International Telecommunication Union

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

Z.105

(12/2011)

SERIES Z: LANGUAGES AND GENERAL SOFTWARE ASPECTS FOR TELECOMMUNICATION SYSTEMS

Formal description techniques (FDT) – Specification and Description Language (SDL)

Specification and Description Language – SDL-2010 combined with ASN.1 modules

Recommendation ITU-T Z.105

International Telecommunication Union

ITU-T Z-SERIES RECOMMENDATIONS

LANGUAGES AND GENERAL SOFTWARE ASPECTS FOR TELECOMMUNICATION SYSTEMS

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

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

Recommendation ITU-T Z.105

Specification and Description Language – SDL-2010 combined with ASN.1 modules

Summary

Recommendation ITU-T Z.105 defines how Abstract Syntax Notation One (ASN.1) modules are usable in combination with Specification and Description Language 2010 (SDL-2010). This text replaces Recommendation ITU-T Z.105 (2003) to align with Recommendations ITU-T Z.100, ITU-T Z.101, ITU-T Z.102, ITU-T Z.103, ITU-T Z.104 and ITU-T Z.106 for SDL-2010. Recommendation ITU-T Z.105 (2003) replaced the semantic mappings from ASN.1 to SDL-2000 defined in Recommendation ITU-T Z.105 (1999). The use of ASN.1 notation embedded in the Specification and Description Language previously defined in Recommendation ITU-T Z.107 (1999) is not defined by this Recommendation.

The main area of application of this Recommendation is the specification of telecommunication systems. The combined use of SDL-2010 and ASN.1 permits a coherent way to specify the structure and behaviour of telecommunication systems, together with data, messages and encoding of messages that these systems use.

This version of Recommendation ITU-T Z.105 include necessary alignments with ASN.1:2002 Recommendations, mapping of XML values, improved mapping of bit string values and mapping of relevant ASN.1 constructs for extensions.

History

Edition	Recommendation	Approval	Study Group
1.0	ITU-T Z.105	1995-03-06	10
2.0	ITU-T Z.105	1999-11-19	10
3.0	ITU-T Z.105	2001-10-29	17
4.0	ITU-T Z.105	2003-07-07	17
5.0	ITU-T Z.105	2011-12-22	17

FOREWORD

The International Telecommunication Union (ITU) is the United Nations specialized agency in the field of telecommunications, information and communication technologies (ICTs). The ITU Telecommunication Standardization Sector (ITU-T) is a permanent organ of ITU. ITU-T is responsible for studying technical, operating and tariff questions and issuing Recommendations on them with a view to standardizing telecommunications on a worldwide basis.

The World Telecommunication Standardization Assembly (WTSA), which meets every four years, establishes the topics for study by the ITU-T study groups which, in turn, produce Recommendations on these topics.

The approval of ITU-T Recommendations is covered by the procedure laid down in WTSA Resolution 1.

In some areas of information technology which fall within ITU-T's purview, the necessary standards are prepared on a collaborative basis with ISO and IEC.

NOTE

In this Recommendation, the expression "Administration" is used for conciseness to indicate both a telecommunication administration and a recognized operating agency.

Compliance with this Recommendation is voluntary. However, the Recommendation may contain certain mandatory provisions (to ensure, e.g., interoperability or applicability) and compliance with the Recommendation is achieved when all of these mandatory provisions are met. The words "shall" or some other obligatory language such as "must" and the negative equivalents are used to express requirements. The use of such words does not suggest that compliance with the Recommendation is required of any party.

INTELLECTUAL PROPERTY RIGHTS

ITU draws attention to the possibility that the practice or implementation of this Recommendation may involve the use of a claimed Intellectual Property Right. ITU takes no position concerning the evidence, validity or applicability of claimed Intellectual Property Rights, whether asserted by ITU members or others outside of the Recommendation development process.

As of the date of approval of this Recommendation, ITU had not received notice of intellectual property, protected by patents, which may be required to implement this Recommendation. However, implementers are cautioned that this may not represent the latest information and are therefore strongly urged to consult the TSB patent database at http://www.itu.int/ITU-T/ipr/.

Table of Contents

Introduction

Objective

This Recommendation defines how Abstract Syntax Notation One (ASN.1) modules are usable in combination with Specification and Description Language 2010 (SDL-2010). The intention is that the structure and the behaviour of systems are described with SDL-2010, while parameters of exchanged messages are described with ASN.1. This Recommendation defines a mapping of ASN.1 constructs to already existing SDL-2010 constructs and contains only a small extension to Recommendations ITU-T Z.101, ITU-T Z.102, ITU-T Z.103 and ITU-T Z.104 to allow ASN.1 modules to be used.

Coverage

This Recommendation presents a semantic definition for the combination of SDL-2010 and ASN.1 modules. A mapping of the ASN.1 data defined in a module to the corresponding SDL-2010 constructs defined in Recommendations ITU-T Z.101, ITU-T Z.102, ITU-T Z.103 and ITU-T Z.104 is given, including the operators that applicable to the ASN.1 data. The ASN.1 data items are then usable within SDL-2010 (using SDL-2010 notation).

Application

The main area of application of this Recommendation is the specification of telecommunication systems. The combined use of SDL-2010 and ASN.1 permits a coherent way to specify the structure and behaviour of telecommunication systems, together with data, messages and encoding of messages that these systems use.

NOTE – "Specification" in this Recommendation includes definition of requirements in a standard, Recommendation, or procurement document, and description of an implementation.

A specification conforms to this Recommendation if and only if it conforms to the syntactic and semantic grammar rules for the formal technical language defined by the Recommendation (which includes the referenced ASN.1 and SDL-2010 languages). Conformance implies that every possibly dynamic interpretation of the specification conforms to the language rules. A specification that uses extensions of the language does not conform.

A tool does not fully support the language if it rejects some constructs of the language or that has a static or dynamic interpretation of a specification in the language that does not conform to language semantics.

Status/Stability

This text replaces Recommendation ITU-T Z.105 (2003) to align with Recommendations ITU-T Z.100, ITU-T Z.101, ITU-T Z.102, ITU-T Z.103, ITU-T Z.104 and ITU-T Z.106 for SDL-2010. Recommendation ITU-T Z.105 (2003) defined the semantic mappings from ASN.1 to SDL-2000 defined in Recommendation ITU-T Z.105 (1999) and aligned with ASN.1:2002 Recommendations. The use of ASN.1 notation embedded in the Specification and Description Language previously defined in the withdrawn Recommendation ITU-T Z.107 (1999) is not defined by this Recommendation.

It is likely that changes to Recommendations ITU-T X.680, ITU-T X.681, ITU-T X.682 and ITU-T X.683, ITU-T Z.100, ITU-T Z.101, ITU-T Z.102, ITU-T Z.103, ITU-T Z.104 or ITU-T Z.106 will require modifications to this Recommendation.

This Recommendation is the complete reference manual describing the combination of SDL-2010 and ASN.1 modules.

Recommendation ITU-T Z.105

Specification and Description Language – SDL-2010 combined with ASN.1 modules

1 Scope and objective

This Recommendation defines how ASN.1 modules are usable in combination with SDL-2010. ASN.1 modules are imported in SDL-2010 descriptions so that ASN.1 data definitions are mapped to internal SDL-2010 representation using equivalent SDL-2010 constructs and forming together with the rest of the SDL-2010 description a complete specification.

SDL-2010 is a language for the specification and description of telecommunication systems. SDL-2010 has concepts for:

- structuring systems;
- defining behaviour of systems;
- defining data used by systems.

ASN.1 is a language for the definition of data. Related to ASN.1 are encoding rules that define how ASN.1 values are transferred as bit streams during communication.

1.1 Objective

The combination of SDL-2010 and ASN.1 permits a coherent way of specifying the structure and behaviour of telecommunication systems, together with data, messages, and encoding of messages that these systems use. It is possible to describe structure and behaviour using SDL-2010, and data and messages using ASN.1. It is possible to describe the encoding of these messages by reference to the relevant encoding rules defined for ASN.1.

The full use of SDL-2010 (including data types) is supported by this Recommendation.

1.2 The characteristics of the combination of SDL-2010 and ASN.1 modules

Systems described in SDL-2010 combined with ASN.1 modules have the following characteristics:

- structure and behaviour are defined using SDL-2010 concepts;
- the SDL-2010 signal structure, i.e., the signal parameter types and their subtypes are defined in ASN.1 modules;
- it is allowed to define internal data by either ASN.1 types or SDL-2010 sorts;
- it is possible to define encoding of data values defined in ASN.1 by reference to the relevant encoding rules. Encoding is not in the scope of this Recommendation.

1.3 ASN.1 that is usable in combination with SDL-2010

The use of ASN.1 as defined in [ITU-T X.680], [ITU-T X.681], [ITU-T X.682] and [ITU-T X.683] is supported in combination with SDL-2010, with a recognition that it is not possible to successfully map some ASN.1 constructs to SDL-2010 (or at least the mapping has not been identified and specified in this Recommendation). Source ASN.1 constructs that cannot be mapped to SDL-2010 are treated during the transformation to SDL-2010 as if they were not present and should thus not cause any problems for the successful transformation of other constructs. Such constructs are the extension marker and exception marker defined in [ITU-T X.680], which are optionally present in ASN.1 but are ignored in the transformation to SDL-2010. Some constructs of ASN.1 are never transformed to SDL-2010 as such, but contain information that directs the transformation or is used

in the transformation. The prominent examples of such constructs are relational constraints as defined in [ITU-T X.682], and information object classes and information object sets (see clause 8).

The use of SDL-2010 as defined in [ITU-T Z.100], [ITU-T Z.101], [ITU-T Z.102], [ITU-T Z.103], [ITU-T Z.104], and [ITU-T Z.106] is supported.

ASN.1 modules that are used in the transformation to SDL-2010 are also usable for the generation of encoders and decoders, provided that encoding rules are defined. The SDL-2010 data specification implicitly derived from ASN.1 modules should not be used for generation of encoders and decoders, because it is possible to lose some information that is relevant for encoding in the transformation to SDL-2010.

1.4 The structure of this Recommendation

This Recommendation is not self-contained: the mapping defined in this Recommendation is based on [ITU-T X.680], [ITU-T X.681], [ITU-T X.682], [ITU-T X.683], [ITU-T Z.100], [ITU-T Z.101], [ITU-T Z.102], [ITU-T Z.103], [ITU-T Z.104] and [ITU-T Z.106]. The language as defined in [ITU-T Z.100], [ITU-T Z.101], [ITU-T Z.102], [ITU-T Z.103], [ITU-T Z.104] and [ITU-T Z.106] applies, except that the rule <sdl specification> is extended to allow direct use of ASN.1 modules. This Recommendation is structured in the following manner:

Clause 6 defines the changes to SDL-2010 in order to incorporate ASN.1 modules.

Clause 7 defines the mapping of [ITU-T X.680] ASN.1 types and values to SDL-2010 data in order to incorporate ASN.1 data types and values.

Clause 8 defines the mapping of ASN.1 types defined using information objects, classes and information object sets. The use of [ITU-T X.682] constructs is also treated in this clause.

Clause 9 defines the mapping of parameterized ASN.1 types to SDL-2010 data in order to incorporate parameterized ASN.1 data types.

Clause 10 lists items in the **package** Predefined needed to support the use of ASN.1.

2 References

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

[ITU-T X.680] Recommendation ITU-T X.680 (2008) | ISO/IEC 8824-1:2008, *Information technology – Abstract Syntax Notation One (ASN.1): Specification of basic notation*.

[ITU-T X.681] Recommendation ITU-T X.681 (2008) | ISO/IEC 8824-2:2008, *Information technology – Abstract Syntax Notation One (ASN.1): Information object specification.*

[ITU-T X.682] Recommendation ITU-T X.682 (2008) | ISO/IEC 8824-3:2008, *Information technology – Abstract Syntax Notation One (ASN.1): Constraint specification.*

[ITU-T X.683] Recommendation ITU-T X.683 (2008) | ISO/IEC 8824-4:2008, Information technology – *Abstract Syntax Notation One (ASN.1): Parameterization of ASN.1 specifications*.

[ITU-T Z.100] Recommendation ITU-T Z.100 (2011), *Specification and Description Language – Overview of SDL-2010*.

[ITU-T Z.101] Recommendation ITU-T Z.101 (2011), *Specification and Description Language – Basic SDL-2010*.

[ITU-T Z.102] Recommendation ITU-T Z.102 (2011), *Specification and Description Language – Comprehensive SDL-2010*.

[ITU-T Z.103] Recommendation ITU-T Z.103 (2011), *Specification and Description Language – Shorthand notation and annotation in SDL-2010*.

[ITU-T Z.104] Recommendation ITU-T Z.104 (2011), *Specification and Description Language – Data and action language in SDL-2010*.

[ITU-T Z.106] Recommendation ITU-T Z.106 (2011), *Specification and Description Language – Common interchange format for SDL-2010*.

3 Definitions

3.1 Terms defined elsewhere

This Recommendation uses the following terms defined elsewhere: the definitions of [ITU-T Z.100] apply for SDL-2010 items and the definitions of [ITU-T X.680] apply for ASN.1 items.

3.2 Term defined in this Recommendation

None.

4 Abbreviations and acronyms

This Recommendation uses the following abbreviations and acronyms: the abbreviations defined in [ITU-T Z.100] and [ITU-T X.680] apply, in particular SDL-2010 from [ITU-T Z.100] and ASN.1 from [ITU-T X.680].

5 Conventions

In subsequent clauses, the *ASN.1 grammar* identifies the ASN.1 production rules and items that are mapped to SDL-2010, and the *Model* describes the mapping to SDL-2010.

6 Package

ASN.1 grammar

ModuleDefinition, **ModuleIdentifier**, **DefinitiveIdentification**, **Imports** and **Exports** are defined in clause 13.1 of [ITU-T X.680].

Model

The rule <sdl specification> is extended as follows:

```
<sdl specification>     ::=
                        {       { <package diagram> | <package definition> |
                                        <system specification> | <textual system specification> |
                                        <module definition> }
                                <referenced definition>* }set
```

NOTE 1 – This extends the rule <sdl specification> in [ITU-T Z.106]. After mapping according to this Recommendation the ASN.1 module definition is replaced by an SDL-2010 <package diagram> (or equivalent <package definition>).

<module definition> ::=

ModuleDefinition

where the non-terminal **ModuleDefinition** is defined in clause 13.1 of [ITU-T X.680].

A <module definition> has the same meaning as a <package diagram> with a <package text area> where:

- **ModuleIdentifier** (without any **DefinitiveIdentification**) corresponds to the <package name>;
- **Imports** corresponds to the <package use clause>s;
- **Exports** corresponds to the <package public> in the <package text area> of the <package diagram>;
- the remaining parts of the **ModuleDefinition** are transformed into items in the <package text area> of the <package diagram>.

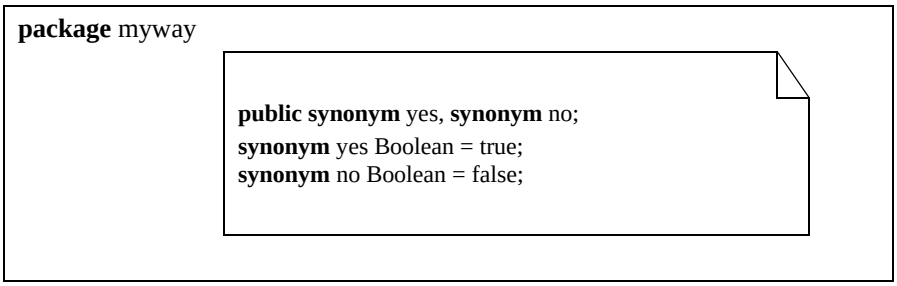
An ASN.1 **ModuleDefinition** is transformed into the equivalent SDL-2010 before it is considered as a package, and before any SDL-2010 transformations. In this transformation, names are transformed into fully qualified identifiers where SDL-2010 requires or allows an identifier rather than a name. However, for conciseness, this is often omitted from the examples in this Recommendation.

Example

The ASN.1 module definition:

```
myway DEFINITIONS ::=
     BEGIN
     EXPORTS yes, no;
     yes  BOOLEAN ::= TRUE
     no   BOOLEAN ::= FALSE
END
```

is the same as:



Similarly, when the package is used in the **imports** of another package:

```
IMPORTS yes FROM myway;
```

This is the same as the <package reference clause>:

use myway/yes;

NOTE 2 – Because SDL-2010 does not support object identifier values for package identification, ASN.1 modules with the same **modulereference** (see clause 12.5 of [ITU-T X.680]) but different **DefinitiveIdentification** will potentially cause name resolution problems.

7 Definition and use of data

The different definitions of the use of data are described in the following way:

ASN.1 grammar	Defining the grammar production rules representing the construction to be represented in SDL-2010.
Model	Describing the transformations of the different parts of the ASN.1 grammar into SDL-2010 productions. This part is referencing both the SDL-2010 grammar, represented as <sdl grammar rule>, and the ASN.1 grammar, represented as ASN1GrammarRule.

7.1 Name mapping

ASN.1 grammar

ASN.1 names are ASN.1 lexical items defined in clause 11 of [ITU-T X.680] such that only letters, digits, and hyphens are allowed in ASN.1 names. If the hyphen character is used in SDL-2010, this would be interpreted as the minus operator.

Model

ASN.1 names containing hyphen characters are mapped to lexically similar SDL-2010 names except that hyphen characters are converted to underline characters.

Example

The ASN.1 name **`my-example-name`** is mapped to my_example_name in SDL-2010.

7.2 Variable and data definitions

7.2.1 Type assignment

ASN.1 grammar

TypeAssignment, **Type**, **ConstrainedType** and **Constraint** are defined in clause 16.1 of [ITU-T X.680], clause 17.1 of [ITU-T X.680], clause 49.1 of [ITU-T X.680] and clause 49.6 of [ITU-T X.680], respectively.

Model

If the **Type** is a **typereference**, then the **TypeAssignment** is the same as a <syntype definition> (see [ITU-T Z.101] and clause 12.1.8.1 of [ITU-T Z.104]) containing only the SDL-2010 equivalent of the **Type**.

If the **Type** is a **ConstrainedType**, then the **TypeAssignment** is the same as a <syntype definition> (see [ITU-T Z.101] and clause 12.1.8.1 of [ITU-T Z.104]) containing only the SDL-2010 equivalent of the **Constraint**.

If the **Type** is a neither a **typereference** nor a **ConstrainedType,** the **TypeAssignment** is represented by a <data type definition> with an empty or omitted <data type definition body> and where <formal context parameters> is omitted (see [ITU-T Z.101] and clause 12.1.1 of [ITU-T Z.104]).

Example

The ASN.1 type assignment:

```
Mytype ::= AnotherType -- typereference
```

is the same as:

syntype Mytype = AnotherType **endsyntype** Mytype; /* full qualification omitted here. */

The ASN.1 type assignment:

```
S ::= INTEGER (0..5 | 10)
```

is the same as:

syntype S = <<**package** Predefined>>Integer **constants** (0..5,10) **endsyntype** S;

The ASN.1 type assignment:

```
Integerlist ::= SEQUENCE OF INTEGER
```

is the same as:

value type Integerlist {

inherits <<**package** Predefined>>String

< <<**package** Predefined>> Integer> (" = <<**package** Predefined>> Emptystring) }

7.2.2 Value assignment

ASN.1 grammar

ValueAssignment and **XMLValueAssignment** are defined in clause 16.2 of [ITU-T X.680].

Model

The **ValueAssignment** and **XMLValueAssignment** are represented by an <internal synonym definition item> in a <synonym definition> (see clause 12.1.8.3 of [ITU-T Z.104]).

Example

The ASN.1 definition:

```
yes BOOLEAN ::= TRUE
```

is the same as:

synonym yes <<**package** Predefined>>Boolean = <<**package** Predefined>> true;

7.3 Type expressions

7.3.1 Sequence

ASN.1 grammar

SequenceType, **ComponentType**, **ExtensionAndException**, **OptionalExtensionMarker**, **ExtensionAdditionGroup** and **VersionNumber** are defined in clause 25.1 of [ITU-T X.680]. **SetType** is defined in clause 27.1 of [ITU-T X.680].

Model

A **SequenceType** is represented as a <structure definition> (see [ITU-T Z.101] and clause 12.1.6.2 of [ITU-T Z.104]) containing a <field> for each **NamedType** of the **SequenceType**. The <field> contains one <field name>, which is the same as the ASN.1 **identifier** of the **NamedType**, and a <field sort> that is the **Type** transformed to an SDL-2010 <sort identifier>.

If the **ComponentType** containing the **NamedType** is **OPTIONAL**, the SDL-2010 field has the keyword **optional**.

If the **ComponentType** containing the **NamedType** has a **DEFAULT Value**, the SDL-2010 field has the keyword **default** and the value is transformed into the <constant expression> after **default**.

A **ComponentType** that is **COMPONENTS OF Type** is represented as a list of ordered <field>s, one for each field associated to **Type**. These fields are inserted in the position of the **COMPONENTS OF Type** in the order that the fields exist in the **Type**.

The occurrences of **ExtensionAndException** and **OptionalExtensionMarker** in **SequenceType** are ignored in the transformation.

The occurrences of **ExtensionAdditionGroup** in **ExtensionAddition** are transformed so that version brackets ("[[", "]]") and **VersionNumber** are ignored.

Example

The ASN.1 type:

```
S    ::=  SEQUENCE {
          a    INTEGER,
          b    IA5String OPTIONAL,
          c    PrintableString DEFAULT "d"}
```

is the same as:

value type S

{ **struct**

a <<**package** Predefined>> Integer;

b <<**package** Predefined>> IA5String optional;

c <<**package** Predefined>> PrintableString default 'd';

}

NOTE 1 – There is no distinction between use of keyword **SEQUENCE** and **SET**. This is a relaxation compared to [ITU-T X.680].

NOTE 2 – In this Recommendation, tags are not necessary to distinguish between components of the same type: ASN.1 automatic tagging is assumed.

7.3.2 Sequenceof

ASN.1 grammar

SequenceOfType is defined in clause 26.1 of [ITU-T X.680].

Model

Specifying a **SequenceOfType** is the same as specifying the Predefined String sort having the SDL-2010 transform of **Type** as the first <actual context parameter> and the name Emptystring defined as the literal name for the empty string.

If an ASN.1 size constraint is specified for **Type**, the **SequenceOfType** is a **syntype** having the transformed size constraint (see clause 7.4) as a <constraint> that is a <size constraint> (see clause 12.1.8.2 of [ITU-T Z.101]). The parent sort of the **syntype** is the **SequenceOfType** without the ASN.1 size constraint. This parent sort has an implicit and unique name and is defined in the nearest scope unit enclosing the occurrence of the **SequenceOfType**.

Example

The ASN.1definition:

```
phonenumber ::= SEQUENCE SIZE (8) OF INTEGER (0..9)
```

is the same as the three SDL-2010 definitions:

value type S1

{

inherits <<**package** Predefined>> String <S2> (" = Emptystring)

}

syntype S2 = <<**package** Predefined>> Integer **constants** (0..9) **endsyntype**;

syntype phonenumber = S1 **constants size** (8) **endsyntype** phonenumber;

7.3.3 Choice

ASN.1 grammar

ChoiceType, **ExtensionAdditionAlternative** and **ExtensionAdditionAlternativesGroup** are defined in clause 29.1 of [ITU-T X.680].

Model

A **ChoiceType** is represented as a <choice definition> (see [ITU-T Z.101] and clause 12.1.6.3 of [ITU-T Z.104]) containing a <field> for each **NamedType** of the **ChoiceType**.

The occurrences of **ExtensionAndException** and **OptionalExtensionMarker** in **ChoiceType** are ignored in the transformation.

The occurrences of **ExtensionAdditionAlternativesGroup** in **ExtensionAdditionAlternatives** are transformed so that version brackets ("[[", "]]") and **VersionNumber** are ignored.

Example

The ASN.1 choice type:

```
C ::= CHOICE {
a    INTEGER,
b    REAL }
```

is the same as:

value type C

{ **choice**

a <<**package** Predefined>> Integer;

b <<**package** Predefined>> Real;

}

7.3.4 Enumerated

ASN.1 grammar

EnumeratedType and **EnumerationItem** are defined in clause 20.1 of [ITU-T X.680] and **ExceptionSpec** (used in **EnumeratedType**) is defined in clause 53.4 of [ITU-T X.680].

Model

An **EnumeratedType** is represented by a <data type definition> where <formal context parameters> is omitted and the <data type definition body> (see [ITU-T Z.101] and clause 12.1.1 of [ITU-T Z.104]) is a <data type constructor> that is a <literal list> (see [ITU-T Z.101] and clause 12.1.6.1 of [ITU-T Z.104]). For each **EnumerationItem**, the **identifier** is transformed into a <literal signature> that has the same name as the **EnumerationItem**. If the **EnumerationItem** contains a **SignedNumber** (or **DefinedValue**), the <literal name> of the <literal signature> is followed by the SDL-2010 transform of the **SignedNumber** (or **DefinedValue** respectively) to form a <named number>.

The extension markers ("...") and **ExceptionSpec** in **EnumeratedType** are ignored in the transformation to SDL-2010.

The definition:

```
colours ::= ENUMERATED {blue(3),red, yellow(0)};
```

is the same as:

value type colours {

literals blue = 3, red, yellow = 0

}

7.3.5 Integer

ASN.1 grammar

IntegerType, **NamedNumberList** and **NamedNumber** are defined in clause 19.1 of [ITU-T X.680].

Model

The ASN.1 **IntegerType** is mapped to SDL-2010 <<**package** Predefined>> Integer.

Specifying an **IntegerType** with a **NamedNumberList** is the same as specifying a <synonym definition> (See clause 12.1.8.3 of [ITU-T Z.104]) in the nearest enclosing scope unit with one <synonym definition item> for each **NamedNumber**. The **identifier** of the **NamedNumber** is transformed into the <synonym name>. The <sort> of the <synonym definition item> is <<**package** Predefined>>Integer. The **SignedNumber** or **DefinedValue** of the **NamedNumber** is used as the <constant expression> of an <internal synonym definition item>.

Example

The ASN.1 definition:

```
Standards ::=  SEQUENCE OF INTEGER{z100(0),x680(1),z10x(2)}
```

is the same as:

value type standards **inherits**

<< **package** Predefined >> String < <<**package** Predefined>> Integer> ("= EmptyString);

synonym z100 Integer = 0,

x680 Integer = 1,

z10x Integer = 2;

7.3.6 ValueRange

ASN.1 grammar

ValueRange is defined in clause 51.4 of [ITU-T X.680].

Model

Specifying an ASN.1 ValueRange restriction is represented as specifying the constrained <sort> and adding the representation of the ASN.1 **ValueRange** restriction after the **constants** keyword in the <syntype definition> (see [ITU-T Z.101] and clause 12.1.8.1 of [ITU-T Z.104]).

Example

The ASN.1 definition:

```
S    ::=  INTEGER(0..5 | 10)
```

is equivalent to:

syntype S = <<**package** Predefined>> Integer **constants** (0..5, 10) **endsyntype** S;

How the <constraint> is derived is described in clause 7.4 below.

7.3.7 BitString

ASN.1 grammar

BitStringType, **NamedBitList** and **NamedBit** are defined in clause 22.1 of [ITU-T X.680].

Model

The ASN.1 **BitStringType** is mapped to SDL-2010 <<**package** Predefined>> Bitstring.

Specifying a **BitStringType** with a **NamedBitList** is the same as specifying a <synonym definition> (See clause 12.1.8.3 of [ITU-T Z.104]) in the nearest enclosing scope unit with one <synonym definition item> for each **NamedBit**. The **identifier** of the **NamedBit** is transformed into the <synonym name>. The <sort> of the <synonym definition item> is <<**package** Predefined>>Integer. The **number** or the **DefinedValue** is used as the <constant expression> of the <synonym definition item>.

7.3.8 OctetString

ASN.1 grammar

OctetStringType is defined in clause 23.1 of [ITU-T X.680].

Model

The ASN.1 type **OctetStringType** is mapped to SDL-2010 <<**package** Predefined >> Octetstring.

7.3.9 Setof

ASN.1 grammar

SetOfType is defined in clause 27.1 of [ITU-T X.680].

Model

Specifying a **SetOfType** is the same as specifying the <<**package** Predefined>> Bag sort having the SDL-2010 transform of **Type** as the first <actual context parameter> and the name Emptybag defined as the literal name for the empty bag.

If an ASN.1 size constraint is specified for **Type**, the **SetOfType** is a **syntype** having the transformed size constraint as a <constraint> (see clause 7.4). The parent sort of the **syntype** is the **SetOfType** without the ASN.1 size constraint. This parent sort has an implicit and unique name and is defined in the nearest scope unit enclosing the occurrence of the **SetOfType**.

7.4 Range condition and size constraint

ASN.1 grammar

See clause 51.4 of [ITU-T X.680].

Model

In SDL-2010 **syntype** has a <constraint> that is either a <range condition> or a <size constraint> (see clause 12.1.8.2 of [ITU-T Z.101]).

A range condition defines a set of values. It is used in SDL-2010 for defining a **syntype**. It has an associated parent sort, which is the sort specified in the **syntype** definition. A value is within the value set if the operator denoted by the operator identifier yields true when applied to the value.

The operator identifier for a given range condition is thus defined as:

value type A

operators o: S -> Boolean;

/* where o is derived from the ASN.1 concrete syntax as explained below */

endvalue type A;

Each Range in the ASN.1 range condition contributes to the properties of the operator defining the value set:

o(V) == range1 or range2 or ... or rangeN

If a **syntype** is specified without a range condition, then the operator result is true.

In the following explanation of how each Range contributes to the operator result, V denotes the argument value. Each contribution shall be well-formed, which means that used operators shall exist with a signature appropriate for the context.

- If neither of the keywords **MIN** and **MAX** are specified in a **ValueRange**, a **ValueRange** contributes with:

  E1 rel1 V and V rel2 E2

  where E1 is Value of **LowerEndValue** and E2 is Value of **UpperEndValue**.

  If "<" is specified for **LowerEndValue** then rel1 is the "<" operator; otherwise it is the "<=" operator.

  If "<" is specified for **UpperEndValue** then rel2 is the "<" operator; otherwise it is the "<=" operator.

  If the keyword **MIN** is specified and the keyword **MAX** is not specified, **ValueRange** contributes with:

  V rel2 E2

  If the keyword **MAX** is specified and the keyword **MIN** is not specified, **ValueRange** contributes with:

  E1 rel1 V

  If both keywords **MIN** and **MAX** are specified, the operator always yields true.
- A **ContainedSubtype** (see clause 51.3 of [ITU-T X.680]) contributes with:

  o1(V)

  where o1 is the implicit operator defining the value set for the **Type** mentioned in the **ContainedSubtype**.

- A **SizeConstraint** (see clause 51.5 of [ITU-T X.680]) contributes with:

  o1(length(V))

  where o1 is the implicit operator defining the value set for the <range condition> mentioned in the **SizeConstraint**.

- **InnerTypeConstraints** (see clause 51.8 of [ITU-T X.680]) contributes with either:

  if length(V) = 0 **then** true **else** o1(first(V)) **and** o(Substring(V,2,length(V)-1)) **fi**; or

  if length(V) = 0 **then** true **else** o1(take(V)) **and** o(del(take(V), V)) **fi**

  whatever is appropriate for the sort of V. o is the implicit operator **InnerTypeConstraints** contributes to and o1 is the implicit operator for **Range** specified in **InnerTypeConstraints**.

  InnerTypeConstraints has a contribution for each contained **NamedConstraint** that specifies constraints of the field (see clause 7.2.1) denoted by **Identifier** of the parent sort.

  For the purpose of deriving the contributions, a derived ASN.1 type is created as follows:

  a) the keyword **PRESENT** is added to the **NamedConstraints** that have no ending keyword (**PRESENT**, **ABSENT** or **OPTIONAL**);

  b) **NamedConstraints** of the form **Identifier ABSENT** are added for all fields (i.e., Identifiers) not mentioned explicitly in a **NamedConstraint**. The **NamedConstraints** are added to the **InnerTypeConstraints** before the contributions of each **NamedConstraint** are derived.

  NOTE – In case the governing type is CHOICE, it is possible that the derived type is illegal with respect to [ITU-T X.680], but it is used only for the purpose of mapping to SDL-2010 and therefore has no impact on the original ASN.1 type or its encoding.

  If a **ValueRange** is specified for a **NamedConstraint**, the contribution is:

  E and if FPresent(V) then o1(V) else true fi

  where E is the present constraint for the field, F (from the operator name FPresent) is the name of the optional field and o1 is the implicit operator for the **ValueRange**. If the **ValueRange** is omitted, the contribution is only the present constraint E.

  The present constraint for a field F is:

  FPresent(V)

  in case the **NamedConstraint** for the field contains the keyword **PRESENT**; and

  not FPresent(V)

  in case the **NamedConstraint** for the field contains the keyword **ABSENT**. In all other cases, the present constraint is true.

7.5 Value expressions

7.5.1 Choice value

ASN.1 grammar

ChoiceValue is defined in clause 29.11 of [ITU-T X.680].

Model

A **ChoiceValue** is represented as an <operator application> (see clause 12.2.6 of [ITU-T Z.101]) having the **Value** as argument. The <operator identifier> in the <operator application> contains a <qualifier> representing the **Type** and an operator name being the **identifier**.

Example

The **`ChoiceValue`**:

```
myvalue : Mychoice
```

is represented as:

myvalue(Mychoice)

In the case that a **ChoiceValue** denotes one of several operator applications (e.g., a field of more than one choice sort), a qualifier is used:

```
MyType ::= CHOICE ……
myvalue : Mychoice
```

which is then represented as:

<<**type** Mytype>> myvalue(Mychoice)

7.5.2 Composite primary

A composite primary is built up of the values for the SDL-2010 representation of respective composite types.

7.5.2.1 Sequence value

ASN.1 grammar

SequenceValue, **XMLSequenceValue**, **ComponentValueList** and **XMLComponentValueList** are defined in clause 25.18 of [ITU-T X.680].

NOTE – There is no distinction between **`SetValue`** and **`SequenceValue`**. This is a relaxation compared to [ITU-T X.680].

Model

The **SequenceValue** and **XMLSequenceValue** are mapped to <synonym definition item>. In the mapping the **ComponentValueList** or **XMLComponentValueList** is provided to the operator to make a structure value in SDL-2010. The SDL-2010 data type constructor requires that all the fields are given as input so that fields that are omitted in **ComponentValueList** have to be provided empty in SDL-2010. The application of structure data type constructor will have the same effects in SDL-2010 as it would in ASN.1.

Example

```
MYTYPE ::= SEQUENCE{
     a    INTEGER,
     b    INTEGER OPTIONAL,
     c    INTEGER DEFAULT 0,
     d    INTEGER,
     e    INTEGER OPTIONAL,
     f    INTEGER DEFAULT 0
}
myValue MYTYPE ::= {a 1, b 1, c 1, d 1}
```

In this example fields **`a`**, **`b`**, **`c`** and **`d`** of **`myValue`** have a value assigned and fields **`e`** and **`f`** have no assignment.

synonym myValue MYTYPE = (. 1, 1, 1, 1, , .);

The consequence would be that fields a, b, c and d of myValue would be set to 1, e would be absent and f would get the default value 0.

7.5.2.2 Sequence of value

ASN.1 grammar

SequenceOfValue and **XMLSequenceOfValue** are defined in clause 26.3 of [ITU-T X.680].

Model

A **SequenceOfValue** and **XMLSequenceOfValue** are both represented as:

MkString(E1) // MkString(E2)// ... // MkString(En)

where E1, E2, ..., En are the values of the **SequenceOfValue** or **XMLValueOrEmpty** of the **XMLSequenceOfValue** in the order of appearance. If no **Value** or **XMLValue** is specified, the **SequenceOfValue** or **XMLSequenceOfValue** are represented as the name Emptystring.

The **Type** qualifier of the Composite Primary that contains the **SequenceOfValue** precedes each MkString operator or the Emptystring literal, respectively.

7.5.2.3 Object identifier value

ASN.1 grammar

ObjectIdentifierValue is defined in clause 32.3 of [ITU-T X.680].

Model

ObjectIdentifierValue is ignored in the transformation to SDL-2010.

ObjectIdentifierValue is in ASN.1 used to distinguish between the modules that have the same names but different object identifiers. Because the module names and object identifiers are not uniquely mapped to a package identifier that is used in package use clauses, the object identifier component is ignored in the transformation to SDL-2010. The identification of appropriate modules is thus open to manual or tool specific solutions.

7.5.2.4 Real value

ASN.1 grammar

RealValue and **XMLRealValue** are defined in clause 21.6 of [ITU-T X.680].

The form **0** is used for zero values; the alternate form for **NumericRealValue** shall not be used for zero values.

The associated type for value definition and subtyping purposes is:

```
SEQUENCE {
     mantissa  INTEGER,
     base      INTEGER (2|10),
     exponent  INTEGER
          -- The associated mathematical real number is "mantissa"
          -- multiplied by "base" raised to the power "exponent"
     }
```

Model

An ASN.1 **NumericalRealValue** and **XMLNumericRealValue** are mapped to an SDL-2010 real sort value with the actual value calculated in the transformation. The **SpecialRealValue** and **XMLSpecialRealValue** shall be transformed to the largest possible positive or negative value respectfully.

NOTE – The transformation of **SpecialRealValue** is not in accordance with the intended ASN.1 semantics because this is a directive to the encoder/decoder to use a special code indicating the –∞ (minus infinite) values. Since encoding is not related to data in SDL-2010 transformed from ASN.1data, such relaxation should be acceptable.

Example

The ASN.1 definition:

```
r50  REAL ::= { mantissa 5, base 10, exponent 1}
```

is the same as:

synonym r50 Real = 50.0;

7.5.2.5 Integer value

ASN.1 grammar

IntegerValue and **XMLIntegerValue** are defined in clause 19.9 of [ITU-T X.680].

Model

An **IntegerValue** and **XMLIntegerValue** are mapped to an SDL-2010 <<**package** Predefined >> Integer sort value with the same actual value.

7.5.2.6 Boolean value

ASN.1 grammar

BooleanValue and **XMLBooleanValue** are defined in clause 18.3 of [ITU-T X.680].

Model

A **BooleanValue** and **XMLBooleanValue** are mapped to an SDL-2010 <<**package** Predefined >> Boolean sort value where **`TRUE`** and **<true>** map to Boolean literal true and **`FALSE`** `and` **<false>** map to Boolean literal false.

7.5.3 String primary

ASN.1 grammar

CharacterStringValue and **XMLCharacterStringValue** are defined in clause 40.3 of [ITU-T X.680].

BitStringValue and **XMLBitStringValue** are defined in clause 22.9 of [ITU-T X.680].

Model

An ASN.1 **StringValue** containing a **cstring** (ASN.1 name for character string delimited by **`"`** at both beginning and end) represents a <character string literal identifier> consisting of the **Type** and a <character string literal> with the same <text> as the ASN.1 String **Text**. The **Type** for **cstring** is an **IA5Type** as defined by this Recommendation.

A **BitStringValue** containing a **bstring** or **hstring** are mapped to SDL-2010 <<**package** Predefined>> Bitstring operators with the same syntax provided that the length of the governing bit string type is not constrained.

Provided that the length of the governing bit string type is constrained, the rules below apply.

A **BitStringValue** containing a **bstring** or **hstring** are mapped to SDL-2010 <<**package** Predefined>> Bitstring operators with the same syntax. However, if the length of the **bstring** or **hstring** is smaller than the maximal length of the governing bit string type, the **bstring** or **hstring** is expanded to the maximal length with all trailing bits sets to 0.

A **BitStringValue** defined using **IdentifierList** is evaluated as having the bit value 1 at all bit positions defined by **identifier** listed in **IdentifierList**. The value of the bits remaining until the maximal length of the governing bit string type is set to 0. The resulting string is mapped to SDL-2010 <<**package** Predefined>> Bitstring operators preceded by an ' character and followed by the pair of characters 'B.

An **XMLBitStringValue** containing an **xmlbstring** is mapped to SDL-2010 <<**package** Predefined>> Bitstring operators preceded by a ' character and followed by the pair of characters 'B. However, if the length of the **xmlbstring** is smaller than the maximal length of the governing bit string type, the **xmlbstring** is expanded to the maximal length with all trailing bits sets to 0.

Example

Use a bit string type to model the values of a **bit map**, a fixed-size ordered collection of logical variables indicating whether a particular condition holds for each of a correspondingly ordered collection of objects.

```
DaysOfTheWeek ::= BIT STRING {
          sunday(0), monday (1), tuesday(2),
          wednesday(3), thursday(4), friday(5),
          saturday(6) } (SIZE (7))

     sunnyDaysLastWeek1 DaysOfTheWeek ::= {sunday, monday, wednesday}
     sunnyDaysLastWeek2 DaysOfTheWeek ::= '1101000'B
```

The mapping to SDL-2010 gives the following:

synonym sunday Integer = 0;

synonym monday Integer = 1;

synonym tuesday Integer = 2;

synonym wednesday Integer = 3;

synonym thursday Integer = 4;

synonym friday Integer = 5;

synonym saturday Integer = 6;

synonym sunnyDaysLastWeek1 DaysOfTheWeek = '1101000'B;

synonym sunnyDaysLastWeek2 DaysOfTheWeek = '1101000'B;

Use a bit string type to model the values of a **bit map**, a variable-size ordered collection of logical variables indicating whether a particular condition holds for each of a correspondingly ordered collection of objects.

```
DaysOfTheWeekVar ::= BIT STRING {
          sunday(0), monday (1), tuesday(2),
          wednesday(3), thursday(4), friday(5),
          saturday(6) } (SIZE (0..7))
     sunnyDaysLastWeek3 DaysOfTheWeekVar ::= {sunday, monday, wednesday}
     sunnyDaysLastWeek4 DaysOfTheWeekVar ::= '1101'B
```

The mapping to SDL-2010 would give the following (not repeating the synonyms for bit names):

synonym sunnyDaysLastWeek3 DaysOfTheWeek = '1101000'B;

synonym sunnyDaysLastWeek4 DaysOfTheWeek = '1101000'B;

7.5.4 Element set specification

ASN.1 grammar

ElementSetSpecs is defined in clause 50.1 of [ITU-T X.680].

Model

It is possible to combine two or more value sets using this notation. The resulting set is evaluated in the transformation and the result is mapped to SDL-2010.

The extension markers ("...") in **ElementSetSpecs** are ignored in the transformation to SDL-2010.

8 Mapping for information objects, classes and object sets

The mapping of ASN.1 types defined in ASN.1 modules using information objects, information object classes and information object sets is defined below.

8.1 Introduction

[ITU-T X.681] provides the ASN.1 notation that allows information object classes as well as individual information objects and sets thereof to be defined and given reference names. An information object class is a template for a collection of information that makes up the attributes of any members of that class. Information objects provide a generic table mechanism within the ASN.1 language. Such a generic table defines the association of specific sets of field values or types. This feature replaces the earlier MACRO construct (available in ASN.1:1990) and is primarily used to fill in gaps in a type definition dependent on one or more key fields.

This clause assumes that all ASN.1 constructs defined in [ITU-T X.681], [ITU-T X.682] and [ITU-T X.683] are usable in ASN.1 modules. It then identifies what information contained in ASN.1 information object classes, information objects and information object sets are useful when mapped to appropriate SDL-2010 targets. The mappings that are possible and useful are defined. It has to be noted that some information will not be represented in SDL-2010 because of the differences in nature of the two languages.

8.2 Information object class definition and assignment

ASN.1 grammar

ObjectClassAssignment is defined in clause 9.1 of [ITU-T X.681].

Model

The **ObjectClass** definitions (see clause 9.2 of [ITU-T X.681]) in ASN.1 have no direct correspondence in SDL-2010.

8.3 Information object class field type

ASN.1 grammar

ObjectClassFieldType, **FixedTypeValueFieldSpec** and **FixedTypeValueSetFieldSpec** are defined in clauses 14.1, 9.6 and 9.9 of [ITU-T X.681], respectively.

Model

It is possible to define ASN.1 types using **ObjectClassFieldType** notation to extract information from the fields of information object class specifications without presence of table constraints. Such ASN.1 types are able to be mapped to SDL-2010, provided that in their definition only **FixedTypeValueFieldSpec** or **FixedTypeValueSetFieldSpec** items are used. The mapping to an SDL-2010 **value type** is done as defined in clause 7.3 once the meaning of the

FixedTypeValueFieldSpec or **FixedTypeValueSetFieldSpec** is determined from the referenced information object class specifications.

ObjectClassFieldType notation is also used in relation to table constraints as defined in clause 8.6.2.

Example

If the ASN.1 contains the following specification:

```
EXAMPLE-CLASS ::= CLASS {
     &TypeField                                  OPTIONAL,    -- class field 1
     &fixedTypeValueField         INTEGER        OPTIONAL,    -- class field 2
     &variableTypeValueField      &TypeField     OPTIONAL,    -- class field 3
     &FixedTypeValueSetField      INTEGER        OPTIONAL,    -- class field 4
     &VariableTypeValueSetField   &TypeField     OPTIONAL     -- class field 5
}
WITH SYNTAX {
     [TYPE-FIELD                        &TypeField]
     [FIXED-TYPE-VALUE-FIELD            &fixedTypeValueField]
     [VARIABLE-TYPE-VALUE-FIELD         &variableTypeValueField]
     [FIXED-TYPE-VALUE-SET-FIELD        &FixedTypeValueSetField]
     [VARIABLE-TYPE-VALUE-SET-FIELD     &VariableTypeValueSetField]
}
ExampleType ::= SEQUENCE {
     integerComponent1  EXAMPLE-CLASS.&fixedTypeValueField,  -- field 1
     integerComponent2  EXAMPLE-CLASS.&FixedTypeValueSetField      -- field 2
}

exampleValue ExampleType ::= {
     integerComponent1       123,                 -- field 1
     integerComponent2       456                  -- field 2
}
```

Things that are able to be mapped to SDL-2010 are `ExampleType` and `exampleValue`:

value type ExampleType {

struct

integerComponent1 <<**package** Predefined>> Integer, /* field 1 */

integerComponent2 <<**package** Predefined>> Integer /* field 2 */

}

synonym exampleValue ExampleType = (. 123, 456 .);

8.4 Information object definition and assignment

ASN.1 grammar

ObjectAssignment is defined in clause 11.1 of [ITU-T X.681].

Model

Information object definitions in the ASN.1 module have no equivalent mapping in SDL-2010.

8.5 Information from information objects

ASN.1 grammar

InformationFromObjects is defined in clause 15.1 of [ITU-T X.681].

Model

It is possible to reference information from the column of the associated table for an information object (or an information object set) by the various cases of the **InformationFromObjects** notation.

In the ASN.1 module, an ASN.1 type is specifiable with fields defined using **InformationFromObjects** notation. Such an ASN.1 type is able to be mapped to SDL-2010, provided that all occurrences of **InformationFromObjects** notation are expandable to a value or a type. The ASN.1 type as such is mapped as specified in clause 7.3, while the semantics of **InformationFromObjects** expansion follows the ASN.1 semantics.

8.6 Constraint specification

ASN.1 grammar

GeneralConstraint, **TableConstraint** and **UserDefinedConstraint** are defined in clause 8.1 of [ITU-T X.682].

Model

The types specified using **TableConstraint** are mapped to SDL-2010 according to rules given in clause 8.6.2. It is not possible to map types specified using **UserDefinedConstraint** to SDL-2010.

8.6.1 User-defined constraints

ASN.1 grammar

UserDefinedConstraint is defined in clause 9.1 of [ITU-T X.682].

Model

This form of constraint specification is regarded as a special form of ASN.1 comment, since it is not fully machine-processable. Therefore, it is not possible to map ASN.1 type specifications using **UserDefinedConstraint** to SDL-2010.

8.6.2 Table constraints

ASN.1 grammar

TableConstraint, **SimpleTableConstraint** and **ComponentRelationConstraint** are defined in clauses 10.3, 10.3 and 10.7 of [ITU-T X.682], respectively.

Model

Constraint notation is allowed to appear (in round brackets) after any use of the syntactic construct "Type". Application designers optionally use this notation to define a structured data type with further constraints on their field values. Examples of such constraints are restricting the range of some component(s), or to specify a relation between components. The former is a **SimpleTableConstraint** and the latter is a **ComponentRelationConstraint**.

For types with **SimpleTableConstraint**, the following transformation rules apply.

Before the constrained type is mapped to SDL-2010, some SDL-2010 **value type**s need to be constructed from the information object class specification and the constraining information object set specification in the following manner.

a) For each information object set a number of SDL-2010 **value type** items are created. The types are generated so that for each field of the information object CLASS associated with the information object set, one SDL-2010 **value type** is generated. The name of the type is the concatenation of the name of the information object set, an underscore ('_') and the name of the matching class field.

b) If the class field is a **FixedTypeValueFieldSpec**, an SDL-2010 **syntype** is constructed. The **syntype** has a range constraint that is a union of values specified by the matching field of each information object in the information object set.

c) If the information object class field is a **VariableTypeValueFieldSpec**, an SDL-2010 choice type is constructed. The choice type is constructed so that all the types found in the matching field of all the information objects belonging to the constraining information object set are included in the choice. The choice field names are derived as lower case equivalents of the matching types.

The constrained ASN.1-type is now mapped to SDL-2010. The type as such is mapped as defined in clause 7.3. The SDL-2010 field names are the same as ASN.1 field names. The ASN.1 specification of optionality is preserved in the transformation. For each ASN.1 field constrained by an information object set, the SDL-2010 type is specified as type constructed from the information object class specification and the constraining information object set specification (items (a) to (c)).

For ASN.1-type specifications using **ComponentRelationConstraint**, the same type transformation rules are applied. On top of that, for each ASN.1 type with **ComponentRelationConstraint**, a check method that traverses the information object and checks the constraints is also generated. The check method returns "true" if all relational constraints are respected and "false" if any of the relational constraints are violated.

The steps for constructing the check method are:

For each element that is involved in relational constraint (has a **ComponentRelationConstraint** attached to it or is mentioned in any **ComponentRelationConstraint**), a local test variable declaration is generated. The generation follows the following scheme:

'**dcl** <test var name> <field type>; <test var name> := <field ref>;'

where <test var name> is a unique name for each test variable, <field type> is the type of the element, <field ref> is a reference to the element. If the element is present, the variable is initialized to the value of the corresponding field of the object.

For each relational constraint, one test is generated for each combination of constraining values or types in the object set definition. Each test is generated using the following scheme:

'**if** (<test expr> **and not** (<value test>)) **then** { **return** false; }

where the <test expr> is the result of combining one test for each constraining value or type using the '**and**' operator. For constraining values the test is defined as:

'<test var name> = <test value>'

where <test var name> is the name of the test variable as described above and <test value> is the corresponding value from the object set definition.

For constraining types, the test is defined as:

'<test var name>.<ispresent method>'

where <test var name> is the name of the test variable as described above and the <ispresent method> is the method that checks that the corresponding type is present.

The <value test> is the result of combining one test for each value or type of the constrained element in the object set definition that corresponds to the values and types in the <test expr> above, using the 'or' operator. For values each test is given as:

'<test var name> = <value>'

where the <test var name> is the name of the variable corresponding to the constrained field and <value> is a value from the object set definition.

For types, the test is defined as:

'<test var name>.<ispresent method>'

where <test var name> is the name of the variable corresponding to the constrained field and the <ispresent method> is the method that checks that the corresponding type is present.

For each String field in the type, a loop is generated according to the following scheme:

'**loop**(**dcl** <loop var> Integer := 1; <loop var> <= length(<string field>);

<loop var> := <loop var> + 1) { <loop body> }'

where <loop var> is a unique variable name, <string field> is a reference to the treated string field and <loop body> is the result of applying the transformation steps in this clause to the elements in the string.

Example 1

An example of a type with **SimpleTableConstraint**:

```
ErrorReturn ::= SEQUENCE
{
  errorCategory ERROR-CLASS.&category({ErrorSet}) OPTIONAL,
  errors SEQUENCE OF SEQUENCE
  {
    errorCode ERROR-CLASS.&code
                          ({ErrorSet}),
    errorInfo ERROR-CLASS.&Type
                          ({ErrorSet})
  } OPTIONAL
}
```

Provided that the specifications of class and object set were:

```
ERROR-CLASS ::= CLASS
{
  &category PrintableString (SIZE(1)),
  &code     INTEGER,
  &Type
}
WITH SYNTAX {&category &code &Type }
ErrorSet ERROR-CLASS ::=
{
  { "A" 1 INTEGER } |
  { "A" 2 REAL } |
  { "B" 1 CHARACTER STRING } |
  { "B" 2 GeneralString }
}
```

The SDL-2010 types derived from constraint specification would be:

```
syntype ErrorSet_category = PrintableString (SIZE(1))
  constants 'A', 'B'
endsyntype;

syntype ErrorSet_code = <<package Predefined>> Integer
  constants 1, 2
endsyntype;

value type ErrorSet_Type { choice
  integer                 <<package Predefined>> Integer;
  real                    <<package Predefined>> Real;
  characterString         <<package Predefined>> CharacterString;
  generalString           <<package Predefined>> GeneralString;
}
```

The constructed SDL-2010 type would be the following:

```
value type ErrorReturn { struct
  errorCategory ErrorSet_category optional;
  errors String <
    { struct
      errorCode ErrorSet_code,
      errorInfo ErrorSet_Type } > optional;
}
```

No check method would be generated.

Example 2

An example of a type with `ComponentRelationConstraint`.

```
ErrorReturn ::= SEQUENCE
{
  errorCategory ERROR-CLASS.&category({ErrorSet}) OPTIONAL,
  errors SEQUENCE OF SEQUENCE
  {
    errorCode ERROR-CLASS.&code
                          ({ErrorSet}{@errorCategory}),
    errorInfo ERROR-CLASS.&Type
                          ({ErrorSet}{@errorCategory,@.errorCode})
  } OPTIONAL
}
```

The corresponding SDL-2010 type would be the following:

```
value type ErrorReturn {
struct
  errorCategory ErrorSet_category optional;
  errors String < value
    { struct
      errorCode ErrorSet_code;
      errorInfo ErrorSet_Type; } > optional;
  method Check() -> Boolean
  {       dcl t1 ErrorSet_category;
          dcl p1 Boolean;
          p1 := this.errorCategoryPresent();
          if (p1 = true) { t1 := this.errorCategory; } else { return false; };
          if ((p1 = false) and (this.errorsPresent() =  true)) { return false; }
          loop (dcl i1 Integer := 1; i1<=length(errors); i1 := i1+1)
          {       dcl t2 ErrorSet_code, t3 ErrorSet_Type;
                  t2 := this.errors[i1].errorCode;
                  t3 := this.errors[i1].errorInfo ;
                  if (t1="A" and not( t2=1 or t2=2)) { return false; }
                  if (t1="B" and not( t2=1 or t2=2)) { return false; }
                  if (t1="A" and t2=1 and not (t3.integerPresent())) { return false; }
                  if (t1="A" and t2=2 and not (t3.realPresent())) { return false; }
                  if (t1="B" and t2=1 and not (t3.characterStringPresent())) { return false; }
                  if (t1="B" and t2=2 and not (t3.generalStringPresent())) { return false; }
           } /* end of loop */
          return true;
    } /* end of Check*/
} /* end of ErrorReturn */
```

9 Mapping of parameterized ASN.1 specifications

[ITU-T X.683] defines the way to parameterize ASN.1 specifications. All ASN.1 concepts are able to be parameterized. This feature allows the partial specification of types or values within an ASN.1 module with the specification being completed by the addition of the actual parameters at instantiation time.

Clause 8.3 of [ITU-T Z.102] defines an equivalent concept of formal context parameters.

9.1 Parameterized assignment

ASN.1 grammar

There are parameterized assignment statements corresponding to each of the assignment statements specified in [ITU-T X.680] and [ITU-T X.681].

The **ParameterizedAssignment**, **ParameterizedTypeAssignment**, **ParameterizedValueSetTypeAssignment**, **ParameterizedObjectClassAssignment**, **ParameterizedObjectAssignment**, **ParameterizedObjectSetAssignment** constructs are defined in clauses 8.1 and 8.2 of [ITU-T X.683].

Model

The use of all forms of **ParameterizedAssignment** is supported within ASN.1 modules.

ParameterizedTypeAssignment is mapped to SDL-2010 as defined in clause 9.2 relying on the SDL-2010 formal context parameters mechanisms.

ParameterizedValueSetTypeAssignment, **ParameterizedObjectClassAssignment**, **ParameterizedObjectAssignment**, **ParameterizedObjectSetAssignment** are usable in ASN.1 modules (so they are usable in other ASN.1 specifications), but they are not mapped to SDL-2010 themselves.

9.2 Parameterized type assignment

ASN.1 grammar

ParameterizedTypeAssignment and **ParameterList** are defined in clauses 8.2 and 8.3 of [ITU-T X.683].

Model

The difference between ordinary and parameterized ASN.1 types is that **ParameterList** follows the **typereference** and formal parameters contained in **ParameterList** are used in the **Type** definition.

A **Type** defined in ASN.1 using parameters from the **ParameterList** is mapped to the appropriate SDL-2010 type (as defined in clause 7.2.1) provided that ASN.1 parameters are either value or type parameters. Such parameters are mapped to <formal context parameters> of the SDL-2010 type. An ASN.1 type parameter is mapped to SDL-2010 <sort context parameter> (see clause 8.3.10 of [ITU-T Z.102]) and an ASN.1 value parameter is mapped to an SDL-2010 <synonym context parameter list> item (see clause 8.3.9 of [ITU-T Z.102]).

It is not possible to map ASN.1 parameterized types that have parameters other than types or values to SDL-2010 directly. However, if the parameters are expanded first into types or values, the resulting ASN.1 type or value is then mapped to SDL-2010 as defined in clause 9.3.

Example

The ASN.1-type definition:

```
TemplateMessage {INTEGER : minSize, INTEGER : maxSize, IndicatorType } ::=
SEQUENCE
{
     asp            INTEGER,
     pdu            OCTET STRING(SIZE(minSize..maxSize)),
     indicator      IndicatorType
}
```

is mapped to SDL-2010 type:

```
value type TemplateMessage
<       synonym minSize <<package Predefined>> Integer;
        synonym maxSize <<package Predefined>> Integer;
        value type IndicatorType        >
struct
{
        asp                     Integer;
        pdu                     <<package Predefined>> Octetstring (size(minSize:maxSize));
        indicator       IndicatorType;
}
```

9.3 Referencing ASN.1 parameterized type definitions

ASN.1 grammar

ParameterizedType, **ParameterizedValue** and **ActualParameterList** are defined in clauses 9.2, 9.2 and 9.5 of [ITU-T X.683], respectively.

Model

ParameterizedType and **ParameterizedValue** references are used in ASN.1 to define new ASN.1 types and values by providing an **ActualParameterList**.

If the **ParameterizedType** definition was such that it was possible to map it to SDL-2010 and the **ActualParameterList** contains only **Type** and **Value** parameters, then ASN.1 references to such definitions are mapped to SDL-2010 instantiations of the type with context parameters so that elements of **ActualParameterList** are mapped to <actual context parameters>. Example 1 illustrates one such mapping.

If, according to clause 9.2, the **ParameterizedType** definition could not be mapped to an SDL-2010-type definition with context parameters, it is possible to map to SDL-2010 the references to such a **ParameterizedType**. In this case, the meaning of the **ParameterizedType** definition is fully expanded within ASN.1 to the level of the types defined in clause 7 before a mapping to SDL-2010 is done.

If the **ActualParameterList** contains **ValueSet**, **DefinedObjectClass**, **Object** or **ObjectSet** the mapping of such a type to SDL-2010 is done so that the meaning of the type is fully expanded to the level of the types defined in clause 7 before a mapping to SDL-2010 is done. Example 2 illustrates one such mapping.

The ASN.1 types and values derived from referenced ASN.1 parameterized definitions are mapped to SDL-2010 as defined in clause 7.

Example 1

The parameterized type used in the example in clause 9.2 is used to define simple ASN.1 as follows:

```
ActualMessage ::= TemplateMessage{10, 20, BOOLEAN}
```

This is mapped to SDL-2010 type:

value type ActualMessage : TemplateMessage < 10, 20, <<**package** Predefined>> Boolean >

Example 2

What follows is an example of the ASN.1-type definition derived using a parameter that is an information object. The ASN.1 modules need to contain the relevant information object class definition with parameterized assignment having object of that class as dummy parameter, information object value assignment and parameterized type definition reference.

```
MESSAGE-PARAMETERS ::= CLASS {
     &maximum-priority-level          INTEGER,
```

```
    &maximum-message-buffer-size     INTEGER
}
WITH SYNTAX {
    THE MAXIMUM PRIORITY LEVEL IS         &maximum-priority-level
    THE MAXIMUM MESSAGE BUFFER SIZE IS    &maximum-message-buffer-size
}
Message-PDU { MESSAGE-PARAMETERS : param } ::= SEQUENCE {
    priority-level     INTEGER        (0..param.&maximum-priority-level),
    message        BMPString (SIZE    (0..param.&maximum-message-buffer-size))
}
my-message-parameters MESSAGE-PARAMETERS ::= {
    THE MAXIMUM PRIORITY LEVEL IS 10
    THE MAXIMUM MESSAGE BUFFER SIZE IS 2000
}
MY-Message-PDU ::= Message-PDU { my-message-parameters }
```

The semantics of ASN.1 is that with such definition of class, parameterized type definition and object value definition, the type resulting from transformation of MY-Message-PDU is equivalent to:

```
MY-Message-PDU ::=      SEQUENCE {
    priority-level      INTEGER   (0..10),
    message             BMPString (SIZE    (0..2000))
}
```

The resulting ASN.1 type is mapped to SDL-2010 type as:

value type MY_Message_PDU {

struct

priority_level <<**package** Predefined>> Integer (0..10);

message <<**package** Predefined>> BMPCharString (**size** (0..2000));

}

9.4 Referencing ASN.1 parameterized value definitions

ASN.1 grammar

ParameterizedType, **ParameterizedValue** and **ActualParameterList** are defined in clauses 9.2, 9.2 and 9.5 of [ITU-T X.683], respectively.

Model

ParameterizedValue references are used in ASN.1 to define new ASN.1 values by providing an **ActualParameterList**.

ParameterizedValue references are mapped to SDL-2010 in such a way that the meaning of such a value specification is fully expanded to the level of value assignments defined in clause 7 before a mapping to SDL-2010 is done.

9.5 Referencing other ASN.1 parameterized definitions

ParameterizedValueSetType, **ParameterizedObjectClass**, **ParameterizedObjectSet** and **ParameterizedObject** are defined in clause 9.2 of [ITU-T X.683].

ANS.1 modules that contain the specification of value sets, object classes, object sets and objects defined by referencing the **SimpleDefinedType** with **ActualParameterList** are not mapped to SDL-2010.

10 Definitions in package `Predefined` for SDL-2010

In SDL-2010 the following items are defined in the **`package`** `Predefined and` support the combination of ASN.1 modules with SDL-2010:

`syntype` NumericChar,
`value type` NumericString,
`syntype` PrintableChar,
`value type` PrintableString,
`syntype` TeletexChar,
`syntype` VideotexChar,
`value type` VideotexString,
`syntype` IA5Char,
`syntype` IA5String,
`value type` GeneralChar,
`value type` UniversalChar,
`value type` UniversalCharString,
`syntype` UTF8String,
`value type` GeneralCharString,
`syntype` GraphicChar,
`syntype` VisibleChar,
`value type` VisibleString,
`syntype` BMPChar,
`value type` BMPCharString, and
`value type` NULL.

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	Terminals and subjective and objective assessment methods
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, 2012