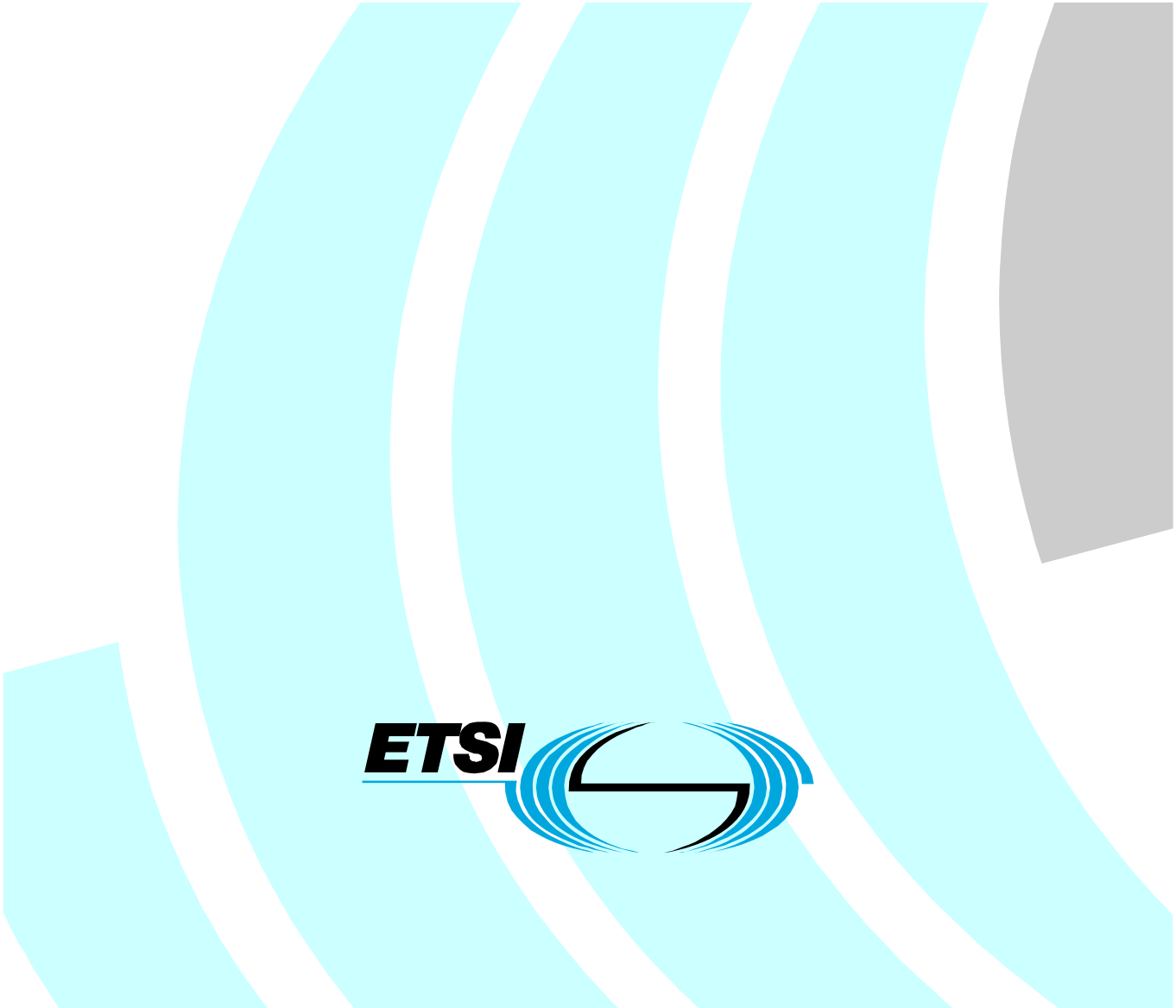
ETSI ES 201 873-7 V4.1.1 (2009-06)

ETSI Standard

Methods for Testing and Specification (MTS);
The Testing and Test Control Notation version 3;
Part 7: Using ASN.1 with TTCN-3



Reference
RES/MTS-00107-7 T3 ed411 asn1

Keywords
ASN.1, MTS, testing, TTCN

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

Individual copies of the present document can be downloaded from:
http://www.etsi.org

The present document may be made available in more than one electronic version or in print. In any case of existing or perceived difference in contents between such versions, the reference version is the Portable Document Format (PDF). In case of dispute, the reference shall be the printing on ETSI printers of the 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:
http://portal.etsi.org/chaircor/ETSI_support.asp

Copyright Notification

Contents

Intellectual Property Rights

IPRs essential or potentially essential to the present document may have been declared to ETSI. The information pertaining to these essential IPRs, if any, is publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (http://webapp.etsi.org/IPR/home.asp).

Pursuant to the ETSI IPR Policy, no investigation, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Foreword

This ETSI Standard (ES) has been produced by ETSI Technical Committee Methods for Testing and Specification (MTS).

The present document is part 7 of a multi-part deliverable. Full details of the entire series can be found in part 1 [1].

1 Scope

The present document defines a normative way of using ASN.1 as defined in the ITU-T Recommendation X.680 [2], ITU-T Recommendation X.681 [3], ITU-T Recommendation X.682 [4] and ITU-T Recommendation X.683 [5] with TTCN-3. The harmonization of other languages with TTCN-3 is not covered by the present document.

2 References

References are either specific (identified by date of publication and/or edition number or version number) or non-specific.

- For a specific reference, subsequent revisions do not apply.
- Non-specific reference may be made only to a complete document or a part thereof and only in the following cases:
  - if it is accepted that it will be possible to use all future changes of the referenced document for the purposes of the referring document;
  - for informative references.

Referenced documents which are not found to be publicly available in the expected location might be found at http://docbox.etsi.org/Reference.

NOTE: While any hyperlinks included in this clause were valid at the time of publication ETSI cannot guarantee their long term validity.

2.1 Normative references

The following referenced documents are indispensable for the application of the present document. For dated references, only the edition cited applies. For non-specific references, the latest edition of the referenced document (including any amendments) applies.

[1] ETSI ES 201 873-1 (V4.1.1): "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language".

[2] ITU-T Recommendation X.680 (2002): "Information technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation".

[3] ITU-T Recommendation X.681 (2002): "Information technology - Abstract Syntax Notation One (ASN.1): Information object specification".

[4] ITU-T Recommendation X.682 (2002): "Information technology - Abstract Syntax Notation One (ASN.1): Constraint specification".

[5] ITU-T Recommendation X.683 (2002): "Information technology - Abstract Syntax Notation One (ASN.1): Parameterization of ASN.1 specifications".

[6] ITU-T Recommendation X.690 (2002): "Information technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)".

[7] ITU-T Recommendation X.691 (2002): "Information technology - ASN.1 encoding rules: Specification of Packed Encoding Rules (PER)".

[8] ITU-T Recommendation X.693 (2001): "Information technology - ASN.1 encoding rules: XML Encoding Rules (XER)".

[9] ITU-T Recommendation T.100 (1988): "International information exchange for interactive videotex".

[10] ITU-T Recommendation T.101 (1994): "International interworking for Videotex services".

[11] ITU-T Recommendation X.660 (2004): "Information technology - Open systems interconnection - Procedures for the operation of OSI Registration Authorities: General procedures and top arcs of the ASN.1 Object Identifier tree".

[12] ETSI ES 201 873-6 (V4.1.1): "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 6: TTCN-3 Control Interface (TCI)".

NOTE: References to ITU-T Recommendations include the Recommendation and all Amendments and Corrigenda published to the Recommendation.

2.2 Informative references

The following referenced documents are not essential to the use of the present document but they assist the user with regard to a particular subject area**.** For non-specific references, the latest version of the referenced document (including any amendments) applies.

[i.1] ITU-T Recommendation X.208: "Specification of Abstract Syntax Notation One (ASN.1)".

[i.2] ITU-T Recommendation T.61 (1988): "Character repertoire and coded character sets for the international teletex service".

3 Definitions and abbreviations

3.1 Definitions

For the purposes of the present document, the terms and definitions given in ES 201 873-1 [1], ITU-T Recommendation X.660 [11] and the following apply:

associated TTCN-3 type: TTCN-3 type definition resulted from the transformation of an ASN.1 type definition by applying the transformation rules in clause 9.1

NOTE: Associated TTCN-3 types and values may not exist in a visible way; this term is used to identify the part of the abstract information carried by the related ASN.1 type or value, which have significance from the point of view of TTCN-3 (also called the TTCN-3 view).

metatype "OPEN TYPE": term used to explain the ASN.1 to TTCN-3 conversion process

NOTE: It does not exist in the input ASN.1 module or the output TTCN-3 module.

root type: Definition in ES 201 873-1 [1] applies with the following addition: in case of types based on imported ASN.1 types, the root type is determined from the associated TTCN-3 type (see clause 8).

3.2 Abbreviations

For the purposes of the present document, the abbreviations given in ES 201 873-1 [1] and the following apply:

ASN.1	Abstract Syntax Notation One
OID	Object IDentifier
TTCN-3	Tree and Tabular Combined Notation version 3

4 Introduction

When using ASN.1 with TTCN-3 all features of TTCN-3 and statements given in clause 4 of ES 201 873-1 [1] do apply. In addition, when supporting this part of the standard, TTCN-3 becomes fully harmonized with ASN.1 which may be used with TTCN-3 modules as an alternative data type and value syntax. This part of the standard defines the capabilities required in addition of those specified in ES 201 873-1 [1] when ASN.1 is supported. The approach used to combine ASN.1 and TTCN-3 could be applied to support the use of other type and value systems with TTCN-3. However, the details of this are not defined in the present document.

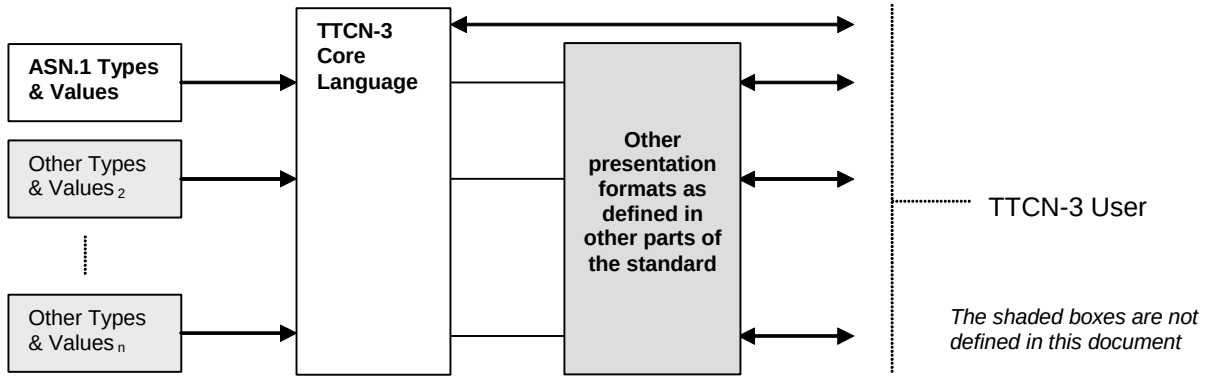


Figure 1: User's view of the core language and the various presentation formats

4.1 Conformance

For an implementation claiming to support the use of ASN.1 with TTCN-3, all features specified in the present document shall be implemented consistently with the requirements given in the present document and in ES 201 873-1 [1].

5 General

TTCN-3 provides a clean interface for using ASN.1 definitions (as specified in ITU-T Recommendation X.680 [2], ITU-T Recommendation X.681 [3], ITU-T Recommendation X.682 [4] and ITU-T Recommendation X.683 [5]) in TTCN-3 modules.

6 Amendments to the core language

Void.

7 Additional TTCN-3 types

7.1 General

TheTTCN-3 types summarized in table 1 shall be supported in addition to those specified in clause 6 of ES 201 873-1 [1].

Table 1: Overview of TTCN-3 types

Class of type	Keyword	Sub-type
Simple basic types	`objid`	list, range

7.2 The object identifier type

The object identifier type shall be supported as follows:

- **objid:** a type whose distinguished values are the set of all syntactically correct object identifier values. The value notations for the objid type shall conform to clause 31 of ITU-T Recommendation X.680 [2] with the exception that hyphens are replaced with underscores.

NOTE 1: This definition also allows object identifier values outside the collection of values defined in ITU-T Recommendation X.660 [11] (e.g. with a node beneath the root not defined in ITU-T Recommendation X.660 [11]).

The name form of object identifier components shall be used only for components defined in ITU-T Recommendation X.660 [11]. These predefined object identifier components are given in annex C for information. In case of any conflict between ITU-T Recommendation X.660 [11] and annex C of the present document, the former shall take precedence.

In cases when the identifier of a value referenced within an object identifier value notation is identical to any of the predefined component names, i.e. independently of the position of the predefined component or the referenced value inside the notation (considering name conversion rules in clause 8.2), the name of the referenced value shall be prefixed with the name of the module in which it is defined (see definition of ASN.1 modules in clause 12 of ITU-T Recommendation X.680 [2] and TTCN-3 modules in clause 8.1 of the core language standard ES 201 873-1 [1]). The prefix and the identifier shall be separated by a dot (.). Predefined object identifier component names may also be prefixed with the name "X660".

NOTE 2: To increase readability it is recommended to use the "X660" prefix also in object identifier values referring to a value identifier that is clashing with any of the predefined component names.

NOTE 3: Rules to resolve name clashes caused by imports are defined in clause 8.2.3.1 of the core language standard ES 201 873-1 [1].

EXAMPLE:

```
objid{itu_t(0) identified_organization(4) etsi(0)}
// or alternatively
objid {itu_t identified_organization etsi(0)}
// or alternatively
objid { 0 4 0}

// or alternatively
const integer etsi := 0;
const objid itu_idOrg := objid{ itu_t identified_organization }
objid{ itu_idOrg etsi } // note, that both names are referencing value definitions

const integer x := 162;
objid{ itu_t recommendation x A.x }         // it is mandatory to use the module name ('A')
                                            // to prefix the ambiguous identifier
                                            // or alternatively
objid{ itu_t recommendation X660.x A.x }    // the module name shall be present even if
                                            // the "X660" prefix is used
```

7.2.1 Sub-typing of the **objid** type

7.2.1.1 Subtrees of the **objid** type

The object identifier type is a collection of principally infinite set of unique identifier values, each containing a sequence of components; each given sequence of arbitrary length compose an object identifier node as shown on figure 2 (see also annex C). Thus, each node of the object identifier tree - except being a unique identifier itself - is the root of a subtree, containing a potentially infinite number of unique identifiers. The first n components of all the identifiers in the subtree are identical to the components of the node, being the root of the subtree, where n is the number of components of that node. Hence, each object identifier node distinguishes also a unique subset (subtype) of the **objid** type. Each member of this subtype (the subtree) is longer than the node identifying the subtree.

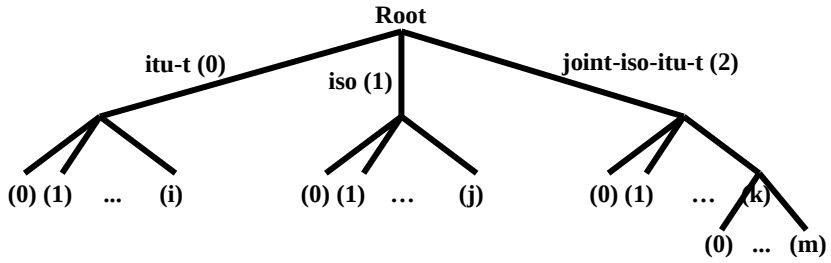


Figure 2: The object identifier tree

Note that object identifiers may also be relative identifiers, i.e. when the given objid value contains only the additional components related to a defined base node. However, the above remains true for relative object identifiers as well, as they also do denote a unique node in the object identifier tree (the node defined by the base node + the relative identifier). If a node is identified by a relative object identifier, all nodes in its subtree will have relative identifiers too.

7.2.1.2 List subtypes

In addition to the types listed in clause 6.1.2.1 of ES 201 873-1 [1], value list subtyping of the **objid** type shall be supported. For value lists of the **objid** type the rules in this clause apply. The **objid** nodes in the list shall be of **objid** type and shall be a true subset of the values defined by the **objid** type or the base type being restricted. Clause 7.2.1.1 shall govern in determining if the nodes are a true subset. The subtype defined by this list restricts the allowed values to the concrete nodes on the list and the subtrees identified by them.

EXAMPLE:

```
//identifies the nodes {0 4 0 0}, {0 4 0 1} and all other nodes beneath them
type objid MyObjids (objid{0 4 0 0}, objid{0 4 0 1});

//Further restricting the base type MyObjids
type MyObjids MyNarrowerObjids (objid{0 4 0 0 1 0}, objid{0 4 1 1}, objid{0 4 1 3});

//invalid definition as the node {0 4 2} is not member of the type MyObjids
type MyObjids MyNarrowestObjids (objid{0 4 2 1});
```

7.2.1.3 Range subtypes

In addition to the types listed in clause 6.1.2.2 of ES 201 873-1 [1], value list subtyping of the **objid** type shall be supported. For range subtyping of the **objid** type the rules in this clause apply. The **objid** nodes determining the lower and the upper bounds of the subtype shall be of **objid** type of equal length and shall be a true subset of the values defined by the **objid** type or the base type being restricted. Clause 7.2.1.1 shall govern in determining if the nodes are a true subset. The subtype defined by the range restricts the allowed values to the nodes between the lower and the upper bounds inclusive and the subtrees identified by them.

EXAMPLE:

```
//identifies the nodes {0 4 0 0}, {0 4 0 1} … {0 4 0 5} and all other nodes beneath them
type objid MyObjidRange (objid{0 4 0 0} .. objid{0 4 0 5});
```

7.2.1.4 Mixing list and range subtypings

It is allowed to mix the list and the range subtyping mechanisms for **objid** types. The nodes identified by the different subtyping mechanisms shall not overlap.

7.2.2 Object identifier values

When defining **objid** values, rules in clauses 5.4.1.1, 8.2.1, 10 and 11.1 of ES 201 873-1 [1] and in this clause shall apply. In case of inconsistency the present document takes precedence.

Each object identifier node is an object identifier value. In this case the value identifies the concrete node (i.e. with a definite number of components) and does not denote the **objid** subtree beneath it (see clause 7.2.1.1).

7.2.3 Using **objid** values to identify modules

7.2.3.1 Identifying module definitions

When ASN.1 is supported, module names (of the form of a TTCN-3 identifier) may optionally be followed by an object identifier, which shall be a valid value as defined in ITU-T Recommendation X.660 [11].

NOTE: Module names in a test suite may differ in the object identifier part only. However, in this case, due precaution has to be exercised at import to avoid name clash, as prefixing of TTCN-3 identifiers (see clause 8.2.3.1 of ES 201 873-1 [1]) is unable to resolve such kind of clashes.

7.2.3.2 Identifying modules in import statements

When ASN.1 is supported, in addition to the module names, their object identifiers may also be provided in TTCN-3 import statements. If an object identifier is used as part of the module identifier, this object identifier shall be used by TTCN-3 test systems to identify the correct module.

7.2.4 Object identifier templates

When defining templates of **objid** types, rules in clause 15 and annex B of ES 201 873-1 [1] and in this clause shall apply. In case of inconsistency the present document takes precedence.

7.2.4.1 In-line Templates

The type of **objid** values can be identified from the value notation alone, hence in addition to the types listed in note 2 of clause 15.4 in ES 201 873-1 [1], the type specification may also be omitted in case of objid values.

7.2.4.2 Template matching mechanisms

Applicability of matching mechanisms to templates of **objid** types is defined in table 2.

Table 2: TTCN-3 Matching Mechanisms

Used with values of	Value		Instead of values								Inside values			Attributes	
	SpecificValue	OmitValue	ComplementedList	ValueList	AnyValue (?)	AnyValueOrNone (*)	Range	Superset	Subtype	Pattern	AnyElement (?)	AnyElementsOrNone (*)	Permutation	LengthRestriction	IfPresent
objid	Yes	Yes	Yes	Yes	Yes	Yes (see note 1)	Yes				Yes	Yes		Yes	Yes (see note 2)
NOTE 1: When used, shall be applied to optional fields of record and set types only (without restriction on the type of that field).															
NOTE 2: When used, shall be applied to record and set fields only (without restriction on the type of that field).															

The matching mechanisms SpecificValue, OmitValue, AnyValue, AnyValueOrNone and IfPresent are applicable to **objid** fields as well according to the rules defined in ES 201 873-1 [1].

The value list and complemented value list matching mechanisms can also be used for **objid** templates and template fields. Rules in clauses B.1.2.1 and B.1.2.2 of ES 201 873-1 [1] also shall apply to **objid** templates.

NOTE: This also means that only the concrete node values on the list is to be considered but not the subtrees identified by them. I.e. in case of a complemented list, a node within a given subtree will match even if the node being the root of the subtree is on the list.

The value range matching mechanism, in addition to types listed in clause B.1.2.5 of ES 201 873-1 [1], can also be used for **objid** templates. When applied to **objid**s, the values matching the range shall be determined according to clause 7.2.1.3 of the present document, with the exception, that subtrees are not considered.

The inside value matching mechanism *AnyElement*, in addition to types listed in clause B.1.3.1 of ES 201 873-1 [1], can also be used within objid templates. When applied to objids, it replaces exactly one component.

The inside value matching mechanism *AnyElementsOrNone*, in addition to types listed in clause B.1.3.2 of ES 201 873-1 [1], can also be used within objid templates. When applied to objids, it matches the longest sequence of components possible, according to the pattern as specified by the components surrounding the "*".

The length restriction matching attribute, in addition to types listed in clause B.1.4.1 of ES 201 873-1 [1], can also be used with **objid** templates. When applied to objids, it identifies the number of components within an **objid** value matching the **objid** template.

7.2.5 Using **objid** with operators

7.2.5.1 List operators

When ASN.1 is supported, the concatenation operator specified in clause 7.1.2 of ES 201 873-1 [1] shall be permitted for **objid** values as well. The operation is a simple concatenation of the numerical values of the components from left to right. If necessary (e.g. for logging purposes), the names of the components in the resulted **objid** value shall be determined from the resulted **objid** value (i.e. names of the components will change when the component is changing its position related to the input objid value). The result type is **objid**.

EXAMPLE:

```
 objid{itu_t identified_organization etsi(0)} & objid{inDomain(1) in_Network(1)}
     gives {0 4 0 1 1} that can also be presented as objid{itu_t identified_organization etsi(0)
inDomain(1) in_Network(1)}

 objid{itu_t identified_organization etsi(0)} & objid{iso(1) registration_authority(1)}
     gives {0 4 0 1 1} that can also be presented as objid{itu_t identified_organization etsi(0)
inDomain(1) in_Network(1)}
```

7.2.5.2 Relational operators

It is allowed to use **objid** values as operands of relational operators equality (==), less than (<), greater than (>), non-equality to (!=), greater than or equal to (>=) and less than or equal to (<=).

Two **objid** values are equal, if they have equal number of components and the primary integer values at all positions are the same.

The less than (<), greater than (>), greater than or equal to (>=) and less than or equal to (<=) operations shall use the numerical values of objid value components for the decision, and the decision process shall comply with the following rules:

- the comparison shall start by comparing the first primary integer values of the two objid values and shall be continued in a recursive way until the smaller objid value is found or the two objid values are found to be equal;
- the objid value in which a smaller primary integer value is found first , is less than the other objid value;

- if all compared pairs of primary integer values of the two objid values are equal and one of the objid values has further primary integer values while the other does not, the shorter objid value is less than the longer objid value.

EXAMPLE:

```
// Given
const objid c_etsiMobNet := objid{itu_t identified_organization etsi(0)
                                  mobile_domain(0) umts_Network(1)}
const objid c_etsiINNet  := objid{itu_t identified_organization etsi(0)
                                  inDomain(1) in_Network(1)}
const objid c_etsiIN     := objid{itu_t identified_organization etsi(0)
                                  inDomain(1)}
var objid   v_etsiInIso  := objid{ iso identified_organization dod(6)
                                   internet(1) private(4) enterprise(1) etsi(13019)}
// then
c_etsiMobNet == c_etsiINNet // returns false
c_etsiMobNet <  c_etsiINNet // returns true as the mobile_domain(0) component is numerically
                            // smaller than the inDomain(1) component
c_etsiINNet ==  c_etsiIN    // returns false as c_etsiINNet has more components
c_etsiINNet  >   c_etsiIN   // returns true as c_etsiINNet has more components
v_etsiInIso <= c_etsiMobNet // returns false as the component itu_t(0) is numerically smaller
                            // than the component iso(1))
```

7.2.6 Using **objid** with predefined functions

7.2.6.1 Number of components of an **objid** value or template

In excess the input parameter types given in clause C.28 of ES 201 873-1 [1], the **lengthof** predefined function shall allow values and templates of **objid** types as input parameter. The actual value to be returned is the sequential number of the last component.

When the function **lengthof** is applied to templates of **objid** types, inpar shall only contain the matching mechanisms: specific value, value list, complemented list, "?" (*AnyValue* instead of value), "*" (*AnyValueOrNone* instead of value), "?" (*AnyElement* inside value) and "*" (*AnyElementsOrNone* inside value) and the length restriction matching attribute. The parameter inpar shall only match values, for which the **lengthof** function would give the same result.

Additional error cases are:

- inpar can match **objid** values with different number of components.

EXAMPLE:

```
// Given
 var objid v_etsiMobNet := objid{itu_t identified_organization etsi(0)
                                 mobile_domain(0) umts_Network (1)}
 // then
 numElements := lengthof(v_etsiMobNet); // returns 5
```

7.2.6.2 The Substring function

When ASN.1 is supported, the **substr** predefined function shall support **objid** types, i.e. it shall allow objid as type of the input parameter and return an object identifier value containing a fragment (sequence of components) of the input parameter inpar. Rules specified in clause C.33 of ES 201 873-1 [1] shall apply with the following exceptions: index zero identifies the first component of the input object identifier value or template. The third input parameter (count) defines the number of components in the returned **objid** value.

EXAMPLE:

```
 var objid v_etsiMobNet := objid{itu_t identified_organization etsi(0)
                                 mobile_domain(0) umts_Network (1)}

 substr (v_etsiMobNet, 0, 2)  // returns {itu_t identified_organization}

 substr (v_etsiMobNet, 2, 3)  // returns {etsi(0) mobile_domain(0) umts_Network (1)}
```

```
substr (v_etsiMobNet, 0, 0)  // causes error as the number of components to be returned
                             // shall be more than 0

substr (v_etsiMobNet, 0, 6)  // causes error as the input objid value contains less
                             // than 6 components
```

7.2.6.3 The isvalue function

When ASN.1 is supported, the **isvalue** predefined function shall be supported for **objid** templates too. Rules specified in clause C.37 of ES 201 873-1 [1] shall apply.

7.2.7 Supporting objid in TCI

7.2.7.1 objid and TciTypeClass

The `TciTypeClass` defined in clause 7.2.2.1 (Abstract TTCN-3 data types) of ES 201 873-6 [12] is extended with the constant `OBJID`.

7.2.7.2 objid and TCI type hierarchy

The type hierarchy given in figure 4 in clause 7.2.2.2 (Abstract TTCN 3 values) of ES 201 873-6 [12] is extended by the `ObjidValue` abstract type.

8 ASN.1 and TTCN-3 type equivalents

8.1 General

The ASN.1 types listed in table 3 are considered to be equivalent to their TTCN-3 counterparts.

Table 3: List of ASN.1 and TTCN-3 equivalents

ASN.1 type	Maps to TTCN-3 equivalent
BOOLEAN	**boolean**
INTEGER	**integer**
REAL (note)	**float**
OBJECT IDENTIFIER	**objid**
BIT STRING	**bitstring**
OCTET STRING	**octetstring**
SEQUENCE	**record**
SEQUENCE OF	**record of**
SET	**set**
SET OF	**set of**
ENUMERATED	**enumerated**
CHOICE	**union**
VisibleString	**charstring**
IA5String	**charstring**
UniversalString	**universal charstring**
NOTE: The ASN.1 type REAL is equivalent to the TTCN-3 type **float** until the base is unrestricted or restricted to base 10 explicitly or implicitly. The ASN.1 notation allows explicit restriction by e.g. inner subtyping but from ASN.1-TTCN-3 type mapping point of view an explicit restriction is an ASN.1 value notation. Implicit restriction may be defined by the textual description of the given protocol, i.e. outside of the ASN.1 module(s). However, in both cases the TTCN-3 value notation can be used irrespective if the base in ASN.1 (see also note 3 in clause 8.1).	

All TTCN-3 operators, functions, matching mechanisms, value notations etc. that can be used with a TTCN-3 type given in table 3 may also be used with the corresponding ASN.1 type.

8.1bis Importing from ASN.1 modules

When importing from ASN.1 modules, it is mandatory to use one of the following language identifier strings:

- "ASN.1:2002" for ASN.1 version 2002;
- "ASN.1:1997" for ASN.1 version 1997;
- "ASN.1:1994" for ASN.1 version 1994;
- "ASN.1:1988" for ASN.1 version 1988 (CCITT Blue Book).

NOTE 1: Language identifiers "ASN.1:1997", "ASN.1:1994" and "ASN.1:1988" refer to versions of ASN.1 based on superseded ITU-T Recommendations (including the base document and all published Amendments and Technical Corrigenda published so far complementing the base document). The only purpose to include them into the present document is to allocate unique identifiers if protocol modules based on these ASN.1 versions are used with TTCN-3. When ASN.1 version 1997 is supported, the support of Amendment 3 to ITU-T Recommendation X.680 [2] is not considered.

NOTE 2: When "ASN.1:1988" is supported, the ASN.1 definitions will be imported from such modules according to the syntactical and semantical rules of ITU-T Recommendation X.208 [i.1] (Blue Book).

NOTE 3: References to the 1994, 1997 and the Blue Book (1988) versions of ASN.1 can be found in annex E.

When importing ASN.1 types, values and value sets, first they have to be transformed to TTCN-3 types and values respectively, according to the rules given in clauses 8 and 9.1 of the present document and import the resulted definitions afterwards based on the rules defined below. All ASN.1 definitions are **public** by default (see clause 8.2.3 of ES 201 873-1 [1]).

NOTE 4: ASN.1 value sets are semantically equivalent to subtypes. Hence, in the present document they are not handled separately but all rules specified for ASN.1 types also apply to ASN.1 value sets.

8.2 Identifiers

In converting ASN.1 identifiers to TTCN-3 identifiers, hyphen "-" characters shall be changed to underscore "_" characters. When TTCN-3 keywords are used as identifiers in ASN.1 modules, these identifiers shall be appended with a single underscore "_" character at import.

EXAMPLE:

```
MyASN1module DEFINITIONS ::=
BEGIN
    Misleading-ASN1-Name ::=    INTEGER     -- ASN.1 type identifier using '-'
    TypeWithTTCN-3Keyword ::= SEQUENCE {
        value       INTEGER,
        message     OCTET STRING
    }

END

module MyTTCNModule
{
    import from MyASN1module language "ASN.1:2002" all;

    // TTCN-3 reference to ASN.1 type using underscores
    const Misleading_ASN1_Name cg_Example1 := 1;

    // TTCN-3 reference to identifiers which are TTCN-3 keywords
    const TypeWithTTCN_3Keyword cg_Example2 := {
        value_ := 5,
        message_ := 'FF'O
    }
}
```

9 ASN.1 data types and values

9.1 Transformation rules

ASN.1 types and values may be used in TTCN-3 modules. ASN.1 definitions are made using a separate ASN.1 module. ASN.1 types and values are referenced by their type references and value references as produced according to clauses 9.3 and 9.4 of ITU-T Recommendation X.680 [2] within the ASN.1 module(s). Basic ASN.1 value notation and XML ASN.1 value notation shall be transformed equally, i.e. a basic and an XML value notation referring to the same value of the type shall produce the same associated TTCN-3 value.

EXAMPLE 1:

```
MyASN1module DEFINITIONS ::=
BEGIN
     Z::=    INTEGER             -- Simple type definition

     BMessage::= SEQUENCE            -- ASN.1 type definition
     {
         name    IA5String,
         title   VisibleString,
         date    IA5String
     }

     johnValues Bmessage ::=             -- ASN.1 value definition
     {
         name    "John Doe",
         title   "Mr",
         date    "April 12th"
     }

     johnValuesXML ::=                   -- XML ASN.1 value definition
         <Bmessage>
             <name>John Doe</name>
             <title>Mr</title>
             <date>April 12th<date>
         <Bmessage>

     DefinedValuesForField1 Z ::= {0 | 1} -- ASN.1 subtype definition
END
```

The ASN.1 module shall conform to the syntax and semantics of the ITU-T Recommendation X.680 [2], ITU-T Recommendation X.681 [3], ITU-T Recommendation X.682 [4] and ITU-T Recommendation X.683 [5]. Once declared and imported, ASN.1 types and values may be used within TTCN-3 modules in a similar way than TTCN-3 types and values, imported from other TTCN-3 modules. Each imported ASN.1 definition produces an associated type or value. All TTCN-3 definitions or assignments based on imported ASN.1 definitions shall be done according the rules imposed by the related associated type or value. Also, the matching mechanism shall use the associated type when matching at a receiving or a **`match`** operation.

Associated types and values are derived from ASN.1 definitions by applying the transformation rules below. Transformations shall be started on a valid ASN.1 module and end in a valid TTCN-3 representation. The order corresponds to the order of execution of the individual transformations:

0) Ignore all type prefixes and all encoding control sections (see note 1). ASN.1 type prefixes consist of tags and encoding prefixes.

0bis) Ignore names of `SEQUENCE OF` and `SET OF` types, values and fields.

1) Ignore any extension markers and exception specifications.

2) Ignore any user defined constraints (see clause 9 of ITU-T Recommendation X.682 [4]).

3) Ignore any contents constraint (see clause 9 of ITU-T Recommendation X.682 [4]).

4) Ignore any pattern constraint (see clause 48.9 of ITU-T Recommendation X.680 [2]).

5) Create equivalent TTCN-3 subtypes for all ASN.1 types constrained using contained subtyping by replacing included types by the set of values they represent. More detailed information on the conversion of ASN.1 type constraints to TTCN-3 subtypes is given in table 4. Table 4 shows the applicability of ASN.1 type constraint mechanisms to different ASN.1 types. Where the cell contains "No", the type constraint is disallowed for the given type. Shaded cells identify type constraints applicable to a given type and text in the cell defines TTCN-3 subtyping mechanisms to be used when transforming constrained ASN.1 types.

6) Execute the `COMPONENTS OF` transformation according to clause 24.4 of ITU-T Recommendation X.680 [2] on any `SEQUENCE` types and according to clause 26.2 on any `SET` types containing the keywords "`COMPONENTS OF`".

7) Replace any `EMBEDDED PDV` type with its associated type obtained by expanding inner subtyping in the associated type of the `EMBEDDED PDV` type (see clause 32.5 of ITU-T Recommendation X.680 [2]) to a full type definition.

8) Replace the `EXTERNAL` type with its associated type obtained by expanding inner subtyping in the associated type of the `EXTERNAL` type (see clause 33.5 of ITU-T Recommendation X.680 [2]) to a full type definition (see note 3).

9) Replace the `CHARACTER STRING` type with its associated type obtained by expanding inner subtyping in the associated type of the `CHARACTER STRING` type (see clause 39.5 of ITU-T Recommendation X.680 [2]) to a full type definition.

10) Replace the `INSTANCE OF` type with its associated type obtained by substituting `INSTANCE OF` DefinedObjectClass by its associated ASN.1 type (see clause C.7 of ITU-T Recommendation X.681 [3]) and replace all ASN.1 types with their TTCN-3 equivalents according to table 3. The resulted type is the TTCN-3 associated type.

11) Ignore any remaining inner subtyping (see note 4).

12) Ignore any named numbers and named bits in ASN.1 types. In ASN.1 values replace any named number by its value and substitute any named bits or sequence of named bits by a bitstring without trailing zeros, where bit positions identified by names present are replaced by "1"s, other bit positions are replaced by "0"s.

13) Replace any selection type with the type referenced by the selection type; if the denoted choice type (the "Type" in clause 29.1 of ITU-T Recommendation X.680 [2]) is a constrained type, the selection has to be done on the parent type of the denoted choice type.

14) Convert any `RELATIVE-OID` type or value to an **`objid`** type or value (see note 5).

15) Replace any of the following restricted character string types with their associated types obtained as (see note 6):

- `BMPString`: **`universal charstring`** (char (0,0,0,0) .. char (0,0,255,255));
- `UTF8String`: **`universal charstring`**;
- `NumericString`: **`charstring`** constrained to the set of characters as given in clause 36.2 of ITU-T Recommendation X.680 [2];
- `PrintableString`: **`charstring`** constrained to the set of characters as given in clause 36.4 of ITU-T Recommendation X.680 [2];
- `TeletexString` and `T61String`: **`universal charstring`** constrained to the set of characters as given in ITU-T Recommendation T.61 [i.2];
- `VideotexString`: **`universal charstring`** constrained to the set of characters as given in ITU-T Recommendation T.100 [9] and ITU-T Recommendation T.101 [10];
- `GraphicString`: **`universal charstring`**;
- `GeneralString`: **`universal charstring`**.

16) Replace any `GeneralizedTime` and `UTCTime` types or values with the type or value of **`charstring`**.

17) Replace any `ObjectDescriptor` type or value by the **`universal charstring`** type or value.

18) Replace any notations for the object class field types (see clause 14 of ITU-T Recommendation X.681 [3]) by the ASN.1 definition they are referring to (see note 8); open types has to be replaced by the metatype "OPEN TYPE" for the purpose of the transformation (and only for that).

19) Replace all information from objects notations (see clause 15 of ITU-T Recommendation X.681 [3]) by the ASN.1 definition they are referencing to.

20) Revert table constraints (see clause 10 of ITU-T Recommendation X.682 [4]) to list subtyping and ignore all relational constraints (see note 7).

21) Replace all occurrences of `NULL` type with the following associated TTCN-3 type (see note 13):

- **`type enumerated`** <*identifier*> { `NULL` },
  where <*identifier*> is the ASN.1 Type reference converted according to clause 8.2, if a synonym of the NULL type is defined, or with

- the nested type definition **`enumerated { NULL }`** <*identifier*>,
  where <*identifier*> is the ASN.1 field identifier, if **`NULL`** is used within a structured type.

22) Replace all references to open types with the metatype "OPEN TYPE" (see note 11).

23) Replace ASN.1 types with their equivalents according to table 3 and ASN.1 values with equivalent TTCN-3 values based on the associated types. Fields of ASN.1 SEQUENCE and SET types identified as OPTIONAL or with a DEFAULT value shall be optional fields in the associated type (see note 12). Missing (i.e. implicitly omitted) optional fields in structured ASN.1 values (of the types (`SET`, `SEQUENCE`, etc.) shall be explicitly omitted in the resulted structured TTCN-3 values (see note 9).

24) Replace the metatype "OPEN TYPE" by **`anytype`**.

NOTE 1: Associated types contain abstract information only, thus do not contain alone all information needed for the correct encoding of values based on ASN.1 types. The way of handling the extra information needed by the test system to provide correct encoding and/or decoding is implementation dependent and remains hidden for the user; this knowledge is not required to make valid TTCN-3 declarations or assignments involving imported ASN.1 types and values.

NOTE 2: When importing `ENUMERATED` types, integer numbers assigned by the user to enumerations will also be imported.

NOTE 3: The data-value field of the `EXTERNAL` type may be encoded as a single-ASN1-type, octet-aligned or arbitrary (see clause 8.18.1 of ITU-T Recommendation X.690 [6]) at the discretion of the encoder; if the user wants to enforce one given form of encoding or wants to allow only one specific encoding form at matching, it has to use the appropriate encoding attribute for the type or the given constant, variable, template or template field (see clause 11.3).

NOTE 4: Inner subtyping has to be taken into account by the user when defining TTCN-3 values or templates based on an ASN.1 type constrained by inner subtyping.

NOTE 5: Equivalence with the **`objid`** type is limited to the syntax to be used for value notations only. When encoding/decoding an **objid** value retrieved from an ASN.1 `RELATIVE-OID` value using an ASN.1 encoding rule, the encoding/decoding will occur according to rules specified for the `RELATIVE-OID` type.

NOTE 6: `VisibleString`, `IA5String` and `UniversalString` have their equivalent TTCN-3 types and are replaced directly.

NOTE 7: Relational constraints have to be taken into account by the user when declaring values and templates (also may be handled by tools implicitly).

NOTE 8: This replacement does not affect constraints applied to the "notation for the object class field type" itself.

NOTE 9: Missing optional fields in values of structured ASN.1 types (`SET`, `SEQUENCE`, `EXTERNAL`, etc.) are equivalent to explicitly omitted fields in structured TTCN-3 values.

EXAMPLE 2:

```
module MyTTCNModule
{
        import from MyASN1module language "ASN.1:2002" all;

        const Bmessage MyTTCNConst:= johnValues;
        const DefinedValuesForField1 Value1:= 1;
}
```

NOTE 10: ASN.1 definitions other than types and values (i.e. information object classes or information object sets) are not directly accessible from the TTCN-3 notation. Such definitions will be resolved to a type or value within the ASN.1 module before they can be referenced from within the TTCN-3 module.

NOTE 11: The metatype "OPEN TYPE" is just used to describe the transformation process. It does not exist neither before nor after the transformation.

NOTE 12: Most ASN.1 encoding rules require that fields with DEFAULT values are omitted in the encoded message when their actual contents equal to the default values. However, in TTCN-3, it may be required that the default value is also encoded and present. If fields with default values are omitted or present in the encoded message, is a TTCN-3 test system runtime configuration option. It is also a TTCN-3 test system runtime configuration option, if fields with default values missing in the received encoded message are omitted or substituted by their default values in the abstract TTCN-3 value (the decoded message).

NOTE 13: The associated type for the ASN.1 NULL type is introduced to specify the TTCN-3 value notation for this type. The encoding/decoding of NULL values and fields have to be as defined for the NULL type in the ASN.1 Recommendations (see e.g. in ITU-T Recommendations X.690 [6], X.691 [7] and X.693 [8]). Also, the restriction in clause 7.1.3 of ES 201 873-1 [1] (relational operators) [1] that only values of the same enumerated types are allowed to be compared, does not apply to imported ASN.1 NULL types.

Table 4: ASN.1 type constraint to TTCN-3 subtype conversions

Type (or derived from such a type by tagging or subtyping)	Single Value	Contained Subtype (h)	Value Range	Size Constraint	Permitted Alphabet	Type Constraint	Inner Subtyping (see i)	Pattern Constraint	User defined constraint	Table constraint (see k)	Relation constraint (see k)	Content constraint
Bit String	list	single value: list, size: length	No	length	No	No	No	No	ignore	No	No	ignore
Boolean	list	list	No	No	No	No	No	No	ignore	No	No	No
Choice	list	list	No	No	No	No	convert to full type	No	ignore	No	No	No
Embedded-pdv (see a)	list	No	No	No	No	No	convert to full type	No	ignore	No	No	No
Enumerated	list	list	No	No	No	No	No	No	ignore	No	No	No
External (see a)	list	No	No	No	No	No	convert to full type	No	ignore	No	No	No
Instance-of (see a and b)	list	list	No	No	No	No	convert to full type	No	ignore	No	No	No
Integer	list	single value: list, value range: range	range	No	No	No	No	No	ignore	No	No	No
Null	ignore	ignore	No	No	No	No	No	No	ignore	No	No	No
Object class field type	(see c)	(see c)	No	No	No	No	No	No	ignore	list	ignore	No
Object Descriptor (see e)	list	single value: list, size: length, perm.alphabet: range	No	length	range	No	No	No	ignore	No	No	No
Object Identifier	list	*list*	No	No	No	No	No	No	ignore	No	No	No
Octet String	list	single value: list, size: length	No	length	No	No	No	No	ignore	No	No	ignore
open type	No	No	No	No	No	anytype with list constraint	No	No	ignore	No (see m)	No (see m)	No
Real	list	single value: list, value range: range	range	No	No	No	convert to full type	No	ignore	No	No	No
Relative Object Identifier (see d)	list	list	No	No	No	No	No	No	ignore	No	No	No
Restricted Character String Types	list	single value: list, size: length, perm.alphab.: range	range	length	range	No	No	Ignore (see g)	ignore	No	No	No
Sequence	list	list	No	No	No	No	convert to full type	No	ignore	No	No	No
Sequence-of	list	single value: list, value range: range	No	length	No	No	convert to full type	No	ignore	No	No	No

Type (or derived from such a type by tagging or subtyping)	Single Value	Contained Subtype (h)	Value Range	Size Constraint	Permitted Alphabet	Type Constraint	Inner Subtyping (see i)	Pattern Constraint	User defined constraint	Table constraint (see k)	Relation constraint (see k)	Content constraint
Set	list	list	No	No	No	No	convert to full type	No	ignore	No	No	No
Set-of	list	single value: list, value range: range	No	length	No	No	convert to full type	No	ignore	No	No	No
Time Types (see a)	list	list	No	No	No	No	No	No	ignore	No	No	No
Unrestricted Character String Type (see a)	list	No	No	length (applied to field "string-value")	No	No	convert to full type	No	ignore	No	No	No

NOTES:
(a) These types are seen from TTCN-3 as being equivalent to their associated types.
(b) Type-id field of the associated type for Instance of shall be replaced by the type of the &id field the value field is anytype (annex C of ITU-T Recommendation X.681 [3]).
(c) Replaced by the referenced type, thus applicable as to the referenced type.
(d) Seen as object identifier from TTCN-3.
(e) Its associated type is a restricted character string type.
(f) Open type is replaced by **`anytype`**.
(g) Character patterns can only be used in constants, variables, templates and module parameters in TTCN-3 but cannot be used for subtyping.
(h) Contained subtype constraints shall be replaced by literal constraints at import.
(i) Information in this column relates to the TTCN-3 views of ASN.1 definitions. Encoding/decoding shall be according to the root type, thus extra information for encoding also has to be stored which are not shown in this table.
(j) Applicable to notations for the object class field type only.
(k) Applicable when the open type is defined using the notation for the object class field type (see above).

9.2 Scope of ASN.1 identifiers

Imported ASN.1 identifiers follow the same scope rules as imported TTCN-3 types and values (see clause 5.2 of ES 201 873-1 [1]).

10 Parameterization in ASN.1

It is not permitted to reference parameterized ASN.1 definitions from within the TTCN-3 module. However, non-parameterized ASN.1 definitions may reference parameterized ASN.1 definitions by providing the actual parameters. Such ASN.1 definitions can be imported to and used in TTCN-3 and when importing all definitions of an ASN.1 module, such definitions shall also be imported.

11 Defining ASN.1 message templates

11.1 General

Imported ASN.1 values can be used as messages in both **send** and **receive** operations.

EXAMPLE:

```
MyASN1module DEFINITIONS ::=
BEGIN
    -- ASN.1 Module definition

    -- The message definition
    MyMessageType::= SEQUENCE
    {   field1  [1] IA5STRING,          -- Like TTCN-3 charstring
        field2  [2] INTEGER OPTIONAL,   -- like TTCN-3 integer
        field3  [4] Field3Type,         -- Like TTCN-3 record
        field4  [5] Field4Type          -- Like TTCN-3 array
    }

    Field3Type::= SEQUENCE {field31 BIT STRING, field32 INTEGER, field33 OCTET STRING},
    Field4Type::= SEQUENCE OF BOOLEAN


    -- may have the following value
    myValue MyMessageType::=
    {
        field1      "A string", -- IA5STRING
        field2      123, -- INTEGER
        field3      {field31 '11011'B, field32 456789, field33 'FF'O}, -- SEQUENCE
        field4      {true, false} - SEQUENCE OF
    }
END
```

11.2 Receiving messages based on ASN.1 types

Matching mechanisms are not supported by the ASN.1 syntax. Thus, if matching mechanisms are wished to be used with a received ASN.1 message, a TTCN-3 template shall be defined based on the ASN.1 type and this shall be used in the receiving operation.

EXAMPLE:

```
import from MyASN1module language "ASN.1:2002" {
    type myMessageType
}

// a message template using matching mechanisms is defined within a TTCN-3 module
template myMessageType  MyValue:=
{
    field1 :=           pattern"A?tr*g",
    field2 :=           *,
    field3.field31 :=   '110??'B,
    field3.field32 :=   ?,
    field3.field33 :=   'F?'O,
    field4.[0] :=       true,
    field4.[1] :=       false
}

// the following syntax is equally valid
template myMessageType  MyValue:=
{
    field1 := pattern"A?tr*g",      // string with wildcards
    field2 := *,                        // any integer or none at all
    field3 := {'110??'B, ?, 'F?'O},
    field4 := {?, false}
}
```

11.3 Ordering of template fields

When TTCN-3 templates are used for ASN.1 types the significance of the order of the fields in the template will depend on the type of ASN.1 construct used to define the message type. For example: if `SEQUENCE` or `SEQUENCE OF` is used then the message fields shall be sent or matched in the order specified in the template. If `SET` or `SET OF` is used then the message fields may be sent or matched in any order.

12 Encoding information

12.1 General

TTCN-3 allows references to encoding rules and variations within encoding rules to be associated with various TTCN-3 language elements. It is also possible to define invalid encodings. This encoding information is specified using the **`with`** statement (see clause 27 of ES 201 873-1 [1]) according to the following syntax:

EXAMPLE:

```
module MyModule
{
     :
    import from MyASN1module language "ASN.1:2002" {
        type myMessageType
    }
    with {
        encode "PER-BASIC-ALIGNED:1997" // All instances of MyMessageType should be encoded
using PER:1997
    }
     :
} // end module
with { encode "BER:1997" } // Default encoding for the entire module (test suite) is BER:1997
```

12.2 ASN.1 encoding attributes

The following strings are the predefined (standardized) encoding attributes for the current version of ASN.1:

a) "BER:2002" means encoded according to ITU-T Recommendation X.690 [6] (BER);

b) "CER:2002" means encoded according to ITU-T Recommendation X.690 [6] (CER);

c) "DER:2002" means encoded according to ITU-T Recommendation X.690 [6] (DER);

d) "PER-BASIC-UNALIGNED:2002" means encoded according to (Unaligned PER) ITU-T Recommendation X.691 [7];

e) "PER-BASIC-ALIGNED:2002" means encoded according to ITU-T Recommendation X.691 [7] (Aligned PER);

f) "PER-CANONICAL-UNALIGNED:2002" means encoded according to (Canonical Unaligned PER) ITU-T Recommendation X.691 [7];

g) "PER-CANONICAL-ALIGNED:2002" means encoded according to ITU-T Recommendation X.691 [7] (Canonical Aligned PER);

h) "BASIC-XER:2002" means encoded according to ITU-T Recommendation X.693 [8] (Basic XML encoding rules);

i) "CANONICAL-XER:2002" means encoded according to ITU-T Recommendation X.693 [8] (Canonical XML encoding rules);

j) "EXTENDED-XER:2002" means encoded according to Amd. 1 ITU-T Recommendation X.693 [8] (Extended XML encoding rules).

The encodings of previous ASN.1 versions rule (e.g. 1988, 1994 or 1997) can be used as well. In this case, the date has to be replaced accordingly. For example, for ASN.1 1997 the following encoding attributes apply: "BER:1997", "CER:1997", "DER:1997", "PER-BASIC-UNALIGNED:1997", "PER-BASIC-ALIGNED:1997", "PER-CANONICAL-UNALIGNED:1997" and "PER-CANONICAL-ALIGNED:1997".

12.3 ASN.1 variant attributes

The following strings are predefined (standardized) variant attributes. They have predefined meaning only when applied jointly with predefined ASN.1 encoding attributes (see clause 12.2). Handling of these predefined attributes, when applied jointly with other attributes or to an TTCN-3 object without an attribute, is out of scope of the present document (see note):

a) "length form 1" means, that the given value shall only be encoded and decoded using the short form of the length octets (see clause 8.1.3 of ITU-T Recommendation X.690 [6]) in case of BER, CER and DER encodings or the single octet length determinant (see clause 10.9 of Recommendation X.691 [7]) in case of any form of the PER encoding.

b) "length form 2" means, that the given value shall only be encoded and decoded using the long form of the length octets (see clause 8.1.3 of ITU-T Recommendation X.690 [6]) in case of BER, CER and DER encodings or the two octets length determinant (see clause 10.9 of ITU-T Recommendation X.691 [7]) in case of any form of the PER encoding.

c) "length form 3" means, that the given value shall only be encoded and decoded using the indefinite form of the length octets (see clause 8.1.3 of ITU-T Recommendation X.690 [6]) in case of BER, CER and DER encodings.

d) "REAL base 2" means that the given value shall be encoded or matched according to the REAL binary encoding form. This attribute can be used on constants, variables or templates only and when used on any kind of a grouping (e.g. to groups or to the whole import statement) it shall have effect on these TTCN-3 objects only.

e) "single-ASN1-type", "octet-aligned" and "arbitrary" mean, that the given value based on an ASN.1 EXTERNAL type shall be encoded using the specified by the attribute encoding form or matched if received with the specified choice only (see clause 8.18 of ITU-T Recommendation X.690 [6]). This attribute can be used on imported ASN.1 EXTERNAL types and constants, variables, templates or template fields based on these types only; when used on any kind of a grouping (e.g. to groups or to the whole import statement) it shall have effect on these TTCN-3 objects only. If the conditions set in clauses 8.18.6 to 8.18.8 of ITU-T Recommendation X.690 [6] and the specified attribute do not met, this shall cause a run-time error.

f) "TeletexString" means that the given value shall be encoded and decoded as the ASN.1 type TeletexString (see clause 8.20 of ITU-T Recommendation X.690 [6] and clause 26 of ITU-T Recommendation X.691 [7]).

g) "VideotexString" means that the given value shall be encoded and decoded as the ASN.1 type VideotexString (see clause 8.20 of ITU-T Recommendation X.690 [6] and clause 26 of ITU-T Recommendation X.691 [7]).

h) "GraphicString" means that the given value shall be encoded and decoded as the ASN.1 type GraphicString (see clause 8.20 of ITU-T Recommendation X.690 [6] and clause 26 of ITU-T Recommendation X.691 [7]).

i) "GeneralString" means that the given value shall be encoded and decoded as the ASN.1 type GeneralString (see clause 8.20 of ITU-T Recommendation X.690 [6] and clause 26 of ITU-T Recommendation X.691 [7]).

NOTE: These attributes may be reused in implementation specific encoding rules with a different meaning than specified in the current clause, may be ignored or a warning/error indication may be given. However, the strategy to be applied is implementation dependent.

Application of these variant attributes may lead to invalid ASN.1 encoding (e.g. using the indefinite length form to primitive values in BER or not using the minimum necessary number of length octets). This is allowed intentionally and users shall allocate these variant attributes to constants, variables, templates or template fields used for receiving cautiously.

Annex A (normative): Additional BNF and static semantics

When ASN.1 is supported, rules defined in annex A of ES 201 873-1 [1] shall apply, supplemented by the BNF and semantic rules specified in this annex.

In addition to those listed in table 3 "List of TTCN-3 terminals which are reserved words" (see clause A.1.5 of ES 201 873-1 [1]), the word **objid** shall also be a TTCN-3 reserved word (keyword).

Amendments to clause A.1.6 of ES 201 873-1 [1] are specified in the subsequent clauses of this annex.

A.1 New productions for ASN.1 support

```
1000. DefinitiveIdentifier ::= ObjectIdentifierKeyword "{" DefinitiveObjIdComponentList "}"
1001. ObjectIdentifierKeyword ::= "objid"
1002. DefinitiveObjIdComponentList ::= {DefinitiveObjIdComponent}+
1003. DefinitiveObjIdComponent ::= NameForm |
                                   DefinitiveNumberForm |
                                   DefinitiveNameAndNumberForm
1004. NameForm ::= Identifier
1005. DefinitiveNumberForm ::= Number
1006. DefinitiveNameAndNumberForm ::= Identifier "(" DefinitiveNumberForm ")"
1007. ObjectIdentifierValue ::= ObjectIdentifierKeyword "{" ObjIdComponentList "}"
1008. ObjIdComponentList ::= {ObjIdComponent}+
1009. ObjIdComponent ::= DefinitiveObjIdComponent | ReferencedValue
/* STATIC SEMANTICS - ReferencedValue shall be an object identifier value */
1010. NameAndNumberForm ::= Identifier "(" NumberForm | ReferencedValue ")"
/* STATIC SEMANTICS - ReferencedValue shall be an integer value */
```

A.2 Amended core language BNF productions and static semantics

In addition to those specified in clause A.1.5 of ES 201 873-1 [1], when the use of ASN.1 is supported, the keywords listed in clause 11.18 of ITU-T Recommendation X.680 [2], after applying the name conversion rules defined in clause 8.2 of the present document, shall not be used as identifiers in a TTCN-3 module. ASN.1 keywords shall follow the requirements of ITU-T Recommendation X.680 [2].

In addition to those listed in table A.3 of ES 201 873-1 [1], when the use of ASN.1 is supported, "**objid"** shall be a reserved word.

Additions to clause A.1.6 of ES 201 873-1 [1] are identified by underlined font, deletions are identified by strikethrough font.

```
5. GlobalModuleId ::= ModuleIdentifier [Dot DefinitiveIdentifier]
51. ValueOrRange ::= RangeDef | ConstantExpression
/* STATIC SEMANTICS - RangeDef production shall only be used with integer, charstring, universal
charstring, or float or object identifier based types */
/* STATIC SEMANTICS - When subtyping charstring or universal charstring range and values shall not
be mixed in the same SubTypeSpec */
144 LowerBound ::= SingleConstExpression | Minus InfinityKeyword
145 UpperBound ::= SingleConstExpression | InfinityKeyword
/* STATIC SEMANTICS - LowerBound and UpperBound shall evaluate to types integer, charstring,
universal charstring, or float or object identifier. In case LowerBound or UpperBound evaluates to
types charstring, or universal charstring or object identifier, only SingleConstExpression may be
present and in case of charstring and universal charstring types the string length shall be 1*/
438. PredefinedType ::= BitStringKeyword |
                        BooleanKeyword |
                        CharStringKeyword |
                        UniversalCharString |
                        IntegerKeyword |
                        OctetStringKeyword |
                        HexStringKeyword |
                        VerdictTypeKeyword |
```

```
                        FloatKeyword |
                        AddressKeyword |
                        DefaultKeyword |
                        AnyTypeKeyword |
                        ObjectIdentifierKeyword
459. PredefinedValue ::= BitStringValue |
                         BooleanValue |
                         CharStringValue |
                         IntegerValue |
                         OctetStringValue |
                         HexStringValue |
                         VerdictTypeValue |
                         EnumeratedValue |
                         FloatValue |
                         AddressValue |
                         OmitValue |
                         ObjectIdentifierValue
```

Annex B (normative): Additional Pre-defined TTCN-3 functions

Void.

Annex C (informative): Predefined object identifier components

ITU-T Recommendation X.660 [11] defines the tree of object identifier components shown below. Only the object identifier components defined in ITU-T Recommendation X.660 [11] shall use the name form (without defining the numerical value of the component) in object identifier value notations. These predefined components have specified numerical values when used at their predefined positions only. Names in *italic* are reserved for historical reasons, therefore their use in TTCN-3 codes is deprecated, but it is recommended that TTCN-3 test systems are able to recognize them and substitute with the correct numerical value.

NOTE: Names below are given according to the TTCN-3 syntax, i.e. all dash characters are replaced by underscore characters.

itu_t(0), *ccitt(0)*, itu_r(0)
- recommendation(0)
  - a(1)
  - b(2)
  - c(3)
  - d(4)
  - e(5)
  - f(6)
  - g(7)
  - h(8)
  - i(9)
  - j(10)
  - k(11)
  - l(12)
  - m(13)
  - n(14)
  - o(15)
  - p(16)
  - q(17)
  - r(18)
  - s(19)
  - t(20)
  - u(21)
  - v(22)
  - x(24)
  - y(25)
  - z(26)
- question(1)
- administration(2)
- network_operator(3)
- identified_organization(4)
- r_recommendation(5)

iso(1)
- standard(0)
- registration_authority(1)
- member_body(2)
- identified_organization(3)

joint_iso_itu_t(2), *joint_iso_ccitt(2)*

Annex D (informative): Deprecated features

D.1 Using **sizeof** for **objid** values and templates

The previous version (up to and including 3.1.1) of the standard allowed to use the **sizeof** predefined function to objid values. This feature is replaced by the use of the length of predefined function (see clause 7.2.6.1) and, in consequence, is deprecated in this edition of the standard and may be fully removed in the next published edition.

D.2 The decompose function

The previous version (up to and including 3.1.1) of the standard defined the **decomp** predefined function. This feature is replaced by the use of the **substr** predefined function for **objid** values and templates (see clause 7.2.6.2) and, in consequence, is deprecated in this edition of the standard and may be fully removed in the next published edition.

Annex E (informative): Bibliography

- ITU-T Recommendation X.680 (07/94): "Information technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation".
- ITU-T Recommendation X.680 (12/97): "Information technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation".
- ITU-T Recommendation X.681 (07/94): "Information technology - Abstract Syntax Notation One (ASN.1): Information object specification".
- ITU-T Recommendation X.681 (12/97): "Information technology - Abstract Syntax Notation One (ASN.1): Information object specification".
- ITU-T Recommendation X.682 (07/94): "Information technology - Abstract Syntax Notation One (ASN.1): Constraint specification".
- ITU-T Recommendation X.682 (12/97): "Information technology - Abstract Syntax Notation One (ASN.1): Constraint specification".
- ITU-T Recommendation X.683 (07/94): "Information technology - Abstract Syntax Notation One (ASN.1): Parameterization of ASN.1 specifications".
- ITU-T Recommendation X.683 (12/97): "Information technology - Abstract Syntax Notation One (ASN.1): Parameterization of ASN.1 specifications".
- ISO/IEC 6429 (1992): "Information technology - Control functions for coded character sets".
- ITU-T Recommendation T.50 (1992): "International Reference Alphabet (IRA) (Formerly International Alphabet No. 5 or IA5) - Information technology - 7-bit coded character set for information interchange".
- ISO/IEC 8859-1: "Information technology - 8-bit single-byte coded graphic character sets - Part 1: Latin alphabet No. 1".
- A repository of Object IDentifiers (OIDs) is freely available at http://www.oid-info.com/.

NOTE: References to ITU-T Recommendations include the Recommendation and all Amendments and Corrigenda published to the Recommendation except when specified otherwise in other parts of the present document.

In the present document the following Amendments and Corrigenda were considered for X.680 (2002):

- ITU-T Recommendation X.680 Amendment 1 (10/2003): "Information technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation; Amendment 1: Support for EXTENDED-XER".
- ITU-T Recommendation X.680 Amendment 2 (08/2004): "Information technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation; Amendment 2: Alignment with changes made to ITU-T Rec. X.660 | ISO/IEC 9834-1 for identifiers in object identifier value notation".
- ITU-T Recommendation X.680 Corrigendum 1 (05/2005): "Information technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation; Technical Corrigendum 1".
- ITU-T Recommendation X.680 Amendment 3 (06/2006): "Information technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation; Amendment 3: Time type support".
- ITU-T Recommendation X.680 Amendment 4 (05/2007): "Information technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation; Amendment 4".

History

Document history		
V3.1.1	June 2005	Publication
V3.3.2	April 2008	Publication
V4.1.1	March 2009	Membership Approval Procedure MV 20090522: 2009-03-24 to 2009-05-22
V4.1.1	June 2009	Publication