ETSI ES 203 790 V1.2.1 (2020-05)

Methods for Testing and Specification (MTS);

The Testing and Test Control Notation version 3;

TTCN-3 Language Extensions: Object-Oriented Features

**ETSI Standard**

Reference

RES/MTS-203790-OOFv1.2.1

Keywords

language, TTCN-3

***ETSI***

650 Route des Lucioles

F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - NAF 742 C

Association à but non lucratif enregistrée à la

Sous-Préfecture de Grasse (06) N° 7803/88

***Important notice***

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

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

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

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

***Copyright Notification***

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

© ETSI 2020.

All rights reserved.

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

#### 5.1.1.0 General

***Syntactical Structure***

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

***Semantic Description***

A class is a type where the values are called objects. A class can declare fields (variables, constants, templates, timers, classes) and methods as its members. Each member name inside the class shall be unique, there is no overloading. The private and protected fields and methods are only accessible by the methods of the class, while the public members of the class can be accessed also from behaviour not defined in the class. The private members of the class can be accessed directly only by members of the class itself. All members which are neither private nor public are protected and can also be accessed by members of subclasses.

All fields may be declared without initializer, even const and template fields.

A class can extend another class. The extended class is called the superclass, while the extending class is called the subclass. The resulting type of a class definition is the set of object instances of the class itself and all instances of its direct or indirect subclasses. A subclass is a subtype of its direct and indirect superclasses and its object instances are type compatible with them. If a class does not explicitly extend another class type, it implicitly extends the root class type **object**. Thus, all classes are directly or indirectly extensions of the **object** class.

A class can have optional runs on, mtc and system clauses. This restricts the type of component context that can create objects of that class and all methods of this class. If a class does not have one of these clauses, it inherits it from its superclass, if the superclass has one. If the superclass has or inherits a runs on, mtc or system clause, the subclass may declare each of these clauses with a more specific component type than the one inherited. The function members of classes shall not have runs on, system or mtc classes but inherit them from their surrounding class or its superclasses.

***Restrictions***

1. Void
2. Passing of object references to the create operation of a component type or a function started on another component is not allowed.
3. No subtyping definition is allowed for class types via the normal subtype definition.
4. No local/global constants or module parameters of class type or containing class type fields or elements are allowed.
5. Class type cannot be the contained value of an anytype value.
6. The functions of a class shall not have a runs on, mtc or system clause.
7. The runs on type of a class shall be runs on compatible with the runs on type of the behaviour creating a class.
8. The runs on type of a class shall be runs on compatible with the runs on type of the superclass.
9. The mtc and system type of a class shall be mtc and system compatible with the mtc and system types of the superclass, respectively.

#### 5.1.2.6 Object Templates

***Syntactical Structure***

"{" { ( *FieldName* | *FunctionInstance* ) ":=" *TemplateBody* [","] } "}"

***Semantic Description***

An object template is a matching mechanism to be used for objects similar to those used for record values. It can be used as a *TemplateBody* in template declarations for templates of class type and everywhere else where a *TemplateBody* is acceptable except in sending and receiving operations as objects can neither be sent nor received.

If an object template is used in a matching operation to match against an object reference, it matches if and only if for all assignments in the compound assignment notation the matching mechanism on the right hand side of the assignment matches the value yielded by the evaluation of applying the left hand side of the assignment as dotted notation to the object.

NOTE1: It is not necessary to add an assignment in the template for *all* public properties that exist in the class. For all public properties that are not assigned any matching mechanism in the template, a “don’t care” semantics is assumed. This is especially useful for using templates for objects of subclasses that might have additional properties.

Object templates declared for a class can also be used to match against objects of all subclasses.

The modifies operation is allowed also for Object templates with the same procedure as for record templates, but only *FieldName* assignments are modified while *FunctionInstance* assignments are not modified, though additional FunctionInstance assignments may be added.

It is allowed to have multiple *FunctionInstance* assignments of the same method in the same object template, possibly with different actual parameters.

NOTE2: Usage of *FunctionInstance* assignments can lead to contradictory assignments that would lead to the object template not matching any object of the class.

***Restrictions***

1. The *FieldName* in an assignment in the template shall be the name of a public value property of the template’s class. The type of the property shall be compatible with the corresponding *TemplateBody* on the right hand side.
2. The name of the function in the *FunctionInstance* in an assignment in the template shall be the name of a public method of the template’s class. The actual parameter lists given in the *FunctionInstance* shall be compatible with the formal parameter lists of that method and the return type of the function shall be a value type compatible with the corresponding *TemplateBody* on the right hand side.
3. The getter of a public property or the function used on the left hand side of an assignment in an object template shall be deterministic and shall fulfill the restrictions imposed on content of functions used in special places given in clause 16.1.4.
4. The names in *FieldName* assignments in an object template shall be unique, i.e no *FieldName* shall appear more than once on the left hand side.
5. Object templates shall not be used in sending or receiving operations.
6. Object template shall not be used as values and can not be converted to a value with the **valueof** operation.

***Examples***

EXAMPLE1:

**type class** Pair { **public var integer @property** a, **@property** b }  
  
**template** Pair t := { a := (1 .. 20) }  
  
**type class** Triple **extends** Pair { **public var integer @property** c }  
  
**match**(Triple.**create**(1,2,3), t) // returns true

EXAMPLE2:

**type** **class** @**abstract** Shape { **public** **function** @**abstract** area() **return** **float**; }

// smallShape would match for all objects whose class is derived from Shape

// and where the result of the method call to area() fulfills the constraint.  
**template** Shape smallShape := { area() := (0.0 .. 20.0) }

// contradictory template:

**template** Shape empty := {  
  area() := (0.0 .. **infinity**),  
  area() := {-**infinity** .. !0.0)  
}

**BNF Changes in section A.1.6.1.3 of ETSI ES 201 873-1**

100.FieldSpec ::= ( FieldReference | FunctionInstance ) AssignmentChar (TemplateBody | Minus)