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 Classes and Objects

### 5.1.0 General

This clause introduces the concepts of class types and their values, called objects as well as the operations allowed to be applied to these objects.

### 5.1.1 Classes

#### 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.

Classes can be used as field or element types of structured types.

***Restrictions***

1. Templates are not allowed for class types and structured types containing fields or elements of class type.
2. Passing of object references and structured types containing fields or elements of class type 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 structured types containing fields or elements of class type 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.
10. encoding or decoding, orand neither used as an actual parameter (or part therof) to a function started on another component

### 5.1.2 Objects

#### 5.1.2.0 General

Objects are the instances of classes. Each instance comprises an instance of the data of the fields of the class (including all superclasses) and allows invocation of its public methods by other behaviour and protected or private methods by behaviour defined by the object's class itself.

#### 5.1.2.1 Ownership

Each object is owned by the component on which it was created. The owning component of an object can be referenced via the self component reference. Methods of objects can only be invoked by behaviour that also runs on the owning component. An object is created on a component if its constructor was invoked by a behaviour running on that component.

#### 5.1.2.2 Object References

Objects are always passed by reference (even though their formal parameters can still be in, inout or out, dependent on the usage of that parameter). A variable of a class type contains only a reference to the object instance and the object is not copied when used as an actual parameter or assigned to a variable, but only the reference to the object. Therefore, multiple variables can contain a reference to the same object simultaneously.

***Restrictions***

1. Object References shall not be passed as actual parameter or part of an actual parameter to either the create operation of a component type or a function started on a component.

NOTE: Since objects cannot be shared by different component contexts and for each component at most one behaviour is running, no parallel conflicting access to any of the objects fields or methods is possible.

EXAMPLE:

**type class** MyClass() {

 **var** **integer** a := 0;

 **function** increment() {

 a := a + 1;

 }

 **function** getter() **return integer** {

 **return** a;

 }

}

**type record of** MyClass ROC;

**…**

**var** MyClass v\_a := MyClass.**create**();

**var** ROC my\_roc := {v\_a};

**var** ROC my\_roc2 := my\_roc; //create a copy

v\_a.increment();

my\_roc[0].increment();

my\_roc2[0].increment();

**var integer** v\_temp1 := my\_roc[0].getter(); // returns 3

**var integer** v\_temp2 := my\_roc2[0].getter(); // returns 3