Final draft ETSI ES 201 873-1 V4.8.1 (2016-05)

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-201873-1 T3ed481

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.

© European Telecommunications Standards Institute 2016.

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.

### 6.2.1 Record type and values

#### 6.2.1.0 General

TTCN‑3 supports ordered structured types known as **record**. The elements of a **record** type may be any of the basic types or user-defined data types (such as other records, sets or arrays). The values of a **record** shall be compatible with the types of the **record** fields. The element identifiers are local to the **record** and shall be unique within the **record** (but do not have to be globally unique).

EXAMPLE 1:

 **type** **record** MyRecordType

 {

 **integer** field1**,**

 MyOtherRecordType field2 **optional,**

 **charstring** field3

 }

 **type** **record** MyOtherRecordType

 {

 **bitstring** field1**,**

 **boolean** field2

 }

Records may be defined with no fields, i.e. as empty records.

EXAMPLE 2:

 **type** **record** MyEmptyRecord {}

A **record** value is assigned on an individual element basis. The order of field values in the value list notation shall be the same as the order of fields in the related type definition.

EXAMPLE 3:

 **var** **integer** v­\_myIntegerValue := 1;

 **const** MyOtherRecordType c\_myOtherRecordValue:=

 {

 field1 := '11001'B,

 field2 := **true**

 }

 **var** MyRecordType v\_myRecordValue :=

 {

 field1 := v\_myIntegerValue,

 field2 := c\_myOtherRecordValue,

 field3 := "A string"

 }

The same value specified with a value list.

EXAMPLE 4:

 v\_myRecordValue:= {v\_myIntegerValue, {'11001'B, **true**}, "A string"};

When the assignment notation is used for **record**‑s, fields wished to be changed shall be identified explicitly and a value, the not used symbol "-" or the **omit** keyword can be associated with them. The **omit** keyword shall only be used for optional fields. Its result is that the given field is not present in the given value.

NOTE: Please note the difference between omitted and uninitialized fields. Omitted optional fields are not present in the record or set value intentionally, i.e. the field is initialized and it does not prevent the whole record or set from being completely initialized.

EXAMPLE 5:

 **type** **record** MyRecordType

 {

 **bitstring** field1**,**

 **boolean** field2 **optional,**

 **charstring** field3

 }

 **var** MyRecordType v\_myVariable :=

 {

 field1 := '111'B,

 field2 := **false,**

field3 := -

 }

 v\_myVariable := { '10111'B, -, - };

 // after this, v\_myVariable contains:

 // { '10111'B, **false** /\* unchanged \*/, <undefined> /\* unchanged \*/ }

 v\_myVariable :=

 {

 field2 := **true**

}

 // after this, v\_myVariable contains:

 // { '10111'B /\* unchanged \*/, **true**, <undefined> /\* unchanged \*/ }

 v\_myVariable :=

 {

 field1 := -,

 field2 := **false,**

field3 := -

 }

 // after this, v\_myVariable contains:

 // { '10111'B /\* unchanged \*/, **false**, <undefined> /\* unchanged \*/}

When the assignment notation is used in a scope, where the **optional** attribute is implicitly or explicitly set to **"explicit** **omit"**, fields, not explicitly referred to in the notation, shall remain unchanged. It is also possible to leave fields explicitly unspecified using the not used symbol "-". In particular, when specifying partial values (i.e. setting the value of only a subset of the fields) using the assignment notation, at initialization, only the fields to be assigned values shall be specified. Fields not mentioned are implicitly left uninitialized. When re-assigning a previously initialized value, using the not used symbol or just skipping a field in an assignment notation, will cause that field to remain unchanged.
Optional fields of variables must always be assigned explicitly, otherwise they are uninitialized (i.e the optional attribute shall not have any effect on variables as described in section 27.7 restriction a ).

When the assignment notation is used in a scope, where the **optional** attribute is set to **"implicit** **omit"**, optional fields, not directly referred to in the notation, shall implicitly be set to omit, while mandatory fields shall remain unchanged (see also clause 27.7).

EXAMPLE 6:

 **type** **record** MyRecordType

 {

 **bitstring** field1**,**

 **boolean** field2 **optional,**

 **charstring** field3

 }

 **const** MyRecordType c\_myConst1 :=

 {

 field1 := '111'B,

field3 := “A string”

 } // { '10111'B, <undefined>, “A string”}

 **const** MyRecordType c\_myConst2 :=

 {

 field1 := '111'B,

field3 := “A string”

 } **with** { **optional** "implicit omit" }

 // { '10111'B, **omit** /\* because of the optional attribute \*/, “A string”}

When using the value list notation, all fields listed in the notation shall be specified either with a value, the not used symbol "‑" or the **omit** keyword. The **omit** keyword shall only be used for optional fields. Its result is that the given field is not present in the given value. The first component of the list (a value, a "-" or **omit**) is associated with the first field, the second list component is associated with the second field, etc. No empty assignment is allowed (i.e. two commas, the second immediately following the first or only with white space between them). Fields to be left unchanged shall be explicitly skipped in the list by using the not-used-symbol "-".

When using value list notation in a scope where the **optional** attribute is set to **"implicit omit"**, optional fields wished to be omitted by the implicit mechanism, but followed by fields to which a value or template is assigned explicitly, shall be skipped by using the not used symbol "-". When all remaining fields at the end of the type definition are optional and they are wished to be omitted by the implicit mechanism, either the not used symbol "-" can be used for some or all of them or they can simply be left out from the notation.

 EXAMPLE 7:

 **type** **record** R {
 **integer** f1,
 **integer** f2 **optional**,
 **integer** f3,
 **integer** f4 **optional**,
 **integer** f5 **optional**
 }

 **const** R c\_x := { 1, -, 2 } **with** { **optional** "implicit omit" }
 // after the assignment c\_x contains { 1, omit, 2, omit, omit }
 **const** R c\_x2 := { 1, 2, 3, - } **with** { **optional** "implicit omit" }
 // after the assignment c\_x2 contains { 1, 2, 3, omit, omit }

When using direct assignment notation in a scope where the **optional** attribute is set to **"implicit omit"**, the uninitialized optional fields in the referenced value, shall implicitly be set to omit after the assignment in the new value, while mandatory fields shall remain unchanged (see also clause 27.7)

EXAMPLE 8:

**const** R c\_x3 := { 1, -, 2 }
// after the assignment c\_x3 contains { 1, <undefined>, 2, <undefined>, <undefined>}
**const** R c\_x4 := c\_x3 **with** { **optional** "implicit omit" }
// after the assignment c\_x4 contains { 1, omit, 2, omit, omit }

## 27.7 Optional attributes

The **optional** attribute can be used to indicate that optional fields of constants, module parameters or templates of record and set types are implicitly set to **omit**.

***Syntactical Structure***

**optional**

***Semantic Description***

TTCN‑3 constants, module parameters, and templates can have an **optional** attribute. Also, TTCN-3 language elements that contain such definitions, i.e. module, group, function, altstep, test case, control, and component type definitions can have an **optional** attribute. When an **optional** attribute is associated to a function, altstep, test case, control or component type definitions, it shall have effect on all the constants, module parameters, and templates declared within these definitions and not on the enframing definition itself.

**Special optional strings:**

The following strings are the predefined (standardized) **optional** attributes.

a) "implicit omit" means that all optional fields, that have no assigned value definition in the statement on which the attribute operates, are set to omit. This applies recursively to the optional fields of the entity and to subfields of the mandatory fields.

b) "explicit omit" means that all optional fields, that have no assigned value definition in the statement on which the attribute operates, are left undefined. This applies recursively to the optional fields of the entity and to subfields of the mandatory fields.

***Restrictions***

In addition to the general static rules of TTCN‑3 given in clause 5, the following restrictions apply:

a) Data type, port type, procedure signature and variable definitions and import statements shall not have an **optional** attribute associated to them directly. When an **optional** attribute is associated to module, group, function, altstep, test case, control or component type containing such definitions, it shall not have any effect on the included data type, port type, procedure signature, variable or import statement.

***Examples***

**type** **record** MyRecord1 { **integer** a, **boolean** b **optional**}
**type record** MyRecord2{MyRecord1m}
// reference templates with explicitly set fields
**template** MyRecord1mw\_myTemplate1:= { a := ?, b := **omit** }
**template** MyRecord2mw\_myTemplate2:= { m := { a := ?, b := **omit** }}// reference templates
**template** MyRecord1mw\_myTemplate1a:= {a := ? } // b is undefined
**template** MyRecord1mw\_myTemplate1b:= {a := ? } **with** {**optional** "explicit omit**"**} // b is undefined
 **template** MyRecord2mw\_myTemplate2a:= {} // m and its subfields are undefined

**template** MyRecord2mw\_myTemplate2b:= { m := { a := ?}}; // m.b is undefined

// templates with attribute

 **template** MyRecord1mw\_myTemplate11 **:=** { a := ? } **with** {**optional "**implicit omit**"**}
 // same as mw\_myTemplate1, b is set to omit

**template** MyRecord2mw\_myTemplate21:= { m := { a := ?}} **with** {**optional "**implicit omit**"**}
// same as mw\_myTemplate2, by recursive application of the attribute

**template** MyRecord2mw\_myTemplate22:= { m := mw\_myTemplate1a } **with** {**optional "**implicit omit**"**}
// same as mw\_myTemplate2, by recursive application of the attribute

**template** MyRecord2mw\_myTemplate23:= {} **with** {**optional "**implicit omit**"**}
 // same as mw\_myTemplate2a, m remains undefined

**template** MyRecord2mw\_myTemplate24 **:=** { m := mw\_myTemplate1b } **with** {**optional "**implicit omit**"**}
// same as mw\_myTemplate2b, the attribute on the lower scope is not overwritten

**template** MyRecord2mw\_myTemplate25:= { m := MyTemplate1b } **with** {**optional override "**implicit omit**"**}
 // same as mw\_myTemplate2, the attribute on the lower scope is overwritten

 // implicitly omitted fields stay omitted after assignment
**template** MyRecord1 mw\_myTemplate3a := mw\_myTemplate1a **with** {**optional** "implicit omit**"**}
 // same as mw\_myTemplate1, b is set to omit
**template** MyRecord1 mw\_myTemplate3b := mw\_myTemplate3a;
 // same as mw\_myTemplate1, b is set to omit, by implicit omit attribute of mw\_myTemplate3a
**template** MyRecord1 mw\_myTemplate3c := mw\_myTemplate3a **with** {**optional** "explicit omit**"**}
 // same as mw\_myTemplate1, b is set to omit, by implicit omit attribute of mw\_myTemplate3a