Final draft ETSI ES 201 873-7 V4.5.1 (2013-02)

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-201873-7 T3ed451ASN1

Keywords
ASN.1, language, testing, TTCN, XML

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://ipr.etsi.org).

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 final draft ETSI Standard (ES) has been produced by ETSI Technical Committee Methods for Testing and Specification (MTS), and is now submitted for the ETSI standards Membership Approval Procedure.

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 Recommendations ITU-T X.680 [2], X.681 [3], X.682 [4] and 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 specific references,only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

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 necessary for the application of the present document.

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

[2] Recommendation ITU-T X.680 (2008): "Information technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation".

[3] Recommendation ITU-T X.681 (2008): "Information technology - Abstract Syntax Notation One (ASN.1): Information object specification".

[4] Recommendation ITU-T X.682 (2008): "Information technology - Abstract Syntax Notation One (ASN.1): Constraint specification".

[5] Recommendation ITU-T X.683 (2008): "Information technology - Abstract Syntax Notation One (ASN.1): Parameterization of ASN.1 specifications".

[6] Recommendation ITU-T X.690 (2008): "Information technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)".

[7] Recommendation ITU-T X.691 (2008): "Information technology - ASN.1 encoding rules: Specification of Packed Encoding Rules (PER)".

[8] Recommendation ITU-T X.693 (2008): "Information technology - ASN.1 encoding rules: XML Encoding Rules (XER)".

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

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

[11] Recommendation ITU-T X.660 (2011): "Information technology - Procedures for the operation of object identifier registration authorities: General procedures and top arcs of the international object identifier tree".

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

[13] Void.

2.2 Informative references

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

[i.1] ISO 8601 (2004): "Data elements and interchange formats - Information interchange - Representation of dates and times".

[i.2] ETSI ES 201 873-10: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 10: TTCN-3 Documentation Comment Specification".

[i.3] ISO 3166-1: "Codes for the representation of names of countries and their subdivisions - Part 1: Country codes".

[i.4] Recommendation ITU-T X.121: "Public data networks - Network aspects - International numbering plan for public data networks".

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

[i.5] Recommendation ITU-T X.208: "Specification of Abstract Syntax Notation One (ASN.1)" (Blue Book).

[i.6] Recommendation ITU-T X.680 (1994): "Information technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation".

[i.7] Recommendation ITU-T X.680 (1997): "Information technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation".

[i.8] Recommendation ITU-T X.680 (2002): "Information technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation".

[i.9] Recommendation ITU-T X.681 (1994): "Information technology - Abstract Syntax Notation One (ASN.1): Information object specification".

[i.10] Recommendation ITU-T X.681 (1997): "Information technology - Abstract Syntax Notation One (ASN.1): Information object specification".

[i.11] Recommendation ITU-T X.681 (2002): "Information technology - Abstract Syntax Notation One (ASN.1): Information object specification".

[i.12] Recommendation ITU-T X.682 (1994): "Information technology - Abstract Syntax Notation One (ASN.1): Constraint specification".

[i.13] Recommendation ITU-T X.682 (1997): "Information technology - Abstract Syntax Notation One (ASN.1): Constraint specification".

[i.14] Recommendation ITU-T X.682 (2002): "Information technology - Abstract Syntax Notation One (ASN.1): Constraint specification".

[i.15] Recommendation ITU-T X.683 (1994): "Information technology - Abstract Syntax Notation One (ASN.1): Parameterization of ASN.1 specifications".

[i.16] Recommendation ITU-T X.683 (1997): "Information technology - Abstract Syntax Notation One (ASN.1): Parameterization of ASN.1 specifications".

[i.17] Recommendation ITU-T X.683 (2002): "Information technology - Abstract Syntax Notation One (ASN.1): Parameterization of ASN.1 specifications".

[i.18] Recommendation ITU-T X.690 (2002): "Information technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)".

[i.19] Recommendation ITU-T X.691 (2002): "Information technology - ASN.1 encoding rules: Specification of Packed Encoding Rules (PER)".

[i.20] ETSI ES 202 781: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; TTCN-3 Language Extensions: Configuration and Deployment Support".

[i.21] ETSI ES 202 782: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; TTCN-3 Language Extensions: TTCN-3 Performance and Real Time Testing".

[i.22] ETSI ES 202 784: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; TTCN-3 Language Extensions: Advanced Parameterization".

[i.23] ETSI ES 202 785: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; TTCN-3 Language Extensions: Behaviour Types".

[i.24] ETSI ES 202 786: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; TTCN-3 Language Extensions: Support of interfaces with continuous signals".

[i.25] ETSI ES 202 789: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; TTCN-3 Language Extensions: Extended TRI".

[i.26] CCITT Blue Book.

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], Recommendation ITU-T 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": 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
BER	Basic Encoding Rules of ASN.1
CER	Canonical Encoding Rules of ASN.1
OID	Object IDentifier
PER	Packed Encoding Rules of ASN.1
XER	XML Encoding Rules of ASN.1

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 the present document, TTCN-3 becomes fully harmonized with ASN.1 which may be used with TTCN-3 modules as an alternative data type and value syntax. The present document 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.

Please note that ES 201 873-1 [1] specifies the core capabilities of the TTCN-3 language. Other documents, as ES 202 781 [i.20], ES 202 782 [i.21], ES 202 784 [i.22], ES 202 785 [i.23], ES 202 786 [i.24], ES 202 789 [i.25] specify extensions to the core language that may define additional ASN.1 to TTCN-3 mapping rules.

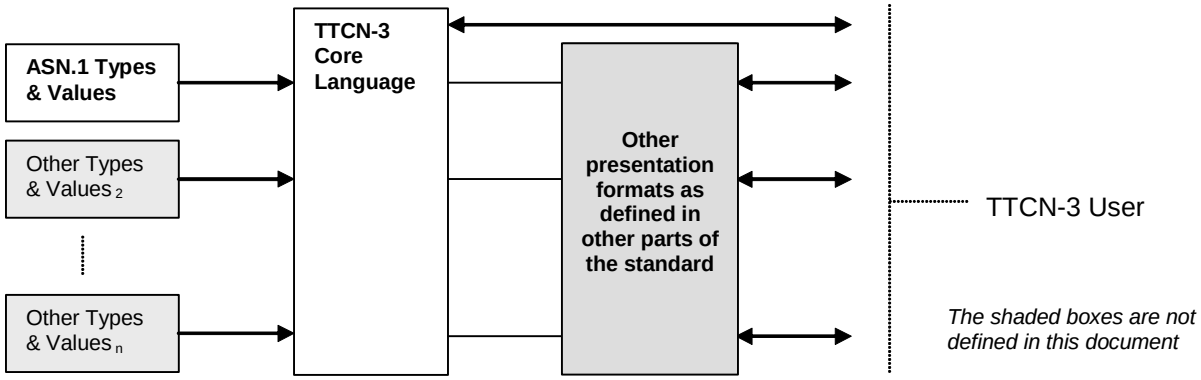

Figure 1: User's view of the core language and the various presentation formats

4.1 Conformance and compatibility

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

The language mapping presented in the present document is compatible to:

- ES 201 873-1 [1], V4.2.1.
- ES 201 873-10 [i.2], V4.2.1.

NOTE: Only the informative annex E uses features from ES 201 873-10 [i.2].

If later versions of those parts are available and should be used instead, the compatibility of the language mapping presented in the present document has to be checked individually.

5 General

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

In general, there are two approaches to the integration of other languages with TTCN-3, which will be referred to as implicit and explicit mapping. The implicit mapping makes use of the import mechanism of TTCN-3, denoted by the keywords *language* and *import*, in which case the TTCN-3 tool shall produce an internal representation of the imported objects, which representation shall retain all the structural and encoding information. This internal representation is not accessible by the user. It facilitates the immediate use of the abstract data specified in the other language. Therefore, the definition of a specific data interface for each of these languages is required.

The explicit mapping translates the definitions of the other language directly into appropriate TTCN-3 language artefacts. This also means that all information needed for correct encoding and decoding shall be present in the TTCN-3 module(s) generated by this translation.

In case of the ASN.1 to TTCN-3 mapping no TTCN-3 encoding instructions are defined by the present document, hence only the implicit mapping is specified.

6 Amendments to the core language

Using ASN.1 with TTCN-3 is handled at the static type-value level. Though it mainly means additions described in the subsequent clauses, some of the core language syntactical structures shall also be amended to support the use of ASN.1. These are specified in clause A.2.

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 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 Recommendation ITU-T 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 Recommendation ITU-T X.660 [11] (e.g. with a node beneath the root not defined in Recommendation ITU-T X.660 [11]).

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

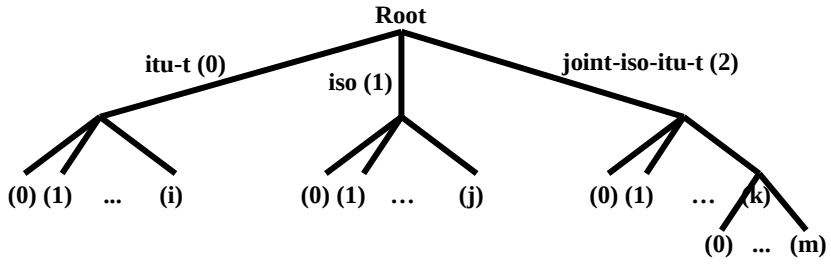

Figure 2: The object identifier tree

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, table 3 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, table 3 of ES 201 873-1 [1], range 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 Recommendation ITU-T 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 (see note)	Yes	Yes	Yes	Yes (see note)	Yes				Yes	Yes		Yes	Yes (see note)
NOTE: Can be assigned to templates, however when used, shall be applied to optional fields of record and set types 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 operator

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: SpecificValue, value list, complemented list, *AnyValue*, *AnyValueOrNone*, *AnyElement* and *AnyElementsOrNone* and the length 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

This clause describes the changes in and additions to ES 201 873-6 [12] that shall be undertaken to support the objid type and values in TCI.

7.2.7.1 Adding objid to abstract data types and values

In clause "7.2.2.1 Abstract TTCN-3 data types" [12], at the operation `TciTypeClassType getTypeClass()`, the list of values of `TciTypeClassType` shall be extended with the constant `OBJID`.

The hierarchy of abstract TTCN-3 values, presented on figure 3 in clause "7.2.2.2 Abstract TTCN-3 values" [12], shall be extended with the `ObjidValue` as shown on figure 3. Please note that the abstract data type `Value` is present on figure 3 to illustrate the addition of `ObjidValue` only, but it is defined in ES 201 873-6 [12].

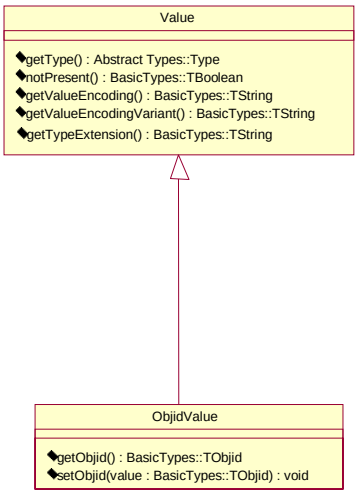

Figure 3: Adding objid to the abstract value hierarchy

Clause "7.2.2.2 Abstract TTCN 3 values" [12] shall be extended by the `ObjidValue` abstract type as the last clause of clause 7.2.2.2 [12] as follows:

7.2.2.2.xThe abstract data type ObjidValue

The abstract data type `ObjidValue` is based on the abstract data type `Value`. It represents TTCN-3 `objid` values.

The following operations are defined on the abstract data type `ObjidValue`:

`TObjid getObjid()` Returns the object id value of the TTCN-3 `objid`

`void setObjid(in TObjid value)` Sets this `ObjidValue` to `value`

In clause "7.3.2.1.1 getTypeForName" [12], in the "In Parameters" row, the list of reserved type names that shall return a predefined type, shall be extended by "objid".

Clause "7.3.2.1 TCI-CD required" [12] shall be extended with the following operation:

7.3.2.1.5 getObjid

Signature	`Type getObjid()`
Return Value	An instance of `Type` representing a TTCN-3 object id type.
Effect	Constructs and returns a basic TTCN-3 object id type.

7.2.7.2 Adding objid to Java language mapping

Add the following sentence to clause "8.2.1 [12] Basic type mapping": "The native type TObjId is defined in the respective section of the ObjidValue interface.".

Add to clause "8.2.2.4 TciTypeClassType" [12], within `public interface TciTypeClass`:

```
" public final static int OBJID          = 11 ;"
```

In clause "8.2.3.1 [12] Type" of, for the `getTypeClass()` method, add `OBJID` to the list of allowed constants that `TciTypeClassType` can take.

Extend clause "8.2.4 Abstract value mapping" [12] with the following clauses:

8.2.4.5 ObjidValue

`ObjidValue` is mapped to the following interface:

```
// TCI IDL ObjidValue
package org.etsi.ttcn.tci;
public interface ObjidValue {
  TciObjId    getObjid ();
  void        setObjid (TciObjId value);
}
```

Methods:

- `getObjid()` Returns the object id value of the TTCN-3 `objid`.
- `setObjid(`TciObjId `value)` Sets this `ObjidValue` to `value`.

8.2.4.6 TciObjId

`TciObjId` is mapped to the following interface. The native java representation of a TTCN-3 ObjectId consists of an ordered sequence of `TciObjIdElements`.

```
package org.etsi.ttcn.tci;
public interface TciObjId {
  public int              size() ;
  public void             setObjElement(TciObjIdElement[] objElemens) ;
  public TciObjIdElement  getObjElement(int index) ;
}
```

Methods:

- `size()` Returns the size of this Object Id in TciObjIdElements.
- `setObjElement(TciObjIdElement[] objElements)` Sets this ObjId to the list of objElements.
- `getObjElement(int index)` Return the TciObjIdElement at position `index`.

8.2.4.7 TciObjIdElement

A TciObjIdElement represent a single object element within a TTCN-3 ObjId value. It can be set using different representations like the ASCII representation or as integer.

`TciObjIdElement` is mapped to the following interface:

```
package org.etsi.ttcn.tci;
public interface TciObjIdElement {
  public void     setElementAsAscii(String element) ;
  public void     setElementAsNumber(int element) ;
  public String   getElementAsAscii() ;
  public int      getElementAsNumber() ;
}
```

Methods:

- `setElementAsAscii(String element)` Sets the internal representation of this ObjIdElement to string value `element`.
- `setElementAsNumber(int element)` Set this the internal representation of this ObjIdElement to the integer value `element`.
- `getElementAsAscii()` Returns the internal representation of this ObjIdElement as string.
- `getElementAsNumber()` Returns the internal representation of this ObjIdElement as integer.

In clause "8.3 Constants" [12], the list of constants that shall be used for value handling, shall be extended with:

- `org.etsi.ttcn.tci.TciTypeClass.OBJID;`

In clause "8.4.2.2 TCI-CD required" [12] of, within `public interface TciCDRequired`, the following shall be added:

```
public Type     getObjid ();
```

7.2.7.3 Adding objid to ANSI C language mapping

Clause "9.2 Value interface" [12] shall be extended with the following:

ObjidValue		
TObjid getObjid()	`TciObjidValue tciGetTciObjidValue(Value inst)`	
void setObjid(in TObjid value)	`void tciSetObjidValue(Value inst, TciObjidValue value)`	

The TCI-CD Required interface in clause "9.4.2.2 TCI-CD required" [12] shall be extended by:

```
Type tciGetTciObjidType()
```

In clause "9.5 Data" [12], the type definition of `TciTypeClassType` shall be extended with the value: "`TCI_OBJID_TYPE`".

Clause "9.6 Miscellaneous" [12] shall be extended with the following:

Objid representation		

Objid	`typedef struct TciObjidValue`
	`{`
	`  long int  length;`
	`  TciObjidElem  *elements;`
	`} TciObjidValue;`
		Since the Objid value is returned "as is" via the Objid value interface, a representation shall be defined.

TciObjidElem	`typedef struct TciObjidElemValue`
	`{`
	`  char*  elem_as_ascii;`
	`  long int elem_as_number;`
	`  void*  aux;`
	`} TciObjidElemValue;`	

In clause "10.3.3.1 Value" [12], the <xsd:choice> child element of the complex type definition "Value" shall be extended by the local element "objid" as follows:

```
<xsd:element name="objid" type="Values:ObjidValue"/>
```

In the same clause the following shall be added to the list of "Choice of Elements":

- objid An objid value.

Add the following new clause to clause "10.3.3 Abstract value mapping" [12]:

10.3.3.5 ObjidValue

ObjidValue is mapped to the following complex type:

```
<xsd:complexType name="ObjidValue">
    <xsd:simpleContent>
        <xsd:extension base="SimpleTypes:TString">
                <xsd:attributeGroup ref="Values:ValueAtts"/>
        </xsd:extension>
    </xsd:simpleContent>
</xsd:complexType>
```

Simple Content:

- value The objid value as string.
- null If no value is given.
- omit If the value is omitted.

Attributes:

- The same attributes as those of Value.

In clause "10.3.3.12 RecordValue" [12], the <xsd:choice> child element of the complex type definition "RecordValue" shall be extended by the local element "objid" as follows:

```
<xsd:element name="objid" type="Values:ObjidValue"/>
```

In the same clause the following shall be added to the list of "Sequence of Elements:":

- objid An objid value.

In clause "10.3.3.13 RecordOfValue" [12], the <xsd:choice> child element of the complex type definition "RecordOfValue" shall be extended by the local element "objid" as follows:

```
<xsd:element name="objid" type="Values:ObjidValue" minOccurs="0"
        maxOccurs="unbounded"/>
```

In the same clause the following shall be added to the list of "Choice of Sequence of Elements":

- objid An objid value.

In clause "10.3.3.14 SetValue" [12], the <xsd:choice> child element of the complex type definition "SetValue" shall be extended by the local element "objid" as follows:

```
<xsd:element name="objid" type="Values:ObjidValue"/>
```

In the same clause the following shall be added to the list of "Sequence of Elements":

- objid An objid value.

In clause "10.3.3.15 SetOfValue" [12], the <xsd:choice> child element of the complex type definition "SetOfValue" shall be extended by the local element "objid" as follows:

```
<xsd:element name="objid" type="Values:ObjidValue" minOccurs="0"
    maxOccurs="unbounded"/>
```

In the same clause the following shall be added to the list of "Choice of Sequence of Elements":

- objid An objid value.

In clause "10.3.3.17 UnionValue" [12], the <xsd:choice> child element of the complex type definition "UnionValue" shall be extended by the local element "objid" as follows:

```
<xsd:element name="objid" type="Values:ObjidValue"/>
```

In the same clause the following shall be added to the list of "Choice of Elements":

- objid An objid value.

In clause "10.3.3.18 AnytypeValue" [12], the <xsd:choice> child element of the complex type definition "AnytypeValue" shall be extended by the local element "objid" as follows:

```
<xsd:element name="objid" type="Values:ObjidValue"/>
```

In the same clause the following shall be added to the list of "Choice of Elements":

- objid An objid value.

In clause "10.3.3.19 AddressValue" [12], the <xsd:choice> child element of the complex type definition "AddressValue" shall be extended by the local element "objid" as follows:

```
<xsd:element name="objid" type="Values:ObjidValue"/>
```

In the same clause the following shall be added to the list of "Choice of Elements":

- objid An objid value.

In clause "10.3.4.1 TciValueTemplate" [12], the <xsd:choice minOccurs="0"> element shall be extended by the local element "objid" as follows:

```
<xsd:element name="objid" type="Templates:ObjidTemplate"/>
```

In the same clause the following shall be added to the list of "Choice of Elements":

- objid An objid value.

In Annex A "IDL Specification of TCI" [12], within module tciInterface, below "General Abstract Data Types",

- add the basic definition "native TObjid;", and
- extend the type enum TciTypeClassType with the value "OBJID_CLASS,"

In annex A "IDL Specification of TCI" [12], within module tciInterface, below "Abstract TTCN-3 Data Types And Values", add the following declaration:

```
interface ObjidValue : Value {
  TObjid getObjid ();
  void   setObjid (in TObjid value);
};
```

In annex A "IDL Specification of TCI" [12], within `module tciInterface`, below "`Coding Decoding Interface - Required`", within `interface TCI_CD_Required`, add: "`Type getObjid ();`".

In clause "B.3 TCI-TL XML Schema for Values" [12], the `<xsd:choice>` child element of the complex type definition "`Value`" shall be extended by the local element "objid" as follows:

```
<xsd:element name="objid" type="Values:ObjidValue"/>
```

In the same clause add the new complex type definition:

```
<xsd:complexType name="ObjidValue">
    <xsd:choice>
        <xsd:element name="value" type="SimpleTypes:TString"/>
        <xsd:element name="null" type="Templates:null"/>
        <xsd:element name="omit" type="Templates:omit"/>
    </xsd:choice>
    <xsd:attributeGroup ref="Values:ValueAtts"/>
</xsd:complexType>
```

In the same clause, the `<xsd:choice minOccurs="0" maxOccurs="unbounded">` element of the complex type definition "Record`Value`" shall be extended by the local element "objid" as follows:

```
<xsd:element name="objid" type="Values:ObjidValue"/>
```

In the same clause, the `<xsd:choice>` element of the complex type definition "RecordOf`Value`" shall be extended by the local element "objid" as follows:

```
<xsd:element name="objid" type="Values:ObjidValue" minOccurs="0"
    maxOccurs="unbounded"/>
```

In the same clause, the `<xsd:choice minOccurs="0" maxOccurs="unbounded">` element of the complex type definition "Set`Value`" shall be extended by the local element "objid" as follows:

```
<xsd:element name="objid" type="Values:ObjidValue"/>
```

In the same clause, the `<xsd:choice>` element of the complex type definition "SetOf`Value`" shall be extended by the local element "objid" as follows:

```
<xsd:element name="objid" type="Values:ObjidValue" minOccurs="0"
    maxOccurs="unbounded"/>
```

In the same clause, the `<xsd:choice>` element of the complex type definition "Union`Value`" shall be extended by the local element "objid" as follows:

```
<xsd:element name="objid" type="Values:ObjidValue"/>
```

In the same clause, the `<xsd:choice minOccurs="0" maxOccurs="unbounded">` element of the complex type definition "Anytype`Value`" shall be extended by the local element "objid" as follows:

```
<xsd:element name="objid" type="Values:ObjidValue"/>
```

In the same clause, the `<xsd:choice minOccurs="0" maxOccurs="unbounded">` element of the complex type definition "Address`Value`" shall be extended by the local element "objid" as follows:

```
<xsd:element name="objid" type="Values:ObjidValue"/>
```

In clause "B.4 TCI-TL XML Schema for Templates" [12], the `<xsd:choice>` child element of the complex type definition "`TciValueTemplate`" shall be extended by the local element "objid" as follows:

```
<xsd:element name="objid" type="Templates:ObjidTemplate"/>
```

In the same clause add the new complex type definition:

```
<xsd:complexType name="ObjidTemplate">
    <xsd:choice>
        <xsd:element name="value" type="SimpleTypes:TString"/>
        <xsd:element name="templateDef" type="SimpleTypes:TString"/>
        <xsd:element name="omit" type="Templates:omit"/>
        <xsd:element name="any" type="Templates:any"/>
        <xsd:element name="anyoromit" type="Templates:anyoromit"/>
        <xsd:element name="null" type="Templates:null"/>
    </xsd:choice>
    <xsd:attributeGroup ref="Values:ValueAtts"/>
</xsd:complexType>
```

In the same clause, the `<xsd:choice minOccurs="0">` element of the complex type definition "`RecordTemplate`" shall be extended by the local element "objid" as follows:

```
<xsd:element name="objid" type="Templates:ObjidTemplate"/>
```

In the same clause, the <xsd:choice minOccurs="0" maxOccurs="unbounded"> element of the complex type definition "`RecordOfTemplate`" shall be extended by the local element "objid" as follows:

```
<xsd:element name="objid" type="Templates:ObjidTemplate" minOccurs="0"
        maxOccurs="unbounded"/>
```

In the same clause, the `<xsd:choice minOccurs="0">` element of the complex type definition "`SetTemplate`" shall be extended by the local element "objid" as follows:

```
<xsd:element name="objid" type="Templates:ObjidTemplate"/>
```

In the same clause, the `<xsd:choice minOccurs="0" maxOccurs="unbounded">` element of the complex type definition "`SetOfTemplate`" shall be extended by the local element "objid" as follows:

```
<xsd:element name="objid" type="Templates:ObjidTemplate" minOccurs="0"
        maxOccurs="unbounded"/>
```

In the same clause, the `<xsd:choice minOccurs="0">` element of the complex type definition "`UnionTemplate`" shall be extended by the local element "objid" as follows:

```
<xsd:element name="objid" type="Templates:ObjidTemplate"/>
```

In the same clause, the `<xsd:choice minOccurs="0">` element of the complex type definition "`AnytypeTemplate`" shall be extended by the local element "objid" as follows:

```
<xsd:element name="objid" type="Templates:ObjidTemplate"/>
```

In the same clause, the `<xsd:choice minOccurs="0">` element of the complex type definition "`AddressTemplate`" shall be extended by the local element "objid" as follows:

```
<xsd:element name="objid" type="Templates:ObjidTemplate"/>
```

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 the ASN.1 to TTCN-3 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 of the base in ASN.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.

8.1.a Importing from ASN.1 modules

8.1.a.1 Language specification strings

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

- "ASN.1:2008" for ASN.1 version 2008;
- "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 [i.26]).

NOTE 1: Language identifiers "ASN.1:1988", "ASN.1:1994", "ASN.1:1997" and "ASN.1:2002" refer to superseded versions of ASN.1. 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 Recommendation ITU-T X.680 [2] is not considered.

NOTE 2: References to the 1994, 1997, 2002 and the Blue Book (1988) versions of ASN.1 can be found in clause 2, [i.5] to [i.19].

8.1.a.2 Importing definitions from ASN.1 modules

ASN.1 IMPORTS are transitive by default (see clause 12.13 of Recommendation ITU-T X.680 [2]), however this is not the case when importing from ASN.1 to TTCN-3. In this latter case only visible definitions shall be imported (see importability of ASN.1 definitions in clause 8.1.a.4).

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]).

When the TTCN-3 import statement is importing single definitions or definitions of the same kind from ASN.1 (see clauses 8.2.3.2 and 8.2.3.4 of ES 201 873-1 [1]), or an import all statement contains an exception list (see clause 8.2.3.5 of ES 201 873-1 [1]), the **`type`** syntactical branch shall be used for ASN.1 type and value list definitions and the **`const`** syntactical branch shall be used for ASN.1 values. Using of other syntactical branches (e.g. **`group`**, **`template`**, **`testcase`** etc.) are not allowed in import statements importing from ASN.1.

NOTE: 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.

8.1.a.3 Importing import statements from ASN.1 modules

It is not allowed to import ASN.1 IMPORTS statements to TTCN-3 (like importing TTCN-3 import statements to other TTCN-3 modules, see clause 8.2.3.7 "Importing of import statements" from TTCN-3 modules of ES 201 873-1 [1]).

8.1.a.4 Import Visibility of ASN.1 definitions

All ASN.1 definitions declared directly in an ASN.1 module can be imported into TTCN-3 if the ASN.1 module does not have an EXPORTS statement (the empty alternative of the Exports production is used, see clause 12.1 of Recommendation ITU-T X.680 [2]) or exports all definition (the `EXPORTS ALL` alternative of the Exports production is used).

If the ASN.1 module has an export list (the `EXPORTS` SymbolsExported alternative of the Exports production is used), only the definitions on the export list AND defined in the given ASN.1 module shall be imported into TTCN-3, all other definitions shall not be imported.

ASN.1 definitions imported from an ASN.1 module to other ASN.1 module(s) are not importable when importing from ASN.1 to TTCN-3 (i.e. importing from ASN.1 is not transitive when importing into TTCN-3).

8.2 Identifiers

In converting ASN.1 identifiers to TTCN-3 identifiers, hyphen "-" characters shall be replaced by 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.

NOTE: ES 201 873-1 [1] clause A.1.5 table A.2 defines the keywords of the core language. However, TTCN-3 language extensions (see [i.20] to [i.24], but other extensions may also be published after the publication of the present document) may define additional keywords and rules for handling those keywords in TTCN-3 modules requiring the given extension.

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 for ASN.1 types and values

ASN.1 value sets are handled in the present document the same way as ASN.1 types. Therefore, when referring to "ASN.1 types" in the present document, both ASN.1 value set definitions and type definitions are meant. 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 11.2 and 11.4 of Recommendation ITU-T 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 Recommendations ITU-T X.680 [2], X.681 [3], X.682 [4] and 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 may consist of tags and encoding prefixes.

0bis) Ignore names of inner types and values in `SEQUENCE OF` and `SET OF` definitions (see note 1).

1) Ignore any extension markers and exception specifications.

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

3) Ignore any contents constraint (see clause 11 of Recommendation ITU-T X.682 [4]).

4) Convert pattern constraints (see clause 51.9 of Recommendation ITU-T X.680 [2]) to TTCN-3 pattern subtypes (see clause 6.1.2.5 of ES 201 873-1 [1]).

5) Execute the `COMPONENTS OF` transformation according to clause 25.5 of Recommendation ITU-T X.680 [2] on any `SEQUENCE` types and according to clause 27.2 on any `SET` types containing the keywords "`COMPONENTS OF`".

6) 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.

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 36.5 of Recommendation ITU-T 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 37.5 of Recommendation ITU-T 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 44.5 of Recommendation ITU-T 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 Recommendation ITU-T 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 bits in ASN.1 types. For each named number a TTCN-3 constant definition shall be generated: the type of the constant is the TTCN-3 type corresponding to the ASN.1 type being translated and the name of the constant is constructed as: the name of the TTCN-3 type corresponding to the given ASN.1 type, followed by a "_" (LOW LINE) character, followed by the name of the number converted according to clause 8.2 of the present document, followed by a "_" (LOW LINE) character. 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.

EXAMPLE 1:

```
-- The definition in ASN.1:
Color ::= INTEGER {red(0),green(1), blue(255) }

// is mapped to the TTCN-3 type and constants:
type integer Color;
const Color Color_red_ := 0;
const Color Color_green_ := 1;
const Color Color_blue_ := 255;
```

13) Replace any selection type with the type referenced by the selection type; if the denoted choice type (the "Type" in clause 30.1 of Recommendation ITU-T 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 41.2 of Recommendation ITU-T X.680 [2];
- `PrintableString`: **`charstring`** constrained to the set of characters as given in clause 41.4 of Recommendation ITU-T X.680 [2];
- `TeletexString` and `T61String`: **`universal charstring`** constrained to the set of characters as given in clause 41.1, table 8, row **`TeletexString`** (**`T61String`**) of Recommendation ITU-T X.680 [2], in clause 9.1 of Part-7.
- `VideotexString`: **`universal charstring`** constrained to the set of characters as given in Recommendations ITU-T T.100 [9] and T.101 [10];
- `GraphicString`: **`universal charstring`**;
- `GeneralString`: **`universal charstring`**.

16) Replace any of the following time types with their associated types obtained as (see notes 14 and 15):

- `GeneralizedTime` types or values with the type or value of **`charstring`**;
- `UTCTime` types or values with the type or value of **`charstring`**;
- `TIME`, `DATE`, `TIME-OF-DAY`, `DATE-TIME` and `DURATION` types or values with the type or value of **`charstring`**. Properties settings, inner and range subtyping, if any, shall be ignored (see also table 4).

17) Replace any of the following types with their associated types obtained as:

- `ObjectDescriptor` type or value by the **`universal charstring`** type or value;
- `OID-IRI` type or value by the **`universal charstring`** type or value (see note 14);
- `RELATIVE-OID-IRI` type or value by the **`universal charstring`** type or value (see note 14).

18) Replace any notations for the object class field types (see clause 14 of Recommendation ITU-T 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 Recommendation ITU-T X.681 [3]) by the ASN.1 definition they are referencing to.

20) Revert table constraints (see clause 10 of Recommendation ITU-T 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 clause8.2, 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, converted according to clause 8.2, if the ASN.1 **`NULL`** type 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 and values contain abstract information only, thus do not contain all information needed for correct encoding. The way of handling the information needed by the test system to provide correct encoding and/or decoding (both embedded in ASN.1 definitions and provided in the encoding reference default, tag default and extension default settings of ASN.1 modules and encoding control sections, if any and information coming from the ASN.1 specification itself, like tag values of built-in ASN.1 types) 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 Recommendation ITU-T 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 of the present document).

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 Recommendations ITU-T 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) that only values of the same enumerated types are allowed to be compared, does not apply to imported ASN.1 NULL types.

NOTE 14: The ASN.1 `time` types (including its useful types) are transformed to a restricted TTCN-3 **`charstring`** type and the `OID-IRI` and `RELATIVE-OID-IRI` types are transformed to the TTCN-3 **`universal charstring`** type primarily for tool efficiency reasons (though this approach also allows sending some invalid values without the need to create a specific type). This, however, means that the user should exercise specific caution in receiving templates, as AnyValue and AnyValuesOrNone will accept incorrectly formatted time values as well. When the correctness of the received values is important, the **`pattern`** matching (possibly appended with the **`ifpresent`** matching attribute for optional fields) should be used instead of AnyValue and AnyValuesOrNone. TTCN-3 patterns for the time types are given in annex E and for the OID-IRI type is given in clause C.2.

NOTE 15: Though all ASN.1 time types are transformed to a restricted TTCN-3 **`charstring`** type, they differ in their tag values (see clauses and 38.1.1, 38.4, 46.3 and 47.3 of Recommendation ITU-T X.680 [2]) and encodings (see clauses 8.25 and 8.26 of Recommendation ITU-T X.690 [6] and clauses 10.6.5 and 32 of Recommendation ITU-T X.691 [7]). Therefore it is necessary that TTCN-3 tools retain the type information and encode the values accordingly.

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	Property settings
Bit String	list	single value: list, size: length	No	length	No	No	No	No	ignore	No	No	ignore	No
Boolean	list	list	No	No	No	No	No	No	ignore	No	No	No	No
Choice	list	list	No	No	No	No	convert to full type	No	ignore	No	No	No	No
Embedded-pdv (see a)	list	No	No	No	No	No	convert to full type	No	ignore	No	No	No	No
Enumerated	list	list	No	No	No	No	No	No	ignore	No	No	No	No
External (see a)	list	No	No	No	No	No	convert to full type	No	ignore	No	No	No	No
Instance-of (see a and b)	list	list	No	No	No	No	convert to full type	No	ignore	No	No	No	No
Integer	list	single value: list, value range: range	range and/or list (l,m)	No	No	No	No	No	ignore	No	No	No	No
Null	ignore	ignore	No	No	No	No	No	No	ignore	No	No	No	No
Object class field type	(see c)	(see c)	No	No	No	No	No	No	ignore	list	ignore	No	No
Object Descriptor (see e)	list	single value: list, size: length, perm.alphabet: range	No	length	range	No	No	No	ignore	No	No	No	No
Object Identifier	list	list	No	No	No	No	No	No	ignore	No	No	No	No
Octet String	list	single value: list, size: length	No	length	No	No	No	No	ignore	No	No	ignore	No
open type	No	No	No	No	No	anytype with list constraint	No	No	ignore	No (see m)	No (see m)	No	No
Real	list	single value: list, value range: range	range and/or list (n,o)	No	No	No	convert to full type	No	ignore	No	No	No	No
Relative Object Identifier (see d)	list	list	No	No	No	No	No	No	ignore	No	No	No	No
Restricted Character String Types	list	single value: list, size: length, perm.alpha-bet: range	range	length	range	No	No	Ignore (see g)	ignore	No	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	Property settings
Sequence	list	list	No	No	No	No	convert to full type	No	ignore	No	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	No
Set	list	list	No	No	No	No	convert to full type	No	ignore	No	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	No
TIME <including its derivations>	list	list	ignore (p)	No	No	No	ignore	No	No	No	No	No	ignore
Time Types (see a)	list	list	No	No	No	No	No	No	ignore	No	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	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 Recommendation ITU-T 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).
(l) If the lower and the upper boundaries of an ASN.1 range equal, the range shall be translated to a TTCN-3 list subtyping, corresponding to the allowed ASN.1 value (please note, there may be more than one ranges in an ASN.1 range subtype specification).

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	Property settings

(m) For each range, not obeying rule (l) above a TTCN-3 subtype range shall be generated, considering the following:
If the lower boundary of an ASN.1 range is MIN, the lower boundary of the corresponding TTCN-3 range shall be inclusive and its value shall be the lower boundary of the ASN.1 type's parent type or **`-infinity`** (if the parent type is a built-in type or has no lower boundary, i.e. MIN - either open or closed - is used along the whole derivation chain).
If the lower boundary of an ASN.1 range is MIN<, the lower boundary of the corresponding TTCN-3 range shall be inclusive and its value shall be the lower boundary of the ASN.1 type's parent type plus 1 or **`-infinity`** (if the parent type is a built-in type or has no lower boundary, i.e. MIN - either open or closed - is used along the whole derivation chain).
If the lower boundary of an ASN.1 range is a value and is a closed boundary (i.e. not MIN and does not include the "<" symbol), the lower boundary of the corresponding TTCN-3 range shall be inclusive and its value shall be the ASN.1 lower boundary.
If the lower boundary of an ASN.1 range is a value and is an open boundary (i.e. not MIN< and does include the "<" symbol), the lower boundary of the corresponding TTCN-3 range shall be inclusive and its value shall be the ASN.1 lower boundary plus 1.

If the upper boundary of an ASN.1 range is MAX, the upper boundary of the corresponding TTCN-3 range shall be inclusive and its value shall be the upper boundary of the ASN.1 type's parent type or **`infinity`** (if the parent type is a built-in type or has no upper boundary, i.e. MAX - either open or closed - are used along the whole derivation chain).
If the upper boundary of an ASN.1 range is <MAX, the upper boundary of the TTCN-3 range shall be inclusive and its value shall be the upper boundary of the ASN.1 type's parent type minus 1 or **`infinity`** (if the parent type is a built-in type or has no upper boundary, i.e. MAX - either open or closed - are used along the whole derivation chain).
If the upper boundary of an ASN.1 range is a value and is a closed boundary (i.e. not MAX and does not include the "<" symbol), the upper boundary of the TTCN-3 range shall be inclusive and its value shall be the ASN.1 upper boundary.
If the upper boundary of an ASN.1 range is a value and is an open boundary (i.e. not <MAX and does include the "<" symbol), the upper boundary of the TTCN-3 range shall be inclusive and its value shall be the ASN.1 upper boundary minus 1.

(n) If the lower and the upper boundaries of an ASN.1 range equal, the range shall be translated to a TTCN-3 list subtyping, corresponding to the allowed ASN.1 value (please note, there may be more than one ranges in an ASN.1 range subtype specification).

(o) For each range, not obeying rule (n) above a TTCN-3 subtype range shall be generated, considering the following:
If the lower boundary of an ASN.1 range is MINUS-INFINITY, the lower bound of the corresponding TTCN-3 range shall be **`-infinity`**.
If the lower boundary of an ASN.1 range is MIN, the lower boundary of the corresponding TTCN-3 range shall be inclusive and its value shall be the lower boundary of the ASN.1 type's parent type or **`-infinity`** (if the parent type is a built-in type or has no lower boundary, i.e. MIN and/or MINUS-INFINITY - either open or closed - are used along the whole derivation chain).
If the lower boundary of an ASN.1 range is MIN<, the lower boundary of the corresponding TTCN-3 range shall be either an exclusive value equivalent to the lower boundary of the ASN.1 type's parent type or **`-infinity`** (if the parent type is a built-in type or has no lower boundary, i.e. MIN and/or MINUS-INFINITY - either open or closed - are used along the whole derivation chain).
If the lower boundary of an ASN.1 range is a numerical value and is a closed boundary (i.e. not MIN or MINUS-INFINITY and does not include the "<" symbol), the lower boundary of the corresponding TTCN-3 range shall be inclusive and its value shall be the ASN.1 lower boundary.
If the lower boundary of an ASN.1 range is a numerical value and is an open boundary (i.e. not MIN< or MINUS-INFINITY< and does include the "<" symbol), the lower boundary of the corresponding TTCN-3 range shall be exclusive and its value shall be the ASN.1 lower boundary.

If an upper boundary of an ASN.1 range is NOT-A-NUMBER, a TTCN-3 range with the upper boundary **`infinity`** and the list subtype value **`not_a_number`** shall be generated for this range.
If an upper boundary of an ASN.1 range is <NOT-A-NUMBER, a TTCN-3 range with the upper boundary **`infinity`** shall be generated for this range.
If the upper boundary of an ASN.1 range is PLUS-INFINITY, the upper bound of the corresponding TTCN-3 range shall be **`infinity`**.
If the upper boundary of an ASN.1 range is MAX, the upper boundary of the corresponding TTCN-3 range shall be inclusive and its value shall be the upper boundary of the ASN.1 type's parent type or **`infinity`** (if the parent type is a built-in type or has no upper boundary, i.e. MAX, PLUS-INFINITY or NOT-A-NUMBER - either open or closed - are used along the whole derivation chain).
If the upper boundary of an ASN.1 range is <MAX, the upper boundary of the corresponding TTCN-3 range shall be either an exclusive value equivalent to the upper boundary of the ASN.1 type's parent type or **`infinity`** (if the parent type is a built-in type or has no upper boundary, i.e. MAX, PLUS-INFINITY or NOT-A-NUMBER - either open or closed - are used along the whole derivation chain).
If the upper boundary of an ASN.1 range is a numerical value and is a closed boundary (i.e. not MAX, PLUS-INFINITY or NOT-A-NUMBER and does not include the "<" symbol), the upper boundary of the corresponding TTCN-3 range shall be inclusive and its value shall be the ASN.1 upper boundary.
If the upper boundary of an ASN.1 range is a numerical value and is an open boundary (i.e. not <MAX, <PLUS-INFINITY or <NOT-A-NUMBER and does include the "<" symbol), the upper boundary of the corresponding TTCN-3 range shall be exclusive and its value shall be the ASN.1 upper boundary.

(p) In case of TIME types range subtyping means duration, time point and recurrance ranges rather than value range.

ASN.1 allows using set arithmethics in subtype constraints. In many cases these set arithmetics expressions shall be calculated before the translation to be able to convert the ASN.1 subtype into its associated TTCN-3 type. However, in some cases expressions can be translated to TTCN-3 in different ways; e.g. a UNION of consequtive integer values can be translated into a list of values, into a value range or a mixture of the two, while all these forms denote the same set of values. This standard leaves this choice open for tool implementations. The only requirement is that the resulted associated TTCN-3 type shall conform to this clause and clause 6 of of ES 201 873-1 [1].

9.2 Transformation rules for values

In case of real values, the base used in the value notation (2 or 10) shall be retained by the tool to be able to produce the correct encoding of the value. However, from the point of view of TTCN-3 relational operations only the numerical value counts.

9.3 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:2008" means encoded according to Recommendation ITU-T X.690 [6] (BER);

b) "CER:2008" means encoded according to Recommendation ITU-T X.690 [6] (CER);

c) "DER:2008" means encoded according to Recommendation ITU-T X.690 [6] (DER);

d) "PER-BASIC-UNALIGNED:2008" means encoded according to (Unaligned PER) Recommendation ITU-T X.691 [7];

e) "PER-BASIC-ALIGNED:2008" means encoded according to Recommendation ITU-T X.691 [7] (Aligned PER);

f) "PER-CANONICAL-UNALIGNED:2008" means encoded according to (Canonical Unaligned PER) Recommendation ITU-T X.691 [7];

g) "PER-CANONICAL-ALIGNED:2008" means encoded according to Recommendation ITU-T X.691 [7] (Canonical Aligned PER);

h) "BASIC-XER:2008" means encoded according to Recommendation ITU-T X.693 [8] (Basic XML encoding rules);

i) "CANONICAL-XER:2008" means encoded according to Recommendation ITU-T X.693 [8] (Canonical XML encoding rules);

j) "EXTENDED-XER:2008" means encoded according to Recommendation ITU-T X.693 [8] (Extended XML encoding rules).

The encodings of previous ASN.1 versions rule (e.g. 1988, 1994, 1997 or 2002) 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 definite form of the length octets (see clause 8.1.3.4 of Recommendation ITU-T X.690 [6]) in case of BER, CER and DER encodings or the single octet length determinant (see clause 11.9.3.6 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 8.1.3.5 of Recommendation ITU-T X.690 [6]) in case of BER, CER and DER encodings or the two octets length determinant (see clause 11.9.3.7 of Recommendation ITU-T 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 8.1.3.6 of Recommendation ITU-T 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 form specified by selected alternative of the **`encoding`** field (see clause 8.18 of Recommendation ITU-T X.690 [6] and 29 of Recommendation ITU-T X.691 [7]). When this attribute is used for import statements, component type definitions, groups or TTCN-3 modules, it shall have effect on types, constants, variables and templates based on ASN.1 EXTERNALtypes only. If the conditions set in clauses 8.18.6 to 8.18.8 of Recommendation ITU-T X.690 [6] and the specified attribute are not met, this shall cause an error.

f) "TeletexString" means that the given value shall be encoded and decoded as the ASN.1 type TeletexString (see clause 8.23 of Recommendation ITU-T X.690 [6] and clause 30 of Recommendation ITU-T X.691 [7]).

g) "VideotexString" means that the given value shall be encoded and decoded as the ASN.1 type VideotexString (see clause 8.23 of Recommendation ITU-T X.690 [6] and clause 30 of Recommendation ITU-T X.691 [7]).

h) "GraphicString" means that the given value shall be encoded and decoded as the ASN.1 type GraphicString (see clause 8.23 of Recommendation ITU-T X.690 [6] and clause 30 of Recommendation ITU-T X.691 [7]).

i) "GeneralString" means that the given value shall be encoded and decoded as the ASN.1 type GeneralString (see clause 8.23 of Recommendation ITU-T X.690 [6] and clause 30 of Recommendation ITU-T 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 A.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 ::= Identifier [ "(" Number ")"] |
                                   Number
1004. ObjectIdentifierValue ::= ObjectIdentifierKeyword "{" ObjIdComponentList "}"
1005. ObjIdComponentList ::= {ObjIdComponent}+
1006. ObjIdComponent ::= DefinitiveObjIdComponent | ReferencedValue
/* STATIC SEMANTICS - ReferencedValue shall be an object identifier value */
1007. NameAndNumberForm ::= Identifier "(" NumberForm | ReferencedValue ")"
/* STATIC SEMANTICS - ReferencedValue shall be an integer value */
1008. ObjectIdentifierSpec := ObjectIdentifierKeyword "{" ObjIdComponentListSpec "}"
1009. ObjIdComponentListSpec := {ObjIdComponentSpec}+
1010. ObjIdComponentSpec := ObjIdComponent | AnyValue | AnyOrOmit | TemplateRefWithParList
/* STATIC SEMANTICS - TemplateRefWithParList shall be of integer type */
```

A.2 Amended core language BNF productions and static semantics

Additions to clause A.1.6 of ES 201 873-1 [1] are identified by underlined font, deletions are identified by strikethrough font. In case of contradiction between the above clause of Part 1 [1] and this clause (i.e. parts of the productions not marked by ~~strikethrough~~ font is changed in Part 1), Part 1 [1] takes precedence, i.e. tools supporting the present document shall apply the insertions and deletions of this clause to the actual Part 1 [1] production or static semantics rule automtically.

```
5. GlobalModuleId ::= ModuleIdentifier [Dot DefinitiveIdentifier]
51. ValueOrRange ::= RangeDef | ConstantExpression | Type
/* 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 */
101. TemplateBody ::= (SimpleSpec | FieldSpecList | ArrayValueOrAttrib | ObjectIdentifierSpec )
[ExtraMatchingAttributes]
/* STATIC SEMANTICS - Within TeplateBody the ArrayValueOrAttrib can be used for array, record,
record of and set of types. */
148. LowerBound ::= SingleConstExpression | ( Minus InfinityKeyword )
149. 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*/
455. PredefinedType ::= BitStringKeyword |
                        BooleanKeyword |
                        CharStringKeyword |
                        UniversalCharString |
                        IntegerKeyword |
                        OctetStringKeyword |
                        HexStringKeyword |
                        VerdictTypeKeyword |
                        FloatKeyword |
                        AddressKeyword |
```

```
                        DefaultKeyword |
                        AnyTypeKeyword |
                        ObjectIdentifierKeyword
474. 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): Additional information on object identifiers

C.1 The top-level arcs of the OID tree

Annex A of Recommendation ITU-T X.660 [11] defines the tree of object identifier components shown below. Some object identifier components defined in Recommendation ITU-T X.660 [11] may use the name form (without defining the numerical value of the component) in object identifier value notations (see table C.1). 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 1: Names below are given according to the TTCN-3 syntax, i.e. all dash characters are replaced by underscore characters.

NOTE 2: Further arcs can be found via the repository of Object IDentifiers (OIDs) that is freely available at http://www.oid-info.com/.

Table C.1: The top-level arcs of the OID tree

root arcs		arcs beneath the root		third-level arcs		allowed as
secondary identifier (primary integer)	Unicode label integer valued --- non-integer	secondary identifier (primary integer)	Unicode label integer valued --- non-integer		Unicode label integer valued | non-integer	***NameForm***
itu_t(0), *ccitt(0)*	"0" --- "ITU-T"	recommendation(0)	**"0"** --- **"Recommendation"**	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) w(23) x(24) y(25) z(26)	"1" | "A" "2" | "B" "3" | "C" "4" | "D" "5" | "E" "6" | "F" "7" | "G" "8" | "H" "9" | "I" "10" | "J" "11" | "K" "12" | "L" "13" | "M" "14" | "N" "15" | "O" "16" | "P" "17" | "Q" "18" | "R" "19" | "S" "20" | "T" "21" | "U" "22" | "V" "23" | "W" "24" | "X" "25" | "Y" "26" | "Z"	Yes
		question(1)	"1" --- (see note 3)	(see note 4)		arc beneath root: Yes, third-level arc: No
		administration(2)	"2" --- "Administration"	(see note 5)		arc beneath root: Yes, third-level arc: No

root arcs		arcs beneath the root		third-level arcs		allowed as *NameForm*
secondary identifier (primary integer)	Unicode label integer valued --- non-integer	secondary identifier (primary integer)	Unicode label integer valued --- non-integer		Unicode label integer valued | non-integer	
		network_operator(3)	"3" --- "Network-Operator"	(see note 6)		arc beneath root: Yes, third-level arc: No
		identified_organization(4)	"4" --- "Identified-Organization"			Yes
		r_recommendation(5)	"5" --- "R-Recommendation"			No
iso(1)	"1" --- **"ISO"**	standard(0) (note 7)	"0" --- "Standard"			Yes
		registration_authority(1)	"1" --- "Registration-Authority"			Yes
		member_body(2)	"2" --- "Member-Body"			Yes
		identified_organization(3)	"3" --- "Identified-Organization"			Yes
joint_iso_itu_t(2), *joint_iso_ccitt(2)*	"2" --- "Joint-ISO-ITU-T"					Yes

NOTE 3: The arcs beneath the arc with the primary integer value **1** (secondary identifier **question**) have never been used and are of historical interest only. A non-integer Unicode label has not been assigned to these arcs.

NOTE 4: Arcs below the arc with the primary integer value **1** (secondary identifier **question**) have primary integer values corresponding to ITU-T study groups, qualified by the study period. The value is computed by the formula:

Study Group number + (Study Period * 32)

where "Study Period" has the value 0 for 1984-1988, 1 for 1988-1992, etc., and the multiplier is 32 decimal. The arcs below each study group have primary integer values corresponding to the Questions assigned to that study group. Arcs below this are determined as necessary by the group (e.g. Working Party or special Rapporteur group) assigned to study the question.

NOTE 5: The primary integer values (and hence integer-valued Unicode labels) that are the values of data country codes (DCCs) as defined in annex J of Recommendation ITU-T X.121 [i.4]. These arcs have a non-integer Unicode label and a secondary identifier, both consisting of the two-letter alpha-2 code element (see reference to ISO 3166-1 [i.3], country code elements for the corresponding country.

NOTE 6: The primary integer values (and hence integer-valued Unicode labels) that are the values of data network identification codes (DNICs) as defined in annex I of Recommendation ITU-T X.121 [i.4]. These arcs have no non-integer Unicode labels and no secondary identifiers assigned by default.

NOTE 7: This arc (but only this) can also be used as iec(1) standard(0)….

C.2 Character patterns to match OID IRI-s

The template `t_OID_IRI` below is matching all legal OID-IRI values, containing at least the arcs given in table C.1. It can also be used as an example and basis to specify patterns to match OID-IRI values containing further arc(s) or to create template to match values containing less arcs by deleting the unneeded parts of the pattern below (please note, that the root arc is mandatory). Please note, that the pattern below should not be used for long arcs (see Recommendation ITU-T X.660 [11] and http://www.oid-info.com/faq.htm#iri for more information).

```
module OID_IRI {

  //*****************************************************************************
  //*
  //*  @reference ES 201 873-7 table C.1
  //*
  //*  @desc Matches any valid (non-relative) International Resource Identifier
  //*        containing at least the arcs in table C.1
  //*
  //*  @remark It does not guarantee that the whole identifier is correct, check
  //*          the first arcs only; when the further arcs are known and wanted
  //*          to be checked, the "(/?*)#(,1)" fragment shall be replaced by
  //*          the pattern corresponding to the arcs beneath the ones defined
  //*          in table C.1
  //*
  //*  @status non-verified
  //*
  //*****************************************************************************
  template universal charstring t_OID_IRI := pattern
  "/((0|ITU-T)/(((0|Recommendation)/((1|A)|(2|B)|(3|C)|(4|D)|(5|E)|(6|F)|(7|G)|" &
     "(8|H)|(9|I)|(10|J)|(11|K)|(12|L)|(13|M)|(14|N)|(15|O)|(16|P)|(17|Q)|" &
     "(18|R)|(19|S)|(20|T)|(21|U)|(22|V)|(23|W)|(24|X)|(25|Y)|(26|Z)))|" &
     "1|(2|Administration)|(3|Network-Operator)|(4|Identified-Organization)|" &
     "(5|R-Recommendation))|" &
  "((1|ISO)/((0|Standard)|(1|Registration-Authority)|(2|Member-Body)|" &
     "(3|Identified-Organization)))|" &
  "(2|Joint-ISO-ITU-T))" &
  "(/?*)#(,1)" //this fragment may be replaced by the arcs beneath the ones in C.1


  //*****************************************************************************
  //*
  //*  @reference ITU-T X.680 $35
  //*
  //*  @desc Matches valid relative International Resource Identifiers
  //*
  //*  @remark It does not guarantee that the identifier is correct, checks
  //*          only if it is provided like a relative value
  //*
  //*  @status non-verified
  //*
  //*****************************************************************************
  template universal charstring t_RELATIVE_OID_IRI := pattern
  "[^/]" &
  "(/?*)#(,1)" //this fragment may be replaced by the arcs wanted

}
```

Annex D (informative): Deprecated features

Void.

Annex E (informative): Example patterns for ASN.1 time types

As specified in clause 9.1 of the present document, the ASN.1 date and time types are transformed to the TTCN-3 **charstring** type. This annex contains TTCN-3 patterns that can be used either to create pattern-constrained types corresponding to the ASN.1 date and time types or be used in templates to match correctly formed date and time value only.

In this annex the TTCN-3 source code documentation format specified in [i.2] is used (please see the compatibility statement in clause 4.1 of the present document).

E.1 Patterns corresponding to unconstrained time types

This clause contains two TTCN-3 modules. The module called "**nc**" contains subsidiary constants that specify the patterns for atomic components of the complete date and time patterns.

For matching an unconstrained TIME type use the template **t_ISO8601AllFormats** from the module "**ISO8601DateTimePatterns**" below (values of the ASN.1 TIME type are character strings formatted according to ISO 8601 [i.1]).

```
module nc
{
  //*****************************************************************************
  //*
  //*  @desc Unconstrained charstring constants referenced from the
  //*        ISO8601 date/time patterns
  //*
  //*  @remark Whatever components are possible, defined as constant strings
  //*          referenced in the patterns (e.g. the optional T designator in
  //*          time representations or the optional century expansion in dates)
  //*          to allow easy modification via changing the constants only
  //*  @remark No. of century expansion digits and digits of decimal fraction
  //*          are not limited; intentionally the "#(1,)" pattern is used
  //*          instead of "+" to allow adding digit number limitations easily
  //*  @remark Defined constants are: dash, colon, century, year,
  //*          centuryExpansion, month, monthDurAlt, week, dayOfWeek,
  //*          dayOfMonth, dayOfMonthDurAlt, dayOfYear, dayOfYearDurAlt,
  //*          hour, minute, second, fraction, endOfDay, endOfDayExt,
  //*          nums, timeZone, timeZoneExt, durTime
  //*  @remark Components used also as optional (i.e. followed by #(,1)
  //*          in any of the date/time patterns) shall have an external
  //*          enclosing bracket; other constants need not have this
  //*
  //*  @status verified
  //*
  //*****************************************************************************
  const charstring
    dash := "-",
    colon := ":",
    century := "[0-9]#2",
    year := "[0-9]#4",
    yearExpansion := "([\+\-][0-9]#(0,))", //also allows zero additional digits!
    yearExpansionOpt := "([\+\-][0-9]#(0,))#(,1)", //also allows zero additional
    month := "(0[1-9]|1[0-2])",//need outer brackets: optional in some patterns
    monthDurAlt := "(0[0-9]|1[01])",
    week := "(W(0[1-9]|[1-4][0-9]|5[0-3]))",//need outer brackets: optional
    dayOfWeek := "[1-7]",
    dayOfMonth := "(0[1-9]|[12][0-9]|3[01])",//need outer brackets: optional
    dayOfMonthDurAlt := "[012][0-9]|30",
    dayOfYear := "(0(0[1-9]|[1-9][0-9])|[12][0-9][0-9]|3([0-5][0-9]|6[0-5]))",//optional
    dayOfYearDurAlt := "[012][0-9][0-9]|3([0-5][0-9]|6[0-4])",
    hour := "([01][0-9]|2[0-3])",
    minute := "([0-5][0-9])", //need outer brackets: optional
    second := "([0-5][0-9])",
    //differentiation of fractions used in hours, minutes and seconds and duration
    //is needed to allow setting of different number of decimal digits & subtyping
    hFraction := "([,.][0-9]#(1,))", //need outer brackets: optional
    mFraction := nc.hFraction,
```

```
    sFraction := nc.hFraction,
    dFraction := nc.hFraction,
    optionalT := "T#(,1)",
    endOfDay := "24(00(00([,.]0#(1,))#(,1)|[,.]0#(1,))#(,1)|[,.]0#(1,))#(,1)",
    endOfDayExt := "24:00(:00([,.]0#(1,))#(,1)|[,.]0#(1,))#(,1)",
    nums := "[0-9]#(1,)",
    timeZone := "[\+\-]([01][0-9]|2[0-3])([0-5][0-9])#(,1)",
    optZorTimeZone := "(Z|[\+\-]([01][0-9]|2[0-3])([0-5][0-9])#(,1))#(,1)",
    timeZoneExt := "[\+\-]([01][0-9]|2[0-3])(:[0-5][0-9])#(,1)",
    optZorTimeZoneExt := "(Z|[\+\-]([01][0-9]|2[0-3])(:[0-5][0-9])#(,1))#(,1)",
    durTime := "(T[0-9]#(1,)"&
             "(H([0-9]#(1,)(M([0-9]#(1,)(S|[,.][0-9]#(1,)S))#(,1)|[,.][0-9]#(1,)" &
             "[MS]|S))#(,1)|M([0-9]#(1,)(S|[,.][0-9]#(1,)S)|[,.][0-9]#(1,)M)#(,1)|" &
             "S|[,.][0-9]#(1,)[HMS]))" //optional
//Used in atomic patterns only
    ,endOfDaywFraction := "24(00(00[,.]0#(1,)|[,.]0#(1,))|[,.]0#(1,))"
    ,endOfDaywFractionExt := "24:00(:00[,.]0#(1,)|[,.]0#(1,))"

}//end module


module ISO8601DateTimePatterns {

  //=========================================================================
  // Imports
  //=========================================================================

  import from nc all;

  //=========================================================================
  // Templates
  //=========================================================================

  //============================DATE FORMS=====================================

  //*************************************************************************
  //*
  //*  @reference ISO_8601 $4.1.2
  //*
  //*  @desc Matches all calendar date representations (complete, reduced
  //*        accuracy and expanded) in basic formats
  //*
  //*  @status verified
  //*
  //*************************************************************************
  template charstring  t_ISO8601DateCalendarBasic := pattern
    "{nc.yearExpansionOpt}({nc.century}|{nc.year}({nc.month}{nc.dayOfMonth}|" &
    "{dash}{nc.month})#(,1))";


  //*************************************************************************
  //*
  //*  @reference ISO_8601 $4.1.2
  //*
  //*  @desc Matches all calendar date representations (complete and expanded)
  //*        in extended formats
  //*
  //*  @status verified
  //*
  //*************************************************************************
  template charstring  t_ISO8601DateCalendarExtended := pattern
    "{nc.yearExpansionOpt}{nc.year}{dash}{nc.month}{dash}{nc.dayOfMonth}";


  //*************************************************************************
  //*
  //*  @reference ISO_8601 $4.1.2
  //*
  //*  @desc Matches all calendar date representations (complete, reduced
  //*        accuracy and expanded) in basic and extended formats
  //*
  //*  @status verified
  //*
```

```
//*****************************************************************************
template charstring  t_ISO8601DateCalendar := (
    t_ISO8601DateCalendarBasic,
    t_ISO8601DateCalendarExtended
)


//*****************************************************************************
//*
//*  @reference ISO_8601 $4.1.3
//*
//*  @desc Matches all ordinal date representations (complete, reduced
//*        accuracy and expanded) in basic formats
//*
//*  @status verified
//*
//*****************************************************************************
template charstring  t_ISO8601DateOrdinalBasic := pattern
  "{nc.yearExpansionOpt}{nc.year}{nc.dayOfYear}";


//*****************************************************************************
//*
//*  @reference ISO_8601 $4.1.3
//*
//*  @desc Matches all ordinal date representations (complete, reduced
//*        accuracy and expanded) in basic formats
//*
//*  @status verified
//*
//*****************************************************************************
template charstring  t_ISO8601DateOrdinalExtended := pattern
  "{nc.yearExpansionOpt}{nc.year}{dash}{nc.dayOfYear}";


//*****************************************************************************
//*
//*  @reference ISO_8601 $4.1.3
//*
//*  @desc Matches all ordinal date representations (complete, reduced
//*        accuracy and expanded) in basic and extended formats
//*
//*  @status verified
//*
//*****************************************************************************
template charstring  t_ISO8601DateOrdinal := (
   t_ISO8601DateOrdinalBasic,
   t_ISO8601DateOrdinalExtended
)


//*****************************************************************************
//*
//*  @reference ISO_8601 $4.1.4
//*
//*  @desc Matches all week date representations (complete, reduced
//*        accuracy and expanded) in basic formats
//*
//*  @status verified
//*
//*****************************************************************************
template charstring  t_ISO8601DateWeekBasic := pattern
  "{nc.yearExpansionOpt}{nc.year}{nc.week}{nc.dayOfWeek}#(,1)";


//*****************************************************************************
//*
//*  @reference ISO_8601 $4.1.4
//*
//*  @desc Matches all week date representations (complete, reduced
//*        accuracy and expanded) in extended formats
//*
//*  @status verified
//*
```

```
//*****************************************************************************
template charstring  t_ISO8601DateWeekExtended := pattern
  "{nc.yearExpansionOpt}{nc.year}{dash}{nc.week}({dash}{nc.dayOfWeek})#(,1)";


//*****************************************************************************
//*
//*  @reference ISO_8601 $4.1.4
//*
//*  @desc Matches all week date representations (complete, reduced
//*        accuracy and expanded) in basic and extended formats
//*
//*  @status verified
//*
//*****************************************************************************
template charstring  t_ISO8601DateWeek := (
   t_ISO8601DateWeekBasic,
   t_ISO8601DateWeekExtended
)


//*****************************************************************************
//*
//*  @reference ISO_8601 $4.1
//*
//*  @desc Matches all date representations (complete, reduced accuracy
//*        and expanded) in basic formats
//*
//*  @status verified
//*
//*****************************************************************************
template charstring  t_ISO8601DateBasic := (
   t_ISO8601DateCalendarBasic,
   t_ISO8601DateOrdinalBasic,
   t_ISO8601DateWeekBasic
)


//*****************************************************************************
//*
//*  @reference ISO_8601 $4.1
//*
//*  @desc Matches all date representations (complete, reduced accuracy
//*        and expanded) in extended formats
//*
//*  @status verified
//*
//*****************************************************************************
template charstring  t_ISO8601DateExtended := (
  t_ISO8601DateCalendarExtended,
  t_ISO8601DateOrdinalExtended,
  t_ISO8601DateWeekExtended
)


//*****************************************************************************
//*
//*  @reference ISO_8601
//*
//*  @desc Matches all date representations (complete, reduced accuracy and
//*        expanded) in basic and extended formats
//*
//*  @status verified
//*
//*****************************************************************************
template charstring  t_ISO8601Date := (
   t_ISO8601DateBasic,
   t_ISO8601DateExtended
)

//============================TIME-OF-DAY============================
```

```
//*****************************************************************************
//*
//*  @reference ISO_8601 $4.2
//*
//*  @desc Matches all time of day representations in basic formats
//*        Supports all time of day representations as local time: complete
//*        ($4.2.2.2), reduced accuracy($4.2.2.3) and with decimal fraction
//*        ($4.2.2.4), midnight($4.2.3); UTC of day ($4.2.4) and difference
//*        between local time and UTC of day ($4.2.5)and the optional time
//*        designator
//*
//*  @status verified
//*
//*****************************************************************************
template charstring  t_ISO8601TimeBasic := pattern
  "{nc.optionalT}({nc.hour}({nc.minute}({nc.second}{nc.sFraction}#(,1)|60|" &
  "{nc.mFraction})#(,1)|60|{nc.hFraction})#(,1)|{nc.endOfDay})" &
  "{nc.optZorTimeZone}";


//*****************************************************************************
//*
//*  @reference ISO_8601 $4.2
//*
//*  @desc Matches all time of day representations in extended formats
//*        Supports all time of day representations as local time: complete
//*        ($4.2.2.2), reduced accuracy($4.2.2.3) and with decimal fraction
//*        ($4.2.2.4), midnight($4.2.3); UTC of day ($4.2.4) and difference between
//*        local time and UTC of day ($4.2.5)and the optional time designator
//*
//*  @status verified
//*
//*****************************************************************************
template charstring  t_ISO8601TimeExtended := pattern
  "{nc.optionalT}({nc.hour}{colon}({nc.minute}({colon}({nc.second}" &
  "{nc.sFraction}#(,1)|60)|{nc.mFraction})#(,1)|60)|{nc.endOfDayExt})" &
  "{nc.optZorTimeZoneExt}";


//*****************************************************************************
//*
//*  @reference ISO_8601 $4.2
//*
//*  @desc Matches all time of day representations in basic and extended formats
//*        Supports all time of day representations as local time: complete
//*        ($4.2.2.2), reduced accuracy($4.2.2.3) and with decimal fraction
//*        ($4.2.2.4), midnight($4.2.3); UTC of day ($4.2.4) and difference between
//*        local time and UTC of day ($4.2.5)and the optional time designator
//*
//*  @status verified
//*
//*****************************************************************************
template charstring  t_ISO8601Time := (
   t_ISO8601TimeBasic,
   t_ISO8601TimeExtended
)


//============================DATE-TIME FORMATS================================

//*****************************************************************************
//*
//*  @reference ISO_8601 $4.3
//*
//*  @desc Matches all date/time of day representations in basic formats
//*
//*  @remark Acc. to $4.3... ?) the date part shall always be complete
//*  @remark Omitting the T designer between the date and time parts is not
//*          supported
//*
//*  @status verified
//*
//*****************************************************************************
```

```
template charstring  t_ISO8601DateTimeBasic := pattern
  "{nc.yearExpansionOpt}{nc.year}({nc.month}{nc.dayOfMonth}|{nc.dayOfYear}|" &
  "{nc.week}{nc.dayOfWeek})T({nc.hour}({nc.minute}({nc.second}" &
  "{nc.sFraction}#(,1)|60|{nc.mFraction})#(,1)|60|{nc.hFraction})#(,1)" &
  "{nc.optZorTimeZone}|{nc.endOfDay}{nc.optZorTimeZone})";


//***************************************************************************
//*
//*  @reference ISO_8601 $4.3
//*
//*  @desc Matches all date/time of day representations in all formats
//*
//*  @remark Acc. to $4.3.3 c) the date part shall always be complete
//*  @remark Omitting the T designer between the date and time parts is not
//*          supported
//*
//*  @status verified
//*
//***************************************************************************
template charstring  t_ISO8601DateTimeExtended := pattern
  "{nc.yearExpansionOpt}{nc.year}{dash}({nc.month}{dash}{nc.dayOfMonth}|" &
  "{nc.dayOfYear}|{nc.week}{dash}{nc.dayOfWeek})T({nc.hour}{colon}" &
  "({nc.minute}({colon}({nc.second}{nc.sFraction}#(,1)|60)|{nc.mFraction})" &
  "#(,1)|60){nc.optZorTimeZoneExt}|{nc.endOfDayExt}{nc.optZorTimeZoneExt})";


//***************************************************************************
//*
//*  @reference ISO_8601 $4.3
//*
//*  @desc Matches all date/time of day representations in all formats
//*
//*  @remark Acc. to $4.3... ?) the date part shall always be complete
//*  @remark Omitting the T designer between the date and time parts is not
//*          supported
//*
//*  @status verified
//*
//***************************************************************************
template charstring  t_ISO8601DateTime := (
   t_ISO8601DateTimeBasic,
   t_ISO8601DateTimeExtended
)


//============================DURATION============================

//***************************************************************************
//*
//*  @reference ISO_8601 $4.4.3.2
//*
//*  @desc Matches duration representations containing only date components
//*        and using the format with designators; Both week and caledar dates
//*        are supported (the alternative format in $4.4.3.3 is excluded: it is
//*        covered by t_ISO8601DurationAlternative...)
//*
//*  @status verified
//*
//***************************************************************************
template charstring  t_ISO8601DurationDesignDate :=  pattern
  "P{nc.nums}(Y({nc.nums}(M({nc.nums}(D|{nc.dFraction}D))#(,1)|" &
  "{nc.dFraction}[MD]|D))#(,1)|{nc.dFraction}[YMDW]|M({nc.nums}" &
  "(D|{nc.dFraction}D)|{nc.dFraction}D)#(,1)|D|W)"


//***************************************************************************
//*
//*  @reference ISO_8601 $4.4.3.2
//*
//*  @desc Matches duration representations containing time components
//*        only and using the format with designators
//*
//*  @status verified
//*
//***************************************************************************
template charstring  t_ISO8601DurationDesignTime :=  pattern
  "PT{nc.nums}(H({nc.nums}(M({nc.nums}(S|{nc.dFraction}S))#(,1)|" &
```

```
  "{nc.dFraction}[MS]|S))#(,1)|{nc.dFraction}[HMS]|M({nc.nums}(S|" &
  "{nc.dFraction}S)|{nc.dFraction}M)#(,1)|S)"


//***************************************************************************
//*
//*  @reference ISO_8601 $4.4.3.2
//*
//*  @desc Matches duration representations using the format with designators;
//*        Both week and caledar dates are supported (the alternative format in
//*        $4.4.3.3 is excluded: it is covered by t_ISO8601DurationAlternative...)
//*
//*  @status verified
//*
//***************************************************************************
template charstring  t_ISO8601DurationDesign :=  pattern
  "P({nc.nums}((Y({nc.nums}(M({nc.nums}(D{nc.durTime}#(,1)|{nc.dFraction}D)|" &
  "{nc.durTime}#(,1))|{nc.dFraction}[MD]|D{nc.durTime}#(,1))|{nc.durTime}" &
  "#(,1))|{nc.dFraction}[YMDW])|M({nc.nums}(D{nc.durTime}#(,1)|" &
  "{nc.dFraction}D)|{nc.dFraction}D|{nc.durTime}#(,1))|D{nc.durTime}#(,1)|" &
  "W{nc.durTime}#(,1))|{nc.durTime})"


//***************************************************************************
//*
//*  @reference ISO_8601 $4.4.3.3
//*
//*  @desc Matches duration representations using the alternative basic format
//*
//*  @remark Both caledar and ordinal dates are supported
//*  @remark Any of the components may be numerically 0;
//*  @remark Carry over points are 12 month, 30 days, (365 days in case of
//*          ordinal dates), 24 hours, 60 minutes and
//*          60 seconds; e.g.
//*          P00011200T000000,         P00010130T000000,
//*          P001365T000000,           P0001001T006000 and
//*          P0001-001T00:00:60
//*          are NOT accepted, only
//*          P00020000T000000,         P00010200T000000,
//*          P002000T000000,           P0001001T010000 and
//*          P0001001T000100
//*          are valid
//*  @remark The time part may be of reduced accuracy (by using decimal fraction
//*          or omitting lower order time element(s)), the date part shall be
//*          complete (i.e. numerically 0 elements shall not be omitted)
//*
//*  @status verified
//*
//***************************************************************************
template charstring  t_ISO8601DurationAlternativeBasic :=  pattern
  "P{nc.year}({nc.monthDurAlt}{nc.dayOfMonthDurAlt}|{nc.dayOfYearDurAlt})" &
  "T{nc.hour}({nc.minute}({nc.second}{nc.sFraction}#(,1)|{nc.mFraction})" &
  "#(,1)|{nc.hFraction})#(,1)"


//***************************************************************************
//*
//*  @reference ISO_8601 $4.4.3.3
//*
//*  @desc Matches duration representations using the alternative extended format
//*
//*  @remark Both caledar and ordinal dates are supported
//*  @remark Any of the components may be numerically 0;
//*  @remark Carry over points are 12 month, 30 days, (365 days in case of
//*          ordinal dates), 24 hours, 60 minutes and
//*          60 seconds; e.g.
//*          P0001-12-00T00:00:00,     P0001-01-30T00:00:00,
//*          P001-365T00:00:00,        P0001-001T00:60:00 and
//*          P0001-001T00:00:60
//*          are NOT accepted, only
//*          P0002-00-00T00:00:00,     P0001-02-00T00:00:00,
//*          P002-000T00:00:00,        P0001-001T01:00:00 and
//*          P0001-001T00:01:00
//*          are valid
//*  @remark The time part may be of reduced accuracy (by using decimal fraction
//*          or omitting lower order time element(s)), the date part shall be
//*          complete (i.e. numerically 0 elements shall not be omitted)
//*
```

```
//*  @status verified
//*
//*****************************************************************************
template charstring  t_ISO8601DurationAlternativeExtended :=  pattern
  "P{nc.year}{dash}({nc.monthDurAlt}{dash}{nc.dayOfMonthDurAlt}|" &
  "{nc.dayOfYearDurAlt})T{nc.hour}{colon}{nc.minute}({colon}{nc.second}" &
  "{nc.sFraction}#(,1)|{nc.mFraction})#(,1)"


//*****************************************************************************
//*
//*  @reference ISO_8601 $4.4.3.3
//*
//*  @desc Matches duration representations using the alternative format;
//*        both the basic and the extended formats are supported
//*
//*  @remark Both caledar and ordinal dates are supported
//*  @remark Any of the components may be numerically 0;
//*  @remark Carry over points are 12 month, 30 days, (365 days in case of
//*          ordinal dates), 24 hours, 60 minutes and
//*          60 seconds; e.g.
//*          P0001-12-00T00:00:00,     P0001-01-30T00:00:00,
//*          P001-365T00:00:00,        P0001-001T00:60:00 and
//*          P0001-001T00:00:60
//*          are NOT accepted, only
//*          P0002-00-00T00:00:00,     P0001-02-00T00:00:00,
//*          P002-000T00:00:00,        P0001-001T01:00:00 and
//*          P0001-001T00:01:00
//*          are valid
//*  @remark The time part may be of reduced accuracy (by using decimal fraction
//*          or omitting lower order time element(s)), the date part shall be
//*          complete (i.e. numerically 0 elements shall not be omitted)
//*
//*  @status verified
//*
//*****************************************************************************
template charstring  t_ISO8601DurationAlternative :=  (
  t_ISO8601DurationAlternativeBasic,
  t_ISO8601DurationAlternativeExtended
)


//*****************************************************************************
//*
//*  @reference ISO_8601 $4.4.3
//*
//*  @desc Matches duration representations using the both formats: with
//*        designators and the alternative representation; in the alternative
//*        representation only the basic format is allowed
//*
//*  @remark both caledar and ordinal dates are supported
//*  @remark any of the components may be numerically 0;
//*  @remark Carry over points are 12 month, 30 days, (365 days in case of
//*          ordinal dates), 24 hours, 60 minutes and
//*          60 seconds; e.g.
//*          P00011200T000000,         P00010130T000000,
//*          P001365T000000,           P0001001T006000 and
//*          P0001001T000060
//*          are NOT accepted, only
//*          P00020000T000000,         P00010200T000000,
//*          P002000T000000,           P0001001T010000 and
//*          P0001001T000100
//*          are valid
//*  @remark The time part may be of reduced accuracy (by using decimal fraction
//*          or omitting lower order time element(s)), the date part shall be
//*          complete (i.e. numerically 0 elements shall not be omitted)
//*
//*  @status verified
//*
//*****************************************************************************
template charstring  t_ISO8601DurationBasic :=  pattern
  "P({nc.year}({nc.monthDurAlt}{nc.dayOfMonthDurAlt}|{nc.dayOfYearDurAlt})" &
  "T{nc.hour}(({nc.minute}({nc.second}{nc.sFraction}#(,1)|{nc.mFraction})" &
  "#(,1)|{nc.hFraction})#(,1))|({nc.nums}((Y({nc.nums}(M({nc.nums}" &
  "(D{nc.durTime}#(,1)|{nc.dFraction}D)|{nc.durTime}#(,1))|{nc.dFraction}" &
```

```
  "[MD]|D{nc.durTime}#(,1))|{nc.durTime}#(,1))|{nc.dFraction}[YMDW])|" &
  "M({nc.nums}(D{nc.durTime}#(,1)|{nc.dFraction}D)|{nc.dFraction}D|" &
  "{nc.durTime}#(,1))|D{nc.durTime}#(,1)|W{nc.durTime}#(,1))|{nc.durTime}))"


//***************************************************************************
//*
//*  @reference ISO_8601 $4.4.3
//*
//*  @desc Matches duration representations using the both formats: with designators
//*        and the alternative representation; in the alternative representation only
//*        the extended format is allowed
//*
//*  @remark both caledar and ordinal dates are supported
//*  @remark any of the components may be numerically 0;
//*  @remark Carry over points are 12 month, 30 days, (365 days in case of
//*          ordinal dates), 24 hours, 60 minutes and
//*          60 seconds; e.g.
//*          P0001-12-00T00:00:00,     P0001-01-30T00:00:00,
//*          P001-365T00:00:00,        P0001-001T00:60:00 and
//*          P0001-001T00:00:60
//*          are NOT accepted, only
//*          P0002-00-00T00:00:00,     P0001-02-00T00:00:00,
//*          P002-000T00:00:00,        P0001-001T01:00:00 and
//*          P0001-001T00:01:00
//*          are valid
//*  @remark The time part may be of reduced accuracy (by using decimal fraction
//*          or omitting lower order time element(s)), the date part shall be
//*          complete (i.e. numerically 0 elements shall not be omitted)
//*
//*  @status verified
//*
//***************************************************************************
template charstring  t_ISO8601DurationExtended :=  pattern
  "P({nc.year}{dash}({nc.monthDurAlt}{dash}{nc.dayOfMonthDurAlt}|" &
  "{nc.dayOfYearDurAlt})T{nc.hour}{colon}{nc.minute}({colon}{nc.second}" &
  "{nc.sFraction}#(,1)|{nc.mFraction})#(,1)|({nc.nums}((Y({nc.nums}" &
  "(M({nc.nums}(D{nc.durTime}#(,1)|{nc.dFraction}D)|{nc.durTime}#(,1))|" &
  "{nc.dFraction}[MD]|D{nc.durTime}#(,1))|{nc.durTime}#(,1))|{nc.dFraction}" &
  "[YMDW])|M({nc.nums}(D{nc.durTime}#(,1)|{nc.dFraction}D)|{nc.dFraction}D|" &
  "{nc.durTime}#(,1))|D{nc.durTime}#(,1)|W{nc.durTime}#(,1))|{nc.durTime}))"


//***************************************************************************
//*
//*  @reference ISO_8601 $4.4.3
//*
//*  @desc Matches duration representations using both formats: with designators
//*        and the alternative representation; in the alternative representation both
//*        the basic and the extended formats are allowed
//*
//*  @remark both caledar and ordinal dates are supported
//*  @remark any of the components may be numerically 0;
//*  @remark Carry over points are 12 month, 30 days, (365 days in case of
//*          ordinal dates), 24 hours, 60 minutes and
//*          60 seconds; e.g.
//*          P0001-12-00T00:00:00,     P0001-01-30T00:00:00,
//*          P001-365T00:00:00,        P0001-001T00:60:00 and
//*          P0001-001T00:00:60
//*          are NOT accepted, only
//*          P0002-00-00T00:00:00,     P0001-02-00T00:00:00,
//*          P002-000T00:00:00,        P0001-001T01:00:00 and
//*          P0001-001T00:01:00
//*          are valid
//*  @remark The time part may be of reduced accuracy (by using decimal fraction
//*          or omitting lower order time element(s)), the date part shall be
//*          complete (i.e. numerically 0 elements shall not be omitted)
//*
//*  @status verified
//*
//***************************************************************************
template charstring  t_ISO8601Duration :=  (
   t_ISO8601DurationBasic,
   t_ISO8601DurationExtended
)


//***************************************************************************
```

```
//*
//*  @reference ISO_8601 $4.4.3
//*
//*  @desc Matches both complete and reduced accuracy (see note below)time
//*        interval representations identified by start and end and using the
//*        basic format
//*
//*  @remark $ 4.4.4.5 is contradictory as allows omitting higher order time
//*          elements in the part representing end date/time (the expression
//*          after the solidus), while the time part may also be of reduced
//*          accuracy (either by omitting the lower order elements or by a
//*          decimal fraction). This pattern resolves this conflict the
//*          following way: Higher order elements can be omitted in the date
//*          part of the expression following the solidus (in case of omitting
//*          the weeks also the "W" is omitted!) but not in the time part. The
//*          time part may be of reduced accuracy.
//*  @remark There is no requirement in ISO8601 to keep reduced accuracy
//*          compatible in the start and the end date/time; this may result
//*          "strange" or in some cases unclear (in these cases agreement is
//*          required between the communicating sides) time interval
//*          representations, however these are allowed by this template
//*
//*  @status verified
//*
//*****************************************************************************
template charstring  t_ISO8601IntervalStartEndBasic := pattern
  "{nc.yearExpansionOpt}{nc.year}({nc.month}{nc.dayOfMonth}|{nc.dayOfYear}|" &
  "{nc.week}{nc.dayOfWeek})T({nc.hour}({nc.minute}({nc.second}{nc.sFraction}" &
  "#(,1)|60|{nc.mFraction})#(,1)|60|{nc.hFraction})#(,1){nc.optZorTimeZone}|" &
  "{nc.endOfDay}{nc.optZorTimeZone})/({nc.yearExpansionOpt}{nc.year}" &
  "({nc.month}{nc.dayOfMonth}|{nc.dayOfYear}|{nc.week}{nc.dayOfWeek})|" &
  "({nc.month}#(,1){nc.dayOfMonth}|{nc.dayOfYear}|{nc.week}#(,1){nc.dayOfWeek}))"&
  "T({nc.hour}({nc.minute}({nc.second}{nc.sFraction}#(,1)|60|{nc.mFraction})" &
  "#(,1)|60|{nc.hFraction})#(,1){nc.optZorTimeZone}|{nc.endOfDay}" &
  "{nc.optZorTimeZone})"


//*****************************************************************************
//*
//*  @reference ISO_8601 $4.4.3
//*
//*  @desc Matches both complete and reduced accuracy (see note below)time interval
//*        representations identified by start and end and using the extended format
//*
//*  @remark $ 4.4.4.5 is contradictory as allows omitting higher order time
//*          elements in the part representing end date/time (the expression
//*          after the solidus), while the time part may also be of reduced
//*          accuracy (either by omitting the lower order elements or by a
//*          decimal fraction). This pattern resolves this conflict the
//*          following way: Higher order elements can be omitted in the date
//*          part of the expression following the solidus (in case of omitting
//*          the weeks also the "W" is omitted!) but not in the time part. The
//*          time part may be of reduced accuracy.
//*  @remark There is no requirement in ISO8601 to keep reduced accuracy
//*          compatible in the start and the end date/time; this may result
//*          "strange" or in some cases unclear (in these cases agreement is
//*          required between the communicating sides) time interval
//*          representations, however these are allowed by this template
//*
//*  @status verified
//*
//*****************************************************************************
template charstring  t_ISO8601IntervalStartEndExtended := pattern
  "{nc.yearExpansionOpt}{nc.year}{dash}({nc.month}{dash}{nc.dayOfMonth}|" &
  "{nc.dayOfYear}|{nc.week}{dash}{nc.dayOfWeek})T({nc.hour}{colon}" &
  "({nc.minute}({colon}({nc.second}{nc.sFraction}#(,1)|60)|{nc.mFraction})" &
  "#(,1)|60)|{nc.endOfDayExt}){nc.optZorTimeZoneExt}/({nc.yearExpansionOpt}" &
  "{nc.year}({dash}{nc.month}{dash}{nc.dayOfMonth}|{dash}{nc.dayOfYear}|" &
  "{dash}{nc.week}{dash}{nc.dayOfWeek})|({nc.month}{dash})#(,1)" &
  "{nc.dayOfMonth}|{nc.week}{dash}{nc.dayOfWeek})T({nc.hour}{colon}" &
  "({nc.minute}({colon}({nc.second}{nc.sFraction}#(,1)|60)|{nc.mFraction})" &
  "#(,1)|60)|{nc.endOfDayExt}){nc.optZorTimeZoneExt}"


//*****************************************************************************
//*
//*  @reference ISO_8601 $4.4.3
//*
```

```
//*  @desc Matches both complete and reduced accuracy (see note below)time
//*        interval representations identified by start and end and using the basic
//*        or the extended format
//*
//*  @remark $ 4.4.4.5 is contradictory as allows omitting higher order time
//*         elements in the part representing end date/time (the expression
//*         after the solidus), while the time part may also be of reduced
//*         accuracy (either by omitting the lower order elements or by a
//*         decimal fraction). This pattern resolves this conflict the
//*         following way: Higher order elements can be omitted in the date
//*         part of the expression following the solidus (in case of omitting
//*          the weeks also the "W" is omitted!) but not in the time part. The
//*         time part may be of reduced accuracy.
//*  @remark There is no requirement in ISO8601 to keep reduced accuracy
//*         compatible in the start and the end date/time; this may result
//*         "strange" or in some cases unclear (in these cases agreement is
//*         required between the communicating sides) time interval
//*         representations, however these are allowed by this template
//*
//*  @status verified
//*
//*****************************************************************************
template charstring  t_ISO8601IntervalStartEnd := (
  t_ISO8601IntervalStartEndBasic,
  t_ISO8601IntervalStartEndExtended
)


//*****************************************************************************
//*
//*  @reference ISO_8601 $4.4.3
//*
//*  @desc Matches both complete and reduced accuracy time interval
//*        representations identified by start and duration and using the basic
//*        format
//*
//*  @status verified
//*
//*****************************************************************************
template charstring  t_ISO8601IntervalStartDurationBasic := pattern
  "{nc.yearExpansionOpt}{nc.year}({nc.month}{nc.dayOfMonth}|{nc.dayOfYear}|" &
  "{nc.week}{nc.dayOfWeek})T({nc.hour}({nc.minute}({nc.second}{nc.sFraction}" &
  "#(,1)|60|{nc.mFraction})#(,1)|60|{nc.hFraction})#(,1){nc.optZorTimeZone}|" &
  "{nc.endOfDay}{nc.optZorTimeZone})/P({nc.year}({nc.monthDurAlt}" &
  "{nc.dayOfMonthDurAlt}|{nc.dayOfYearDurAlt})T{nc.hour}(({nc.minute}" &
  "({nc.second}{nc.sFraction}#(,1)|{nc.mFraction})#(,1)|{nc.hFraction})" &
  "#(,1))|({nc.nums}((Y({nc.nums}(M({nc.nums}(D{nc.durTime}#(,1)|" &
  "{nc.dFraction}D)|{nc.durTime}#(,1))|{nc.dFraction}[MD]|D{nc.durTime}" &
  "#(,1))|{nc.durTime}#(,1))|{nc.dFraction}[YMDW])|M({nc.nums}(D{nc.durTime}" &
  "#(,1)|{nc.dFraction}D)|{nc.dFraction}D|{nc.durTime}#(,1))|D{nc.durTime}" &
  "#(,1)|W{nc.durTime}#(,1))|{nc.durTime}))"


//*****************************************************************************
//*
//*  @reference ISO_8601 $4.4.3
//*
//*  @desc Matches both complete and reduced accuracy time interval
//*        representations identified by start and duration and using the
//*        extended format
//*
//*  @status verified
//*
//*****************************************************************************
template charstring  t_ISO8601IntervalStartDurationExtended := pattern
  "{nc.yearExpansionOpt}{nc.year}{dash}({nc.month}{dash}{nc.dayOfMonth}|" &
  "{nc.dayOfYear}|{nc.week}{dash}{nc.dayOfWeek})T({nc.hour}{colon}" &
  "({nc.minute}({colon}({nc.second}{nc.sFraction}#(,1)|60)|{nc.mFraction})" &
  "#(,1)|60){nc.optZorTimeZoneExt}|{nc.endOfDayExt}{nc.optZorTimeZoneExt})" &
  "/P({nc.year}{dash}({nc.monthDurAlt}{dash}{nc.dayOfMonthDurAlt}|" &
  "{nc.dayOfYearDurAlt})T{nc.hour}{colon}{nc.minute}({colon}{nc.second}" &
  "{nc.sFraction}#(,1)|{nc.mFraction})#(,1)|({nc.nums}((Y({nc.nums}" &
  "(M({nc.nums}(D{nc.durTime}#(,1)|{nc.dFraction}D)|{nc.durTime}#(,1))|" &
  "{nc.dFraction}[MD]|D{nc.durTime}#(,1))|{nc.durTime}#(,1))|{nc.dFraction}" &
  "[YMDW])|M({nc.nums}(D{nc.durTime}#(,1)|{nc.dFraction}D)|{nc.dFraction}D|" &
  "{nc.durTime}#(,1))|D{nc.durTime}#(,1)|W{nc.durTime}#(,1))|{nc.durTime}))"

//*****************************************************************************
```

```
//*
//*  @reference ISO_8601 $4.4.3
//*
//*  @desc Matches both complete and reduced accuracy time interval
//*        representations identified by start and duration and using either the
//*        basic or the extended formats
//*
//*  @status verified
//*
//*****************************************************************************
template charstring  t_ISO8601IntervalStartDuration := (
  t_ISO8601IntervalStartDurationBasic,
  t_ISO8601IntervalStartDurationExtended
)


//*****************************************************************************
//*
//*  @reference ISO_8601 $4.4.3
//*
//*  @desc Matches both complete and reduced accuracy time interval
//*        representations identified by duration and end and using the basic
//*        format
//*
//*  @status verified
//*
//*****************************************************************************
template charstring  t_ISO8601IntervalDurationEndBasic := pattern
  "P({nc.year}({nc.monthDurAlt}{nc.dayOfMonthDurAlt}|{nc.dayOfYearDurAlt})" &
  "T{nc.hour}(({nc.minute}({nc.second}{nc.sFraction}#(,1)|{nc.mFraction})" &
  "#(,1)|{nc.hFraction})#(,1))|({nc.nums}((Y({nc.nums}(M({nc.nums}" &
  "(D{nc.durTime}#(,1)|{nc.dFraction}D)|{nc.durTime}#(,1))|{nc.dFraction}" &
  "[MD]|D{nc.durTime}#(,1))|{nc.durTime}#(,1))|{nc.dFraction}[YMDW])|" &
  "M({nc.nums}(D{nc.durTime}#(,1)|{nc.dFraction}D)|{nc.dFraction}D|" &
  "{nc.durTime}#(,1))|D{nc.durTime}#(,1)|W{nc.durTime}#(,1))|" &
  "{nc.durTime}))/{nc.yearExpansionOpt}{nc.year}({nc.month}{nc.dayOfMonth}|" &
  "{nc.dayOfYear}|{nc.week}{nc.dayOfWeek})T({nc.hour}({nc.minute}" &
  "({nc.second}{nc.sFraction}#(,1)|60|{nc.mFraction})#(,1)|60|" &
  "{nc.hFraction})#(,1){nc.optZorTimeZone}|{nc.endOfDay}{nc.optZorTimeZone})"


//*****************************************************************************
//*
//*  @reference ISO_8601 $4.4.3
//*
//*  @desc Matches both complete and reduced accuracy time interval
//*        representations identified by duration and end and using the extended
//*        format
//*
//*  @status verified
//*
//*****************************************************************************
template charstring  t_ISO8601IntervalDurationEndExtended := pattern
  "P({nc.year}{dash}({nc.monthDurAlt}{dash}{nc.dayOfMonthDurAlt}|" &
  "{nc.dayOfYearDurAlt})T{nc.hour}{colon}{nc.minute}({colon}{nc.second}" &
  "{nc.sFraction}#(,1)|{nc.mFraction})#(,1)|({nc.nums}((Y({nc.nums}" &
  "(M({nc.nums}(D{nc.durTime}#(,1)|{nc.dFraction}D)|{nc.durTime}#(,1))|" &
  "{nc.dFraction}[MD]|D{nc.durTime}#(,1))|{nc.durTime}#(,1))|{nc.dFraction}" &
  "[YMDW])|M({nc.nums}(D{nc.durTime}#(,1)|{nc.dFraction}D)|{nc.dFraction}D|" &
  "{nc.durTime}#(,1))|D{nc.durTime}#(,1)|W{nc.durTime}#(,1))|{nc.durTime}))/" &
  "{nc.yearExpansionOpt}{nc.year}{dash}({nc.month}{dash}{nc.dayOfMonth}|" &
  "{nc.dayOfYear}|{nc.week}{dash}{nc.dayOfWeek})T({nc.hour}{colon}" &
  "({nc.minute}({colon}({nc.second}{nc.sFraction}#(,1)|60)|{nc.mFraction})" &
  "#(,1)|60){nc.optZorTimeZoneExt}|{nc.endOfDayExt}{nc.optZorTimeZoneExt})"


//*****************************************************************************
//*
//*  @reference ISO_8601 $4.4.3
//*
//*  @desc Matches both complete and reduced accuracy time interval
//*        representations identified by duration and end and using either the
//*        basic or the extended formats
//*
//*  @status verified
//*
//*****************************************************************************
template charstring  t_ISO8601IntervalDurationEnd := (
```

```
  t_ISO8601IntervalDurationEndBasic,
  t_ISO8601IntervalDurationEndExtended
)


//*****************************************************************************
//*
//*  @reference ISO_8601 $4.4.3
//*
//*  @desc Matches both complete and reduced accuracy time interval
//*        representations identified by duration and end and using either the
//*        basic or the extended formats
//*
//*  @status verified
//*
//*****************************************************************************
template charstring  t_ISO8601IntervalBasic := (
  t_ISO8601DurationBasic,
  t_ISO8601IntervalStartEndBasic,
  t_ISO8601IntervalStartDurationBasic,
  t_ISO8601IntervalDurationEndBasic
)

//*****************************************************************************
//*
//*  @reference ISO_8601 $4.4.3
//*
//*  @desc Matches both complete and reduced accuracy time interval
//*        representations identified by duration and end and using either the
//*        basic or the extended formats
//*
//*  @status verified
//*
//*****************************************************************************
template charstring  t_ISO8601IntervalExtended := (
  t_ISO8601DurationExtended,
  t_ISO8601IntervalStartEndExtended,
  t_ISO8601IntervalStartDurationExtended,
  t_ISO8601IntervalDurationEndExtended
)

//*****************************************************************************
//*
//*  @reference ISO_8601 $4.4.3
//*
//*  @desc Matches both complete and reduced accuracy time interval
//*        representations identified by duration and end and using either the
//*        basic or the extended formats
//*
//*  @status verified
//*
//*****************************************************************************
template charstring  t_ISO8601Interval := (
  t_ISO8601IntervalBasic,
  t_ISO8601IntervalExtended
)

//*****************************************************************************
//*
//*  @reference ISO_8601 $4
//*
//*  @desc Matches all ISO8601 date and time formats
//*
//*  @remark Exceptions see at the members of the value list
//*
//*  @status
//*
//*****************************************************************************
template charstring  t_ISO8601AllFormats := (
  t_ISO8601Date,
  t_ISO8601Time,
  t_ISO8601DateTime,
  t_ISO8601DurationDesign,
  t_ISO8601DurationAlternative,
  t_ISO8601Interval
)
```

```
}  // end of module
```

E.2 Constructing patterns corresponding to constrained time types

This clause describes how patterns corresponding to properties of ASN.1 time types (see clause 38.2 of Recommendation ITU-T X.680 [2]) can be constructed.

To constrain the format to the different basic natures use the templates from the module ISO8601DateTimePatterns (see clause E.1) according to table E.1.

Table E.1: Templates for basic nature properties

Basic nature property	Corresponding template definition
Date	t_ISO8601Date
Time	t_ISO8601Time
Date-Time	t_ISO8601DateTime
Interval	(t_ISO8601DurationAlternative, t_ISO8601Interval)
Rec-Interval	to be completed

Annex F (informative): Bibliography

- 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".
- ISO/IEC 6429 (1992): "Information technology - Control functions for coded character sets".
- 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).

NOTE: Freely available at http://www.oid-info.com/.

History

Document history		
V3.1.1	June 2005	Publication
V3.3.2	April 2008	Publication
V4.1.1	June 2009	Publication
V4.2.1	July 2010	Publication
V4.3.1	June 2011	Publication
V4.4.1	April 2012	Publication
V4.5.1	February 2013	Membership Approval Procedure MV 20130423: 2013-02-22 to 2013-04-23