The qualified name of the test case.

**Attributes:**

* none.

#### 11.3.2.18 TciParameterType

**TciParameterType** is mapped to the following complex type:

<xsd:complexType name="TciParameterType">

<xsd:sequence>

<xsd:element name="val" type="Values:Value"/>

</xsd:sequence>

<xsd:attribute name="name" type="SimpleTypes:TString"/>

<xsd:attribute name="mode" type="SimpleTypes:TciParameterPassingModeType"/>

</xsd:complexType>

**Elements:**

* val The encoded parameter.

**Attributes:**

* name The parameter name.
* mode The parameter passing mode.

#### 11.3.2.19 TciParameterListType

**TciParameterListType** is mapped to the following complex type:

<xsd:complexType name="TciParameterListType">

<xsd:sequence>

<xsd:element name="par" type="Types:TciParameterType"

minOccurs="0" maxOccurs="unbounded"/>

</xsd:sequence>

</xsd:complexType>

**Sequence of Elements:**

* par The parameters in that list.

**Attributes:**

* none.

#### 11.3.2.20 TriPortIdListType

**TriPortIdListType** is mapped to the following complex type:

<xsd:complexType name="TriPortIdListType">

<xsd:sequence>

<xsd:element name="port" type="Types:TriPortIdType" minOccurs="0"

maxOccurs="unbounded"/>

</xsd:sequence>

</xsd:complexType>

**Elements:**

* port The identifiers of TRI ports in that list.

**Attributes:**

* none.

### 11.3.3 Abstract value mapping

#### 11.3.3.1 Value

**Value** is mapped to the following complex type:

<xsd:complexType name="Value" mixed="true">

<xsd:group ref="Values:Value"/>

<xsd:attributeGroup ref="Values:ValueAtts"/>

</xsd:complexType>

<xsd:group name="Value">

<xsd:choice>

<xsd:element name="integer" type="Values:IntegerValue"/>

<xsd:element name="float" type="Values:FloatValue"/>

<xsd:element name="boolean" type="Values:BooleanValue"/>

<xsd:element name="verdicttype" type="Values:VerdictValue"/>

<xsd:element name="bitstring" type="Values:BitstringValue"/>

<xsd:element name="hexstring" type="Values:HexstringValue"/>

<xsd:element name="octetstring" type="Values:OctetstringValue"/>

<xsd:element name="charstring" type="Values:CharstringValue"/>

<xsd:element name="universal\_charstring" type="Values:UniversalCharstringValue"/>

<xsd:element name="record" type="Values:RecordValue"/>

<xsd:element name="record\_of" type="Values:RecordOfValue"/>

<xsd:element name="set" type="Values:SetValue"/>

<xsd:element name="set\_of" type="Values:SetOfValue"/>

<xsd:element name="enumerated" type="Values:EnumeratedValue"/>

<xsd:element name="union" type="Values:UnionValue"/>

<xsd:element name="anytype" type="Values:AnytypeValue"/>

<xsd:element name="address" type="Values:AddressValue"/>

<xsd:element name="component" type="Values:ComponentValue"/>

<xsd:element name="port" type="Values:PortValue"/>

<xsd:element name="default" type="Values:DefaultValue"/>

<xsd:element name="timer" type="Values:TimerValue"/>

</xsd:choice>

</xsd:group>

<xsd:simpleType name="ValueModifier">

<xs:restriction base="SimpleTypes:TString">  
 <xs:enumeration value="lazy"/>  
 <xs:enumeration value="fuzzy"/>

</xs:restriction>

</xsd:simpleType>

<xsd:attributeGroup name="ValueAtts">

<xsd:attribute name="name" type="SimpleTypes:TString" use="optional"/>

<xsd:attribute name="type" type="SimpleTypes:TString" use="optional"/>

<xsd:attribute name="module" type="SimpleTypes:TString" use="optional"/>

<xsd:attribute name="modifier" type="Values:ValueModifier" use="optional"/>

<xsd:attribute name="annotation" type="SimpleTypes:TString" use="optional"/> </xsd:attributeGroup>

<xsd:group name="BaseValue">

<xsd:choice>

<xsd:sequence>

<xsd:choice>

<xsd:element name="value" type="SimpleTypes:TString"/>

<xsd:element name="matching\_symbol" type="Templates:MatchingSymbol"/>

</xsd:choice>

<xsd:element name="ifpresent" type="SimpleTypes:TEmpty" minOccurs="0"/>

<xsd:element name="length" type="Values:LengthRestriction" minOccurs="0"/>

</xsd:sequence>

<xsd:element name="null" type="SimpleTypes:TEmpty"/>

<xsd:element name="omit" type=" SimpleTypes:TEmpty"/>

<xsd:element name="not\_evaluated" type=" SimpleTypes:TEmpty"/>

</xsd:choice>

</xsd:group>

<xsd:complexType name="LengthRestriction">

<xsd:sequence>

<xsd:element name="lower" type="SimpleTypes:TInteger" />

<xsd:element name="upper" type="SimpleTypes:TInteger" minOccurs="0" />

</xsd:sequence>

</xsd:complexType>

**Value Group:**

* integer An integer value.
* float A float value.
* boolean A boolean value.
* verdicttype A verdicttype value.
* bitstring A bitstring value.
* hexstring A hexstring value.
* octetstring An octetstring value.
* charstring A charstring value.
* universal\_charstring A universal charstring value.
* record A record value.
* record\_of A record of value.
* array An array value.
* set A set value.
* set\_of A set of value.
* enumerated An enumerated value.
* union A union value.
* anytype An anytype value.
* address An address value.
* component A component value.
* port A port value.
* default A default value.
* timer A timer value.

**Attributes:**

* name The name of the value, if known.
* type The type of the value, if known.
* module The module of the value, if known.
* modifier The value modifier, if used: either lazy or fuzzy.
* annotation A helper attribute to provide additional matching/mismatching information, etc.

**BaseValue Group:**

* value A value in the string format.
* matching\_symbol A matching symbol when used instead of a value.
* ifpresent The ifpresent matching attribute.
* length The length matching attribute.
* null If no value is given.
* omit If the value is omitted.
* not\_evaluated Used if a @lazy or @fuzzy value contains not evaluated content.

**LengthRestriction Element:**

* lower The lower bound of the length matching attribute.
* upper The upper bound of the length matching attribute. Omitted when equal to infinity.

#### 11.3.3.2 IntegerValue

**IntegerValue** is mapped to the following complex type:

<xsd:complexType name="IntegerValue">

<xsd:group ref="Values:BaseValue"/>

<xsd:attributeGroup ref="Values:ValueAtts"/>

</xsd:complexType>

**Items:**

* BaseValue Integer value content described in clause 11.3.3.1.
* ValueAtts Value attributes described in clause 11.3.3.1.

#### 11.3.3.3 FloatValue

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

<xsd:complexType name="FloatValue">

<xsd:group ref="Values:BaseValue"/>

<xsd:attributeGroup ref="Values:ValueAtts"/>

</xsd:complexType>

**Items:**

* BaseValue Float value content described in clause 11.3.3.1.
* ValueAtts Value attributes described in clause 11.3.3.1.

#### 11.3.3.4 BooleanValue

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

<xsd:complexType name="BooleanValue">

<xsd:group ref="Values:BaseValue"/>

<xsd:attributeGroup ref="Values:ValueAtts"/>

</xsd:complexType>

**Items:**

* BaseValue Boolean value content described in clause 11.3.3.1.
* ValueAtts Value attributes described in clause 11.3.3.1.

#### 11.3.3.5 Void

#### 11.3.3.6 VerdictValue

**VerdictValue** is mapped to the following complex type:

<xsd:complexType name="VerdictValue">

<xsd:group ref="Values:BaseValue"/>

<xsd:attributeGroup ref="Values:ValueAtts"/>

</xsd:complexType>

**Items:**

* BaseValue Verdict value content described in clause 11.3.3.1.
* ValueAtts Value attributes described in clause 11.3.3.1.

#### 11.3.3.7 BitstringValue

**BitstringValue** is mapped to the following complex type:

<xsd:complexType name="BitstringValue">

<xsd:group ref="Values:BaseValue"/>

<xsd:attributeGroup ref="Values:ValueAtts"/>

</xsd:complexType>

**Items:**

* BaseValue Bitstring value content described in clause 11.3.3.1.
* ValueAtts Value attributes described in clause 11.3.3.1.

#### 11.3.3.8 HexstringValue

**HexstringValue** is mapped to the following complex type:

<xsd:complexType name="HexstringValue">

<xsd:attributeGroup ref="Values:ValueAtts"/>

<xsd:attributeGroup ref="Values:ValueAtts"/>

</xsd:complexType>

**Items:**

* BaseValue Hexstring value content described in clause 11.3.3.1.
* ValueAtts Value attributes described in clause 11.3.3.1.

#### 11.3.3.9 OctetstringValue

**OctetstringValue** is mapped to the following complex type:

<xsd:complexType name="OctetstringValue">

<xsd:attributeGroup ref="Values:ValueAtts"/>

<xsd:attributeGroup ref="Values:ValueAtts"/>

</xsd:complexType>

**Items:**

* BaseValue Octetstring value content described in clause 11.3.3.1.
* ValueAtts Value attributes described in clause 11.3.3.1.

#### 11.3.3.10 CharstringValue

**CharstringValue** is mapped to the following complex type:

<xsd:complexType name="CharstringValue">

<xsd:attributeGroup ref="Values:ValueAtts"/>

<xsd:attributeGroup ref="Values:ValueAtts"/>

</xsd:complexType>

**Items:**

* BaseValue Charstring value content described in clause 11.3.3.1.
* ValueAtts Value attributes described in clause 11.3.3.1.

#### 11.3.3.11 UniversalCharstringValue

**UniversalCharstringValue** is mapped to the following complex type:

<xsd:complexType name="UniversalCharstringValue">

<xsd:attributeGroup ref="Values:ValueAtts"/>

<xsd:attributeGroup ref="Values:ValueAtts"/>

</xsd:complexType>

**Items:**

* BaseValue Universal charstring value content described in clause 11.3.3.1.
* ValueAtts Value attributes described in clause 11.3.3.1.

#### 11.3.3.12 RecordValue

**RecordValue** is mapped to the following complex type:

<xsd:complexType name="RecordValue">

<xsd:choice>

<xsd:sequence>  
 <xsd:choice>  
 <xsd:group ref="Values:Value" minOccurs="0" maxOccurs="unbounded"/>

    <xsd:element name="matching\_symbol" type="Templates:MatchingSymbol"/>  
 </xsd:choice>  
 <xsd:element name="ifpresent" type="SimpleTypes:TEmpty" minOccurs="0"/>

</xsd:sequence>

<xsd:element name="null" type="SimpleTypes:TEmpty"/>

<xsd:element name="omit" type="SimpleTypes:TEmpty"/>

<xsd:element name="not\_evaluated" type="SimpleTypes:TEmpty"/>

</xsd: choice>

<xsd:attributeGroup ref="Values:ValueAtts"/>

</xsd:complexType>

**Sequence of Elements:**

* Value The value group is specified in clause 11.3.3.1. It is used for describing individual elements of the record.
* matching\_symbol A matching symbol if used instead of a value.
* ifpresent The ifpresent matching attribute.
* null If no field is given.
* omit If the field is omitted.
* not\_evaluated Used if a @lazy or @fuzzy value contains not evaluated content.

**Attributes:**

* The same attributes as those of Value.

#### 11.3.3.13 RecordOfValue

**RecordOfValue** is mapped to the following complex type:

<xsd:complexType name="RecordOfValue">

<xsd:group ref="Values:Values"/>

<xsd:attributeGroup ref="Values:ValueAtts"/>

</xsd:complexType>

<xsd:group name="Values">

<xsd:choice>

<xsd:sequence>  
 <xsd:choice>

<xsd:element name="integer" type="Values:IntegerValue" minOccurs="0"

maxOccurs="unbounded"/>

<xsd:element name="float" type="Values:FloatValue" minOccurs="0"

maxOccurs="unbounded"/>

<xsd:element name="boolean" type="Values:BooleanValue" minOccurs="0"

maxOccurs="unbounded"/>

<xsd:element name="bitstring" type="Values:BitstringValue"

minOccurs="0" maxOccurs="unbounded"/>

<xsd:element name="hexstring" type="Values:HexstringValue"

minOccurs="0" maxOccurs="unbounded"/>

<xsd:element name="octetstring" type="Values:OctetstringValue"

minOccurs="0" maxOccurs="unbounded"/>

<xsd:element name="charstring" type="Values:CharstringValue"

minOccurs="0" maxOccurs="unbounded"/>

<xsd:element name="universal\_charstring"

type="Values:UniversalCharstringValue" minOccurs="0"

maxOccurs="unbounded"/>

<xsd:element name="record" type="Values:RecordValue" minOccurs="0"

maxOccurs="unbounded"/>

<xsd:element name="record\_of" type="Values:RecordOfValue"

minOccurs="0" maxOccurs="unbounded"/>

<xsd:element name="set" type="Values:SetValue" minOccurs="0"

maxOccurs="unbounded"/>

<xsd:element name="set\_of" type="Values:SetOfValue"

minOccurs="0" maxOccurs="unbounded"/>

<xsd:element name="enumerated" type="Values:EnumeratedValue"

minOccurs="0" maxOccurs="unbounded"/>

<xsd:element name="union" type="Values:UnionValue" minOccurs="0"

maxOccurs="unbounded"/>

<xsd:element name="anytype" type="Values:AnytypeValue" minOccurs="0"

maxOccurs="unbounded"/>

<xsd:element name="address" type="Values:AddressValue" minOccurs="0"

maxOccurs="unbounded"/>

<xsd:element name="component" type="Values:ComponentValue" minOccurs="0"

maxOccurs="unbounded"/>

<xsd:element name="port" type="Values:PortValue" minOccurs="0"

maxOccurs="unbounded"/>

<xsd:element name="default" type="Values:DefaultValue" minOccurs="0"

maxOccurs="unbounded"/>

<xsd:element name="timer" type="Values:TimerValue" minOccurs="0"

maxOccurs="unbounded"/>

    <xsd:element name="matching\_symbol" type="Templates:MatchingSymbol"/>  
 </xsd:choice>  
 <xsd:element name="ifpresent" type="SimpleTypes:TEmpty" minOccurs="0"/>  
 <xsd:element name="length" type="Values:LengthRestriction" minOccurs="0"/>  
 </xsd:sequence>

<xsd:element name="null" type="SimpleTypes:TEmpty"/>

<xsd:element name="omit" type="SimpleTypes:TEmpty"/>

<xsd:element name="not\_evaluated" type="SimpleTypes:TEmpty"/>

</xsd:choice>

</xsd:group>

**Values Group:**

* integer An integer value.
* float A float value.
* boolean A boolean value.
* verdicttype A verdicttype value.
* bitstring A bitstring value.
* hexstring A hexstring value.
* octetstring An octetstring value.
* charstring A charstring value.
* universal\_charstring A universal charstring value.
* record A record value.
* record\_of A record of value.
* array An array value.
* set A set value.
* set\_of A set of value.
* enumerated An enumerated value.
* union A union value.
* anytype An anytype value.
* address An address value.
* component A component value.
* default A default value.
* matching\_symbol A matching symbol if used instead of a value.
* ifpresent The ifpresent matching attribute.
* length The length matching attribute.
* null If no field is given.
* omit If the field is omitted.
* not\_evaluated Used if a @lazy or @fuzzy value contains not evaluated content.

**Attributes:**

* The same attributes as those of Value.

#### 11.3.3.14 ArrayValue

**ArrayValue** is mapped to the following complex type:

<xsd:complexType name="ArrayValue">

<xsd:group ref="Values:Values"/>

<xsd:attributeGroup ref="Values:ValueAtts"/>

</xsd:complexType>

**Items:**

* Values Array value content described in clause 11.3.3.13.
* ValueAtts Value attributes described in clause 11.3.3.1.

#### 11.3.3.15 SetValue

**SetValue** is mapped to the following complex type:

<xsd:complexType name="SetValue">

<xsd:choice>

<xsd:sequence>  
 <xsd:choice>  
 <xsd:group ref="Values:Value" minOccurs="0" maxOccurs="unbounded"/>

    <xsd:element name="matching\_symbol" type="Templates:MatchingSymbol"/>  
 </xsd:choice>  
 <xsd:element name="ifpresent" type="SimpleTypes:TEmpty" minOccurs="0"/>

</xsd:sequence>

<xsd:element name="null" type="SimpleTypes:TEmpty"/>

<xsd:element name="omit" type="SimpleTypes:TEmpty"/>

<xsd:element name="not\_evaluated" type="SimpleTypes:TEmpty""/>

</xsd:choice>

<xsd:attributeGroup ref="Values:ValueAtts"/>

</xsd:complexType>

**Sequence of Elements:**

* Value The value group is specified in clause 11.3.3.1. It is used for describing individual elements of the record.
* matching\_symbol A matching symbol if used instead of a value.
* ifpresent The ifpresent matching attribute.
* null If no field is given.
* omit If the field is omitted.
* not\_evaluated Used if a @lazy or @fuzzy value contains not evaluated content.

**Attributes:**

* The same attributes as those of Value.

#### 11.3.3.16 SetOfValue

**SetOfValue** is mapped to the following complex type:

<xsd:complexType name="SetOfValue">

<xsd:group ref="Values:Values"/>

<xsd:attributeGroup ref="Values:ValueAtts"/>

</xsd:complexType>

**Items:**

* Values Set of value content described in clause 11.3.3.13.
* ValueAtts Value attributes described in clause 11.3.3.1.

#### 11.3.3.17 EnumeratedValue

**EnumeratedValue** is mapped to the following complex type:

<xsd:complexType name="EnumeratedValue">

<xsd:attributeGroup ref="Values:ValueAtts"/>

<xsd:attributeGroup ref="Values:ValueAtts"/>

</xsd:complexType>

**Items:**

* BaseValue Enumerated value content described in clause 11.3.3.1.
* ValueAtts Value attributes described in clause 11.3.3.1.

#### 11.3.3.18 UnionValue

**UnionValue** is mapped to the following complex type:

<xsd:complexType name="UnionValue">

<xsd:choice>

<xsd:sequence>  
 <xsd:choice>  
 <xsd:group ref="Values:Value"/>

    <xsd:element name="matching\_symbol" type="Templates:MatchingSymbol"/>  
 </xsd:choice>  
 <xsd:element name="ifpresent" type="SimpleTypes:TEmpty" minOccurs="0"/>

</xsd:sequence>

<xsd:element name="null" type="SimpleTypes:TEmpty"/>

<xsd:element name="omit" type="SimpleTypes:TEmpty"/>

<xsd:element name="not\_evaluated" type="SimpleTypes:TEmpty"/>

</xsd:choice>

<xsd:attributeGroup ref="Values:ValueAtts"/>

</xsd:complexType>

**Choice of Elements:**

* Value The chosen value. The value group is specified in clause 11.3.3.1.
* matching\_symbol A matching symbol if used instead of a value.
* ifpresent The ifpresent matching attribute.
* null If no field is given.
* omit If the field is omitted.
* not\_evaluated Used if a @lazy or @fuzzy value contains not evaluated content.

**Attributes:**

* The same attributes as those of Value.

#### 11.3.3.19 AnytypeValue

**AnytypeValue** is mapped to the following complex type:

<xsd:complexType name="AnytypeValue">

<xsd:choice>

<xsd:sequence>  
 <xsd:choice>  
 <xsd:group ref="Values:Value"/>

    <xsd:element name="matching\_symbol" type="Templates:MatchingSymbol"/>  
 </xsd:choice>  
 <xsd:element name="ifpresent" type="SimpleTypes:TEmpty" minOccurs="0"/>

</xsd:sequence>

<xsd:element name="null" type="SimpleTypes:TEmpty"/>

<xsd:element name="omit" type="SimpleTypes:TEmpty"/>

<xsd:element name="matching\_symbol" type="Templates:MatchingSymbol"/>

<xsd:element name="not\_evaluated" type="SimpleTypes:TEmpty"/>

</xsd:choice>

<xsd:attributeGroup ref="Values:ValueAtts"/>

</xsd:complexType>

**Choice of Elements:**

* Value The chosen value. The value group is specified in clause 11.3.3.1.
* matching\_symbol A matching symbol if used instead of a value.
* ifpresent The ifpresent matching attribute.
* null If no field is given.
* omit If the field is omitted.
* not\_evaluated Used if a @lazy or @fuzzy value contains not evaluated content.

**Attributes:**

* The same attributes as those of Value.

#### 11.3.3.20 AddressValue

**AddressValue** is mapped to the following complex type:

<xsd:complexType name="AddressValue">

<xsd:group ref="Values:Value"/>

<xsd:attributeGroup ref="Values:ValueAtts"/>

</xsd:complexType>

**Items:**

* Value The value group is specified in clause 11.3.3.1. It is used for describing the content of the address value.
* ValueAtts Value attributes described in clause 11.3.3.1.

#### 11.3.3.21 ComponentValue

Value type used for component instances is mapped to the complex type specified below. The content of the XML elements based on the ComponentValue type shall be equal to the string produced by the valueToString operation (described in clause 7.2.2.2.1):

<xsd:complexType name="ComponentValue">

<xsd:group ref="Values:BaseValue"/>

<xsd:attributeGroup ref="Values:ValueAtts"/>

</xsd:complexType>

**Items:**

* BaseValue Enumerated value content described in clause 11.3.3.1.
* ValueAtts Value attributes described in clause 11.3.3.1.

#### 11.3.3.22 PortValue

Value type used for port instances is mapped to the complex type specified below. The content of the XML elements based on the PortValue type shall be equal to the string produced by the valueToString operation (described in clause 7.2.2.2.1):

<xsd:complexType name="PortValue">

<xsd:choice>

<xsd:element name="value" type="SimpleTypes:TString"/>

<xsd:element name="null" type="SimpleTypes:TEmpty"/>

<xsd:element name="omit" type="SimpleTypes:TEmpty"/>

</xsd:choice>

<xsd:attributeGroup ref="Values:ValueAtts"/>

</xsd:complexType>

**Choice of Elements:**

* value The universal charstring value as string.
* null If no value is given.
* omit If the value is omitted.

**Attributes:**

* The same attributes as those of Value.

#### 11.3.3.23 DefaultValue

Value type used for default instances is mapped to the complex type specified below. The content of the XML elements based on the DefaultValue type shall be equal to the string produced by the valueToString operation (described in clause 7.2.2.2.1):

<xsd:complexType name="DefaultValue">

<xsd:choice>

<xsd:element name="value" type="SimpleTypes:TString"/>

<xsd:element name="null" type="SimpleTypes:TEmpty"

<xsd:element name="omit" type="SimpleTypes:TEmpty"/>

<xsd:element name="not\_evaluated" type="Values:NotEvaluated"/>

</xsd:choice>

<xsd:attributeGroup ref="Values:ValueAtts"/>

</xsd:complexType>

**Choice of Elements:**

* value The universal charstring value as string.
* null If no value is given.
* omit If the value is omitted.
* not\_evaluated Used if a @lazy or @fuzzy value contains not evaluated content.

**Attributes:**

* The same attributes as those of Value.

#### 11.3.3.24 TimerValue

Value type used for timer instances is mapped to the complex type specified below. The content of the XML elements based on the TimerValue type shall be equal to the string produced by the valueToString operation (described in clause 7.2.2.2.1):

<xsd:complexType name="TimerValue">

<xsd:choice>

<xsd:element name="value" type="SimpleTypes:TString"/>

<xsd:element name="null" type="SimpleTypes:TEmpty"/>

<xsd:element name="omit" type="SimpleTypes:TEmpty"/>

</xsd:choice>

<xsd:attributeGroup ref="Values:ValueAtts"/>

</xsd:complexType>

**Choice of Elements:**

* value The universal charstring value as string.
* null If no value is given.
* omit If the value is omitted.

**Attributes:**

* The same attributes as those of Value.

#### 11.3.3.25 MatchingMechanism

**MatchingMechanism** is mapped into a sub-element of a typed value element. The sub-element is based on the complex type specified below:

<xsd:complexType name="MatchingSymbol">

<xsd:choice>

<xsd:element name="any\_value" type="SimpleTypes:TEmpty"/>

<xsd:element name="any\_value\_or\_none" type="SimpleTypes:TEmpty"/>

<xsd:element name="any\_element " type="SimpleTypes:TEmpty"/>

<xsd:element name="any\_element\_or\_none" type="SimpleTypes:TEmpty"/>

<xsd:element name="range" type="Templates:Range"/>

<xsd:element name="list" type="Templates:MatchingList"/>

<xsd:element name="complement" type="Templates:MatchingList"/>

<xsd:element name="subset" type="Templates:MatchingList"/>

<xsd:element name="superset" type="Templates:MatchingList"/>

<xsd:element name="permutation" type="Templates:MatchingList"/>

<xsd:element name="decmatch" type="Templates:DecMatch"/>

</xsd:choice>

</xsd:complexType>

**Choice of Elements:**

* any\_value The *AnyValue* matching symbol.
* any\_value\_or\_none The *AnyValueOrNone* matching symbol.
* any\_element The *AnyElement* matching symbol.
* any\_element\_or\_none The *AnyElementOrNone* matching symbol.
* range A range template.
* list A template list.
* complement A complemented template list.
* subset A subset template.
* superset A superset template.
* permutation A permutation.
* pattern A pattern.
* decmatch A *MatchDecodedContent* matching mechanism.

**Attributes:**

* The same attributes as those of Value.

#### 11.3.3.26 MatchingList

**MatchingList** is mapped to the following complex type. This complex type is also used for mapping of permutations.

<xsd:complexType name="MatchingList">

<xsd:sequence>

<xsd:group ref="Values:Value" minOccurs="0" maxOccurs="unbounded"/>

</xsd:sequence>

</xsd:complexType>

**Items:**

* Value Individual values present in the list. The Value group is specified in clause 11.3.3.1.

#### 11.3.3.27 ValueRange

**ValueRange** is mapped to the following complex type:

<xsd:complexType name="Range">

<xsd:sequence>

<xsd:element name="excludeLower" minOccurs="0"/>

<xsd:element name="lower" type="Values:Value" minOccurs="0"/>

<xsd:element name="excludeUpper" minOccurs="0"/>

<xsd:element name="upper" type="Values:Value" minOccurs="0"/>

</xsd:sequence>

</xsd:complexType>

**Items:**

* excludeLower Present if the lower bound is excluded.
* lower The lower bound of the range. The element is omitted in case of integer range whose lower bound is equal to -infinity. The associated Value type is described in clause 11.3.3.1.
* excludeUpper Present if the upper bound is excluded.
* upper The upper bound of the range. The element is omitted in case of integer range whose upper bound is equal to infinity. The associated Value type is described in clause 11.3.3.1.

#### 11.3.3.28 CharacterPattern

**CharacterPattern** is mapped to the following complex type:

<xsd:complexType name="Pattern">

<xsd:sequence>

<xsd:choice>

<xsd:element name="charstring" type="Values:CharstringValue"/>

<xsd:element name="universal\_charstring" type="Values:UniversalCharstringValue"/>

</xsd:choice>

</xsd:sequence>

</xsd:complexType>

**Items:**

* charstring A pattern string in the charstring format.
* universal\_charstring A pattern string in the universal charstring format.

#### 11.3.3.29 MatchDecodedContent

**MatchDecodedContent** is mapped to the following complex type:

<xsd:complexType name="DecMatch">

<xsd:sequence>

<xsd:group ref="Values:Value"/>

</xsd:sequence>

</xsd:complexType>

**Items:**

* Value The content of the *MatchDecodedContent* matching mechanism. The Value group is described in clause 11.3.3.1.

### 11.3.4 Abstract logging types mapping

#### 11.3.4.1 TciValueTemplate

**TciValueTemplate** is mapped to the following complex type:

<xsd:group name="TypedTemplate">

<xsd:choice>

<xsd:element name="integer" type="Templates:SimpleTemplate"/>

<xsd:element name="float" type="Templates:SimpleTemplate"/>

<xsd:element name="boolean" type="Templates:SimpleTemplate"/>

<xsd:element name="verdicttype" type="Templates:SimpleTemplate"/>

<xsd:element name="bitstring" type="Templates:SimpleTemplate"/>

<xsd:element name="hexstring" type="Templates:SimpleTemplate"/>

<xsd:element name="octetstring" type="Templates:SimpleTemplate"/>

<xsd:element name="charstring" type="Templates:SimpleTemplate"/>

<xsd:element name="universal\_charstring" type="Templates:SimpleTemplate"/>

<xsd:element name="record" type="Templates:RecordTemplate"/>

<xsd:element name="record\_of" type="Templates:RecordOfTemplate"/>

<xsd:element name="array" type="Templates: RecordOfTemplate"/>

<xsd:element name="set" type="Templates:RecordTemplate"/>

<xsd:element name="set\_of" type="Templates:RecordOfTemplate"/>

<xsd:element name="enumerated" type="Templates:SimpleTemplate"/>

<xsd:element name="union" type="Templates:UnionTemplate"/>

<xsd:element name="anytype" type="Templates:AnytypeTemplate"/>

<xsd:element name="address" type="Templates:AddressTemplate"/>

</xsd:choice>

</xsd:group>

<xsd:group name="SpecialTemplate">

<xsd:choice>

<xsd:element name="omit" type="Templates:omit"/>

<xsd:element name="any" type="Templates:any"/>

<xsd:element name="anyoromit" type="Templates:anyoromit"/>

<xsd:element name="templateDef" type="SimpleTypes:TString"/>

</xsd:choice>

</xsd:group>

<xsd:complexType name="TciValueTemplate">

<xsd:choice>

<xsd:group ref="Values:Value"/>

<xsd:group ref="Templates:TypedTemplate"/>

<xsd:group ref="Templates:SpecialTemplate"/>

</xsd:choice>

</xsd:complexType>

**Choice of Elements:**

* Value A structured and typed template definition.
* TypedTemplate A typed template definition. It is an obsolete feature kept for backwards compatibility reasons. The referenced group contains a choice of the following elements:
  + integer An integer template.
  + float A float template.
  + boolean A boolean template.
  + verdicttype A verdicttype template.
  + bitstring A bitstring template.
  + hexstring A hexstring template.
  + octetstring An octetstring template.
  + charstring A charstring template.
  + universal\_charstring   
     A universal charstring template.
  + record A record template.
  + record\_of A record of template.
  + array An array template.
  + set A set template.
  + set\_of A set of template.
  + enumerated An enumerated template.
  + union A union template.
  + anytype An anytype template.
  + address An address template.
* SpecialTemplate This group contains special matching elements. It is an obsolete feature kept for backwards compatibility reasons. The referenced group contains a choice of the following elements:
  + omit An omit template.
  + any An any template.
  + anyoromit An anyoromit template.
  + templateDef A complex template definition in text format.

**Attributes:**

* none.

#### 11.3.4.2 TciNonValueTemplate

**TciNonValueTemplate** is mapped to the following complex type:

<xsd:complexType name="TciNonValueTemplate">

<xsd:sequence>

<xsd:choice>

<xsd:element name="any" type="Templates:any"/>

<xsd:element name="all" type="Templates:all"/>

<xsd:element name="templateDef" type="SimpleTypes:TString"/>

<xsd:element name="null" type="Templates:null"/>

<xsd:group ref="Values:Value"/>

</xsd:choice>

</xsd:sequence>

</xsd:complexType>

**Choice of Elements:**

* any An any template.
* all An all template.
* templateDef A complex template definition in text format.
* null No template is given.
* Value A structured and typed template definition.

**Attributes:**

* none.

#### 11.3.4.3 TciValueList

**TciValueList** is mapped to the following complex type:

<xsd:complexType name="TciValueListType">

<xsd:sequence>

<xsd:element name="val" type="Values:Value"

maxOccurs="unbounded"/>

</xsd:sequence>

</xsd:complexType>

**Sequence of Elements:**

* val The values in the value list.

**Attributes:**

* none.

#### 11.3.4.4 TciValueDifference

**TciValueDifference** is mapped to the following complex type:

<xsd:complexType name="TciValueDifference">

<xsd:sequence>

<xsd:element name="val" type="SimpleTypes:xpath"/>

<xsd:element name="tmpl"

type="SimpleTypes:xpath"/>

</xsd:sequence>

<xsd:attributeGroup ref="Values:ValueAtts"/>

<xsd:attribute name="desc" type="SimpleTypes:TString"

use="optional"/>

</xsd:complexType>

**Sequence of Elements:**

* val A reference to the mismatching value.
* tmpl A reference to the template.

**Attributes:**

* The same attributes as those of Value.
* desc The reason of the mismatch.

#### 11.3.4.5 TciValueDifferenceList

**TciValueDifferenceList** is mapped to the following complex type:

<xsd:complexType name="TciValueDifferenceList">

<xsd:sequence>

<xsd:element name="diff" type="Templates:TciValueDifference"

maxOccurs="unbounded"/>

</xsd:sequence>

</xsd:complexType>

**Sequence of Elements:**

* diff The value/template differences in the value difference list.

**Attributes:**

* none.

## 11.4 Mapping of the operations on the logging interface

### 11.4.0 Mapping rules

Every operation provided at the logging interface has a corresponding complex type definition in XML. These complex type definitions are extensions of Event.

### 11.4.1 Event

**Event** is mapped to the following complex type:

<!‑‑ common definition for all events ‑‑>

<xsd:complexType name="Event" mixed="true">

<xsd:sequence>

<xsd:element name="am" type="SimpleTypes:TString"/>

</xsd:sequence>

<xsd:attribute name="ts" type="xsd:long" use="required"/>

<xsd:attribute name="src" type="SimpleTypes:TString" use="optional"/>

<xsd:attribute name="line" type="SimpleTypes:TInteger" use="optional"/>

<!‑‑ general identifier structure for test components, ports and timer ‑‑>

<xsd:attribute name="name" type="SimpleTypes:TString" use="required"/>

<xsd:attribute name="id" type="SimpleTypes:TString " use="required"/>

<xsd:attribute name="type" type="SimpleTypes:TString" use="required"/>

</xsd:complexType>

**Elements:**

* am A message, to be used for further information in the log.

**Attributes:**

* ts The time when the event is produced.
* src The source file of the test specification.
* line The line number where the request is performed.
* name The name of the component which produces this event.
* id The id of the component which produces this event.
* type The type of the component which produces this event.

### 11.4.2 The TCI‑TL interface

#### 11.4.2.1 TCI‑TL provided

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

<!‑‑ testcases ‑‑>

<xsd:complexType name="tliTcExecute">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="tcId" type="Types:TciTestCaseIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="dur" type="SimpleTypes:TriTimerDurationType" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliTcStart">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="tcId" type="Types:TciTestCaseIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="dur" type="SimpleTypes:TriTimerDurationType" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliTcStop">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event"/>

<xsd:sequence>

<xsd:element name="reason" type="SimpleTypes:TString" minOccurs="0"/>

</xsd:sequence>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliTcStarted">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="tcId" type="Types:TciTestCaseIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="dur" type="SimpleTypes:TriTimerDurationType" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliTcTerminated">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="tcId" type="Types:TciTestCaseIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="verdict" type="Values:VerdictValue"/>

<xsd:element name="reason" type="SimpleTypes:TString" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<!‑‑ control ‑‑>

<xsd:complexType name="tliCtrlStart">

<xsd:complexContent>

<xsd:extension base="Events:Event"/>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliCtrlStop">

<xsd:complexContent>

<xsd:extension base="Events:Event"/>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliCtrlTerminated">

<xsd:complexContent>

<xsd:extension base="Events:Event"/>

</xsd:complexContent>

</xsd:complexType>

<!‑‑ asynchronous communication ‑‑>

<xsd:complexType name="tliMSend\_m">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="to" type="Types:TriPortIdType" minOccurs="0"/>

<xsd:element name="msgValue" type="Values:Value"/>

<xsd:element name="addrValue " type="Values:Value" minOccurs="0"/>

<xsd:choice>

<xsd:element name="encoder‑failure" type="SimpleTypes:TciStatusType" minOccurs="0"/>

<xsd:sequence>

<xsd:element name="msg" type="Types:TriMessageType" minOccurs="0"/>

<xsd:element name="address" type="Types:TriAddressType" minOccurs="0"/>

<xsd:element name="transmission‑failure" type="SimpleTypes:TriStatusType" minOccurs="0"/>

</xsd:sequence>

</xsd:choice>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliMSend\_m\_BC">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="to" type="Types:TriPortIdType" minOccurs="0"/>

<xsd:element name="msgValue" type="Values:Value"/>

<xsd:choice>

<xsd:element name="encoder‑failure" type="SimpleTypes:TciStatusType" minOccurs="0"/>

<xsd:sequence>

<xsd:element name="msg" type="Types:TriMessageType" minOccurs="0"/>

<xsd:element name="transmission‑failure" type="SimpleTypes:TriStatusType" minOccurs="0"/>

</xsd:sequence>

</xsd:choice>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliMSend\_m\_MC">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="to" type="Types:TriPortIdType" minOccurs="0"/>

<xsd:element name="msgValue" type="Values:Value"/>

<xsd:element name="addrValues" type="Types:TciValueListType" minOccurs="0"/>

<xsd:choice>

<xsd:element name="encoder‑failure" type="SimpleTypes:TciStatusType" minOccurs="0"/>

<xsd:sequence>

<xsd:element name="msg" type="Types:TriMessageType" minOccurs="0"/>

<xsd:element name="addresses" type="Types:TriAddressListType" minOccurs="0"/>

<xsd:element name="transmission‑failure" type="SimpleTypes:TriStatusType" minOccurs="0"/>

</xsd:sequence>

</xsd:choice>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliMSend\_c">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="to" type="Types:TriPortIdType" minOccurs="0"/>

<xsd:element name="msgValue" type="Values:Value"/>

<xsd:element name="transmission‑failure" type="SimpleTypes:TriStatusType" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliMSend\_c\_BC">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="to" type="Types:TriPortIdListType" minOccurs="0"/>

<xsd:element name="msgValue" type="Values:Value"/>

<xsd:element name="transmission‑failure" type="SimpleTypes:TriStatusType" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliMSend\_c\_MC">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="to" type="Types:TriPortIdListType" minOccurs="0"/>

<xsd:element name="msgValue" type="Values:Value"/>

<xsd:element name="transmission‑failure" type="SimpleTypes:TriStatusType" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliMDetected\_m">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="from" type="Types:TriPortIdType" minOccurs="0"/>

<xsd:element name="msgValue" type="Types:TriMessageType"/>

<xsd:element name="address" type="Types:TriAddressType" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliMDetected\_c">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="from" type="Types:TriPortIdType" minOccurs="0"/>

<xsd:element name="msgValue" type="Values:Value"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliMMismatch\_m">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="msgValue" type="Values:Value"/>

<xsd:element name="msgTmpl" type="Templates:TciValueTemplate"/>

<xsd:element name="diffs" type="Templates:TciValueDifferenceList"/>

<xsd:element name="addrValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="addressTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliMMismatch\_c">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="msgValue" type="Values:Value"/>

<xsd:element name="msgTmpl" type="Templates:TciValueTemplate"/>

<xsd:element name="diffs" type="Templates:TciValueDifferenceList"/>

<xsd:element name="from" type="Types:TriComponentIdType" minOccurs="0"/>

<xsd:element name="fromTmpl" type="Templates:TciNonValueTemplate" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliMReceive\_m">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="msgValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="msgTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>

<xsd:element name="addrValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="addressTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliMReceive\_c">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="msgValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="msgTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>

<xsd:element name="from" type="Types:TriComponentIdType" minOccurs="0"/>

<xsd:element name="fromTmpl" type="Templates:TciNonValueTemplate" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<!‑‑ synchronous communication ‑‑>

<xsd:complexType name="tliPrCall\_m">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="to" type="Types:TriPortIdType" minOccurs="0"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="addrValue" type="Values:Value" minOccurs="0"/>

<xsd:choice>

<xsd:element name="encoder‑failure" type="SimpleTypes:TciStatusType" minOccurs="0"/>

<xsd:sequence>

<xsd:element name="triPars" type="Types:TriParameterListType" minOccurs="0"/>

<xsd:element name="address" type="Types:TriAddressType" minOccurs="0"/>

<xsd:element name="transmission‑failure" type="SimpleTypes:TriStatusType" minOccurs="0"/>

</xsd:sequence>

</xsd:choice>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrCall\_m\_BC">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="to" type="Types:TriPortIdType" minOccurs="0"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:choice>

<xsd:element name="encoder‑failure" type="SimpleTypes:TciStatusType" minOccurs="0"/>

<xsd:sequence>

<xsd:element name="triPars" type="Types:TriParameterListType" minOccurs="0"/>

<xsd:element name="transmission‑failure" type="SimpleTypes:TriStatusType" minOccurs="0"/>

</xsd:sequence>

</xsd:choice>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrCall\_m\_MC">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="to" type="Types:TriPortIdType" minOccurs="0"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="addrValues" type="Types:TciValueListType" minOccurs="0"/>

<xsd:choice>

<xsd:element name="encoder‑failure" type="SimpleTypes:TciStatusType" minOccurs="0"/>

<xsd:sequence>

<xsd:element name="triPars" type="Types:TriParameterListType" minOccurs="0"/>

<xsd:element name="addresses" type="Types:TriAddressListType" minOccurs="0"/>

<xsd:element name="transmission‑failure" type="SimpleTypes:TriStatusType" minOccurs="0"/>

</xsd:sequence>

</xsd:choice>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrCall\_c">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="to" type="Types:TriPortIdType" minOccurs="0"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="transmission‑failure" type="SimpleTypes:TriStatusType" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrCall\_c\_BC">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="to" type="Types:TriPortIdListType" minOccurs="0"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="transmission‑failure" type="SimpleTypes:TriStatusType" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrCall\_c\_MC">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="to" type="Types:TriPortIdListType" minOccurs="0"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="transmission‑failure" type="SimpleTypes:TriStatusType" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrGetCallDetected\_m">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="from" type="Types:TriPortIdType" minOccurs="0"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="triPars" type="Types:TriParameterListType" minOccurs="0"/>

<xsd:element name="address" type="Types:TriAddressType" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrGetCallDetected\_c">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="from" type="Types:TriPortIdType" minOccurs="0"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrGetCallMismatch\_m">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="parsTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>

<xsd:element name="diffs" type="Templates:TciValueDifferenceList"/>

<xsd:element name="addrValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="addressTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrGetCallMismatch\_c">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="parsTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>

<xsd:element name="diffs" type="Templates:TciValueDifferenceList"/>

<xsd:element name="from" type="Types:TriComponentIdType" minOccurs="0"/>

<xsd:element name="fromTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrGetCall\_m">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="parsTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>

<xsd:element name="addrValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="addressTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrGetCall\_c">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="parsTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>

<xsd:element name="from" type="Types:TriComponentIdType" minOccurs="0"/>

<xsd:element name="fromTmpl" type="Templates:TciNonValueTemplate" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrReply\_m">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="to" type="Types:TriPortIdType" minOccurs="0"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="replValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="addrValue" type="Values:Value" minOccurs="0"/>

<xsd:choice>

<xsd:element name="encoder‑failure" type="SimpleTypes:TciStatusType"

minOccurs="0"/>

<xsd:sequence>

<xsd:element name="triPars" type="Types:TriParameterListType" minOccurs="0"/>

<xsd:element name="repl" type="Types:TriParameterType" minOccurs="0"/>

<xsd:element name="address" type="Types:TriAddressType" minOccurs="0"/>

<xsd:element name="transmission‑failure"

type="SimpleTypes:TriStatusType" minOccurs="0"/>

</xsd:sequence>

</xsd:choice>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrReply\_m\_BC">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="to" type="Types:TriPortIdType" minOccurs="0"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="replValue" type="Values:Value" minOccurs="0"/>

<xsd:choice>

<xsd:element name="encoder‑failure" type="SimpleTypes:TciStatusType"

minOccurs="0"/>

<xsd:sequence>

<xsd:element name="triPars" type="Types:TriParameterListType" minOccurs="0"/>

<xsd:element name="repl" type="Types:TriParameterType" minOccurs="0"/>

<xsd:element name="transmission‑failure"

type="SimpleTypes:TriStatusType" minOccurs="0"/>

</xsd:sequence>

</xsd:choice>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrReply\_m\_MC">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="to" type="Types:TriPortIdType" minOccurs="0"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="replValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="addrValues" type="Types:TciValueListType" minOccurs="0"/>

<xsd:choice>

<xsd:element name="encoder‑failure" type="SimpleTypes:TciStatusType"

minOccurs="0"/>

<xsd:sequence>

<xsd:element name="triPars" type="Types:TriParameterListType" minOccurs="0"/>

<xsd:element name="repl" type="Types:TriParameterType" minOccurs="0"/>

<xsd:element name="addresses" type="Types:TriAddressListType" minOccurs="0"/>

<xsd:element name="transmission‑failure"

type="SimpleTypes:TriStatusType" minOccurs="0"/>

</xsd:sequence>

</xsd:choice>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrReply\_c">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="to" type="Types:TriPortIdType" minOccurs="0"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="replValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="transmission‑failure" type="SimpleTypes:TriStatusType" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrReply\_c\_BC">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="to" type="Types:TriPortIdListType" minOccurs="0"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="replValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="transmission‑failure" type="SimpleTypes:TriStatusType" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrReply\_c\_MC">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="to" type="Types:TriPortIdListType" minOccurs="0"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="replValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="transmission‑failure" type="SimpleTypes:TriStatusType" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrGetReplyDetected\_m">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="from" type="Types:TriPortIdType" minOccurs="0"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="triPars" type="Types:TriParameterListType" minOccurs="0"/>

<xsd:element name="repl" type="Types:TriParameterType" minOccurs="0"/>

<xsd:element name="address" type="Types:TriAddressType" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrGetReplyDetected\_c">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="from" type="Types:TriPortIdType" minOccurs="0"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="replValue" type="Values:Value" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrGetReplyMismatch\_m">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="parsTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>

<xsd:element name="replValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="replTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>

<xsd:element name="diffs" type="Templates:TciValueDifferenceList"/>

<xsd:element name="addrValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="addressTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrGetReplyMismatch\_c">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="parsTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>

<xsd:element name="replValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="replTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>

<xsd:element name="diffs" type="Templates:TciValueDifferenceList"/>

<xsd:element name="from" type="Types:TriComponentIdType" minOccurs="0"/>

<xsd:element name="fromTmpl" type="Templates:TciNonValueTemplate" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrGetReply\_m">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="parsTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>

<xsd:element name="replValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="replTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>

<xsd:element name="addrValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="addressTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrGetReply\_c">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="parsTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>

<xsd:element name="replValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="replTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>

<xsd:element name="from" type="Types:TriComponentIdType" minOccurs="0"/>

<xsd:element name="fromTmpl" type="Templates:TciNonValueTemplate" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrRaise\_m">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="to" type="Types:TriPortIdType" minOccurs="0"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="excValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="addrValue" type="Values:Value" minOccurs="0"/>

<xsd:choice>

<xsd:element name="encoder‑failure" type="SimpleTypes:TciStatusType" minOccurs="0"/>

<xsd:sequence>

<xsd:element name="exc" type="Types:TriExceptionType" minOccurs="0"/>

<xsd:element name="address" type="Types:TriAddressType" minOccurs="0"/>

<xsd:element name="transmission‑failure" type="SimpleTypes:TriStatusType" minOccurs="0"/>

</xsd:sequence>

</xsd:choice>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrRaise\_m\_BC">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="to" type="Types:TriPortIdType" minOccurs="0"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="excValue" type="Values:Value" minOccurs="0"/>

<xsd:choice>

<xsd:element name="encoder‑failure" type="SimpleTypes:TciStatusType" minOccurs="0"/>

<xsd:sequence>

<xsd:element name="exc" type="Types:TriExceptionType" minOccurs="0"/>

<xsd:element name="transmission‑failure" type="SimpleTypes:TriStatusType" minOccurs="0"/>

</xsd:sequence>

</xsd:choice>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrRaise\_m\_MC">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="to" type="Types:TriPortIdType" minOccurs="0"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="excValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="addrValues" type="Types:TciValueListType" minOccurs="0"/>

<xsd:choice>

<xsd:element name="encoder‑failure" type="SimpleTypes:TciStatusType" minOccurs="0"/>

<xsd:sequence>

<xsd:element name="exc" type="Types:TriExceptionType" minOccurs="0"/>

<xsd:element name="addresses" type="Types:TriAddressListType" minOccurs="0"/>

<xsd:element name="transmission‑failure" type="SimpleTypes:TriStatusType" minOccurs="0"/>

</xsd:sequence>

</xsd:choice>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrRaise\_c">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="to" type="Types:TriPortIdType" minOccurs="0"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="excValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="transmission‑failure" type="SimpleTypes:TriStatusType" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrRaise\_c\_BC">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="to" type="Types:TriPortIdListType" minOccurs="0"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="excValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="transmission‑failure" type="SimpleTypes:TriStatusType" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrRaise\_c\_MC">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="to" type="Types:TriPortIdListType" minOccurs="0"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="excValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="transmission‑failure" type="SimpleTypes:TriStatusType" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrCatchDetected\_m">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="from" type="Types:TriPortIdType" minOccurs="0"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="exc" type="Types:TriExceptionType" minOccurs="0"/>

<xsd:element name="address" type="Types:TriAddressType" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrCatchDetected\_c">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="from" type="Types:TriPortIdType" minOccurs="0"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="excValue" type="Values:Value" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrCatchMismatch\_m">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="excValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="excTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>

<xsd:element name="diffs" type="Templates:TciValueDifferenceList"/>

<xsd:element name="addrValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="addressTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrCatchMismatch\_c">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="excValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="excTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>

<xsd:element name="diffs" type="Templates:TciValueDifferenceList"/>

<xsd:element name="from" type="Types:TriComponentIdType" minOccurs="0"/>

<xsd:element name="fromTmpl" type="Templates:TciNonValueTemplate" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrCatch\_m">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="excValue" type="Values:Value"/>

<xsd:element name="excTmpl" type="Templates:TciValueTemplate"/>

<xsd:element name="addrValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="addressTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrCatch\_c">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="excValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="excTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>

<xsd:element name="from" type="Types:TriComponentIdType" minOccurs="0"/>

<xsd:element name="fromTmpl" type="Templates:TciNonValueTemplate" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrCatchTimeoutDetected">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrCatchTimeout">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<!‑‑ components ‑‑>

<xsd:complexType name="tliCCreate">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="comp" type="Types:TriComponentIdType"/>

<xsd:element name="name" type="SimpleTypes:TString"/>

<xsd:element name="hostId" type="Values:Value" minOccurs="0"/>

<xsd:element name="alive" type="SimpleTypes:TBoolean"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliCStart">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="comp" type="Types:TriComponentIdType"/>

<xsd:element name="name" type="Types:TciBehaviourIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliCRunning">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="comp" type="Types:TriComponentIdType"/>

<xsd:element name="status" type="SimpleTypes:ComponentStatusType"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliCAlive">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="comp" type="Types:TriComponentIdType"/>

<xsd:element name="status" type="SimpleTypes:ComponentStatusType"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliCStop">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="comp" type="Types:TriComponentIdType"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliCKill">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="comp" type="Types:TriComponentIdType"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliCDoneMismatch">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="comp" type="Types:TriComponentIdType"/>

<xsd:element name="compTmpl" type="Templates:TciNonValueTemplate"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliCKilledMismatch">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="comp" type="Types:TriComponentIdType"/>

<xsd:element name="compTmpl" type="Templates:TciNonValueTemplate"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliCDone">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="compTmpl" type="Templates:TciNonValueTemplate"/>

<xsd:element name="verdict" type="Values:VerdictValue" minOccurs="0" />

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliCKilled">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="compTmpl" type="Templates:TciNonValueTemplate"/>

<xsd:element name="verdict" type="Values:VerdictValue" minOccurs="0" />

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliCTerminated">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="verdict" type="Values:VerdictValue" />

<xsd:element name="reason" type="SimpleTypes:TString" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<!‑‑ ports ‑‑>

<xsd:complexType name="tliPConnect">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:PortConfiguration"/>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPDisconnect">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:PortConfiguration"/>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPMap">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:PortConfiguration"/>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPMapParam">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:tliPMap">

<xsd:sequence>

<xsd:element name="tciPars" type="Types:TciParameterListType" />

<xsd:choice>

<xsd:element name="encoder‑failure" type="SimpleTypes:TciStatusType" minOccurs="0"/>

<xsd:element name="triPars" type="Types:TriParameterListType" />

</xsd:choice>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPUnmap">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:PortConfiguration"/>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPUnmapParam">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:tliPUnmap">

<xsd:sequence>

<xsd:element name="tciPars" type="Types:TciParameterListType" />

<xsd:choice>

<xsd:element name="encoder‑failure" type="SimpleTypes:TciStatusType" minOccurs="0"/>

<xsd:element name="triPars" type="Types:TriParameterListType"/>

</xsd:choice>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPClear">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:PortStatus"/>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPStart">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:PortStatus"/>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPStop">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:PortStatus"/>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPHalt">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:PortStatus"/>

</xsd:complexContent>

</xsd:complexType>

<!‑‑ codec ‑‑>

<xsd:complexType name="tliEncode">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="val" type="Values:Value"/>

<xsd:choice>

<xsd:element name="msg" type="Types:TriMessageType"/>

<xsd:element name="encoder‑failure" type="SimpleTypes:TciStatusType" minOccurs="0"/>

</xsd:choice>

<xsd:element name="codec" type="SimpleTypes:TString"

minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliDecode" mixed="true">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="msg" type="Types:TriMessageType"/>

<xsd:choice>

<xsd:element name="decoder‑failure" type="SimpleTypes:TciStatusType" minOccurs="0"/>

<xsd:element name="val" type="Values:Value"/>

</xsd:choice>

<xsd:element name="codec" type="SimpleTypes:TString"

minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<!‑‑ timers ‑‑>

<xsd:complexType name="tliTTimeoutDetected">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="timer" type="Types:TriTimerIdType" />

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliTTimeoutMismatch">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="timer" type="Types:TriTimerIdType" />

<xsd:element name="timerTmpl" type="Templates:TciNonValueTemplate" />

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliTTimeout">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="timer" type="Types:TriTimerIdType" />

<xsd:element name="timerTmpl" type="Templates:TciNonValueTemplate" />

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliTStart">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="timer" type="Types:TriTimerIdType"/>

<xsd:element name="dur" type="SimpleTypes:TriTimerDurationType"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliTStop">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="timer" type="Types:TriTimerIdType"/>

<xsd:element name="dur" type="SimpleTypes:TriTimerDurationType"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliTRead">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="timer" type="Types:TriTimerIdType"/>

<xsd:element name="elapsed" type="SimpleTypes:TriTimerDurationType"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliTRunning">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="timer" type="Types:TriTimerIdType"/>

<xsd:element name="status" type="SimpleTypes:TimerStatusType"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<!‑‑ scope ‑‑>

<xsd:complexType name="tliSEnter">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="name" type="Types:QualifiedName" />

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="kind" type="SimpleTypes:TString"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliSLeave">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="name" type="Types:QualifiedName" />

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="returnValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="kind" type="SimpleTypes:TString"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<!‑‑ variables and module parameter ‑‑>

<xsd:complexType name="tliVar">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="name" type="Types:QualifiedName" />

<xsd:element name="val" type="Values:Value" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliModulePar">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="name" type="Types:QualifiedName" />

<xsd:element name="val" type="Values:Value" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<!‑‑ verdicts ‑‑>

<xsd:complexType name="tliGetVerdict">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="verdict" type="Values:VerdictValue"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliSetVerdict">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="verdict" type="Values:VerdictValue"/>

<xsd:element name="reason" type="SimpleTypes:TString" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<!‑‑ log ‑‑>

<xsd:complexType name="tliLog">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="log" type="SimpleTypes:TString"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<!‑‑ alt ‑‑>

<xsd:complexType name="tliAEnter">

<xsd:complexContent>

<xsd:extension base="Events:Event"/>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliALeave">

<xsd:complexContent>

<xsd:extension base="Events:Event"/>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliADefaults">

<xsd:complexContent>

<xsd:extension base="Events:Event"/>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliAActivate">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="name" type="Types:QualifiedName" />

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="ref" type="Values:Value"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliADeactivate">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="ref" type="Values:Value"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliANomatch">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event"/>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliARepeat">

<xsd:complexContent>

<xsd:extension base="Events:Event"/>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliAWait">

<xsd:complexContent>

<xsd:extension base="Events:Event"/>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliAction">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="action" type="SimpleTypes:TString"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliMatch">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="expr" type="Values:Value"/>

<xsd:element name="tmpl" type="Templates:TciValueTemplate"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliMatchMismatch">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="expr" type="Values:Value"/>

<xsd:element name="tmpl" type="Templates:TciValueTemplate"/>

<xsd:element name="diffs" type="Templates:TciValueDifferenceList"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliInfo">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="level" type="SimpleTypes:TInteger"/>

<xsd:element name="info" type="SimpleTypes:TString"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliMChecked\_m">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="msgValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="msgTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>

<xsd:element name="addrValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="addressTmpl" type="Templates:TciValueTemplate"

minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliMChecked\_c">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="msgValue" type="Values:Value"/>

<xsd:element name="msgTmpl" type="Templates:TciValueTemplate"

minOccurs="0"/>

<xsd:element name="from" type="Types:TriComponentIdType" minOccurs="0"/>

<xsd:element name="fromTmpl" type="Templates:TciNonValueTemplate"

minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrGetCallChecked\_m">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="parsTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>

<xsd:element name="addrValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="addressTmpl" type="Templates:TciValueTemplate"

minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrGetCallChecked\_c">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="parsTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>

<xsd:element name="from" type="Types:TriComponentIdType" minOccurs="0"/>

<xsd:element name="fromTmpl" type="Templates:TciNonValueTemplate"

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrGetReplyChecked\_m">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="parsTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>

<xsd:element name="replValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="replTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>

<xsd:element name="addrValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="addressTmpl" type="Templates:TciValueTemplate"

minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrGetReplyChecked\_c">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="parsTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>

<xsd:element name="replValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="replTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>

<xsd:element name="from" type="Types:TriComponentIdType" minOccurs="0"/>

<xsd:element name="fromTmpl" type="Templates:TciNonValueTemplate"

minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrCatchChecked\_m">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="excValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="excTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>

<xsd:element name="addrValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="addressTmpl" type="Templates:TciValueTemplate"

minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrCatchChecked\_c">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="excValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="excTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>

<xsd:element name="from" type="Types:TriComponentIdType" minOccurs="0"/>

<xsd:element name="fromTmpl" type="Templates:TciNonValueTemplate"

minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliCheckedAny\_m">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="addrValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="addressTmpl" type="Templates:TciValueTemplate"

minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliCheckedAny\_c">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="from" type="Types:TriComponentIdType"

minOccurs="0"/>

<xsd:element name="fromTmpl" type="Templates:TciNonValueTemplate"

minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliCheckMismatch\_m">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="addrValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="addressTmpl" type="Templates:TciValueTemplate"

minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliCheckMismatch\_c">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="from" type="Types:TriComponentIdType"

minOccurs="0"/>

<xsd:element name="fromTmpl" type="Templates:TciNonValueTemplate"

minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliRnd">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="val" type="Values:FloatValue"/>

<xsd:element name="seed" type="Values:FloatValue"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliEvaluate">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="name" type="Types:QualifiedName" />

<xsd:element name="evalResult" type="Values:Value" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>  
  
 <xsd:complexType name="tliCCall">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="comp" type="Types:TriComponentIdType"/>

<xsd:element name="name" type="Types:TciBehaviourIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliCCallTerminated">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="verdict" type="Values:VerdictValue" />

<xsd:element name="reason" type="SimpleTypes:TString" minOccurs="0"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="returnValue" type="Values:Value" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<!‑‑ control ‑‑>

<xsd:complexType name="tliCtrlStartWithParameters">

<xsd:complexContent>

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliCtrlTerminatedWithResult">

<xsd:complexContent>

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="result" type="Values:Value" minOccurs="0"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

# 12 C# mapping

## 12.1 Introduction

The C# mapping for the TTCN-3 Control Interface defines how the IDL [6] definitions described in clause 7 are mapped to the .Net language C# [13].

## 12.2 Names and scopes

### 12.2.1 Names

Although there are almost no conflicts between identifiers used in the IDL definition and C#, some naming translation rules are applied to the IDL identifiers.

C# interfaces are omitting the trailing Type used in the IDL definition. In addition to that, the capital letter "I" is added to the beginning of interface names.

EXAMPLE 1: The IDL type **TciTestCaseIdType** maps to ITciTestCaseId in C#.

C# names of enumerated items start with a capital letter and the remaining letters are low-case letters. If the enumerated item name is composed of several worlds, each word starts with a capital letter.

EXAMPLE 2: The identifier for boolean type defined in TciTypeClassType enumeration is BooleanType in C#.

The resulting mapping conforms to the standard C# coding conventions.

### 12.2.2 Scopes

The TCI interfaces are mapped to the namespace Etsi.Ttcn3.Tci. All IDL type declarations are mapped to C# interface declarations within this namespace. The associated assembly file is Etsi.Ttcn3.Tci.dll.

## 12.3 Null value mapping

The distinct value null specified in the IDL definition is equal to null in C#.

## 12.4 Type mapping

### 12.4.1 Basic type mapping

#### 12.4.1.0 Mapped types

Table 9 gives an overview on how the used basic IDL types are mapped to the .Net types.

Table : Basic type mapping

| IDL Type | C# Type/Interface |
| --- | --- |
| Boolean | bool |
| TChar | char |
| TFloat | double |
| TInteger | int / TciVerdict |
| TString | string |
| TStringSeq | string[] |
| TUniversalChar | uint |

**TBoolean**The IDL TBoolean type is mapped to the C# type bool.

**TFloat**  
The IDL TFloat type is mapped to the C# type double.

**TChar**The IDL TChar type is mapped to the C# type char.

**TInteger**The IDL TInteger type is usually mapped to the C# type int. Only in case of operations defined for the IDL type TciVerdictValue, the IDL TInteger type is mapped to Etsi.Ttcn3.Tci.TciVerdict enumeration.

**TString**The IDL TString type is mapped to the C# class string without range checking or bounds for characters in the string. All possible strings defined in TTCN-3 can be converted to C# string class.

**TStringSeq**The IDL TStringSeq type is mapped to a string array.

**TUniversalChar**  
The IDL TUniversalChar type is mapped to the C# type uint. The integer uses the canonical form as defined in ISO/IEC 10646 [5], clause 6.2.

#### 12.4.1.1 TciVerdict

In case of verdict operations, the IDL TInteger type is mapped to the **TciVerdict** enumeration. This enumeration is defined as follows:

public enum TciVerdict {   
 None = 0,   
 Pass = 1,   
 Inconc = 2,   
 Fail = 3,   
 Error = 4  
 User\_Error = 5  
}

### 12.4.2 Structured type mapping

#### 12.4.2.0 Mapping rules

The TCI IDL description defines user-defined types as native types. In the C# mapping, these types are mapped to C# interfaces. The interfaces define methods and properties being available for classes implementing this interface.

#### 12.4.2.1 TciParameterPassingModeType

**TciParameterPassingModeType** is mapped to the following enumeration:

public enum TciParameterPassingMode {   
 TciIn = 0,   
 TciInOut = 1,   
 TciOut = 2  
}

#### 12.4.2.2 TciParameterType

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

public interface ITciParameter {  
 string ParameterName { get; set; }  
 TciParameterPassingMode ParameterPassingMode { get; set; }  
 ITciValue Parameter { get; set; }  
}

**Members:**

* ParameterName  
  Gets or sets the parameter name.
* ParameterPassingMode  
  Gets or sets the parameter passing mode of this parameter.
* Parameter  
  Used for getting or setting value of the parameter. The parameter can be an instance of ITciValue or the distinct value null.

#### 12.4.2.3 TciParameterListType

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

public interface ITciParameterList: System.Collections.IEnumerable {  
 int Size { get; }  
 bool IsEmpty { get; }  
 ITciParameter this[int index] { get; }  
 void Clear();  
 void Add(ITciParameter comp);  
}

**Members:**

* Size  
  Returns the number of parameters in this list.
* IsEmpty  
  Returns true if this list contains no parameters.
* GetEnumerator  
  Inherited from IEnumerable. Returns an enumerator for this object and allows to use the list in a foreach loop.
* operator  
  Returns a ITciParameter instance at the specified position. IndexOutOfRangeException is thrown if the index is less than zero or greater or equal to the list size.
* Clear  
  Removes all parameters from the list.
* Add  
  Adds a parameter to the end of the list.

#### 12.4.2.4 TciTypeClassType

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

public enum TciTypeClass {  
 Address = 0,  
 Anytype = 1,  
 Bitstring = 2,   
 BooleanType = 3,  
 Charstring = 5,  
 Component = 6,  
 Enumerated = 7,  
 Float = 8,  
 Hexstring = 9,  
 IntegerType = 10,   
 Octetstring = 12,  
 Record = 13,  
 RecordOf = 14,  
 Array = 15  
 Set = 16,  
 SetOf = 17,  
 Union = 18,  
 UniversalCharstring = 20,  
 Verdict = 21  
}

#### 12.4.2.5 TciTestComponentKindType

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

public enum TciTestComponentKind {   
 TciCtrlComp = 0,   
 TciMtcComp = 1,   
 TciPtcComp = 2,  
 TciSystemComp = 3,   
 TciAliveComp = 4   
}

#### 12.4.2.6 TciBehaviourIdType

**TciBehaviourIdType** C# mapping is derived from the IQualifiedName interface:

public interface ITciBehaviourId : IQualifiedName {}

#### 12.4.2.7 TciTestCaseIdType

**TciTestCaseIdType** C# mapping is derived from the IQualifiedName interface:

public interface ITciTestCaseId : IQualifiedName {}

#### 12.4.2.8 TciTestCaseIdListType

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

public interface ITciTestCaseIdList: System.Collections.IEnumerable {  
 int Size { get; }  
 bool IsEmpty { get; }  
 ITciTestCaseId this[int index] { get; }  
}

**Members:**

* Size  
  Returns the number of test case identifiers in this list.
* IsEmpty  
  Returns true if this list contains no parameters.
* GetEnumerator  
  Inherited from IEnumerable. Returns an enumerator for this object and allows to use the list in a foreach loop.
* operator  
  Returns a ITciTestCaseId instance at the specified position. IndexOutOfRangeException is thrown if the index is less than zero or greater or equal to the list size.

#### 12.4.2.9 TciModuleIdType

**TciModuleIdType** C# mapping is derived from the IQualifiedName interface:

public interface ITciModuleId : IQualifiedName {   
}

#### 12.4.2.10 TciModuleIdListType

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

public interface ITciModuleIdList: System.Collections.IEnumerable {  
 int Size { get; }  
 bool IsEmpty { get; }  
 ITciModuleId this[int index] { get; }  
}

**Members:**

* Size  
  Returns the number of module identifiers in this list.
* IsEmpty  
  Returns true if this list contains no parameters.
* GetEnumerator  
  Inherited from IEnumerable. Returns an enumerator for this object and allows to use the list in a foreach loop.
* operator  
  Returns a ITciModuleId instance at the specified position. IndexOutOfRangeException is thrown if the index is less than zero or greater or equal to the list size.

#### 12.4.2.11 TciModuleParameterIdType

**TciModuleIdType** C# mapping is derived from the IQualifiedName interface:

public interface ITciModuleParameterId : IQualifiedName {  
}

#### 12.4.2.12 TciModuleParameterType

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

public interface ITciModuleParameter {  
 ITciModuleParameterId ModuleParameterName { get; }  
 ITciValue DefaultValue { get; }  
}

**Members:**

* ModuleParameterName  
  Returns the qualified module parameter name as defined in the TTCN-3 specification.
* DefaultValue  
  Returns the default value of this TciModuleParameter or the distinct value null if the default value is not specified.

#### 12.4.2.13 TciModuleParameterListType

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

public interface ITciModuleParameterList: System.Collections.IEnumerable {  
 int Size { get; }  
 bool IsEmpty { get; }  
 ITciModuleParameter this[int index] { get; }  
}

**Members:**

* Size  
  Returns the number of module identifiers in this list.
* IsEmpty  
  Returns true if this list contains no parameters.
* GetEnumerator()  
  Inherited from IEnumerable. Returns an enumerator for this object and allows to use the list in a foreach loop.
* Indexing operator  
  Returns a ITciModuleId instance at the specified position. IndexOutOfRangeException is thrown if the index is less than zero or greater or equal to the list size.

#### 12.4.2.14 TciParameterTypeType

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

public interface ITciParameterType {  
 string ParameterName {get; }  
 ITciType ParameterType { get; }  
 TciParameterPassingMode ParameterPassingMode { get; }  
}

**Members:**

* ParameterName  
  Returns the name of the parameter.
* ParameterType  
  Returns the type of the parameter.
* ParameterPassingMode  
  Returns the passing mode of this parameter.

#### 12.4.2.15 TciParameterTypeListType

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

public interface ITciParameterTypeList: System.Collections.IEnumerable {  
 int Size { get; }  
 bool IsEmpty { get; }  
 ITciParameterType this[int index] { get; }  
}

**Members:**

* Size  
  Returns the number of parameters in this list.
* IsEmpty  
  Returns true if this list contains no parameters.
* GetEnumerator  
  Inherited from IEnumerable. Returns an enumerator for this object and allows to use the list in a foreach loop.
* Indexing operator  
  Returns a ITciParameter instance at the specified position. IndexOutOfRangeException is thrown if the index is less than zero or greater or equal to the list size.
* Clear  
  Removes all parameters from the list.
* Add  
  Adds a parameter to the end of the list.

#### 12.4.2.16 TciMatchingTypeType

**TciMatchingTypeType** is mapped to the following enumeration:

public enum TciMatchingType

{  
 TemplateList = 0,  
 ComplementedList = 1,  
 AnyValue = 2,   
 AnyValueOrNone = 3,  
 ValueRange = 4,

Subset = 5,  
 Superset = 6,  
 AnyElement = 7,  
 AnyElementsOrNone = 8,  
 Pattern = 9,  
 MatchDecodedContent = 10,

OmitTemplate = 11  
}

#### 12.4.2.17 LengthRestriction

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

public interface ITciLengthRestriction

{

long LowerBoundary { get; set; }

long UpperBoundary { get; set; }

bool IsUpperBoundaryInfinity { get; set; }

}

**Methods:**

* LowerBoundary Gets or sets the lower boundary of the length restriction.
* UpperBoundary Gets or sets the upper boundary of the length restriction.
* IsUpperBoundaryInfinity Gets or sets the upper boundary infinity setting.

#### 12.4.2.18 Permutation

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

public interface ITciPermutation

{

long StartPosition { get; set; }

long Length { get; set; }

}

**Methods:**

* StartPosition Gets or sets the position of the first item of the permutation in the RecordOfValue.
* Length Gets or sets the number of elements or matching mechanisms of the RecordOfValue that are included in the permutation.

#### 12.4.2.19 RangeBoundary

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

public interface ITciRangeBoundary

{

ITciValue Boundary { get; set; }

bool IsInclusive { get; set; }

bool IsInfinity { get; set; }

}

**Methods:**

* Boundary Gets or sets the boundary value.
* IsInclusive Gets or sets whether the boundary value is a part of the allowed range or not.
* IsInfinity Gets or sets whether the boundary is infinity.

### 12.4.3 Abstract type mapping

#### 12.4.3.0 Mapping rules

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

#### 12.4.3.1 Type

The IDL type **Type** is mapped to the following interface:

public interface ITciType {  
 ITciModuleId DefiningModule { get; }  
 string Name { get; }  
 TciTypeClass TypeClass { get; }  
 ITciValue NewInstance();  
 ITciMatchingMechanism NewTemplate(TciMatchingType matchingType);  
 string TypeEncoding { get; }  
 string TypeEncodingVariant { get; }  
 string[] EncodeAttributes { get; }  
 string[] GetVariantAttributes(string encoding);  
 string[] TypeExtension { get; }  
 ITciValue ParseValue (string val);  
 ITciRangeBoundary LowerTypeBoundary { get; }

ITciRangeBoundary UpperTypeBoundary { get; }

ITciLengthRestriction TypeLengthRestriction { get; }

ITciMatchingMechanism TypeMatchingMechanism { get; }  
}

**Members:**

* DefiningModule  
  Returns the module identifier of the module the type has been defined in. If the type represents a TTCN-3 base type the distinct value null will be returned.
* Name  
  Returns name of the type as defined in the TTCN-3 module.
* TypeClass  
  Returns the type class of the respective type.
* NewInstance  
  Returns a freshly created value of the given type. This initial value of the created value is undefined.
* NewTemplate  
  Returns a freshly created matching mechanism of this type. The matchingType parameter determines what kind of matching mechanism will be created. If the created matching mechanism contains additional data properties, these properties are uninitialized in the created matching mechanism.
* TypeEncoding  
  Returns the type encoding attribute as defined in the TTCN-3 module, if any. If no encoding attribute has been defined, the distinct value null will be returned.
* TypeEncodingVariant  
  This property returns the type encoding variant attribute as defined in TTCN-3, if any. If no encoding variant attribute has been defined, the distinct value null will be returned.
* EncodeAttributes  
  Returns all encode attributes of the type as defined in the TTCN-3 module. If no encode attribute has been defined, the distinct value null is returned.
* GetVariantAttributes  
  Returns all variant attributes of the type as defined in the TTCN-3 module. If no encoding attribute has been defined, the distinct value null is returned.
* TypeExtension  
  Returns the type extension attributes as defined in the TTCN-3 module. If no extension attributes have been defined, the distinct value null will be returned.
* ParseValue   
  Parses the value provided in the parameter and in case of successful parsing returns a Value object representing the parsed value. In case of an error or if value parsing is not supported by the tool, the method returns null.
* LowerTypeBoundary  
  Gets the lower range boundary of the type or null.
* UpperTypeBoundary  
  Gets the upper range boundary of the type or null.
* TypeLengthRestriction  
  Gets the length restriction of the type or null.
* TypeMatchingMechanism  
  Gets the matching mechanism restricting the type or null.

### 12.4.4 Abstract value mapping

#### 12.4.4.0 Mapping rules

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

#### 12.4.4.1 Value

The IDL type **Value** is mapped to the following interface:

public interface ITciValue {  
 ITciType Type { get; }  
 bool NotPresent { get; }  
 string ValueEncoding { get; }  
 string ValueEncodingVariant { get; }  
 string[] EncodeAttributes { get; }  
 string[] GetVariantAttributes(string encoding);  
 bool IsMatchingSymbol { get; }  
 string ValueToString();  
 bool IsLazy { get; }  
 bool IsFuzzy { get; }  
 bool IsEvaluated { get; }

ITciLengthRestriction LengthRestriction { get; set; }

ITciLengthRestriction NewLengthRestriction ();

bool IsIfPresentEnabled { get; set; }

ITciRangeBoundary LowerTypeBoundary { get; }

ITciRangeBoundary UpperTypeBoundary { get; }

ITciLengthRestriction TypeLengthRestriction { get; }

ITciMatchingMechanism TypeMatchingMechanism { get; }  
 bool IsOptional { get; }

}

**Members:**

* Type  
  Returns the type of the specified value.
* NotPresent  
  Returns true if the specified value is omit, false otherwise.
* ValueEncoding  
  This property returns the value encoding attribute as defined in TTCN-3, if any. If no encoding attribute has been defined the distinct value null will be returned.
* ValueEncodingVariant  
  This property returns the value encoding variant attribute as defined in TTCN-3, if any. If no encoding variant attribute has been defined the distinct value null will be returned.
* EncodeAttributes  
  Returns all encode attributes of the value as defined in the TTCN-3 module. If no encode attribute has been defined, the distinct value null is returned.
* GetVariantAttributes  
  Returns all variant attributes of the value as defined in the TTCN-3 module. If no encoding attribute has been defined, the distinct value null is returned.
* IsMatchingSymbol  
  Returns true if the specified value is a matching symbol (see clause 7.2.2.2.1 for more details), false otherwise.
* ValueToString  
  Returns the same string as produced by the any2unistr predefined function with the specified value as its parameter.
* IsLazy  
  Returns true if the specified value is @lazy, false otherwise.
* IsFuzzy  
  Returns true if the specified value is @fuzzy, false otherwise.
* IsEvaluated  
  Returns true if the specified value contains an evaluation result, false otherwise (see clause 7.2.2.2.1 for more details).
* LengthRestriction  
  Gets or sets a length restriction matching attribute attached to the value.
* NewLengthRestriction  
  Creates a new instance of the LengthRestriction interface.
* IfPresentEnabled  
  Gets or sets whether the ifpresent matching attribute is attached to the value or not.
* LowerTypeBoundary  
  Gets the value's type's lower range boundary or null.
* UpperTypeBoundary  
  Gets the value's type's upper range boundary or null.
* TypeLengthRestriction  
  Gets the value's type's length restriction or null.
* TypeMatchingMechanism  
  Gets the values's type restriction matching mechanism.
* IsOptional  
  Returns whether the value is either an optional field or a template without the present or value restriction.

#### 12.4.4.2 IntegerValue

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

public interface ITciIntegerValue : ITciValue {  
 long IntegerValue { get; set; }  
 string StringValue { get; set; }  
}

**Members:**

* IntegerValue  
  Gets or sets the numeric value of the object. In case the numeric value exceeds the allowed value range of the long type, long.MaxValue or long.MinValue is returned.
* StringValue  
  Get or sets the value of the object. The string assigned to the property shall have the same format as TTCN-3 integer literals. The integer literal can be optionally preceded by a sign character ('+' or '-').

#### 12.4.4.3 FloatValue

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

public interface ITciFloatValue : ITciValue {  
 double FloatValue { get; set; }  
 string StringValue { get; set; }  
}

**Members:**

* FloatValue  
  Gets or sets the numeric value of the object. In case the numeric value exceeds the allowed value range of the double type, double.MaxValue or double.MinValue is returned.
* StringValue  
  Get or sets the value of the object. The string assigned to the property shall have the same format as TTCN-3 float literals. The float literal can be optionally preceded by a sign character ('+' or '-').

#### 12.4.4.4 BooleanValue

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

public interface ITciBooleanValue : ITciValue {  
 bool BooleanValue { get; set; }  
}

**Members:**

* BooleanValue  
  Gets or sets the boolean value of the object.

#### 12.4.4.5 CharstringValue

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

public interface ITciCharstringValue : ITciValue {  
 string StringValue { get; set; }  
 char this[int position] { get; set; }  
 int Length { get; set; }  
}

**Members:**

* StringValue  
  Gets or sets the string value of the TTCN-3 charstring. Strings assigned to this property shall contain only characters allowed in TTCN-3 charstring type.
* Indexing operator  
  Get or sets the character value of the TTCN-3 charstring at the specified position. IndexOutOfRangeException is thrown if the position is less than zero or greater or equal to the string length.
* Length  
  Gets or sets the length of this ITciCharstringValue in characters. The property returns zero if the value of this object is omit. In case the new length is greater than the length of the current string, characters with ordinal value 0 are added to the end of the string. If the new length is less than the length of the current string, the current string is truncated.

#### 12.4.4.6 BitstringValue

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

public interface ITciBitstringValue : ITciValue {  
 string StringValue { get; set; }  
 byte this[int position] { get; set; }  
 int Length { get; set; }  
 System.IO.Stream GetInputStream ();

void SetInputStream(System.IO.Stream stream, int numberOfBits);  
 bool IsMatchingAt (int position);

ITciMatchingMechanism GetMatchingAt (int position);

void SetMatchingAt (int position, ITciMatchingMechanism template);

}

**Members:**

* StringValue  
  Gets or sets the string value of the TTCN-3 bitstring. The only allowed characters in the string passed to this property are '0' and '1'. The string returned by the property contains a sequence of '0' and '1' digits. In some cases, the message data are available in the form of a stream and cannot be converted into a string by TCI (e.g. because of memory restrictions). In such cases, this property returns null and the GetInputStream method shall be used for reading the data.
* Indexing operator  
  Get or sets the value of the bit at the specified position. All non-zero values shall be interpreted as if the bit was present. IndexOutOfRangeException is thrown if the position is less than zero or greater or equal to the string length.
* Length  
  Gets or sets the length of this ITciBitstringValue in bits. The property returns zero if the value of this object is omit. In case the new length is greater than the length of the current bitstring, the bitstring is padded with empty bits. If the new length is less than the length of the current bitstring, the current bitstring is truncated.
* GetInputStream  
  Returns the bits in the form of an input stream. Repeated calls to the same method return different stream instances. The method returns null if the bitstring contains matching symbols.
* SetInputStream  
  Sets the value of this BitstringValue by providing a source stream that is used by the BitstringValue object to read the value content. Values set this way do not contain matching symbols.
* IsMatchingAt  
  Returns true if the item at position of this TTCN‑3 bitstring is a matching mechanism inside a value (AnyElement, AnyElementsOrNone) and false otherwise.
* GetMatchingAt  
  If the position of this TTCN‑3 bitstring contains a matching mechanism inside a value (AnyElement, AnyElementsOrNone), the method returns it. Otherwise the distinct value null is returned.
* SetMatchingAt  
  Sets a matching mechanism at position. Only two matching mechanisms are allowed: AnyElement and AnyElementsOrNone.

#### 12.4.4.7 OctetstringValue

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

public interface ITciOctetstringValue : ITciValue {  
 string StringValue { get; set; }  
 byte this[int position] { get; set; }  
 int Length { get; set; }  
 System.IO.Stream GetInputStream ();

void SetInputStream(System.IO.Stream stream);  
 bool IsMatchingAt (int position);

ITciMatchingMechanism GetMatchingAt (int position);

void SetMatchingAt (int position, ITciMatchingMechanism template);

}

**Members:**

* StringValue  
  Gets or sets the string value of the TTCN-3 octetstring. The only allowed characters in the string passed to this property are hexadecimal digits. The length of the string passed to this property shall be even. The string returned by this property is a sequence of pairs of hexadecimal digits. In some cases, the message data are available in the form of a byte stream and cannot be converted into a string by TCI (e.g. because of memory restrictions). In such cases, this property returns null and the GetInputStream method shall be used for reading the data.
* Indexing operator  
  Get or sets the value of the octet at the specified position. IndexOutOfRangeException is thrown if the position is less than zero or greater or equal to the string length.
* Length  
  Gets or sets the length of this ITciOctetstringValue in octets. The property returns zero if the value of this object is omit. In case the new length is greater than the length of the current octetstring, the octetstring is padded with empty octets. If the new length is less than the length of the current octetstring, the current octetstring is truncated.
* GetInputStream  
  Returns the octets in the form of an input stream. Repeated calls to the same method return different stream instances. The method returns null if the octetstring contains matching symbols.
* SetInputStream  
  Sets the value of this OctetstringValue by providing a source stream that is used by the OctetstringValue object to read the value content. Values set this way do not contain matching symbols.
* IsMatchingAt  
  Returns true if the item at position of this TTCN‑3 octetstring is a matching mechanism inside a value (AnyElement, AnyElementsOrNone) and false otherwise.
* GetMatchingAt  
  If the position of this TTCN‑3 octetstring contains a matching mechanism inside a value (AnyElement, AnyElementsOrNone), the method returns it. Otherwise the distinct value null is returned.
* SetMatchingAt  
  Sets a matching mechanism at position. Only two matching mechanisms are allowed: AnyElement and AnyElementsOrNone.

#### 12.4.4.8 UniversalCharstringValue

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

public interface ITciUniversalCharstringValue : ITciValue {  
 string StringValue { get; set; }  
 uint this[int position] { get; set; }  
 int Length { get; set; }  
}

**Members:**

* StringValue  
  Gets or sets the string value of the TTCN-3 universal charstring. If the TTCN-3 universal charstring value contains characters that have higher ordinal value than char.MaxValue, these characters will be represented by a character 0xFFFD (the Unicode replacement character) in the string returned by this property.
* Indexing operator  
  Get or sets the character value of the TTCN-3 universal charstring at the specified position. The unsigned number used by this property is character ordinal value. IndexOutOfRangeException is thrown if the position is less than zero or greater or equal to the string length.
* Length  
  Gets or sets the length of this ITciUniversalCharstringValue in characters. The property returns zero if the value of this object is omit. In case the new length is greater than the length of the current string, characters with ordinal value 0 are added to the end of the string. If the new length is less than the length of the current string, the current string is truncated.

#### 12.4.4.9 HexstringValue

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

public interface ITciHexstringValue : ITciValue {  
 string StringValue { get; set; }  
 byte this[int position] { get; set; }  
 int Length { get; set; }  
 System.IO.Stream GetInputStream ();

void SetInputStream(System.IO.Stream stream, int numberOfItems);  
 bool IsMatchingAt (int position);

ITciMatchingMechanism GetMatchingAt (int position);

void SetMatchingAt (int position, ITciMatchingMechanism template);

}

**Members:**

* StringValue  
  Gets or sets the string value of the TTCN-3 hexstring. The only allowed characters in the string passed to this property are hexadecimal digits. The string returned by this property is a sequence of hexadecimal digits. In some cases, the message data are available in the form of a stream and cannot be converted into a string by TCI (e.g. because of memory restrictions). In such cases, this property returns null and the GetInputStream method shall be used for reading the data.
* Indexing operator  
  Get or sets the hex digit at the specified position. Only the lower four bits of the passed value are used in this assignment. The upper four bits as ignored. IndexOutOfRangeException is thrown if the position is less than zero or greater or equal to the string length.
* Length  
  Gets or sets the length of this ITciHexstringValue in hex digits. The property returns zero if the value of this object is omit. In case the new length is greater than the length of the current hexstring, the hexstring is padded with zeroes. If the new length is less than the length of the current hexstring, the current hexstring is truncated.
* GetInputStream  
  Returns the content in the form of an input stream. Repeated calls to the same method return different stream instances. The method returns null if the hexstring contains matching symbols.
* SetInputStream  
  Sets the value of this HexstringValue by providing a source stream that is used by the HexstringValue object to read the value content. Values set this way do not contain matching symbols.
* IsMatchingAt  
  Returns true if the item at position of this TTCN‑3 hexstring is a matching mechanism inside a value (AnyElement, AnyElementsOrNone) and false otherwise.
* GetMatchingAt  
  If the position of this TTCN‑3 hexstring contains a matching mechanism inside a value (AnyElement, AnyElementsOrNone), the method returns it. Otherwise the distinct value null is returned.
* SetMatchingAt  
  Sets a matching mechanism at position. Only two matching mechanisms are allowed: AnyElement and AnyElementsOrNone.

#### 12.4.4.10 RecordValue

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

public interface ITciRecordValue : ITciValue {  
 ITciValue GetField(string fieldName);  
 void SetField(string fieldName, ITciValue value);  
 string[] GetFieldNames();  
 void SetFieldOmitted(string fieldName);  
}

**Members:**

* GetField  
  Returns the value of the field named fieldName. The return value is the common abstract base type ITciValue, as a record field can have any type defined in TTCN-3. If the field cannot be obtained from the record the distinct value null will be returned.
* SetField  
  Sets the field named fieldName of the record to value. No assumption shall be made on how a field is stored in a record. An internal implementation might choose to store a reference to this value or to copy the value. It is safe to assume that the value will be copied. Therefore it should be assumed that subsequent modifications of value will not be considered in the record.
* GetFieldNames  
  Returns an array of String of field names, the empty sequence, if the record has no fields.
* SetFieldOmitted  
  Sets the field named fieldName of the record to omit.

#### 12.4.4.11 RecordOfValue

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

public interface ITciRecordOfValue : ITciValue, System.Collections.IEnumerable {  
 ITciValue this[int position] { get; set; }  
 void AppendField(ITciValue value);  
 ITciType ElementType { get; }  
 int Length { get; set; }  
 int Offset { get; }  
 int PermutationCount { get; }

ITciPermutation GetPermutation (int index);

ITciPermutation NewPermutation ();

void DefinePermutation (ITciPermutation permutation);

void RemovePermutation (int index);

void ClearPermutations ();

}

**Members:**

* Indexing operator  
  Returns or sets the value of the record of at the specified position. The class of this property is the common abstract base interface ITciValue, as a record of can have fields of any type defined in TTCN‑3. When getting the value an ITciValue instance is returned only if position is between zero and (length  
   - 1). The distinct value null is returned otherwise. When setting the value, if position is greater than the current length, the record of will be extended to have the length (position + 1). The record of elements between the original position at length and (position - 1) will be set to omit. No assumption shall be made on how a field is stored in a record of. An internal implementation might choose to store a reference to this value or to copy the value. It is safe to assume that the value will be copied. Therefore it should be assumed that subsequent modifications of value will not be considered in the record of.
* GetEnumerator  
  Inherited from IEnumerable. Returns an enumerator for this object and allows to use the object in a foreach loop.
* appendField  
  Appends the value at the end of the record of, i.e. at position length. No assumption shall be made on how a field is stored in a record of. An internal implementation might choose to store a reference to this value or to copy the value. It is safe to assume that the value will be copied. Therefore it should be assumed that subsequent modifications of value will not be considered in the record of.
* ElementType  
  Returns the type of the elements of this record of.
* Length  
  Gets or sets the actual length of the record of value. When getting the length, zero is returned if the record of value is omit. When setting the length, if the new length is greater than the original length, newly created elements have the value omit. If the new length is less or equal than the original length, this operation will be ignored.
* Offset  
  Returns the lowest possible index. For a record of or set of value this is always 0. For an array value, this is the lower index bound used in the type definition.
* PermutationCount  
  Returns the number of permutations in the record of or array value.
* GetPermutation  
  Returns the permutation at the specified index. The allowed index range is from 0 to (getPermutationCount() – 1).
* NewPermutation  
  Creates a new instance of the Permutation interface.
* DefinePermutation  
  Creates permutation from existing elements of a record of value. The permutation parameter shall not include elements that are already a part of other existing permutations attached to the same record of. No elements are added to the record of by this operation.
* RemovePermutation  
  Removes the permutation at the specified index. The allowed index range is from 0 to (getPermutationCount() – 1). No elements are removed from the record of by this operation. When the operation completes, the existing elements at positions specified by the removed permutation do not belong to any permutation.
* ClearPermutations  
  Removes all permutations from the value. The elements that belonged to the removed permutation are not removed.

#### 12.4.4.12 UnionValue

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

public interface ITciUnionValue : ITciValue {  
 ITciValue GetVariant(string variantName);  
 void SetVariant(string variantName, ITciValue value);  
 string PresentVariantName { get; }  
 string[] GetVariantNames();  
}

**Members:**

* GetVariant  
  Returns the value of the TTCN-3 union variant, if variantName equals the result of GetPresentVariantName. The distinct value null is returned otherwise. variantName denotes the name of the union variant as defined in TTCN‑3.
* SetVariant  
  Sets variantName of the union to value. If variantName is not defined for this union this operation will be ignored. If another variant was selected the new variant will be selected instead.
* GetPresentVariantName  
  Returns the variant name that has a value in this union set as a string. The distinct value null will be returned if no variant is selected.
* GetVariantNames  
  Returns an array of string of variant names, the empty sequence, if the union has no fields. If the UnionValue represents the TTCN-3 anytype, i.e. the value of the Type property is TciTypeClass.Anytype, all predefined and user‑defined TTCN-3 types will be returned.

#### 12.4.4.13 EnumeratedValue

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

public interface ITciEnumeratedValue : ITciValue {  
 string EnumValue { get; set; }  
 int IntValue { get; set; }  
}

**Members:**

* EnumValue  
  Returns or sets the enumerated value. The value of the property is equal to the identifier in the TTCN‑3 specification. If the value assigned to the property is not an allowed value for this enumeration, the assignment will be ignored.
* IntValue  
  Returns or sets the integer value. This integer should equal the user-assigned integer value in the TTCN‑3 specification or the automatically assigned integer value. If the integer assigned to the property is not allowed for this enumeration, the assignment will be ignored.

#### 12.4.4.14 VerdictValue

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

public interface ITciVerdictValue : ITciValue {  
 TciVerdict Verdict { get; set; }  
}

**Members:**

* Verdict  
  Returns the value of this VerdictValue. Note that a VerdictValue can be set to any of the verdicts defined in the TciVerdict enumeration at any time. The VerdictValue does not perform any verdict calculations as defined in TTCN-3. For example, it is legal to set the VerdictValue first to TciVerdict.ErrorVerdict and then to TciVerdict.Pass.

#### 12.4.4.15 AddressValue

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

public interface ITciAddressValue : ITciValue {  
 ITciValue Address { get; set; }  
}

**Members:**

* Address  
  Gets or sets the value represented by this AddressValue.

### 12.4.5 Abstract template mapping

#### 12.4.5.0 Mapping rules

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

#### 12.4.5.1 MatchingMechanism

The IDL type **MatchingMechanism** is mapped to the following interface:

public interface ITciMatchingMechanism : ITciValue

{  
 TciMatchingType MatchingType { get; }  
}

**Members:**

* MatchingType  
  Returns the matching mechanism type.

#### 12.4.5.2 MatchingList

The IDL type **MatchingList** is mapped to the following interface:

public interface ITciMatchingList : ITciMatchingMechanism

{

int Size { get; }

ITciValue this[int position] { get; }

void Add (ITciValue item);

void Remove (int position);

void Clear ();

}

**Methods:**

* Size   
  Returns the number of items in the matching mechanism.
* Indexing operator  
  Returns a value or template at the specified position.
* Add  
  Adds a value or template to the matching mechanism.
* Remove  
  Removes a value or template from the specified position.
* Clear  
  Removes all values and templates from the matching mechanism.

#### 12.4.5.3 ValueRange

The IDL type **ValueRange** is mapped to the following interface:

public interface ITciValueRange : ITciMatchingMechanism

{

ITciRangeBoundary LowerBoundary { get; set; }

ITciRangeBoundary UpperBoundary { get; set; }

}

**Methods:**

* LowerBoundary  
  Gets of sets the lower boundary of the range.
* UpperBoundary  
  Gets or sets the upper boundary of the range.

#### 12.4.5.4 CharacterPattern

The IDL type **CharacterPattern** is mapped to the following interface:

public interface ITciCharacterPattern : ITciMatchingMechanism

{

ITciValue PatternString { get; set; }

}

**Methods:**

* PatternString Gets or sets the character pattern definition of this pattern (either a ITciCharstringValue or ITciUniversalCharstringValue).

#### 12.4.5.5 MatchDecodedContent

The IDL type **MatchDecodedContent** is mapped to the following interface:

public interface ITciMatchDecodedContent : ITciMatchingMechanism

{

ITciValue Content { get; set; }

}

**Methods:**

* Content Gets or sets the value or matching mechanism used as an argument of the decmatch matching mechanism.

### 12.4.6 Abstract logging types mapping

#### 12.4.6.0 Mapping rules

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

#### 12.4.6.1 TciValueTemplate

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

public interface ITciValueTemplate {  
 bool IsOmit { get; }  
 bool IsAny { get; }  
 bool IsAnyOrOmit { get; }  
 string TemplateDef { get; }  
}

**Members:**

* IsOmit  
  Returns true if the template is omit, false otherwise.
* IsAny  
  Returns true if the template is any, false otherwise.
* IsAnyOrOmit  
  Returns true if the template is AnyValueOrNone, false otherwise.
* TemplateDef  
  This property returns the template definition.

#### 12.4.6.2 TciNonValueTemplate

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

public interface ITciNonValueTemplate {  
 bool IsAny { get; }  
 bool IsAll { get; }  
 string TemplateDef { get; }  
}

**Members:**

* IsAny  
  Returns true if the template is any, false otherwise.
* IsAll  
  Returns true if the template is all, false otherwise.
* TemplateDef  
  This operation returns the template definition.

#### 12.4.6.3 TciValueList

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

public interface ITciValueList: IEnumerable {  
 int Size { get; }  
 bool IsEmpty { get; }  
 ITciValue this[int index] { get; }  
}

**Members:**

* Size  
  Returns the number of values in this list.
* IsEmpty  
  Returns true if this list contains no values.
* GetEnumerator  
  Inherited from IEnumerable. Returns an enumerator for this object and allows to use the list in a foreach loop.
* Indexing operator  
  Returns a ITciValue instance at the specified position. IndexOutOfRangeException is thrown if the index is less than zero or greater or equal to the list size.

#### 12.4.6.4 TciValueDifference

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

public interface ITciValueDifference {  
 ITciValue Value { get; }  
 ITciValueTemplate ValueTemplate { get; }  
 string Description { get; }  
}

**Members:**

* Value  
  Returns the value of this ITciValueDifference.
* Template  
  Returns the template of this ITciValueDifference.
* Description  
  Returns the description of the mismatch.

#### 12.4.6.5 TciValueDifferenceList

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

public interface ITciValueDifferenceList : IEnumerable {  
 int Size { get; }  
 bool IsEmpty { get; }  
 ITciValueDifference this[int index] { get; }  
}

**Members:**

* Size  
  Returns the number of differences in this list.
* IsEmpty  
  Returns true if this list contains no parameters.
* GetEnumerator  
  Inherited from IEnumerable. Returns an enumerator for this object and allows to use the list in a foreach loop.
* Indexing operator  
  Returns a ITciValueDifference instance at the specified position. IndexOutOfRangeException is thrown if the index is less than zero or greater or equal to the list size.

#### 12.4.6.6 TciStatusType

**TciStatusType** is mapped to the following enumeration:

public enum TciStatus {   
 TciOk = 0,   
 TciError = -1  
}

#### 12.4.6.7 ComponentStatusType

**ComponentStatusType** is mapped to the following enumeration:

public enum TciComponentStatus {   
 InactiveC = 0,   
 RunningC = 1,   
 StoppedC = 2,   
 KilledC = 3,   
 NullC = 4   
}

#### 12.4.6.8 TimerStatusType

**TimerStatusType** is mapped to the following enumeration:

public enum TciTimerStatus {   
 RunningT = 0,   
 InactiveT = 1,   
 ExpiredT = 2,   
 NullT = 3   
}

## 12.5 Mapping of interfaces

### 12.5.0 Calling rules

The TCI IDL definition defines four interfaces, the TCI-TM, the TCI-CH, the TCI-CD, and the TCI-TL interface. The operations are defined for different directions within this interface, i.e. some operations can only be called by the TTCN-3 Executable (TE), the System Adaptor (SA) or the Platform Adaptor (PA) on the Test Management and Control (TMC) while others can only be called by the TMC on the TE. This is reflected by dividing the TCI IDL interfaces in two sub interfaces, each suffixed by Required or Provided.

Table : TCI sub-interfaces

| Calling | Called | Interface |
| --- | --- | --- |
| TE | TMC | ITciTMProvided |
| TMC | TE | ITciTMRequired |
| TE | CD | ITciCDProvided |
| CD | TE | ITciCDRequired |
| TE | CH | ITciCHProvided |
| CH | TE | ITciCHRequired |
| TE, TMC, CD, CH, SA, PA | TL | ITciTLProvided |

For better readability, references to types and interfaces defined in the Etsi.Ttcn3.Tci namespace and Etsi.Ttcn3.Tri namespace (described in [3]) that appear in the following interface definitions use a simple identifier form instead of a fully qualified one. Names of these types and interfaces start with a prefix that can be used for identification of its origin.

### 12.5.1 TCI-TM interface

#### 12.5.1.1 TCI-TM provided

The **TCI-TM provided** interface is mapped to the following interface:

public interface ITciTMProvided {  
 void TciTestCaseStarted(ITciTestCaseId testCaseId,  
 ITciParameterList parameterList, double timer);  
 void TciTestCaseTerminated(ITciVerdictValue verdict,  
 ITciParameterList parameterList);  
 void TciControlTerminated();  
 ITciValue TciGetModulePar(ITciModuleParameterId parameterId);  
 void TciLog(ITriComponentId testComponentId,  
 string message);  
 void TciError(string message);  
 void TciControlTerminatedWithResult (ITciValue result,  
 ITciParameterList parameterList);  
}

#### 12.5.1.2 TCI-TM required

The **TCI-TM required** interface is mapped to the following interface:

public interface ITciTMRequired {  
 void TciRootModule(ITciModuleId moduleId);  
 ITciModuleIdList TciGetImportedModules();  
 ITciModuleParameterList TciGetModuleParameters(ITciModuleId moduleId);  
 ITciTestCaseIdList TciGetTestCases();  
 ITciParameterTypeList TciGetTestCaseParameters(ITciTestCaseId TestCaseId);  
 ITriPortIdList TciGetTestCaseTsi(  
 ITciTestCaseId testCaseId);  
 void TciStartTestCase(ITciTestCaseId testCaseId,  
 ITciParameterList parameterList);  
 void TciStopTestCase();  
 ITriComponentId TciStartControl();  
 void TciStopControl();  
 ITciParameterTypeList TciGetControlParameters ();  
 ITriComponentId TciStartControlWithParameters (ITciParameterList parameterList);  
}

### 12.5.2 TCI-CD interface

#### 12.5.2.1 TCI-CD provided

The **TCI-CD provided** interface is mapped to the following interface:

public interface ITciCDProvided {  
 ITciValue Decode(ITriMessage message,  
 ITciType decodingHypothesis);  
 ITriMessage Encode(ITciValue value);  
 int DecodeValue(ITriMessage message,  
 ITciType decodingHypothesis,

string decodingInfo,

out ITciValue decodedValue);  
 ITriMessage EncodeValue(ITciValue value,

string encodingInfo);  
}

#### 12.5.2.2 TCI-CD required

The **TCI-CD required** interface is mapped to the following interface:

public interface ITciCDRequired {  
 ITciType GetTypeForName(string typeName);  
 ITciType GetInteger();  
 ITciType GetFloat();  
 ITciType GetBoolean();  
 ITciType GetCharstring();  
 ITciType GetUniversalCharstring();  
 ITciType GetHexstring();  
 ITciType GetBitstring();  
 ITciType GetOctetstring();  
 ITciType GetVerdict();  
 void TciErrorReq(string message);  
}

### 12.5.3 TCI-CH interface

#### 12.5.3.1 TCI-CH provided

The **TCI-CH provided** interface is mapped to the following interface:

public interface ITciCHProvided {  
 void TciSendConnected(ITriPortId sender,  
 ITriComponentId receiver, ITciValue sendMessage);  
 void TciSendConnectedBC(ITriPortId sender,  
 ITciValue sendMessage);  
 void TciSendConnectedMC(ITriPortId sender,  
 ITriComponentIdList receivers,  
 ITciValue sendMessage);  
 void TciCallConnected(ITriPortId sender,  
 ITriComponentId receiver,  
 ITriSignatureId signature,  
 ITciParameterList parameterList);  
 void TciCallConnectedBC(ITriPortId sender,  
 ITriSignatureId signature,  
 ITciParameterList parameterList);  
 void TciCallConnectedMC(ITriPortId sender,  
 ITriComponentIdList receivers,   
 ITriSignatureId signature,  
 ITciParameterList parameterList);  
 void TciReplyConnected(ITriPortId sender,  
 ITriComponentId receiver,  
 ITriSignatureId signature,  
 ITciParameterList parameterList, ITciValue returnValue);  
 void TciReplyConnectedBC(ITriPortId sender,  
 ITriSignatureId signature,  
 ITciParameterList parameterList, ITciValue returnValue);  
 void TciReplyConnectedMC(ITriPortId sender,  
 ITriComponentIdList receivers,  
 ITriSignatureId signature,  
 ITciParameterList parameterList, ITciValue returnValue);  
 void TciRaiseConnected(ITriPortId sender,  
 ITriComponentId receiver,   
 ITriSignatureId signature, ITciValue except);  
 void TciRaiseConnectedBC(ITriPortId sender,  
 ITriSignatureId signature, ITciValue except);  
 void TciRaiseConnectedMC(ITriPortId sender,  
 ITriComponentIdList receivers,  
 ITriSignatureId signature, ITciValue except);  
 ITriComponentId TciCreateTestComponentReq(int kind,  
 ITciType componentType, string name, ITciValue hostId);  
 void TciStartTestComponentReq(ITriComponentId comp,  
 ITciBehaviourId behavior, ITciParameterList parameterList);  
 void TciStopTestComponentReq(ITriComponentId comp);  
 void TciConnectReq(ITriPortId fromPort,  
 ITriPortId toPort);  
 void TciDisconnectReq(ITriPortId fromPort,  
 ITriPortId toPort);  
 void TciTestComponentTerminatedReq(ITriComponentId comp,  
 ITciVerdictValue verdict);  
 bool TciTestComponentRunningReq(ITriComponentId comp);  
 ITriComponentId TciGetMmcReq();  
 void TciMapReq(ITriPortId fromPort,  
 ITriPortId toPort);  
 void TciMapParamReq(ITriPortId fromPort,  
 ITriPortId toPort, Etsi.Ttcn3.Tci.ITciParameterList parameterList);  
 void TciUnmapReq(ITriPortId fromPort,  
 ITriPortId toPort);  
 void TciUnmapParamReq(ITriPortId fromPort,  
 ITriPortId toPort, Etsi.Ttcn3.Tci.ITciParameterList parameterList);  
 void TciExecuteTestCaseReq(ITriComponentId component,  
 ITriPortIdList tsiPortList);  
 void TciResetReq();  
 bool TciTestComponentDoneReq(ITriComponentId component, out TciVerdict verdict);  
 void TciKillTestComponentReq(ITriComponentId component);  
 bool TciTestComponentAliveReq(ITriComponentId component);  
 bool TciTestComponentKilledReq(ITriComponentId component, out TciVerdict verdict);  
 void TciCallTestComponentReq (ITriComponentId comp,  
 ITciBehaviourId behavior, ITciParameterList parameterList);  
 void TciTestComponentCallTerminatedReq (ITriComponentId comp,  
 ITciVerdictValue verdict, ITciParameterList parameterList, ITciValue returnValue);  
}

#### 12.5.3.2 TCI-CH required

The **TCI-CH required** interface is mapped to the following interface:

public interface ITciCHRequired {  
 void TciEnqueueMsgConnected(ITriPortId sender,  
 ITriComponentId receiver,  
 ITciValue receivedMessage);  
 void TciEnqueueCallConnected(ITriPortId sender,  
 ITriComponentId receiver,  
 ITriSignatureId signature,  
 ITciParameterList parameterList);  
 void TciEnqueueReplyConnected(ITriPortId sender,  
 ITriComponentId receiver,  
 ITriSignatureId signature,  
 ITciParameterList parameterList, ITciValue returnValue);  
 void TciEnqueueRaiseConnected(ITriPortId sender,  
 ITriComponentId receiver,  
 ITriSignatureId signature, ITciValue except);  
 ITriComponentId TciCreateTestComponent(int kind,  
 ITciType componentType, string name);  
 void TciStartTestComponent(ITriComponentId comp,  
 ITciBehaviourId behavior, ITciParameterList parameterList);  
 void TciStopTestComponent(ITriComponentId comp);  
 void TciConnect(ITriPortId fromPort,  
 ITriPortId toPort);  
 void TciDisconnect(ITriPortId fromPort,  
 ITriPortId toPort);  
 void TciTestComponentTerminated(ITriComponentId comp,  
 ITciVerdictValue verdict);  
 bool TciTestComponentRunning(ITriComponentId comp);  
 bool TciTestComponentDone(ITriComponentId comp, out TciVerdict verdict);  
 ITriComponentId TciGetMtc();  
 void TciExecuteTestCase (ITciTestCaseId testCaseId,  
 ITriPortIdList tsiPortList);  
 void TciReset();  
 void TciMap(ITriPortId fromPort,  
 ITriPortId toPort);  
 void TciMapParam(ITriPortId fromPort,  
 ITriPortId toPort, Etsi.Ttcn3.Tci.ITciParameterList parameterList);  
 void TciUnmap(ITriPortId fromPort,  
 ITriPortId toPort);  
 void TciUnmapParam(ITriPortId fromPort,  
 ITriPortId toPort, Etsi.Ttcn3.Tci.ITciParameterList parameterList);  
 void TciKillTestComponent(ITriComponentId component);  
 bool TciTestComponentAlive (ITriComponentId component);  
 bool TciTestComponentKilled(ITriComponentId component, out TciVerdict verdict);  
 void TciCallTestComponent (ITriComponentId comp,  
 ITciBehaviourId behavior, ITciParameterList parameterList);  
 void TciTestComponentCallTerminated (ITriComponentId comp,  
 ITciVerdictValue verdict, ITciParameterList parameterList, ITciValue returnValue);  
}

### 12.5.4 TCI-TL interface

#### 12.5.4.1 TCI-TL provided

The **TCI-TL provided** interface is mapped to the following interface:

public interface ITciTLProvided {  
 void TliTcExecute(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, ITciTestCaseId tcId,  
 ITciParameterList tciPars, ITriTimerDuration dur);  
 void TliTcStart(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, ITciTestCaseId tcId,  
 ITciParameterList tciPars, ITriTimerDuration dur);  
 void TliTcStop(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, string reason);  
 void TliTcStarted(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, ITciTestCaseId tcId,  
 ITciParameterList tciPars, ITriTimerDuration dur);  
 void TliTcTerminated(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, ITciTestCaseId tcId,  
 ITciParameterList tciPars, ITciVerdictValue verdict,  
 string reason);  
 void TliCtrlStart(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c);  
 void TliCtrlStop(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c);  
 void TliCtrlTerminated(string am, System.DateTime ts, string src, int  
 line, ITriComponentId c);  
 void TliMSend\_m(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, ITriPortId at,  
 ITriPortId to, ITciValue msgValue,  
 ITciValue addrValue, TciStatus encoderFailure,  
 ITriMessage msg, ITriAddress address,  
 TriStatus transmissionFailure);  
 void TliMSend\_m\_BC(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, ITriPortId at,  
 ITriPortId to, ITciValue msgValue,  
 TciStatus encoderFailure, ITriMessage msg,  
 TriStatus transmissionFailure);  
 void TliMSend\_m\_MC(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, ITriPortId at,  
 ITriPortId to, ITciValue msgValue,  
 ITciValueList addrValues, TciStatus encoderFailure,  
 ITriMessage msg,  
 ITriAddressList addresses,  
 TriStatus transmissionFailure);  
 void TliMSend\_c(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, ITriPortId at,  
 ITriPortId to, ITciValue msgValue,  
 TriStatus transmissionFailure);  
 void TliMSend\_c\_BC(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, ITriPortId at,  
 ITriPortIdList to, ITciValue msgValue,  
 TriStatus transmissionFailure);  
 void TliMSend\_c\_MC(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, ITriPortId at,  
 ITriPortIdList to, ITciValue msgValue,  
 TriStatus transmissionFailure);  
 void TliMDetected\_m(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, ITriPortId at,  
 ITriPortId from, ITriMessage msg,  
 ITriAddress address);  
 void TliMDetected\_c(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, ITriPortId at,  
 ITriPortId from, ITciValue msgValue);  
 void TliMMismatch\_m(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, ITriPortId at,  
 ITciValue msgValue, ITciValueTemplate msgTmpl,  
 ITciValueDifferenceList diffs, ITciValue address,  
 ITciValueTemplate addressTmpl);  
 void TliMMismatch\_c(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, ITriPortId at,  
 ITciValue msgValue, ITciValueTemplate msgTmpl,  
 ITciValueDifferenceList diffs, ITriComponentId from,  
 ITciNonValueTemplate fromTmpl);  
 void TliMReceive\_m(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, ITriPortId at,  
 ITciValue msgValue, ITciValueTemplate msgTmpl, ITciValue address,  
 ITciValueTemplate addressTmpl);  
 void TliMReceive\_c(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, ITriPortId at,  
 ITciValue msgValue, ITciValueTemplate msgTmpl,  
 ITriComponentId fromComp,  
 ITciNonValueTemplate fromTmpl);  
 void TliPrCall\_m(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, ITriPortId at,  
 ITriPortId to,  
 ITriSignatureId signature,  
 ITciParameterList tciPars, ITciValue addrValue,  
 TciStatus encoderFailure, ITriParameterList triPars,  
 ITriAddress address,  
 TriStatus transmissionFailure);  
 void TliPrCall\_m\_BC(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, ITriPortId at,  
 ITriPortId to,  
 ITriSignatureId signature, ITciParameterList tciPars,  
 TciStatus encoderFailure, ITriParameterList triPars,  
 TriStatus transmissionFailure);  
 void TliPrCall\_m\_MC(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, ITriPortId at,  
 ITriPortId to,  
 ITriSignatureId signature, ITciParameterList tciPars,  
 ITciValueList addrValues, TciStatus encoderFailure,  
 ITriParameterList triPars,  
 ITriAddressList addresses,  
 TriStatus transmissionFailure);  
 void TliPrCall\_c(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, ITriPortId at,  
 ITriPortId to,  
 ITriSignatureId signature,  
 ITciParameterList tciPars,  
 TriStatus transmissionFailure);  
 void TliPrCall\_c\_BC(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, ITriPortId at,  
 ITriPortIdList to,  
 ITriSignatureId signature, ITciParameterList tciPars,  
 TriStatus transmissionFailure);  
 void TliPrCall\_c\_MC(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, ITriPortId at,  
 ITriPortIdList to,  
 ITriSignatureId signature, ITciParameterList tciPars,  
 TriStatus transmissionFailure);  
 void TliPrGetCallDetected\_m(string am, System.DateTime ts, string src,  
 int line, ITriComponentId c,  
 ITriPortId at, ITriPortId from,  
 ITriSignatureId signature,  
 ITriParameterList triPars,  
 ITriAddress address);  
 void TliPrGetCallDetected\_c(string am, System.DateTime ts, string src,  
 int line, ITriComponentId c,  
 ITriPortId at, ITriPortId from,  
 ITriSignatureId signature,  
 ITciParameterList tciPars);  
 void TliPrGetCallMismatch\_m(string am, System.DateTime ts, string src,  
 int line, ITriComponentId c,  
 ITriPortId at,  
 ITriSignatureId signature, ITciParameterList tciPars,  
 ITciValueTemplate parsTmpl, ITciValueDifferenceList diffs,  
 ITciValue address, ITciValueTemplate addressTmpl);  
 void TliPrGetCallMismatch\_c(string am, System.DateTime ts, string src,  
 int line, ITriComponentId c,  
 ITriPortId at,  
 ITriSignatureId signature,  
 ITciParameterList tciPars, ITciValueTemplate parsTmpl,  
 ITciValueDifferenceList diffs, ITriComponentId from,  
 ITciNonValueTemplate fromTmpl);  
 void TliPrGetCall\_m(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, ITriPortId at,  
 ITriSignatureId signature,  
 ITciParameterList tciPars, ITciValueTemplate parsTmpl,  
 ITciValue address, ITciValueTemplate addressTmpl);  
 void TliPrGetCall\_c(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, ITriPortId at,  
 ITriSignatureId signature,  
 ITciParameterList tciPars, ITciValueTemplate parsTmpl,  
 ITriComponentId from, ITciNonValueTemplate fromTmpl);  
 void TliPrReply\_m(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, ITriPortId at,  
 ITriPortId to,  
 ITriSignatureId signature, ITciParameterList tciPars,  
 ITciValue replValue, ITciValue addrValue, TciStatus encoderFailure,  
 ITriParameterList triPars,  
 ITriParameter repl,  
 ITriAddress address,  
 TriStatus transmissionFailure);  
 void TliPrReply\_m\_BC(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, ITriPortId at,  
 ITriPortId to,  
 ITriSignatureId signature, ITciParameterList tciPars,  
 ITciValue replValue, TciStatus encoderFailure,  
 ITriParameterList triPars,  
 ITriParameter repl,  
 TriStatus transmissionFailure);  
 void TliPrReply\_m\_MC(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, ITriPortId at,  
 ITriPortId to,  
 ITriSignatureId signature, ITciParameterList tciPars,  
 ITciValue replValue, ITciValueList addrValues,  
 TciStatus encoderFailure, ITriParameterList triPars,  
 ITriParameter repl,  
 ITriAddressList addresses,  
 TriStatus transmissionFailure);  
 void TliPrReply\_c(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, ITriPortId at,  
 ITriPortId to,  
 ITriSignatureId signature, ITciParameterList tciPars,  
 ITciValue replValue, TriStatus transmissionFailure);  
 void TliPrReply\_c\_BC(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, ITriPortId at,  
 ITriPortIdList to,  
 ITriSignatureId signature, ITciParameterList tciPars,  
 ITciValue replValue, TriStatus transmissionFailure);  
 void TliPrReply\_c\_MC(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, ITriPortId at,  
 ITriPortIdList to,  
 ITriSignatureId signature, ITciParameterList tciPars,  
 ITciValue replValue, TriStatus transmissionFailure);  
 void TliPrGetReplyDetected\_m(string am, System.DateTime ts, string src,  
 int line, ITriComponentId c,  
 ITriPortId at, ITriPortId from,  
 ITriSignatureId signature,  
 ITriParameterList triPars,  
 ITriParameter repl,  
 ITriAddress address);  
 void TliPrGetReplyDetected\_c(string am, System.DateTime ts, string src,  
 int line, ITriComponentId c,  
 ITriPortId at, ITriPortId from,  
 ITriSignatureId signature, ITciParameterList tciPars,  
 ITciValue replValue);  
 void TliPrGetReplyMismatch\_m(string am, System.DateTime ts, string src,  
 int line, ITriComponentId c,  
 ITriPortId at,  
 ITriSignatureId signature, ITciParameterList tciPars,  
 ITciValueTemplate parsTmpl, ITciValue replValue,  
 ITciValueTemplate replyTmpl, ITciValueDifferenceList diffs,  
 ITciValue address, ITciValueTemplate addressTmpl);  
 void TliPrGetReplyMismatch\_c(string am, System.DateTime ts, string src,  
 int line, ITriComponentId c,  
 ITriPortId at,  
 ITriSignatureId signature, ITciParameterList tciPars,  
 ITciValueTemplate parsTmpl, ITciValue replValue,  
 ITciValueTemplate replyTmpl, ITciValueDifferenceList diffs,  
 ITriComponentId from, ITciNonValueTemplate fromTmpl);  
 void TliPrGetReply\_m(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, ITriPortId at,  
 ITriSignatureId signature, ITciParameterList tciPars,  
 ITciValueTemplate parsTmpl, ITciValue replValue,  
 ITciValueTemplate replyTmpl, ITciValue address,  
 ITciValueTemplate addressTmpl);  
 void TliPrGetReply\_c(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, ITriPortId at,  
 ITriSignatureId signature,  
 ITciParameterList tciPars, ITciValueTemplate parsTmpl,  
 ITciValue replValue, ITciValueTemplate replyTmpl,  
 ITriComponentId from, ITciNonValueTemplate fromTmpl);  
 void TliPrRaise\_m(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, ITriPortId at,  
 ITriPortId to,  
 ITriSignatureId signature, ITciParameterList tciPars,  
 ITciValue excValue, ITciValue addrValue, TciStatus encoderFailure,  
 ITriException exc,  
 ITriAddress address,  
 TriStatus transmissionFailure);  
 void TliPrRaise\_m\_BC(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, ITriPortId at,  
 ITriPortId to,  
 ITriSignatureId signature, ITciParameterList tciPars,  
 ITciValue excValue, TciStatus encoderFailure,  
 ITriException exc,  
 TriStatus transmissionFailure);  
 void TliPrRaise\_m\_MC(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, ITriPortId at,  
 ITriPortId to,  
 ITriSignatureId signature, ITciParameterList tciPars,  
 ITciValue excValue, ITciValueList addrValues,  
 TciStatus encoderFailure, ITriException exc,  
 ITriAddressList addresses,  
 TriStatus transmissionFailure);  
 void TliPrRaise\_c(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, ITriPortId at,  
 ITriPortId to,  
 ITriSignatureId signature, ITciParameterList tciPars,  
 ITciValue excValue, TriStatus transmissionFailure);  
 void TliPrRaise\_c\_BC(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, ITriPortId at,  
 ITriPortIdList to,  
 ITriSignatureId signature,  
 ITciParameterList tciPars, ITciValue excValue,  
 TriStatus transmissionFailure);  
 void TliPrRaise\_c\_MC(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, ITriPortId at,  
 ITriPortIdList to,  
 ITriSignatureId signature, ITciParameterList tciPars,  
 ITciValue excValue, TriStatus transmissionFailure);  
 void TliPrCatchDetected\_m(string am, System.DateTime ts, string src,  
 int line, ITriComponentId c,  
 ITriPortId at, ITriPortId from,  
 ITriSignatureId signature,  
 ITriException exc,  
 ITriAddress address);  
 void TliPrCatchDetected\_c(string am, System.DateTime ts, string src,  
 int line, ITriComponentId c,  
 ITriPortId at, ITriPortId from,  
 ITriSignatureId signature, ITciValue excValue);  
 void TliPrCatchMismatch\_m(string am, System.DateTime ts, string src,  
 int line, ITriComponentId c,  
 ITriPortId at,  
 ITriSignatureId signature, ITciValue excValue,  
 ITciValueTemplate excTmpl, ITciValueDifferenceList diffs,  
 ITciValue address, ITciValueTemplate addressTmpl);  
 void TliPrCatchMismatch\_c(string am, System.DateTime ts, string src,  
 int line, ITriComponentId c,  
 ITriPortId at,  
 ITriSignatureId signature,  
 ITciValue excValue, ITciValueTemplate excTmpl,  
 ITciValueDifferenceList diffs, ITriComponentId from,  
 ITciNonValueTemplate fromTmpl);  
 void TliPrCatch\_m(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, ITriPortId at,  
 ITriSignatureId signature,  
 ITciValue excValue, ITciValueTemplate excTmpl, ITciValue address,  
 ITciValueTemplate addressTmpl);  
 void TliPrCatch\_c(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, ITriPortId at,  
 ITriSignatureId signature,  
 ITciValue excValue, ITciValueTemplate excTmpl,  
 ITriComponentId from, ITciNonValueTemplate fromTmpl);  
 void TliPrCatchTimeoutDetected(string am, System.DateTime ts, string src,  
 int line, ITriComponentId c,  
 ITriPortId at,  
 ITriSignatureId signature);  
 void TliPrCatchTimeout(string am, System.DateTime ts, string src,  
 int line, ITriComponentId c,  
 ITriPortId at,  
 ITriSignatureId signature);  
 void TliCCreate(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c,  
 ITriComponentId comp, string name, bool alive);  
 void TliCStart(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c,  
 ITriComponentId comp, ITciBehaviourId name,  
 ITciParameterList tciPars);  
 void TliCRunning(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c,  
 ITriComponentId comp, TciComponentStatus status);  
 void TliCAlive(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c,  
 ITriComponentId comp, TciComponentStatus status);  
 void TliCStop(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c,   
 ITriComponentId comp);  
 void TliCKill(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c,  
 ITriComponentId comp);  
 void TliCDoneMismatch(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c,  
 ITriComponentId comp, ITciNonValueTemplate compTmpl);  
 void TliCDone(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, ITciNonValueTemplate compTmpl, ITciVerdictValue verdict);  
 void TliCKilledMismatch(string am, System.DateTime ts, string src,  
 int line, ITriComponentId c,  
 ITriComponentId comp, ITciNonValueTemplate compTmpl);  
 void TliCKilled(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, ITciNonValueTemplate compTmpl, ITciVerdictValue verdict);  
 void TliCTerminated(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, ITciVerdictValue verdict,  
 string reason);  
 void TliPConnect(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, ITriPortId port1,  
 ITriPortId port2);  
 void TliPDisconnect(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, ITriPortId port1,  
 ITriPortId port2);  
 void TliPMap(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, ITriPortId port1,  
 ITriPortId port2);  
 void TliPUnmap(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, ITriPortId port1,  
 ITriPortId port2);  
 void TliPClear(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, ITriPortId port);  
 void TliPStart(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, ITriPortId port);  
 void TliPStop(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, ITriPortId port);  
 void TliPHalt(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, ITriPortId port);  
 void TliEncode(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, ITciValue val,  
 TciStatus encoderFailure, ITriMessage msg,  
 string codec);  
 void TliDecode(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, ITriMessage msg,  
 TciStatus decoderFailure, ITciValue val, string codec);  
 void TliTTimeoutDetected(string am, System.DateTime ts, string src,  
 int line, ITriComponentId c,  
 ITriTimerId timer);  
 void TliTTimeoutMismatch(string am, System.DateTime ts, string src,  
 int line, ITriComponentId c,  
 ITriTimerId timer, ITciNonValueTemplate timerTmpl);  
 void TliTTimeout(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, ITriTimerId timer,  
 ITciNonValueTemplate timerTmpl);  
 void TliTStart(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, ITriTimerId timer,  
 ITriTimerDuration dur);  
 void TliTStop(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, ITriTimerId timer,  
 ITriTimerDuration dur);  
 void TliTRead(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, ITriTimerId timer,  
 ITriTimerDuration elapsed);  
 void TliTRunning(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, ITriTimerId timer,  
 TciTimerStatus status);  
 void TliSEnter(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c,  
 IQualifiedName name, ITciParameterList tciPars,  
 string kind);  
 void TliSLeave(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c,  
 IQualifiedName name, ITciParameterList tciPars,  
 ITciValue returnValue, string kind);  
 void TliVar(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c,  
 IQualifiedName name, ITciValue varValue);  
 void TliModulePar(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c,  
 IQualifiedName name, ITciValue parValue);  
 void TliGetVerdict(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, ITciVerdictValue verdict);  
 void TliSetVerdict(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, ITciVerdictValue verdict,  
 string reason);  
 void TliLog(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, string log);  
 void TliAEnter(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c);  
 void TliALeave(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c);  
 void TliADefaults(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c);  
 void TliAActivate(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c,  
 IQualifiedName name, ITciParameterList tciPars,  
 ITciValue expr);  
 void TliADeactivate(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, ITciValue expr);  
 void TliANomatch(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c);  
 void TliARepeat(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c);  
 void TliAWait(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c);  
 void TliAction(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, string action);  
 void TliMatch(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, ITciValue expr,  
 ITciValueTemplate tmpl);  
 void TliMatchMismatch(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, ITciValue expr,  
 ITciValueTemplate tmpl, ITciValueDifferenceList diffs);  
 void TliInfo (string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, int level, string info);  
 void TliMChecked\_m(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, ITriPortId at,  
 ITciValue msgValue, ITciValueTemplate msgTmpl,

ITciValue address, ITciValueTemplate addressTmpl);  
 void TliMChecked\_c(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, ITriPortId at,  
 ITciValue msgValue, ITciValueTemplate msgTmpl,  
 ITriComponentId fromComp, ITciNonValueTemplate fromTmpl);  
 void TliPrGetCallChecked\_m(string am, System.DateTime ts, string src,  
 int line, ITriComponentId c,  
 ITriPortId at, ITriSignatureId signature,  
 ITciParameterList tciPars, ITciValueTemplate parsTmpl,  
 ITciValue address, ITciValueTemplate addressTmpl);  
 void TliPrGetCallChecked\_c(string am, System.DateTime ts, string src,  
 int line, ITriComponentId c,  
 ITriPortId at, ITriSignatureId signature,  
 ITciParameterList tciPars, ITciValueTemplate parsTmpl,  
 ITriComponentId from, ITciNonValueTemplate fromTmpl);  
 void TliPrGetReplyChecked\_m(string am, System.DateTime ts, string src,  
 int line, ITriComponentId c,  
 ITriPortId at, ITriSignatureId signature,  
 ITciParameterList tciPars, ITciValueTemplate parsTmpl,

ITciValue replValue, ITciValueTemplate replyTmpl,

ITciValue address, ITciValueTemplate addressTmpl);  
 void TliPrGetReplyChecked\_c(string am, System.DateTime ts, string src,  
 int line, ITriComponentId c,  
 ITriPortId at, ITriSignatureId signature,

ITciParameterList tciPars, ITciValueTemplate parsTmpl,   
 ITciValue replValue, ITciValueTemplate replyTmpl,  
 ITriComponentId from, ITciNonValueTemplate fromTmpl);  
 void TliPrCatchChecked\_m(string am, System.DateTime ts, string src,  
 int line, ITriComponentId c,  
 ITriPortId at, ITriSignatureId signature,  
 ITciValue excValue, ITciValueTemplate excTmpl,

ITciValue address, ITciValueTemplate addressTmpl);  
 void TliPrCatchChecked\_c(string am, System.DateTime ts, string src,  
 int line, ITriComponentId c,  
 ITriPortId at, ITriSignatureId signature,

ITciValue excValue, ITciValueTemplate excTmpl,  
 ITriComponentId from, ITciNonValueTemplate fromTmpl);

void TliCheckedAny\_m(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, ITriPortId at,  
 ITciValue address, ITciValueTemplate addressTmpl);  
 void TliCheckedAny\_c(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, ITriPortId at,  
 ITriComponentId from, ITciNonValueTemplate fromTmpl);

void TliCheckAnyMismatch\_m(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, ITriPortId at,

ITciValue address, ITciValueTemplate addressTmpl);  
 void TliCheckAnyMismatch\_c(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, ITriPortId at,

ITriComponentId from, ITciNonValueTemplate fromTmpl);

void TliRnd(string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, ITciFloatValue val, ITciFloatValue seed);

void TliEvaluate (string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, IQualifiedName name, ITciValue evalResult);  
 void TliCCall (string am, System.DateTime ts, string src, int line,  
 ITriComponentId c,  
 ITriComponentId comp, ITciBehaviourId name,  
 ITciParameterList tciPars);  
 void TliCCallTerminated (string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, ITciVerdictValue verdict,  
 string reason, ITciParameterList tciPars, ITciValue returnValue);  
 void TliCtrlStartWithParameters (string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, ITciParameterList tciPars);  
 void TliCtrlTerminatedWithResult (string am, System.DateTime ts, string src, int line,  
 ITriComponentId c, ITciValue result, ITciParameterList tciPars);  
}

## 12.6 Optional parameters

Clause 7.3 in [1] defines that a reserved value shall be used to indicate the absence of an optional parameter. For the C# language mapping the distinct value null shall be used to indicate the absence of an optional value. For example, if in the TciReplyConnected operation the value parameter shall be omitted the operation invocation shall be TciReplyConnected (sender, receiver, signature, parameterList, null).

## 12.7 Error Handling

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

Beside this error handling and exceptions specified for indexing operators no additional error handling is defined in the C# mapping.

Annex A (normative):  
IDL Specification of TCI

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

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// \* Interface definitions for the TTCN‑3 Control Interfaces

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

module tciInterface {

/\* Forward declaration \*/

interface Value;

interface Type;

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// \* Data types taken from the TRI definitions

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Connection

native TriPortIdType ;

native TriPortIdListType;

native TriComponentIdType ;

native TriComponentIdListType;

// Communications

native TriMessageType;

native TriParameterType;

native TriParameterListType;

native TriAddressType;

native TriAddressListType;

native TriExceptionType;

native TriSignatureIdType;

// Miscellaneous

native TriStatusType;

native TriTimerIdType;

native TriTimerDurationType;

native TciStatusType;

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// \* General Abstract Data Types

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Basic definitions

native TBoolean;

native TFloat;

native TChar;

native TInteger;

native TString;

native TUniversalChar;

typedef sequence <TString> TStringSeq;

struct QualifiedName {

TString moduleName;

TString baseName;

};

// General TCI abstract data types

typedef QualifiedName TciBehaviourIdType;

typedef QualifiedName TciModuleIdType;

typedef QualifiedName TciModuleParameterIdType;

typedef QualifiedName TciTestCaseIdType;

enum TciParameterPassingModeType {

IN\_MODE,

OUT\_MODE,

INOUT\_MODE

};

struct TciParameterType {

TciModuleParameterIdType parameterName;

Value parameterValue;

TciParameterPassingModeType mode;

};

typedef sequence <TciParameterType> TciParameterListType;

struct TciParameterTypeType {

Type parameterType;

TciParameterPassingModeType mode;

};

typedef sequence <TciParameterTypeType> TciParameterTypeListType;

struct TciModuleParameterType {

TciModuleParameterIdType parameterName;

Value defaultValue;

};

typedef sequence <TciModuleIdType> TciModuleIdListType ;

typedef sequence <TciModuleParameterType> TciModuleParameterListType;

typedef sequence <TciTestCaseIdType> TciTestCaseIdListType;

enum TciTestComponentKindType {

CONTROL,

MTC,

PTC,

SYSTEM,

PTC\_ALIVE

};

enum ComponentStatusType{

inactiveC,

runningC,

stoppedC,

killedC,

nullC

};

enum TimerStatusType{

runningT,

inactiveT,

expiredT,

nullT

};

enum PortStatusType{

startedP,

haltedP,

stoppedP

};

enum TciTypeClassType {

ADDRESS\_CLASS,

ANYTYPE\_CLASS,

BITSTRING\_CLASS,

BOOLEAN\_CLASS,

CHARSTRING\_CLASS,

COMPONENT\_CLASS,

ENUMERATED\_CLASS,

FLOAT\_CLASS,

HEXSTRING\_CLASS,

INTEGER\_CLASS,

OCTETSTRING\_CLASS,

RECORD\_CLASS,

RECORDOF\_CLASS,

ARRAY\_CLASS,

SET\_CLASS,

SETOF\_CLASS,

UNION\_CLASS,

UNIVERSALCHARSTRING\_CLASS,

VERDICT\_CLASS

DEFAULT\_CLASS,

PORT\_CLASS,

TIMER\_CLASS

};

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// \* Abstract TTCN‑3 Data Types And Values

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Abstract data type "Type"

interface Type {

TciModuleIdType getDefiningModule ();

TString getName ();

TciTypeClassType getTypeClass ();

Value newInstance ();

TString getTypeEncoding ();

TString getTypeEncodingVariant ();

TStringSeq getTypextension ();

Value parseValue(in Tstring val);

};

// Abstract TTCN‑3 Values

interface Value {

TString getValueEncoding ();

TString getValueEncodingVariant ();

Type getType ();

TBoolean notPresent ();

TBoolean isMatchingSymbol ();

TString valueToString ();

};

interface RecordOfValue : Value {

Value getField (in TInteger position);

void setField (

in TInteger position,

in Value value

);

void appendField (in Value value);

Type getElementType ();

TInteger getLength ();

void setLength (in TInteger len);

TInteger getOffset ();

};

interface RecordValue : Value {

Value getField (in TString fieldName);

void setField (

in TString fieldName,

in Value value

);

TStringSeq getFieldNames ();

void setFieldOmitted (in TString fieldName);

};

interface VerdictValue : Value {

TInteger getVerdict ();

void setVerdict (in TInteger verdict);

};

interface BitstringValue : Value {

TString getString ();

void setString (in TString value);

TInteger getBit (in TInteger position);

void setBit (

in TInteger position,

in TInteger value

);

TInteger getLength ();

void setLength (in TInteger len);

};

interface OctetstringValue : Value {

TString getString ();

void setString (in TString value);

TInteger getOctet (in TInteger position);

void setOctet (

in TInteger position,

in TInteger value

);

TInteger getLength ();

void setLength (in TInteger len);

};

interface FloatValue : Value {

TFloat getFloat ();

void setFloat (in TFloat value);

};

interface HexstringValue : Value {

TString getString ();

void setString (in TString value);

TInteger getHex (in TInteger position);

void setHex (

in TInteger position,

in TInteger value

);

TInteger getLength ();

void setLength (in TInteger len);

};

interface EnumeratedValue : Value {

void setEnum (in TString enumValue);

TString getEnum ();

};

interface IntegerValue : Value {

TInteger getInt ();

void setInt (in TInteger value);

};

interface CharValue : Value {

TChar getChar ();

void setChar (in TChar value);

};

interface CharstringValue : Value {

TString getString ();

void setString (in TString value);

TChar getChar (in TInteger position);

void setChar (

in TInteger position,

in TChar value

);

TInteger getLength ();

void setLength (in TInteger len);

};

interface BooleanValue : Value {

TBoolean getBoolean ();

void setBoolean (in TBoolean value);

};

interface UniversalCharValue : Value {

TUniversalChar getUniversalChar ();

void setUniversalChar (in TUniversalChar value);

};

interface UniversalCharstringValue : Value {

TString getString ();

void setString (in TString value);

TUniversalChar getChar (in TInteger position);

void setChar (

in TInteger position,

in TUniversalChar value

);

TInteger getLength ();

void setLength (in TInteger len);

};

interface UnionValue : Value {

Value getVariant (in TString variantName);

void setVariant (

in TString variantName,

in Value value

);

TString getPresentVariantName ();

TStringSeq getVariantNames ();

};

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// \* Abstract Logging Types

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

interface TciValueTemplate : Value {

TBoolean isOmit ();

TBoolean isAny();

TBoolean isAnyOrOmit();

TString getTemplateDef();

};

interface TciNonValueTemplate {

TBoolean isAny();

TBoolean isAll();

TString getTemplateDef();

};

typedef sequence <Value> TciValueListType;

struct TciValueDifferenceType

{

TString desc;

Value val;

TciValueTemplate tmpl;

};

typedef sequence <TciValueDifferenceType> TciValueDifferenceListType;

interface TciValueList {

attribute TciValueListType inst;

TInteger size();

TBoolean isEmpty();

Value get(in TInteger index);

};

interface TciValueDifference {

attribute TciValueDifferenceType inst;

Value getValue();

TciValueTemplate getTciValueTemplate();

TString getDescription();

};

interface TciValueDifferenceList {

attribute TciValueDifferenceListType inst;

TInteger size();

TBoolean isEmpty();

TciValueDifference get(in TInteger index);

};

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Coding Decoding Interface

// ‑ Required

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

interface TCI\_CD\_Required {

Type getTypeForName (in TString typeName);

Type getInteger ();

Type getFloat ();

Type getBoolean ();

Type getChar ();

Type getUniversalChar ();

Type getCharstring ();

Type getUniversalCharstring ();

Type getHexstring ();

Type getBitstring ();

Type getOctetstring ();

Type getVerdict ();

void tciErrorReq (in TString message);

};

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Coding Decoding interface

// ‑ Provided

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

interface TCI\_CD\_Provided {

Value decode (

in TriMessageType message,

in Type decodingHypothesis

);

TriMessageType encode (in Value value);

};

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Test Management Interface

// ‑ Required

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

interface TCI\_TM\_Required : TCI\_CD\_Required {

void tciRootModule (in TciModuleIdType moduleName);

TciModuleIdListType tciGetImportedModules();

TciModuleParameterListType tciGetModuleParameters (in TciModuleIdType moduleName);

TciTestCaseIdListType tciGetTestCases ();

TciParameterTypeListType tciGetTestCaseParameters (

in TciTestCaseIdType testCaseId

);

TriPortIdListType tciGetTestCaseTSI (

in TciTestCaseIdType testCaseId

);

void tciStartTestCase (

in TciTestCaseIdType testCaseId,

in TciParameterListType parameterList

);

void tciStopTestCase ();

TriComponentIdType tciStartControl ();

void tciStopControl ();

TciParameterTypeListType tciGetControlParameters ();

TriComponentIdType tciStartControlWithParameters (in TciParameterListType parameterList);

};

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Test Management Interface

// ‑ Provided

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

interface TCI\_TM\_Provided {

void tciTestCaseStarted (

in TciTestCaseIdType testCaseId,

in TciParameterListType parameterList,

in TFloat timer

);

void tciTestCaseTerminated (

in VerdictValue verdict,

in TciParameterListType parameterList

);

void tciControlTerminated ();

Value tciGetModulePar (

in TciModuleParameterIdType parameterId

);

void tciLog (

in TriComponentIdType testComponentId,

in TString message

);

void tciError (in TString message);

void tciControlTerminatedWithResult (

in Value result,

in TciParameterListType parameterList

);

};

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Component Handling Interface

// ‑ Required

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

interface TCI\_CH\_Required : TCI\_CD\_Required {

void tciEnqueueMsgConnected (

in TriPortIdType sender,

in TriComponentIdType receiver,

in Value receivedMessage

);

void tciEnqueueCallConnected (

in TriPortIdType sender,

in TriComponentIdType receiver,

in TriSignatureIdType signature,

in TciParameterListType parameterList

);

void tciEnqueueReplyConnected (

in TriPortIdType sender,

in TriComponentIdType receiver,

in TriSignatureIdType signature,

in TciParameterListType parameterList,

in Value returnValue

);

void tciEnqueueRaiseConnected (

in TriPortIdType sender,

in TriComponentIdType receiver,

in TriSignatureIdType signature,

in Value except

);

TriComponentIdType tciCreateTestComponent (

in TciTestComponentKindType kind,

in Type componentType,

in TString name

);

void tciStartTestComponent (

in TriComponentIdType comp,

in TciBehaviourIdType behavior,

in TciParameterListType parameterList

);

void tciStopTestComponent (

in TriComponentIdType comp

);

void tciConnect (

in TriPortIdType fromPort,

in TriPortIdType toPort

);

void tciDisconnect (

in TriPortIdType fromPort,

in TriPortIdType toPort

);

void tciTestComponentTerminated (

in TriComponentIdType comp,

in VerdictValue verdict

);

TBoolean tciTestComponentRunning (

in TriComponentIdType comp

);

TriComponentIdType tciGetMTC ();

void tciMap (

in TriPortIdType fromPort,

in TriPortIdType toPort

);

void tciUnmap (

in TriPortIdType fromPort,

in TriPortIdType toPort

);

void tciExecuteTestCase (

in TciTestCaseIdType testCaseId,

in TriPortIdListType tsiPortList

);

TBoolean tciTestComponentDone (

in TriComponentIdType comp

);

void tciReset ();

};

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Component Handling Interface

// ‑ Provided

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

interface TCI\_CH\_Provided {

void tciSendConnected (

in TriPortIdType sender,

in TriComponentIdType receiver,

in Value sendMessage

);

void tciSendConnectedBC (

in TriPortIdType sender,

in Value sendMessage

);

void tciSendConnectedMC (

in TriPortIdType sender,

in TriComponentIdListType receivers,

in Value sendMessage

);

void tciCallConnected (

in TriPortIdType sender,

in TriComponentIdType receiver,

in TriSignatureIdType signature,

in TciParameterListType parameterList

);

void tciCallConnectedBC (

in TriPortIdType sender,

in TriSignatureIdType signature,

in TciParameterListType parameterList

);

void tciCallConnectedMC (

in TriPortIdType sender,

in TriComponentIdListType receivers,

in TriSignatureIdType signature,

in TciParameterListType parameterList

);

void tciReplyConnected (

in TriPortIdType sender,

in TriComponentIdType receiver,

in TriSignatureIdType signature,

in TciParameterListType parameterList,

in Value returnValue

);

void tciReplyConnectedBC (

in TriPortIdType sender,

in TriSignatureIdType signature,

in TciParameterListType parameterList,

in Value returnValue

);

void tciReplyConnectedMC (

in TriPortIdType sender,

in TriComponentIdListType receivers,

in TriSignatureIdType signature,

in TciParameterListType parameterList,

in Value returnValue

);

void tciRaiseConnected (

in TriPortIdType sender,

in TriComponentIdType receiver,

in TriSignatureIdType signature,

in Value except

);

void tciRaiseConnectedBC (

in TriPortIdType sender,

in TriSignatureIdType signature,

in Value except

);

void tciRaiseConnectedMC (

in TriPortIdType sender,

in TriComponentIdListType receivers,

in TriSignatureIdType signature,

in Value except

);

TriComponentIdType tciCreateTestComponentReq (

in TciTestComponentKindType kind,

in Type componentType,

in TString name,

in Value hostId

);

void tciStartTestComponentReq (

in TriComponentIdType comp,

in TciBehaviourIdType behavior,

in TciParameterListType parameterList

);

void tciStopTestComponentReq (

in TriComponentIdType comp

);

void tciConnectReq (

in TriPortIdType fromPort,

in TriPortIdType toPort

);

void tciDisconnectReq (

in TriPortIdType fromPort,

in TriPortIdType toPort

);

void tciTestComponentTerminatedReq (

in TriComponentIdType comp,

in VerdictValue verdict

);

TBoolean tciTestComponentRunningReq (

in TriComponentIdType comp

);

TriComponentIdType tciGetMTCReq ();

void tciMapReq (

in TriPortIdType fromPort,

in TriPortIdType toPort

);

void tciUnmapReq (

in TriPortIdType fromPort,

in TriPortIdType toPort

);

void tciExecuteTestCaseReq (

in TciTestCaseIdType testCaseId,

in TriPortIdListType tsiPortList

);

void tciResetReq ();

TBoolean tciTestComponentDoneReq (

in TriComponentIdType comp

);

};

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Test Logging Interface

// ‑ Provided

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

interface TCI\_TL\_Provided {

void tliTcExecute(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TciTestCaseIdType tcId,

in TciParameterListType tciPars, in TriTimerDurationType dur

);

void tliTcStart(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TciTestCaseIdType tcId,

in TciParameterListType tciPars, in TriTimerDurationType dur

);

void tliTcStop(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TString reason

);

void tliTcStarted(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TciTestCaseIdType tcId,

in TciParameterListType tciPars, in TriTimerDurationType dur

);

void tliTcTerminated(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TciTestCaseIdType tcId,

in TciParameterListType tciPars, in VerdictValue verdict, in TString reason);

void tliCtrlStart(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c

);

void tliCtrlStop(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c

);

void tliCtrlTerminated(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c);

void tliMSend\_m(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TriPortIdType at, in TriPortIdType to, in Value msgValue,

in Value addrValue, in TciStatusType encoderFailure,

in TriMessageType msg, in TriAddressType address, in TriStatusType transmissionFailure

);

void tliMSend\_m\_BC(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TriPortIdType at, in TriPortIdType to, in Value msgValue,

in TciStatusType encoderFailure, in TriMessageType msg,

in TriStatusType transmissionFailure

);

void tliMSend\_m\_MC(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TriPortIdType at, in TriPortIdType to, in Value msgValue,

in TciValueList addrValues, in TciStatusType encoderFailure,

in TriMessageType msg, in TriAddressListType addresses,

in TriStatusType transmissionFailure

);

void tliMSend\_c(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TriPortIdType at, in TriPortIdType to, in Value msgValue,

in TriStatusType transmissionFailure

);

void tliMSend\_c\_BC(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TriPortIdType at, in TriPortIdListType to, in Value msgValue,

in TriStatusType transmissionFailure

);

void tliMSend\_c\_MC(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TriPortIdType at, in TriPortIdListType to, in Value msgValue,

in TriStatusType transmissionFailure);

void tliMDetected\_m(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TriPortIdType at, in TriPortIdType from,

in TriMessageType msg,

in TriAddressType address

);

void tliMDetected\_c(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TriPortIdType at, in TriPortIdType from, in Value msgValue

);

void tliMMismatch\_m(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TriPortIdType at, in Value msgValue,

in TciValueTemplate msgTmpl, in TciValueDifferenceList diffs,

in Value addrValue, in TciValueTemplate addressTmpl

);

void tliMMismatch\_c(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TriPortIdType at, in Value msgValue,

in TciValueTemplate msgTmpl, in TciValueDifferenceList diffs,

in TriComponentIdType from, in TciNonValueTemplate fromTmpl

);

void tliMReceive\_m(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TriPortIdType at, in Value msgValue,

in TciValueTemplate msgTmpl, in Value addrValue,

in TciValueTemplate addressTmpl

);

void tliMReceive\_c(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TriPortIdType at, in Value msgValue,

in TciValueTemplate msgTmpl, in TriComponentIdType from,

in TciNonValueTemplate fromTmpl

);

void tliPrCall\_m(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TriPortIdType at, in TriPortIdType to,

in TriSignatureIdType signature, in TciParameterListType tciPars,

in Value addrValue, in TciStatusType encoderFailure,

in TriParameterListType triPars, in TriAddressType address,

in TriStatusType transmissionFailure

);

void tliPrCall\_m\_BC(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TriPortIdType at, in TriPortIdType to,

in TriSignatureIdType signature, in TciParameterListType tciPars,

in TciStatusType encoderFailure, in TriParameterListType triPars,

in TriStatusType transmissionFailure

);

void tliPrCall\_m\_MC(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TriPortIdType at, in TriPortIdType to,

in TriSignatureIdType signature, in TciParameterListType tciPars,

in TciValueList addrValues, in TciStatusType encoderFailure,

in TriParameterListType triPars, in TriAddressListType addresses,

in TriStatusType transmissionFailure

);

void tliPrCall\_c(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TriPortIdType at, in TriPortIdType to,

in TriSignatureIdType signature, in TciParameterListType tciPars,

in TriStatusType transmissionFailure

);

void tliPrCall\_c\_BC(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TriPortIdType at, in TriPortIdListType to,

in TriSignatureIdType signature, in TciParameterListType tciPars,

in TriStatusType transmissionFailure

);

void tliPrCall\_c\_MC(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TriPortIdType at, in TriPortIdListType to,

in TriSignatureIdType signature, in TciParameterListType tciPars,

in TriStatusType transmissionFailure

);

void tliPrGetCallDetected\_m(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TriPortIdType at, in TriPortIdType from,

in TriSignatureIdType signature, in TriParameterListType triPars,

in TriAddressType address

);

void tliPrGetCallDetected\_c(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TriPortIdType at, in TriPortIdType from,

in TriSignatureIdType signature, in TciParameterListType tciPars

);

void tliPrGetCallMismatch\_m(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TriPortIdType at,

in TriSignatureIdType signature, in TciParameterListType tciPars,

in TciValueTemplate parsTmpl, in TciValueDifferenceList diffs,

in Value addrValue, in TciValueTemplate addressTmpl

);

void tliPrGetCallMismatch\_c(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TriPortIdType at,

in TriSignatureIdType signature, in TciParameterListType tciPars,

in TciValueTemplate parsTmpl, in TciValueDifferenceList diffs,

in TriComponentIdType from, in TciNonValueTemplate fromTmpl

);

void tliPrGetCall\_m(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TriPortIdType at,

in TriSignatureIdType signature, in TciParameterListType tciPars,

in TciValueTemplate parsTmpl, in Value addrValue,

in TciValueTemplate addressTmpl

);

void tliPrGetCall\_c(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TriPortIdType at,

in TriSignatureIdType signature, in TciParameterListType tciPars,

in TciValueTemplate parsTmpl, in TriComponentIdType from,

in TciNonValueTemplate fromTmpl

);

void tliPrReply\_m(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TriPortIdType at, in TriPortIdType to,

in TriSignatureIdType signature, in TciParameterListType tciPars,

in Value replValue, in Value addrValue,

in TciStatusType encoderFailure, in TriParameterListType triPars,

in TriParameterType repl, in TriAddressType address, in TriStatusType transmissionFailure

);

void tliPrReply\_m\_BC(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TriPortIdType at, in TriPortIdType to,

in TriSignatureIdType signature, in TciParameterListType tciPars, in Value replValue,

in TciStatusType encoderFailure, in TriParameterListType triPars,

in TriParameterType repl, in TriStatusType transmissionFailure

);

void tliPrReply\_m\_MC(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TriPortIdType at, in TriPortIdType to,

in TriSignatureIdType signature, in TciParameterListType tciPars, in Value replValue,

in TciValueListType addrValues, in TciStatusType encoderFailure,

in TriParameterListType triPars, in TriParameterType repl,

in TriAddressListType addresses, in TriStatusType transmissionFailure

);

void tliPrReply\_c(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TriPortIdType at, in TriPortIdType to,

in TriSignatureIdType signature, in TciParameterListType tciPars,

in Value replValue, in TriStatusType transmissionFailure

);

void tliPrReply\_c\_BC(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TriPortIdType at, in TriPortIdListType to,

in TriSignatureIdType signature, in Value parsValue, in Value replValue,

in TriStatusType transmissionFailure

);

void tliPrReply\_c\_MC(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TriPortIdType at, in TriPortIdListType to,

in TriSignatureIdType signature, in Value parsValue, in Value replValue,

in TriStatusType transmissionFailure

);

void tliPrGetReplyDetected\_m(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TriPortIdType at, in TriPortIdType from,

in TriSignatureIdType signature, in TriParameterListType triPars,

in TriParameterType repl, in TriAddressType address

);

void tliPrGetReplyDetected\_c(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TriPortIdType at, in TriPortIdType from,

in TriSignatureIdType signature, in TciParameterListType tciPars,

in Value replValue

);

void tliPrGetReplyMismatch\_m(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TriPortIdType at,

in TriSignatureIdType signature,

in TciParameterListType tciPars, in TciValueTemplate parsTmpl,

in Value replValue, in TciValueTemplate replyTmpl,

in TciValueDifferenceList diffs, in Value addrValue,

in TciValueTemplate addressTmpl

);

void tliPrGetReplyMismatch\_c(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TriPortIdType at,

in TriSignatureIdType signature,

in TciParameterListType tciPars, in TciValueTemplate parsTmpl,

in Value replValue, in TciValueTemplate replyTmpl,

in TciValueDifferenceList diffs, in TriComponentIdType from,

in TciNonValueTemplate fromTmpl

);

void tliPrGetReply\_m(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TriPortIdType at,

in TriSignatureIdType signature,

in TciParameterListType tciPars, in TciValueTemplate parsTmpl,

in Value replValue, in TciValueTemplate replyTmpl,

in Value addrValue, in TciValueTemplate addressTmpl

);

void tliPrGetReply\_c(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TriPortIdType at,

in TriSignatureIdType signature,

in TciParameterListType tciPars, in TciValueTemplate parsTmpl,

in Value replValue, in TciValueTemplate replyTmpl,

in TriComponentIdType from, in TciNonValueTemplate fromTmpl

);

void tliPrRaise\_m(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TriPortIdType at, in TriPortIdType to,

in TriSignatureIdType signature, in TciParameterListType tciPars,

in Value excValue, in Value addrValue, in TciStatusType encoderFailure,

in TriExceptionType exc, in TriAddressType address, in TriStatusType transmissionFailure

);

void tliPrRaise\_m\_BC(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TriPortIdType at, in TriPortIdType to,

in TriSignatureIdType signature, in TciParameterListType tciPars,

in Value excValue, in TciStatusType encoderFailure, in TriExceptionType exc,

in TriStatusType transmissionFailure

);

void tliPrRaise\_m\_MC(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TriPortIdType at, in TriPortIdType to,

in TriSignatureIdType signature, in TciParameterListType tciPars,

in Value excValue, in TciValueListType addrValues,

in TciStatusType encoderFailure, in TriExceptionType exc,

in TriAddressListType addresses, in TriStatusType transmissionFailure

);

void tliPrRaise\_c(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TriPortIdType at, in TriPortIdType to,

in TriSignatureIdType signature, in TciParameterListType tciPars,

in Value excValue, in TriStatusType transmissionFailure

);

void tliPrRaise\_c\_BC(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TriPortIdType at, in TriPortIdListType to,

in TriSignatureIdType signature, in TciParameterListType tciPars,

in Value excValue, in TriStatusType transmissionFailure

);

void tliPrRaise\_c\_MC(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TriPortIdType at, in TriPortIdListType to,

in TriSignatureIdType signature, in TciParameterListType tciPars,

in Value excValue, in TriStatusType transmissionFailure

);

void tliPrCatchDetected\_m(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TriPortIdType at, in TriPortIdType from,

in TriSignatureIdType signature,

in TriExceptionType exc, in TriAddressType address

);

void tliPrCatchDetected\_c(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TriPortIdType at, in TriPortIdType from,

in TriSignatureIdType signature, in Value excValue

);

void tliPrCatchMismatch\_m(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TriPortIdType at,

in TriSignatureIdType signature,

in Value excValue, in TciValueTemplate excTmpl,

in TciValueDifferenceList diffs, in Value addrValue,

in TciValueTemplate addressTmpl

);

void tliPrCatchMismatch\_c(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TriPortIdType at,

in TriSignatureIdType signature,

in Value excValue, in TciValueTemplate excTmpl,

in TciValueDifferenceList diffs, in TriComponentIdType from,

in TciNonValueTemplate fromTmpl

);

void tliPrCatch\_m(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TriPortIdType at,

in TriSignatureIdType signature,

in Value excValue, in TciValueTemplate excTmpl,

in Value addrValue, in TciValueTemplate addressTmpl

);

void tliPrCatch\_c(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TriPortIdType at,

in TriSignatureIdType signature,

in Value excValue, in TciValueTemplate excTmpl,

in TriComponentIdType from, in TciNonValueTemplate fromTmpl

);

void tliPrCatchTimeoutDetected(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TriPortIdType at,

in TriSignatureIdType signature

);

void tliPrCatchTimeout(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TriPortIdType at,

in TriSignatureIdType signature

);

void tliCCreate(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TriComponentIdType comp,

in TString name, in TBoolean alive

);

void tliCStart(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TriComponentIdType comp,

in TciBehaviourIdType name, in TciParameterListType tciPars

);

void tliCRunning(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TriComponentIdType comp, in ComponentStatusType status

);

void tliCAlive(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c,

in TriComponentIdType comp, in ComponentStatusType status

);

void tliCStop(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TriComponentIdType comp

);

void tliCKill(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TriComponentIdType comp

);

void tliCDoneMismatch(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TriComponentIdType comp, in TciNonValueTemplate compTmpl

);

void tliCKilledMismatch(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TriComponentIdType comp, in TciNonValueTemplate compTmpl

);

void tliCDone(in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TciNonValueTemplate compTmpl

);

void tliCKilled(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TciNonValueTemplate compTmpl

);

void tliCTerminated(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in VerdictValue verdict, in TString reason

);

void tliPConnect(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TriPortIdType port1, in TriPortIdType port2

);

void tliPDisconnect(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TriPortIdType port1,

in TriPortIdType port2

);

void tliPMap(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TriPortIdType port1, in TriPortIdType port2

);

void tliPUnmap(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TriPortIdType port1,

in TriPortIdType port2

);

void tliPClear(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TriPortIdType port

);

void tliPStart(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TriPortIdType port

);

void tliPStop(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TriPortIdType port

);

void tliPHalt(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TriPortIdType port

);

void tliEncode(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in Value val, in TciStatusType encoderFailure,

in TriMessageType msg, in TString codec

);

void tliDecode(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TriMessageType msg,

in TciStatusType decoderFailure, in Value val, in TString codec

);

void tliTTimeoutDetected(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TriTimerIdType timer

);

void tliTTimeoutMismatch(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TriTimerIdType timer, in TciNonValueTemplate timerTmpl

);

void tliTTimeout(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TriTimerIdType timer, in TciNonValueTemplate timerTmpl

);

void tliTStart(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TriTimerIdType timer,

in TriTimerDurationType dur

);

void tliTStop(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TriTimerIdType timer, in TriTimerDurationType dur

);

void tliTRead(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TriTimerIdType timer,

in TriTimerDurationType elapsed

);

void tliTRunning(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TriTimerIdType timer, in TimerStatusType status

);

void tliSEnter(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in QualifiedName name, in TciParameterListType tciPars,

in TString kind

);

void tliSLeave(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in QualifiedName name, in TciParameterListType tciPars,

in Value returnValue, in TString kind

);

void tliVar(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in QualifiedName name, in Value varValue

);

void tliModulePar(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in QualifiedName name, in Value parValue

);

void tliGetVerdict(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in VerdictValue verdict

);

void tliSetVerdict(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in VerdictValue verdict, in TString reason

);

void tliLog(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TString log

);

void tliAEnter(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c

);

void tliALeave(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c

);

void tliADefaults(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c

);

void tliAActivate(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in QualifiedName name, in TciParameterListType tciPars,

in Value ref

);

void tliADeactivate(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in Value ref

);

void tliANomatch(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c

);

void tliARepeat(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c

);

void tliAWait(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c

);

void tliAction(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TString action

);

void tliMatch(

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in Value expr, in TciValueTemplate tmpl

);

void tliMatchMismatch(

in TString am, in TInteger ts, in TString src,

in TInteger line, in TriComponentIdType c, in Value expr,

in TciValueTemplate tmpl, in TciValueDifferenceList diffs

);

void tliInfo(

in TString am, in TInteger ts, in TString src,

in TInteger line, in TriComponentIdType c,

in TInteger level, in TString info

);

void tliMChecked\_m(

in TString am, in TInteger ts,

in TString src, in TInteger line, in TriComponentIdType c,

in TriPortIdType at,

in Value msgValue, in TciValueTemplate msgTmpl,

in Value addrValue, in TciValueTemplate addressTmpl

);

void tliMChecked\_c(

in TString am, in TInteger ts,

in TString src, in TInteger line, in TriComponentIdType c,

in TriPortIdType at,

in Value msgValue in TciValueTemplate msgTmpl,

in TriComponentIdType from, in TciNonValueTemplate fromTmpl

);

void tliPrGetCallChecked\_m(

in TString am, in TInteger ts,

in TString src,

in TInteger line, in TriComponentIdType c,

in TriPortIdType at,

in TriSignatureIdType signature,

in TciParameterListType tciPars, in TciValueTemplate parsTmpl,

in Value addrValue, in TciValueTemplate addressTmpl

);

void tliPrGetCallChecked\_c(

in TString am, in TInteger ts,

in TString src,

in TInteger line, in TriComponentIdType c,

in TriPortIdType at,

in TriSignatureIdType signature,

in TciParameterListType tciPars, in TciValueTemplate parsTmpl,

in TriComponentIdType from, in TciNonValueTemplate fromTmpl

);

void tliPrGetReplyChecked\_m(

in TString am, in TInteger ts,

in TString src,

in TInteger line, in TriComponentIdType c,

in TriPortIdType at,

in TriSignatureIdType signature,

in TciParameterListType tciPars, in TciValueTemplate parsTmpl,

in Value replValue, in TciValueTemplate replyTmpl,

in Value addrValue, in TciValueTemplate addressTmpl

);

void tliPrGetReplyChecked\_c(

in TString am, in TInteger ts,

in TString src,

in TInteger line, in TriComponentIdType c,

in TriPortIdType at,

in TriSignatureIdType signature,

in TciParameterListType tciPars, in TciValueTemplate parsTmpl,

in Value replValue, in TciValueTemplate replyTmpl,

in TriComponentIdType from, in TciNonValueTemplate fromTmpl

);

void tliPrCatchChecked\_m(

in TString am, in TInteger ts,

in TString src,

in TInteger line, in TriComponentIdType c,

in TriPortIdType at,

in TriSignatureIdType signature,   
 in Value excValue, in TciValueTemplate excTmpl,

in Value addrValue, in TciValueTemplate addressTmpl

);

void tliPrCatchChecked\_c(

in TString am, in TInteger ts,

in TString src,

in TInteger line, in TriComponentIdType c,

in TriPortIdType at,

in TriSignatureIdType signature,

in Value excValue, in TciValueTemplate excTmpl,

in TriComponentIdType from, in TciNonValueTemplate fromTmpl

);

void tliCheckedAny\_m(

in TString am, in TInteger ts,

in TString src, in TInteger line, in TriComponentIdType c,

in TriPortIdType at,

in Value addrValue,

in TciValueTemplate addressTmpl

);

void tliCheckedAny\_c(

in TString am, in TInteger ts,

in TString src, in TInteger line, in TriComponentIdType c,

in TriPortIdType at,

in TriComponentIdType from,

in TciNonValueTemplate fromTmpl

);

void tliCheckAnyMismatch\_m(

in TString am, in TInteger ts,

in TString src, in TInteger line, in TriComponentIdType c,

in TriPortIdType at,

in Value addrValue,

in TciValueTemplate addressTmpl

);

void tliCheckAnyMismatch\_c(

in TString am, in TInteger ts,

in TString src, in TInteger line, in TriComponentIdType c,

in TriPortIdType at,

in TriComponentIdType from,

in TciNonValueTemplate fromTmpl

);

void tliRnd(

in TString am, in TInteger ts,

in TString src, in TInteger line, in TriComponentIdType c,

in FloatValue val, in FloatValue seed

);

};

void tliEvaluate(

in TString am, in TInteger ts,

in TString src, in TInteger line, in TriComponentIdType c,

in QualifiedName name, in Value evalResult

);

void tliCCall(

in TString am, in TInteger ts,

in TString src, in TInteger line, in TriComponentIdType c,

in TciBehaviourIdType name, in TciParameterListType tciPars

);

void tliCCallTerminated(

in TString am, in TInteger ts,

in TString src, in TInteger line, in TriComponentIdType c,

in VerdictValue verdict, in TString reason,  
 in TciParameterListType tciPars, in Value returnValue

);

void tliCtrlStartWithParameters (

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in TciParameterListType tciPars

);

void tliCtrlTerminatedWithResult (

in TString am, in TInteger ts, in TString src, in TInteger line,

in TriComponentIdType c, in Value result, in TciParameterListType tciPars);

};

Annex B (normative):  
XML Mapping for TCI TL Provided

# B.0 Introduction

This annex defines a mapping for the logging interface of TCI using eXtended Markup Language (XML) schema definitions.

# B.1 TCI‑TL XML Schema for Simple Types

<?xml version="1.0" encoding="UTF‑8"?>

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"

targetNamespace="http://uri.etsi.org/ttcn‑3/tci/SimpleTypes\_v4\_10\_1.xsd"

xmlns:SimpleTypes="http://uri.etsi.org/ttcn‑3/tci/SimpleTypes\_v4\_10\_1.xsd"

elementFormDefault="qualified">

<!‑‑ Basic definitions ‑‑>

<xsd:simpleType name="xpath">

<!‑‑ this string should be XPATH compliant ‑‑>

<xsd:restriction base="xsd:string"/>

</xsd:simpleType>

<xsd:simpleType name="TBoolean">

<xsd:restriction base="xsd:boolean"/>

</xsd:simpleType>

<xsd:simpleType name="TString">

<xsd:restriction base="xsd:string"/>

</xsd:simpleType>

<xsd:simpleType name="TInteger">

<xsd:restriction base="xsd:integer"/>

</xsd:simpleType>

<!‑‑ Miscellaneous ‑‑>

<xsd:simpleType name="TriTimerDurationType">

<xsd:restriction base="xsd:float"/>

</xsd:simpleType>

<xsd:simpleType name="TciParameterPassingModeType">

<xsd:restriction base="xsd:string">

<xsd:enumeration value="in"/>

<xsd:enumeration value="inout"/>

<xsd:enumeration value="out"/>

</xsd:restriction>

</xsd:simpleType>

<xsd:simpleType name="TriStatusType">

<xsd:restriction base="xsd:string">

<xsd:enumeration value="TRI\_Ok"/>

<xsd:enumeration value="TRI\_Error"/>

</xsd:restriction>

</xsd:simpleType>

<xsd:simpleType name="TciStatusType">

<xsd:restriction base="xsd:string">

<xsd:enumeration value="TCI\_Ok"/>

<xsd:enumeration value="TCI\_Error"/>

</xsd:restriction>

</xsd:simpleType>

<xsd:simpleType name="ComponentStatusType">

<xsd:restriction base="xsd:string">

<xsd:enumeration value="inactiveC"/>

<xsd:enumeration value="runningC"/>

<xsd:enumeration value="stoppedC"/>

<xsd:enumeration value="killedC"/>

<xsd:enumeration value="nullC"/>

</xsd:restriction>

</xsd:simpleType>

<xsd:simpleType name="TimerStatusType">

<xsd:restriction base="xsd:string">

<xsd:enumeration value="runningT"/>

<xsd:enumeration value="inactiveT"/>

<xsd:enumeration value="expiredT"/>

<xsd:enumeration value="nullT"/>

</xsd:restriction>

</xsd:simpleType>

<xsd:simpleType name="PortStatusType">

<xsd:restriction base="xsd:string">

<xsd:enumeration value="startedP"/>

<xsd:enumeration value="haltedP"/>

<xsd:enumeration value="stoppedP"/>

</xsd:restriction>

</xsd:simpleType>

<xsd:complexType name="TEmpty" />

</xsd:schema>

# B.2 TCI‑TL XML Schema for Types

<?xml version="1.0" encoding="UTF‑8"?>

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"

targetNamespace="http://uri.etsi.org/ttcn‑3/tci/Types\_v4\_10\_1.xsd"

xmlns:Types="http://uri.etsi.org/ttcn‑3/tci/Types\_v4\_10\_1.xsd"

xmlns:SimpleTypes="http://uri.etsi.org/ttcn‑3/tci/SimpleTypes\_v4\_10\_1.xsd"

xmlns:Values="http://uri.etsi.org/ttcn‑3/tci/Values\_v4\_10\_1.xsd"

xmlns:Templates="http://uri.etsi.org/ttcn‑3/tci/Templates\_v4\_10\_1.xsd"

elementFormDefault="qualified">

<xsd:import namespace="http://uri.etsi.org/ttcn‑3/tci/Values\_v4\_10\_1.xsd"

schemaLocation="Values\_v4\_10\_1.xsd"/>

<xsd:import namespace="http://uri.etsi.org/ttcn‑3/tci/SimpleTypes\_v4\_10\_1.xsd"

schemaLocation="SimpleTypes\_v4\_10\_1.xsd"/>

<xsd:import namespace="http://uri.etsi.org/ttcn‑3/tci/Templates\_v4\_10\_1.xsd"

schemaLocation="Templates\_v4\_10\_1.xsd"/>

<!‑‑ Connection ‑‑>

<xsd:complexType name="TriPortIdType">

<xsd:sequence>

<xsd:element name="comp" type="Types:TriComponentIdType" />

<xsd:element name="port" type="Types:Port" />

</xsd:sequence>

</xsd:complexType>

<xsd:complexType name="TriPortIdListType">

<xsd:sequence>

<xsd:element name="port" type="Types:TriPortIdType" minOccurs="0"

maxOccurs="unbounded"/>

</xsd:sequence>

</xsd:complexType>

<xsd:complexType name="Port">

<xsd:sequence>

<xsd:element name="id" type="Types:Id" />

<xsd:element name="index" type="xsd:int" minOccurs="0" />

</xsd:sequence>

</xsd:complexType>

<xsd:complexType name="TriComponentIdType">

<xsd:sequence>

<xsd:choice>

<xsd:element name="null" type="Templates:null"/>

<xsd:element name="id" type="Types:Id" />

</xsd:choice>

</xsd:sequence>

</xsd:complexType>

<xsd:complexType name="TriComponentIdListType">

<xsd:sequence>

<xsd:element name="comp" type="Types:TriComponentIdType" minOccurs="0"

maxOccurs="unbounded"/>

</xsd:sequence>

</xsd:complexType>

<!‑‑ Communication ‑‑>

<xsd:complexType name="TriMessageType">

<xsd:attribute name="val" type="xsd:hexBinary"/>

<xsd:attribute name="paddingBits" type="xsd:integer" use="optional" default="0"/>

</xsd:complexType>

<xsd:complexType name="TriParameterType">

<xsd:attribute name="val" type="xsd:hexBinary" />

<xsd:attribute name="paddingBits" type="xsd:integer" use="optional" default="0"/>

<xsd:attribute name="name" type="SimpleTypes:TString"/>

<xsd:attribute name="mode" type="SimpleTypes:TciParameterPassingModeType"/>

</xsd:complexType>

<xsd:complexType name="TriParameterListType">

<xsd:sequence>

<xsd:element name="par" type="Types:TriParameterType" minOccurs="0" maxOccurs="unbounded"/>

</xsd:sequence>

</xsd:complexType>

<xsd:complexType name="TriExceptionType">

<xsd:attribute name="val" type="xsd:hexBinary"/>

</xsd:complexType>

<xsd:complexType name="TciValueListType">

<xsd:complexContent>

<xsd:extension base="Values:RecordValue"/>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="TriSignatureIdType">

<xsd:attribute name="val" type="SimpleTypes:TString" use="required"/>

</xsd:complexType>

<xsd:complexType name="TriAddressType">

<xsd:attribute name="val" type="xsd:hexBinary"/>

<xsd:attribute name="paddingBits" type="xsd:integer" use="optional" default="0"/>

</xsd:complexType>

<xsd:complexType name="TriAddressListType">

<xsd:sequence>

<xsd:element name="addr" type="Types:TriAddressType" minOccurs="0"

maxOccurs="unbounded"/>

</xsd:sequence>

</xsd:complexType>

<!‑‑ Miscellaneous ‑‑>

<xsd:complexType name="TriTimerIdType">

<xsd:sequence>

<xsd:element name="id" type="Types:Id" />

</xsd:sequence>

</xsd:complexType>

<!‑‑ Basic definitions ‑‑>

<xsd:complexType name="QualifiedName">

<xsd:attribute name="moduleName" type="SimpleTypes:TString" use="required"/>

<xsd:attribute name="baseName" type="SimpleTypes:TString" use="required"/>

</xsd:complexType>

<!‑‑ general TCI abstract data types ‑‑>

<xsd:complexType name="TciBehaviourIdType">

<xsd:sequence>

<xsd:element name="name" type="Types:QualifiedName" />

</xsd:sequence>

</xsd:complexType>

<xsd:complexType name="TciTestCaseIdType">

<xsd:sequence>

<xsd:element name="name" type="Types:QualifiedName" />

</xsd:sequence>

</xsd:complexType>

<xsd:complexType name="TciParameterType">

<xsd:sequence>

<xsd:element name="val" type="Values:Value" />

</xsd:sequence>

<xsd:attribute name="name" type="SimpleTypes:TString"/>

<xsd:attribute name="mode" type="SimpleTypes:TciParameterPassingModeType"/>

</xsd:complexType>

<xsd:complexType name="TciParameterListType">

<xsd:sequence>

<xsd:element name="par" type="Types:TciParameterType" minOccurs="0" maxOccurs="unbounded"/>

</xsd:sequence>

</xsd:complexType>

<!‑‑ general identifier structure for test components, ports and timer ‑‑>

<xsd:complexType name="Id">

<xsd:sequence>

<xsd:element name="name" type="SimpleTypes:TString" />

<xsd:element name="id" type="SimpleTypes:TString" minOccurs="0"/>

<xsd:element name="type" type="SimpleTypes:TString" minOccurs="0"/>

</xsd:sequence>

</xsd:complexType>

</xsd:schema>

# B.3 TCI‑TL XML Schema for Values

<?xml version="1.0" encoding="UTF‑8"?>

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"

targetNamespace="http://uri.etsi.org/ttcn‑3/tci/Values\_v4\_10\_1.xsd"

xmlns:Values="http://uri.etsi.org/ttcn‑3/tci/Values\_v4\_10\_1.xsd"

xmlns:Templates="http://uri.etsi.org/ttcn‑3/tci/Templates\_v4\_10\_1.xsd"

xmlns:SimpleTypes="http://uri.etsi.org/ttcn‑3/tci/SimpleTypes\_v4\_10\_1.xsd" elementFormDefault="qualified">

<xsd:import namespace="http://uri.etsi.org/ttcn‑3/tci/Templates\_v4\_10\_1.xsd"

schemaLocation="Templates\_v4\_10\_1.xsd"/>

<xsd:import namespace="http://uri.etsi.org/ttcn‑3/tci/SimpleTypes\_v4\_10\_1.xsd"

schemaLocation="SimpleTypes\_v4\_10\_1.xsd"/>

<xsd:simpleType name="ValueModifier">

<xsd:restriction base="SimpleTypes:TString">  
      <xsd:enumeration value="lazy"/>  
      <xsd:enumeration value="fuzzy"/>

    </xsd:restriction>

</xsd:simpleType>

<xsd:attributeGroup name="ValueAtts">

<xsd:attribute name="name" type="SimpleTypes:TString" use="optional"/>

<xsd:attribute name="type" type="SimpleTypes:TString" use="optional"/>

<xsd:attribute name="module" type="SimpleTypes:TString" use="optional"/>

<xsd:attribute name="modifier" type="Values:ValueModifier" use="optional"/>

<xsd:attribute name="annotation" type="SimpleTypes:TString" use="optional"/> </xsd:attributeGroup>

<xsd:group name="BaseValue">

<xsd:choice>

<xsd:sequence>  
 <xsd:choice>  
     <xsd:element name="value" type="SimpleTypes:TString"/>  
     <xsd:element name="matching\_symbol" type="Templates:MatchingSymbol"/>  
 </xsd:choice>  
 <xsd:element name="ifpresent" type="SimpleTypes:TEmpty" minOccurs="0"/>  
 <xsd:element name="length" type="Values:LengthRestriction" minOccurs="0"/>  
 </xsd:sequence>

<xsd:element name="null" type="SimpleTypes:TEmpty"/>

<xsd:element name="omit" type=" SimpleTypes:TEmpty"/>  
 <xsd:element name="not\_evaluated" type=" SimpleTypes:TEmpty"/>

</xsd:choice>

</xsd:group>

<xsd:group name="Value">

<xsd:choice>

<xsd:element name="integer" type="Values:IntegerValue"/>

<xsd:element name="float" type="Values:FloatValue"/>

<xsd:element name="boolean" type="Values:BooleanValue"/>

<xsd:element name="verdicttype" type="Values:VerdictValue"/>

<xsd:element name="bitstring" type="Values:BitstringValue"/>

<xsd:element name="hexstring" type="Values:HexstringValue"/>

<xsd:element name="octetstring" type="Values:OctetstringValue"/>

<xsd:element name="charstring" type="Values:CharstringValue"/>

<xsd:element name="universal\_charstring" type="Values:UniversalCharstringValue"/>

<xsd:element name="record" type="Values:RecordValue"/>

<xsd:element name="record\_of" type="Values:RecordOfValue"/>

<xsd:element name="set" type="Values:SetValue"/>

<xsd:element name="set\_of" type="Values:SetOfValue"/>

<xsd:element name="enumerated" type="Values:EnumeratedValue"/>

<xsd:element name="union" type="Values:UnionValue"/>

<xsd:element name="anytype" type="Values:AnytypeValue"/>

<xsd:element name="address" type="Values:AddressValue"/>

<xsd:element name="component" type="Values:ComponentValue"/>

<xsd:element name="port" type="Values:PortValue"/>

<xsd:element name="default" type="Values:DefaultValue"/>

<xsd:element name="timer" type="Values:TimerValue"/>

</xsd:choice>

</xsd:group>

<xsd:group name="Values">

<xsd:choice>

<xsd:sequence>  
 <xsd:choice>

<xsd:element name="integer" type="Values:IntegerValue" minOccurs="0"

maxOccurs="unbounded"/>

<xsd:element name="float" type="Values:FloatValue" minOccurs="0"

maxOccurs="unbounded"/>

<xsd:element name="boolean" type="Values:BooleanValue" minOccurs="0"

maxOccurs="unbounded"/>

<xsd:element name="bitstring" type="Values:BitstringValue"

minOccurs="0" maxOccurs="unbounded"/>

<xsd:element name="hexstring" type="Values:HexstringValue"

minOccurs="0" maxOccurs="unbounded"/>

<xsd:element name="octetstring" type="Values:OctetstringValue"

minOccurs="0" maxOccurs="unbounded"/>

<xsd:element name="charstring" type="Values:CharstringValue"

minOccurs="0" maxOccurs="unbounded"/>

<xsd:element name="universal\_charstring"

type="Values:UniversalCharstringValue" minOccurs="0"

maxOccurs="unbounded"/>

<xsd:element name="record" type="Values:RecordValue" minOccurs="0"

maxOccurs="unbounded"/>

<xsd:element name="record\_of" type="Values:RecordOfValue"

minOccurs="0" maxOccurs="unbounded"/>

<xsd:element name="set" type="Values:SetValue" minOccurs="0"

maxOccurs="unbounded"/>

<xsd:element name="set\_of" type="Values:SetOfValue"

minOccurs="0" maxOccurs="unbounded"/>

<xsd:element name="enumerated" type="Values:EnumeratedValue"

minOccurs="0" maxOccurs="unbounded"/>

<xsd:element name="union" type="Values:UnionValue" minOccurs="0"

maxOccurs="unbounded"/>

<xsd:element name="anytype" type="Values:AnytypeValue" minOccurs="0"

maxOccurs="unbounded"/>

<xsd:element name="address" type="Values:AddressValue" minOccurs="0"

maxOccurs="unbounded"/>

<xsd:element name="component" type="Values:ComponentValue" minOccurs="0"

maxOccurs="unbounded"/>

<xsd:element name="port" type="Values:PortValue" minOccurs="0"

maxOccurs="unbounded"/>

<xsd:element name="default" type="Values:DefaultValue" minOccurs="0"

maxOccurs="unbounded"/>

<xsd:element name="timer" type="Values:TimerValue" minOccurs="0"

maxOccurs="unbounded"/>

    <xsd:element name="matching\_symbol" type="Templates:MatchingSymbol"/>  
 </xsd:choice>  
 <xsd:element name="ifpresent" type="SimpleTypes:TEmpty" minOccurs="0"/>  
 <xsd:element name="length" type="Values:LengthRestriction" minOccurs="0"/>  
 </xsd:sequence>

<xsd:element name="null" type="SimpleTypes:TEmpty"/>

<xsd:element name="omit" type="SimpleTypes:TEmpty"/>

<xsd:element name="not\_evaluated" type="SimpleTypes:TEmpty"/>

</xsd:choice>

</xsd:group>

<xsd:complexType name="Value" mixed="true">

<xsd:group ref="Values:Value"/> <xsd:attributeGroup ref="Values:ValueAtts"/>

</xsd:complexType>

<xsd:complexType name="LengthRestriction">  
 <xsd:sequence>  
 <xsd:element name="lower" type="SimpleTypes:TInteger" />  
 <xsd:element name="upper" type="SimpleTypes:TInteger" minOccurs="0" />  
 </xsd:sequence>  
 </xsd:complexType>

<xsd:complexType name="Value" mixed="true">

<xsd:group ref="Values:Value"/>

<xsd:attributeGroup ref="Values:ValueAtts"/>

</xsd:complexType>

<!‑‑ general event elements ‑‑>

<xsd:complexType name="IntegerValue">

<xsd:group ref="Values:BaseValue"/>

<xsd:attributeGroup ref="Values:ValueAtts"/>

</xsd:complexType>

<xsd:complexType name="FloatValue">

<xsd:group ref="Values:BaseValue"/>

<xsd:attributeGroup ref="Values:ValueAtts"/>

</xsd:complexType>

<xsd:complexType name="BooleanValue">

<xsd:group ref="Values:BaseValue"/>

<xsd:attributeGroup ref="Values:ValueAtts"/>

</xsd:complexType>

<xsd:complexType name="VerdictValue">

<xsd:group ref="Values:BaseValue"/>

<xsd:attributeGroup ref="Values:ValueAtts"/>

</xsd:complexType>

<xsd:complexType name="BitstringValue">

<xsd:group ref="Values:BaseValue"/>

<xsd:attributeGroup ref="Values:ValueAtts"/>

</xsd:complexType>

<xsd:complexType name="HexstringValue">

<xsd:group ref="Values:BaseValue"/>

<xsd:attributeGroup ref="Values:ValueAtts"/>

</xsd:complexType>

<xsd:complexType name="OctetstringValue">

<xsd:group ref="Values:BaseValue"/>

<xsd:attributeGroup ref="Values:ValueAtts"/>

</xsd:complexType>

<xsd:complexType name="CharstringValue">

<xsd:group ref="Values:BaseValue"/>

<xsd:attributeGroup ref="Values:ValueAtts"/>

</xsd:complexType>

<xsd:complexType name="UniversalCharstringValue">

<xsd:group ref="Values:BaseValue"/>

<xsd:attributeGroup ref="Values:ValueAtts"/>

</xsd:complexType>

<xsd:complexType name="RecordValue">

<xsd:choice>

<xsd:sequence>  
 <xsd:choice>  
 <xsd:group ref="Values:Value" minOccurs="0" maxOccurs="unbounded"/>

    <xsd:element name="matching\_symbol" type="Templates:MatchingSymbol"/>  
 </xsd:choice>  
 <xsd:element name="ifpresent" type="SimpleTypes:TEmpty" minOccurs="0"/>

</xsd:sequence>

<xsd:element name="null" type="SimpleTypes:TEmpty"/>

<xsd:element name="omit" type="SimpleTypes:TEmpty"/>

<xsd:element name="not\_evaluated" type="SimpleTypes:TEmpty"/>

</xsd:choice>

<xsd:attributeGroup ref="Values:ValueAtts"/>

</xsd:complexType>

<xsd:complexType name="RecordOfValue">

<xsd:group ref="Values:Values"/>

<xsd:attributeGroup ref="Values:ValueAtts"/>

</xsd:complexType>

<xsd:complexType name="ArrayValue">

<xsd:group ref="Values:Values"/>

<xsd:attributeGroup ref="Values:ValueAtts"/>

</xsd:complexType>

<xsd:complexType name="SetValue">

<xsd:choice>

<xsd:sequence>  
 <xsd:choice>  
 <xsd:group ref="Values:Value" minOccurs="0" maxOccurs="unbounded"/>

    <xsd:element name="matching\_symbol" type="Templates:MatchingSymbol"/>  
 </xsd:choice>  
 <xsd:element name="ifpresent" type="SimpleTypes:TEmpty" minOccurs="0"/>

</xsd:sequence>

<xsd:element name="null" type="SimpleTypes:TEmpty"/>

<xsd:element name="omit" type="SimpleTypes:TEmpty"/>

<xsd:element name="not\_evaluated" type="SimpleTypes:TEmpty"/>

</xsd:choice>

<xsd:attributeGroup ref="Values:ValueAtts"/>

</xsd:complexType>

<xsd:complexType name="SetOfValue">

<xsd:group ref="Values:Values"/>

<xsd:attributeGroup ref="Values:ValueAtts"/>

</xsd:complexType>

<xsd:complexType name="EnumeratedValue">

<xsd:group ref="Values:BaseValue"/>

<xsd:attributeGroup ref="Values:ValueAtts"/>

</xsd:complexType>

<xsd:complexType name="UnionValue">

<xsd:choice>

<xsd:sequence>  
 <xsd:choice>  
 <xsd:group ref="Values:Value"/>

    <xsd:element name="matching\_symbol" type="Templates:MatchingSymbol"/>  
 </xsd:choice>  
 <xsd:element name="ifpresent" type="SimpleTypes:TEmpty" minOccurs="0"/>

</xsd:sequence>

<xsd:element name="null" type="SimpleTypes:TEmpty"/>

<xsd:element name="omit" type="SimpleTypes:TEmpty"/>

<xsd:element name="not\_evaluated" type="SimpleTypes:TEmpty"/>

</xsd:choice>

<xsd:attributeGroup ref="Values:ValueAtts"/>

</xsd:complexType>

<xsd:complexType name="AnytypeValue">

<xsd:choice>

<xsd:sequence>  
 <xsd:choice>  
 <xsd:group ref="Values:Value"/>

    <xsd:element name="matching\_symbol" type="Templates:MatchingSymbol"/>  
 </xsd:choice>  
 <xsd:element name="ifpresent" type="SimpleTypes:TEmpty" minOccurs="0"/>

</xsd:sequence>

<xsd:element name="null" type="SimpleTypes:TEmpty"/>

<xsd:element name="omit" type="SimpleTypes:TEmpty"/>

<xsd:element name="not\_evaluated" type="SimpleTypes:TEmpty"/>

</xsd:choice>

<xsd:attributeGroup ref="Values:ValueAtts"/>

</xsd:complexType>

<xsd:complexType name="AddressValue">

<xsd:group ref="Values:Value"/>

<xsd:attributeGroup ref="Values:ValueAtts"/>

</xsd:complexType>

<xsd:complexType name="ComponentValue">

<xsd:group ref="Values:BaseValue"/>

<xsd:attributeGroup ref="Values:ValueAtts"/>

</xsd:complexType>

<xsd:complexType name="PortValue">

<xsd:choice>

<xsd:element name="value" type="SimpleTypes:TString"/>

<xsd:element name="null" type="SimpleTypes:TEmpty"/>

<xsd:element name="omit" type="SimpleTypes:TEmpty"/>

</xsd:choice>

<xsd:attributeGroup ref="Values:ValueAtts"/>

</xsd:complexType>

<xsd:complexType name="DefaultValue">

<xsd:choice>

<xsd:element name="value" type="SimpleTypes:TString"/>

<xsd:element name="null" type="SimpleTypes:TEmpty"/>

<xsd:element name="omit" type="SimpleTypes:TEmpty"/>

<xsd:element name="not\_evaluated" type="SimpleTypes:TEmpty"/>

</xsd:choice>

<xsd:attributeGroup ref="Values:ValueAtts"/>

</xsd:complexType>

<xsd:complexType name="TimerValue">

<xsd:choice>

<xsd:element name="value" type="SimpleTypes:TString"/>

<xsd:element name="null" type="SimpleTypes:TEmpty"/>

<xsd:element name="omit" type="SimpleTypes:TEmpty"/>

</xsd:choice>

<xsd:attributeGroup ref="Values:ValueAtts"/>

</xsd:complexType>

</xsd:schema>

# B.4 TCI‑TL XML Schema for Templates

<?xml version="1.0" encoding="UTF‑8"?>

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"

targetNamespace="http://uri.etsi.org/ttcn‑3/tci/Templates\_v4\_10\_1.xsd"

xmlns:Templates="http://uri.etsi.org/ttcn‑3/tci/Templates\_v4\_10\_1.xsd"

xmlns:Values="http://uri.etsi.org/ttcn‑3/tci/Values\_v4\_10\_1.xsd"

xmlns:SimpleTypes="http://uri.etsi.org/ttcn‑3/tci/SimpleTypes\_v4\_10\_1.xsd" elementFormDefault="qualified">

<xsd:import namespace="http://uri.etsi.org/ttcn‑3/tci/Values\_v4\_10\_1.xsd"

schemaLocation="Values\_v4\_10\_1.xsd"/>

<xsd:import namespace="http://uri.etsi.org/ttcn‑3/tci/SimpleTypes\_v4\_10\_1.xsd"

schemaLocation="SimpleTypes\_v4\_10\_1.xsd"/>

<xsd:group name="TypedTemplate">

<xsd:choice>

<xsd:element name="integer" type="Templates:SimpleTemplate"/>

<xsd:element name="float" type="Templates:SimpleTemplate"/>

<xsd:element name="boolean" type="Templates:SimpleTemplate"/>

<xsd:element name="verdicttype" type="Templates:SimpleTemplate"/>

<xsd:element name="bitstring" type="Templates:SimpleTemplate"/>

<xsd:element name="hexstring" type="Templates:SimpleTemplate"/>

<xsd:element name="octetstring" type="Templates:SimpleTemplate"/>

<xsd:element name="charstring" type="Templates:SimpleTemplate"/>

<xsd:element name="universal\_charstring" type="Templates:SimpleTemplate"/>

<xsd:element name="record" type="Templates:RecordTemplate"/>

<xsd:element name="record\_of" type="Templates:RecordOfTemplate"/>

<xsd:element name="array" type="Templates:RecordOfTemplate"/>

<xsd:element name="set" type="Templates:RecordTemplate"/>

<xsd:element name="set\_of" type="Templates:RecordOfTemplate"/>

<xsd:element name="enumerated" type="Templates:SimpleTemplate"/>

<xsd:element name="union" type="Templates:UnionTemplate"/>

<xsd:element name="anytype" type="Templates:AnytypeTemplate"/>

<xsd:element name="address" type="Templates:AddressTemplate"/>

</xsd:choice>

</xsd:group>

<xsd:group name="SpecialTemplate">

<xsd:choice>

<xsd:element name="omit" type="Templates:omit"/>

<xsd:element name="any" type="Templates:any"/>

<xsd:element name="anyoromit" type="Templates:anyoromit"/>

<xsd:element name="templateDef" type="SimpleTypes:TString"/>

</xsd:choice>

</xsd:group>

<xsd:complexType name="TciValueTemplate">

<xsd:choice>

<xsd:group ref="Values:Value"/>

<xsd:group ref="Templates:TypedTemplate"/>

<xsd:group ref="Templates:SpecialTemplate"/>

</xsd:choice>

</xsd:complexType>

<xsd:complexType name="omit">

<xsd:attributeGroup ref="Values:ValueAtts"/>

</xsd:complexType>

<xsd:complexType name="any">

<xsd:attributeGroup ref="Values:ValueAtts"/>

</xsd:complexType>

<xsd:complexType name="anyoromit">

<xsd:attributeGroup ref="Values:ValueAtts"/>

</xsd:complexType>

<xsd:complexType name="MatchingSymbol">

<xsd:choice>

<xsd:element name="any\_value" type="SimpleTypes:TEmpty"/>

<xsd:element name="any\_value\_or\_none" type="SimpleTypes:TEmpty"/>

<xsd:element name="any\_element " type="SimpleTypes:TEmpty"/>

<xsd:element name="any\_element\_or\_none" type="SimpleTypes:TEmpty"/>

<xsd:element name="range" type="Templates:Range"/>

<xsd:element name="list" type="Templates:MatchingList"/>

<xsd:element name="complement" type="Templates:MatchingList"/>

<xsd:element name="subset" type="Templates:MatchingList"/>

<xsd:element name="superset" type="Templates:MatchingList"/>

<xsd:element name="permutation" type="Templates:MatchingList"/>

<xsd:element name="pattern" type="Templates:Pattern"/>

<xsd:element name="decmatch" type="Templates:DecMatch"/>

</xsd:choice>

</xsd:complexType>

<xsd:complexType name="Range">

<xsd:sequence>

<xsd:element name="excludeLower" minOccurs="0"/>

<xsd:element name="lower" type="Values:Value" minOccurs="0"/>

<xsd:element name="excludeUpper" minOccurs="0"/>

<xsd:element name="upper" type="Values:Value" minOccurs="0"/>

</xsd:sequence>

</xsd:complexType>

<xsd:complexType name="MatchingList">

<xsd:sequence>

<xsd:group ref="Values:Value" minOccurs="0" maxOccurs="unbounded"/>

</xsd:sequence>

</xsd:complexType>

<xsd:complexType name="Pattern">

<xsd:sequence>

<xsd:choice>

<xsd:element name="charstring" type="Values:CharstringValue"/>

<xsd:element name="universal\_charstring" type="Values:UniversalCharstringValue"/>

</xsd:choice>

</xsd:sequence>

</xsd:complexType>

<xsd:complexType name="DecMatch">

<xsd:sequence>

<xsd:group ref="Values:Value"/>

</xsd:sequence>

</xsd:complexType>

<xsd:complexType name="TciNonValueTemplate">

<xsd:sequence>

<xsd:choice>

<xsd:element name="any" type="Templates:any"/>

<xsd:element name="all" type="Templates:all"/>

<xsd:element name="templateDef" type="SimpleTypes:TString"/>

<xsd:element name="null" type="Templates:null"/>

<xsd:group ref="Values:Value"/>

</xsd:choice>

</xsd:sequence>

</xsd:complexType>

<xsd:complexType name="all">

<xsd:attributeGroup ref="Values:ValueAtts"/>

</xsd:complexType>

<xsd:complexType name="null">

<xsd:attributeGroup ref="Values:ValueAtts"/>

</xsd:complexType>

<xsd:complexType name="TciValueDifference">

<xsd:sequence>

<xsd:element name="val" type="SimpleTypes:xpath"/>

<xsd:element name="tmpl" type="SimpleTypes:xpath"/>

</xsd:sequence>

<xsd:attributeGroup ref="Values:ValueAtts"/>

<xsd:attribute name="desc" type="SimpleTypes:TString" use="optional"/>

</xsd:complexType>

<xsd:complexType name="TciValueDifferenceList">

<xsd:sequence>

<xsd:element name="diff" type="Templates:TciValueDifference" maxOccurs="unbounded"/>

</xsd:sequence>

</xsd:complexType>

<xsd:complexType name="SimpleTemplate">

<xsd:choice>

<xsd:element name="value" type="SimpleTypes:TString"/>

<xsd:group ref="Templates:SpecialTemplate"/>

</xsd:choice>

<xsd:attributeGroup ref="Values:ValueAtts"/>

</xsd:complexType>

<xsd:complexType name="RecordTemplate">

<xsd:choice>

<xsd:group ref="Templates:TypedTemplate" minOccurs="0" maxOccurs="unbounded"/>

<xsd:group ref="Templates:SpecialTemplate"/>

<xsd:element name="null" type="Templates:null"/>

</xsd:choice>

<xsd:attributeGroup ref="Values:ValueAtts"/>

</xsd:complexType>

<xsd:complexType name="RecordOfTemplate">

<xsd:choice minOccurs="0" maxOccurs="unbounded">

<xsd:element name="integer" type="Templates:SimpleTemplate" minOccurs="0"

maxOccurs="unbounded"/>

<xsd:element name="float" type="Templates:SimpleTemplate" minOccurs="0"

maxOccurs="unbounded"/>

<xsd:element name="boolean" type="Templates:SimpleTemplate" minOccurs="0"

maxOccurs="unbounded"/>

<xsd:element name="verdicttype" type="Templates:SimpleTemplate" minOccurs="0"

maxOccurs="unbounded"/>

<xsd:element name="bitstring" type="Templates:SimpleTemplate"

minOccurs="0" maxOccurs="unbounded"/>

<xsd:element name="hexstring" type="Templates:SimpleTemplate"

minOccurs="0" maxOccurs="unbounded"/>

<xsd:element name="octetstring" type="Templates:SimpleTemplate"

minOccurs="0" maxOccurs="unbounded"/>

<xsd:element name="charstring" type="Templates:SimpleTemplate"

minOccurs="0" maxOccurs="unbounded"/>

<xsd:element name="universal\_charstring"

type="Templates:SimpleTemplate" minOccurs="0"

maxOccurs="unbounded"/>

<xsd:element name="record" type="Templates:RecordTemplate" minOccurs="0"

maxOccurs="unbounded"/>

<xsd:element name="record\_of" type="Templates:RecordOfTemplate"

minOccurs="0" maxOccurs="unbounded"/>

<xsd:element name="array" type="Templates:RecordOfTemplate" minOccurs="0"

maxOccurs="unbounded"/>

<xsd:element name="set" type="Templates:RecordTemplate" minOccurs="0"

maxOccurs="unbounded"/>

<xsd:element name="set\_of" type="Templates:RecordOfTemplate"

minOccurs="0" maxOccurs="unbounded"/>

<xsd:element name="enumerated" type="Templates:SimpleTemplate"

minOccurs="0" maxOccurs="unbounded"/>

<xsd:element name="union" type="Templates:UnionTemplate" minOccurs="0"

maxOccurs="unbounded"/>

<xsd:element name="anytype" type="Templates:AnytypeTemplate" minOccurs="0"

maxOccurs="unbounded"/>

<xsd:element name="address" type="Templates:AddressTemplate" minOccurs="0"

maxOccurs="unbounded"/>

<xsd:group ref="Templates:SpecialTemplate"/> <xsd:element name="null" type="Templates:null"/>

</xsd:choice>

<xsd:attributeGroup ref="Values:ValueAtts"/>

</xsd:complexType>

<xsd:complexType name="UnionTemplate">

<xsd:choice>

<xsd:group ref="Templates:TypedTemplate" />

<xsd:group ref="Templates:SpecialTemplate"/>

<xsd:element name="null" type="Templates:null"/>

</xsd:choice>

<xsd:attributeGroup ref="Values:ValueAtts"/>

</xsd:complexType>

<xsd:complexType name="AnytypeTemplate">

<xsd:choice>

<xsd:group ref="Templates:TypedTemplate" />

<xsd:group ref="Templates:SpecialTemplate"/>

<xsd:element name="null" type="Templates:null"/>

</xsd:choice>

<xsd:attributeGroup ref="Values:ValueAtts"/>

</xsd:complexType>

<xsd:complexType name="AddressTemplate">

<xsd:choice>

<xsd:group ref="Templates:TypedTemplate" />

<xsd:group ref="Templates:SpecialTemplate"/>

<xsd:element name="null" type="Templates:null"/>

</xsd:choice>

<xsd:attributeGroup ref="Values:ValueAtts"/>

</xsd:complexType>

</xsd:schema>

# B.5 TCI‑TL XML Schema for Events

<?xml version="1.0" encoding="UTF‑8"?>

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"

targetNamespace="http://uri.etsi.org/ttcn‑3/tci/Events\_v4\_10\_1.xsd"

xmlns:Events="http://uri.etsi.org/ttcn‑3/tci/Events\_v4\_10\_1.xsd"

xmlns:Types="http://uri.etsi.org/ttcn‑3/tci/Types\_v4\_10\_1.xsd"

xmlns:Templates="http://uri.etsi.org/ttcn‑3/tci/Templates\_v4\_10\_1.xsd"

xmlns:SimpleTypes="http://uri.etsi.org/ttcn‑3/tci/SimpleTypes\_v4\_10\_1.xsd"

xmlns:Values="http://uri.etsi.org/ttcn‑3/tci/Values\_v4\_10\_1.xsd"

elementFormDefault="qualified">

<xsd:import namespace="http://uri.etsi.org/ttcn‑3/tci/SimpleTypes\_v4\_10\_1.xsd"

schemaLocation="SimpleTypes\_v4\_10\_1.xsd"/>

<xsd:import namespace="http://uri.etsi.org/ttcn‑3/tci/Types\_v4\_10\_1.xsd" schemaLocation="Types\_v4\_10\_1.xsd"/>

<xsd:import namespace="http://uri.etsi.org/ttcn‑3/tci/Values\_v4\_10\_1.xsd"

schemaLocation="Values\_v4\_10\_1.xsd"/>

<xsd:import namespace="http://uri.etsi.org/ttcn‑3/tci/Templates\_v4\_10\_1.xsd"

schemaLocation="Templates\_v4\_10\_1.xsd"/>

<!‑‑ common definition for all events ‑‑>

<xsd:complexType name="Event" mixed="true">

<xsd:sequence>

<xsd:element name="am" type="SimpleTypes:TString"/>

</xsd:sequence>

<xsd:attribute name="ts" type="xsd:long" use="required"/>

<xsd:attribute name="src" type="SimpleTypes:TString" use="optional"/>

<xsd:attribute name="line" type="SimpleTypes:TInteger" use="optional"/>

<!‑‑ general identifier structure for test components, ports and timer ‑‑>

<xsd:attribute name="name" type="SimpleTypes:TString" use="required"/>

<xsd:attribute name="id" type="SimpleTypes:TString" use="required"/>

<xsd:attribute name="type" type="SimpleTypes:TString" use="required"/>

</xsd:complexType>

<!‑‑ this event is extended by all port configuration events ‑‑>

<xsd:complexType name="PortConfiguration">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="port1" type="Types:TriPortIdType" />

<xsd:element name="port2" type="Types:TriPortIdType" />

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<!‑‑ this event is extended by all port status events ‑‑>

<xsd:complexType name="PortStatus">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="port" type="Types:TriPortIdType"/>

<xsd:element name="stat" type="SimpleTypes:PortStatusType" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<!‑‑ testcases ‑‑>

<xsd:complexType name="tliTcExecute">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="tcId" type="Types:TciTestCaseIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="dur" type="SimpleTypes:TriTimerDurationType" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliTcStart">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="tcId" type="Types:TciTestCaseIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="dur" type="SimpleTypes:TriTimerDurationType" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliTcStop">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event"/>

<xsd:sequence>

<xsd:element name="reason" type="SimpleTypes:TString" minOccurs="0"/>

</xsd:sequence>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliTcStarted">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="tcId" type="Types:TciTestCaseIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="dur" type="SimpleTypes:TriTimerDurationType" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliTcTerminated">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="tcId" type="Types:TciTestCaseIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="verdict" type="Values:VerdictValue"/>

<xsd:element name="reason" type="SimpleTypes:TString" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<!‑‑ control ‑‑>

<xsd:complexType name="tliCtrlStart">

<xsd:complexContent>

<xsd:extension base="Events:Event"/>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliCtrlStop">

<xsd:complexContent>

<xsd:extension base="Events:Event"/>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliCtrlTerminated">

<xsd:complexContent>

<xsd:extension base="Events:Event"/>

</xsd:complexContent>

</xsd:complexType>

<!‑‑ asynchronous communication ‑‑>

<xsd:complexType name="tliMSend\_m">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="to" type="Types:TriPortIdType" minOccurs="0"/>

<xsd:element name="msgValue" type="Values:Value"/>

<xsd:element name="addrValue " type="Values:Value" minOccurs="0"/>

<xsd:choice>

<xsd:element name="encoder‑failure" type="SimpleTypes:TciStatusType" minOccurs="0"/>

<xsd:sequence>

<xsd:element name="msg" type="Types:TriMessageType" minOccurs="0"/>

<xsd:element name="address" type="Types:TriAddressType" minOccurs="0"/>

<xsd:element name="transmission‑failure" type="SimpleTypes:TriStatusType" minOccurs="0"/>

</xsd:sequence>

</xsd:choice>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliMSend\_m\_BC">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="to" type="Types:TriPortIdType" minOccurs="0"/>

<xsd:element name="msgValue" type="Values:Value"/>

<xsd:choice>

<xsd:element name="encoder‑failure" type="SimpleTypes:TciStatusType" minOccurs="0"/>

<xsd:sequence>

<xsd:element name="msg" type="Types:TriMessageType" minOccurs="0"/>

<xsd:element name="transmission‑failure" type="SimpleTypes:TriStatusType" minOccurs="0"/>

</xsd:sequence>

</xsd:choice>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliMSend\_m\_MC">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="to" type="Types:TriPortIdType" minOccurs="0"/>

<xsd:element name="msgValue" type="Values:Value"/>

<xsd:element name="addrValues" type="Types:TciValueListType" minOccurs="0"/>

<xsd:choice>

<xsd:element name="encoder‑failure" type="SimpleTypes:TciStatusType" minOccurs="0"/>

<xsd:sequence>

<xsd:element name="msg" type="Types:TriMessageType" minOccurs="0"/>

<xsd:element name="addresses" type="Types:TriAddressListType" minOccurs="0"/>

<xsd:element name="transmission‑failure" type="SimpleTypes:TriStatusType" minOccurs="0"/>

</xsd:sequence>

</xsd:choice>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliMSend\_c">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="to" type="Types:TriPortIdType" minOccurs="0"/>

<xsd:element name="msgValue" type="Values:Value"/>

<xsd:element name="transmission‑failure" type="SimpleTypes:TriStatusType" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliMSend\_c\_BC">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="to" type="Types:TriPortIdListType" minOccurs="0"/>

<xsd:element name="msgValue" type="Values:Value"/>

<xsd:element name="transmission‑failure" type="SimpleTypes:TriStatusType" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliMSend\_c\_MC">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="to" type="Types:TriPortIdListType" minOccurs="0"/>

<xsd:element name="msgValue" type="Values:Value"/>

<xsd:element name="transmission‑failure" type="SimpleTypes:TriStatusType" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliMDetected\_m">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="from" type="Types:TriPortIdType" minOccurs="0"/>

<xsd:element name="msgValue" type="Types:TriMessageType"/>

<xsd:element name="address" type="Types:TriAddressType" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliMDetected\_c">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="from" type="Types:TriPortIdType" minOccurs="0"/>

<xsd:element name="msgValue" type="Values:Value"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliMMismatch\_m">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="msgValue" type="Values:Value"/>

<xsd:element name="msgTmpl" type="Templates:TciValueTemplate"/>

<xsd:element name="diffs" type="Templates:TciValueDifferenceList"/>

<xsd:element name="addrValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="addressTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliMMismatch\_c">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="msgValue" type="Values:Value"/>

<xsd:element name="msgTmpl" type="Templates:TciValueTemplate"/>

<xsd:element name="diffs" type="Templates:TciValueDifferenceList"/>

<xsd:element name="from" type="Types:TriComponentIdType" minOccurs="0"/>

<xsd:element name="fromTmpl" type="Templates:TciNonValueTemplate" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliMReceive\_m">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="msgValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="msgTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>

<xsd:element name="addrValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="addressTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliMReceive\_c">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="msgValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="msgTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>

<xsd:element name="from" type="Types:TriComponentIdType" minOccurs="0"/>

<xsd:element name="fromTmpl" type="Templates:TciNonValueTemplate" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<!‑‑ synchronous communication ‑‑>

<xsd:complexType name="tliPrCall\_m">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="to" type="Types:TriPortIdType" minOccurs="0"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="addrValue" type="Values:Value" minOccurs="0"/>

<xsd:choice>

<xsd:element name="encoder‑failure" type="SimpleTypes:TciStatusType" minOccurs="0"/>

<xsd:sequence>

<xsd:element name="triPars" type="Types:TriParameterListType" minOccurs="0"/>

<xsd:element name="address" type="Types:TriAddressType" minOccurs="0"/>

<xsd:element name="transmission‑failure" type="SimpleTypes:TriStatusType" minOccurs="0"/>

</xsd:sequence>

</xsd:choice>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrCall\_m\_BC">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="to" type="Types:TriPortIdType" minOccurs="0"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:choice>

<xsd:element name="encoder‑failure" type="SimpleTypes:TciStatusType" minOccurs="0"/>

<xsd:sequence>

<xsd:element name="triPars" type="Types:TriParameterListType" minOccurs="0"/>

<xsd:element name="transmission‑failure" type="SimpleTypes:TriStatusType" minOccurs="0"/>

</xsd:sequence>

</xsd:choice>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrCall\_m\_MC">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="to" type="Types:TriPortIdType" minOccurs="0"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="addrValues" type="Types:TciValueListType" minOccurs="0"/>

<xsd:choice>

<xsd:element name="encoder‑failure" type="SimpleTypes:TciStatusType" minOccurs="0"/>

<xsd:sequence>

<xsd:element name="triPars" type="Types:TriParameterListType" minOccurs="0"/>

<xsd:element name="addresses" type="Types:TriAddressListType" minOccurs="0"/>

<xsd:element name="transmission‑failure" type="SimpleTypes:TriStatusType" minOccurs="0"/>

</xsd:sequence>

</xsd:choice>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrCall\_c">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="to" type="Types:TriPortIdType" minOccurs="0"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="transmission‑failure" type="SimpleTypes:TriStatusType" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrCall\_c\_BC">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="to" type="Types:TriPortIdListType" minOccurs="0"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="transmission‑failure" type="SimpleTypes:TriStatusType" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrCall\_c\_MC">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="to" type="Types:TriPortIdListType" minOccurs="0"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="transmission‑failure" type="SimpleTypes:TriStatusType" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrGetCallDetected\_m">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="from" type="Types:TriPortIdType" minOccurs="0"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="triPars" type="Types:TriParameterListType" minOccurs="0"/>

<xsd:element name="address" type="Types:TriAddressType" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrGetCallDetected\_c">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="from" type="Types:TriPortIdType" minOccurs="0"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrGetCallMismatch\_m">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="parsTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>

<xsd:element name="diffs" type="Templates:TciValueDifferenceList"/>

<xsd:element name="addrValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="addressTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrGetCallMismatch\_c">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="parsTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>

<xsd:element name="diffs" type="Templates:TciValueDifferenceList"/>

<xsd:element name="from" type="Types:TriComponentIdType" minOccurs="0"/>

<xsd:element name="fromTmpl" type="Templates: TciNonValueTemplate" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrGetCall\_m">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="parsTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>

<xsd:element name="addrValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="addressTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrGetCall\_c">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="parsTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>

<xsd:element name="from" type="Types:TriComponentIdType" minOccurs="0"/>

<xsd:element name="fromTmpl" type="Templates:TciNonValueTemplate" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrReply\_m">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="to" type="Types:TriPortIdType" minOccurs="0"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="replValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="addrValue" type="Values:Value" minOccurs="0"/>

<xsd:choice>

<xsd:element name="encoder‑failure" type="SimpleTypes:TciStatusType"

minOccurs="0"/>

<xsd:sequence>

<xsd:element name="triPars" type="Types:TriParameterListType" minOccurs="0"/>

<xsd:element name="repl" type="Types:TriParameterType" minOccurs="0"/>

<xsd:element name="address" type="Types:TriAddressType" minOccurs="0"/>

<xsd:element name="transmission‑failure"

type="SimpleTypes:TriStatusType" minOccurs="0"/>

</xsd:sequence>

</xsd:choice>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrReply\_m\_BC">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="to" type="Types:TriPortIdType" minOccurs="0"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="replValue" type="Values:Value" minOccurs="0"/>

<xsd:choice>

<xsd:element name="encoder‑failure" type="SimpleTypes:TciStatusType"

minOccurs="0"/>

<xsd:sequence>

<xsd:element name="triPars" type="Types:TriParameterListType" minOccurs="0"/>

<xsd:element name="repl" type="Types:TriParameterType" minOccurs="0"/>

<xsd:element name="transmission‑failure"

type="SimpleTypes:TriStatusType" minOccurs="0"/>

</xsd:sequence>

</xsd:choice>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrReply\_m\_MC">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="to" type="Types:TriPortIdType" minOccurs="0"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="replValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="addrValues" type="Types:TciValueListType" minOccurs="0"/>

<xsd:choice>

<xsd:element name="encoder‑failure" type="SimpleTypes:TciStatusType"

minOccurs="0"/>

<xsd:sequence>

<xsd:element name="triPars" type="Types:TriParameterListType" minOccurs="0"/>

<xsd:element name="repl" type="Types:TriParameterType" minOccurs="0"/>

<xsd:element name="addresses" type="Types:TriAddressListType" minOccurs="0"/>

<xsd:element name="transmission‑failure"

type="SimpleTypes:TriStatusType" minOccurs="0"/>

</xsd:sequence>

</xsd:choice>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrReply\_c">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="to" type="Types:TriPortIdType" minOccurs="0"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="replValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="transmission‑failure" type="SimpleTypes:TriStatusType" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrReply\_c\_BC">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="to" type="Types:TriPortIdListType" minOccurs="0"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="replValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="transmission‑failure" type="SimpleTypes:TriStatusType" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrReply\_c\_MC">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="to" type="Types:TriPortIdListType" minOccurs="0"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="replValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="transmission‑failure" type="SimpleTypes:TriStatusType" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrGetReplyDetected\_m">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="from" type="Types:TriPortIdType" minOccurs="0"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="triPars" type="Types:TriParameterListType" minOccurs="0"/>

<xsd:element name="repl" type="Types:TriParameterType" minOccurs="0"/>

<xsd:element name="address" type="Types:TriAddressType" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrGetReplyDetected\_c">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="from" type="Types:TriPortIdType" minOccurs="0"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="replValue" type="Values:Value" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrGetReplyMismatch\_m">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="parsTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>

<xsd:element name="replValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="replTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>

<xsd:element name="diffs" type="Templates:TciValueDifferenceList"/>

<xsd:element name="addrValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="addressTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrGetReplyMismatch\_c">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="parsTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>

<xsd:element name="replValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="replTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>

<xsd:element name="diffs" type="Templates:TciValueDifferenceList"/>

<xsd:element name="from" type="Types:TriComponentIdType" minOccurs="0"/>

<xsd:element name="fromTmpl" type="Templates:TciNonValueTemplate" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrGetReply\_m">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="parsTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>

<xsd:element name="replValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="replTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>

<xsd:element name="addrValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="addressTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrGetReply\_c">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="parsTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>

<xsd:element name="replValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="replTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>

<xsd:element name="from" type="Types:TriComponentIdType" minOccurs="0"/>

<xsd:element name="fromTmpl" type="Templates:TciNonValueTemplate" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrRaise\_m">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="to" type="Types:TriPortIdType" minOccurs="0"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="excValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="addrValue" type="Values:Value" minOccurs="0"/>

<xsd:choice>

<xsd:element name="encoder‑failure" type="SimpleTypes:TciStatusType" minOccurs="0"/>

<xsd:sequence>

<xsd:element name="exc" type="Types:TriExceptionType" minOccurs="0"/>

<xsd:element name="address" type="Types:TriAddressType" minOccurs="0"/>

<xsd:element name="transmission‑failure" type="SimpleTypes:TriStatusType" minOccurs="0"/>

</xsd:sequence>

</xsd:choice>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrRaise\_m\_BC">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="to" type="Types:TriPortIdType" minOccurs="0"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="excValue" type="Values:Value" minOccurs="0"/>

<xsd:choice>

<xsd:element name="encoder‑failure" type="SimpleTypes:TciStatusType" minOccurs="0"/>

<xsd:sequence>

<xsd:element name="exc" type="Types:TriExceptionType" minOccurs="0"/>

<xsd:element name="transmission‑failure" type="SimpleTypes:TriStatusType" minOccurs="0"/>

</xsd:sequence>

</xsd:choice>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrRaise\_m\_MC">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="to" type="Types:TriPortIdType" minOccurs="0"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="excValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="addrValues" type="Types:TciValueListType" minOccurs="0"/>

<xsd:choice>

<xsd:element name="encoder‑failure" type="SimpleTypes:TciStatusType" minOccurs="0"/>

<xsd:sequence>

<xsd:element name="exc" type="Types:TriExceptionType" minOccurs="0"/>

<xsd:element name="addresses" type="Types:TriAddressListType" minOccurs="0"/>

<xsd:element name="transmission‑failure" type="SimpleTypes:TriStatusType" minOccurs="0"/>

</xsd:sequence>

</xsd:choice>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrRaise\_c">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="to" type="Types:TriPortIdType" minOccurs="0"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="excValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="transmission‑failure" type="SimpleTypes:TriStatusType" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrRaise\_c\_BC">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="to" type="Types:TriPortIdListType" minOccurs="0"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="excValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="transmission‑failure" type="SimpleTypes:TriStatusType" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrRaise\_c\_MC">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="to" type="Types:TriPortIdListType" minOccurs="0"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="excValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="transmission‑failure" type="SimpleTypes:TriStatusType" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrCatchDetected\_m">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="from" type="Types:TriPortIdType" minOccurs="0"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="exc" type="Types:TriExceptionType" minOccurs="0"/>

<xsd:element name="address" type="Types:TriAddressType" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrCatchDetected\_c">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="from" type="Types:TriPortIdType" minOccurs="0"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="excValue" type="Values:Value" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrCatchMismatch\_m">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="excValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="excTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>

<xsd:element name="diffs" type="Templates:TciValueDifferenceList"/>

<xsd:element name="addrValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="addressTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrCatchMismatch\_c">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="excValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="excTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>

<xsd:element name="diffs" type="Templates:TciValueDifferenceList"/>

<xsd:element name="from" type="Types:TriComponentIdType" minOccurs="0"/>

<xsd:element name="fromTmpl" type="Templates:TciNonValueTemplate" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrCatch\_m">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="excValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="excTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>

<xsd:element name="addrValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="addressTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrCatch\_c">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="excValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="excTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>

<xsd:element name="from" type="Types:TriComponentIdType" minOccurs="0"/>

<xsd:element name="fromTmpl" type="Templates:TciNonValueTemplate" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrCatchTimeoutDetected">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrCatchTimeout">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<!‑‑ components ‑‑>

<xsd:complexType name="tliCCreate">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="comp" type="Types:TriComponentIdType"/>

<xsd:element name="name" type="SimpleTypes:TString"/>

<xsd:element name="hostId" type="Values:Value" minOccurs="0"/>

<xsd:element name="alive" type="SimpleTypes:TBoolean"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliCStart">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="comp" type="Types:TriComponentIdType"/>

<xsd:element name="name" type="Types:TciBehaviourIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliCRunning">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="comp" type="Types:TriComponentIdType"/>

<xsd:element name="status" type="SimpleTypes:ComponentStatusType"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliCAlive">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="comp" type="Types:TriComponentIdType"/>

<xsd:element name="status" type="SimpleTypes:ComponentStatusType"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliCStop">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="comp" type="Types:TriComponentIdType"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliCKill">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="comp" type="Types:TriComponentIdType"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliCDoneMismatch">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="comp" type="Types:TriComponentIdType"/>

<xsd:element name="compTmpl" type="Templates:TciNonValueTemplate"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliCKilledMismatch">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="comp" type="Types:TriComponentIdType"/>

<xsd:element name="compTmpl" type="Templates:TciNonValueTemplate"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliCDone">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="compTmpl" type="Templates:TciNonValueTemplate"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliCKilled">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="compTmpl" type="Templates:TciNonValueTemplate"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliCTerminated">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="verdict" type="Values:VerdictValue" />

<xsd:element name="reason" type="SimpleTypes:TString" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<!‑‑ ports ‑‑>

<xsd:complexType name="tliPConnect">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:PortConfiguration"/>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPDisconnect">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:PortConfiguration"/>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPMap">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:PortConfiguration"/>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPMapParam">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:tliPMap">

<xsd:sequence>

<xsd:element name="tciPars" type="Types:TciParameterListType" />

<xsd:choice>

<xsd:element name="encoder‑failure" type="SimpleTypes:TciStatusType" minOccurs="0"/>

<xsd:element name="triPars" type="Types:TriParameterListType" />

</xsd:choice>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPUnmap">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:PortConfiguration"/>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPUnmapParam">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:tliPUnmap">

<xsd:sequence>

<xsd:element name="tciPars" type="Types:TciParameterListType" />

<xsd:choice>

<xsd:element name="encoder‑failure" type="SimpleTypes:TciStatusType" minOccurs="0"/>

<xsd:element name="triPars" type="Types:TriParameterListType"/>

</xsd:choice>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPClear">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:PortStatus"/>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPStart">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:PortStatus"/>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPStop">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:PortStatus"/>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPHalt">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:PortStatus"/>

</xsd:complexContent>

</xsd:complexType>

<!‑‑ codec ‑‑>

<xsd:complexType name="tliEncode">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="val" type="Values:Value"/>

<xsd:choice>

<xsd:element name="msg" type="Types:TriMessageType"/>

<xsd:element name="encoder‑failure" type="SimpleTypes:TciStatusType" minOccurs="0"/>

</xsd:choice>

<xsd:element name="codec" type="SimpleTypes:TString"

minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliDecode" mixed="true">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="msg" type="Types:TriMessageType"/>

<xsd:choice>

<xsd:element name="decoder‑failure" type="SimpleTypes:TciStatusType" minOccurs="0"/>

<xsd:element name="val" type="Values:Value"/>

</xsd:choice>

<xsd:element name="codec" type="SimpleTypes:TString"

minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<!‑‑ timers ‑‑>

<xsd:complexType name="tliTTimeoutDetected">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="timer" type="Types:TriTimerIdType" />

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliTTimeoutMismatch">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="timer" type="Types:TriTimerIdType" />

<xsd:element name="timerTmpl" type="Templates:TciNonValueTemplate" />

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliTTimeout">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="timer" type="Types:TriTimerIdType" />

<xsd:element name="timerTmpl" type="Templates:TciNonValueTemplate" />

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliTStart">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="timer" type="Types:TriTimerIdType"/>

<xsd:element name="dur" type="SimpleTypes:TriTimerDurationType"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliTStop">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="timer" type="Types:TriTimerIdType"/>

<xsd:element name="dur" type="SimpleTypes:TriTimerDurationType"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliTRead">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="timer" type="Types:TriTimerIdType"/>

<xsd:element name="elapsed" type="SimpleTypes:TriTimerDurationType"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliTRunning">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="timer" type="Types:TriTimerIdType"/>

<xsd:element name="status" type="SimpleTypes:TimerStatusType"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<!‑‑ scope ‑‑>

<xsd:complexType name="tliSEnter">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="name" type="Types:QualifiedName" />

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="kind" type="SimpleTypes:TString"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliSLeave">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="name" type="Types:QualifiedName" />

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="returnValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="kind" type="SimpleTypes:TString"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<!‑‑ variables and module parameter ‑‑>

<xsd:complexType name="tliVar">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="name" type="Types:QualifiedName" />

<xsd:element name="val" type="Values:Value" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliModulePar">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="name" type="Types:QualifiedName" />

<xsd:element name="val" type="Values:Value" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<!‑‑ verdicts ‑‑>

<xsd:complexType name="tliGetVerdict">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="verdict" type="Values:VerdictValue"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliSetVerdict">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="verdict" type="Values:VerdictValue"/>

<xsd:element name="reason" type="SimpleTypes:TString" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<!‑‑ log ‑‑>

<xsd:complexType name="tliLog">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="log" type="SimpleTypes:TString"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<!‑‑ alt ‑‑>

<xsd:complexType name="tliAEnter">

<xsd:complexContent>

<xsd:extension base="Events:Event"/>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliALeave">

<xsd:complexContent>

<xsd:extension base="Events:Event"/>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliADefaults">

<xsd:complexContent>

<xsd:extension base="Events:Event"/>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliAActivate">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="name" type="Types:QualifiedName" />

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="ref" type="Values:Value"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliADeactivate">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="ref" type="Values:Value"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliANomatch">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event"/>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliARepeat">

<xsd:complexContent>

<xsd:extension base="Events:Event"/>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliAWait">

<xsd:complexContent>

<xsd:extension base="Events:Event"/>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliAction">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="action" type="SimpleTypes:TString"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliMatch">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="expr" type="Values:Value"/>

<xsd:element name="tmpl" type="Templates:TciValueTemplate"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliMatchMismatch">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="expr" type="Values:Value"/>

<xsd:element name="tmpl" type="Templates:TciValueTemplate"/>

<xsd:element name="diffs" type="Templates:TciValueDifferenceList"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliInfo">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="level" type="SimpleTypes:TInteger"/>

<xsd:element name="info" type="SimpleTypes:TString"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliMChecked\_m">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="msgValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="msgTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>

<xsd:element name="addrValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="addressTmpl" type="Templates:TciValueTemplate"

minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliMChecked\_c">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="msgValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="msgTmpl" type="Templates:TciValueTemplate"

minOccurs="0"/>

<xsd:element name="from" type="Types:TriComponentIdType" minOccurs="0"/>

<xsd:element name="fromTmpl" type="Templates:TciNonValueTemplate"

minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrGetCallChecked\_m">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="parsTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>

<xsd:element name="addrValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="addressTmpl" type="Templates:TciValueTemplate"

minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrGetCallChecked\_c">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="parsTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>

<xsd:element name="from" type="Types:TriComponentIdType" minOccurs="0"/>

<xsd:element name="fromTmpl" type="Templates:TciNonValueTemplate"

minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrGetReplyChecked\_m">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="parsTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>

<xsd:element name="replValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="replTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>

<xsd:element name="addrValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="addressTmpl" type="Templates:TciValueTemplate"

minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrGetReplyChecked\_c">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="parsTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>

<xsd:element name="replValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="replTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>

<xsd:element name="from" type="Types:TriComponentIdType" minOccurs="0"/>

<xsd:element name="fromTmpl" type="Templates:TciNonValueTemplate"

minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrCatchChecked\_m">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="excValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="excTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>

<xsd:element name="addrValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="addressTmpl" type="Templates:TciValueTemplate"

minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliPrCatchChecked\_c">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="signature" type="Types:TriSignatureIdType"/>

<xsd:element name="excValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="excTmpl" type="Templates:TciValueTemplate" minOccurs="0"/>

<xsd:element name="from" type="Types:TriComponentIdType" minOccurs="0"/>

<xsd:element name="fromTmpl" type="Templates:TciNonValueTemplate"

minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliCheckedAny\_m">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="addrValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="addressTmpl" type="Templates:TciValueTemplate"

minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliCheckedAny\_c">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="from" type="Types:TriComponentIdType"

minOccurs="0"/>

<xsd:element name="fromTmpl" type="Templates:TciNonValueTemplate"

minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliCheckMismatch\_m">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="addrValue" type="Values:Value" minOccurs="0"/>

<xsd:element name="addressTmpl" type="Templates:TciValueTemplate"

minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliCheckMismatch\_c">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="at" type="Types:TriPortIdType"/>

<xsd:element name="from" type="Types:TriComponentIdType"

minOccurs="0"/>

<xsd:element name="fromTmpl" type="Templates:TciNonValueTemplate"

minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliRnd">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="val" type="Values:FloatValue"/>

<xsd:element name="from" type="Values:FloatValue"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliEvaluate">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="name" type="Types:QualifiedName" />

<xsd:element name="evalResult" type="Values:Value" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliCCall">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="comp" type="Types:TriComponentIdType"/>

<xsd:element name="name" type="Types:TciBehaviourIdType"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliCCallTerminated">

<xsd:complexContent mixed="true">

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="verdict" type="Values:VerdictValue" />

<xsd:element name="reason" type="SimpleTypes:TString" minOccurs="0"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

<xsd:element name="returnValue" type="Values:Value" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliCtrlStartWithParameters">

<xsd:complexContent>

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="tliCtrlTerminatedWithResult">

<xsd:complexContent>

<xsd:extension base="Events:Event">

<xsd:sequence>

<xsd:element name="result" type="Values:Value" minOccurs="0"/>

<xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

</xsd:schema>

# B.6 TCI‑TL XML Schema for a Log

<?xml version="1.0" encoding="UTF‑8"?>

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"

targetNamespace="http://uri.etsi.org/ttcn‑3/tci/TLI\_v4\_10\_1.xsd"

xmlns:TLI="http://uri.etsi.org/ttcn‑3/tci/TLI\_v4\_10\_1.xsd"

xmlns:Events="http://uri.etsi.org/ttcn‑3/tci/Events\_v4\_10\_1.xsd"

elementFormDefault="qualified">

<xsd:import namespace="http://uri.etsi.org/ttcn‑3/tci/Events\_v4\_10\_1.xsd"

schemaLocation="Events\_v4\_10\_1.xsd"/>

<xsd:element name="logfile" type="TLI:LogModule"/>

<xsd:complexType name="LogModule">

<xsd:sequence>

<xsd:element name="header" type="TLI:Header"/>

<xsd:element name="body" type="TLI:Body"/>

<xsd:element name="trailer" type="TLI:Trailer" minOccurs="0"/>

</xsd:sequence>

</xsd:complexType>

<xsd:complexType name="Header">

<xsd:sequence>

<!‑‑ logging version ‑‑>

<xsd:element name="version" type="xsd:string"/>

<!‑‑ begin of the log ‑‑>

<xsd:element name="ts" type="xsd:long"/>

</xsd:sequence>

</xsd:complexType>

<xsd:complexType name="Trailer">

<xsd:choice>

<xsd:any namespace="##any" processContents="skip" minOccurs="0" maxOccurs="unbounded"/>

</xsd:choice>

</xsd:complexType>

<xsd:complexType name="Body">

<xsd:choice maxOccurs="unbounded">

<!‑‑ test cases operations ‑‑>

<xsd:element name="tliTcExecute" type="Events:tliTcExecute"/>

<xsd:element name="tliTcStart" type="Events:tliTcStart"/>

<xsd:element name="tliTcStop" type="Events:tliTcStop"/>

<xsd:element name="tliTcStarted" type="Events:tliTcStarted"/>

<xsd:element name="tliTcTerminated" type="Events:tliTcTerminated"/>

<!‑‑ control operations ‑‑>

<xsd:element name="tliCtrlStart" type="Events:tliCtrlStart"/>

<xsd:element name="tliCtrlStop" type="Events:tliCtrlStop"/>

<xsd:element name="tliCtrlTerminated" type="Events:tliCtrlTerminated"/>

<!‑‑ asynchronous communication ‑‑>

<xsd:element name="tliMSend\_m" type="Events:tliMSend\_m"/>

<xsd:element name="tliMSend\_c" type="Events:tliMSend\_c"/>

<xsd:element name="tliMSend\_m\_BC" type="Events:tliMSend\_m\_BC"/>

<xsd:element name="tliMSend\_c\_BC" type="Events:tliMSend\_c\_BC"/>

<xsd:element name="tliMSend\_m\_MC" type="Events:tliMSend\_m\_MC"/>

<xsd:element name="tliMSend\_c\_MC" type="Events:tliMSend\_c\_MC"/>

<xsd:element name="tliMDetected\_m" type="Events:tliMDetected\_m"/>

<xsd:element name="tliMDetected\_c" type="Events:tliMDetected\_c"/>

<xsd:element name="tliMMismatch\_m" type="Events:tliMMismatch\_m"/>

<xsd:element name="tliMMismatch\_c" type="Events:tliMMismatch\_c"/>

<xsd:element name="tliMReceive\_m" type="Events:tliMReceive\_m"/>

<xsd:element name="tliMReceive\_c" type="Events:tliMReceive\_c"/>

<!‑‑ synchronous communication ‑‑>

<xsd:element name="tliPrCall\_m" type="Events:tliPrCall\_m"/>

<xsd:element name="tliPrCall\_c" type="Events:tliPrCall\_c"/>

<xsd:element name="tliPrCall\_m\_BC" type="Events:tliPrCall\_m\_BC"/>

<xsd:element name="tliPrCall\_c\_BC" type="Events:tliPrCall\_c\_BC"/>

<xsd:element name="tliPrCall\_m\_MC" type="Events:tliPrCall\_m\_MC"/>

<xsd:element name="tliPrCall\_c\_MC" type="Events:tliPrCall\_c\_MC"/>

<xsd:element name="tliPrGetCallDetected\_m" type="Events:tliPrGetCallDetected\_m"/>

<xsd:element name="tliPrGetCallDetected\_c" type="Events:tliPrGetCallDetected\_c"/>

<xsd:element name="tliPrGetCallMismatch\_m" type="Events:tliPrGetCallMismatch\_m"/>

<xsd:element name="tliPrGetCallMismatch\_c" type="Events:tliPrGetCallMismatch\_c"/>

<xsd:element name="tliPrGetCall\_m" type="Events:tliPrGetCall\_m"/>

<xsd:element name="tliPrGetCall\_c" type="Events:tliPrGetCall\_c"/>

<xsd:element name="tliPrReply\_m" type="Events:tliPrReply\_m"/>

<xsd:element name="tliPrReply\_c" type="Events:tliPrReply\_c"/>

<xsd:element name="tliPrReply\_m\_BC" type="Events:tliPrReply\_m\_BC"/>

<xsd:element name="tliPrReply\_c\_BC" type="Events:tliPrReply\_c\_BC"/>

<xsd:element name="tliPrReply\_m\_MC" type="Events:tliPrReply\_m\_MC"/>

<xsd:element name="tliPrReply\_c\_MC" type="Events:tliPrReply\_c\_MC"/>

<xsd:element name="tliPrGetReplyDetected\_m" type="Events:tliPrGetReplyDetected\_m"/>

<xsd:element name="tliPrGetReplyDetected\_c" type="Events:tliPrGetReplyDetected\_c"/>

<xsd:element name="tliPrGetReplyMismatch\_m" type="Events:tliPrGetReplyMismatch\_m"/>

<xsd:element name="tliPrGetReplyMismatch\_c" type="Events:tliPrGetReplyMismatch\_c"/>

<xsd:element name="tliPrGetReply\_m" type="Events:tliPrGetReply\_m"/>

<xsd:element name="tliPrGetReply\_c" type="Events:tliPrGetReply\_c"/>

<xsd:element name="tliPrRaise\_m" type="Events:tliPrRaise\_m"/>

<xsd:element name="tliPrRaise\_c" type="Events:tliPrRaise\_c"/>

<xsd:element name="tliPrRaise\_m\_BC" type="Events:tliPrRaise\_m\_BC"/>

<xsd:element name="tliPrRaise\_c\_BC" type="Events:tliPrRaise\_c\_BC"/>

<xsd:element name="tliPrRaise\_m\_MC" type="Events:tliPrRaise\_m\_MC"/>

<xsd:element name="tliPrRaise\_c\_MC" type="Events:tliPrRaise\_c\_MC"/>

<xsd:element name="tliPrCatchDetected\_m" type="Events:tliPrCatchDetected\_m"/>

<xsd:element name="tliPrCatchDetected\_c" type="Events:tliPrCatchDetected\_c"/>

<xsd:element name="tliPrCatchMismatch\_m" type="Events:tliPrCatchMismatch\_m"/>

<xsd:element name="tliPrCatchMismatch\_c" type="Events:tliPrCatchMismatch\_c"/>

<xsd:element name="tliPrCatch\_m" type="Events:tliPrCatch\_m"/>

<xsd:element name="tliPrCatch\_c" type="Events:tliPrCatch\_c"/>

<xsd:element name="tliPrCatchTimeoutDetected"

type="Events:tliPrCatchTimeoutDetected "/>

<xsd:element name="tliPrCatchTimeout" type="Events:tliPrCatchTimeout"/>

<!‑‑ components ‑‑>

<xsd:element name="tliCCreate" type="Events:tliCCreate"/>

<xsd:element name="tliCStart" type="Events:tliCStart"/>

<xsd:element name="tliCRunning" type="Events:tliCRunning"/>

<xsd:element name="tliCAlive" type="Events:tliCAlive"/>

<xsd:element name="tliCStop" type="Events:tliCStop"/>

<xsd:element name="tliCKill" type="Events:tliCKill"/>

<xsd:element name="tliCDoneMismatch" type="Events:tliCDoneMismatch"/>

<xsd:element name="tliCDone" type="Events:tliCDone"/>

<xsd:element name="tliCKilledMismatch" type="Events:tliCKilledMismatch"/>

<xsd:element name="tliCKilled" type="Events:tliCKilled"/>

<xsd:element name="tliCTerminated" type="Events:tliCTerminated"/>

<!‑‑ ports ‑‑>

<xsd:element name="tliPConnect" type="Events:tliPConnect"/>

<xsd:element name="tliPDisconnect" type="Events:tliPDisconnect"/>

<xsd:element name="tliPMap" type="Events:tliPMap"/>

<xsd:element name="tliPMapParam" type="Events:tliPMapParam"/>

<xsd:element name="tliPUnmap" type="Events:tliPUnmap"/>

<xsd:element name="tliPUnmapParam" type="Events:tliPUnmapParam"/>

<xsd:element name="tliPClear" type="Events:tliPClear"/>

<xsd:element name="tliPStart" type="Events:tliPStart"/>

<xsd:element name="tliPStop" type="Events:tliPStop"/>

<xsd:element name="tliPHalt" type="Events:tliPHalt"/>

<!‑‑ codec ‑‑>

<xsd:element name="tliDecode" type="Events:tliDecode"/>

<xsd:element name="tliEncode" type="Events:tliEncode"/>

<!‑‑ timers ‑‑>

<xsd:element name="tliTTimeoutDetected" type="Events:tliTTimeoutDetected"/>

<xsd:element name="tliTTimeoutMismatch" type="Events:tliTTimeoutMismatch"/>

<xsd:element name="tliTTimeout" type="Events:tliTTimeout"/>

<xsd:element name="tliTStart" type="Events:tliTStart"/>

<xsd:element name="tliTStop" type="Events:tliTStop"/>

<xsd:element name="tliTRead" type="Events:tliTRead"/>

<xsd:element name="tliTRunning" type="Events:tliTRunning"/>

<!‑‑ scopes ‑‑>

<xsd:element name="tliSEnter" type="Events:tliSEnter"/>

<xsd:element name="tliSLeave" type="Events:tliSLeave"/>

<!‑‑ statements ‑‑>

<xsd:element name="tliVar" type="Events:tliVar"/>

<xsd:element name="tliModulePar" type="Events:tliModulePar"/>

<xsd:element name="tliGetVerdict" type="Events:tliGetVerdict"/>

<xsd:element name="tliSetVerdict" type="Events:tliSetVerdict"/>

<xsd:element name="tliLog" type="Events:tliLog"/>

<!‑‑ alt ‑‑>

<xsd:element name="tliAEnter" type="Events:tliAEnter"/>

<xsd:element name="tliALeave" type="Events:tliALeave"/>

<xsd:element name="tliADefaults" type="Events:tliADefaults"/>

<xsd:element name="tliAActivate" type="Events:tliAActivate"/>

<xsd:element name="tliADeactivate" type="Events:tliADeactivate"/>

<xsd:element name="tliANomatch" type="Events:tliANomatch"/>

<xsd:element name="tliARepeat" type="Events:tliARepeat"/>

<xsd:element name="tliAWait" type="Events:tliAWait"/>

<!‑‑ action ‑‑>

<xsd:element name="tliAction" type="Events:tliAction"/>

<!‑‑ match ‑‑>

<xsd:element name="tliMatch" type="Events:tliMatch"/>

<xsd:element name="tliMatchMismatch" type="Events:tliMatchMismatch"/>

<!‑‑ info ‑‑>

<xsd:element name="tliInfo" type="Events:tliInfo"/>

<!‑‑ check ‑‑>

<xsd:element name="tliMChecked\_m" type="Events:tliMChecked\_m"/>

<xsd:element name="tliMChecked\_c" type="Events:tliMChecked\_c"/>

<xsd:element name="tliPrGetCallChecked\_m" type="Events:tliPrGetCallChecked\_m"/>

<xsd:element name="tliPrGetCallChecked\_c" type="Events:tliPrGetCallChecked\_c"/>

<xsd:element name="tliPrGetReplyChecked\_m" type="Events:tliPrGetReplyChecked\_m"/>

<xsd:element name="tliPrGetReplyChecked\_c" type="Events:tliPrGetReplyChecked\_c"/>

<xsd:element name="tliPrCatchChecked\_m" type="Events:tliPrCatchChecked\_m"/>

<xsd:element name="tliPrCatchChecked\_c" type="Events:tliPrCatchChecked\_c"/>

<xsd:element name="tliCheckedAny\_m" type="Events:tliCheckedAny\_m"/>

<xsd:element name="tliCheckedAny\_c" type="Events:tliCheckedAny\_c"/>

<xsd:element name="tliCheckAnyMismatch\_m" type="Events:tliCheckAnyMismatch\_m"/>

<xsd:element name="tliCheckAnyMismatch\_c" type="Events:tliCheckAnyMismatch\_c"/>

<!-- rnd -->

<xsd:element name="tliRnd" type="Events:tliRnd"/>

<!-- evaluation of @lazy and @fuzzy variables -->

<xsd:element name="tliEvaluate" type="Events:tliEvaluate"/>

<!‑‑ components ‑‑>

<xsd:element name="tliCCall" type="Events:tliCCall"/>

<xsd:element name="tliCCallTerminated" type="Events:tliCCallTerminated"/>

<!-- module control function -->

<xsd:element name="tliCtrlStartWithParameters"   
 type="Events:tliCtrlStartWithParameters"/>

<xsd:element name="tliCtrlTerminatedWithResult"   
 type="Events:tliCtrlTerminatedWithResult"/>

</xsd:choice>

</xsd:complexType>

</xsd:schema>

Annex C (informative):  
Use scenarios

# C.0 Introduction

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

The scenarios are defined in terms of UML sequence diagrams. The sequence diagram shows the interactions between the TCI entities. The scenarios are explained and where applicable underpinned with a TTCN‑3 fragment corresponding to the scenario.

# C.1 Initialization, collecting information, logging

## C.1.1 Use scenario: initialization

### C.1.1.0 Scenario description

The scenario in figure C.1 shows the initialization phase for a test system when a TTCN‑3 module is to be selected for execution. At first, a root module has to be set with tciRootModule. The module parameters of the root module can be obtained with tciGetModuleParameters. Module parameter information can be used to ask the test system user for concrete values for each module parameter. The list of test cases available in the root module can be retrieved with tciGetTestCases. These test cases can be directly executed from the test management. Their parameters and their test system interface can be obtained with tciGetTestCaseParameters and tciGetTestCaseTSI, respectively.

### C.1.1.1 Sequence diagram



Figure C.: Use scenario - initialization

### C.1.1.2 TTCN‑3 fragment

The initialization is outside the scope of TTCN‑3.

## C.1.2 Use scenario: requesting module parameters

### C.1.2.0 Scenario description

The scenario in figure C.2 shows how a test component requests the actual value of a module parameter needed for the execution of its test behaviour. At first, the type of a module parameter is requested, then the value can be constructed by the TM and given to the TE.

### C.1.2.1 Sequence diagram



Figure C.: Use scenario - requesting module Pars

### C.1.2.2 TTCN‑3 fragment

module AModule {

…

modulepar {

integer AModulePar

}

…

function AFunction (…) … {

integer x;

…

x:= 2+AModulePar; // an expression with a module parameter

…

}

…

}

## C.1.3 Use scenario: logging

### C.1.3.0 Scenario description

The scenario in figure C.3 shows logging of information during the execution of a test behaviour by a test component. The message to be logged is propagated to the test logging.

### C.1.3.1 Sequence diagram



Figure C.: Use scenario - logging

### C.1.3.2 TTCN‑3 fragment

module AModule {

…

function AFunction (…) … {

…

log("AMessage");

…

}

…

}

# C.2 Execution of test cases and control

## C.2.1 Use scenario: execution of control

### C.2.1.0 Scenario description

The scenario in figure C.4 shows the sequence of operations to execute the control part of a TTCN‑3 module. The module containing the control part is selected first, then the control is started, then it is executed until the execution is terminated by TE.

### C.2.1.1 Sequence diagram



Figure C.: Use scenario - execution of control

### C.2.1.2 TTCN‑3 fragment

module AModule {

…

control {

…

}

…

}

## C.2.2 Use scenario: test case execution within control

### C.2.2.0 Scenario description

The scenario in figure C.5 shows how a test case is executed within the control part.

### C.2.2.1 Sequence diagram



Figure C.: Use scenario - test case execution within control

### C.2.2.2 TTCN‑3 fragment

module AModule {

…

testcase ATestCase(…)… {

… //the test case behaviour

}

…

control {

…

execute(ATestCase(…));

…

}

…

}

## C.2.3 Use scenario: direct test case execution

### C.2.3.0 Scenario description

The scenario in figure C.6 shows how a test case can be directly executed from the test management outside the control part. After selecting the TTCN‑3 module containing the test case to be executed, the start of the test case is requested. When the test case completes its execution, the test management is informed by the TE of the test case termination.

### C.2.3.1 Sequence diagram



Figure C.: Use scenario - direct test case execution

### C.2.3.2 TTCN‑3 fragment

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

## C.2.4 Use scenario: execute test case to TRI

### C.2.4.0 Scenario description

The scenario in figure C.7 shows how the TRI is informed about the execution of a test case so that it can set up and initialize system ports when needed. The execute test case request has to be issued before the test behaviour on the MTC of the current test case is started.

### C.2.4.1 Sequence diagram



Figure C.: Use scenario - execute test case to TRI

### C.2.4.2 TTCN‑3 fragment

module AModule {

…

testcase ATestCase(…)… {

… //the test case behaviour

}

…

control {

…

execute(ATestCase(…));

…

}

…

}

# C.3 Component handling

## C.3.1 Use scenario: local control component creation

### C.3.1.0 Scenario description

The scenario in figure C.8 demonstrates the creation of the control component on the same node where the user interface to the test management TCI‑TM resides. A control component is created whenever the control part of a TTCN‑3 module is executed. Whenever the test management TCI‑TM issues the start of the control part, a create test component request is sent to the TCI‑CH, which propagates it to the TE where the control component should be created. In this case it is the TE on the same node. The identifier for the control component is returned and given to the TCI‑TM. The identifier is then used to start the behaviour of the control part on the control component.

### C.3.1.1 Sequence diagram



Figure C.: Use scenario - local control component creation

### C.3.1.2 TTCN‑3 fragment

module AModule {

…

control {

…

}

…

}

## C.3.2 Use scenario: remote control component creation

### C.3.2.0 Scenario description

The scenario in figure C.9 demonstrates the creation of the control component on another node than where the user interface to the test management TCI‑TM resides. A control component is created whenever the control part of a TTCN‑3 module is executed. Whenever the test management TCI‑TM issues the start of the control part, a create test component request is sent to the TCI‑CH, which propagates it to the TE where the control component should be created. In this case it is the TE on another remote node. The identifier for the control component is returned and given to the TCI‑TM. The identifier is then used to start the behaviour of the control part on the control component.

### C.3.2.1 Sequence diagram



Figure C.: Use scenario - remote control component creation

### C.3.2.2 TTCN‑3 fragment

module AModule {

…

control {

…

}

…

}

## C.3.3 Use scenario: local MTC creation

### C.3.3.0 Scenario description

The scenario in figure C.10 demonstrates the local creation of the main test component. Local is meant for two cases:

1) on the same node where the user interface to the test management TCI‑TM resides (when a test case is started directly); or

2) on the same node where the control component resides (when a test case is executed from a control part).

A main test component is created whenever a test case is executed: a create test component request is sent to the TCI‑CH, which propagates it to the TE where the main test component should be created. In this case it is the TE on the same node. The identifier for the main test component is returned and given to the TCI‑TM. The identifier is then used to start the test case behaviour on the main test component (this is not shown here, but handled the same way as in the scenarios described in clauses C.2.2 and C.2.3).

### C.3.3.1 Sequence diagram



Figure C.: Use scenario - local MTC creation

### C.3.3.2 TTCN‑3 fragment

module AModule {

…

testcase ATestCase (…)runs on MTCType… {

… //the test case behaviour

}

…

}

## C.3.4 Use scenario: remote MTC creation

### C.3.4.0 Scenario description

The scenario in figure C.11 demonstrates the remote creation of the main test component. Remote is meant for two cases:

1) on another node than where the user interface to the test management TCI‑TM resides (when a test case is started directly); or

2) on another node than where the control component resides (when a test case is executed from a control part).

A main test component is created whenever a test case is executed: a create test component request is sent to the TCI‑CH, which propagates it to the TE where the main test component should be created. In this case it is the TE on another node. The identifier for the main test component is returned and given to the TCI‑TM. The identifier is then used to start the test case behaviour on the main test component (this is not shown here, but handled the same way as in the scenarios described in clauses C.2.2 and C.2.3).

### C.3.4.1 Sequence diagram



Figure C.: Use scenario - remote MTC creation

### C.3.4.2 TTCN‑3 fragment

module AModule {

…

testcase ATestCase(…)runs on MTCType … {

… //the test case behaviour

}

…

}

## C.3.5 Use scenario: component handling for test case execution within control

### C.3.5.0 Scenario description

The scenario in figure C.12 demonstrates the handling of components for the test case execution within a control part. When the control part is started, a control component is created and its component identifier returned to the test management. For each test case to be executed within the control part, a main test component is created and the component identifier returned to the control component. Afterwards, the test case behaviour is started on the main test component and the test management is informed about the start of the test case. When the main test component terminates, a request for the main test component termination together with the local verdict of the main test component is propagated to enable the derivation of the global test verdict and to enable the information about the test case termination.

### C.3.5.1 Sequence diagram



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

### C.3.5.2 TTCN‑3 fragment

module AModule {

…

testcase ATestCase(…)… {

… //the test case behaviour

}

…

control {

…

execute(ATestCase(…));

…

}

…

}

## C.3.6 Use scenario: component handling for direct test case execution

### C.3.6.0 Scenario description

The scenario in figure C.13 shows how test components are handled when a test case is executed directly, i.e. outside a control part. When a test case is started, the main test component is created and the test case behaviour started on this main test component at first. Whenever a parallel test component is used within a test case, it is handled the same: the parallel test component is started first: giving a test component create request to the TCI‑CH entity, which propagates the test component create to the TE in which the parallel test component is to be created. The identifier for the created parallel test component is returned. The identifier is then used to start the PTC behaviour for the start operation. When the PTC terminates its execution, a test component terminate request together with the local test verdict is issued to inform TCI‑CH about this termination. The same is done when the main test component terminates. In addition, the termination of the main test component leads to the overall termination of the test case.

### C.3.6.1 Sequence diagram



Figure C.: Use scenario: component handling for direct test case execution

### C.3.6.2 TTCN‑3 fragment

module AModule {

…

function APTCBehaviour(…) runs on APTCType {

… //the PTC behaviour

}

…

testcase ATestCase(…)… {

… //the test case behaviour

var APTCType PTC:= APTCType.create;

…

PTC.start(APTCBehaviour(…));

…

}

…

}

## C.3.7 Use scenario: propagation of map/connect

### C.3.7.0 Scenario description

The scenario in figure C.14 shows how ports are mapped. The request to map a port is propagated to the TE where the map is finally performed. The propagation of connect requests works analogously.

### C.3.7.1 Sequence diagram



Figure C.: Use scenario: propagation of map

### C.3.7.2 TTCN‑3 fragment

module AModule {

…

type port A { … }

type component CA { port A a }

type component CB { port A a }

…

testcase ATestCase(…)runs on CA system CB {

var CA ptc := CA.create;

… //the test case behaviour

map(ptc:a, System:a);

…

}

…

}

## C.3.8 Use scenario: propagation of unmap/disconnect

### C.3.8.0 Scenario description

The scenario in figure C.15 shows how ports are unmapped. The request to unmap a port is propagated to the TE where the unmap is finally performed. The propagation of disconnect requests works analogously.

### C.3.8.1 Sequence diagram



Figure C.: Use scenario - propagation of map

### C.3.8.2 TTCN‑3 fragment

module AModule {

…

type port A { … }

type component CA { port A a }

type component CB { port A a }

…

testcase ATestCase(…)runs on CA system CB {

var CA ptc := CA.create;

… //the test case behaviour

unmap(ptc:a,system:a);

…

}

…

}

# C.4 Termination of test cases and control

## C.4.1 Use scenario: stop a test case

### C.4.1.0 Scenario description

The scenario in figure C.16 shows how a test case is stopped from the test management during test case execution. Once the TM has received information about a started test case, a stop test case can be requested up until receiving the information that the test case has been terminated. Upon stopping a test case, all parallel test components will be stopped and the test system will be reset.

### C.4.1.1 Sequence diagram



Figure C.: Use scenario: stop a test case

### C.4.1.2 TTCN‑3 fragment

There is no TTCN‑3 code related to how the TM chooses to implement test case termination. This is outside the scope of TTCN‑3.

## C.4.2 Use scenario: stop control

### C.4.2.0 Scenario description

The scenario in figure C.17 shows how a control part is stopped from the test management during control part execution. A control part can be stopped in between starting the control and its termination. If the control part receives a stop test case request while a test case is executing, the executing test case is to be stopped. Furthermore, the test system is to be reset as described in figure C.16.

### C.4.2.1 Sequence diagram



Figure C.: Use scenario - stop control

### C.4.2.2 TTCN‑3 fragment

Stopping a control part from the test management is outside the scope of TTCN‑3 so that no TTCN‑3 fragment exists.

## C.4.3 Use scenario: termination of control after error

### C.4.3.0 Scenario description

The scenario in figure C.18 shows the handling of error situations during the execution of a control part when no test case is being executed. The test management is informed about the error situation and has then to terminate the execution of the control part explicitly. Upon termination of the control part, the test system will be reset.

### C.4.3.1 Sequence diagram



Figure C.: Use scenario - termination of control after error

### C.4.3.2 TTCN‑3 fragment

There is no TTCN‑3 fragment for this scenario since error situations are exceptional cases in a test system and not a TTCN‑3 concept as such. Rather, the TTCN‑3 semantics describes various potential error situations in a test system.

## C.4.4 Use scenario: termination of a test case after error

### C.4.4.0 Scenario description

The scenario in figure C.19 shows the handling of error situations during the direct execution of a test case. The test management is informed about the error situation. The TM has then to explicitly terminate test case execution. Upon stopping a test case, the parallel test components will be stopped and the test system is to be reset.

### C.4.4.1 Sequence diagram



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

### C.4.4.2 TTCN‑3 fragment

There is no TTCN‑3 fragment for this scenario since error situations are exceptional cases in a test system and not a TTCN‑3 concept as such. Rather, the TTCN‑3 semantics describes various potential error situations in a test system.

## C.4.5 Use scenario: reset

### C.4.5.0 Scenario description

The scenario in figure C.20 shows the reset of the test system. In that case all involved TEs together with their TRI System Adaptors (SA) and Platform Adaptors (PA) are reset.

### C.4.5.1 Sequence diagram



Figure C.: Use scenario - reset

### C.4.5.2 TTCN‑3 fragment

There is no TTCN‑3 fragment for this scenario since reset as required after error situations are exceptional cases in a test system and not a TTCN‑3 concept as such.

# C.5 Communication

## C.5.1 Use scenario: local intercomponent communication

### C.5.1.0 Scenario description

The scenario in figure C.21 shows the communication between test components (main test component or parallel test components), which reside on the same node. A communication request is given to the TCI‑CH, which then decide where to enqueue this communication template. In this case, the communication is done locally via the TE on the same node. The scenario shows a message‑based communication using the send operation ‑ the scenario is the same for call, reply, and raise operations.

### C.5.1.1 Sequence diagram



Figure C.: Use scenario - local intercomponent communication

### C.5.1.2 TTCN‑3 fragment

module AModule {

…

type port APortType message { … }

…

type component ATCType {

…

APortType APort;

…

}

…

template AType AMessageTemplate { … }

…

function APTCBehaviour(…) runs on APTCType {

… //the PTC behaviour

…

}

…

testcase ATestCase(…) runs on ATCType… {

… //the test case behaviour

var ATCType PTC1:= ATCType.create;

connect(PTC1:APort,mtc:APort);

…

PTC1.start(APTCBehaviour(…));

APort.send(AMessageTemplate); //sending data to a test component

…

}

…

}

## C.5.2 Use scenario: internode communication between test components

### C.5.2.0 Scenario description

The scenario in figure C.22 shows the communication between test components (main test component or parallel test components), which reside on different nodes. A communication request is given to the TCI‑CH, which then decides where to enqueue this communication template. In this case, the communication is done remotely via the TE on another node. The scenario shows a message based communication using the send operation ‑ the scenario is the same for call, reply, and raise operations.

### C.5.2.1 Sequence diagram



Figure C.: Use scenario - internode communication between test components

### C.5.2.2 TTCN‑3 fragment

module AModule {

…

type port APortType message { … }

…

type component ATCType {

…

APortType APort;

…

}

…

template AType AMessageTemplate { … }

…

function APTCBehaviour(…) runs on APTCType {

… //the PTC behaviour

…

}

…

testcase ATestCase(…) runs on ATCType… {

… //the test case behaviour

var ATCType PTC1:= ATCType.create;

connect(PTC1:APort,mtc:APort);

…

PTC1.start(APTCBehaviour(…));

APort.send(AMessageTemplate); //sending data to a test component

…

}

…

}

## C.5.3 Use scenario: encoding

### C.5.3.0 Scenario description

The scenario in figure C.23 shows the encoding of data, which is sent to the SUT. The encoded data is received from the coding/decoding entity via the TCI‑CD. The encoded value is sent to the SUT via the TRI‑SA. The scenario is the same for the call, the reply, and the raise operations.

### C.5.3.1 Sequence diagram



Figure C.: Use scenario - encoding

### C.5.3.2 TTCN‑3 fragment

module AModule {

…

type port APortType message { … }

…

type component APTCType {

…

APortType APort;

…

}

…

template AType AMessageTemplate { … }

…

testcase ATestCase(…) runs on APTCType system APTCType {

… //the test case behaviour

map(mtc:APort,system:APort);

…

APort.send(AMessageTemplate); //sending data to the SUT

…

}

…

} with { encoding "…" }

## C.5.4 Use scenario: decoding

### C.5.4.0 Scenario description

The scenario in figure C.24 shows the decoding of data, which is received from the SUT via the TRI‑SA. The decoded data is received from the coding/decoding entity via the TCI‑CD. The scenario is the same for the receive, the getcall, the getreply, the catch, and the check operations.

### C.5.4.1 Sequence diagram



Figure C.: Use scenario - decoding

### C.5.4.2 TTCN‑3 fragment

module AModule {

…

type port APortType message { … }

…

type component APTCType {

…

APortType APort;

…

}

…

template AType AMessageTemplate { … }

…

testcase ATestCase(…) runs on APTCType system APTCType {

… //the test case behaviour

map(mtc:APort,system:APort);

…

APort.receive(AMessageTemplate); //receiving data from the SUT

…

}

…

} with { encoding "…" }

Annex D (informative):  
Bibliography

* INTOOL CGI/NPL038 (V2.2): "Generic Compiler/Interpreter interface; GCI Interface Specification" Infrastructural Tools, December 1996.
* ISO/IEC 9646‑3 (1998): "Information technology - Open Systems Interconnection - Conformance testing methodology and framework - Part 3: The Tree and Tabular combined Notation (TTCN)".

# History

|  |  |  |
| --- | --- | --- |
| **Document history** | | |
| V1.1.1 | July 2003 | Publication |
| V3.1.1 | June 2005 | Publication |
| V3.2.1 | February 2007 | Publication |
| V3.3.1 | April 2008 | Publication |
| V3.4.1 | September 2008 | Publication |
| V4.1.1 | June 2009 | Publication |
| V4.2.1 | July 2010 | Publication |
| V4.3.1 | June 2011 | Publication |
| V4.4.1 | April 2012 | Publication |
| V4.5.1 | April 2013 | Publication |
| V4.6.1 | June 2014 | Publication |
| V4.7.1 | June 2015 | Publication |
| V4.8.1 | July 2016 | Publication |
| V4.9.1 | May 2017 | Publication |
| V4.10.1 | May 2018 | Publication |
| V4.11.1 | April 2019 | Publication |
| V4.12.1 | February 2020 | Membership Approval Procedure MV 20200428: 2020-02-28 to 2020-04-28 |
| V4.12.1 | May 2020 | Publication |