ETSI ES 201 873-1 V4.7.1 (2015-06)

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-201873-1 T3ed471

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 <http://portal.etsi.org/tb/status/status.asp>

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 2015.

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.
**GSM**® and the GSM logo are Trade Marks registered and owned by the GSM Association.

### 6.2.7 Arrays

Arrays can be used in TTCN-3 as a shorthand notation to specify record of types. They may be specified also at the point of a variable declaration. Arrays may be declared as single or multi-dimensional. Array dimensions shall be specified using constant expressions, which shall evaluate to a positive **integer** values. Constants used in the constant expressions shall meet with the restrictions in clause 10.

EXAMPLE 1:

 **type integer** MyArrayType1[3]; // A type with 3 integer elements

 **type** **record** **length** (3) **of** **integer** MyRecordOfType1; // The corresponding record of

 **var** MyArrayType1 a1:= { 7, 8, 9 };

 **var** MyRecordOfType1 r1:= a1; // MyArrayType1 and MyRecordOfType1 are compatible

 **var** **integer** myArray1[3]:= r1; // Instantiates an integer array of 3 elements

 // with the index 0 to 2

 // being compatible to MyArrayType1 and MyRecordOfType1

 **var** **integer** myArray2[2][3]; // Instantiates a two-dimensional integer array of 2 × 3 elements // with indexes from (0,0) to (1,2)

Array elements are accessed by means of the index notation ([]), which shall specify a valid index within the array's range. Individual elements of multi-dimensional arrays can be accessed by repeated use of the index notation. An array or record of integer restricted to a single size can be used in the index notation as a short-hand for the repeated index notation. Accessing elements outside the array's range will cause a compile-time or test case error.

EXAMPLE 2:

 MyArray1[1] := 5;
 MyArray2[1][2] := 12;

 MyArray1[4] := 12; // ERROR: index shall be between 0 and 2
 MyArray2[3][2] := 15; // ERROR: first index shall be 0 or 1

Array dimensions may also be specified using ranges (with inclusive boundaries only). In such cases, the lower and upper values of the range define the lower and upper index values. Such an array is corresponding to a record of with a fixed length restriction computed as the difference between upper and lower index bound plus 1 and indexing starting from the lower bound of the array definition.

EXAMPLE 3:

 **type integer** MyArrayType2[2 .. 5]; // A type with 4 integer elements, indices starting with 2

 **type** **record** **length** (4) **of** **integer** MyRecordOfType2; // The corresponding record of

 **var** **integer** MyArray3[1 .. 5]; // Instantiates an integer array of 5 elements

 // with the index 1 to 5

 MyArray3[1] := 10; // Lowest index

 MyArray3[5] := 50; // Highest index

 **var** **integer** MyArray4[1 .. 5][2 .. 3 ]; // Instantiates a two-dimensional integer array of
 // 5 × 2 elements with indexes from (1,2) to (5,3)

NOTE: It is not possible to define an array type with a variable amount of elements. Neither is it possible to define an unlimited array with a lower bound on the array index.

The values of array elements shall be compatible with the corresponding variable or type declaration. Values may be assigned individually by a value list notation or index notation or more than one or all at once by a value list notation or index assignment notation. For using the valu list or assignment notation for arrays, the rules described in clause 6.2.3 are valid for arrays as well.

Index notation can be used on both the right-hand side and left-hand side of assignments. The index of the first element shall be zero or the lower bound if an index range has been given. The index shall not exceed the limitations given by either the length or the upper bound of the index. If the value of the element indicated by the index at the right‑hand of an assignment is undefined or if the index notation is applied to an uninitialized or omitted array value on the right hand side of an assignment, error shall be caused. Sending an array value with undefined elements shall cause an error. All elements in an array value that are not set explicitly are undefined. When referencing an element of an uninitialized array value or field or omitted field on the left hand side of an assignment, the rules for record of values specified in clause 6.2.3 apply.

For assigning values to multi-dimensional arrays, each dimension that is assigned shall resolve to a set of values enclosed in curly braces. When specifying values for multi-dimensional arrays, the leftmost dimension corresponds to the outermost structure of the value, and the rightmost dimension to the innermost structure. The use of array slices of multi-dimensional arrays, i.e. when the number of indexes of the array value is less than the number of dimensions in the corresponding array definition, is allowed. Indexes of array slices shall correspond to the dimensions of the array definition from left to right (i.e. the first index of the slice corresponds to the first dimension of the definition). Slice indexes shall conform to the related array definition dimensions.

EXAMPLE 4:

 MyArray1[0]:= 10;

 MyArray1[1]:= 20;

 MyArray1[3]:= 30;

 // or using an value list

 MyArray1:= {10, 20, **-**, 30};

 MyArray4:= {{1, 2}, {3, 4}, {5, 6}, {7, 8}, {9, 10}};

 // the array value is completely defined

 **var** **integer** MyArray5[2][3][4] :=

 {
 {
 {1, 2, 3, 4}, // assigns a value to MyArray5 slice [0][0]
 {5, 6, 7, 8}, // assigns a value to MyArray5 slice [0][1]
 {9, 10, 11, 12} // assigns a value to MyArray5 slice [0][2]
 }, // end assignments to MyArray5 slice [0]
 {
 {13, 14, 15, 16}, {17, 18, 19, 20}, {21, 22, 23, 24}

 } // assigns a value to MyArray5 slice [1]

 };

 MyArray4[2] := {20, 20};

 // yields {{1, 2}, {3, 4}, {20, 20}, {7, 8}, {9, 10}};
 MyArray5[1] := { {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}};
 // yields {{{1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12}},
 // {{0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}}};

 MyArray5[0][2] := {3, 3, 3, 3};

 // yields {{{1, 2, 3, 4}, {5, 6, 7, 8}, {3, 3, 3, 3}},

 // {{0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}}};

 **var integer** MyArrayInvalid[2][2];

 MyArrayInvalid := { 1, 2, 3, 4 }

 // causes an error as the dimension of the value notation

 // does not correspond to the dimensions of the definition

 MyArrayInvalid[2] := { 1, 2 }

 // causes an error as the index of the slice should be 0 or 1