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.

### 5.1.2 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. If a structured type contains a field of a class type, this type is not seen as a data type and its values cannot be used for sending and receiving or as an argument to any expression other than the equality/inequality operator.

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.

#### 5.1.2.3 Null reference

An object variable that is not initialized with an object instance contains the special value null. An object variable or parameter may be compared with the special value null with the equality and inequality operators or can be assigned the special value null explicitly.

#### 5.1.2.4 Select class-statement

***Syntactical Stucture***

**select** **class** "(" *Object* ")"
"{" { **case** "(" *ClassReference* ")" *StatementBlock* }+ [*ElseCase*] "}"

***Semantic Description***

The class of an object can be discriminated for via the 'select class' statement that is similar to a select union statement insofar that it allows only superclasses and known subclasses of the object reference's class in the context. If more than one case contains a superclass of the actual class of the given object instance, the first of these cases will be chosen by the select class statement.

In case that the *Object* is not an instance of any of the *ClassReference*s in the different cases, the statement block in the *ElseCase*, if present, will be executed.

EXAMPLE:

**type class** A {}

**type class** B **extends** A {}

…

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

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

 **case** (B) { … } // will be chosen

 **case** (A) { … } // will not be chosen

}

***Restrictions***

1. If a class from one case is a superclass of a class from another case, then the case of the subclass shall be precede the case of the superclass.

#### 5.1.2.5 Of-operator (Dynamic Class Discrimination)

***Syntactical Structure***

Object **of** ClassReference

***Semantic Description***

To check whether an object is an instance is of a certain class, the of operator may be used.

It yields a Boolean value which is true if and only if the most specific class of the object referenced on the left-hand side is either equal to or a subclass derived from the class type reference on the right-hand side.

#### 5.1.2.6 Casting

***Syntactical Structure***

ObjectReference "=>" ( ClassIdentifier | " (" ClassReference ")" )

***Semantic Description***

An object reference can be cast to another class of the object's known class's set of direct or indirect superclasses and direct or indirect subclasses. This operation yields an object reference to the same object but can be used as being of the type being cast to. If the referenced class to be cast to is an expression that is not a simple identifier, the expression shall be written in parenthesis.

***Restrictions***

1. If the class the object is being cast to is not in the set of superclasses or the concrete class of the object, the cast operation shall result in an error.

### 5.1.3 Extension to ETSI ES 201 873-1, clause 7.1.8 (Presence checking operators)

**Clause 7.1.8.0 General**

The presence checking operators (**ispresent**, **ischosen**, **isvalue** and **isbound)** shall also apply to object references and invocations of methods of objects.

If resolving a dot notation on an object would produce an error (the object referenced or returned by the invoked method having the null value) the following happens:

* No error is produced.
* Evaluation of all remaining unresolved fields in the *ExtendedFieldReference* is stopped. All remaining parts of the *ExtendedFieldReference* that are located right from the operation that would normally produce an error up to the end of the presence checking operator are not evaluated.
* The presence checking operator yields the value **false**.

The rule on special handling of dot notation, index notation, object method invocation and decoded field references is not applied recursively. Errors occurring during the invocation of an object’s method are not affected by this rule.

**Clause 7.1.8.1 The ispresent operator**

The **ispresent** operator can also be used to check if an object instance is present.

For an object instance argument the **ispresent** operator returns:

* The value **false** if the object is set to the null value.
* The value **true** otherwise.

***Examples***

EXAMPLE:

 // Given

**type class** A {}

**var** A v\_a1;

**var** A v\_a2 := A.**create**();

**var** **boolean** v\_checkResult := **ispresent**(v\_a1) // yields false
v\_checkResult := **ispresent**(v\_a2) // yields true

**Clause 7.1.8.3 The isvalue operator**

The **isvalue** operator always returns the value true for an object instance.

***Examples***

EXAMPLE:

 // Given

**type class** A {}

**var** A v\_a1;

**var** A v\_a2 := A.**create**();

**var** **boolean** v\_checkResult := **isvalue**(v\_a1) // yields true
v\_checkResult := **isvalue**(v\_a2) // yields true

**Clause 7.1.8.4 The isbound operator**

The **isbound** operator always returns the value true for an object instance.

***Examples***

EXAMPLE:

 // Given

**type class** A {}

**var** A v\_a1;

**var** A v\_a2 := A.**create**();

**var** **boolean** v\_checkResult := **isbound**(v\_a1) // yields true
v\_checkResult := **isbound**(v\_a2) // yields true