ETSI ES 201 873-1 V4.10.1 (2018-05)

Methods for Testing and Specification (MTS);

The Testing and Test Control Notation version 3;

Part 1: TTCN‑3 Core Language

**ETSI Standard**

Reference

RES/MTS-201873-1v4A1

Keywords

language, methodology, 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.

© ETSI 2018.

All rights reserved.

**DECT**TM, **PLUGTESTS**TM, **UMTS**TM and the ETSI logo are trademarks of ETSI registered for the benefit of its Members.
**3GPP**TM and **LTE**TM are trademarks 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 trademarks registered and owned by the GSM Association.

### 6.2.4 Enumerated type and values

TTCN‑3 supports **enumerated** types. Enumerated types are used to model types that take only a distinct named set of values. Such distinct values are called enumerated values. Each enumerated value shall have an identifier and referencing the values shall only use these identifiers. The identifiers of enumerated values shall be unique within the enumerated type (but do not have to be globally unique) and are consequently visible in the context of the given type only. This means that for any instantiation or value reference of an **enumerated** type, the given type shall be implicitly or explicitly identified.

NOTE 1: For example, if the enumerated type is an element or field of a user defined structured type, the enumerated type is implicitly referenced via the given element/field (i.e. by the identifier of the field or the position of the value in a value list notation) at value assignment, instantiation, etc. Another example is passing an enumerated value as actual parameter, in which case the type of the corresponding formal parameter establishes the type context needed to make the enumeration value visible. The third example is the comparison operators: if the type of one of the operands is uniquely identified, it is used as a type context for the other operand (see example 2 below). The fourth example is the match operation, where the type of the template parameter establishes the type context for the operation, if the type of the value parameter is not identified (see example 2 in clause 15.9).

The identifiers of enumerated values, within the same module, shall only be reused within other structured type definitions and shall not be used for identifiers of local or global visibility at the same or a lower level of the same branch of the scope hierarchy (see scope hierarchy in clause 5.2).

EXAMPLE 1: Declaration of enumerated types and values

 **type** **enumerated** MyFirstEnumType {

 Monday, Tuesday, Wednesday, Thursday, Friday

 };

 **type** **integer** Monday;

 // This definition does not clash with the previous one

 // as Monday in MyFirstEnumType is of local scope

 **type** **enumerated** MySecondEnumType {

 Saturday, Sunday, Monday

 };

 // This definition is legal as it reuses the Monday identifier within

 // a different enumerated type

 **type** **record** MyRecordType {

 **integer** Monday

 };

 // This definition is legal as it reuses the Monday identifier within

 // a distinct structured type as identifier of a given field of this type

 **type** **record** MyNewRecordType {

 MyFirstEnumType firstField,

 **integer** secondField

 };

 **var** MyNewRecordType v\_newRecordValue := { Monday, 0 }

 // MyFirstEnumType is implicitly referenced via the firstField element of MyNewRecordType

EXAMPLE 2: Using enumerated types (see also example 5 of clause 8.2.3.1)

 // Valid instantiations of MyFirstEnumType and MySecondEnumType would be

 **var** MyFirstEnumType v\_today := Tuesday;

 **var** MySecondEnumType v\_tomorrow := Monday;

 // The following statements however cause an error as the two variables are instances

 // of different enumeration types

 v\_today := v\_tomorrow;

 v\_today == v\_tomorrow;

 // The following operation is correct

 **if** (v\_today == Monday ) {...}

 // the type of variable v\_today identifies the type context of MyFirstEnumType for the

 // equality operator

 // But the following causes an error

 **if** ( Tuesday == Wednesday ) {...}

 // there is no TTCN-3 type(d) object to establish the type context for the equality operator

 // Please note that the values Tuesday and Wednesday are defined within the type

 // MyFirstEnumType only, but this is not sufficient to establish the type context

Each enumerated value may optionally have a user-assigned integer expression or non-empty list of integer literal values or ranges of integer literal values, which is defined after the name of the enumerated value in parenthesis. Each user assigned expression shall be statically bound, known in compilation time, and evaluate to an integer value. Each user-assigned integer value shall be distinct within a single **enumerated** type, all ranges of all the values lists shall be disjoint and shall not include any of the used single integer values. For each enumerated value without an assigned integer value, the system successively associates an integer number in the textual order of the enumerated values, starting at the left-hand side, beginning with zero, by step 1 and skipping any number occupied by any of the enumerated values with a manually assigned value or value list. These values are only used by the system to allow the use of relational operators. Enumerated names with an associated value list shall only be used as values together with a specific integer value, which shall be one from the associated list, in parenthesis after the name. They can be used as a template of the enumerated type by adding a list of integer template(s) and ranges in parenthesis after the name. For enumerated values with no value assigned or with a specific integer value assigned, the user shall not directly use associated integer values, but can access them and convert integer values into enumerated values by using the predefined functions **enum2int** and **int2enum** (see clauses 16.1.2, C.1.30 and C.1.4).

NOTE 2: The integer value also may be used by the system to encode/decode enumerated values. This, however is outside the scope of the present document (with the exception that TTCN‑3 allows the association of encoding attributes to TTCN‑3 items).

EXAMPLE 3: Enumeration example with associated integers

 **type** **enumerated** MyThirdEnumType {

 Blue(0),

 Yellow(1),

 Green(3),

 Other(2, 4..255)

 }

 **var** MyThirdEnumType v\_color := Other(5);

 **if** (v\_color == Other(4)) { // is false

 }

 **if** (v\_color > Other(4)) { // is true

 }

 **if** (**match**(v\_color, Other(?))) { // is true

 }

 **if** (**match**(v\_color, Other(6..10))) { // is false

 }

 **if** (**match**(v\_color, Other((6..10), 15, 20..25))) { // is false

 }

 v\_color := Blue(0) //causes an error as enumerated values with a specific integer value assigned
 //shall not use the associated integer value

**type** **enumerated** MyEnum {

 e\_num (1),

 e\_expr (2+2), // same as e\_expr (4)

 e\_bin\_conv (**bit2int**('0111'B)), // same as e\_bin\_conv(7)

 e\_oct\_conv (**oct2int**('34'O)), // same as e\_oct\_conv(52)

 e\_hex\_conv (**hex2int**('AC'H)) // same as e\_hex\_conv(172)

}

When a TTCN-3 module parameter, formal parameter, constant, variable, non-parameterized template or parameterized template with all formal parameters having default values of an imported enumerated type is defined, the name of that definition shall not be the same as any of the enumerated values of that type.

### 6.3.2 Compatibility of structured types

#### 6.3.2.0 General

This clause defines compatibility rules for structured types. In subsequent clauses, "value "b"" is called the value to be assigned, e.g. when passed as parameter, to an object of type "A".

#### 6.3.2.1 Compatibility of enumerated types

Enumerated types are only compatible with other **enumerated** types. An enumerated value "b" of an enumerated type "B" is compatible with enumerated type "A" if the identifier of the value "b" is also defined in "A" and the integer(s) associated with value "b" are also associated with the same identifier in "A".

EXAMPLE: Assigning enumerated values

// Given

**type enumerated** EWeekDays {

 Mon, Tue, Wed, Thu, Fri, Sat, Sun

};

**type enumerated** EWorkDays {

 Mon, Tue, Wed, Thu, Fri

};

**type enumerated** EDesWeekDays {

 Tue, Wed, Thu, Fri, Sat, Sun, Mon

};

**type** **enumerated** EComplexValues {

 e\_num (1),

 e\_expr (2+2),

 e\_bin\_conv (**bit2int**('0111'B)),

 e\_oct\_conv (**oct2int**('34'O)),

 e\_hex\_conv (**hex2int**('AC'H))

}

**type** **enumerated** ESimpleValues {

 e\_num (1),

 e\_expr (4),

 e\_bin\_conv (7),

 e\_oct\_conv (52),

 e\_hex\_conv (172)

}

**var** EWeekDays v\_myWeekDayMon := Mon

**var** EWeekDays v\_myWeekDaySun := Sun

**var** EWorkDays v\_myWorkDayMon := Mon

**var** EDesWeekDays v\_myDesWeekDayMon := Mon

**var** EComplexValues v\_myComplexValuedEnum := e\_bin\_conv;

**var** ESimpleValues v\_mySimpleValuedEnum := e\_bin\_conv;

v\_myWorkDayMon := v\_myWeekDayMon

 // works

v\_mySimpleValuedEnum := v\_myComplexValuedEnum;

 // works

v\_myWorkDayMon := v\_myWeekDaySun

 // causes an error as Sun is not a member of EWorkDays

v\_myDesWeekDayMon := v\_myWeekDayMon

 // causes an error as Mon in EDesWeekDays and EWeekDays have different associated

 // numbers; since this is true for all enumerated values in EWeekDays, these two

 // types are fully incompatible