International Telecommunication Union

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

Z.167
(11/2008)

SERIES Z: LANGUAGES AND GENERAL SOFTWARE ASPECTS FOR TELECOMMUNICATION SYSTEMS

Formal description techniques (FDT) – Testing and Test Control Notation (TTCN)

Testing and Test Control Notation version 3: TTCN-3 mapping from ASN.1

Recommendation ITU-T Z.167

ITU-T Z-SERIES RECOMMENDATIONS

LANGUAGES AND GENERAL SOFTWARE ASPECTS FOR TELECOMMUNICATION SYSTEMS

FORMAL DESCRIPTION TECHNIQUES (FDT)	
Specification and Description Language (SDL)	Z.100–Z.109
Application of formal description techniques	Z.110–Z.119
Message Sequence Chart (MSC)	Z.120–Z.129
User Requirements Notation (URN)	Z.150–Z.159
Testing and Test Control Notation (TTCN)	**Z.160–Z.179**
PROGRAMMING LANGUAGES	
CHILL: The ITU-T high level language	Z.200–Z.209
MAN-MACHINE LANGUAGE	
General principles	Z.300–Z.309
Basic syntax and dialogue procedures	Z.310–Z.319
Extended MML for visual display terminals	Z.320–Z.329
Specification of the man-machine interface	Z.330–Z.349
Data-oriented human-machine interfaces	Z.350–Z.359
Human-machine interfaces for the management of telecommunications networks	Z.360–Z.379
QUALITY	
Quality of telecommunication software	Z.400–Z.409
Quality aspects of protocol-related Recommendations	Z.450–Z.459
METHODS	
Methods for validation and testing	Z.500–Z.519
MIDDLEWARE	
Processing environment architectures	Z.600–Z.609

For further details, please refer to the list of ITU-T Recommendations.

Recommendation ITU-T Z.167

Testing and Test Control Notation version 3: (TTCN-3) mapping from ASN.1

Summary

Recommendation ITU-T Z.167 defines a normative way of using ASN.1 as defined in Recommendations ITU-T X.680, ITU-T X.681, ITU-T X.682 and ITU-T X.683 with TTCN-3. The harmonization of other languages with TTCN-3 is not covered by this Recommendation.

Source

Recommendation ITU-T Z.167 was approved on 29 November 2008 by ITU-T Study Group 17 (2009-2012) under Recommendation ITU-T A.8 procedures.

CONTENTS

Recommendation ITU-T Z.167

Testing and Test Control Notation version 3: (TTCN-3) mapping from ASN.1

1 Scope

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

2 References

The following ITU-T Recommendations and other references contain provisions which, through reference in this text, constitute provisions of this Recommendation. At the time of publication, the editions indicated were valid. All Recommendations and other references are subject to revision; users of this Recommendation are therefore encouraged to investigate the possibility of applying the most recent edition of the Recommendations and other references listed below. A list of the currently valid ITU-T Recommendations is regularly published. The reference to a document within this Recommendation does not give it, as a stand-alone document, the status of a Recommendation.

[ITU-T T.100]	Recommendation ITU-T T.100 (1988), *International information exchange for interactive Videotex*.
[ITU-T T.101]	Recommendation ITU-T T.101 (1994), *International interworking for Videotex services*.
[ITU-T X.660]	Recommendation ITU-T X.660 (2004) | ISO/IEC 9834-1:2005, *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*.
[ITU-T X.680]	Recommendation ITU-T X.680 (2002) | ISO/IEC 8824-1:2002, *Information technology – Abstract Syntax Notation One (ASN.1): Specification of basic notation*.
[ITU-T X.681]	Recommendation ITU-T X.681 (2002) | ISO/IEC 8824-2:2002, *Information technology – Abstract Syntax Notation One (ASN.1): Information object specification*.
[ITU-T X.682]	Recommendation ITU-T X.682 (2002) | ISO/IEC 8824-3:2002, *Information technology – Abstract Syntax Notation One (ASN.1): Constraint specification*.
[ITU-T X.683]	Recommendation ITU-T X.683 (2002) | ISO/IEC 8824-4:2002, *Information technology – Abstract Syntax Notation One (ASN.1): Parameterization of ASN.1 specifications*.
[ITU-T X.690]	Recommendation ITU-T X.690 (2002) | ISO/IEC 8825-1:2002, *Information technology – ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)*.
[ITU-T X.691]	Recommendation ITU-T X.691 (2002) | ISO/IEC 8825-2:2002, *Information technology – ASN.1 encoding rules: Specification of Packed Encoding Rules (PER)*.
[ITU-T X.693]	Recommendation ITU-T X.693 (2001) | ISO/IEC 8825-4:2002, *Information technology – ASN.1 encoding rules: XML Encoding Rules (XER)*.

[ITU-T Z.161] Recommendation ITU-T Z.161 (2007), *Testing and Test Control Notation version 3: (TTCN-3) core language.*

technically aligned with:

ETSI ES 201 873-1 V3.3.1:2005, *Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language*.

3 Definitions and abbreviations

3.1 Definitions

For the purposes of this Recommendation, the terms and definitions given in [ITU-T Z.161], [ITU-T X.660] and the following apply:

3.1.1 metatype "OPEN TYPE": This term is used to explain the ASN.1 to TTCN-3 conversion process. It does not exist in the input ASN.1 module or the output TTCN-3 module.

3.1.2 root type: The definition in [ITU-T Z.161] 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 this Recommendation, the abbreviations given in [ITU-T Z.161] and the following apply:

ASN.1	Abstract Syntax Notation One
BNF	Backus-Naur Form

4 Introduction

When using ASN.1 with TTCN-3, all features of TTCN-3 and statements given in clause 4 of [ITU-T Z.161] do apply. In addition, when supporting this part of the Recommendation, 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 Recommendation defines the capabilities required in addition to those specified in [ITU-T Z.161] 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 this Recommendation.

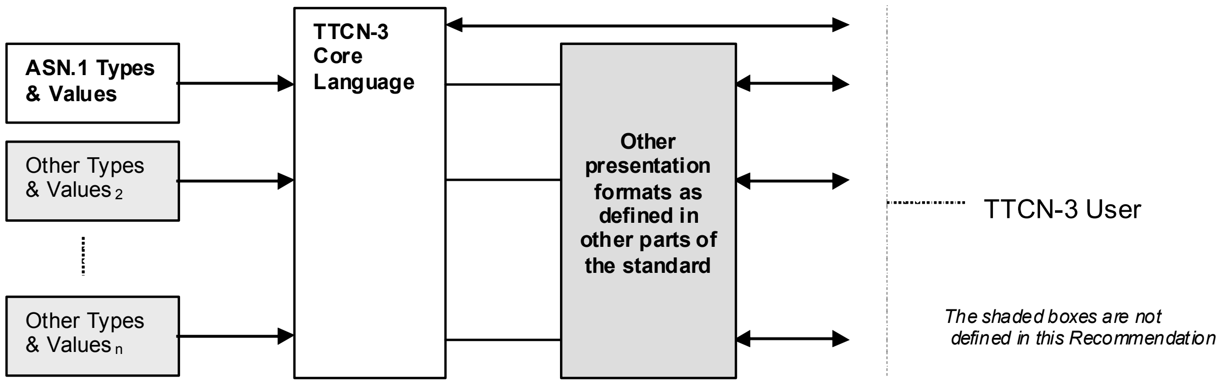
ASN.1 Types & Values
Other Types & Values 2
Other Types & Values n
TTCN-3 Core Language
Other presentation formats as defined in other parts of the standard
TTCN-3 User
The shaded boxes are not defined in this Recommendation

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 this Recommendation shall be implemented consistently with the requirements given in this Recommendation and in [ITU-T Z.161].

5 General

TTCN-3 provides a clean interface for using ASN.1 definitions (as specified in [ITU-T X.680], [ITU-T X.681], [ITU-T X.682] and [ITU-T X.683]) in TTCN-3 modules. XML-related capabilities of ASN.1 are currently not considered.

6 Amendments to the core language

Void.

7 Additional TTCN-3 types

7.1 General

The TTCN-3 types summarized in Table 1 shall be supported in addition to those specified in clause 6 of [ITU-T Z.161].

Table 1 – Overview of TTCN-3 types

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

7.2 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 X.680], 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 X.660] (e.g., with a node beneath the root not defined in [ITU-T X.660]).

NOTE 2 – The name form of object identifier components shall be used only for components defined in [ITU-T X.660]. These predefined object identifier components are given in Appendix C for information. In case of any conflict between [ITU-T X.660] and Appendix C, 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 X.680] and TTCN-3 modules in clause 8.1 of the core language standard [ITU-T Z.161]). 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 3 – 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 4 – Rules to resolve name clashes caused by imports are defined in clause 8.2.3.1 of the core language Recommendation [ITU-T Z.161].

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 is composed of an object identifier node as shown on Figure 2 (see also Appendix 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.

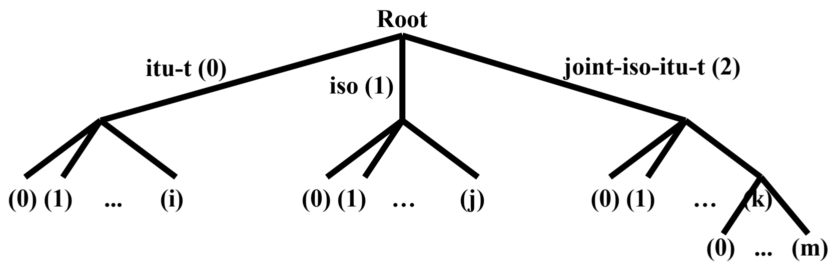

Figure 2 – Object identifier tree

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 [ITU-T Z.161], 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 [ITU-T Z.161], 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 [ITU-T Z.161] and in this clause shall apply. In case of inconsistency, this Recommendation 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 X.660].

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 [ITU-T Z.161]) is unable to resolve such kinds 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 a 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 [ITU-T Z.161] and in this clause shall apply. In case of inconsistency, this Recommendation 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 [ITU-T Z.161], 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	Complem entedList	ValueList	AnyValue(?)	AnyValueOrNone(*)	Range	Superset	Subtype	Pattern	AnyElement(?)	AnyElementsOrNone(*)	Permutation	LengthRestriction	IfPresent
objid	Yes	Yes	Yes	Yes	Yes	Yes[1]	Yes				Yes	Yes		Yes	Yes[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 [ITU-T Z.161].

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 [ITU-T Z.161] also shall apply to `objid` templates.

NOTE – This also means that only the concrete node values on the list are to be considered but not the subtrees identified by them. In other words, 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 [ITU-T Z.161], 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, 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 [ITU-T Z.161], can also be used within **objid** templates. When applied to **objid**s, it replaces exactly one component.

The inside value matching mechanism *AnyElementsOrNone*, in addition to types listed in clause B.1.3.2 of [ITU-T Z.161], can also be used within **objid** templates. When applied to **objid**s, 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 [ITU-T Z.161], can also be used with **objid** templates. When applied to **objid**s, 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 [ITU-T Z.161] 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 an 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 addition to the input parameter types given in clause C.28 of [ITU-T Z.161], 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:

Error causes are:

- `inpar` can match **`objid`** values with a 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 Substring function

When ASN.1 is supported, the **`substr`** predefined function shall support **`objid`** types, i.e., it shall allow **`objid`** as the 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 [ITU-T Z.161] 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.

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`**

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

ASN.1 type	Maps to TTCN-3 equivalent
`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 an 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*bis*).	

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.

8.1*bis* 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 ([b-ITU-T X.208] *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 Recommendation and all published amendments and technical corrigenda published so far complementing the base Recommendation). The only purpose when including them in this Recommendation 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 X.680] 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 [b-ITU-T X.208] (*Blue Book*).

NOTE 3 – References to the 1994, 1997 and the *Blue Book* (1988) versions of ASN.1 can be found in Appendix D.

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, and import the resulted definitions afterwards based on the rules defined below.

NOTE 4 – ASN.1 value sets are semantically equivalent to subtypes. Hence, in this Recommendation, 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 X.680] within the ASN.1 module(s).

EXAMPLE 1:

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


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

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

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

The ASN.1 module shall conform to the syntax and semantics of [ITU-T X.680], [ITU-T X.681], [ITU-T X.682] and [ITU-T X.683]. Once declared and imported, ASN.1 types and values may be used within TTCN-3 modules in a similar way than ordinary 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 to 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.

NOTE – Associated ASN.1 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 has significance from the point of view of TTCN-3 (also called the TTCN-3 view).

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:

1) Ignore any extension markers and exception specifications.

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

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

4) Ignore any pattern constraint (see clause 47.9 of [ITU-T X.680]).

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 X.680] on any `SEQUENCE` types and according to clause 26.2 of [ITU-T X.680] 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 33.5 of [ITU-T X.680]) 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 34.5 of [ITU-T X.680]) 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 40.5 of [ITU-T X.680]) 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 X.681]) 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 X.680]) 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 37.2 of [ITU-T X.680];
- `PrintableString`: **`charstring`** constrained to the set of characters as given in clause 37.4 of [ITU-T X.680];
- `TeletexString` and `T61String`: **`universal charstring`** constrained to the set of characters as given in [b-ITU-T T.61];
- `VideotexString`: **`universal charstring`** constrained to the set of characters as given in [ITU-T T.100] and [ITU-T T.101];
- `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 X.681]) by the ASN.1 definition they are referring to (see Note 8); open types have to be replaced by the metatype "OPEN TYPE" for the purpose of the transformation (and only for that).

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

20) Revert table constraints (see clause 10 of [ITU-T X.682]) 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 (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 they do not contain alone all the 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 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 X.690]) 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 shall 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 effect 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 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 are 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, it 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 X.690], [ITU-T X.691] and [ITU-T X.693]). Also, the restriction in clause 7.1.3 (relational operators) of [ITU-T Z.161] 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 [i]	Pattern constraint	User defined constraint	Table constraint [k]	Relation constraint [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[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[a]	list	No	No	No	No	No	convert to full type	No	ignore	No	No	No
Instance-of[a], [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	[c]	[c]	No	No	No	No	No	No	ignore	list	ignore	No
Object Descriptor[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[f]	No	No	No	No	No	anytype with list constraint	No	No	ignore	No [m]	No [m]	No

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 [i)]	Pattern constraint	User defined constraint	Table constraint [k)]	Relation constraint [k)]	Content constraint
Real	list	single value: list, value range: range	range	No	No	No	convert to full type	No	ignore	No	No	No
Relative Object Identifier[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 [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
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[a)]	list	List	No	No	No	No	No	No	ignore	No	No	No
Unrestricted Character String Type[a)]	list	No	No	length (applied to field "string-value")	No	No	convert to full type	No	ignore	No	No	No

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

a)	These types are seen from TTCN-3 as being equivalent to their associated types.
b)	Type-id field of the associated type for Instances-of shall be replaced by the type of the &id field the value field is anytype (Annex C of [ITU-T X.681]).
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 can not 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, this is not shown in this table.
k)	Applicable to notations for the object class field type only.
m)	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 [ITU-T Z.161]).

10 Parameterization in ASN.1

It is permitted to reference parameterized ASN.1 type and value definitions from within the TTCN-3 module. However, all ASN.1 parameterized definitions used in a TTCN-3 module shall be provided with actual parameters (open types are not permitted) and the actual parameters provided shall be resolvable at compile-time.

The TTCN-3 core language does not support the import of ASN.1 definitions, which employ uniquely ASN.1 specific objects as formal or actual parameter(s). ASN.1 specific parameterization which involves objects which cannot be defined directly in the TTCN-3 core language shall therefore be resolved in the ASN.1 part before use within the TTCN-3. The ASN.1 specific objects are:

a) information object classes;

b) information objects;

c) information object sets.

For example, the following is not legal because it would associate a TTCN-3 type which should take an ASN.1 object set as an actual parameter.

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

   -- Information object class definition
   MESSAGE ::= CLASS { &msgTypeValue  INTEGER UNIQUE,
                                      &MsgFields}

   -- Information object definition
   setupMessage MESSAGE ::= {   &msgTypeValue      1,
                                &MsgFields         OCTET STRING}

   setupAckMessage MESSAGE ::= {   &msgTypeValue   2,
                                   &MsgFields      BOOLEAN}

   -- Information object set definition
   MyProtocol MESSAGE ::= {setupMessage | setupAckMessage}

   -- ASN.1 type constrained by object set
   MyMessage{ MESSAGE : MsgSet} ::= SEQUENCE
   {
      code   MESSAGE.&msgTypeValue({MsgSet}),
      Type MESSAGE.&MsgFields({MsgSet})
   }
END

module MyTTCNModule
{
   // TTCN-3 module definition
   import from MyASN1module language "ASN.1:2002" all;

   // Illegal TTCN-3 type with object set as parameter
   type record Q(MESSAGE MyMsgSet) ::= {  Z                      field1,
                                          MyMessage(MyMsgSet)    field2}
}
```

To make this a legal definition, the extra ASN.1 type MyMessage1 has to be defined as shown below. This resolves the information object set parameterization and can therefore be directly used in the TTCN-3 module.

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

   …

   MyProtocol MESSAGE ::= {setupMessage | setupAckMessage}

   -- Extra ASN.1 type to remove object set parameterization
   MyMessage1 ::= MyMessage{MyProtocol}
END

module MyTTCNModule
{
   // TTCN-3 module definition
   import from MyASN1module language "ASN.1:2002" all;

   // Legal TTCN-3 type with no object set as parameter
   type record Q := { Z                field1,
                      MyMessage1       field2}
}
```

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 [ITU-T Z.161]) 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 X.690] (BER);

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

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

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

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

f) "PER-CANONICAL-UNALIGNED:2002" means encoded according to (canonical unaligned PER) [ITU-T X.691];

"PER-CANONICAL-ALIGNED:2002" means encoded according to [ITU-T X.691] (canonical aligned PER);

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

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

j) "EXTENDED-XER:2002" means encoded according to Amd. 1 to [ITU-T X.693] (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 a TTCN-3 object without an attribute, is out of scope of this Recommendation (see Note 1):

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 X.690]) in case of BER, CER and DER encodings or the single octet length determinant (see clause 10.9 of [ITU-T X.691]) in case of any form of the PER encoding (see Note 2).

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 X.690) in case of BER, CER and DER encodings or the two octets length determinant (see clause 10.9 of [ITU-T X.691]) in case of any form of the PER encoding (see Note 2).

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 X.690]) 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 as specified by the attribute encoding form or matched if received with the specified choice only (see clause 8.18 of [ITU-T X.690]). 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 X.690] and the specified attribute do not meet, 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 X.690] and clause 26 of [ITU-T X.691]).

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 X.690] and clause 26 of [ITU-T X.691]).

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 X.690] and clause 26 of [ITU-T X.691]).

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 X.690] and clause 26 of [ITU-T X.691]).

NOTE 1 – These attributes may be reused in implementations-specific encoding rules with a different meaning than that specified in this clause, may be ignored or a warning/error indication may be given. However, the strategy to be applied is implementation dependent.

NOTE 2 – 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

Additional BNF and static semantics

(This annex forms an integral part of this Recommendation)

When ASN.1 is supported, rules defined in Annex A of [ITU-T Z.161] shall apply, supplemented by the Backus-Naur Form (BNF) and semantic rules specified in this annex.

In addition to those listed in Table 3 of [ITU-T Z.161] "List of TTCN-3 terminals which are reserved words" (see clause A.1.5 of [ITU-T Z.161]), the word **`objid`** shall also be a TTCN-3 reserved word (keyword).

Amendments to clause A.1.6 of [ITU-T Z.161] 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 [ITU-T Z.161], when the use of ASN.1 is supported, the keywords listed in clause 11.27 of [ITU-T X.680], after applying the name conversion rules defined in clause 8.2 of this Recommendation, shall not be used as identifiers in a TTCN-3 module. ASN.1 keywords shall follow the requirements of [ITU-T X.680].

In addition to those listed in Table A.3 of [ITU-T Z.161], when the use of ASN.1 is supported, "**`objid`**" shall be a reserved word.

Additions to clause A.1.6 of [ITU-T Z.161] are identified by underlined font, deletions are identified by strikethrough font.

```
5. GlobalModuleId ::= ModuleIdentifier [Dot DefinitiveIdentifier]
52. 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

Additional predefined TTCN-3 functions

(This annex forms an integral part of this Recommendation)

Void.

Appendix C

Predefined object identifier components

(This appendix does not form an integral part of this Recommendation)

[ITU-T X.660] defines the tree of object identifier components shown below. Only the object identifier components defined in [ITU-T X.660] 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 them 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)*

Appendix D

Bibliography

(This appendix does not form an integral part of this Recommendation)

[b-ITU-T T.50] Recommendation ITU-T T.50 (1992), *International Reference Alphabet (IRA) (Formerly International Alphabet No. 5 or IA5) – Information technology – 7-bit coded character set for information interchange*.

[b-ITU-T T.61] Recommendation ITU-T T.61 (1988), *Character repertoire and coded character sets for the international teletex service*.

[b-ITU-T X.208] Recommendation ITU-T X.208 (1988), *Specification of Abstract Syntax Notation One (ASN.1)*.

[b-ITU-T X.680 v1] Recommendation ITU-T X.680 (1994) | ISO/IEC 8824-1:1995, *Information technology – Abstract Syntax Notation One (ASN.1): Specification of basic notation*.

[b-ITU-T X.680 v2] Recommendation ITU-T X.680 (1997) | ISO/IEC 8824-1:1998, *Information technology – Abstract Syntax Notation One (ASN.1): Specification of basic notation*.

[b-ITU-T X.681 v1] Recommendation ITU-T X.681 (1994) | ISO/IEC 8824-2:1995, *Information technology – Abstract Syntax Notation One (ASN.1): Information object specification*.

[b-ITU-T X.681 v2] Recommendation ITU-T X.681 (1997) | ISO/IEC 8824-2:1998, *Information technology – Abstract Syntax Notation One (ASN.1): Information object specification*.

[b-ITU-T X.682 v1] Recommendation ITU-T X.682 (1994) | ISO/IEC 8824-3:1995, *Information technology – Abstract Syntax Notation One (ASN.1): Constraint specification*.

[b-ITU-T X.682 v2] Recommendation ITU-T X.682 (1997) | ISO/IEC 8824-3:1998, *Information technology – Abstract Syntax Notation One (ASN.1): Constraint specification*.

[b-ITU-T X.683 v1] Recommendation ITU-T X.683 (1994) | ISO/IEC 8824-4:1995, *Information technology – Abstract Syntax Notation One (ASN.1): Parameterization of ASN.1 specifications*.

[b-ITU-T X.683 v2] Recommendation ITU-T X.683 (1997) | ISO/IEC 8824-4:1998, *Information technology – Abstract Syntax Notation One (ASN.1): Parameterization of ASN.1 specifications*.

[b-ISO/IEC 6429] ISO/IEC 6429:1992, *Information technology – Control functions for coded character sets*.

[b-ISO/IEC 8859-1] ISO/IEC 8859-1:1998, *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 this Recommendation.

Appendix E

Deprecated features

(This appendix does not form an integral part of this Recommendation)

E.1 Using `sizeof` for `objid` values and templates

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

E.2 The decompose function

The previous version of this Recommendation 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 this Recommendation and may be fully removed in the next published edition.

Printed in Switzerland
Geneva, 2009