I n t e r n a t i o n a l   T e l e c o m m u n i c a t i o n   U n i o n

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

Z.146
(03/2006)

SERIES Z: LANGUAGES AND GENERAL SOFTWARE ASPECTS FOR TELECOMMUNICATION SYSTEMS

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

Testing and Test Control Notation version 3 (TTCN-3): Using ASN.1 with TTCN-3

ITU-T Recommendation Z.146

ITU-T Z-SERIES RECOMMENDATIONS

LANGUAGES AND GENERAL SOFTWARE ASPECTS FOR TELECOMMUNICATION SYSTEMS

FORMAL DESCRIPTION TECHNIQUES (FDT)	
Specification and Description Language (SDL)	Z.100–Z.109
Application of formal description techniques	Z.110–Z.119
Message Sequence Chart (MSC)	Z.120–Z.129
Extended Object Definition Language (eODL)	Z.130–Z.139
Testing and Test Control Notation (TTCN)	**Z.140–Z.149**
User Requirements Notation (URN)	Z.150–Z.159
PROGRAMMING LANGUAGES	
CHILL: The ITU-T high level language	Z.200–Z.209
MAN-MACHINE LANGUAGE	
General principles	Z.300–Z.309
Basic syntax and dialogue procedures	Z.310–Z.319
Extended MML for visual display terminals	Z.320–Z.329
Specification of the man-machine interface	Z.330–Z.349
Data-oriented human-machine interfaces	Z.350–Z.359
Human-machine interfaces for the management of telecommunications networks	Z.360–Z.379
QUALITY	
Quality of telecommunication software	Z.400–Z.409
Quality aspects of protocol-related Recommendations	Z.450–Z.459
METHODS	
Methods for validation and testing	Z.500–Z.519
MIDDLEWARE	
Distributed processing environment	Z.600–Z.609

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

ITU-T Recommendation Z.146

Testing and Test Control Notation version 3 (TTCN-3): Using ASN.1 with TTCN-3

Summary

This Recommendation defines the way of using ASN.1 as defined in ITU-T Recs X.680, X.681, X.682 and X.683 with TTCN-3 (*Testing and Test Control Notation 3*) as defined in ITU-T Rec. Z.140.

Content of this Recommendation has been moved into this separate Recommendation from ITU-T Rec. Z.140 (04/2003). Since then, corrections and editorial changes have been made and included in this Recommendation.

Source

ITU-T Recommendation Z.146 was approved on 16 March 2006 by ITU-T Study Group 17 (2005-2008) under the ITU-T Recommendation A.8 procedure.

CONTENTS

Page

1 Scope 1

2 References 1

3 Definitions and abbreviations 1
- 3.1 Definitions 1
- 3.2 Abbreviations 1

4 Introduction 2

5 General 2

6 Amendments in the core language 2

7 Additional TTCN-3 types 5
- 7.1 General 5
- 7.2 Additional simple basic types and values 5
- 7.3 Sub-typing of additional types 6

8 ASN.1 and TTCN-3 type equivalents 6
- 8.1 General 6
- 8.2 Identifiers 7

9 ASN.1 data types and values 7
- 9.1 General 7
- 9.2 Scope of ASN.1 identifiers 12

10 Parameterization in ASN.1 12

11 Defining ASN.1 message templates 13
- 11.1 General 13
- 11.2 ASN.1 receive messages using the TTCN-3 template syntax 13
- 11.3 Ordering of template fields 14

12 Encoding information 14
- 12.1 General 14
- 12.2 ASN.1 encoding attributes 14
- 12.3 ASN.1 variant attributes 15

Annex A – Additional BNF and static semantics 16
- A.1 ASN.1 support 16

Annex B – Pre-defined TTCN-3 functions 16
- B.1 The Decompose function 16

Annex C – Predefined object identifier components 17

BIBLIOGRAPHY 18

ITU-T Recommendation Z.146

Testing and Test Control Notation version 3 (TTCN-3): Using ASN.1 with TTCN-3

1 Scope

This Recommendation defines a normative way of using of ASN.1 as defined in ITU-T Recs 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 this Recommendation.

2 References

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

[1] ITU-T Recommendation Z.140 (2006), *Testing and Test Control Notation version 3 (TTCN-3): Core language*.

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

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

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

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

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

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

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

[9] Void.

[10] ITU-T Recommendation T.100 (1988), *International information exchange for interactive Videotex*.

[11] ITU-T Recommendation T.101 (1994), *International interworking for Videotex services*.

[12] ITU-T Recommendation X.660 (2004), *Information technology – Open Systems Interconnection – Procedures for the operation of OSI registration authorities: General procedures and top arcs of the ASN.1 Object Identifier tree*.

3 Definitions and abbreviations

3.1 Definitions

For the purposes of this Recommendation, the terms and definitions given in ITU-T Rec. Z.140 [1] apply.

3.2 Abbreviations

For the purposes of this Recommendation, the abbreviations given in ITU-T Rec. Z.140 [1] and the following apply:

ASN.1	Abstract Syntax Notation One

4 Introduction

When using ASN.1 with TTCN-3 all features of TTCN-3 and statements given in clause 4/Z.140 [1] still apply. In addition, when supporting this Recommendation, TTCN-3 becomes fully harmonized with ASN.1 which may be used with TTCN-3 modules as an alternative data type and value syntax. This Recommendation defines the use of ASN.1 in TTCN-3 modules. The approach used to combine ASN.1 and TTCN-3 could be applied to support the use of other type and value systems with TTCN-3. However, the details of this are not defined in this Recommendation.

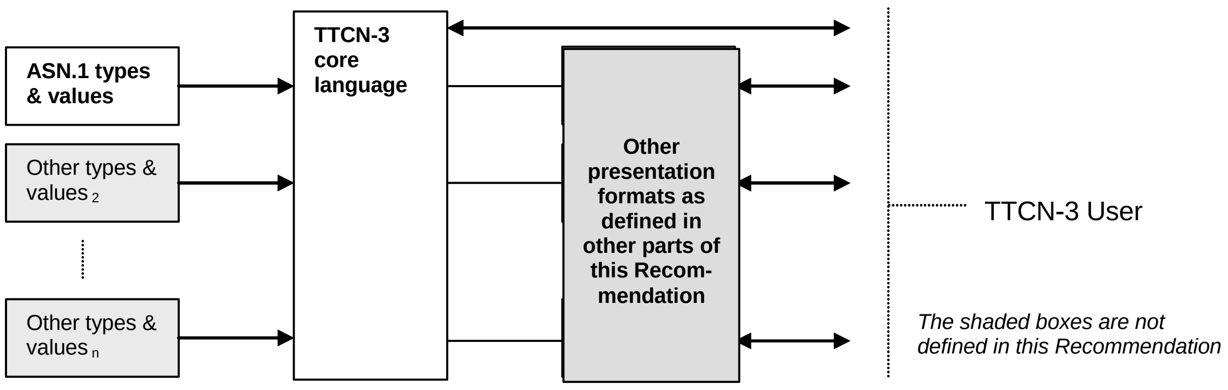
ASN.1 types & values
Other types & values 2
Other types & values n
TTCN-3 core language
Other presentation formats as defined in other parts of this Recom-mendation
TTCN-3 User
The shaded boxes are not defined in this Recommendation

Figure 1/Z.146 – User's view of the core language and the various presentation formats

5 General

TTCN-3 provides a clean interface for using ASN.1 version 2002 (as defined in ITU-T Recs X.680 [2], X.681 [3], X.682 [4] and X.683 [5]) in TTCN-3 modules. XML related capabilities of ASN.1 are currently not considered. When importing from an ASN.1 module one of the following language identifier strings shall be used:

- "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 Blue Book version of ASN.1.

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

NOTE 2 – When "ASN.1:1988" is supported, the ASN.1 items shall be imported from such modules according to the syntactical and semantical rules of ITU-T Rec. X.208 (*Blue Book*)

When ASN.1 is used with TTCN-3, the keywords listed in 11.18/X.680 [2] shall not be used as identifiers in a TTCN-3 module. ASN.1 keywords shall follow the requirements of ITU-T Rec. X.680 [2].

6 Amendments in the core language

When using ASN.1 definitions in TTCN-3 the amendments given in this clause to the TTCN-3 core language as defined in ITU-T Rec. Z.140 [1] shall also apply. **Clause numbers below refer to those in ITU-T Rec. Z.140 [1]**.

Clause 3.1 Definitions

Change the definition "known types": with the following one:

known types: set of defined types, imported ASN.1 types and other imported external types.

Insert the following note after the definition "root type":

NOTE – In case of types based on imported ASN.1 types, the root type is determined from the associated TTCN-3 type (see clause 8).

Clause 3.2 Abbreviations

Add the following abbreviation:

ASN.1	Abstract Syntax Notation One

Clause 7.1 Naming of modules

Append to the end of the 1st sentence of the 1st paragraph:

optionally followed by an object identifier.

Add the following new note after NOTE 1:

NOTE 2 – Module names 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 identifiers (see 7.5.8) is unable to resolve such kind of clashes.

Clause 7.5.0 General

Add the following new paragraph after the 2nd paragraph:

If the object identifier is provided as part of the module name (from which the definitions are imported) in the import statement, this object identifier shall be used to identify the correct module.

Clause 7.5.8

Add the following note after the 1st paragraph:

NOTE 1 – The rule to resolve name clashes within object identifier values is given in 6.1.0 item d).

Clause 14.3 Template matching mechanisms

Append **objid** to the list of types in the note to the 3rd paragraph:

NOTE – The following types may be omitted: ...

Clause 15.3 Relational operators

Add **objid** to the list of types in the 1st paragraph for which relational operators are allowed (All other relational operators shall have only operands of type **integer** (including derivatives of **integer**), **float** (including derivations of **float**)<u>, **objid**</u> ...)

Add a new paragraph after the 2nd paragraph:

Two **objid** values are equal, if they have equal number of components and the numerical 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:

- comparison shall start by comparing the first components of the objid values;
- if the numerical value of the compared component in the first objid value is less than the numerical value of the corresponding component in the second objid value, the first objid value is less than the second;
- if numerical values of the two compared components equal, comparison continues with the next pair of components of the two objid values; the objid value in which the first component with a smaller numerical value is found, is less than the other objid value;
- if all compared pairs of components of the two objid values equal and one of the objid values has further components while the another does not have a non-compared component, 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))
```

Clause 16.1.3 Predefined functions

Add the row below to the section "**Other functions**" of Table 10 (List of TTCN-3 predefined functions):

Other functions		
	Decompose an objid value	**decomp**

Clause A.1.5 TTCN-3 terminals

Add **objid** to Table A.2 (List of TTCN-3 special terminal symbols).

Clause A.1.6.0 TTCN-3 module

Append '`[DefinitiveIdentifier]`' to the end of the production TTCN3ModuleId.

NOTE – It becomes if no other changes by other parts of the standard applied:

```
'3. TTCN3ModuleId ::= ModuleIdentifier [DefinitiveIdentifier]'
```

Clause A.1.6.1.8 Import definitions

Append '`[Dot ObjectIdentifierValue]`' to the end of the production GlobalModuleId.

NOTE – It becomes if no other changes by other parts of the standard applied:

```
'223. GlobalModuleId ::= ModuleIdentifier [Dot ObjectIdentifierValue]'
```

Clause A.1.6.3 Type

Append '`| ObjectIdentifierKeyword`' to the end of the production PredefinedType.

NOTE – It becomes if no other changes by other parts of the standard applied:

```
'410. PredefinedType ::= BitStringKeyword |
                         BooleanKeyword |
                         CharStringKeyword |
                         UniversalCharString |
                         IntegerKeyword |
                         OctetStringKeyword |
                         HexStringKeyword |
                         VerdictTypeKeyword |
                         FloatKeyword |
                         AddressKeyword |
                         DefaultKeyword |
                         AnyTypeKeyword |
                         ObjectIdentifierKeyword'
```

Clause A.1.6.4 Value

Append '`| ObjectIdentifierValue`' to the end of the production PredefinedValue.

NOTE – It becomes if no other changes by other parts of the standard applied:

```
'434. PredefinedValue ::= BitStringValue |
                          BooleanValue |
                          CharStringValue |
                          IntegerValue |
                          OctetStringValue |
                          HexStringValue |
                          VerdictTypeValue |
                          EnumeratedValue |
                          FloatValue |
                          AddressValue |
                          OmitValue |
                          ObjectIdentifierValue'
```

Clause C.14 Number of elements in a structured value

Append **`objid`** to the list of types in the 1st sentence of the first paragraph (i.e., 'This function returns the actual number of elements of a module parameter, constant, variable or **`template`** of a **`record`**, **`record of`**, **`set`**, **`set of`** or **`objid`** type ...').

Add the following new paragraph after the 1st paragraph:

In the case of **`objid`** values, templates or arrays, the actual value to be returned is the sequential number of the last component in the objid value.

Add the following new example to the end of the EXAMPLE section:

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

7 Additional TTCN-3 types

7.1 General

Additional TTCN-3 types to support the use of ASN.1 are summarized in Table 1.

Table 1/Z.146 – Overview of TTCN-3 types

Class of type	Keyword	Sub-type
Simple basic types	**objid**	list

7.2 Additional simple basic types and values

To support the use of ASN.1 in TTCN-3, the following simple types and values shall be supported in addition to those specified in clause 6.1 of the core language Recommendation (ITU-T Rec. Z.140 [1]):

a) **`objid`**: a type whose distinguished values are the set of:

– all object identifier values conforming to Annex A/X.660 [12]; and

– all syntactically correct object identifier values outside the collection of values defined in ITU-T Rec. X.660 [12] (e.g., with a node beneath the root not defined in ITU-T Rec. X.660 [12]).

The value notations for the objid type shall conform to the rules given in clause 31/X.680 [2] with the exception that hyphens in object identifiers are replaced with underscores.

NOTE 1 – The name form of object identifier components shall be used only for components defined in ITU-T Rec. X.660 [12]. These predefined object identifier components are given in Annex C. In case of any conflict between ITU-T Rec. X.660 [12] and Annex C, the former shall take precedence.

In cases when the identifier of a value referenced within an object identifier value notation is identical to any of the predefined component names (i.e., independently of the position of the predefined component or the referenced value inside the notation), the name of the referenced value shall be prefixed with the name of the module in which it is defined (see definition of TTCN-3 modules in clause 7/Z.140 [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 7.5.8/Z.140 [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
```

7.3 Sub-typing of additional types

7.3.1 General

User-defined types shall be denoted by the keyword **`type`**. With user-defined types it is possible to create sub-types (such as lists) on types according to Table 1.

7.3.2 Lists of values

TTCN-3 permits the specification of a list of distinguished values of the **`objid`** type. Sub-typing shall be applied as specified in 6.2.1/Z.140 [1].

EXAMPLE:

```
type objid MyListOfObjids (objid{0 4 0 0 1}, objid{0 4 0 1 1});
```

8 ASN.1 and TTCN-3 type equivalents

8.1 General

The ASN.1 types listed in Table 2 are considered to be equivalent to their TTCN-3 counterparts.

Table 2/Z.146 – List of ASN.1 and TTCN-3 equivalents

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

All TTCN-3 operators, functions, matching mechanisms, value notation, etc., that can be used with a TTCN-3 type given in Table 2 may also be used with the corresponding ASN.1 type.

8.2 Identifiers

In converting ASN.1 identifiers to TTCN-3 identifiers, any hyphen "-" symbols shall be changed to an underscore "_".

EXAMPLE:

```
MyASN1module DEFINITIONS ::=
BEGIN
    Missleading-ASN1-Name::=    INTEGER     -- ASN.1 type identifier using '-'

END


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

    const Missleading_ASN1_Name ExampleConst:= 1;       // TTCN-3 reference to ASN.1 type
                                                        // using underscores
}
```

9 ASN.1 data types and values

9.1 General

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 9.3/X.680 and 9.4/X.680 [2] within the ASN.1 module(s).

EXAMPLE 1:

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


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

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

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

The ASN.1 module shall conform to the syntax of ITU-T Recs X.680 [2], X.681 [3], X.682 [4] and X.683 [5]. Once declared, ASN.1 types and values may be used within TTCN-3 modules in a similar way that ordinary TTCN-3 types and values from other TTCN-3 modules are used (i.e., the required definitions shall be imported). When importing ASN.1 items into a TTCN-3 module, an associated type or value is produced for each ASN.1 item imported. All TTCN-3 definitions or assignments based on imported ASN.1 items shall be done according to the rules imposed by the related associated type or value. Also, the matching mechanism shall use the associated type when matching constants, variables, templates or in-line expressions based on ASN.1 declarations.

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

1) Ignore any extension markers and exception specifications.
2) Ignore any user-defined constraints (see clause 9/X.682 [4]).
3) Ignore any contents constraint (see clause 9/X.682 [4]).
4) Ignore any pattern constraint (see 48.9/X.680 [2]).
5) Create equivalent TTCN-3 subtypes for all ASN.1 types constrained using contained subtyping by replacing included types by the set of values they represent. More detailed information on the conversion of ASN.1 type constraints to TTCN-3 subtypes is given in Table 3. Table 3 shows the applicability of ASN.1 type constraint mechanisms to different ASN.1 types. Where the cell contains "No", the type constraint is disallowed for the given type. Shaded cells identify type constraints applicable to a given type and text in the cell defines TTCN-3 subtyping mechanisms to be used when transforming constrained ASN.1 types.
6) Execute the COMPONENTS OF transformation according to 24.4/X.680 [2] on any SEQUENCE types and according to 26.2/X.680 [2] on any SET types containing the keywords "COMPONENTS OF".
7) Replace any EMBEDDED PDV type with its associated type obtained by expanding inner subtyping in the associated type of the EMBEDDED PDV type (see 33.5/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 34.5/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 40.5/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 C.7/X.681 [3]) and replace all ASN.1 types with their TTCN-3 equivalents according to Table 2. The resulted type is the TTCN-3 associated type.
11) Ignore any remaining inner subtyping (see Note 4).
12) Ignore any named numbers and named bits in ASN.1 types. In ASN.1 values, replace any named number by its value and substitute any named bits or sequence of named bits by a bitstring without trailing zeros, where bit positions identified by names present are replaced by "1"s, other bit positions are replaced by "0"s.
13) Replace any selection type with the type referenced by the selection type; if the denoted choice type (the "Type" in 29.1/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 37.2/X.680 [2];
    – PrintableString: **`charstring`** constrained to the set of characters as given in 37.4/X.680 [2];
    – TeletexString and T61String: **`universal charstring`** constrained to the set of characters as given in ITU-T Rec. T.61 (see Bibliography);
    – VideotexString: **`universal charstring`** constrained to the set of characters as given in ITU-T Recs T.100 [10] and T.101 [11];
    – GraphicString: **`universal charstring`**;
    – GeneralString: **`universal charstring`**.
16) Replace any GeneralizedTime and UTCTime types or values with the type or value of **`charstring`**.
17) Replace any ObjectDescriptor type or value by the **`universal charstring`** type or value.
18) Replace any notations for the object class field types (see 14/X.681 [3]) by the ASN.1 item 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 15/X.681 [3]) by the ASN.1 item they are referencing.

20) Revert table constraints (see clause 10/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:

    – **`type enumerated`** <*identifier*> { NULL }, where <*identifier*> is the ASN.1 Type reference converted according to 7.2.

22) Replace all references to open types with the metatype "OPEN TYPE".

23) Replace ASN.1 types with their equivalents according to Table 2 and ASN.1 values with equivalent TTCN-3 values based on the associated types. 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). The metatype "OPEN TYPE" has to be replaced by **`anytype`**.

NOTE 1 – Associated types alone do not contain all information needed for the correct encoding of values based on ASN.1 types. The way of handling the extra information needed by the system to provide correct encoding is implementation dependent and remains hidden for the user; its 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 are also imported.

NOTE 3 – The data-value field of the EXTERNAL type may be encoded as a single-ASN.1-type, octet-aligned or arbitrary (see 8.18.1/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 shall use the appropriate encoding attribute for the type or the given constant, variable, template or template field (see 11.3).

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

NOTE 5 – Equivalence with the **`objid`** type is limited to the syntax to be used for value notations only. When encoding/decoding an **objid** value retrieved from an ASN.1 RELATIVE-OID value using an ASN.1 encoding rule, the encoding/decoding shall 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 shall be taken into account by the user when declaring values and templates (also may be handled by tools implicitly).

NOTE 8 – This replacement do not effect constraints applied to the "notation for the object class field type" itself.

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

EXAMPLE 2:

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

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

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

Table 3/Z.146 – ASN.1 type constraint to TTCN-3 subtype conversions

Type (or derived from such a type by tagging or subtyping)	Single value	Contained subtype[h)]	Value range	Size constraint	Permitted alphabet	Type constraint	Inner subtyping[i)]	Pattern constraint	User-defined constraint	Table constraint[k)]	Relation constraint[k)]	Content constraint
Bit String	list	single value:list, size: length	No	length	No	No	No	No	ignore	No	No	ignore
Boolean	list	list	No	No	No	No	No	No	ignore	No	No	No
Choice	list	list	No	No	No	No	convert to full type	No	ignore	No	No	No
Embedded-pdv[a)]	list	No	No	No	No	No	convert to full type	No	ignore	No	No	No
Enumerated	list	list	No	No	No	No	No	No	ignore	No	No	No
External[a)]	list	No	No	No	No	No	convert to full type	No	ignore	No	No	No
Instance-of [a), b)]	list	list	No	No	No	No	convert to full type	No	ignore	No	No	No
Integer	list	single value:list, value range: range	range	No	No	No	No	No	ignore	No	No	No
Null	ignore	ignore	No	No	No	No	No	No	ignore	No	No	No
Object class field type	[c)]	[c)]	No	No	No	No	No	No	ignore	list	ignore	No
Object Descriptor[e)]	list	single value: list, size: length, perm.alphabet: range	No	length	range	No	No	No	ignore	No	No	No
Object Identifier	list	list	No	No	No	No	No	No	ignore	No	No	No
Octet String	list	single value:list, size: length	No	length	No	No	No	No	ignore	No	No	ignore
open type[f)]	No	No	No	No	No	anytype with list constraint	No	No	ignore	No[m)]	No[m)]	No
Real	list	single value:list, value range: range	range	No	No	No	convert to full type	No	ignore	No	No	No
Relative Object Identifier[d)]	list	list	No	No	No	No	No	No	ignore	No	No	No

Type (or derived from such a type by tagging or subtyping)	Single value	Contained subtype[h)]	Value range	Size constraint	Permitted alphabet	Type constraint	Inner subtyping[i)]	Pattern constraint	User-defined constraint	Table constraint[k)]	Relation constraint[k)]	Content constraint
Restricted Character String Types	list	single value:list, size: length, perm.alphab.: range	range	length	range	No	No	Ignore[g)]	ignore	No	No	No
Sequence	list	list	No	No	No	No	convert to full type	No	ignore	No	No	No
Sequence-of	list	single value:list, value range: range	No	length	No	No	convert to full type	No	ignore	No	No	No
Set	list	list	No	No	No	No	convert to full type	No	ignore	No	No	No
Set-of	list	single value:list, value range: range	No	length	No	No	convert to full type	No	ignore	No	No	No
Time Types[a)]	list	list	No	No	No	No	No	No	ignore	No	No	No
Unrestricted Character String Type[a)]	list	No	No	length (applied to field "string-value")	No	No	convert to full type	No	ignore	No	No	No

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/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 items. 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.

k) Applicable to notations for the object class field type only.

m) Applicable when the open type is defined using the notation for the object class field type (see above).

9.2 Scope of ASN.1 identifiers

Imported ASN.1 identifiers follow the same scope rules as imported TTCN-3 types and values (see 5.3/Z.140 [1]).

10 Parameterization in ASN.1

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

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

a) Information Object classes;

b) Information Objects;

c) Information Object Sets.

For example the following is not legal because it defines a TTCN-3 type which takes an ASN.1 object set as an actual parameter.

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

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

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

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

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

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

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

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

To make this a legal definition, the extra ASN.1 type My Message1 has to be defined as shown below. This resolves the information object set parameterization and can therefore be directly used in the TTCN-3 module.

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

    …

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

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

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

11 Defining ASN.1 message templates

11.1 General

If messages are defined in ASN.1 using, for example, `SEQUENCE` (or possibly `SET`), then actual messages, for both **`send`** and **`receive`** events, can be specified using the ASN.1 value syntax.

EXAMPLE:

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

    -- The message definition
    MyMessageType::= SEQUENCE
    {   field1  [1] IA5STRING,          // Like TTCN-3 character string
        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",
        field2      123,
        field3      {field31 '11011'B, field32 456789, field33 'FF'O},
        field4      {true, false}
    }
END
```

11.2 ASN.1 receive messages using the TTCN-3 template syntax

Matching mechanisms are not supported in the standard ASN.1 syntax. Thus, if it is wished to use matching mechanisms with an ASN.1 receive message, then the TTCN-3 syntax for receive templates shall be used instead. Note that this syntax includes component references in order to be able to reference the individual components in ASN.1 `SEQUENCE`, `SET`, etc.

EXAMPLE:

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

// a message template using matching mechanisms within TTCN-3 might be
template myMessageType  MyValue:=
{
    field1 :=           "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 := "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 according to the following syntax:

EXAMPLE:

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

12.2 ASN.1 encoding attributes

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

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

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

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

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

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

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

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

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

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

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

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

12.3 ASN.1 variant attributes

The following strings are predefined (standardized) variant attributes. They have predefined meaning when applied jointly with predefined ASN.1 encoding attributes only (see 11.2). 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 this Recommendation (see Note 1):

a) "length form 1" means that the given value shall only be encoded and decoded using the short form of the length octets (see 8.1.3/X.690 [6]) in case of BER, CER and DER encodings or the single octet length determinant (see 10.9/X.691 [7]) in case of any form of the PER encoding (see Note 2).

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

c) "length form 3" means that the given value shall only be encoded and decoded using the indefinite form of the length octets (see 8.1.3/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" means, that the "encoding" field of the value, which shall be based on an ASN.1 EXTERNAL type, shall be encoded using the "encoding" choice specified by the variant attribute or matched at receipt only if it has been encoded by the sender using the specified choice (see 8.18/X.690 [6]). This attribute shall be used only with imported ASN.1 EXTERNAL types and constants, variables, templates or template fields based on such types. When used with any kind of a grouping (e.g., TTCN-3 groups, import of a kind of definitions, import all statements etc.) it shall have effect on TTCN-3 objects based on ASN.1 EXTERNAL types only. If the conditions set in clauses 8.18.6 to 8.18.8/X.690 [6] and the specified attribute do not met, this shall cause a run-time error.

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

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

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

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

NOTE 1 – These attributes may be re-used 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.

NOTE 2 – Application of these variant attributes may lead to invalid ASN.1 encoding (e.g., using the indefinite length form to primitive values in BER or not using the minimum necessary number of length octets). This is allowed intentionally and users shall allocate these variant attributes to constants, variables, templates or template fields used for receiving cautiously.

Annex A

Additional BNF and static semantics

To support the use of ASN.1 in TTCN-3, the TTCN-3 syntax in Annex A/Z.140 [1] shall be supplemented by the BNF and semantic rules specified in this annex.

A.1 ASN.1 support

```
xxx. DefinitiveIdentifier ::= Dot ObjectIdentifierKeyword "{" DefinitiveObjIdComponentList "}"
xxx. ObjectIdentifierKeyword ::= "objid"
xxx. DefinitiveObjIdComponentList ::= {DefinitiveObjIdComponent}+
xxx. DefinitiveObjIdComponent ::= NameForm |
                                  DefinitiveNumberForm |
                                  DefinitiveNameAndNumberForm
xxx. DefinitiveNumberForm ::= Number
xxx. DefinitiveNameAndNumberForm ::= Identifier "(" DefinitiveNumberForm ")"
xxx. ObjectIdentifierValue ::= ObjectIdentifierKeyword "{" ObjIdComponentList "}"
xxx. ObjIdComponentList ::= {ObjIdComponent}+
xxx. ObjIdComponent ::= NameForm |
                        NumberForm |
                        NameAndNumberForm |
                        ReferencedValue
/* STATIC SEMANTICS – ReferencedValue shall be an object identifier value */
xxx. NameForm ::= Identifier
xxx. NumberForm ::= Number | ReferencedValue
/* STATIC SEMANTICS - ReferencedValue shall be a non negative integer value */
xxx. NameAndNumberForm ::= Identifier "(" NumberForm ")"
```

Annex B

Predefined TTCN-3 functions

B.1 The Decompose function

```
decomp (in objid invalue, in integer index, in integer count) return objid
```

This function returns an object identifier value containing a fragment (sequence of components) of the input object identifier value (invalue). The starting point of the fragment is defined by the second in parameter (index). The actual value of the 'index' parameter shall be a non-negative integer. Indexing starts from zero, which identifies the first component of the input object identifier value. The third input parameter (count) defines the number of components in the returned objid value. The actual value of the 'count' parameter shall be a positive non-zero integer. The sum of the 'index' and 'count' parameters shall be less than or equal to the number of components in the input objid value minus 1.

EXAMPLE:

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

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

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

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

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

Annex C

Predefined object identifier components

ITU-T Rec. X.660 [12] defines the tree of object identifier components shown below. Only the object identifier components defined in ITU-T Rec. X.660 [12] shall use the name form (without defining the numerical value of the component) in object identifier value notations. These predefined components have specified numerical values when used at their predefined positions only. Names in *italic* are reserved for historical reasons; therefore, their use in TTCN-3 codes is deprecated, but it is recommended that TTCN-3 tools are able to recognize them and substitute with the correct numerical value.

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

`itu_t(0)`, *ccitt(0)*, `itu_r(0)`

```
    recommendation(0)

        a(1)
        d(4)
        e(5)
        f(6)
        g(7)
        h(8)
        i(9)
        j(10)
        k(11)
        l(12)
        m(13)
        n(14)
        o(15)
        p(16)
        q(17)
        r(18)
        s(19)
        t(20)
        u(21)
        v(22)
        x(24)
        y(25)
        z(26)

    question(1)
    administration(2)
    network_operator(3)
    identified_organization(4)
    r_recommendation(5)

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

`joint_iso_itu_t(2)`, *joint_iso_ccitt(2)*

BIBLIOGRAPHY

- ISO/IEC 6429:1992, *Information technology – Control functions for coded character sets*.
- ITU-T Recommendation T.50 (1992), *International Reference Alphabet (IRA) (Formerly International Alphabet No. 5 or IA5) – Information technology – 7-bit coded character set for information interchange*.
- ITU-T Recommendation X.208, *Specification of Abstract Syntax Notation One (ASN.1)*.
- ISO/IEC 8859-1:1998, *Information technology – 8-bit single-byte coded graphic character sets – Part 1: Latin alphabet No. 1*.
- ITU-T Recommendation T.61 (1993), *Character repertoire and coded character sets for the international teletex service*.
- A repository of object identifiers (OIDs) is freely available at http://oid.elibel.tm.fr.

NOTE – The content of the above website is planned to be moved to the ITU-T website.

SERIES OF ITU-T RECOMMENDATIONS

Series A	Organization of the work of ITU-T
Series D	General tariff principles
Series E	Overall network operation, telephone service, service operation and human factors
Series F	Non-telephone telecommunication services
Series G	Transmission systems and media, digital systems and networks
Series H	Audiovisual and multimedia systems
Series I	Integrated services digital network
Series J	Cable networks and transmission of television, sound programme and other multimedia signals
Series K	Protection against interference
Series L	Construction, installation and protection of cables and other elements of outside plant
Series M	Telecommunication management, including TMN and network maintenance
Series N	Maintenance: international sound programme and television transmission circuits
Series O	Specifications of measuring equipment
Series P	Telephone transmission quality, telephone installations, local line networks
Series Q	Switching and signalling
Series R	Telegraph transmission
Series S	Telegraph services terminal equipment
Series T	Terminals for telematic services
Series U	Telegraph switching
Series V	Data communication over the telephone network
Series X	Data networks, open system communications and security
Series Y	Global information infrastructure, Internet protocol aspects and next-generation networks
Series Z	**Languages and general software aspects for telecommunication systems**

Printed in Switzerland
Geneva, 2006