INTERNATIONAL STANDARD

ISO/IEC 8824-4

Sixth edition
2021-06

Information technology — Abstract Syntax Notation One (ASN.1) —

Part 4: Parameterization of ASN.1 specifications

Technologies de l'information — Notation de syntaxe abstraite numéro un (ASN.1) —

Partie 4: Paramétrage des spécifications de la notation de syntaxe abstraite numéro un

Reference number
ISO/IEC 8824-4:2021(E)

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives or www.iec.ch/members_experts/refdocs)

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents) or the IEC list of patent declarations received (see patents.iec.ch).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see www.iso.org/iso/foreword.html. In the IEC, see www.iec.ch/understanding-standards.

This document was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 6, *Telecommunications and information exchange between systems,* in collaboration with ITU-T. The identical text is published as ITU-T X.683 (02/2021).

This sixth edition cancels and replaces the fifth edition (ISO/IEC 8824-4:2015), which has been technically revised. It also incorporates ISO/IEC 8824-4:2015/Cor 1:2018.

A list of all parts in the ISO/IEC 8824 series can be found on the ISO and IEC websites.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html and www.iec.ch/national-committees.

CONTENTS

Introduction

Application designers need to write specifications in which certain aspects are left undefined. Those aspects will later be defined by one or more other groups (each in its own way), to produce a fully defined specification for use in the definition of an abstract syntax (one for each group).

In some cases, aspects of the specification (for example, bounds) may be left undefined even at the time of abstract syntax definition, being completed by the specification of International Standardized Profiles or functional profiles from some other body.

NOTE 1 – It is a requirement imposed by this Recommendation | International Standard that any aspect that is not solely concerned with the application of constraints has to be completed prior to the definition of an abstract syntax.

In the extreme case, some aspects of the specification may be left for the implementer to complete, and would then be specified as part of the Protocol Implementation Conformance Statement.

While the provisions of Rec. ITU-T X.681 | ISO/IEC 8824-2 and Rec. ITU-T X.682 | ISO/IEC 8824-3 provide a framework for the later completion of parts of a specification, they do not of themselves solve the above requirements.

Additionally, a single designer sometimes requires to define many types, or many information object classes, or many information object sets, or many information objects, or many values, which have the same outer level structure, but differ in the types, or information object classes, or information object sets, or information objects, or values, that are used at an inner level. Instead of writing out the outer level structure for every such occurrence, it is useful to be able to write it out once, with parts left to be defined later, then to refer to it and provide the additional information.

All these requirements are met by the provision for parameterized reference names and parameterized assignments by this Recommendation | International Standard.

The syntactic form of a parameterized reference name is the same as that of the corresponding normal reference name, but the following additional considerations apply:

– When it is assigned in a parameterized assignment statement, it is followed by a list of dummy reference names in braces, each possibly accompanied by a governor; these reference names have a scope which is the right-hand side of the assignment statement, and the parameter list itself.

  NOTE 2 – This is what causes it to be recognized as a parameterized reference name.

– When it is exported or imported, it is followed by a pair of empty braces to distinguish it as a parameterized reference name.

– When it is used in any construct, it is followed by a list of syntactic constructions, one for each dummy reference name, that provide an assignment to the dummy reference name for the purposes of that use only.

Dummy reference names have the same syntactic form as the corresponding normal reference name, and can be used anywhere on the right-hand side of the assignment statement that the corresponding normal reference name could be used. All such usages are required to be consistent.

INTERNATIONAL STANDARD
ITU-T RECOMMENDATION

Information Technology – Abstract Syntax Notation ONE (ASN.1): Parameterization of ASN.1 specifications

1 Scope

This Recommendation | International Standard is part of Abstract Syntax Notation One (ASN.1) and defines notation for parameterization of ASN.1 specifications.

2 Normative references

The following Recommendations and International Standards contain provisions which, through reference in this text, constitute provisions of this Recommendation | International Standard. At the time of publication, the editions indicated were valid. All Recommendations and Standards are subject to revision, and parties to agreements based on this Recommendation | International Standard are encouraged to investigate the possibility of applying the most recent edition of the Recommendations and Standards listed below. Members of IEC and ISO maintain registers of currently valid International Standards. The Telecommunication Standardization Bureau of the ITU maintains a list of currently valid ITU-T Recommendations.

2.1 Identical Recommendations | International Standards

- Recommendation ITU-T X.680 (2021) | ISO/IEC 8824-1:2021, *Information technology – Abstract Syntax Notation One (ASN.1): Specification of basic notation.*
- Recommendation ITU-T X.681 (2021) | ISO/IEC 8824-2:2021, *Information technology – Abstract Syntax Notation One (ASN.1): Information object specification.*
- Recommendation ITU-T X.682 (2021) | ISO/IEC 8824-3:2021, *Information technology – Abstract Syntax Notation One (ASN.1): Constraint specification.*

NOTE – The references above shall be interpreted as references to the identified Recommendations | International Standards together with all their published amendments and technical corrigenda.

3 Definitions

For the purposes of this Recommendation | International Standard, the following definitions apply.

3.1 Specification of basic notation

This Recommendation | International Standard uses the terms defined in Rec. ITU-T X.680 | ISO/IEC 8824-1.

3.2 Information object specification

This Recommendation | International Standard uses the terms defined in Rec. ITU-T X.681 | ISO/IEC 8824-2.

3.3 Constraint specification

This Recommendation | International Standard uses the terms defined in Rec. ITU-T X.682 | ISO/IEC 8824-3.

3.4 Additional definitions

3.4.1 **normal reference name**: A reference name defined, without parameters, by means of an "Assignment" other than a "ParameterizedAssignment". Such a name references a complete definition and is not supplied with actual parameters when used.

3.4.2 **parameterized reference name**: A reference name defined using a parameterized assignment, which references an incomplete definition and which, therefore, must be supplied with actual parameters when used.

3.4.3 **parameterized type**: A type defined using a parameterized type assignment and thus whose components are incomplete definitions which must be supplied with actual parameters when the type is used.

3.4.4 **parameterized value**: A value defined using a parameterized value assignment and thus whose value is incompletely specified and must be supplied with actual parameters when used.

3.4.5 **parameterized value set**: A value set defined using a parameterized value set assignment and thus whose values are incompletely specified and must be supplied with actual parameters when used.

3.4.6 **parameterized object class**: An information object class defined using a parameterized object class assignment and thus whose field specifications are incompletely specified and must be supplied with actual parameters when used.

3.4.7 **parameterized object**: An information object defined using a parameterized object assignment and thus whose components are incompletely specified and must be supplied with actual parameters when used.

3.4.8 **parameterized object set**: An information object set defined using a parameterized object set assignment and thus whose objects are incompletely specified and must be supplied with actual parameters when used.

3.4.9 **variable constraint**: A constraint employed in specifying a parameterized abstract syntax, and which depends on some parameter of the abstract syntax.

4 Abbreviations

For the purposes of this Recommendation | International Standard, the following abbreviation applies:

ASN.1 Abstract Syntax Notation One

5 Convention

This Recommendation | International Standard employs the notational convention defined in Rec. ITU-T X.680 | ISO/IEC 8824-1, clause 5.

6 Notation

This clause summarizes the notation defined in this Recommendation | International Standard.

6.1 Assignments

The following notation which can be used as an alternative for "Assignment" (see Rec. ITU-T X.680 | ISO/IEC 8824-1, clause 13) is defined in this Recommendation | International Standard:

– ParameterizedAssignment (see 8.1).

6.2 Parameterized definitions

6.2.1 The following notation which can be used as an alternative for "DefinedType" (see Rec. ITU-T X.680 | ISO/IEC 8824-1, 14.1) is defined in this Recommendation | International Standard:

– ParameterizedType (see 9.2).

6.2.2 The following notation which can be used as an alternative for "DefinedValue" (see Rec. ITU-T X.680 | ISO/IEC 8824-1, 14.1) is defined in this Recommendation | International Standard:

– ParameterizedValue (see 9.2).

6.2.3 The following notation which can be used as an alternative for "DefinedType" (see Rec. ITU-T X.680 | ISO/IEC 8824-1, 14.1) is defined in this Recommendation | International Standard:

– ParameterizedValueSetType (see 9.2).

6.2.4 The following notation which can be used as an alternative for "ObjectClass" (see Rec. ITU-T X.681 | ISO/IEC 8824-2, 9.2) is defined in this Recommendation | International Standard:

– ParameterizedObjectClass (see 9.2).

6.2.5 The following notation which can be used as an alternative for "Object" (see Rec. ITU-T X.681 | ISO/IEC 8824-2, 11.3) is defined in this Recommendation | International Standard:

– ParameterizedObject (see 9.2).

6.2.6 The following notation which can be used as an alternative for "ObjectSet" (see Rec. ITU-T X.681 | ISO/IEC 8824-2, 12.3) is defined in this Recommendation | International Standard:

– ParameterizedObjectSet (see 9.2).

6.3 Symbols

The following notation which can be used as an alternative for "Symbol" (see Rec. ITU-T X.680 | ISO/IEC 8824-1, 13.1) is defined in this Recommendation | International Standard:

– ParameterizedReference (see 9.1).

7 ASN.1 lexical items

This Recommendation | International Standard makes use of the lexical items specified in Rec. ITU-T X.680 | ISO/IEC 8824-1, clause 12.

8 Parameterized assignments

8.1 There are parameterized assignment statements corresponding to each of the assignment statements specified in Rec. ITU-T X.680 | ISO/IEC 8824-1 and Rec. ITU-T X.681 | ISO/IEC 8824-2. The "ParameterizedAssignment" construct is:

ParameterizedAssignment ::=
ParameterizedTypeAssignment
| ParameterizedValueAssignment
| ParameterizedValueSetTypeAssignment
| ParameterizedObjectClassAssignment
| ParameterizedObjectAssignment
| ParameterizedObjectSetAssignment

8.2 Each "Parameterized<X>Assignment" has the same syntax as "<X>Assignment" except that following the initial lexical item there is a "ParameterList". The initial item thereby becomes a parameterized reference name (see 3.4.2):

NOTE 1 – Rec. ITU-T X.680 | ISO/IEC 8824-1 imposes the requirement that all reference names assigned within a module, whether parameterized or not, must be distinct.

NOTE 2 – Where value notation is governed by a parameterized type (or a type that is a parameter) the validity of value notation within the parameterized assignment can only be determined after instantiation of the parameterized type, and may be valid for some instantiations and invalid for others.

ParameterizedTypeAssignment ::=
typereference
ParameterList
" : :="
Type

ParameterizedValueAssignment ::=
valuereference
ParameterList
Type
" : :="
Value

ParameterizedValueSetTypeAssignment ::=
typereference
ParameterList
Type
" : :="
ValueSet

ParameterizedObjectClassAssignment ::=
objectclassreference
ParameterList
" : :="
ObjectClass

ParameterizedObjectAssignment ::=
objectreference

ParameterList
DefinedObjectClass
":::="
Object

ParameterizedObjectSetAssignment ::=
objectsetreference
ParameterList
DefinedObjectClass
":::="
ObjectSet

8.3 A "ParameterList" is a list of "Parameter"s between braces:

ParameterList ::= "{" Parameter "," + "}"

Each "Parameter" consists of a "DummyReference" and possibly a "ParamGovernor":

Parameter ::= ParamGovernor ":" DummyReference | DummyReference

ParamGovernor ::= Governor | DummyGovernor

Governor ::= Type | DefinedObjectClass

DummyGovernor ::= DummyReference

DummyReference ::= Reference

A "DummyReference" in "Parameter" may stand for:

a) a "Type" or "DefinedObjectClass", in which case there shall be no "ParamGovernor";

b) a "Value" or "ValueSet", in which case the "ParamGovernor" shall be present, and in case "ParamGovernor" is a "Governor" it shall be a "Type", and in case "ParamGovernor" is a "DummyGovernor" the actual parameter for the "ParamGovernor" shall be a "Type";

c) an "Object" or "ObjectSet", in which case the "ParamGovernor" shall be present, and in case "ParamGovernor" is a "Governor" it shall be a "DefinedObjectClass", and in case "ParamGovernor" is a "DummyGovernor" the actual parameter for the "ParamGovernor" shall be a "DefinedObjectClass".

A "DummyGovernor" shall be a "DummyReference" that has no "Governor".

8.4 The scope of a "DummyReference" appearing in a "ParameterList" is the "ParameterList" itself, together with that part of the "ParameterizedAssignment" which follows the "ParameterList". The "DummyReference" hides any other "Reference" with the same name in that scope in any given instantiation.

NOTE – This subclause does not apply to "identifier"s defined in "NamedNumberList"s, "Enumeration"s and "NamedBitList"s, since they are not "Reference"s. The "DummyReference" does not hide these "identifier"s (see Rec. ITU-T X.680 | ISO/IEC 8824-1, 19.12 and 20.11).

8.5 The usage of a "DummyReference" within its scope shall be consistent with its syntactic form, and, where applicable, governor, and all usages of the same "DummyReference" shall be consistent with one another.

NOTE – Where the syntactic form of a dummy reference name is ambiguous (for example, between whether it is an "objectclassreference" or "typereference"), the ambiguity can normally be resolved on the first use of the dummy reference name on the right-hand side of the assignment statement. Thereafter, the nature of the dummy reference name is known. The nature of the dummy reference is, however, not determined solely by the right-hand side of the assignment statement when it is in turn used only as an actual parameter in a parameterized reference; in this case, the nature of the dummy reference must be determined by examining the definition of this parameterized reference. Users of the notation are warned that such a practice can make ASN.1 specifications less clear, and it is suggested that adequate comments are provided to explain this for human readers.

Example

Consider the following parameterized object class assignment:

```
PARAMETERIZED-OBJECT-CLASS { TypeParam, INTEGER:valueParam, INTEGER:ValueSetParam } ::=
    CLASS {
    &valueField1        TypeParam,
    &valueField2        INTEGER DEFAULT valueParam,
    &valueField3        INTEGER (ValueSetParam),
    &ValueSetField      INTEGER DEFAULT { ValueSetParam }
    }
```

For the purpose of determining proper usage of the "DummyReference"s in the scope of the "Parameterized Assignment", and for that purpose only, the "DummyReference"s can be regarded to be defined as follows:

TypeParam ::= UnspecifiedType

valueParam INTEGER ::= unspecifiedIntegerValue

ValueSetParam INTEGER ::= { UnspecifiedIntegerValueSet }

where:

a) **`TypeParam`** is a "DummyReference" which stands for a "Type". Therefore **`TypeParam`** can be used wherever a "typereference" can be used, e.g. as a "Type" for the fixed-type value field **`valueField1`**.

b) **`valueParam`** is a "DummyReference" which stands for a value of an integer type. Therefore **`valueParam`** can be used wherever a "valuereference" of an integer value can be used, e.g. as a default value for the fixed-type value field **`valueField2`**.

c) **`ValueSetParam`** is a "DummyReference" which stands for a value set of an integer type. Therefore **`ValueSetParam`** can be used wherever a "typereference" of an integer value can be used, e.g. as a "Type" in the "ContainedSubtype" notation for **`valueField3`** and **`ValueSetField`**.

8.6 Each "DummyReference" shall be employed at least once within its scope.

NOTE – If the "DummyReference" did not so appear, then the corresponding "ActualParameter" would have no effect on the definition, and would simply be "discarded", while to the user it might seem that some specification was taking place.

"ParameterizedValueAssignment"s, "ParameterizedValueSetTypeAssignment"s, "ParameterizedObjectAssignment"s and "ParameterizedObjectSetAssignment"s that contain either a direct or indirect reference to themselves are illegal.

8.7 Given a set of parameterized assignments (one or more) and a recursive path of parameterized references crossing one or more of those parameterized assignments, all the parameterized references along that path shall satisfy the following condition: Each actual parameter present in the parameterized reference shall either consist solely of a dummy reference (with no lexical items or comments preceding or following the dummy reference) or shall not contain any dummy reference (see A.3).

8.8 In the definition of a "ParameterizedType", "ParameterizedValueSet", or "ParameterizedObjectClass", a circular reference to the item being defined shall not be made, unless such reference is directly or indirectly marked **`OPTIONAL`** or, in the case of "ParameterizedType" and "ParameterizedValueSet", made through a reference to a choice type, at least one of whose alternatives is non-circular in definition.

8.9 The governor of a "DummyReference" shall not include a reference to another "DummyReference" if that other "DummyReference" also has a governor.

8.10 In a parameterized assignment the right side of the "**`::=`**" shall not consist solely of a "DummyReference".

8.11 The governor of a "DummyReference" shall not require knowledge of either the "DummyReference" or of the parameterized reference name being defined.

8.12 When a value or value set is supplied to a parameterized type as an actual parameter, the type of the actual parameter is required to be compatible with the governor of the corresponding dummy parameter. (See Rec. ITU-T X.680 | ISO/IEC 8824-1, C.6.2 and C.6.3 for details.)

8.13 In defining a parameterized type with a value or a value set dummy parameter, the type used to govern that dummy parameter shall be a type, all of whose values are valid for use in all places to the right of the assignment where the dummy parameter is used. (See Rec. ITU-T X.680 | ISO/IEC 8824-1, C.6.5 for details.)

9 Referencing parameterized definitions

9.1 Within a "SymbolList" (in "Exports" or "Imports") a parameterized definition shall be referenced by a "ParameterizedReference":

ParameterizedReference ::= Reference | Reference "`{`" "`}`"

where "Reference" is the first lexical item in the "ParameterizedAssignment", as specified in 8.2 above.

NOTE – The first alternative of "ParameterizedReference" is provided solely as an aid to human understanding. Both alternatives have the same meaning.

9.2 Other than in "Exports" or "Imports", a parameterized definition shall be referenced by a "Parameterized<X>" construct, which can be used as an alternative for the corresponding "<X>":

ParameterizedType ::=
SimpleDefinedType
ActualParameterList

SimpleDefinedType ::=
ExternalTypeReference |
typereference

ParameterizedValue ::=
SimpleDefinedValue
ActualParameterList

SimpleDefinedValue ::=
ExternalValueReference |
valuereference

ParameterizedValueSetType ::=
SimpleDefinedType
ActualParameterList

ParameterizedObjectClass ::=
DefinedObjectClass
ActualParameterList

ParameterizedObjectSet ::=
DefinedObjectSet
ActualParameterList

ParameterizedObject ::=
DefinedObject
ActualParameterList

9.3 The reference name in the "Defined<X>" shall be a reference name to which an assignment is made in a "ParameterizedAssignment".

9.4 The restrictions on the "Defined<X>" alternative to be used, which are specified in Rec. ITU-T X.680 | ISO/IEC 8824-1 and Rec. ITU-T X.681 | ISO/IEC 8824-2 as normal reference names, apply equally to the corresponding parameterized reference names.

NOTE – In essence, the restrictions are as follows: each "Defined<X>" has two alternatives, "<x>reference" and "External<x>Reference". The former is used within the module of definition or if the definition has been imported and there is no name conflict; the latter is used where there is no imports listed (deprecated), or if there is a conflict between the imported name and a local definition (also deprecated) or between imports.

9.5 The "ActualParameterList" is:

ActualParameterList ::=
"{" ActualParameter "," + "}"

ActualParameter ::=
Type
| Value
| ValueSet
| DefinedObjectClass
| Object
| ObjectSet

9.6 There shall be exactly one "ActualParameter" for each "Parameter" in the corresponding "ParameterizedAssignment" and they shall appear in the same order. The particular choice of "ActualParameter", and the governor (if any) shall be determined by examination of the syntactic form of the "Parameter" and the environment in which it occurs in the "ParameterizedAssignment". The form of the "ActualParameter" shall be the form required to replace the "DummyReference" everywhere in its scope (see 8.4).

Example

The parameterized object class definition of the previous example (see 8.5) can be referenced, for instance, as follows:

```
MY-OBJECT-CLASS ::= PARAMETERIZED-OBJECT-CLASS { BIT STRING, 123, {4 | 5 | 6} }
```

9.7 The actual parameter takes the place of the dummy reference name in determining the actual type, value, value set, object class, object, or object set that is being referenced by this instance of use of the parameterized reference name.

9.8 The meaning of any references which appear in the "ActualParameter", and the tag default applicable to any tags which so appear, are determined according to the tagging environment of the "ActualParameter" rather than that of the corresponding "DummyReference".

NOTE – Thus, parameterization, like referencing, selection types, and **COMPONENTS OF**, among others, is not exactly textual substitution.

Example

Consider the following modules:

```
M1 DEFINITIONS AUTOMATIC TAGS ::= BEGIN
        EXPORTS T1;

        T1 ::= SET {
        f1    INTEGER,
        f2    BOOLEAN
}

END

M2 DEFINITIONS EXPLICIT TAGS ::= BEGIN
        IMPORTS T1 FROM  M1;

        T3 ::= T2{T1}

        T2{X} ::= SEQUENCE {
        a     INTEGER,
        b     X
}

END
```

Application of 9.8 implies that the tag for the component **f1** of **T3** (i.e. @T3.b.f1) will be implicitly tagged because the tagging environment of the dummy parameter **X**, namely explicit tagging, does not affect the tagging of the components of the actual parameter **T1**.

Consider the module **M3**:

```
M3 DEFINITIONS AUTOMATIC TAGS ::= BEGIN
        IMPORTS T1 FROM  M1;

        T5 ::= T4{T1}

        T4{Y} ::= SEQUENCE {
        a     INTEGER,
        b     Y
}

END
```

Application of Rec. ITU-T X.680 | ISO/IEC 8824-1, 31.2.7, implies that the tag for the component **b** of **T5** (i.e. @T5.b) will be explicitly tagged because the dummy parameter **Y** is always explicitly tagged, hence **T5** is equivalent to:

```
T5 ::= SEQUENCE {
        a     [0] IMPLICIT INTEGER,
        b     [1] EXPLICIT SET {
        f1    [0] INTEGER,
        f2    [1] BOOLEAN
        }
}
```

while **T3** is equivalent to:

```
T3 ::= SEQUENCE {
        a     INTEGER,
        b     SET {
        f1    [0] IMPLICIT INTEGER,
        f2    [1] IMPLICIT BOOLEAN
        }
}
```

10 Abstract syntax parameters

10.1 Rec. ITU-T X.681 | ISO/IEC 8824-2, Annex B, provides the **ABSTRACT-SYNTAX** information object class and recommends its use to define abstract syntaxes, using as an example an abstract syntax defined as the set of values of a single ASN.1 type which was not parameterized at the outer level.

10.2 Where the ASN.1 type used to define the abstract syntax *is* parameterized, some parameters may be supplied as actual parameters when the abstract syntax is defined, while others may be left as parameters of the abstract syntax itself.

Example

If a parameterized type has been defined called **YYY-PDU** with two dummy references (the first an object set of some defined object class, and the second an integer value for a bound, say), then:

```
yyy-Abstract-Syntax { INTEGER:bound } ABSTRACT-SYNTAX ::=
        { YYY-PDU { {ValidObjects} , bound } IDENTIFIED BY {yyy 5} }
```

defines a parameterized abstract syntax in which the object set has been resolved, but **bound** remains as a parameter of the abstract syntax.

An abstract syntax parameter shall be used:

a) directly or indirectly in the context of a constraint;

b) directly or indirectly as actual parameters that eventually are used in the context of a constraint.

NOTE – See the example in A.2, and the example in Rec. ITU-T X.680 | ISO/IEC 8824-1, H.5.

10.3 A constraint whose value set depends on one or more parameters of the abstract syntax is a variable constraint. Such constraints are determined after the definition of the abstract syntax (perhaps by International Standardized Profiles or in Protocol Implementation Conformance Statements).

NOTE – If somewhere in the chain of definitions involved in the specification of the constraint values a parameter of the abstract syntax appears, the constraint is a variable constraint. It is a variable constraint even if the value set of the resulting constraint is independent of the actual value of the parameter of the abstract syntax.

Example

The value of **(((1..3) EXCEPT a) UNION (1 .. 3))** is always 1..3 no matter what the value of **a** is, nonetheless it is still a variable constraint if **a** is a parameter of the abstract syntax.

10.4 Formally, a variable constraint does not constrain the set of values in the abstract syntax.

NOTE – It is strongly recommended that constraints that are expected to remain as variable constraints in an abstract syntax have an exception specification using the notation provided by Rec. ITU-T X.680 | ISO/IEC 8824-1, 53.4.

Annex A

Examples

(This annex does not form an integral part of this Recommendation | International Standard.)

A.1 Example of the use of a parameterized type definition

Suppose that a protocol designer frequently needs to carry an authenticator with one or more of the fields of the protocol. This will be carried as a **BIT STRING**, alongside the field. Without parameterization, **Authenticator** would need to be defined as a **BIT STRING**, then **authenticator** would need to be added wherever it was to appear, with text to identify what it applied to. Alternatively, the designer could adopt the discipline of turning any field that has an authenticator into a **SEQUENCE** of that field and **authenticator**. The parameterization mechanism provides a convenient short-hand for doing this task.

First we define the parameterized type **SIGNED{}**:

```
SIGNED { ToBeSigned } ::= SEQUENCE
{
        authenticated-data   ToBeSigned,
        authenticator        BIT STRING
}
```

then, in the body of the protocol, the notation (for example):

```
SIGNED { OrderInformation }
```

is a type notation standing for:

```
SEQUENCE
{
        authenticated-data   OrderInformation,
        authenticator        BIT STRING
}
```

Suppose further that for some fields, the sender is to have the option of adding the authenticator or not. This could be achieved by making the **BIT STRING** optional, but a more elegant solution (less bits on the line) would be to define another parameterized type:

```
OPTIONALLY-SIGNED {ToBeSigned} ::= CHOICE
{
        unsigned-data  [0]   ToBeSigned,
        signed-data          [1]   SIGNED { ToBeSigned }
}
```

NOTE – The tagging in the **CHOICE** is not necessary if the writer ensures that none of the uses of the parameterized type produce an actual argument which is a **BIT STRING** (the type of **SIGNED**), but is useful in preventing errors in other parts of the specification.

A.2 Example of use of parameterized definitions together with an information object class

Use information object classes to collect all the parameters for an abstract syntax. In that way the number of parameters for an abstract syntax can be reduced to one which is an instance of the collection class. The "InformationFromObject" production can be used to extract information from the parameter object.

Example

```
-- An instance of this class contains all the parameters for the abstract
-- syntax, Message-PDU.

MESSAGE-PARAMETERS ::= CLASS {
        &maximum-priority-level                 INTEGER,
        &maximum-message-buffer-size            INTEGER,
        &maximum-reference-buffer-size  INTEGER
}
WITH SYNTAX {
        THE MAXIMUM PRIORITY LEVEL IS                   &maximum-priority-level
        THE MAXIMUM MESSAGE BUFFER SIZE IS              &maximum-message-buffer-size
        THE MAXIMUM REFERENCE BUFFER SIZE IS            &maximum-reference-buffer-size
}
        -- The "ValueFromObject" production is used to extract values
        -- from the abstract syntax parameter, "param". The values can be
```

```
        -- used only in constraints. In addition the parameter is passed
        -- through to another parameterized type.

Message-PDU { MESSAGE-PARAMETERS : param } ::= SEQUENCE {
priority-level   INTEGER (0..param.&maximum-priority-level),
message          BMPString (SIZE (0..param.&maximum-message-buffer-size)),
reference        Reference { param }
}

Reference { MESSAGE-PARAMETERS : param } ::=
        SEQUENCE OF IA5String (SIZE (0..param.&maximum-reference-buffer-size))
        -- Definition of a parameterized abstract syntax information object.
        -- The abstract syntax parameter is used only in constraints.

message-Abstract-Syntax { MESSAGE-PARAMETERS : param }
ABSTRACT-SYNTAX ::=
{
        Message-PDU { param }
        IDENTIFIED BY { joint-iso-itu-t example(999) 0 }
}
```

The class **`MESSAGE-PARAMETERS`** and the parameterized abstract syntax object, **`message-Abstract-Syntax`**, are used as follows:

```
        -- This instance of MESSAGE-PARAMETERS defines parameter values
        -- for the abstract syntax.

my-message-parameters MESSAGE-PARAMETERS ::= {
        THE MAXIMUM PRIORITY LEVEL IS 10
        THE MAXIMUM MESSAGE BUFFER SIZE IS  2000
        THE MAXIMUM REFERENCE BUFFER SIZE IS 100
}

        -- The abstract syntax can now be defined with all variable constraints specified.

my-message-Abstract-Syntax ABSTRACT-SYNTAX ::=
        message-Abstract-Syntax { my-message-parameters }
```

A.3 Example of parameterized type definition that is finite

When specifying a parameterized type which represents a generic list, specify the type so that the resulting ASN.1 notation is finite. For example, we may specify:

```
List1 { ElementTypeParam } ::= SEQUENCE {
        elem ElementTypeParam,
        next List1 { ElementTypeParam } OPTIONAL
}
```

which is finite, for when it is used:

```
IntegerList1 ::= List1 { INTEGER }
```

the resulting ASN.1 notation is as you would normally define it:

```
IntegerList1 ::= SEQUENCE {
        elem INTEGER,
        next IntegerList1 OPTIONAL
}
```

Contrast this to the following:

```
List2 { ElementTypeParam } ::= SEQUENCE {
        elem ElementTypeParam,
        next List2 { [0] ElementTypeParam } OPTIONAL
}


IntegerList2 ::= List2 { INTEGER }
```

where the resulting ASN.1 notation is infinite:

```
IntegerList2 ::= SEQUENCE {
elem        INTEGER,
next        SEQUENCE {
            elem [0] INTEGER,
            next SEQUENCE {
                  elem       [0][0] INTEGER,
                  next             SEQUENCE {
                        elem [0][0][0]    INTEGER,
                        next              SEQUENCE {
                                    -- and so on
                        } OPTIONAL
                  } OPTIONAL
            } OPTIONAL
} OPTIONAL
      }
```

A.4 Example of a parameterized value definition

If a parameterized string value is defined as follows:

```
genericBirthdayGreeting { IA5String : name } IA5String ::= { "Happy birthday, ", name, "!!" }
```

then the following two string values are the same:

```
greeting1 IA5String ::= genericBirthdayGreeting { "John" }
greeting2 IA5String ::= "Happy birthday, John!!"
```

A.5 Example of a parameterized value set definition

If two parameterized value sets are defined as follows:

```
QuestList1 {IA5String : extraQuest} IA5String ::= { "Jack" | "John" | extraQuest }
QuestList2 {IA5String : ExtraQuests} IA5String ::= { "Jack" | "John" | ExtraQuests }
```

then the following value sets denote the same value set:

```
SetOfQuests1 IA5String ::= { QuestList1 { "Jill" } }
SetOfQuests2 IA5String ::= { QuestList2 { {"Jill"} } }
SetOfQuests3 IA5String ::= { "Jack" | "John" | "Jill" }
```

and the following value sets denote the same value set:

```
SetOfQuests4 IA5String ::= { QuestList2 { {"Jill" | "Mary"} } }
SetOfQuests5 IA5String ::= { "Jack" | "John" | "Jill" | "Mary"}
```

Notice that a value set is *always* specified within braces, even when it is a parameterized value set reference. By omitting the braces from a reference to an "identifier" that was created in a value set assignment or from a reference to a "ParameterizedValueSetType" the notation is that of a "Type", not a value set.

A.6 Example of a parameterized class definition

The following parameterized class can be used to define error classes which contain error codes of different types. Note that the **`ErrorCodeType`** parameter is used only as a "DummyGovernor" for the **`ValidErrorCodes`** parameter:

```
GENERIC-ERROR { ErrorCodeType, ErrorCodeType : ValidErrorCodes } ::= CLASS {
            &errorCode      ValidErrorCodes
}
WITH SYNTAX {
            CODE &errorCode
}
```

The parameterized class definition can be used as follows to define different classes which share some characteristics like the same defined syntax:

```
ERROR-1 ::= GENERIC-ERROR { INTEGER, { 1 | 2 | 3 } }

ERROR-2 ::= GENERIC-ERROR { ErrorCodeString, { StringErrorCodes } }

ERROR-3 ::= GENERIC-ERROR { EnumeratedErrorCode, { fatal | error } }

ErrorCodeString ::= IA5String (SIZE (4))
```