ETSI TR 104 081 V5.1.1 (2026-04)

TECHNICAL REPORT

Methods for Testing & Specification (MTS); Guide for the Migration to TTCN-3 V5

Reference
DTR/MTS-104081v5.1.1

Keywords
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 - APE 7112B
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° w061004871

Contents

Intellectual Property Rights

Essential patents

IPRs essential or potentially essential to normative deliverables may have been declared to ETSI. The declarations pertaining to these essential IPRs, if any, are 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 IPR online database.

Pursuant to the ETSI Directives including the ETSI IPR Policy, no investigation regarding the essentiality of IPRs, 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.

DECT™, **PLUGTESTS™**, **UMTS™** and the ETSI logo are trademarks of ETSI registered for the benefit of its Members. **3GPP™**, **LTE™** and **5G™** logo 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.

Foreword

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

The present document is the TTCN-3 migration guide and contains all the information required to migrate from previous versions of TTCN-3 to TTCN-3 version 5.1.1.

Modal verbs terminology

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

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

Introduction

The TTCN-3 testing language has intensively been developed by ETSI during the last 25 years. By today, TTCN-3 has become a significantly important testing technology in different domains. The last major release of the TTCN-3 standards was published in 2009 with version 4.1.1. Since then, TTCN-3 was constantly maintained and further developed.

The TTCN-3 language specification has reached the state where modernization and refactoring are required. Currently, language maintenance has become complex, since even small corrections and extensions require changes in many different places in the TTCN-3 language specification and extension packages. Also, TTCN-3 users suffer from this kind of complexity, because specific aspects of a language feature may be spread across different sections and documents. This decreases learning, understanding and, finally, acceptance of TTCN-3.

Therefore, it has been decided to develop a new major revision of TTCN-3 that takes into account user requirements and needs to modernize and update TTCN-3. After a survey in cooperation with TTCN-3 users a discussion on the relevant and needed language features has been started. The feedback has been used and results in an approach to update the language. Detailed information on the proposed changes has been collected and result in the so-called migration guide that provides all changes for the TTCN-3 language standard parts and extensions. This information is given in the present document.

1 Scope

The present document is the TTCN-3 migration guide. It contains the following information:

- A list of TTCN-3 standard documents not supported in TTCN-3 V5.1.1.
- The new structure of the TTCN-3 language specification V5.1.1.
- A list of all language features in the new TTCN-3 core language document V5.1.1.
- A list of all language features supported in the TTCN-3 extensions V5.1.1.
- A list of all semantic changes to language features in TTCN-3 V5.1.1.
- A list of all language features not supported in TTCN-3 V5.1.1 but supported in earlier versions of TTCN-3.
- A list of agreed-upon workflows for implementing all TTCN-3 language specifications V5.1.1.

2 References

2.1 Normative references

Normative references are not applicable in the present document.

2.2 Informative references

References are either specific (identified by date of publication and/or edition number or version number) or nonspecific. 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 may be useful in implementing an ETSI deliverable or add to the reader's understanding but are not required for conformance to the present document.

[i.1] ETSI ES 201 873-1 (V4.17.1): "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language".

[i.2] 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.3] 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.4] 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.5] ETSI ES 201 873-9: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 9: Using XML schema with TTCN-3".

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

[i.7] ETSI ES 201 873-11: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 11: Using JSON with TTCN-3".

[i.8] 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.9] ETSI ES 202 782: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; TTCN-3 Language Extensions: Performance and Real Time Testing".

[i.10] 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.11] 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.12] ETSI ES 202 786: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; TTCN-3 Language Extensions: Support of interfaces with continuous signals".

[i.13] ETSI ES 203 790: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; TTCN-3 Language Extensions: Object-Oriented Features".

[i.14] ETSI ES 203 022: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; TTCN-3 Language extension: Advanced Matching".

[i.15] Recommendation ITU-T X.290: "OSI conformance testing methodology and framework for protocol Recommendations for ITU-T applications - General concepts".

[i.16] ETSI ES 202 789: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; TTCN-3 Language Extensions: Extended TRI".

[i.17] ETSI ES 201 873-3 (V3.2.1): "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 3: TTCN-3 Graphical presentation Format (GFT)".

[i.18] ETSI ES 201 873-4 (V4.6.1): "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 4: TTCN-3 Operational Semantics".

[i.19] ETSI ES 201 873-8 (V4.8.1): "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 8: The IDL to TTCN-3 Mapping".

3 Definition of terms, symbols and abbreviations

3.1 Terms

For the purposes of the present document, the following terms 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

automatic type: notation used in variable, constant and module parameter declarations where the type part of the declaration is missing, and the type is implicitly set by the provided initial value

basic types: set of the following predefined TTCN-3 types: **integer**, **float**, **boolean**, **verdicttype**, **bitstring**, **hexstring**, **octetstring**, **charstring**, and **universal charstring**

NOTE: Basic types are referenced by their names.

communication port: abstract mechanism facilitating communication between test components

NOTE 1: A communication port is modelled as a FIFO queue in the receiving direction. Ports can be message-based or procedure-based.

NOTE 2: Variables, constants, templates, etc. have compatible types if conditions in clause 6.3 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.

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.

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 fuzzy 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.

initialization: first assignment of content to a value or template, or a value or template field

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.

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.

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.

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. A template is initialized if a matching mechanism has been assigned to it, directly or indirectly via expansion.

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

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.

System Under Test (SUT): See Recommendation ITU-T X.290 [i.15].

template: TTCN-3 data object that 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.

test case: See Recommendation ITU-T X.290 [i.15].

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 [i.15].

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.

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

value: 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 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.

3.2 Symbols

Void.

3.3 Abbreviations

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

ASN.1	Abstract Syntax Notation 1
ATS	Abstract Test Suite
BNF	Backus-Naur Form
CR	Change Request
FIFO	First In First Out
IDL	Interface Definition Language
ITU-T	International Telecommunication Union - Telecommunication standardization sector
JSON	JavaScript Object Notation
SUT	System Under Test
TCI	TTCN-3 Control Interfaces
TRI	TTCN-3 Runtime Interfaces
TTCN-3	Testing and Test Control Notation version 3
UCS	Universal Character Set
UTF	Unicode Transformation Format
XML	eXtensible Markup Language

4 Not supported TTCN-3 standards

After the major revision of the TTCN-3 language, the technical content of the following European Standards (ES) will not be supported any more:

- ETSI ES 201 873-3 (V3.2.1) [i.17].
- ETSI ES 201 873-4 (V4.6.1) [i.18].
- ETSI ES 201 873-8 (V4.8.1) [i.19].
- ETSI ES 202 781 [i.8].
- ETSI ES 202 786 [i.12].

5 New structure of TTCN-3 language specifications

The major revision of the TTCN-3 language will consist of the following seven European Standards (ES):

- ETSI ES 201 873-1 (V5.1.1): "TTCN-3 Core Language" will contain the definition of the core language features.
- ETSI ES 201 873-12 (V5.1.1): "TTCN-3 Extensions" will contain selected features from TTCN-3 extensions conforming to previous core language specifications (V4.x.x) and features from the core language that have been moved to ETSI ES 201 873-12 (V5.1.1).
- ETSI ES 201 873-13 (V5.1.1): "TTCN-3 Runtime Interface (TRI) and Control Interface (TCI) " will provide a generalized merger of ETSI ES 201 873-5 [i.2], ETSI ES 201 873-6 [i.3] and ETSI ES 202 789 [i.16].
- ETSI ES 201 873-7 (V5.1.1): "Using ASN.1 with TTCN-3" will adapt previous versions of ETSI ES 201 873-7 [i.4] to the requirements of ETSI ES 201 873-1 (V5.1.1).

- ETSI ES 201 873-9 (V5.1.1): "Using XML schema with TTCN-3" will adapt previous versions of ETSI ES 201 873-9 [i.5] to the requirements of ETSI ES 201 873-1 (V5.1.1).
- ETSI ES 201 873-10 (V5.1.1): "TTCN-3 Documentation Comment Specification" will adapt previous versions of ETSI ES 201 873-10 [i.6] to the requirements of ETSI ES 201 873-1 (V5.1.1).
- ETSI ES 201 873-11 (V5.1.1): "Using JSON with TTCN-3" will adapt previous versions of ETSI ES 201 873-11 [i.7] to the requirements of ETSI ES 201 873-1 (V5.1.1).

NOTE: ETSI ES 201 873-3 [i.17], ETSI ES 201 873-4 [i.18] and ETSI ES 201 873-8 [i.19] will not be supported anymore.

The TTCN-3 ES are accompanied with the following two Technical Reports (TR):

- ETSI TR 104 873 (V5.1.1).
- ETSI TR 104 081 (V5.1.1) (the present document).

6 Features of new core language document

6.0 General

Features of the new core language standard ETSI ES 201 873-1 (V5.1.1) will include selected and modified features from the ETSI ES 201 873-1 (V4.17.1) [i.1].

For each of the following clauses references to base documents and clauses in base documents are provided.

The order of the following clauses does not reflect the order of feature descriptions in the new core language document ETSI ES 201 873-1 (V5.1.1).

6.1 Modification of major concepts affecting several clauses

6.1.0 General

There are two major concepts that will be changed:

- the concept of fuzzy and lazy values and templates and lazy evaluation will be deleted; and
- the concept of procedure-based communication will be moved to ETSI ES 201 873-12 (V5.1.1).

The clauses affected by these modifications are listed below, and in the rest of the present document they will not be separately indicated in the affected clauses.

6.1.1 Fuzzy and lazy values and templates and lazy evaluation

The concepts of Fuzzy and lazy values and templates (see clause 4 in ETSI ES 201 873-1 (V4.17.1) [i.1] and lazy evaluation (see clause 4 in ETSI ES 201 873-1 (V4.17.1) [i.1]) will be removed. This change will also have an effect on several additional clauses. Those parts of these clauses that are related to the concepts Fuzzy and lazy values and templates or to lazy evaluation will also be removed. The affected clauses in ETSI ES 201 873-1 (V4.17.1) [i.1] are the following:

- 3.1 "Terms"
- 5.4.1 "Formal parameters" and all of its clauses
- 5.4.2 "Actual parameters"
- 6.2.1.0 "Record type and values - General"

- 11 "Declaring variables" and all of its clauses
- 15.0 "Declaring templates - General"
- 15.3 "Global and local templates"
- 15.5 "Modified templates"
- 16.1.4 "Invoking functions from specific places"
- 19.1.1 "Basic assignments"
- 19.4.2 "The range-based loop"
- 21.3.5 "The Alive operation"
- 21.3.6 "The Running operation"
- 21.3.7 "The Done operation"
- 21.3.8 "The Killed operation"
- 22.2.2 "The Receive operation"
- 22.2.3 "The Trigger operation" (this operation will be deleted, see clause 9.5 of the present document)
- 22.3.2 "The Getcall operation" (this operation will be moved to ETSI ES 201 873-12 (V5.1.1), see clause 7.1.1 of the present document)
- 22.3.4 "The Getreply operation" (this operation will be moved to ETSI ES 201 873-12 (V5.1.1), see clause 7.1.1 of the present document)
- 22.3.6 "The Catch operation" (this operation will be moved to ETSI ES 201 873-12 (V5.1.1), see clause 7.1.1 of the present document)
- 22.4 "The Check operation"

NOTE: In the present document, these changes will not be separately indicated in the affected clauses, they will contain only the additional changes.

6.1.2 The concept of procedure-based communication

The whole concept of procedure-based communication (see clause 22.3 in ETSI ES 201 873-1 (V4.17.1) [i.1] will be moved to ETSI ES 201 873-12 (V5.1.1). This change will also have an effect on several additional clauses. Those parts of these clauses that are related to the concept of procedure-based communication will also be moved to ETSI ES 201 873-12 (V5.1.1). The affected clauses in ETSI ES 201 873-1 (V4.17.1) [i.1] are the following:

- 6.2.9 "Communication port types"
- 6.3.4 "Type compatibility of communication and connection operations"
- 9.1 "Communication ports"
- 9.2 "Test system interface"
- 14 "Declaring procedure signatures"
- 15.6.4 "Referencing signature parameters"
- 20.1 "The snapshot mechanism"
- 20.2 "The Alt statement"

NOTE: In the present document, these changes will not be separately indicated in the affected clauses, they will contain only the additional changes.

6.2 Definition of terms, symbols and abbreviations

The core language document ETSI ES 201 873-1 (V5.1.1) will include clauses on terms, symbols and definitions (see clause 3 in ETSI ES 201 873-1 (V4.17.1) [i.1]). The clauses on terms, symbols and definitions have to be adapted due to changed ETSI rules and due to changes of content in ETSI ES 201 873-1 (V5.1.1) in comparison to ETSI ES 201 873-1 (V4.17.1) [i.1].

6.3 Introduction

ETSI ES 201 873-1 (V5.1.1) will have an adapted "Introduction" (see clause 4 in ETSI ES 201 873-1 (V4.17.1) [i.1]). Clause 4.1 of ETSI ES 201 873-1 (V4.17.1) [i.1] will be deleted. References to other TTCN-3 documents will be revised or deleted.

6.4 Basic language elements

6.4.0 General

ETSI ES 201 873-1 (V5.1.1) will include an adapted clause on "Basic language elements" (see clause 5.0 in ETSI ES 201 873-1 (V4.17.1) [i.1]).

6.4.1 Identifiers and keywords

ETSI ES 201 873-1 (V5.1.1) will include an adapted clause on "Identifiers and keywords" (see clause 5.1 in ETSI ES 201 873-1 (V4.17.1) [i.1]).

6.4.2 Scope rules

In addition to the scope rules defined in clause 5.2 of ETSI ES 201 873-1 (V4.17.1) [i.1], a "root scope" will be introduced. The root scope simplifies scoping rules for predefined functions (see clause C.1 to C.6 in ETSI ES 201 873-1 (V4.17.1) [i.1]) and Useful TTCN-3 types (see clause E.2 in ETSI ES 201 873-1 (V4.17.1) [i.1]).

6.4.3 Ordering of language elements

The rules for the ordering of language elements (see clause 5.3 in ETSI ES 201 873-1 (V4.17.1) [i.1]) remain unchanged.

6.4.4 Parameterization

In comparison to clauses 5.4 and 5.5 in ETSI ES 201 873-1 (V4.17.1) [i.1] the clause on "Parameterization" in ETSI ES 201 873-1 (V5.1.1) will change in the following manner:

- Parameterization concept will be only about dynamic parameterization in runtime, therefore Modulepar concept will be described in a different clause.

Further changes are minor and concern wording and technical details.

6.5 Types and values

6.5.0 General

Clause 6 in ETSI ES 201 873-1 (V4.17.1) [i.1] will be re-structured (see clause 6.5.1 of the present document) and partially modified (see clauses 6.5.2 to 6.5.5 of the present document). The technical contents of clauses from ETSI ES 201 873-1 (V4.17.1) [i.1] which are not mentioned in this clause remain unchanged.

6.5.1 Re-structuring of types and values

The Types and values (see clause 6 in ETSI ES 201 873-1 (V4.17.1) [i.1]) will be re-structured. This re-structuring will make the type system of TTCN-3 more straightforward, easier to understand and will make the description of the TTCN language elements in this and in the following clauses simpler, however this re-structuring will not influence how these language elements can be used, apart from the proposed changes mentioned in clauses 6.5.2 to 6.5.5 of the present document.

The following new structure of the "Types and values" clause is proposed:

x Types and values
- x.0 General
- x.1 Basic types and values
- x.2 Structured types and values
  - x.2.1 Record types and values
  - x.2.2 Set types and values
  - x.2.3 Records and sets of elements of the same type
    - x.2.3.0 General
    - x.2.3.1 Nested type definitions
    - x.2.3.2 Referencing elements of record of and set of types
    - x.2.3.3 Arrays
  - x.2.4 Unions
  - x.2.5 The anytype
  - x.2.6 Map type
  - x.2.7 Subtypes of structured types
- x.3 Enumerated types and values
- x.4 The address types (in ETSI ES 201 873-1 (V4.17.1) [i.1] Addressing entities inside the SUT)
- x.5 The open type
- x.6 Type synonym
- x.7 Compatibility of values of different data types
  - x.7.1 Compatibility of values of basic types
  - x.7.2 Compatibility of values of structured types
    - x.7.2.1 Compatibility of values of record types
    - x.7.2.2 Compatibility of values of record of types
    - x.7.2.3 Compatibility of values of set types
    - x.7.2.4 Compatibility of values of set of types
    - x.7.2.5 Compatibility of values of union types
    - x.7.2.6 Compatibility of values of anytype types
    - x.7.2.7 Compatibility of values of sub-structures
    - x.7.2.8 Compatibility of values of map types
  - x.7.3 Compatibility of values of enumerated types
  - x.7.4 Compatibility of values used in communication and connection operations
  - x.7.5 Compatibility of values of open types
  - x.7.6 Type conversion
- x.8 TTCN-specific, Test-related types and references
  - x.8.1 Communicating port types and port references
  - x.8.2 Component types and component references
  - x.8.3 Timer type and timer references
  - x.8.4 The default type and default references
    - x.8.4.1 Compatibility of references
    - x.8.4.2 Compatibility of port references
    - x.8.4.3 Compatibility of component references
    - x.8.4.4 Compatibility of timer references
    - x.8.4.5 Compatibility of default references

Automatic type is not a type, but a syntactical rule how constants and variables can be declared, it will be handled in the clauses related to constants and variables.

NOTE: Only part of the content of Automatic type clause (see clause 6.5 in ETSI ES 201 873-1 (V4.17.1) [i.1]) will remain in ETSI ES 201 873-1 (V5.1.1) and the rest will be moved to ETSI ES 201 873-12 (V5.1.1).

6.5.2 Changes in "Basic types and values"

The following modifications are proposed in clause 6.1 of ETSI ES 201 873-1 (V4.17.1) [i.1]:

- Values of types **bitstring**, **hexstring** and **octetstring** can be denoted not only by the uppercase 'B, 'H, 'O characters, but their lowercase equivalents 'b, 'h, 'o will also be enabled, like: '0101'b, 'ABCD'h and 'ABCD'o.
- The default string type will be the current **universal charstring** type. A new keyword **string** will be introduced for it, but for backward compatibility reasons the **universal charstring** keyword will be kept.
- The current **charstring** type will be defined as a subtype of the new **string** type. This will allow to simplify the descriptions in other clauses, but will not cause any difference for the user, the charstrings can be used in the same way as of now, and the keyword **charstring** will still remain.

6.5.3 Changes in Records and sets of single types and Arrays

For the Record and Set of elements of the same type (see clause 6.2.3 in ETSI ES 201 873-1 (V4.17.1) [i.1]) the rules describing how many elements a partially initialized record of/set of has, and what happens if a value on the right-hand side of an assignment has different number of elements than that of on the left-hand side will be clarified.

- If a record of/set of value has uninitialized element(s), they will count into the number of elements of that value, even if they are at the end:

```
type record of integer RoI;
var RoI v_myVar1 := { 0, 1, -, - }        // v_myVar1 will have 4 elements,
                                          // last 2 are uninitialized
var RoI v_myVar2 := { 0, -, 1, -, - }     //v_myVar1 will have 5 elements,
                                          // second and last 2 are uninitialized

var RoI v_myVar3 := {
    [0] := 0,
    [1] := 1,
    [2] := -
}                                         // v_myVar1 will have 3 elements, last is
                                          // uninitialized
```

- In an assignment, the left-hand side will be the same as the right-hand side:

```
v_myVar1 := v_myVar2;       // v_myVar1 will have 5 elements
v_myVar1 := {1,2,-};        // v_myVar1 will have 3 elements {1,2,1}
v_myVar1 := v_myVar3;          // v_myVar1 will have 3 elements
                            // {0,1, <uninitialized>}
```

For arrays (see clause 6.2.7 in ETSI ES 201 873-1 (V4.17.1) [i.1]), it will be clarified that arrays can be used in TTCN-3 as a shorthand notation to specify record of types with a fixed length restriction.

6.5.4 Changes in Port types

Since procedure-based communication will be moved to ETSI ES 201 873-12 (V5.1.1), in the Core Language only the message-based port type will remain. Therefore, in message-based port type definitions the keyword **message** will become optional. It is kept for backward compatibility. The following two definitions will be identical:

```
type port PortTypeIdentifier message "{"
...
"}"

type port PortTypeIdentifier "{"
...
"}"
```

6.5.5 Changes in Automatic type

The concept of Automatic type (see clause 6.5 in ETSI ES 201 873-1 (V4.17.1) [i.1]) will be moved to ETSI ES 201 873-12 (V5.1.1) without the following exception that will remain in ETSI ES 201 873-1 (V5.1.1):

- The automatic typing will only be allowed only if on the right-hand side of the assignment there is a reference or a literal.

NOTE: This can also be applied to the cycle variable of a counter loop.

6.6 Expressions

In Expressions (see clause 7 in ETSI ES 201 873-1 (V4.17.1) [i.1]) following change is proposed:

- Shift and rotate operators will be moved to ETSI ES 201 873-12 (V5.1.1).

6.7 Modules

6.7.0 General

Apart from clause 8.2.3 "Importing from modules" in ETSI ES 201 873-1 (V4.17.1) [i.1] the technical contents of clause 8 in ETSI ES 201 873-1 (V4.17.1) [i.1] remain unchanged. Especially:

- Transfer syntax of TTCN-3 modules will be UTF-8.
- The language keyword remains. The list of language strings will be updated.
- Definitions can be grouped (for readability purposes).

In addition:

- All rules about modulepar (especially from parameterization) will be moved to a clause on module parameters (see clause 8.2.1 in ETSI ES 201 873-1 (V4.17.1) [i.1]).

6.7.1 Changes in Importing from modules

The technical content of clause 8.2.3 "Importing from modules" in ETSI ES 201 873-1 (V4.17.1) [i.1] will be changed in the following manner:

- Only **`import all`** will be supported.
- The rules on import of attributes will be simplified:
  - If an imported definition has attributes (defined by means of a **`with`** statement) then the attributes will also be imported.
  - If the imported module has no attribute, then the **`with`** statement may be attached to the **`import`** statement, and this **`with`** statement cannot have an **`override`** directive.

NOTE: The attributes specified in this **`with`** statement will be applied to all imported definitions that had no attributes defined, that is during import, the existing attributes cannot be changed.

- If a module is imported - explicitly or implicitly - more than once, then the attributes defined for the module will be the mutually identical.

- The rules on import of attributes will be simplified: attributes may be attached only when the original module does not include them and if present, the attributes attached to import of a particular module will be the same through the whole test suite.

6.8 Port types, component types and test configurations

The technical contents of clause 9 in in ETSI ES 201 873-1 (V4.17.1) [i.1] will remain unchanged for message-based communication.

6.9 Declaring constants

The technical contents of clause 10 in ETSI ES 201 873-1 (V4.17.1) [i.1] remain unchanged.

6.10 Declaring variables

The technical contents of clause 11 in ETSI ES 201 873-1 (V4.17.1) [i.1] remain unchanged.

6.11 Declaring timers

The technical contents of clause 12 in ETSI ES 201 873-1 (V4.17.1) [i.1] remain unchanged.

6.12 Declaring messages

The technical contents of clause 13 in ETSI ES 201 873-1 (V4.17.1) [i.1] remain unchanged.

6.13 Declaring templates

The proposed modifications related to templates (see clause 15 in ETSI ES 201 873-1 (V4.17.1) [i.1]) are the following:

- Templates without a (possibly empty) parameter list are static templates. They will be evaluated only once, at the place of their declaration.

NOTE 1: A static template retains the value was assigned to it at the initialization and will not be re-evaluated later at its usage, even when invoking functions or dynamic templates directly or indirectly.

- Templates with a (possibly empty) parameter list are dynamic templates. A dynamic template is evaluated at every invocation, except for being invoked directly or indirectly from a static template; in this case it is invoked only once, at the initialization of that static template that directly invokes it. If the parameter list of a dynamic template is empty, then at invocation the usage of ' ( ) ' is optional. But if all the parameters of a dynamic template have default values and at the invocation all these default values wanted to be used, then the usage of the ' ( ) ' remains mandatory. Dynamic templates can only be global templates.

NOTE 2: Removing the possibility of defining local templates with parameters is a backward-incompatible change.

NOTE 3: The functionality of the proposed dynamic template is the same as the functionality of the fuzzy templates is in ETSI ES 201 873-1 (V4.17.1) [i.1].

A new **`template`** attribute will be introduced. The attribute allows to control how the global templates in a module are interpreted. The **`template`** attribute will have two predefined string values:

- `"static"`: only those templates will be dynamic that have a (possibly empty) parameter list defined (the behaviour described above).
- `"dynamic"`: all the global templates will be treated as dynamic.

Only modules can have **`template`** attribute, and the default value is `"static"`.

```
module M {

...

} with { template "dynamic" }
```

The technical contents of the rest of the clause 15 in ETSI ES 201 873-1 (V4.17.1) [i.1] remain unchanged.

EXAMPLE 1:

```
// in a module definitions part;
// "static" template attribute is defined for the module

template  float t_static := rnd();          /* static template, initialized by random number 1 */

template  float t_dynamic() := rnd();       /* dynamic template, no initialization here */

template  float t_static2 := t_dynamic(); /* static template, initialized by random number 2 */

// e.g. in a testcase

P_PT.send(t_static);            /* random number 1 will be sent, template will not be evaluated
here */

P_PT.send(t_static);            /* random number 1 will be sent, template will not be evaluated
here */

P_PT.send(t_dynamic());      /* template will be evaluated here, random number 3 will be sent;
                                   alternative syntax:
                                   P_PT.send(t_dynamic);
                                   is also possible */

P_PT.send(t_dynamic());      /* template will be evaluated again here, random number 4 will be sent
*/

P_PT.send(t_static2);        /* random number 2 will be sent, template will not be evaluated here,
because if a dynamic template or a function is invoked directly or indirectly on the right-hand side
of a static template declaration, it is only evaluated once, at the declaration of the directly
invoking template, therefore neither t_dynamic() nor rnd() will be evaluated here */
```

EXAMPLE 2:

```
   // in a module definitions part;
   // "dynamic" template attribute is defined for the module

template  float t_static := rnd();       /* because of the "dynamic" template attribute of the
      module, this template will be treated as dynamic, so no initialization here */

template  float t_dynamic() := rnd();    /* dynamic template, no initialization here */

template  float t_static2 := t_dynamic(); /* because of the "dynamic" template attribute of the
      module, this template will be treated as dynamic, so no initialization here */

// e.g. in a testcase

P_PT.send(t_static);            /* template will be evaluated here, random number 1 will be sent */

P_PT.send(t_static);            /* template will be evaluated again here, random number 2 will be
sent */

P_PT.send(t_dynamic());      /* template will be evaluated here, random number 3 will be sent */

P_PT.send(t_dynamic());      /* template will be evaluated again here, random number 4 will be sent
*/

P_PT.send(t_static2);          /* template will be evaluated here, random number 5 will be sent */
```

6.14 Functions, altsteps and testcases

In Functions, altsteps and testcases (see clause 16 in ETSI ES 201 873-1 (V4.17.1) [i.1]) the following change is proposed:

- The not-implemented function will be moved to ETSI ES 201 873-12 (V5.1.1).

Apart from the changes described above, the technical contents of clause 16 will remain unchanged.

6.15 Basic program statements

In Basic program statements (see clauses 18 and 19 in ETSI ES 201 873-1 (V4.17.1) [i.1]) the following changes are proposed:

- The **`goto`** and **`label`** functionality is moved to ETSI ES 201 873-12 (V5.1.1).
- Range based loop allow only locally defined iterator, and it will iterate over the possible uninitialized elements (regardless of whether the uninitialized element is before or after the last initialized element).
- The semantics of **`log`** statement will not change (see clause 19.11 in ETSI ES 201 873-1 (V4.17.1) [i.1]), but the **`log`** statement will become a predefined function and will be moved to that clause.

6.16 Statements and operations for alternative behaviours

6.16.0 General

In clause 20 in ETSI ES 201 873-1 (V4.17.1) [i.1] the following changes will be made:

- Usage and semantics of the **`interleave`** statement will be simplified (see clause 6.5.1). This is a backward-incompatible change.

Apart from these changes the concepts described in clauses 20.0 to 20.3 in ETSI ES 201 873-1 (V4.17.1) [i.1] remain unchanged. Especially the semantics of default handling, snapshot mechanism and the **`alt`** statement and **`repeat`** statement will not be changed.

6.16.1 The Interleave statement

For the interleave statement (clause 20.4 in ETSI ES 201 873-1 (V4.17.1) [i.1]) the following semantical simplification is proposed:

- Complexity is introduced into the interleave statement by allowing reception statements in statement blocks of alternatives in interleave statements. Each reception statement adds to the complexity of interleaving by implicitly introducing (nested) alt statements.

The proposal is to disallow reception statements in the highlighted ***StatementeBlock*** below:

Syntactical Structure

```
interleave [ @nodefault ] "{"
{ "[]" ( TimeoutStatement |
         ReceiveStatement |
         TriggerStatement |
         CheckStatement |
         DoneStatement |
         KilledStatement ) StatementBlock
}
"}"
```

6.17 Configuration operations

For clause 21 in ETSI ES 201 873-1 (V4.17.1) [i.1] the following change is proposed:

- The Call test component behaviour operation (see clause 21.3.10 in ETSI ES 201 873-1 (V4.17.1) [i.1]) will be moved to ETSI ES 201 873-12 (V5.1.1).

Apart from the change described above, the technical contents of clause 21 will remain unchanged.

6.18 Communication Operations

In Communication Operations (see clause 22 in ETSI ES 201 873-1 (V4.17.1) [i.1]), the following changes are planned:

- The **`trigger`** operation will be deleted. This is a backward-incompatible change.
- The syntax of the check operation will be simplified, the **`receive`** keyword will be optional:
  - **check**(**receive**(...)) is the same as **check**(...).
- The TTCN-3 concept of procedure-based communication will be moved to ETSI ES 201 873-12 (V5.1.1). All communication operations related to procedure-based communication (i.e. **`call`**, **`getcall`**, **`reply`**, **`getreply`**, **`raise`**, **`catch`**) will be moved to ETSI ES 201 873-12 (V5.1.1).

Apart from the changes described above, the technical contents of clause 22 will remain unchanged.

6.19 Timer Operations

The concept of timer and timer operations (see clause 23 in ETSI ES 201 873-1 (V4.17.1) [i.1]) will not be changed.

6.20 Test Verdict Operations

The concept of test verdicts and test verdict operations (see clause 24 in ETSI ES 201 873-1 (V4.17.1) [i.1]) will not be changed.

6.21 External actions

For external actions (see clause 25 in ETSI ES 201 873-1 (V4.17.1) [i.1]) the following change is proposed:

- The semantics of external actions will not change, but external action will become a predefined function and will be moved to that clause.

6.22 Module control

The technical contents of clause 26 in ETSI ES 201 873-1 (V4.17.1) [i.1] will remain unchanged.

6.23 Attributes

For clause 27 "Attributes" in ETSI ES 201 873-1 (V4.17.1) [i.1] the following changes are proposed:

- Withdrawal of the **`display`** attribute (including dynamic configuration).
- Removal of code-block-level attributes referencing inner definitions.
  Attributes that are declared on a code block but bound to individual definitions via DefinitionRef or AllRef are removed from the core language. This referencing style will remain only for fields within constructive types and values (FieldReference), where no alternative syntactical mechanism exists.
  As a consequence, attributes for definitions may only be specified directly on the definition itself. This simplifies the attribute model, improves readability, and prevents inconsistencies arising from attributes being specified in multiple locations.
- Support for multiple encodings will be deleted, including Dynamic configuration of encoding used by ports that defines the **`setencode`** operation.
- Operators used for retrieving attribute values will be moved to ETSI ES 201 873-12 (V5.1.1). These operators are primarily relevant for conformance testing and are not essential to the fundamental semantics of TTCN-3.

6.24 Annex A (normative): BNF and static semantics

The core language document ETSI ES 201 873-1 (V5.1.1) will include an annex with BNF rules and static semantics descriptions.

6.25 Annex B (normative): Matching values

The technical contents of Annex B in ETSI ES 201 873-1 (V4.17.1) [i.1] will remain unchanged. The contents of this annex may be merged with the corresponding clause in ETSI ES 201 873-1 (V5.1.1).

6.26 Annex C (normative): Predefined TTCN-3 functions

For clause Annex C in ETSI ES 201 873-1 (V4.17.1) [i.1] the following changes are proposed:

- The annex will remain normative.
- All existing functions will be kept (some of them will be updated, see below).
- Void clauses will be removed.
- The log statement and external action operations will become predefined functions and will be moved to this clause.
- The **`lengthof`** function will be kept as it is, retaining most of its functionality. Possible grey areas will be covered by additional rules. More examples will be added if needed.
- The **`sizeof`** function will be revived with a new purpose: to return the "allocated" number of TTCN-3 language elements in a TTCN-3 list type entity (record of, set of, string etc. values or templates). The **`lengthof`** and **`sizeof`** will produce different results for entities that contain uninitialized values or matching symbols inside values such as '*'.
- Because the main character string type in TTCN-3 will be changed to **`string`**, new conversion functions for this type will be added. Old conversion functions for the legacy type **`charstring`** will be kept as there are many use cases for requiring them.

6.27 Annex D (normative): Preprocessing macros

The technical contents of Annex D in ETSI ES 201 873-1 (V4.17.1) [i.1] will remain unchanged.

6.28 Annex E (informative): Library of Useful Types

The technical contents of Annex E in ETSI ES 201 873-1 (V4.17.1) [i.1] will remain unchanged.

6.29 Annex F (informative): Operations on TTCN-3 active objects

The technical contents of Annex F in ETSI ES 201 873-1 (V4.17.1) [i.1] will be moved to ETSI TR 104 873 (V5.1.1).

7 Features in extensions

7.0 General

This clause lists the features that became part of ETSI ES 201 873-12 (V5.1.1).

Clause 7.1 lists the features that are moved from the core language to ETSI ES 201 873-12 (V5.1.1), while the rest of the clauses list the features that will be moved from the current extensions (ETSI ES 202 781 [i.8] to ETSI ES 203 022 [i.14] and ETSI ES 202 789 [i.16]) to ETSI ES 201 873-12 (V5.1.1).

7.1 Features moved from TTCN-3 core to extensions

7.1.0 General

This clause provides a list of TTCN-3 concepts and features which are moved from the core language ETSI ES 201 873-1 (V4.17.1) [i.1] to ETSI ES 201 873-12 (V5.1.1).

7.1.1 The concept of procedure-based communication

The whole concept of procedure-based communication (see clause 22.3 in ETSI ES 201 873-1 (V4.17.1) [i.1] will be moved to ETSI ES 201 873-12 (V5.1.1). This change will also have an effect on several additional clauses. Those parts of these clauses that are related to the concept of procedure-based communication will also be moved to ETSI ES 201 873-12 (V5.1.1). The affected clauses in ETSI ES 201 873-1 (V4.17.1) [i.1] are the following:

- 6.2.9 "Communication port types"
- 6.3.4 "Type compatibility of communication and connection operations"
- 9.1 "Communication ports"
- 9.2 "Test system interface"
- 14 "Declaring procedure signatures"
- 15.6.4 "Referencing signature parameters"
- 20.1 "The snapshot mechanism"
- 20.2 "The Alt statement"

NOTE: In the present document, these changes will not be separately indicated in the affected clauses, they will contain only the additional changes.

7.1.2 Automatic type

The concept of Automatic type (see clause 6.5 in ETSI ES 201 873-1 (V4.17.1) [i.1]) will be moved to ETSI ES 201 873-12 (V5.1.1) without the following exception that will remain in the core language:

- The automatic typing will only be allowed only if on the right-hand side of the assignment there is a reference or a literal.

NOTE: This can also be applied to the cycle variable of a counter loop.

7.1.3 Shift operators

Shift operators (see clause 7.1.6 in ETSI ES 201 873-1 (V4.17.1) [i.1]) will be moved to ETSI ES 201 873-12 (V5.1.1).

7.1.4 Rotate operators

Rotate operators (see clause 7.1.7 in ETSI ES 201 873-1 (V4.17.1) [i.1]) will be moved to ETSI ES 201 873-12 (V5.1.1).

7.1.5 The not-implemented function

The not-implemented function (see clause 16.1.6 in ETSI ES 201 873-1 (V4.17.1) [i.1]) will be moved to ETSI ES 201 873-12 (V5.1.1).

7.1.6 The Label statement

The Label statement (see clauses 18 and 19.7 in ETSI ES 201 873-1 (V4.17.1) [i.1]) will be moved to ETSI ES 201 873-12 (V5.1.1).

7.1.7 The Goto statement

The Goto statement (see clauses 18 and 19.8 in ETSI ES 201 873-1 (V4.17.1) [i.1]) will be moved to ETSI ES 201 873-12 (V5.1.1).

7.1.8 Call test component behaviour operation

The Call test component behaviour operation (see clause 21.3.10 in ETSI ES 201 873-1 (V4.17.1) [i.1]) will be moved to ETSI ES 201 873-12 (V5.1.1).

7.1.9 Retrieving attribute values

The Retrieving attribute values operations (see clause 27.8 in ETSI ES 201 873-1 (V4.17.1) [i.1]) will be moved to ETSI ES 201 873-12 (V5.1.1). These operators are primarily relevant for conformance testing and are not essential to the fundamental semantics of TTCN-3.

7.2 Features moved from TTCN-3 extension packages to the new TTCN-3 extensions document

7.2.0 General

The following list includes the handling of the TTCN-3 extensions with respect to the major new release of TTCN-3. The language extensions are:

- ETSI ES 202 781 [i.8];
- ETSI ES 202 782 [i.9];
- ETSI ES 202 784 [i.10];
- ETSI ES 202 785 [i.11];
- ETSI ES 202 786 [i.12];
- ETSI ES 203 790 [i.13];
- ETSI ES 203 022 [i.14];
- ETSI ES 202 789 [i.16].

7.2.1 TTCN-3 language extension: Configuration and Deployment Support

Configuration and deployment support (ETSI ES 202 781 [i.8]) will not be supported in the future.

7.2.2 TTCN-3 language extension: TTCN-3 Performance and Real-time Testing

For Performance and real-time testing (ETSI ES 202 782 [i.9]):

- the predefined function **`wait`**; and
- the **`timestamp`** value redirection in **`receive`** operations

will be moved to ETSI ES 201 873-12 (V5.1.1).

None of the other features of this extension will be supported in the future.

7.2.3 TTCN-3 language extension: Advanced Parameterization

For Advanced Parameterization (ETSI ES 202 784 [i.10]):

- type parameterization will be supported in ETSI ES 201 873-12 (V5.1.1), including default type parameters and nested actual type parameters;
- **all** type references in ports and value parameterization will not be supported in the future.

7.2.4 TTCN-3 language extension: Behaviour Types

A simplified version of the TTCN-3 language extension behaviour types (ETSI ES 202 785 [i.11]) will be supported in ETSI ES 201 873-12 (V5.1.1).

- In the function, altstep, and testcase behaviour types definition the following parameters will not be allowed:

  `[ "<" { FormalTypePar [","] } ">" ]`

  `"(" [ { ( FormalValuePar | FormalTimerPar | FormalTemplatePar | FormalPortPar ) [","] } ] ")"`

- Deferred behaviour types will not be supported in the future.

7.2.5 TTCN-3 language extension: Support of interfaces with continuous signals

The TTCN-3 language extension Support of interfaces with continuous signals (ETSI ES 202 786 [i.12]) will not be supported in the future.

7.2.6 TTCN-3 language extension: Extended TRI

The TTCN-3 language extension Extended TRI (ETSI ES 202 789 [i.16]) will not be supported in the future. Parts of this extension will be moved to ETSI ES 201 873-13 (V5.1.1)".

7.2.7 TTCN-3 language extension: Advanced Matching

From the TTCN-3 Language extension Advanced Matching (ETSI ES 203 022 [i.14]), the Dynamic matching will be moved to ETSI ES 201 873-12 (V5.1.1).

None of the other features of this extension will be supported in the future.

7.2.8 TTCN-3 language extension: Object-Oriented Features

The technical contents of the TTCN-3 language extension Object-Oriented Features (ETSI ES 203 790 [i.13]) will be supported in the future.

It has not yet been decided whether ETSI ES 203 790 [i.13] will remain a separate document or be integrated into ETSI ES 201 873-12 (V5.1.1).

8 Changes of features

8.0 General

This clause highlights the proposed changes on the core language, mentioned in clause 6 of the present document.

Clause 8.1 lists the proposed changes that are backward-compatible, while clause 8.2 lists the proposed changes that are backward-incompatible.

8.1 Backward-compatible changes

8.1.0 General

This clause lists the proposed backward-compatible changes to the TTCN-3 core language.

8.1.1 Introduction of 'root scope'

To simplify scoping rules for predefined functions (see clauses C.1 to C.6 in ETSI ES 201 873-1 (V4.17.1) [i.1]) and Useful TTCN-3 types (see clause E.2 in ETSI ES 201 873-1 (V4.17.1) [i.1]) a 'root' scope will be introduced. In the hierarchy of scope units 'root' scope will be above the 'module definitions part' scope unit (see Figure 2 in ETSI ES 201 873-1 (V4.17.1) [i.1], therefore the definitions in the 'root' scope can be used in all scope units of all TTCN-3 modules.

NOTE: The rules for Uniqueness of identifiers (see clause 5.2.2 in ETSI ES 201 873-1 (V4.17.1) [i.1]) will also be applied for the definitions in the 'root' scope. This means that an identifier in the 'root' scope cannot be reused as an identifier in any declaration in any lower level of scope unit.

8.1.2 Module parameters

All parameterization (see clause 5.4 in ETSI ES 201 873-1 (V4.17.1) [i.1]) in TTCN-3 takes effect dynamically at runtime, except for the module parameters that are supplied by the test environment at the beginning of the execution, and then they do not change their values during test execution. Because of this different nature, the module parameters will be defined as integral part of the Modules, but the technical content of module parameterization will be kept.

8.1.3 Bitstrings, Hexstrings and Octetstrings

Values of types **`bitstring`**, **`hexstring`** and **`octetstring`** (see clause 6.1.1 in ETSI ES 201 873-1 (V4.17.1) [i.1]) will not only be denoted by the uppercase 'B, 'H, 'O characters, but their lowercase equivalents 'b, 'h, 'o will also be enabled, like: '0101'b, 'ABCD'h and 'ABCD'o.

8.1.4 Character strings

The default string type will be the current **`universal charstring`** type. The new and shorter **`string`** will be introduced for it, but for backward compatibility reasons the **`universal charstring`** keyword will also be kept.

The current **`charstring`** type will be defined as a subtype of the new **`string`** type. This will allow to simplify the descriptions in other chapters, but will not cause any difference for the user, the **`universal charstring`**s and charstrings can be used in the same way as of now (see clause 6.1.1 in ETSI ES 201 873-1 (V4.17.1) [i.1]), and the keyword **`charstring`** will still remain.

8.1.5 Records and sets of single types and Arrays

For the Record and Set of elements of the same type (see clause 6.2.3 in ETSI ES 201 873-1 (V4.17.1) [i.1]) the rules describing how many elements a partially initialized record of/set of has, and what happens if a value on the right-hand side of an assignment has different number of elements than that of on the left-hand side will be clarified.

- If a record of/set of value has uninitialized element(s), they will count into the number of elements of that value, even if they are at the end:

```
type record of integer RoI;
var RoI v_myVar1 := { 0, 1, -, - }        // v_myVar1 will have 4 elements,
                                          // last 2 are uninitialized
var RoI v_myVar2 := { 0, -, 1, -, - }     // v_myVar1 will have 5 elements,
                                          // second and last 2 are uninitialized
var RoI v_myVar3 := {
    [0] := 0,
    [1] := 1,
    [2] := -
}                                         // v_myVar1 will have 3 elements, last is
                                          // uninitialized
```

- In an assignment, the left-hand side will be the same as the right-hand side:

```
v_myVar1 := v_myVar2;       // v_myVar1 will have 5 elements
v_myVar1 := {1,2,-};        // v_myVar1 will have 3 elements {1,2,1}
v_myVar1 := v_myVar3;          // v_myVar1 will have 3 elements
                            // {0,1, <uninitialized>}
```

For arrays (see clause 6.2.7 in ETSI ES 201 873-1 (V4.17.1) [i.1]), it will be clarified that arrays can be used in TTCN-3 as a shorthand notation to specify record of types with a fixed length restriction.

8.1.6 Communication port types

Since procedure-based communication will be moved to ETSI ES 201 873-12 (V5.1.1), in the core language only the message-based port type (see clause 6.2.9 in ETSI ES 201 873-1 (V4.17.1) [i.1]) will remain. In message-based port type definitions the keyword **message** will be optional (it will be kept only for backward compatibility). The following two definitions will be identical:

```
type port PortTypeIdentifier message "{"
...
"}"

type port PortTypeIdentifier "{"
...
"}"
```

8.1.7 Automatic type

The concept of Automatic type (see clause 6.5 in ETSI ES 201 873-1 (V4.17.1) [i.1]) will be moved to ETSI ES 201 873-12 (V5.1.1) without the following exception that will remain in the core language:

- The automatic typing will only be allowed only if on the right-hand side of the assignment there is a reference or a literal.

NOTE: This can also be applied to the cycle variable of a counter loop.

8.1.8 Declaring templates

The proposed backward-compatible modifications related to templates (see clause 15 in ETSI ES 201 873-1 (V4.17.1) [i.1]) are the following:

- Templates without a (possibly empty) parameter list are static templates. They will be evaluated only once, at the place of their declaration.

NOTE 1: A static template retains the value was assigned to it at the initialization and will not be re-evaluated later at its usage, even when invoking functions or dynamic templates directly or indirectly.

- Templates with a (possibly empty) parameter list are dynamic templates. A dynamic template is evaluated at every invocation, except for being invoked directly or indirectly from a static template; in this case it is invoked only once, at the initialization of that static template that directly invokes it. If the parameter list of a dynamic template is empty, then at invocation the usage of ' ( ) ' is optional. But if all the parameters of a dynamic template have default values and at the invocation all these default values wanted to be used, then the usage of the ' ( ) ' remains mandatory.

NOTE 2: There is an additional, backward-incompatible modification on Templates: Dynamic templates can only be global templates. See clauses 6.13 and 8.2.5 of the present document.

NOTE 3: The functionality of the proposed dynamic template is the same as the functionality of the fuzzy templates is in ETSI ES 201 873-1 (V4.17.1) [i.1].

A new **`template`** attribute will be introduced. The attribute can control how the global templates in a module will be interpreted. The **`template`** attribute will have two predefined string values:

- `"static"`: only those templates will be dynamic that have a (possibly empty) parameter list defined (the behaviour described above).
- `"dynamic"`: all the global templates will be treated as dynamic.

Only modules can have **`template`** attribute, and the default value is `"static"`.

```
module M {

...

} with { template "dynamic" }
```

EXAMPLE 1:

```
// in a module definitions part;
// "static" template attribute is defined for the module

template  float t_static := rnd();    /* static template, initialized by random number 1 */

template  float t_dynamic() := rnd(); /* dynamic template, no initialization here */

template  float t_static2 := t_dynamic(); /* static template, initialized by random number 2 */

// e.g. in a testcase

P_PT.send(t_static);            /* random number 1 will be sent, template will not be evaluated
here */

P_PT.send(t_static);            /* random number 1 will be sent, template will not be evaluated
here */

P_PT.send(t_dynamic());      /* template will be evaluated here, random number 3 will be sent;
                                   alternative syntax:
                                   P_PT.send(t_dynamic);
                                   is also possible */

P_PT.send(t_dynamic());      /* template will be evaluated again here, random number 4 will be sent
*/

P_PT.send(t_static2);          /* random number 2 will be sent, template will not be evaluated
here, because if a dynamic template or a function is invoked directly or indirectly on the right-
hand side of a static template declaration, it is only evaluated once, at the declaration of the
directly invoking template, therefore neither t_dynamic() nor rnd() will be evaluated here */
```

EXAMPLE 2:

```
// in a module definitions part;
// "dynamic" template attribute is defined for the module

template  float t_static := rnd();       /* because of the "dynamic" template attribute of the
     module, this template will be treated as dynamic, so no initialization here */

template  float t_dynamic() := rnd();    /* dynamic template, no initialization here */

template  float t_static2 := t_dynamic(); /* because of the "dynamic" template attribute of the
     module, this template will be treated as dynamic, so no initialization here */
```

```
// e.g. in a testcase

P_PT.send(t_static);            /* template will be evaluated here, random number 1 will be sent */

P_PT.send(t_static);            /* template will be evaluated again here, random number 2 will be
sent */

P_PT.send(t_dynamic());      /* template will be evaluated here, random number 3 will be sent */

P_PT.send(t_dynamic());      /* template will be evaluated again here, random number 4 will be sent
*/

P_PT.send(t_static2);          /* template will be evaluated here, random number 5 will be sent */
```

8.1.9 Check operation

Because the procedure-based communication operations (see clause 22.3 in ETSI ES 201 873-1 (V4.17.1) [i.1]) will be moved to ETSI ES 201 873-12 (V5.1.1), the syntax of the Check operation (see clause 22.4 in ETSI ES 201 873-1 (V4.17.1) [i.1]) can be simplified, it will be optional to explicitly refer to the **receive** operation:

```
(check(receive(matching template))
```

can be simplified to:

```
check(matching template)
```

For backward compatibility reasons, the current syntax will also be kept.

8.1.10 Number of elements of lists (sizeof predefined function)

The functionality of the **sizeof** predefined function (see clause C.2.2 in ETSI ES 201 873-1 (V4.17.1) [i.1]) will be extended. It will be applicable not only for record or set type values and templates, but also for string and record of or set of values or templates.

If it is applied for values and templates of:

- record or set type - the current behaviour will remain;
- string types - it will be the same as **lengthof**;
- record of or set of types - it will be similar to **lengthof**, but it will also count the possible last, but not initialized elements, as well (see also clause 6.5.3 of the present document).

```
var record of integer v_roi := {1,2,-,-}

var integer v_length := lengthof(v_roi);    // v_length becomes 2

var integer v_size := sizeof(v_roi);     // v_size becomes 4
```

NOTE: The **lengthof** predefined function will remain the same as it is defined in clause C.2.1 in ETSI ES 201 873-1 (V4.17.1) [i.1].

8.1.11 Conversion functions

Because the **universal charstring** becomes the 'default' string type, all the to/from character string conversion functions (see clause C.1 in ETSI ES 201 873-1 (V4.17.1) [i.1]) will have to/from universal string equivalents.

Currently the following conversion functions of this kind are missing: int2unistr, bit2unistr, hex2unistr, oct2unichar, unistr2int, unistr2hex, unistr2oct, unistr2float.

8.2 Backward-incompatible changes

8.2.0 General

This clause lists the proposed backward-incompatible changes to the TTCN-3 core language.

8.2.1 Fuzzy and lazy values and templates and lazy evaluation

The concepts of Fuzzy and lazy values and templates (see clause 4 in ETSI ES 201 873-1 (V4.17.1) [i.1]) and lazy evaluation (see clause 4 in ETSI ES 201 873-1 (V4.17.1) [i.1]) will be removed. This change will also have an effect on several additional clauses. Those parts of these clauses that are related to the concepts Fuzzy and lazy values and templates or to lazy evaluation will also be removed. The affected clauses in ETSI ES 201 873-1 (V4.17.1) [i.1] are the following:

- 3.1 "Terms"
- 5.4.1 "Formal parameters" and all of its clauses
- 5.4.2 "Actual parameters"
- 6.2.1.0 "Record type and values - General"
- 11 "Declaring variables" and all of its sub-clauses
- 15.0 "Declaring templates - General"
- 15.3 "Global and local templates"
- 15.5 "Modified templates"
- 16.1.4 "Invoking functions from specific places"
- 19.1.1 "Basic assignments"
- 19.4.2 "The range-based loop"
- 21.3.5 "The Alive operation"
- 21.3.6 "The Running operation"
- 21.3.7 "The Done operation"
- 21.3.8 "The Killed operation"
- 22.2.2 "The Receive operation"
- 22.2.3 "The Trigger operation" (this operation will be deleted, see clause 9.5 of the present document)
- 22.3.2 "The Getcall operation" (this operation will be moved to ETSI ES 201 873-12 (V5.1.1), see clause 7.1.1 of the present document)
- 22.3.4 "The Getreply operation" (this operation will be moved to ETSI ES 201 873-12 (V5.1.1), see clause 7.1.1 of the present document)
- 22.3.6 "The Catch operation" (this operation will be moved to ETSI ES 201 873-12 (V5.1.1), see clause 7.1.1 of the present document)
- 22.4 "The Check operation"

NOTE: In the present document, these changes will not be separately indicated in the affected clauses, they will contain only the additional changes.

8.2.2 Embedded fields

The concept of embedded fields (see clause 6.2.1.4 in ETSI ES 201 873-1 (V4.17.1) [i.1]) will be deleted.

8.2.3 Array dimensions specified using ranges

The possibility of defining arrays by specifying its lower and upper index (see clause 6.2.7 in ETSI ES 201 873-1 (V4.17.1) [i.1]), like:

```
type integer MyArrayType2[2 .. 5];
```

will be removed, indexes of the arrays will always start by 0.

8.2.4 Importing from modules

Importing from modules (see clauses 8.2.3 in ETSI ES 201 873-1 (V4.17.1) [i.1]) will be significantly simplified, only **`import all`** will be supported, that is the import of single definitions (module parameters, user defined types, signatures, constants, templates, functions, external functions, altsteps, test cases, and imports) will no longer be possible.

If an imported definition has attributes (defined by means of a **`with`** statement) then the attributes will also be imported.

If the imported module has no attribute, then the **`with`** statement may be attached to the **`import`** statement, and this **`with`** statement cannot have an **`override`** directive.

NOTE: The attributes specified in this **`with`** statement are applied to all imported definitions that had no attributes defined, that is during import, the existing attributes cannot be changed.

If a module is imported - explicitly or implicitly - more than once, then the attributes defined for the module will be the mutually identical.

The rules on import of attributes will be simplified: attributes may be attached only when the original module does not include them and if present, the attributes attached to import of a particular module will be the same through the whole test suite.

8.2.5 Templates

The proposed backward-incompatible modifications related to templates (see clause 15 in ETSI ES 201 873-1 (V4.17.1) [i.1]) is the following:

Dynamic templates (templates with a possibly empty parameter list) can only be global templates.

NOTE: There are additional, backward-compatible modifications on Templates, see clauses 6.13 and 8.1.8 of the present document.

8.2.6 Range-based loops

The specification of the range-based loops (see clause 19.4.2 in ETSI ES 201 873-1 (V4.17.1) [i.1]) will be clarified.

In order to be able to iterate over partially initialized range expressions, and to keep the existing functionalities of the assignments (see clause 19.1 in ETSI ES 201 873-1 (V4.17.1) [i.1]), the specification of the range-based loops will be clarified at two places:

- The iteration variable is only be defined locally in the loop.
- If the range expression contains uninitialized elements (regardless whether the uninitialized element is before or after the last initialized element), they are also iterated over.

8.2.7 Interleave statement

For the interleave statement (see clause 20.4 in ETSI ES 201 873-1 (V4.17.1) [i.1]) the following semantical simplification is proposed:

- Complexity is introduced into the interleave statement by allowing reception statements in statement blocks of alternatives in interleave statements. Each reception statement adds to the complexity of interleaving by implicitly introducing (nested) alt statements.

The proposal is to disallow reception statements in the highlighted ***StatementeBlock*** below:

Syntactical Structure

```
interleave [ @nodefault ] "{"
{ "[]" ( TimeoutStatement |
              ReceiveStatement |
              TriggerStatement |
              CheckStatement |

              DoneStatement |
              KilledStatement ) StatementBlock
}
"}"
```

8.2.8 The Trigger operation

The Trigger operation (see clause 22.2.3 in ETSI ES 201 873-1 (V4.17.1) [i.1]) will be deleted.

8.2.9 With statement

The with statement (see clause 27.2 in ETSI ES 201 873-1 (V4.17.1) [i.1]) will be simplified.

The **display** attribute will be deleted.

The optional *DefinitionRef* and *AllRef* attributes will be deleted.

8.2.10 Display attributes

Because Display attributes (see clause 27.3 in ETSI ES 201 873-1 (V4.17.1) [i.1]) is not reported to be used, this feature will be deleted.

8.2.11 Encoding attributes and Dynamic configuration of encoding used by ports

The multiple encoding feature (see clause 27.4 in ETSI ES 201 873-1 (V4.17.1) [i.1]) and the Dynamic configuration of encoding used by ports (see clause 27.9 in ETSI ES 201 873-1 (V4.17.1) [i.1]) defining the **`setencode`** operation will be deleted.

9 Deleted features

9.0 General

This clause highlights the proposed features mentioned in clause 6 of the present document that will not be supported in the future by TTCN-3 (V5.1.1).

9.1 Fuzzy and lazy values and templates and lazy evaluation

The concepts of Fuzzy and lazy values and templates (see clause 4 in ETSI ES 201 873-1 (V4.17.1) [i.1]) and lazy evaluation (see clause 4 in ETSI ES 201 873-1 (V4.17.1) [i.1]) will be removed. This change will also have an effect on several additional clauses. Those parts of these clauses that are related to the concepts Fuzzy and lazy values and templates or to lazy evaluation will also be removed. The affected clauses in ETSI ES 201 873-1 (V4.17.1) [i.1] are the following:

- 3.1 "Terms"
- 5.4.1 "Formal parameters" and all of its clauses
- 5.4.2 "Actual parameters"
- 6.2.1.0 "Record type and values - General"
- 11 "Declaring variables" and all of its clauses
- 15.0 "Declaring templates - General"
- 15.3 "Global and local templates"
- 15.5 "Modified templates"
- 16.1.4 "Invoking functions from specific places"
- 19.1.1 "Basic assignments"
- 19.4.2 "The range-based loop"
- 21.3.5 "The Alive operation"
- 21.3.6 "The Running operation"
- 21.3.7 "The Done operation"
- 21.3.8 "The Killed operation"
- 22.2.2 "The Receive operation"
- 22.2.3 "The Trigger operation" (this operation will be deleted, see clause 9.5 of the present document)
- 22.3.2 "The Getcall operation" (this operation will be moved to ETSI ES 201 873-12 (V5.1.1), see clause 7.1.1 of the present document)
- 22.3.4 "The Getreply operation" (this operation will be moved to ETSI ES 201 873-12 (V5.1.1), see clause 7.1.1 of the present document)
- 22.3.6 "The Catch operation" (this operation will be moved to ETSI ES 201 873-12 (V5.1.1), see clause 7.1.1 of the present document)
- 22.4 "The Check operation"

NOTE: In the present document, these changes will not be separately indicated in the affected clauses, they will contain only the additional changes.

9.2 Embedded fields

The concept of embedded fields (see clause 6.2.1.4 in ETSI ES 201 873-1 (V4.17.1) [i.1]) will be deleted.

9.3 Array dimensions specified using ranges

The possibility of defining arrays by specifying its lower and upper index (see clause 6.2.7 in ETSI ES 201 873-1 (V4.17.1) [i.1]), like:

```
type integer MyArrayType2[2 .. 5];
```

will be removed, indexes of the arrays will always start by 0.

9.4 Importing of single definitions from modules

Importing from modules (see clauses 8.2.3 in ETSI ES 201 873-1 (V4.17.1) [i.1]) will be significantly simplified, only **`import all`** will be kept, that is the **`import`** of single definitions (module parameters, user defined types, signatures, constants, templates, functions, external functions, altsteps, test cases, and imports) will no longer be possible.

NOTE: See clause 8.2.4 of the present documents for additional changes on importing from modules.

9.5 The Trigger operation

The Trigger operation (see clause 22.2.3 in ETSI ES 201 873-1 (V4.17.1) [i.1]) will be deleted.

9.6 With statement

The with statement (see clause 27.2 in ETSI ES 201 873-1 (V4.17.1) [i.1]) will be simplified:

- The **display** attribute will be deleted.
- The optional *DefinitionRef* and *AllRef* attributes will be deleted.

9.7 Display attributes

Because Display attributes (see clause 27.3 in ETSI ES 201 873-1 (V4.17.1) [i.1]) is not reported to be used, this feature will be deleted.

9.8 Encoding attributes and Dynamic configuration of encoding used by ports

The multiple encoding feature (see clause 27.4 in ETSI ES 201 873-1 (V4.17.1) [i.1]) and in connection with it the Dynamic configuration of encoding used by ports (see clause 27.9 in ETSI ES 201 873-1 (V4.17.1) [i.1]) defining the **`setencode`** operation will be deleted.

9.9 Deprecated language features

All deprecated language features (see Annex G in ETSI ES 201 873-1 (V4.17.1) [i.1]) will be deleted.

This includes the following:

- Group style definition of module parameters (see clause G.1 in ETSI ES 201 873-1 (V4.17.1) [i.1]).
- Using **`all`** in port type definitions (see clause G.3 in ETSI ES 201 873-1 (V4.17.1) [i.1]).

- sizeof for length of lists (see clause G.4 in ETSI ES 201 873-1 (V4.17.1) [i.1]).

NOTE: only the deprecated meaning of the **`sizeof`** function will be deleted, its current meaning specified in clause C.2.2 in ETSI ES 201 873-1 (V4.17.1) [i.1] will remain, and its applicability will be extended (see clause 8.1.10 of the present document).

- Mixed ports (see clause G.6 in ETSI ES 201 873-1 (V4.17.1) [i.1]).
- Assignment of less restrictive templates to more restrictive templates (see clause G.13 in ETSI ES 201 873-1 (V4.17.1) [i.1]).
- Mixing case and case else branches in select statements (see clause G.14 in ETSI ES 201 873-1 (V4.17.1) [i.1]).
- Partially initialized global and local templates (see clause G.15 in ETSI ES 201 873-1 (V4.17.1) [i.1]).
- Template modification of less restrictive templates to more restrictive templates (see clause G.16 in ETSI ES 201 873-1 (V4.17.1) [i.1]).
- Unrestricted template fields, alternatives and elements (see clause G.17 in ETSI ES 201 873-1 (V4.17.1) [i.1]).

10 Working procedures

10.1 General

This clause contains a list of the agreed-upon work processes that are applied during the development of ETSI ES 201 873-1 (V5.1.1), ETSI ES 201 873-12 (V5.1.1), ETSI ES 201 873-13 (V5.1.1), ETSI ES 201 873-7 (V5.1.1), ETSI ES 201 873-9 (V5.1.1), ETSI ES 201 873-10 (V5.1.1), ETSI ES 201 873-11 (V5.1.1), ETSI TR 104 873 (V5.1.1), and ETSI TR 104 081 (V5.1.1) (the present document).

10.2 CR process

The GitLab issue tracker is used to request changes. Change Requests (CRs) are the basis for managing changes and tracking discussions on the standard.

The detailed operating procedure of submitting a CR is explained at https://labs.etsi.org/rep/mts/ttcn3/standard/-/blob/main/docs/how-to-make-a-change-request.md. The general procedure is the following:

- **Perquisite:** An EOL or an Individual Account.
- **Steps:**
  1) Sign in.
  2) Assure that the same or a similar request has not been submitted before.
  3) Open issue form by clicking on "New issue".
  4) Add a title that describes your change request concisely.
  5) Choose a description template from the dropdown box that fits your change request the most.
  6) Fill in the details.
  7) Add attachments if necessary.
- **Output:** A new CR.

History

Version	Date	Status
V5.1.1	April 2026	Publication