ETSI ES 201 873-1 V4.12.1 (2020-05)

Methods for Testing and Specification (MTS);

The Testing and Test Control Notation version 3;

Part 1: TTCN‑3 Core Language

**ETSI Standard**

Reference

RES/MTS-201873-1V4.12.1

Keywords

language, methodology, 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 13

Foreword 13

Modal verbs terminology 13

1 Scope 14

2 References 14

2.1 Normative references 14

2.2 Informative references 15

3 Definition of terms, symbols and abbreviations 16

3.1 Terms 16

3.2 Symbols 22

3.3 Abbreviations 22

4 Introduction 23

4.0 General 23

4.1 The core language and presentation formats 24

4.2 Unanimity of the specification 25

4.3 Conformance 25

5 Basic language elements 25

5.0 General 25

5.1 Identifiers and keywords 26

5.2 Scope rules 27

5.2.0 General 27

5.2.1 Scope of formal parameters 29

5.2.2 Uniqueness of identifiers 29

5.3 Ordering of language elements 30

5.4 Parameterization 30

5.4.0 General 30

5.4.1 Formal parameters 31

5.4.1.0 General 31

5.4.1.1 Formal parameters of kind value 31

5.4.1.2 Formal parameters of kind template 34

5.4.2 Actual parameters 36

5.5 Cyclic Definitions 41

6 Types and values 42

6.0 General 42

6.1 Basic types and values 43

6.1.0 Simple basic types and values 43

6.1.1 Basic string types and values 44

6.1.1.0 General 44

6.1.1.1 Accessing individual string elements 46

6.1.2 Subtyping of basic types 47

6.1.2.0 General 47

6.1.2.1 Lists of templates 47

6.1.2.2 Lists of types 47

6.1.2.3 Ranges 48

6.1.2.4 String length restrictions 49

6.1.2.5 Pattern subtyping of character string types 49

6.1.2.6 Mixing subtyping mechanisms 49

6.1.2.6.1 Mixing patterns, lists and ranges 49

6.1.2.6.2 Using length restriction with other constraints 50

6.2 Structured types and values 50

6.2.0 General 50

6.2.1 Record type and values 52

6.2.1.0 General 52

6.2.1.1 Referencing fields of a record type 55

6.2.1.2 Optional elements in a record 56

6.2.1.3 Nested type definitions for field types 56

6.2.2 Set type and values 57

6.2.2.0 General 57

6.2.2.1 Referencing fields of a set type 57

6.2.2.2 Optional elements in a set 57

6.2.2.3 Nested type definition for field types 57

6.2.3 Records and sets of single types 57

6.2.3.0 General 57

6.2.3.1 Nested type definitions 60

6.2.3.2 Referencing elements of record of and set of types 60

6.2.4 Enumerated type and values 61

6.2.5 Unions 63

6.2.5.0 General 63

6.2.5.1 Referencing fields of a union type 64

6.2.5.2 Option and union 65

6.2.5.3 Nested type definition for field types 65

6.2.6 The anytype 66

6.2.7 Arrays 66

6.2.8 The default type 68

6.2.9 Communication port types 68

6.2.10 Component types 70

6.2.10.1 Component type definition 70

6.2.10.2 Reuse of component types 71

6.2.11 Component references 73

6.2.12 Addressing entities inside the SUT 75

6.2.13 Subtyping of structured types 77

6.2.13.0 General 77

6.2.13.1 Length subtyping of record ofs and set ofs 77

6.2.13.2 List subtyping of structured types and anytype 78

6.2.13.3 Subtyping of the iterated type of record ofs and set ofs 81

6.2.13.4 Mixing subtyping mechanisms 82

6.2.14 The timer type 82

6.2.15 Map types 82

6.2.15.0 General 82

6.2.15.1 Map Type Definition 82

6.2.15.2 Indexed Assignment Notation 83

6.2.15.3 Unmapping Keys 83

6.2.15.4 Index Notation 84

6.2.15.5 Accessing the Keys of a Map 84

6.2.15.6 Accessing the Values of a Map 85

6.2.15.7 Referencing of Elements of a Map 85

6.2.15.8 Nested type definitions 85

6.2.16 The open type 85

6.3 Type compatibility 86

6.3.0 General 86

6.3.1 Compatibility of non-structured types 86

6.3.2 Compatibility of structured types 88

6.3.2.0 General 88

6.3.2.1 Compatibility of enumerated types 88

6.3.2.2 Compatibility of record and record of types 89

6.3.2.3 Compatibility of set and set of types 90

6.3.2.4 Compatibility of union types 91

6.3.2.5 Compatibility of anytype types 91

6.3.2.6 Compatibility between sub-structures 92

6.3.2.7 Compatibility of the open type 92

6.3.3 Compatibility of component types 93

6.3.4 Type compatibility of communication and connection operations 93

6.3.5 Type conversion 94

6.3.6 Type compatibility of port types 94

6.3.7 Type compatibility of timer types 94

6.3.8 Type Compatibility of Map Types 94

6.4 Type synonym 94

7 Expressions 94

7.0 General 94

7.1 Operators 95

7.1.0 General 95

7.1.1 Arithmetic operators 97

7.1.2 List operator 97

7.1.3 Relational operators 98

7.1.4 Logical operators 101

7.1.5 Bitwise operators 101

7.1.6 Shift operators 102

7.1.7 Rotate operators 102

7.1.8 Presence checking operators 103

7.1.8.0 General 103

7.1.8.1 The ispresent operator 104

7.1.8.2 The ischosen operator 105

7.1.8.3 The isvalue operator 106

7.1.8.4 The isbound operator 108

7.2 Field references and list elements 109

7.3 Decoded field reference 109

8 Modules 110

8.0 General 110

8.1 Definition of a module 110

8.2 Module definitions part 111

8.2.0 General 111

8.2.1 Module parameters 112

8.2.2 Groups of definitions 113

8.2.3 Importing from modules 114

8.2.3.0 General 114

8.2.3.1 General format of import 114

8.2.3.2 Importing single definitions 121

8.2.3.3 Importing groups 122

8.2.3.4 Importing definitions of the same kind 123

8.2.3.5 Importing all definitions of a module 123

8.2.3.6 Import definitions from other TTCN‑3 editions and from non-TTCN‑3 modules 124

8.2.3.7 Importing of import statements from TTCN-3 modules 126

8.2.3.8 Compatibility of language specifications in imports 127

8.2.4 Definition of friend modules 128

8.2.5 Visibility of definitions 128

8.3 Module control part 130

9 Port types, component types and test configurations 131

9.0 General 131

9.1 Communication ports 131

9.2 Test system interface 134

10 Declaring constants 136

11 Declaring variables 136

11.0 General 136

11.1 Value variables 137

11.2 Template variables 138

12 Declaring timers 139

13 Declaring messages 140

14 Declaring procedure signatures 141

15 Declaring templates 142

15.0 General 142

15.1 Declaring message templates 143

15.2 Declaring signature templates 145

15.3 Global and local templates 146

15.4 In-line Templates 147

15.5 Modified templates 148

15.6 Referencing elements of templates or template fields 152

15.6.0 General 152

15.6.1 Referencing individual string elements 152

15.6.2 Referencing **record** and **set** fields 152

15.6.3 Referencing **record** **of** and **set** **of** elements 153

15.6.4 Referencing signature parameters 157

15.6.5 Referencing **union** alternatives 158

15.7 Template matching mechanisms 159

15.7.0 General 159

15.7.1 Specific values 160

15.7.2 Special symbols that can be used instead of values 161

15.7.3 Special symbols that can be used inside values 162

15.7.4 Special symbols which describe attributes of values 162

15.8 Template Restrictions 163

15.9 Match Operation 165

15.10 Valueof Operation 167

15.11 Concatenating templates of string and list types 167

16 Functions, altsteps and testcases 170

16.0 General 170

16.1 Functions 170

16.1.0 General 170

16.1.1 Invoking functions 173

16.1.2 Predefined functions 173

16.1.3 External functions 176

16.1.4 Invoking functions from specific places 176

16.1.5 Explicit control functions 178

16.2 Altsteps 178

16.2.0 General 178

16.2.1 Invoking altsteps 180

16.3 Test cases 181

17 Void 182

18 Overview of program statements and operations 182

19 Basic program statements 185

19.0 General 185

19.1 Assignments 185

19.2 The If-else statement 187

19.3 The Select statements 187

19.3.1 The Select case statement 187

19.3.2 The Select union statement 189

19.4 The For statement 190

19.5 The While statement 190

19.6 The Do-while statement 191

19.7 The Label statement 191

19.8 The Goto statement 192

19.9 The Stop execution statement 193

19.10 The Return statement 193

19.11 The Log statement 194

19.12 The Break statement 196

19.13 The Continue statement 196

19.14 Statement block 197

20 Statement and operations for alternative behaviours 197

20.0 General 197

20.1 The snapshot mechanism 198

20.2 The Alt statement 198

20.3 The Repeat statement 202

20.4 The Interleave statement 203

20.5 Default Handling 205

20.5.0 General 205

20.5.1 The default mechanism 206

20.5.2 The Activate operation 206

20.5.3 The Deactivate operation 207

21 Configuration Operations 208

21.0 General 208

21.1 Connection Operations 209

21.1.0 General 209

21.1.1 The Connect and Map operations 210

21.1.2 The Disconnect and Unmap operations 212

21.2 Test case operations 213

21.2.0 General 213

21.2.1 Test case stop operation 214

21.3 Test Component Operations 214

21.3.0 General 214

21.3.1 The Create operation 214

21.3.2 The Start test component operation 215

21.3.3 The Stop test behaviour operation 217

21.3.4 The Kill test component operation 218

21.3.5 The Alive operation 218

21.3.6 The Running operation 219

21.3.7 The Done operation 221

21.3.8 The Killed operation 223

21.3.9 Summary of the use of any and all with components 225

21.3.10 The Call test component behaviour operation 225

22 Communication operations 227

22.0 General 227

22.1 The communication mechanisms 227

22.1.0 General 227

22.1.1 Principles of message-based communication 227

22.1.2 Principles of procedure-based communication 228

22.1.3 Principles of unicast, multicast and broadcast communication 228

22.1.4 General format of communication operations 229

22.1.4.0 General 229

22.1.4.1 General format of the sending operations 229

22.1.4.2 General format of the receiving operations 230

22.2 Message-based communication 231

22.2.0 General 231

22.2.1 The Send operation 231

22.2.2 The Receive operation 232

22.2.3 The Trigger operation 236

22.3 Procedure-based communication 239

22.3.0 General 239

22.3.1 The Call operation 239

22.3.2 The Getcall operation 243

22.3.3 The Reply operation 246

22.3.4 The Getreply operation 247

22.3.5 The Raise operation 250

22.3.6 The Catch operation 251

22.4 The Check operation 255

22.5 Controlling communication ports 257

22.5.0 General 257

22.5.1 The Clear port operation 257

22.5.2 The Start port operation 258

22.5.3 The Stop port operation 258

22.5.4 The Halt port operation 259

22.5.5 The Checkstate port operation 259

22.6 Use of any and all with ports 261

23 Timer operations 261

23.0 General 261

23.1 The timer mechanism 262

23.2 The Start timer operation 262

23.3 The Stop timer operation 263

23.4 The Read timer operation 263

23.5 The Running timer operation 264

23.6 The Timeout operation 265

23.7 Summary of use of any and all with timers 266

24 Test verdict operations 266

24.0 General 266

24.1 The Verdict mechanism 266

24.2 The Setverdict operation 267

24.3 The Getverdict operation 268

25 External actions 269

26 Module control 269

26.0 General 269

26.1 The Execute statement 270

26.2 Test suite execution 271

27 Specifying attributes 273

27.0 General 273

27.1 The Attribute mechanism 273

27.1.0 General 273

27.1.1 Scope of attributes 274

27.1.2 Overwriting rules for attributes 275

27.1.2.0 General 275

27.1.2.1 Additional default overwriting rules for variant attributes 277

27.1.2.2 Overwriting rules for multiple encoding 278

27.1.3 Changing attributes of imported language elements 278

27.2 The With statement 279

27.3 Display attributes 280

27.4 Encoding attributes 281

27.5 Variant attributes 282

27.6 Extension attributes 284

27.7 Optional attributes 284

27.8 Retrieving attribute values 286

27.9 Dynamic configuration of encoding used by ports 287

Annex A (normative): BNF and static semantics 289

A.1 TTCN‑3 BNF 289

A.1.0 General 289

A.1.1 Conventions for the syntax description 289

A.1.2 Statement terminator symbols 289

A.1.3 Identifiers 289

A.1.4 Comments 290

A.1.5 TTCN‑3 terminals 290

A.1.5.0 General 290

A.1.5.1 Use of whitespaces and newlines 292

A.1.6 TTCN-3 syntax BNF productions 293

A.1.6.0 TTCN-3 module 293

A.1.6.1 Module definitions part 293

A.1.6.1.0 General 293

A.1.6.1.1 Typedef definitions 293

A.1.6.1.2 Constant definitions 295

A.1.6.1.3 Template definitions 295

A.1.6.1.4 Function definitions 297

A.1.6.1.5 Signature definitions 298

A.1.6.1.6 Testcase definitions 298

A.1.6.1.7 Altstep definitions 298

A.1.6.1.8 Import definitions 298

A.1.6.1.9 Group definitions 299

A.1.6.1.10 External function definitions 299

A.1.6.1.11 Void 299

A.1.6.1.12 Module parameter definitions 300

A.1.6.1.13 Friend module definitions 300

A.1.6.2 Module control function 300

A.1.6.3 Local definitions 300

A.1.6.3.1 Variable instantiation 300

A.1.6.3.2 Timer instantiation 300

A.1.6.4 Operations 300

A.1.6.4.1 Component operations 300

A.1.6.4.2 Port operations 301

A.1.6.4.3 Timer operations 303

A.1.6.4.4 Testcase operation 303

A.1.6.5 Type 303

A.1.6.6 Value 304

A.1.6.7 Parameterization 305

A.1.6.8 Statements 305

A.1.6.8.1 With statement 305

A.1.6.8.2 Behaviour statements 306

A.1.6.8.3 Basic statements 307

A.1.6.9 Miscellaneous productions 309

Annex B (normative): Matching values 310

B.1 Template matching mechanisms 310

B.1.0 General 310

B.1.1 Matching specific values 310

B.1.2 Matching mechanisms instead of values 310

B.1.2.0 General 310

B.1.2.1 Template list 310

B.1.2.2 Complemented template list 311

B.1.2.3 Any value 312

B.1.2.4 Any value or none 313

B.1.2.5 Value range 314

B.1.2.6 SuperSet 314

B.1.2.7 SubSet 315

B.1.2.8 Omitting optional fields 317

B.1.2.9 Matching decoded content 317

B.1.2.10 Matching enumerated value with value list 319

B.1.3 Matching mechanisms inside values 319

B.1.3.0 General 319

B.1.3.1 Any element 319

B.1.3.1.0 General 319

B.1.3.1.1 Using single character wildcards 319

B.1.3.2 Any number of elements or no element 320

B.1.3.2.0 General 320

B.1.3.2.1 Using multiple character wildcards 320

B.1.3.3 Permutation 320

B.1.4 Matching attributes of values 322

B.1.4.0 General 322

B.1.4.1 Length restrictions 322

B.1.4.2 The IfPresent indicator 323

B.1.5 Matching character pattern 324

B.1.5.0 General 324

B.1.5.1 Set expression 326

B.1.5.2 Reference expression 326

B.1.5.3 Match expression n times 328

B.1.5.4 Match a referenced character set 328

B.1.5.5 Type compatibility rules for patterns 329

B.1.5.6 Case insensitive pattern matching 329

Annex C (normative): Predefined TTCN‑3 functions 330

C.0 General exception handling procedures 330

C.1 Conversion functions 330

C.1.1 Integer to character 330

C.1.2 Integer to universal character 330

C.1.3 Integer to bitstring 330

C.1.4 Integer to enumerated 331

C.1.5 Integer to hexstring 331

C.1.6 Integer to octetstring 331

C.1.7 Integer to charstring 332

C.1.8 Integer to float 332

C.1.9 Float to integer 332

C.1.10 Character to integer 332

C.1.11 Character to octetstring 332

C.1.12 Universal character to integer 333

C.1.13 Bitstring to integer 333

C.1.14 Bitstring to hexstring 333

C.1.15 Bitstring to octetstring 333

C.1.16 Bitstring to charstring 334

C.1.17 Hexstring to integer 334

C.1.18 Hexstring to bitstring 334

C.1.19 Hexstring to octetstring 335

C.1.20 Hexstring to charstring 335

C.1.21 Octetstring to integer 335

C.1.22 Octetstring to bitstring 335

C.1.23 Octetstring to hexstring 336

C.1.24 Octetstring to character string 336

C.1.25 Octetstring to character string, version II 336

C.1.26 Charstring to integer 337

C.1.27 Character string to hexstring 337

C.1.28 Character string to octetstring 337

C.1.29 Character string to float 338

C.1.30 Enumerated to integer 338

C.1.31 Octetstring to universal character string 339

C.1.32 Universal character string to octetstring 339

C.1.33 Value or template to universal charstring 340

C.2 Length/size functions 341

C.2.1 Length of strings and lists 341

C.2.2 Number of elements in a structured value 342

C.3 Presence checking functions 343

C.3.1 Void 343

C.3.2 Void 343

C.3.3 Void 343

C.3.4 Void 343

C.3.5 Matching mechanism detection 343

C.4 String/list handling functions 344

C.4.1 The Regexp function 344

C.4.2 The Substring function 346

C.4.3 The Replace function 347

C.5 Codec functions 348

C.5.1 The encoding function 348

C.5.2 The decoding function 348

C.5.3 The encoding to universal charstring function 348

C.5.4 The decoding from universal charstring function 349

C.5.5 The encoding to octetstring function 351

C.5.6 The decoding from octetstring function 351

C.5.7 Retrieving the type of string encoding 352

C.5.8 Removing BOMs of UCS encoding schemes 352

C.6 Other functions 353

C.6.1 The random number generator function 353

C.6.2 The testcasename function 353

C.6.3 The hostId function 354

Annex D (normative): Preprocessing macros 355

D.0 General 355

D.1 Preprocessing macro \_\_MODULE\_\_ 355

D.2 Preprocessing macro \_\_FILE\_\_ 355

D.3 Preprocessing macro \_\_BFILE\_\_ 355

D.4 Preprocessing macro \_\_LINE\_\_ 355

D.5 Preprocessing macro \_\_SCOPE\_\_ 356

Annex E (informative): Library of Useful Types 358

E.1 Limitations 358

E.2 Useful TTCN‑3 types 358

E.2.1 Useful simple basic types 358

E.2.1.0 Signed and unsigned single byte integers 358

E.2.1.1 Signed and unsigned short integers 358

E.2.1.2 Signed and unsigned long integers 359

E.2.1.3 Signed and unsigned longlong integers 359

E.2.1.4 IEEE 754 floats 359

E.2.2 Useful character string types 360

E.2.2.0 UTF-8 character string "utf8string" 360

E.2.2.1 BMP character string "bmpstring" 360

E.2.2.2 UTF-16 character string "utf16string" 360

E.2.2.3 ISO/IEC 10646 character string "iso8859string" 360

E.2.2.4 Status values for TTCN-3 objects 361

E.2.2.5 Template kinds of TTCN-3 objects 361

E.2.3 Useful structured types 361

E.2.3.0 Fixed-point decimal literal 361

E.2.4 Useful atomic string types 362

E.2.4.1 Single Recommendation ITU‑T T.50 character type 362

E.2.4.2 Single universal character type 362

E.2.4.3 Single bit type 362

E.2.4.4 Single hex type 362

E.2.4.5 Single octet type 362

Annex F (informative): Operations on TTCN‑3 active objects 363

F.0 General 363

F.1 Test components 363

F.1.1 Test component references 363

F.1.2 Dynamic behaviour of PTCs 364

F.1.3 Dynamic behaviour of the MTC 366

F.2 Timers 367

F.3 Ports 367

F.3.0 General 367

F.3.1 Configuration Operations 367

F.3.2 Port Controlling Operations 368

F.3.3 Communication Operations 369

Annex G (informative): Deprecated language features 370

G.1 Group style definition of module parameters 370

G.2 Void 370

G.3 Using **all** in port type definitions 370

G.4 sizeof for length of lists 370

G.5 Void 370

G.6 Mixed ports 370

G.7 Void 370

G.8 Void 371

G.9 Void 371

G.10 Void 371

G.11 Void 371

G.12 Void 371

G.13 Assignment of less restrictive templates to more restrictive templates 371

G.14 Mixing case and case else branches in select statements 371

G.15 Partially initialized global and local templates 372

Annex H (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 1 of a multi-part deliverable covering the Testing and Test Control Notation version 3, as identified below:

**Part 1: "TTCN‑3 Core Language";**

Part 2: "TTCN‑3 Tabular presentation Format (TFT)";

Part 3: "TTCN‑3 Graphical presentation Format (GFT)";

Part 4: "TTCN‑3 Operational Semantics";

Part 5: "TTCN‑3 Runtime Interface (TRI)";

Part 6: "TTCN‑3 Control Interface (TCI)";

Part 7: "Using ASN.1 with TTCN‑3";

Part 8: "The IDL to TTCN-3 Mapping";

Part 9: "Using XML schema with TTCN-3";

Part 10: "TTCN-3 Documentation Comment Specification";

Part 11: "Using JSON with TTCN-3".

NOTE: Part 2 of this multi-part deliverable is in status "historical" and is not maintained.

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

TTCN‑3 is intended to be used for the specification of test suites which are independent of test methods, layers and protocols. In addition to the textual format defined in the present document, while GFT (ETSI ES 201 873‑3 [i.2]) defines a graphical presentation format for TTCN‑3. The specification of these formats is outside the scope of the present document.

While the design of TTCN‑3 has taken the eventual implementation of TTCN‑3 translators and compilers into consideration the means of realization of Executable Test Suites (ETS) from Abstract Test Suites (ATS) is outside the scope of the present document.

# 2 References

## 2.1 Normative references

References are either specific (identified by date of publication and/or edition number or version number) or non‑specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

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

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

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

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

[2] ISO/IEC 10646 (2017): "Information technology -- Universal Coded Character Set (UCS)".

[3] Recommendation ITU-T X.292: "OSI conformance testing methodology and framework for protocol Recommendations for ITU-T applications - The Tree and Tabular Combined Notation (TTCN)".

NOTE: The corresponding ISO/IEC standard is ISO/IEC 9646-3: "Information technology -- Open Systems Interconnection -- Conformance testing methodology and framework -- Part 3: The Tree and Tabular Combined Notation (TTCN)".

[4] Recommendation ITU-T T.50: "International Reference Alphabet (IRA) (Formerly International Alphabet No. 5 or IA5) - Information technology - 7-bit coded character set for information interchange".

NOTE: The corresponding ISO/IEC standard is ISO/IEC 646: "Information technology -- ISO 7-bit coded character set for information interchange".

[5] Recommendation ITU-T X.290: "OSI conformance testing methodology and framework for protocol Recommendations for ITU-T applications - General concepts".

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

[6] IEEE 754™: "IEEE Standard for Floating-Point Arithmetic".

## 2.2 Informative references

References are either specific (identified by date of publication and/or edition number or version number) or non‑specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

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

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

[i.1] Void.

[i.2] ETSI ES 201 873-3: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 3: TTCN-3 Graphical presentation Format (GFT)".

[i.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)".

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

[i.5] ETSI ES 201 873-7: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 7: Using ASN.1 with TTCN-3".

[i.6] ETSI ES 201 873-8: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 8: The IDL to TTCN-3 Mapping".

[i.7] ETSI ES 201 873-9: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 9: Using XML schema with TTCN-3".

[i.8] ETSI ES 201 873-10: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 10: TTCN-3 Documentation Comment Specification".

[i.9] Void.

[i.10] Object Management Group (OMG) (2001): "The Common Object Request Broker: Architecture and Specification - IDL Syntax and Semantics". Version 2.6, FORMAL/01-12-01.

[i.11] ETSI ES 202 781: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; TTCN-3 Language Extensions: Configuration and Deployment Support".

[i.12] ETSI ES 202 784: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; TTCN-3 Language Extensions: Advanced Parameterization".

[i.13] ETSI ES 202 785: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; TTCN-3 Language Extensions: Behaviour Types".

[i.14] ETSI ES 202 782: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; TTCN-3 Language Extensions: TTCN-3 Performance and Real Time Testing".

[i.15] Void.

[i.16] Void.

[i.17] ETSI ES 201 873-1 (V1.1.2): "Methods for Testing and Specification (MTS); The Tree and Tabular Combined Notation version 3; Part 1: TTCN-3 Core Language", 2001.

[i.18] ETSI ES 201 873-1 (V2.2.1): "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language", 2003.

[i.19] ETSI ES 201 873-1 (V3.1.1): "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language", 2005.

[i.20] ETSI ES 201 873-1 (V3.2.1): "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language", 2007.

[i.21] ETSI ES 201 873-1 (V3.3.2): "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language", 2008.

[i.22] ETSI ES 201 873-1 (V3.4.1): "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language", 2008.

[i.23] ETSI ES 201 873-1 (V4.1.1): "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language", 2009.

[i.24] ETSI ES 201 873-1 (V4.2.1): "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language", 2010.

[i.25] ETSI ES 201 873-1 (V4.3.1): "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language", 2011.

[i.26] ETSI ES 201 873-1 (V4.4.1): "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language", 2012.

[i.27] ETSI ES 201 873-1 (V4.5.1): "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language", 2013.

[i.28] ETSI ES 201 873-1 (V4.6.1): "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language", 2014.

[i.29] ETSI ES 201 873-1 (V4.7.1): "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language", 2015.

[i.30] ETSI ES 201 873-1 (V4.8.1): "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language", 2016.

[i.31] ETSI ES 201 873-1 (V4.9.1): "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language", 2017.

# 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 [5], Recommendation ITU‑T X.292 [3] and the following apply:

**actual parameter:** value, expression, template or name reference (identifier) to be passed as parameter to the invoked entity (function, test case, altstep, etc.) as defined at the place of invoking

**assignment notation:** notation that can be used for record, set, record of and set of values, where the fields or the elements to which a value is assigned are identified explicitly within a pair of curly brackets ("{" and "}") by the field names or the positions of the elements

**basic types:** set of predefined TTCN‑3 types described in clauses 6.1.0 and 6.1.1 of the present document

NOTE: Basic types are referenced by their names.

**behaviour definition:** dynamic test behaviour, which is either a testcase, a function, or an altstep definition

**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 or procedure-based.

**compatible type:** TTCN‑3 is not strongly typed but the language does require type compatibility

NOTE: Variables, constants, templates, etc. have compatible types if conditions in clause 6.2.15 are met.

**completely initialized:** value or template is completely initialized if it is not uninitialized and, if its type is a structured type, all its required parts are completely initialized

NOTE 1: Additionally, templates are completely initialized if they are assigned a matching mechanism all parts of which are completely initialized. If a value or template is completely initialized, it fulfils the requirement of being "at least partially initialized".

NOTE 2: A value or template of a simple, **component** or **default** type is completely initialized if anything but the unchanged symbol "-" has been assigned to it.   
A value or template of a **union** or **anytype** type is completely initialized if one of its variants has been completely initialized.  
A value or template of a **record** or **set** type with only optional fields and the **optional** "**implicit omit**" attribute attached, is completely initialized if the value "{}" is assigned, as all fields are implicitly set to omit.   
A value or template of a **record** or **set** type with no fields is completely initialized with assignment of the value "{}".  
A value or template of a **record of**, **set of** or array type is completely initialized if at least the first n elements are completely initialized, where n is the minimal length imposed by the type length restriction or array definition. Thus in case of n equals 0, the assignment of the value "{}" also completely initializes such a **record of**, **set of** or array.

**component constant:** constant defined in a component type

**component data types:** collection of all data types, component types and structured types whose sub-elements are component data types

**component port:** port defined in a component type

**component template:** template defined in a component type

**component timer:** timer defined in a component type

**component variable:** variable defined in a component type

**control behaviour:** collection of module control functions with the name **control** and functions and altsteps called by **control** directly or through other control functions or altsteps, and are used for the dynamic execution of test cases

NOTE: Such functions and altsteps are called control functions and control altsteps respectively. Module control functions can be used as an entry point of executing a test suite. Declaring functions or altsteps with the modifier **@control** explicitly allows to distinguish them from test case behaviour definitions in their special role. Module control functions and behaviour definitions with the **@control** modifier are called explicit control behaviour definitions, i.e explicit control functions and explicit control altsteps.

**data types:** all types whose values or sub-elements cannot contain object references

NOTE: Data types include simple basic types, basic string types, and the special data type anytype. Data types also include all structured types where all their sub-elements are of a data type. All user defined types based on a data type are data types as well. See more details in table 3 of the present document.

**defined types (defined TTCN‑3 types):** set of all predefined TTCN‑3 types (basic types, all structured types, the type anytype, the address, port and component types and the default type) and all user-defined types declared either in the module or imported from other TTCN‑3 modules

**deterministic function:** function that for the same input in the in and inout parameters always yields the same output both for the return result as well as the inout and out parameters

NOTE 1: A non-deterministic function is one that is not deterministic.

NOTE 2: In general, it cannot be decided if a function is deterministic or not. However, a function can be specified to be deterministic, i.e. the function is supposed to be deterministic. In this case, a violation of the determinism can be detected and handled accordingly. The handling however is tool-specific.

**dynamic parameterization:** form of parameterization, in which actual parameters are dependent on runtime events

EXAMPLE: The value of the actual parameter is a value received during runtime or depends on a received value by a logical relation.

**exception:** in cases of procedure-based communication, an exception (if defined) is raised by an answering entity if it cannot answer a remote procedure call with the normal expected response

**formal parameter:** typed name or typed template reference (identifier) not resolved at the time of the definition of an entity (function, test case, altstep, etc.) but at the time of invoking it

NOTE: Actual values or templates (or their names) to be used at the place of formal parameters are passed from the place of invoking the entity (see also the definition of actual parameter).

**fuzzy value or template:** value or template instance that is declared to be fuzzy and consequently the expression, initializing or partly initializing it (including actual parameters passed to **in** formal parameters), is subject to lazy evaluation

NOTE: During execution, this expression is re-evaluated each time when the fuzzy object is referenced, except when at the left hand side of an assignment or passing it to a fuzzy or lazy formal parameters. The result of this (re)evaluation is used as the actual value or template of the fuzzy instance. When new content is assigned to a fuzzy instance or to its subpart, the right hand side of the assignment is subject to lazy evaluation again.

**global visibility:** attribute of an entity (module parameter, constant, template, etc.) whose identifier can be referenced anywhere within the module where it is defined including all functions, test cases and altsteps defined within the same module

**implementation conformance statement (ICS):** See Recommendation ITU‑T X.290 [5].

**implementation extra information for testing (IXIT):** See Recommendation ITU‑T X.290 [5].

**implementation under test (IUT):** See Recommendation ITU‑T X.290 [5].

**in parameterization:** kind of parameterization where the value of the actual parameter (the argument) is assigned to the formal parameter when the parameterized object is invoked, but the value of the formal parameter is not passed back to the actual parameter when the invoked object completes

NOTE 1: In **in** parameterization, parameters are passed by value.

NOTE 2: The arguments are evaluated before the parameterized object is entered.

NOTE 3: Only the values of the arguments are passed and changes to the arguments within the invoked object have no effect on the arguments as seen by the invoking object.

**index notation:** notation to access individual elements of record of, set of, array and string values or templates, where the element to be accessed is identified explicitly by an index value enclosed in square brackets ("[" and "]") which specifies the position of that element within the referenced value or template and the index value is either an integer value, array of integers or record of integers

NOTE: Integer values used for indexing (either directly or as elements of the record of or array values) always lie within the index range of the type of the referenced value or template. Except for those arrays which are defined with an explicit index range, the index range always has 0 as the index for the first element.

**initialization:** value or template, or a value or template field is initialized when a content is first assigned to it

NOTE: The assignment may be explicit at the declaration of the given object, in which case the same restrictions apply as for the right-hand side of the assignment operation, or at first use on the left-hand side of an assignment, or may be implicit. Implicit initialization occurs when a yet uninitialized object is passed as actual parameter to an out formal parameter of a directly called testcase, function or altstep returns with a non-uninitialized value or template that is assigned to the actual parameter; or when module parameters not initialized in the TTCN-3 code get their runtime values before test suite execution.

**inout parameterization:** kind of parameterization that uses passing by reference, i.e. when the parameterized object is invoked, the formal parameter is linked with the actual parameter and gets direct access to the same data content that is currently represented by the actual parameter

NOTE 1: The invoked object uses the actual parameter directly, so that all changes made in the formal parameter become immediately effective on the actual parameter. If the same actual parameter is passed to two distinct formal parameters, a change in one formal parameter becomes immediately effective in the other one (and in the actual parameter).

NOTE 2: Inout parameters can be used for functions, altsteps, and test cases only, if not restricted by further rules, e.g. altsteps activated as defaults.

**invalid expressions/operations:** an expression or operation is invalid if it does not follow the conditions and restrictions of this standard. Such expressions and operations shall cause a dynamic error during execution when they are evaluated or they might cause a static error when they are statically analyzed Possibly invalid expressions should be warned about during static analysis..

**known types:** set of all TTCN‑3 predefined types, types defined in a TTCN‑3 module and types imported into that module from other TTCN‑3 modules or from non-TTCN‑3 modules

**lazy evaluation:** evaluation of an expression, delayed during execution until the value or template instance, to which the result of the evaluation should have been assigned or passed to as actual parameter, is first referenced at another place than the left hand side of an assignment or an actual parameter passed to a fuzzy or lazy formal parameter

NOTE: During execution, this delayed evaluation is carried out at the first actual reference, even when the result is to be used in an expression that is also subject to lazy evaluation. For the evaluation the actual values at the time of the evaluation are to be used (not the actual values at the time of the assignment or parameter passing). This implies that components of the expression may be uninitialized at the time, when execution reaches the assignment or parameter passing, but may be initialized by the time of the evaluation that can lead to successful evaluation. If, by the time of the evaluation, execution has left the scope unit, in which one or more components of the expression is defined, the actual values of the component(s) at the time of leaving the scope unit are to be stored for the purpose of the delayed evaluation (but only for that, i.e. the values are not accessible for the user).

**lazy value or template:** value or template instance for which the expression, initializing or partly initializing it (including actual parameters passed to in formal parameters), is subject to lazy evaluation

NOTE: When, during execution, the delayed (lazy) evaluation is taking place, its result is stored in the lazy value or template and the lazy instance is used further on like ordinary values and templates, until the next use of the lazy variable or parameter on the left hand side of an assignment. When a new content is assigned to a lazy instance or to its subpart, the right hand side of the assignment is subject to lazy evaluation again. If, during execution, no expression referencing the lazy object is evaluated, the lazy value or template instance is never evaluated.

**left hand side (of assignment):** value or template variable identifier or a field name of a structured type value or template variable (including array index if any), which stands left to an assignment symbol (:=)

NOTE: A constant, module parameter, timer, structured type field name or a template header (including template type, name and formal parameter list) standing left of an assignment symbol (:=) in declarations and or a modified template definitions are out of the scope of this definition as not being part of an assignment.

**local visibility:** attribute of an entity (constant, variable, etc.) that its identifier can be referenced only within the function, test case or altstep where it is defined

**main test component (MTC):** See Recommendation ITU‑T X.292 [3].

**object:** instance of one of the object types (component, default, port and timer)

NOTE: Objects of type default, port or timer, which are owned by the component that instantiated them, are local objects while objects of type component are global objects. Global objects can be referenced from other component scopes while references to local objects can only be used by the component they are bound to.

**object reference:** special kind of value used for instances of component, default, port and timer types which represents a reference to an existing entity in the TE

NOTE: When used in assignments or parameter passing, only the reference to the entity is copied, but not the entity itself. An object reference can also be initialized with the special value null in which case it does not reference an object.

**out parameterization:** kind of parameterization where the actual parameter's content (the argument) is not passed to the formal parameter when the parameterized object is invoked, but the content of the formal parameter is passed back to the actual parameter when the invoked object completes, if the formal parameter has been initialized during the invocation and the actual parameter is the reference evaluated at the time of the invocation

NOTE 1: In **out** parameterization, parameters are passed by value.

NOTE 2: Out parameters can be used for functions, altsteps, and test cases only, if not restricted by further rules, e.g. **altstep**s activated as defaults.

NOTE 3: Formal an **out** parameters are uninitialized (unbound) when the invoked object is entered.

**parallel test component (PTC):** See Recommendation ITU‑T X.292 [3].

**partially initialized:** value or template is partially initialized if initialization has taken place on it or to at least one of its fields or elements

NOTE: A template variable is initialized if a matching mechanism has been assigned to it or to at least one of its fields or elements, directly or indirectly via expansion (see clause 15.6). A template is initialized if a matching mechanism has been assigned to it, directly or indirectly via expansion (see clause 15.6).

**passing by reference:** ability to link an actual parameter with a formal parameter of a function, altstep or test case and to control its actual value within the function, altstep or test case by using the formal parameter reference, i.e. no copy of the data content is made and the actual and formal parameters share the same data content

**passing by value:** ability to make a copy of a data content of an actual or formal parameter before passing it to a formal or actual parameter, i.e. the actual and formal parameters do not share the same data content

NOTE: When passing object references by value, a new reference is created, but the referenced entity remains the same.

**qualified name:** TTCN-3 elements can be identified unambiguously by qualified names

NOTE: For modules, the qualified name is the <module name>. For global definitions such as testcases, functions, etc., the qualified name is <module name>.<definition name>. For control, the qualified name is <module name>.control. For local definitions, such as variables, local templates, etc. within a global definition, the qualified name is <module name>.<global definition name>.<local definition name>.

**right hand side (of assignment):** expression, template reference or signature parameter identifier which stands right to an assignment symbol (:=)

NOTE: Expressions and template references standing right of an assignment symbol (:=) in constant, module parameter, timer, template or modified template declarations are out of the scope of this definition as not being part of an assignment.

**root type:** root types of types derived from TTCN-3 basic types are the respective basic types

NOTE 1: The root type of user defined record types is **record**, the root type of user defined record of and array types is **record of**, the root type of user defined set types is **set**, the root type of user defined set of types is **set of**. The root type of user defined union types is **union** and the root type of anytypes is **anytype**. The root types of special configuration types are **default** or **component,** respectively. Port types do not have a root type.

NOTE 2: As **address** is more a predefined type name than a distinct type with its own properties, the root type of an **address** type and all of its derivatives are the same as the root type was, if the type was defined with a name different from **address**.

**static parameterization:** form of parameterization, in which actual parameters are independent of runtime events; i.e. known at compile time or in case of module parameters are known by the start of the test suite execution

NOTE 1: A static parameter is to be known from the test suite specification, (including imported definitions), or the test system is aware of its value before execution time.

NOTE 2: All types are known at compile time, i.e. are statically bound.

**strong typing:** strict enforcement of type compatibility by type name equivalence with no exceptions

**system under test (SUT):** See Recommendation ITU‑T X.290 [5].

**template:** TTCN-3 data objects are values or templates by definition. A TTCN‑3 template identifies a subset of the values of its type (where the subset may contain a single instance of the type, several instances or all instances) or the matching mechanism **omit**

NOTE: Templates are defined by global and local templates, template variable definitions, or formal template parameters. Any of those are templates from the point of view of their usage, irrespective of their actual content; for example, a template variable containing a specific value is a template.

**template parameterization:** ability to pass a template as an actual parameter into a parameterized object via a template parameter

NOTE 1: This actual template parameter is added to the specification of that object and may complete it.

NOTE 2: Values passed to formal template parameters are considered to be in-line templates (see clause 15.4).

**test behaviour:** (or behaviour) test case, function or altstep started on a test component when executing an **execute** or a **start** component statement and all functions and altsteps called recursively

NOTE: During a test case execution each test component has its own behaviour and hence several test behaviours may run concurrently in the test system (i.e. a test case can be seen as a collection of test behaviour).

**test case:** See Recommendation ITU‑T X.290 [5].

**test case error:** See Recommendation ITU‑T X.290 [5].

**test suite:** set of TTCN‑3 modules that contains a completely defined set of test cases, optionally supplemented with one or more TTCN‑3 control functions

**test system:** See Recommendation ITU‑T X.290 [5].

**test system interface:** test component that provides a mapping of the ports available in the (abstract) TTCN‑3 test system to those offered by the SUT

**type compatibility:** language feature that allows to use values, expressions or templates of a given type as actual values of another type

EXAMPLE: At assignments, as actual parameters at calling a function, referencing a template, etc. or as a return value of a function.

**type context:** "In the context of a type" means that at least one object involved in the given TTCN-3 action (an assignment, operation, parameter passing, etc.) identifies a concrete type unambiguously

NOTE: Either directly (e.g. an in-line template) or by means of a typed TTCN-3 object (e.g. via a constant, variable, formal parameter, etc.).

**uninitialized:** value or template is uninitialized as long as no initialization of it or at least one of its parts has occurred

**unqualified name:** unqualified name of a TTCN-3 element is its name without any qualification

**user-defined type:** type that is defined by subtyping of a basic type or declaring a structured type

NOTE: User-defined types are referenced by their identifiers (names).

**valid expressions/operations:** valid expression/operations are expression/operation that follow the conditions and restrictions of this standard and can be safely compiled and executed.

**value:** TTCN-3 data objects are values or templates by definition. A TTCN‑3 value is an instance of its type

NOTE: Values are defined by module parameters, constants, value variables, or formal value parameters. Any of those are value objects from the point of view of their usage. A template containing only specific value matching - though referring to a single instance of its type - is not a value object, but is a template object.

**value list notation:** notation that can be used for record, set, record of and set of values, where the values of the subsequent fields or elements are listed within a pair of curly brackets ("{" and "}"), without an explicit identification of the field name or element position

**value notation:** notation by which an identifier is associated with a given value or range of a particular type

NOTE: Values may be constants or variables.

**value parameterization:** ability to pass a value as an actual parameter into a parameterized object via a value parameter

NOTE: This actual value parameter is added to the specification of that object and may complete it.