ETSI ES 202 784 V1.6.1 (2017-04)

Methods for Testing and Specification (MTS);

The Testing and Test Control Notation version 3;

TTCN-3 Language Extensions: Advanced Parameterization

**ETSI Standard**

Reference

RES/MTS-202784 ed161AdvParam

Keywords

conformance, 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 only prevailing document is the print of the Portable Document Format (PDF) version kept on a specific network drive within ETSI Secretariat.

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.

© European Telecommunications Standards Institute 2017.

All rights reserved.

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

Contents

Intellectual Property Rights 4

Foreword 4

Modal verbs terminology 4

1 Scope 5

2 References 5

2.1 Normative references 5

2.2 Informative references 6

3 Definitions and abbreviations 6

3.1 Definitions 6

3.2 Abbreviations 6

4 Package conformance and compatibility 7

5 Package concepts for the core language 7

5.1 Extension to ETSI ES 201 873-1, clause 4 (Introduction) 7

5.2 Extension to ETSI ES 201 873-1, clause 5 (Basic language elements) 7

5.3 Extension to ETSI ES 201 873-1, clause 6 (Types and values) 11

5.4 Extension to ETSI ES 201 873-1, clause 8 (Modules) 12

5.5 Extension to ETSI ES 201 873-1, annex A (BNF and static semantics) 12

6 Package semantics 14

6.0 General 14

6.1 Extension to ETSI ES 201 873-4, clause 6 (Restrictions) 14

7 TRI and Extended TRI extensions for the package 14

7.1 Extension to ETSI ES 201 873-5 14

7.2 Extension to ETSI ES 202 789, clause 7 (TRI extensions for the package) 14

8 TCI extensions for the package 15

8.1 Extension to ETSI ES 201 873-6, clause 7 (TTCN 3 control interface and operations) 15

8.2 Extension to ETSI ES 201 873-6, clause 8 (Java™ language mapping) 17

8.3 Extension to ETSI ES 201 873-6, clause 9 (ANSI C language mapping) 18

8.4 Extension to ETSI ES 201 873-6, clause 10 (C++ language mapping) 19

8.5 Extension to ETSI ES 201 873-6, annex A (IDL Specification of TCI) 20

9 Package Extensions for the use of ASN.1 with TTCN-3 21

9.1 Extension to ETSI ES 201 873-7, clause 10 (Parameterization in ASN.1) 21

10 Documentation extensions for the package 25

10.1 Extension to ETSI ES 201 873-10, clause 6 (Tagged paragraphs) 25

10.2 Extension to ETSI ES 201 873-10, annex A (where Tags can be used) 25

History 26

# Intellectual Property Rights

IPRs essential or potentially essential to the present document 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.

# Foreword

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

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

The present document relates to the multi-part standard ETSI ES 201 873 covering the Testing and Test Control Notation version 3, as identified below:

Part 1: "TTCN‑3 Core Language";

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".

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

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

This package defines:

* Value parameters of types.
* Type parameterization.

While the design of TTCN-3 package has taken into account the consistency of a combined usage of the core language with a number of packages, the concrete usages of and guidelines for this package in combination with other packages is outside the scope of the present document.

# 2 References

## 2.1 Normative references

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

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

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

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

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

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

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

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

[5] 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".

[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".

[7] ISO/IEC 9646-1: "Information technology - Open Systems Interconnection - Conformance testing methodology and framework; Part 1: General concepts".

[8] 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".

[9] 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".

[10] Recommendation ITU-T X.683: "Information technology - Abstract Syntax Notation One (ASN.1): Parameterization of ASN.1 specifications".

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

## 2.2 Informative references

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

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

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

Not applicable.

# 3 Definitions and abbreviations

## 3.1 Definitions

For the purposes of the present document, the terms and definitions given in ETSI ES 201 873-1 [1], ETSI ES 201 873‑4 [2], ETSI ES 201 873-5 [3], ETSI ES 201 873-6 [4], ETSI ES 201 873-7 [5], ETSI ES 201 873‑10 [6], ISO/IEC 9646-1 [7] and the following apply:

**subtype compatibility:** A type "A" has a subtype compatibility to another type "B" if all of the values of type A are type compatible with type "B".

NOTE: All subtypes defined via subtype definition are subtype compatible with their supertype.

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

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

## 3.2 Abbreviations

For the purposes of the present document, the abbreviations given in ETSI ES 201 873-1 [1], ETSI ES 201 873‑4 [2], ETSI ES 201 873‑5 [3], ETSI ES 201 873-6 [4], ETSI ES 201 873-7 [5], ETSI ES 201 873-10 [6] and ISO/IEC 9646‑1 [7] apply.

# 4 Package conformance and compatibility

The package presented in the present document is identified by the package tag:

* "TTCN‑3:2014 Advanced Parameterization" - to be used with modules complying with the present document.

NOTE: This version of the package only extends the previous versions, identified with the package tag "TTCN‑3:2009 Advanced Parameterization", with the option of parameterizing objects - in addition to types - also with signatures. For this reason, modules not containing a formal or actual parameter of the kind signature are compatible with both versions.

For an implementation claiming to conform to this package version, all features specified in the present document shall be implemented consistently with the requirements given in the present document, in ETSI ES 201 873‑1 [1] and in ETSI ES 201 873‑4 [2].

The package presented in the present document is compatible to:

ETSI ES 201 873-1 [1], version 4.5.1;

ETSI ES 201 873-4 [2], version 4.5.1;

ETSI ES 201 873-5 [3], version 4.5.1;

ETSI ES 201 873-6 [4], version 4.5.1;

ETSI ES 201 873-7 [5], version 4.5.1;

ETSI ES 201 873-8 [8], version 4.5.1;

ETSI ES 201 873-9 [9], version 4.5.1;

ETSI ES 201 873-10 [6], version 4.5.1.

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

# 5 Package concepts for the core language

## 5.1 Extension to ETSI ES 201 873-1, clause 4 (Introduction)

The present package adds the following essential characteristic to TTCN-3:

* type parameterization.

## 5.2 Extension to ETSI ES 201 873-1, clause 5 (Basic language elements)

Clause 5.2.1 Scope of formal parameters

Add the following text:

Additionally, formal type parameters can be used as types of formal value parameters, return values, **runs on** and **system** clauses, where applicable.

Clause 5.4 Parameterization

Additionally, TTCN-3 supports type and signature parameterization.

Replace table 2 "Overview of parameterizable TTCN-3 objects" with the following table.

Table 2: Overview of parameterizable TTCN‑3 objects

|  |  |  |  |
| --- | --- | --- | --- |
| Keyword | Allowed kind of Parameterization | Allowed form of non-type Parameterization | Allowed types in formal non-type parameter lists |
| **Module** | Value parameterization | Static at start of run-time | all basic types, all user-defined types and **address** type. |
| **type**(notes 1 and 2) | Value parameterization, type parameterization, signature parameterization | Static at compile-time | all basic types, all user-defined types and **address** type. |
| **Template** | Value and template parameterization, type parameterization | Dynamic at run-time | all basic types, all user-defined types, **address** type, **template.** |
| **function**(note 1) | Value, template, port and timer parameterization, type parameterization, signature parameterization | Dynamic at run-time | all basic types, all user-defined types, **address** type, **component** type, **port** type, **default**, **template** and **timer**. |
| **altstep**(note 1) | Value, template, port and timer parameterization, type parameterization, signature parameterization | Dynamic at run-time | all basic types, all user-defined types, **address** type, **component** type, **port** type, **default**, **template** and **timer**. |
| **testcase**(note 1) | Value, template, port and timer parameterization, type parameterization, signature parameterization | Dynamic at run-time | all basic types and of all user-defined types, **address** type, **template**. |
| **Signature** | Value and template parameterization, type parameterization | Dynamic at run-time | all basic types, all user-defined types and **address** type, **component** type. |
| NOTE 1: Type and signature parameterization are always static at compile-time.NOTE 2: Only port and component types are parameterizable with signature formal parameters. |

Clause 5.4.1 Formal parameters

All types in TTCN-3 may be parameterized.

Clause 5.4.1.1 Formal parameters of kind value

In addition to the existing rules, TTCN-3 supports value parameterizations as follows:

* the value parameters of user-defined types shall be in parameters;
* the language element **signature** does not support *static* value parameterization.

Modify the text as follows:

Restriction a) is relaxed to:

1. Language elements which cannot be parameterized are: **const**, **var**, **timer**, **control**,**~~record of~~**~~,~~ **~~set of~~**~~,~~ **~~enumerated~~**~~,~~ **~~port~~**~~,~~ **~~component~~** ~~and sub-type definitions,~~ **group** and **import**.

Restriction e) is changed to:

e) The expression of formal parameter's default value has to be compatible with the type of the parameter. The expression may be any expression that is well-defined at the beginning of the scope of the parameterized entity, but shall not refer to other parameters of the same or any following parameter list.

Clause 5.4.1.2 Formal parameters of kind template

Restriction d) is changed to:

d) The default template instance has to be compatible with the type of the parameter. The template instance may be any template expression that is well-defined at the beginning of the scope of the parameterized entity, but shall not refer to other parameters in the same or any following parameter list.

Clause 5.4.1 Formal Parameters

Is extended by the following clause:

Clause 5.4.1.5 Formal parameters of kind type and signature

Type, template and behaviour definitions in TTCN-3 can have parameters of kind type.

***Syntactical Structure***

[ **in** ] [ *TypeIdentifier* **| type | signature** ]*TypeParIdentifier* [ *":=" ( Type | Signature* ) ]

***Semantic Description***

Types and signatures passed into a parameterized object can be used inside the definition of that object. This includes the usage as type of value, template, and port parameters, as type of return values and within **runs on** and **system** clauses of behaviour definitions.

Any type and signature parameterization shall be resolved statically.

Type and signature parameters shall be written in a separate parameter list, enclosed in angle brackets.

Parameters of type kind may have a default type, which is given by a type assigned to the parameter. Similarly, parameters of signature kind may own a default signature, which is identified by assigning a signature to the parameter.

The actual parameters of a type parameter can be required to be subtype compatible with a specific type. This is indicated by referring to a specific type in the formal parameter list instead of using the keyword **type**.

***Restrictions***

a) Formal type and signature parameters shall be in parameters, which can optionally be indicated by the optional keyword **in**.

b) When a TypeIdentifier is used to specify the kind of the formal parameter, the default types shall be subtype compatible with the type of the parameter. For type compatibility see ETSI ES 201 873-1 [1] TTCN-3 Core Language, clause 6.3. The default type shall not refer to other type parameters in the same parameter list.

c) Void.

d) Void.

***Examples***

 // Definition of a list and a check function

**type** **record of** T MyList <**in** **type** T>;
**function** isElement <**in** **type** T>(**in** MyList<T> list, **in** T elem) **return** **boolean** { … }

// Definition of a protocol message
**type** **record** Data<**in** **type** PayloadType> {
 Header hdr,
 PayloadType payload
}

// restricting the actual type parameters
// the function can create a component of a type that is an extension of CT.

**type** **component** CT { **timer** t\_guard };
**function** MyFunction <**in** CT Comp> (**in** **integer** p) **runs** **on** CT {
 **var** Comp c := Comp.**create**;
 :
};

 **function** MyIntegerFunction<**integer** I := **integer**>(I p) **return** I {

 // actual parameter for I must be subtype compatible to integer

 :

 }

Clause 5.4.1.1 Formal Parameters of kind value

Formal parameters with default values are additionally restricted by:

***Restrictions***

Replace the text as follows:

e) ~~The expression of the default value has to be compatible with the type of the parameter. The expression shall not refer to elements of the component type of the optional~~ **~~runs on~~** ~~clause. The expression shall not refer to other parameters of the same parameter list. The expression shall not contain the invocation of functions with a~~ **~~runs on~~** ~~clause.~~

e) The type of a value parameter with a default value shall not be a type parameter.

Clause 5.4.1.2 Formal Parameters of kind template

Formal parameters with default templates are additionally restricted by:

***Restrictions***

a) ~~Only~~ **~~function, testcase~~**~~,~~ **~~altstep~~** ~~and~~ **~~template~~** ~~definitions may have formal template parameters.~~

a) The type of a template parameter with a default template shall not be a type parameter.

Clause 5.4.2 Actual Parameters

Is extended by a new clause:

Clause 5.4.2.1 Actual type parameters

Types can be passed into parameterized TTCN-3 objects as actual type parameters. Signatures can be passed into parameterized TTCN-3 objects as actual signature parameters. Actual type and signature parameters can be provided both as a list in the same order as the formal parameters as well as in an assignment notation, explicitly using the associated formal parameter names.

***Syntactical Structure***

[( *Type* | *Signature* | *TypeParIdentifier* ) ":=" ] ( *Type | Signature* )

***Semantic Description***

Actual type parameters that are passed to formal type parameters shall be types or formal type parameters. Actual signature parameters that are passed to formal signature parameters shall be signatures or formal signature parameters. Any compatible type/signature can be passed as actual parameter, i.e. actual parameters are not limited to those types/signatures only known in the module containing the parameterized definition itself. Formal type and signature parameters passed as parameters are those from the enclosing scope unit.

An empty type/signature parameter list can be omitted in both the declaration and usage of that object.

***Restrictions***

a) When using list notation, the order of elements in the actual parameter list shall be the same as their order in the corresponding formal parameter list. For each formal parameter without a default type/signature there shall be an actual parameter. If a formal parameter with a default is followed by a formal parameter without a default, the actual parameter can be skipped by using dash "-" as actual parameter. If a formal parameter with a default is not followed by a parameter without a default, then the actual parameter can simply be omitted.

c) When using assignment notation, each formal parameter shall be assigned an actual parameter at most once. For each formal parameter without defaulttype or signature, there shall be an actual parameter. To use the default type or signature of a formal parameter, no assignment for this specific parameter shall be provided.

d) If a formal parameter was defined using a specific component type, then the actual parameter shall be compatible with the type of the formal parameter. For type compatibility of component types see ETSI ES 201 873-1 [1], clause 6.3.3.

e) Instantiating formal type or signature parameters with actual types/signatures shall result in valid TTCN-3.

EXAMPLE 1: Type parameterization

 **function** f <**in** **type** T> (**in** T a, in T b) **return** T {
 **return** a + b}

 **var** **integer** c := f<**integer**>(1, 2); // correct call, result 3

 **var** **integer** c := f<**boolean**>(**true**, **false**); // incorrect

EXAMPLE 2: Signature parameterization

 **type** **port** P <**signature** S> **procedure** { **inout** S }

 **function** g <**signature** S> (**integer** a, P<S> p) **return** **integer** {

 **var** **integer** result;

 p.**cal**l(s:{a}, 10.0) {

 [] p.**getrepl**y(s:{-}) **->** **value** result {}

 [] p.**catch**(s, exc) { **testcase**.**stop** }

 [] p.**catch**(**timeout**) { **testcase**.**stop** }

 }

 **return** result;

 }

 **template** **integer** exc := **?**;

 **signature** s1(**integer** a) **return** **integer** **exception**(**integer**);

 **signature** s2(**float** a) **return** **float** **exception**(**boolean**);

 **type** **component** C<**signature** S> {

 **port** P<S> p;

 }

 **function** h() **runs** **on** C<s1> {

 **var** **integer** r := g<s1>(1, p) // correct

 }

 **function** h2() **runs** **on** C<s2> {

 **var** **float** r := g<s2>(1, p) // incorrect; g returns integer and the actual call within

 // function g also returns integer.

 }

## 5.3 Extension to ETSI ES 201 873-1, clause 6 (Types and values)

Clause 6.1.2.1 Nested type definitions for field types

Nested type definitions shall not have formal value parameters and shall not have formal type or signature parameters.

Clause 6 Types and values

Is extended by the following clause:

Clause 6.2.9

In the Semantic Description part, add the following:

In the lists indicating the allowed collection of (message) types or procedure signatures, actual type parameters for types with formal type parameters can be given as the keyword **all** to indicate that all instances of the referenced parameterized type for any data type are allowed to be sent or received (dependent on the direction of the list) over that port.

In restriction e) the following sentence is added:

MessageType can also be a generic data type instance where the actual formal type parameters can be replaced with the keyword **all**.

Clause 6.5 Value parameterization of types

TTCN-3 allows the use of value parameters in type definitions. This is applicable to all user-defined types, including subtypes of basic types, and excluding behaviour types and signatures.

The formal parameters may be used inside the type definition. The actual parameters have to be either formal value parameters of an enclosing type or they have to satisfy the same restrictions as global constants, see ETSI ES 201 873‑1 [1] TTCN-3 Core Language clause 10.

When referring to a parameterized type, actual parameters have to be provided for each of the formal parameters of the type.

EXAMPLE 1: length restriction

 **type** **record length** ( 0 .. maxAmount ) **of float** MyFloats( **in integer** maxAmount );
**const** MyFloats(3) myConst := { 1.1, 2.1, 3.1 };

EXAMPLE 2: range subtyping

 **type** **integer** MyInt( **in integer** maxInt ) ( 0 .. maxInt );
**const** MyInt(127) myByte := 125;
**const** MyInt(127) myWord := 65335; // incorrect

EXAMPLE 3: passing parameter

 **type** **record** MySquareIndex( **in integer** maxInt ) {
 MyInt(maxInt) x,
 MyInt(maxInt) y
};

EXAMPLE 4: generic type wildcard in port definition

 **type record** R<T> { **record of charstring** headers, T payload }

 **type port** P **message** { **inout** R<all> } // allows any instance of R with any data

// type

## 5.4 Extension to ETSI ES 201 873-1, clause 8 (Modules)

Clause 8.2.3.1 General form of import

Import of definitions is additionally restricted by:

***Restrictions***

i) When importing a parameterized type the parameters are not resolved.

## 5.5 Extension to ETSI ES 201 873-1, annex A (BNF and static semantics)

The BNF is extended with the following clause and productions:

Clause A.1.6.1.14 Type parameter definitions

784001. FormalTypeParList ::= "<" [FormalTypePar](#TFormalTypePar) { "," [FormalTypePar](#TFormalTypePar) } ">"

784002. FormalTypePar ::= [ InParKeyword ] [ Type | SignatureKeyword | TypeDefKeyword ]
 [TypeParIdentifier](#TTypeParIdentifier) [ ":=" Type | Signature ]

784003. TypeParIdentifier ::= Identifier

784004. TypeActualParIdentifier ::= Identifier

784005. TypeParAssignment ::= [TypeActualParIdentifier](#TTypeActualParIdentifier) ":=" TypeActualPar

784006. ActualTypeParList ::= ( "<" [ActualTypePar](#TActualTypePar) { "," [ActualTypePar](#TActualTypePar) } ">" ) |
 ("<" [ActualTypeParAssignment](#TActualTypeParAssignment) { "," [ActualTypeParAssignment](#TActualTypeParAssignment) } ">")

784007. ActualTypePar ::= Type | Signature | "-" | AllKeyword

/\* STATIC SEMANTICS The use of AllKeyword is only allowed inside PortDefAttribs \*/

784008. ActualTypeParAssignment ::= [TypeActualParIdentifier](#TTypeActualParIdentifier) ":=" [ActualTypePar](#TActualTypePar)

784009. StructDefFormalParList ::= "(" [StructDefFormalPar](#TStructDefFormalPar) {"," [StructDefFormalPar](#TStructDefFormalPar)} ")"

784010. StructDefFormalPar ::= [FormalValuePar](#TFormalValuePar)

784011. TypeActualParList ::= ( "(" [TypeActualPar](#TTypeActualPar) {"," [TypeActualPar](#TTypeActualPar)} ")" ) |
 ( "(" TypeParAssignment { "," TypeParAssignment } ")" )

784012. TypeActualPar ::= [ConstantExpression](#TConstantExpression) | TypeActualParIdentifier

The following productions from ETSI ES 201 873-1 [1] TTCN-3 Core Language clause A.1.6 are modified as follows:

15. StructDefBody ::= ([Identifier](#TIdentifier) | [AddressKeyword](#TAddressKeyword))
 [[FormalTypeParList](#TFormalTypeParList)] [[StructDefFormalParList](#TStructDefFormalParList)]

 "{" [[StructFieldDef](#TStructFieldDef) {"," [StructFieldDef](#TStructFieldDef)}] "}"

27. UnionDefBody ::= ([Identifier](#TIdentifier) | [AddressKeyword](#TAddressKeyword))
 [[FormalTypeParList](#TFormalTypeParList)] [[StructDefFormalParList](#TStructDefFormalParList)]

 "{" [UnionFieldDef](#TUnionFieldDef) {"," [UnionFieldDef](#TUnionFieldDef)} "}"

33. StructOfDefBody ::= ([Type](#TType) | [NestedTypeDef](#TNestedTypeDef)) ([Identifier](#TIdentifier) | [AddressKeyword](#TAddressKeyword))
 [[FormalTypeParList](#TFormalTypeParList)] [[StructDefFormalParList](#TStructDefFormalParList)] [[SubTypeSpec](#TSubTypeSpec)]

35. EnumDef ::= [EnumKeyword](#TEnumKeyword) ([Identifier](#TIdentifier) | [AddressKeyword](#TAddressKeyword)) [[StructDefFormalParList](#TStructDefFormalParList)]

 "{" [EnumerationList](#TEnumerationList) "}"

41. SubTypeDef ::= [Type](#TType) ([Identifier](#TIdentifier) | [AddressKeyword](#TAddressKeyword))
 [[FormalTypeParList](#TFormalTypeParList)] [[StructDefFormalParList](#TStructDefFormalParList)] [[ArrayDef](#TArrayDef)] [[SubTypeSpec](#TSubTypeSpec)]

49. PortDefBody ::= [Identifier](#TIdentifier) [[FormalTypeParList](#TFormalTypeParList)] [[StructDefFormalParList](#TStructDefFormalParList)] [PortDefAttribs](#TPortDefAttribs)

73. ComponentDef ::= [ComponentKeyword](#TComponentKeyword) [Identifier](#TIdentifier)
 [[FormalTypeParList](#TFormalTypeParList)] [[StructDefFormalParList](#TStructDefFormalParList)]

 [[ExtendsKeyword](#TExtendsKeyword) [ComponentType](#TComponentType) {"," [ComponentType](#TComponentType)}]

 "{" [[ComponentDefList](#TComponentDefList)] "}"

76. ComponentType ::= [ExtendedIdentifier](#TExtendedIdentifier)
 [[ActualTypeParList](#TActualTypeParList)] [ TypeActualParList ]

79. PortInstance ::= [PortKeyword](#TPortKeyword) [ExtendedIdentifier](#TExtendedIdentifier) [[ActualTypeParList](#TActualTypeParList)]

 [ TypeActualParList ] [PortElement](#TPortElement) {"," [PortElement](#TPortElement)}

86. BaseTemplate ::= ([Type](#TType) | [Signature](#TSignature)) [Identifier](#TIdentifier)
 [[FormalTypeParList](#TFormalTypeParList)] ["("[TemplateOrValueFormalParList](#TTemplateOrValueFormalParList)")"]

100. StructFieldRef ::= [Identifier](#TIdentifier) | PredefinedType | ReferencedType

/\* STATIC SEMANTICS - PredefinedType and ReferencedType shall be used for anytype value notation only. PredefinedType shall not be AnyTypeKeyword.\*/

158. FunctionDef ::= [FunctionKeyword](#TFunctionKeyword) [Identifier](#TIdentifier) [[FormalTypeParList](#TFormalTypeParList)]

 "("[[FunctionFormalParList](#TFunctionFormalParList)] ")" [[RunsOnSpec](#TRunsOnSpec)] [[ReturnType](#TReturnType)] [StatementBlock](#TStatementBlock)

175. FunctionInstance ::= [FunctionRef](#TFunctionRef) [[ActualTypeParList](#TActualTypeParList)] "(" [[ActualParList](#TActualParList)] ")"

178. SignatureDef ::= [SignatureKeyword](#TSignatureKeyword) [Identifier](#TIdentifier) [[FormalTypeParList](#TFormalTypeParList)]

 "("[[SignatureFormalParList](#TSignatureFormalParList)] ")" [[ReturnType](#TReturnType) | [NoBlockKeyword](#TNoBlockKeyword)]

 [[ExceptionSpec](#TExceptionSpec)]

183. Signature ::= [ExtendedIdentifier](#TExtendedIdentifier) [[ActualTypeParList](#TActualTypeParList)]

185. TestcaseDef ::= [TestcaseKeyword](#TTestcaseKeyword) [Identifier](#TIdentifier) [[FormalTypeParList](#TFormalTypeParList)]

 "("[[TemplateOrValueFormalParList](#TTemplateOrValueFormalParList)] ")" [ConfigSpec](#TConfigSpec)

 [StatementBlock](#TStatementBlock)

190. TestcaseInstance ::= [ExecuteKeyword](#TExecuteKeyword) "(" [ExtendedIdentifier](#TExtendedIdentifier) [[ActualTypeParList](#TActualTypeParList)]
 "(" [[ActualParList](#TActualParList)] ")"

 [","([Expression](#TExpression) | [Minus](#TMinus)) ["," [SingleExpression](#TSingleExpression)]] ")"

192. AltstepDef ::= [AltstepKeyword](#TAltstepKeyword) [Identifier](#TIdentifier) [[FormalTypeParList](#TFormalTypeParList)]

 "("[[FunctionFormalParList](#TFunctionFormalParList)] ")" [[RunsOnSpec](#TRunsOnSpec)]

 "{" [AltstepLocalDefList](#TAltstepLocalDefList) [AltGuardList](#TAltGuardList) "}"

196. AltstepInstance ::= [ExtendedIdentifier](#TExtendedIdentifier) [[ActualTypeParList](#TActualTypeParList)] "(" [[ActualParList](#TActualParList)] ")"

408. ReferencedType ::= [[GlobalModuleId](#TGlobalModuleId) [Dot](#TDot)] [TypeReference](#TTypeReference) [[TypeActualParList](#TTypeActualParList)]
 [[ExtendedFieldReference](#TExtendedFieldReference)]

541. ExtendedFieldReference ::= {(([Dot](#TDot) ([Identifier](#TIdentifier) [[ActualTypeParList](#TActualTypeParList)][[TypeActualParList](#TTypeActualParList)]|

 [PredefinedType](#TPredefinedType) ))

 | [ArrayOrBitRef](#TArrayOrBitRef) | ("["[Minus](#TMinus) "]") ) }+

/\* STATIC SEMANTIC - The Identifier refers to a type definition if the type of the VarInstance or ReferencedValue in which the ExtendedFieldReference is used is anytype. ArrayOrBitRef shall be used when referencing elements of values or arrays. The square brackets with dash shall be used when referencing inner types of a record of or set of type.\*/

## 5.6. Extension to ETSI ES 203 790, clause 5.1 (Classes and Objects)

#### Clause 5.1.1.0 General

Change the following paragraph:

***Syntactical Structure***

Change to the following:

[**public** | **private**]
**type** [**external**] **class** [**@final** |**@abstract**]
*Identifier* [*FormalTypeParList*] [**extends** *Type*]
[*runsOnSpec*] [*systemSpec*] [*mtcSpec*]
"{" {*ClassMember*} "}"
[**finally** *StatementBlock*]

***Semantic Description***

Add the following:

Class definitions may be parameterized with formal type parameters.

Classes defined with formal type parameters can be referenced only by providing an actual type parameter list compatible with the formal type parameter list of the class.

***Examples***

Example:

**type** **external class @abstract Iterator<T> {
  function hasNext() return boolean;
  function next() return T;
}

type external class @abstract Container<T> {
  function iterator() return Iterator<T>;
  function size() return integer;
  function isEmpty() return boolean;**

 **function add(T element);**

 **function contains(T element) return boolean;
}

type external class Set<T> extends Container<T> {
}

type external class List<T> extends Container<T> {**

 **function getElement(integer pos) return T;
}

type external class MapEntr<Key, Value> {
  function getKey() return Key;
  function getValue() return Value;
}

type external class Map<Key,Value> {
  function get(Key key) return Value;
  function put(Key key, Value val);
  function keySet() return Set<Key>;
  function values() return List<Value>;
  function entrySet() return Set<MapEntry<Key, Value>>;
  function containsKey(Key key) return boolean;
  function contains(Value val) return boolean;
}**

**var Map<charstring, integer> v\_map := Map<charstring, integer>.create();**

**var Container<charstring> v\_container := …**

**select class (v\_container) {**

 **case (List<charstring>) { … }**

 **case (Set<charstring>) { … }**

**}**

5.7. Extension to ETSI ES 203 790, clause A.3 (Additional TTCN-3 syntax BNF productions)

The following BNF rules are changed:

033001. ClassDef ::= [ [ExtKeyword](#ExtKeyword) ] [ClassKeyword](#ClassKeyword) [ [FinalModifier](#FinalModifier) | [AbstractModifier](#AbstractModifier) ]

 [Identifier](#Identifier) [ FormalTypeParList ] [ [ExtendsKeyword](#ExtendsKeyword) [Type](#Type) ]

 [ [RunsOnSpec](#RunsOnSpec) ] [ [MtcSpec](#MtcSpec) ] [ [SystemSpec](#SystemSpec) ]

 "{" [ClassMemberList](#ClassMemberList) "}"

 [ [FinallyKeyword](#FinallyKeyword) [BasicStatementBlock](#BasicStatementBlock) ]