INTERNATIONAL TELECOMMUNICATION UNION

ITU-T X.681

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

(07/94)

DATA NETWORKS AND OPEN SYSTEM COMMUNICATIONS

OSI NETWORKING AND SYSTEM ASPECTS – ABSTRACT SYNTAX NOTATION ONE (ASN.1)

INFORMATION TECHNOLOGY – ABSTRACT SYNTAX NOTATION ONE (ASN.1): INFORMATION OBJECT SPECIFICATION

ITU-T Recommendation X.681
Superseded by a more recent version

(Previously "CCITT Recommendation")

Foreword

ITU (International Telecommunication Union) is the United Nations Specialized Agency in the field of telecommunications. The ITU Telecommunication Standardization Sector (ITU-T) is a permanent organ of the ITU. Some 179 member countries, 84 telecom operating entities, 145 scientific and industrial organizations and 38 international organizations participate in ITU-T which is the body which sets world telecommunications standards (Recommendations).

The approval of Recommendations by the Members of ITU-T is covered by the procedure laid down in WTSC Resolution No. 1 (Helsinki, 1993). In addition, the World Telecommunication Standardization Conference (WTSC), which meets every four years, approves Recommendations submitted to it and establishes the study programme for the following period.

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. The text of ITU-T Recommendation X.681 was approved on the 1st of July 1994. The identical text is also published as ISO/IEC International Standard 8824-2.

NOTE

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

Superseded by a more recent version

ITU-T X-SERIES RECOMMENDATIONS

DATA NETWORKS AND OPEN SYSTEM COMMUNICATIONS

(February 1994)

ORGANIZATION OF X-SERIES RECOMMENDATIONS

Subject area	Recommendation Series
PUBLIC DATA NETWORKS	
Services and Facilities	X.1-X.19
Interfaces	X.20-X.49
Transmission, Signalling and Switching	X.50-X.89
Network Aspects	X.90-X.149
Maintenance	X.150-X.179
Administrative Arrangements	X.180-X.199
OPEN SYSTEMS INTERCONNECTION	
Model and Notation	X.200-X.209
Service Definitions	X.210-X.219
Connection-mode Protocol Specifications	X.220-X.229
Connectionless-mode Protocol Specifications	X.230-X.239
PICS Proformas	X.240-X.259
Protocol Identification	X.260-X.269
Security Protocols	X.270-X.279
Layer Managed Objects	X.280-X.289
Conformance Testing	X.290-X.299
INTERWORKING BETWEEN NETWORKS	
General	X.300-X.349
Mobile Data Transmission Systems	X.350-X.369
Management	X.370-X.399
MESSAGE HANDLING SYSTEMS	X.400-X.499
DIRECTORY	X.500-X.599
OSI NETWORKING AND SYSTEM ASPECTS	
Networking	X.600-X.649
Naming, Addressing and Registration	X.650-X.679
Abstract Syntax Notation One (ASN.1)	X.680-X.699
OSI MANAGEMENT	X.700-X.799
SECURITY	X.800-X.849
OSI APPLICATIONS	
Commitment, Concurrency and Recovery	X.850-X.859
Transaction Processing	X.860-X.879
Remote Operations	X.880-X.899
OPEN DISTRIBUTED PROCESSING	X.900-X.999

Superseded by a more recent version

CONTENTS

Superseded by a more recent version

Summary

This Recommendation | International Standard provides the ASN.1 notation which 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.

Introduction

An application designer frequently needs to design a protocol which will work with any of a number of instances of some class of information objects, where instances of the class may be defined by a variety of other bodies, and may be added to over time. Examples of such information object classes are the "operations" of ROS and the "attributes" of the OSI Directory.

This Recommendation | International Standard provides notation which allows information object classes as well as individual information objects and information object sets thereof to be defined and given reference names.

An information object class is characterized by the kinds of fields possessed by its instances. A field may contain:

- an arbitrary type (a type field); or
- a single value of a specified type (a fixed-type value field); or
- a single value of a type specified in a (named) type field (a variable-type value field);
- a non-empty set of values of a specified type (a fixed-type value set field); or
- a non-empty set of values of a type specified in a (named) type field (a variable-type value set field); or
- a single information object from a specified information object class (an object field);
- an information object set from a specified information object class (an object set field).

A fixed-type value field of an information object class may be selected to provide unique identification of information objects in that class. This is called the identifier field for that class. Values of the identifier field, if supplied, are required to be unique within any information object set that is defined for that class. They may, but need not, serve to unambiguously identify information objects of that class within some broader scope, particularly by the use of object identifier as the type of the identifier field.

An information object class is defined by specifying:

- the names of the fields;
- for each field, the form of that field (type, fixed-type value, variable-type value, fixed-type value set, variable-type value set, object, or object set);
- optionality and default settings of fields;
- which field, if any, is the identifier field.

An individual information object in the class is defined by providing the necessary information for each field.

The notation defined herein permits an ASN.1 type to be specified by reference to a field of some information object class – the object class field type. In ITU-T Rec. X.682 | ISO/IEC 8824-3, notation is provided to enable this type to be restricted by reference to some specific information object set.

It can be useful to consider the definition of an information object class as defining the form of an underlying conceptual table (the associated table) with one column for each field, and with a completed row defining an information object. The form of the table (determined by the information object class specification) determines the sort of information to be collected and used to complete some protocol specification. The underlying conceptual table provides the link between those specifying information objects of that class and the protocol which needs that information to complete its specification. Typically, the actual information object set used to complete a particular protocol specification will be a parameter of that protocol (see ITU-T Rec. X.683 | ISO/IEC 8824-4).

The "InformationFromObjects" notation referencing a specific object or object set (probably a parameter) can be used to extract information from cells of conceptual tables.

This Recommendation | International Standard:

- specifies a notation for defining an information object class, and for identifying it with a reference name (see clause 9);
- specifies a notation by which the definer of an information object class can provide a defined syntax for the definition of information objects of that class; a default notation is provided for classes for which no defined syntax has been defined (see clause 10);
- specifies a notation for defining an information object, and for assigning it to a reference name (see clause 11), and provides analogous notation for an object set (see clause 12);

- defines the "associated table" for an object or object set of a class (see clause 13);
- specifies notation for the object class field type and its values (see clause 14);

  NOTE – These constructs enable an ASN.1 type to be specified using a named field of a named information object class. Constraints on that type to restrict it to values related to a specific information object set appear in ITU-T Rec. X.682 | ISO/IEC 8824-3.
- specifies notation for extracting information from objects (see clause 15).

Annex A, which is an integral part of this Recommendation | International Standard, specifies the information object class whose object class reference is TYPE-IDENTIFIER. This is the simplest useful class, with just two fields, an identifier field of type object identifier, and a single type field which defines the ASN.1 type for carrying all information concerning any particular object in the class. It is defined herein because of the widespread use of information objects of this form.

Annex B, which is an integral part of this Recommendation | International Standard, specifies the notation for defining an abstract syntax (composed of the set of values of a single ASN.1 type) by the definition of an appropriate information object.

Annex C, which is an integral part of this Recommendation | International Standard, specifies the notation for the instance-of type (the "INSTANCE OF" notation), which is capable of carrying any value from any information object in a specified class (which must have been defined in terms of TYPE-IDENTIFIER).

Annex D, which is not an integral part of this Recommendation | International Standard, provides examples on how to use the notation described in this Recommendation | International Standard.

Annex E, which is not an integral part of this Recommendation | International Standard, provides a summary of the notation defined herein.

INTERNATIONAL STANDARD

ITU-T RECOMMENDATION

INFORMATION TECHNOLOGY – ABSTRACT SYNTAX NOTATION ONE (ASN.1): INFORMATION OBJECT SPECIFICATION

1 Scope

This Recommendation | International Standard is part of Abstract Syntax Notation One (ASN.1) and provides notation for specifying information object classes, information objects and information object sets.

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 editions of the Recommendations and Standards listed below. Members of IEC and ISO maintain registers of currently valid International Standards. The Telecommunications Standardization Bureau of the ITU maintains a list of currently valid ITU-T Recommendations.

2.1 Identical Recommendations | International Standards

- ITU-T Recommendation X.680 (1994) | ISO/IEC 8824-1:1995, *Information technology – Abstract Syntax Notation One (ASN.1): Specification of basic notation.*
- ITU-T Recommendation X.682 (1994) | ISO/IEC 8824-3:1995, *Information technology – Abstract Syntax Notation One (ASN.1): Constraint specification.*
- ITU-T Recommendation X.683 (1994) | ISO/IEC 8824-4:1995, *Information technology – Abstract Syntax Notation One (ASN.1): Parameterization of ASN.1 specifications.*

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 ITU-T Rec. X.680 | ISO/IEC 8824-1.

3.2 Constraint specification

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

- table constraint

3.3 Parameterization of ASN.1 specification

This Recommendation | International Standard uses the following terms defined in ITU-T Rec. X.683 | ISO/IEC 8824-4:

a) parameterized type;

b) parameterized value.

3.4 Additional definitions

3.4.1 associated table: (For some information object or information object set) an abstract table, derivable from the object or object set by flattening the hierarchical structure resulting from the presence of link fields (see 3.4.13).

NOTE – An associated table can be used to determine the precise nature of some constraint (see ITU-T Rec. X.682 | ISO/IEC 8824-3) which has been applied using an object set.

3.4.2 default syntax: The notation which shall be used for defining information objects of classes whose definers have not provided a defined syntax (see example 11.9).

3.4.3 defined syntax: A notation, provided by the definer of a class, which allows information objects of that class to be defined in a user-friendly manner.

NOTE – For example, the defined syntax for the class OPERATION might allow instances of the class to be defined by the word ARGUMENT followed by &ArgumentType, then the word RESULT followed by the &ResultType, then the word CODE followed by &operationCode (see example 11.10).

3.4.4 field: A component of an information object class. Each field is a type field, a fixed-type value field, a variable-type value field, a fixed-type value set field, a variable-type value set field, an information object field or an information object set field.

3.4.5 field name: A name which identifies a field of some class; either the class which specifies the field directly, in which case the name is a primitive field name, or a class which has a chain of link fields to that in which the field is actually specified (see 9.13 and 9.14).

3.4.6 identifier field: A fixed-type value field of a class, selected to provide unique identification of information objects in that class. Values of the identifier field, if supplied, are required to be unambiguous within any information object set that is defined for that class. They may, but need not, serve to unambiguously identify information objects of that class within some broader scope.

NOTES

1 The identifier field has a fixed ASN.1 type, and values of that type can be carried in protocol to identify information objects within the class.

2 The scope within which the identifier is unambiguous is that of an information object set. It could, however, also be made unambiguous within any given abstract syntax, or within an entire application context, or could even be global across all classes and all application contexts by use of the object identifier type for the identifier field.

3.4.7 information object: An instance of some information object class, being composed of a set of fields which conform to the field specifications of the class.

NOTE – For example, one specific instance of the information object class OPERATION (mentioned in the example in 3.4.8) might be invertMatrix, which has an &ArgumentType field containing the type Matrix, a &ResultType field also containing the type Matrix, and an &operationCode field containing the value 7 (see example in 10.13).

3.4.8 information object class (class): A set of fields, forming a template for the definition of a potentially unbounded collection of information objects, the instances of the class.

NOTE – For example, an information object class OPERATION might be defined to correspond to the “operation” concept of Remote Operations (ROS). Each of the various named field specifications would then correspond to some aspect which can vary from one operation instance to another. Thus, there could be &ArgumentType, &ResultType, and &operationCode fields, the first two specifying type fields and the third specifying a value field.

3.4.9 information object field: A field which contains an information object of some specified class.

3.4.10 information object set: A non-empty set of information objects, all of the same information object class.

NOTE – For example, one information object set, MatrixOperations, of the class OPERATION (used in the example in 3.4.8) might contain invertMatrix (mentioned in 3.4.7) together with other related operations, such as addMatrices, multiplyMatrices, etc. Such an object set might be used in defining an abstract syntax that makes provision for the invocation and result reporting of all of these operations (see example in 12.4).

3.4.11 information object set field: A field which contains an information object set of some specified class.

3.4.12 instance-of type: A type, defined by referencing an information object class which associates object identifiers with types.

3.4.13 link field: An object or object set field.

3.4.14 object class field type: A type specified by reference to some field of an information object class. In ITU-T Rec. X.682 | ISO/IEC 8824-3, notation is provided to enable this type to be restricted by reference to an information object set of the class.

3.4.15 primitive field name: The name specified directly in an information object class definition without use of a link field.

3.4.16 type field: A field which contains an arbitrary type.

3.4.17 value field: A field which contains a value. Such a field is either of fixed-type or of variable-type. In the former case the type of the value is fixed by the field specification. In the latter case the type of the value is contained in some (specific) type field of the same information object.

3.4.18 value set field: A field which contains a non-empty set of values of some type. Such a field is either of fixed-type or of variable-type. In the former case the type of the values is fixed by the field specification. In the latter case the type of the values is contained in some (specific) type field of the same information object.

NOTE – The set of values in a value set field for an information object constitutes a subtype of the specified type.

4 Abbreviations

ASN.1 Abstract Syntax Notation One

5 Convention

This Recommendation | International Standard employs the notational convention defined in ITU-T Rec. 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 notations which can be used as alternatives for "Assignment" (see ITU-T Rec. X.680 | ISO/IEC 8824-1, clause 10) are defined in this Recommendation | International Standard:

- ObjectClassAssignment (see 9.1);
- ObjectAssignment (see 11.1);
- ObjectSetAssignment (see 12.1).

6.2 Types

6.2.1 The following notations which can be used as alternatives for "BuiltinType" (see ITU-T Rec. X.680 | ISO/IEC 8824-1, subclause 14.2) are defined in this Recommendation | International Standard:

- ObjectClassFieldType (see 14.1);
- InstanceOfType (see Annex C).

6.2.2 The following notations which can be used as alternatives for "ReferencedType" (see ITU-T Rec. X.680 | ISO/IEC 8824-1, subclause 14.3) are defined in this Recommendation | International Standard:

- TypeFromObject (see clause 15);
- ValueSetFromObjects (see clause 15).

6.3 Values

6.3.1 The following notations which can be used as alternatives for "BuiltinValue" (see ITU-T Rec. X.680 | ISO/IEC 8824-1, subclause 14.8) are defined in this Recommendation | International Standard:

- ObjectClassFieldValue (see 14.6);
- InstanceOfValue (see Annex C).

6.3.2 The following notation which can be used as an alternative for "ReferencedValue" (see ITU-T Rec. X.680 | ISO/IEC 8824-1, subclause 14.9) is defined in this Recommendation | International Standard:

- ValueFromObject (see clause 15).

6.4 Elements

6.4.1 The following notation which can be used as an alternative for "Elements" (see ITU-T Rec. X.680 | ISO/IEC 8824-1, subclause 44.3) is defined in this Recommendation | International Standard:

– ObjectSetElements (see 12.3).

7 ASN.1 items

In addition to the ASN.1 items specified in ITU-T Rec. X.680 | ISO/IEC 8824-1, clause 9, this Recommendation | International Standard makes use of the ASN.1 items specified in the following subclauses. The general rules applicable to these items are as defined in ITU-T Rec. X.680 | ISO/IEC 8824-1, subclause 9.1. These new ASN.1 items make use of the ASN.1 character set, as specified in ITU-T Rec. X.680 | ISO/IEC 8824-1, clause 8, and in addition the character ampersand ("&").

NOTE – The note in ITU-T Rec. X.680 | ISO/IEC 8824-1, subclause 8.1, also applies to the items specified in the subclauses 7.1-7.9 below.

7.1 Information object class references

Name of item – objectclassreference

An "objectclassreference" shall consist of a sequence of characters as specified for a "typereference" in ITU-T Rec. X.680 | ISO/IEC 8824-1, subclause 9.2, except that no lower-case letters shall be included.

7.2 Information object references

Name of item – objectreference

An "objectreference" shall consist of a sequence of characters as specified for a "valuereference" in ITU-T Rec. X.680 | ISO/IEC 8824-1, subclause 9.4.

7.3 Information object set references

Name of item – objectsetreference

An "objectsetreference" shall consist of a sequence of characters as specified for a "typereference" in ITU-T Rec. X.680 | ISO/IEC 8824-1, subclause 9.2.

7.4 Type field references

Name of item – typefieldreference

A "typefieldreference" shall consist of an ampersand ("&") immediately followed by a sequence of characters as specified for a "typereference" in ITU-T Rec. X.680 | ISO/IEC 8824-1, subclause 9.2.

7.5 Value field references

Name of item – valuefieldreference

A "valuefieldreference" shall consist of an ampersand ("&") immediately followed by a sequence of characters as specified for a "valuereference" in ITU-T Rec. X.680 | ISO/IEC 8824-1, subclause 9.4.

7.6 Value set field references

Name of item – valuesetfieldreference

A "valuesetfieldreference" shall consist of an ampersand ("&") immediately followed by a sequence of characters as specified for a "typereference" in ITU-T Rec. X.680 | ISO/IEC 8824-1, subclause 9.2.

7.7 Object field references

Name of item – objectfieldreference

An "objectfieldreference" shall consist of an ampersand ("&") immediately followed by a sequence of characters as specified for an "objectreference" in 7.2.

7.8 Object set field references

Name of item – objectsetfieldreference

An "objectsetfieldreference" shall consist of an ampersand ("&") immediately followed by a sequence of characters as specified for an "objectsetreference" in 7.3.

7.9 Word

Name of item – word

A "word" shall consist of a sequence of characters as specified for a "typereference" in ITU-T Rec. X.680 | ISO/IEC 8824-1, subclause 9.2, except that no lower-case letters or digits shall be included.

7.10 Additional keyword items

The names CLASS, INSTANCE, SYNTAX and UNIQUE are listed in ITU-T Rec. X.680 | ISO/IEC 8824-1, subclause 9.16 as reserved words.

8 Referencing definitions

8.1 The constructs

```
DefinedObjectClass ::=
        ExternalObjectClassReference |
        objectclassreference |
        UsefulObjectClassReference

DefinedObject ::=
        ExternalObjectReference |
        objectreference

DefinedObjectSet ::=
        ExternalObjectSetReference |
        objectsetreference
```

are used to reference class, information object, and information object set definitions, respectively.

8.2 Except as specified in ITU-T Rec. X.680 | ISO/IEC 8824-1, subclause 10.14, the "objectclassreference", "objectreference", and "objectsetreference" alternatives shall only be used within the module in which a class or information object or information object set is assigned (see 9.1, 11.1 and 12.1) to that reference.

The "ExternalObjectClassReference", "ExternalObjectReference", and "ExternalObjectSetReference" alternatives are defined as follows:

```
ExternalObjectClassReference ::=
        modulereference
        "."
        objectclassreference

ExternalObjectReference ::=
        modulereference
        "."
        objectreference

ExternalObjectSetReference ::=
        modulereference
        "."
        objectsetreference
```

These alternatives shall not be used unless the corresponding "objectclassreference", "objectreference", or "objectsetreference" has been assigned a class or information object or information object set respectively (see 9.1, 11.1 and 12.1) within the module (different from the referencing module) identified by the corresponding "modulereference". It is that class or information object or information object set respectively which is referenced.

8.3 The "Usefulobjectclassreference" alternative of "DefinedObjectClass" is defined as follows:

Usefulobjectclassreference ::= TYPE-IDENTIFIER | ABSTRACT-SYNTAX

of which the first alternative is specified in Annex A, and the second in Annex B.

NOTE – The names TYPE-IDENTIFIER and ABSTRACT-SYNTAX are listed in ITU-T Rec. X.680 | ISO/IEC 8824-1, subclause 9.16 as reserved words.

9 Information object class definition and assignment

9.1 The construct "ObjectClassAssignment" is used to assign an information object class to a reference name ("objectclassreference"). This construct is one of the alternatives for "Assignment" in ITU-T Rec. X.680 | ISO/IEC 8824-1, clause 10, and is defined as follows:

ObjectClassAssignment ::=
objectclassreference
"::="
ObjectClass

9.2 The information object class is that defined by the construct "ObjectClass".

ObjectClass ::=
DefinedObjectClass |
ObjectClassDefn |
ParameterizedObjectClass

If the "ObjectClass" is a:

a) "DefinedObjectClass" then the class definition is the same as that of the class referred to;

b) "ObjectClassDefn" then the class is defined as described in 9.3;

c) "ParameterizedObjectClass" then the class is defined as described in ITU-T Rec. X.683 | ISO/IEC 8824-4, subclause 9.2.

9.3 Every class is ultimately defined by an "ObjectClassDefn":

ObjectClassDefn ::=
CLASS
"{" FieldSpec "," + "}"
WithSyntaxSpec?

WithSyntaxSpec ::= WITH SYNTAX SyntaxList

This notation allows the definer of a class to provide the named field specifications, each of which is a "FieldSpec", as defined in 9.4. Optionally, the definer can provide an information object definition syntax ("SyntaxList"), as defined in 10.5. The definer of the class may also specify semantics associated with the definition of the class.

9.4 Each "FieldSpec" specifies and names one of the fields which shall or may be associated with instances of the class.

FieldSpec ::=
TypeFieldSpec |
FixedTypeValueFieldSpec |
VariableTypeValueFieldSpec |
FixedTypeValueSetFieldSpec |
VariableTypeValueSetFieldSpec |
ObjectFieldSpec |
ObjectSetFieldSpec

The various alternatives for "FieldSpec" are specified in the following subclauses.

9.5 A "TypeFieldSpec" specifies that the field is a type field (see 3.4.16).

TypeFieldSpec ::=
typefieldreference
TypeOptionalitySpec?

TypeOptionalitySpec ::= OPTIONAL | DEFAULT Type

The name of the field is "typefieldreference". If the "TypeOptionalitySpec" is absent, all information object definitions for that class are required to include a specification of a type for that field. If "OPTIONAL" is present, then the field can be left undefined. If "DEFAULT" is present, then the following "Type" provides the default setting for the field if it is omitted in a definition.

9.6 A "FixedTypeValueFieldSpec" specifies that the field is a fixed-type value field (see 3.4.17).

FixedTypeValueFieldSpec ::=
valuefieldreference
Type
UNIQUE?
ValueOptionalitySpec?

ValueOptionalitySpec ::= OPTIONAL | DEFAULT Value

The name of the field is "valuefieldreference". The "Type" construct specifies the type of the value contained in the field. The "ValueOptionalitySpec", if present, specifies that the value may be omitted in an information object definition, or, in the "DEFAULT" case, that omission produces the following "Value", which shall be of that type. The presence of the keyword "UNIQUE" specifies that this field is an identifier field. If the keyword is present, the "ValueOptionalitySpec" shall not be "DEFAULT Value".

9.7 Where a value is assigned for an identifier field, that value is required to be unambiguous within any defined information object set.

9.8 A "VariableTypeValueFieldSpec" specifies that the field is a variable-type value field (see 3.4.17).

VariableTypeValueFieldSpec ::=
valuefieldreference
FieldName
ValueOptionalitySpec?

The name of the field is "valuefieldreference". The "FieldName" (see 9.14), which is relative to the class being specified, shall be that of a type field; the type field which is either in the same information object as the value field, or is linked by the chain of object fields whose references appear in the "FieldName", will contain the type of the value. (All link fields whose field references appear in the "FieldName" shall be object fields.) The "ValueOptionalitySpec", if present, specifies that the value may be omitted in an information object definition, or, in the "DEFAULT" case, that omission produces the following "Value". The "ValueOptionalitySpec" shall be such that:

a) if the type field denoted by the "FieldName" has a "TypeOptionalitySpec" of "OPTIONAL", then the "ValueOptionalitySpec" shall also be "OPTIONAL"; and

b) if the "ValueOptionalitySpec" is "DEFAULT Value", then the type field denoted by the "FieldName" shall have a "TypeOptionalitySpec" of "DEFAULT Type", and "Value" shall be a value of that type.

9.9 A "FixedTypeValueSetFieldSpec" specifies that the field is a fixed-type value set field (see 3.4.18):

FixedTypeValueSetFieldSpec ::=
valuesetfieldreference
Type
ValueSetOptionalitySpec?

ValueSetOptionalitySpec ::= OPTIONAL | DEFAULT ValueSet

NOTE – "ValueSet" is defined in ITU-T Rec. X.680 | ISO/IEC 8824-1, subclauses 13.4 and 13.5, and allows the explicit listing (in curly braces) of the set of values, or the use of a "typereference" for a subtype of the "Type".

The name of the field is "valuesetfieldreference". The "Type" construct specifies the type of the values contained in the field. The "ValueSetOptionalitySpec", if present, specifies that the field may be unspecified in information object definition, or, in the "DEFAULT" case, that omission produces the following "ValueSet", which shall be subtype of that type.

9.10 A "VariableTypeValueSetFieldSpec" specifies that the field is a variable-type value set field (see 3.4.18).

VariableTypeValueSetFieldSpec ::=
valuesetfieldreference
FieldName
ValueSetOptionalitySpec?

The name of the field is "valuesetfieldreference". The "FieldName" (see 9.14), which is relative to the class being specified, shall be that of a type field; the type field which is either in the same information object as the value set field, or is linked by the chain of object fields whose references appear in the "FieldName", will contain the type of the values. (All link fields whose field references appear in the "FieldName" shall be object fields.) The "ValueSetOptionalitySpec", if present, specifies that the value set may be omitted in an information object definition, or, in the "DEFAULT" case, that omission produces the following "ValueSet". The "ValueSetOptionalitySpec" shall be such that:

a) if the type field denoted by the "FieldName" has a "TypeOptionalitySpec" of "OPTIONAL", then the "ValueSetOptionalitySpec" shall also be "OPTIONAL"; and

b) if the "ValueSetOptionalitySpec" is "DEFAULT ValueSet", then the type field denoted by the "FieldName" shall have a "TypeOptionalitySpec" of "DEFAULT Type", and "ValueSet" shall be a subtype of that type.

9.11 An "ObjectFieldSpec" specifies that the field is an information object field (see 3.4.9).

ObjectFieldSpec ::=
objectfieldreference
DefinedObjectClass
ObjectOptionalitySpec?

ObjectOptionalitySpec ::= OPTIONAL | DEFAULT Object

The name of the field is "objectfieldreference". The "DefinedObjectClass" references the class of the object contained in the field (which may be the "ObjectClass" currently being defined). The "ObjectOptionalitySpec", if present, specifies that the field may be unspecified in an information object definition, or, in the "DEFAULT" case, that omission produces the following "Object" (see 11.2) which shall be of the "DefinedObjectClass".

9.12 An "ObjectSetFieldSpec" specifies that the field is an information object set field (see 3.4.11).

ObjectSetFieldSpec ::=
objectsetfieldreference
DefinedObjectClass
ObjectSetOptionalitySpec?

ObjectSetOptionalitySpec ::= OPTIONAL | DEFAULT ObjectSet

The name of the field is "objectsetfieldreference". The "DefinedObjectClass" references the class of the objects contained in the field. The "ObjectSetOptionalitySpec", if present, specifies that the field may be unspecified in an information object definition, or, in the "DEFAULT" case, that omission produces the following "ObjectSet" (see 12.2), all of whose objects shall be of "DefinedObjectClass".

9.13 The construct "PrimitiveFieldName" is used to identify a field relative to the class containing its specification:

PrimitiveFieldName ::=
typefieldreference |
valuefieldreference |
valuesetfieldreference |
objectfieldreference |
objectsetfieldreference

The names of all of the fields specified in the class definition shall be distinct.

9.14 The construct "FieldName" is used to identify a field relative to some class which either contains the field specification directly or which has a chain of link fields to the containing class. The chain is indicated by a list of "PrimitiveFieldName"s separated by periods.

FieldName ::= PrimitiveFieldName "." +

9.15 If there is any chain (of length one or more) of specifications of link fields (see 3.4.13) such that:

a) the first is in the class which is being defined; and

b) each subsequent one is a field of the class used in defining the previous; and

c) the last is defined using the class which is being defined,

then at least one of the field specifications shall have an "ObjectOptionalitySpec" or "ObjectSetOptionalitySpec" (as appropriate).

NOTE – This is to prevent recursive information object class definitions with no finite representation for an information object of that recursive class.

9.16 Examples

An expanded version of the information object class described informally as an example in 3.4.8 could be defined as follows:

```
OPERATION ::= CLASS
{
    &ArgumentType       OPTIONAL,
    &ResultType         OPTIONAL,
    &Errors             ERROR OPTIONAL,
    &Linked             OPERATION OPTIONAL,
    &resultReturned     BOOLEAN DEFAULT TRUE,
    &code               INTEGER UNIQUE

}

ERROR ::= CLASS
{
    &ParameterType      OPTIONAL,
    &code               INTEGER UNIQUE

}
```

NOTES

1 This example is based upon the operation and error concepts of the Remote Operations standard, but simplified for the present purposes.

2 The fields specified for this class include two type fields (&ArgumentType and &ResultType) two object set fields (&Errors and &Linked) and two value fields (&resultReturned and &code) the latter being an identifier field.

3 Any information object set made up of OPERATIONs must be such that no two objects in the set have the same value for the &code field. (The same applies to object sets of ERRORs.)

4 The OPERATION information object class includes a chain of link fields as described in 9.15 above. The chain is of length one and is formed by the &Linked field, which is specified (recursively) by means of OPERATION. However, this is quite valid, because the field is designated OPTIONAL.

5 Neither of these examples includes a "WithSyntaxSpec". However, corresponding examples which do are provided in 10.13.

10 Syntax List

10.1 It is frequently the case that a single specification defines an information object class, for which many other independent specifications separately define information objects. It can be appropriate for the definer of the class to provide a user-friendly notation for the definition of information objects in that class.

NOTE – This is what the (historical) ASN.1 "macro notation" was mainly used for, prior to its replacement by the text in this Recommendation | International Standard.

10.2 This clause specifies a notation by which the specifier of an information object class defines the class-specific defined syntax for the specification of information objects of that class.

10.3 The notation is the syntactic construct "SyntaxList", which occurs in the syntactic construct "ObjectClassDefn".

10.4 A "SyntaxList" specifies the syntax for the definition of a single information object of the class being defined. The syntax appears as the "DefinedSyntax" in the following subclause.

NOTE – It is a property of this specification that the end of any syntactic construct defined by a "SyntaxList" (an instance of "DefinedSyntax") can be determined by:

a) ignoring ASN.1 comments;

b) treating character string values as lexical tokens;

c) expecting an initial "{", matching nested "{" and "}", and terminating on an unmatched "}".

10.5 The "SyntaxList" specifies the sequence of "DefinedSyntaxToken" that is to appear in the "DefinedSyntax" (see 11.5).

SyntaxList ::= "{" TokenOrGroupSpec empty + "}"

TokenOrGroupSpec ::= RequiredToken | OptionalGroup

OptionalGroup ::= "[" TokenOrGroupSpec empty + "]"

RequiredToken ::=
Literal |
PrimitiveFieldName

NOTE – The writer of "SyntaxList" is **not** given the full power of BNF. Roughly, the notational power is equivalent to that commonly used in specifying command line syntaxes for command interpreters. The list of possible "RequiredToken"s are given in the order they are permitted; one or more consecutive tokens can be made optional by enclosing them in square brackets.

10.6 A "word" token used as a "Literal" shall not be one of the following:

BIT
BOOLEAN
CHARACTER
CHOICE
EMBEDDED
END
ENUMERATED
EXTERNAL
FALSE
INSTANCE
INTEGER
INTERSECTION
MINUS-INFINITY
NULL
OBJECT
OCTET
PLUS-INFINITY
REAL
SEQUENCE
SET
TRUE
UNION

NOTE – This list comprises only and all those ASN.1 reserved words which can appear as the first item of a "Type", "Value", "ValueSet", "Object" or "ObjectSet", and also the reserved word "END". Use of other ASN.1 reserved words does not cause ambiguity and is permitted. Where the defined syntax is used in an environment in which a "word" is also a "typereference" or "objectsetreference", the use as a "word" takes precedence.

10.7 A "Literal" specifies the actual inclusion of that "Literal", which is either a "word" or a comma (","), at that position in the defined syntax.

Literal ::=
word |
","

10.8 Each "PrimitiveFieldName" specifies the inclusion (at that position in the new syntax) of a "Setting" (see 11.6) for the corresponding field.

10.9 Each "PrimitiveFieldName" of the information object class shall appear precisely once.

10.10 When, in the parse process, an "OptionalGroup" is encountered, and the following ASN.1 item is syntactically acceptable as the first ASN.1 item in the optional group, then that group is assumed to be present. If it is not syntactically acceptable as the first ASN.1 item in the optional group, then that group is assumed to be absent.

NOTE – In order to avoid unexpected effects, designers should normally make the first ASN.1 item in an optional group a "Literal".

10.11 An instance of use of the "DefinedSyntax" is invalid unless it specifies all mandatory fields for the information object class.

10.12 In order to ensure easy parsing of the new syntax and to prevent abuses, the following additional restrictions are placed on the definer of new syntax:

a) every "OptionalGroup" is required to have at least one "PrimitiveFieldName" or "OptionalGroup" within it;

NOTE 1 – This is to help prevent the apparent collection of information which is not reflected in any field of the information object.

b) the use of "OptionalGroup"s shall be such that at no time in the parsing process can a "Setting" appear that could potentially be a setting for more than one "FieldName";

c) if an "OptionalGroup" starts with a "Literal", then the first token following the "OptionalGroup" shall also be a "Literal" and shall be different from the first "Literal" of all immediately preceding "OptionalGroup"s;

while the following restriction is placed upon the user of the "DefinedSyntax":

d) whenever a "Literal" is present in a "DefinedSyntax" that occurs in an "OptionalGroup" a "Setting" for a "PrimitiveFieldName" in that "OptionalGroup" shall also be present;

NOTE 2 – This is to help prevent the apparent collection of information which is not reflected in any field of the information object;

NOTE 3 – The following example is a legal syntax but restriction d) prevents the user from writing "LITERAL" without following it by one or both of the optional groups.

[LITERAL [A &field] [B &field2]]

10.13 Examples

The examples of class definitions from 9.16 above can be equipped with defined syntax to provide a "user-friendly" way of defining instances of the classes. (This defined syntax is used in the example in 11.10.)

```
OPERATION ::= CLASS
{
    &ArgumentType       OPTIONAL,
    &ResultType         OPTIONAL,
    &Errors             ERROR OPTIONAL,
    &Linked             OPERATION OPTIONAL,
    &resultReturned     BOOLEAN DEFAULT TRUE,
    &operationCode      INTEGER UNIQUE

}
WITH SYNTAX
{
    [ARGUMENT           &ArgumentType]
    [RESULT             &ResultType]
    [RETURN RESULT      &resultReturned]
    [ERRORS             &Errors]
    [LINKED             &Linked]
    CODE                &operationCode
}

ERROR ::= CLASS
{
    &ParameterType      OPTIONAL,
    &errorCode          INTEGER UNIQUE
}

WITH SYNTAX
{
    [PARAMETER          &ParameterType]
    CODE                &errorCode
}
```

11 Information object definition and assignment

11.1 The syntactic construct "ObjectAssignment" is used to assign an information object of a specified class to a reference name ("objectreference"). This construct is one of the alternatives for "Assignment" in ITU-T Rec. X.680 | ISO/IEC 8824-1, clause 10, and is defined as follows:

ObjectAssignment ::=
objectreference
DefinedObjectClass
"::="
Object

11.2 The information object, which shall be of the class referenced by "DefinedObjectClass", is that defined by the construct "Object".

Object ::=
DefinedObject |
ObjectDefn |
ObjectFromObject |
ParameterizedObject

If the "Object" is a:

a) "DefinedObject", then the object is the same as that referred to;

b) "ObjectDefn", then the object is as specified in 11.3;

c) "ObjectFromObject", then the object is as specified in clause 15;

d) "ParameterizedObject", then the object is defined as specified in ITU-T Rec. X.683 | ISO/IEC 8824-4, subclause 9.2.

11.3 Every information object is ultimately defined by an "ObjectDefn":

ObjectDefn ::=
DefaultSyntax |
DefinedSyntax

The "ObjectDefn" shall be "DefaultSyntax" (see 11.4) if the class definition does not include a "WithSyntaxSpec" and shall be "DefinedSyntax" (see 11.5) if it does include one.

11.4 The "DefaultSyntax" construct is defined as follows:

DefaultSyntax ::= "{" FieldSetting "," * "}"

FieldSetting ::= PrimitiveFieldName Setting

There shall be precisely one "FieldSetting" for each "FieldSpec" in the class definition which is not "OPTIONAL" and does not have a "DEFAULT", and at most one "FieldSetting" for each other "FieldSpec". The "FieldSetting"s can appear in any order. The "PrimitiveFieldName" in each "FieldSetting" shall be the name of the corresponding "FieldSpec". The construct "Setting" is specified in 11.6.

11.5 The "DefinedSyntax" construct is defined as follows:

DefinedSyntax ::= "{" DefinedSyntaxToken empty * "}"

DefinedSyntaxToken ::=
Literal |
Setting

The "SyntaxList" in the "WithSyntaxSpec" (see clause 10) determines the sequence of "DefinedSyntaxToken"s that are to appear in the "DefinedSyntax". The construct "Setting" is specified in 11.6; each occurrence specifies the setting for some field of the information object. The construct "Literal" is defined in 10.7; "Literal"s are present for human readability.

11.6 A "Setting" specifies the setting of some field within an information object being defined.

Setting ::=
Type |
Value |
ValueSet |
Object |
ObjectSet

If the field is:

a) a type field, the "Type" alternative;

b) a value field, the "Value" alternative;

c) a value set field, the "ValueSet" alternative;

d) an information object field, the "Object" alternative;

e) an information object set field, the "ObjectSet" alternative,

shall be selected.

NOTE – The setting is further restricted as described in the appropriate subclause of 9.5-9.12 above, and 11.7-11.8.

11.7 A setting of a variable-type value field shall be a value of the type specified by the appropriate type field of the same or linked object (that is, the value notation for an open type is not employed).

11.8 A setting of a variable-type value set field shall be a value set of the type specified by the appropriate type field of the same or linked object (that is, the value notation for an open type is not employed).

11.9 Examples (Default Syntax)

Given the information object class definitions of 9.16 above (which do not include a "WithSyntaxSpec") instances of the classes are defined using the "DefaultSyntax". For example (an expanded version of the example given in 3.4.7):

```
invertMatrix OPERATION ::=
{
    &ArgumentType     Matrix,
    &ResultType       Matrix,
    &Errors           {determinantIsZero},
    &operationCode    7
}

determinantIsZero ERROR ::=
{
    &errorCode        1
}
```

11.10 Examples (Defined Syntax)

In 10.13, the example classes are provided "WithSyntaxSpec" and thus, instances of the classes are defined using the "DefinedSyntax". The examples of 11.9 would be written thus:

```
invertMatrix OPERATION ::=
{
    ARGUMENT          Matrix
    RESULT            Matrix
    ERRORS            {determinantIsZero}
    CODE              7
}

determinantIsZero ERROR ::=
{
    CODE              1
}
```

12 Information object set definition and assignment

12.1 The syntactic construct "ObjectSetAssignment" is used to assign a set of information objects of a specified class to a reference name ("objectsetreference"). This construct is one of the alternatives for "Assignment" in ITU-T Rec. X.680 | ISO/IEC 8824-1, clause 10, and is defined as follows:

```
ObjectSetAssignment ::=
    objectsetreference
    DefinedObjectClass
    "::="
    ObjectSet
```

12.2 The information object set, which shall be of the class referenced by "DefinedObjectClass", is that defined by the construct "ObjectSet"

ObjectSet ::= "{" ElementSetSpec "}"

"ElementSetSpec" is specified in ITU-T Rec. X.680 | ISO/IEC 8824-1 and enables an information object set to be specified in terms of information objects or sets thereof of the governing class. There shall be at least one information object in the set. "ElementSetSpec" in turn employs the construct "ObjectSetElements" defined in 12.3.

12.3 The notation for "ObjectSetElements" is as follows:

ObjectSetElements ::=
Object |
DefinedObjectSet |
ObjectSetFromObjects |
ParameterizedObjectSet

The elements specified by this notation are determined by which alternative is employed, as follows:

a) If the "Object" alternative is used, then only the object so designated is specified. That object shall be of the governing class.

b) If any of the remaining alternatives is used, then all of the objects of the set so designated are specified. The objects shall be of the governing class. If the "DefinedObjectSet" alternative is used, the object set is that referred to. If the "ObjectSetFromObjects" alternative is used then the object set is as specified in clause 15. If the "ParameterizedObjectSet" alternative is used then the object set is as specified in ITU-T Rec. X.683 | ISO/IEC 8824-4, subclause 9.2.

12.4 Example

The information object set described informally in the Note in 3.4.10 can be specified as follows:

MatrixOperations OPERATION ::=
{
invertMatrix |
addMatrices |
subtractMatrices |
multiplyMatrices
}

13 Associated tables

13.1 Every information object or information object set can be viewed as a table: its associated table. Each cell of the associated table corresponds to the setting of some field of an information object, or is empty. The set of columns of the associated table is determined by the class to which the object or objects belong; the set of rows, however, is determined by the object or objects involved.

13.2 Given the definition of a class, the set of columns is determined as follows:

a) There is one column for each field specification in the class definition. Each such column is named by the corresponding "PrimitiveFieldName".

b) There is an additional set of columns corresponding to each link field specification. This set of columns is that determined by the application of these rules for the governing class of the link field, except that their names are prefixed by the "PrimitiveFieldName" of the link field, and a period (".").

NOTE – These rules are recursive, and are such that if a class is directly or indirectly self-referential the set of columns is not finite. This is not prohibited.

13.3 Given an information object of some class, the associated table is that which would result from applying 13.4 to the object set containing just that object.

13.4 Given an information object set of some class, the set of rows in the associated table are those which would result from performing the following recursive procedure:

a) Start with one row for each object in the object set. In each such row, the cells in the columns named by "PrimitiveFieldName"s will correspond to the setting of the appropriate field in the object, while all other cells will be empty.

b) For each link field appearing in some row in the set:

1) Generate the (subordinate) associated table of the contents of the link field.

2) Next, replace the row in which the link field appears by a collection of rows, one for each row of the subordinate associated table. Each of the rows in this collection is the same as that being replaced, except that the cells from the selected row of the subordinate associated table are used to fill the corresponding cells, hitherto empty, whose "FieldName"s are prefixed by the link field's "PrimitiveFieldName".

NOTE – These rules are recursive, and are such that if an information object is directly or indirectly self-referential, the procedure will not terminate. This is not prohibited. In practice it is only necessary to know the contents of cells with names of a finite length, and a bounded procedure can be devised for this.

13.5 Examples of valid "FieldName"s

The following "FieldName"s are among those which are valid for the associated table for objects or object sets of class OPERATION (as defined in 10.3):

&ArgumentType
&Errors.&Parameter
&Errors.&errorCode
&Linked.&ArgumentType
&Linked.&Linked.&operationCode
&Linked.&Linked.&Linked.&Linked.&Linked.&Errors.&errorCode

Because the class OPERATION is self-referential (through the &Linked field) the number of columns is not finite.

14 Notation for the object class field type

The type that is referenced by this notation depends on the category of the field name. For the different categories of field names, clauses 14.2-14.5 specify the type that is referenced.

14.1 The notation for an object class field type (see 3.4.14) shall be "ObjectClassFieldType":

ObjectClassFieldType ::=
DefinedObjectClass
"."
FieldName

where the "FieldName" is as specified in 9.14 relative to the class identified by the "DefinedObjectClass".

14.2 For a type field, the notation defines an open type, that is, one whose set of values is the complete set of all possible values that can be specified using ASN.1. The specification of constraints using a corresponding information object set (see ITU-T Rec. X.682 | ISO/IEC 8824-3) may restrict this type to a specific type. The following constraints on the use of this notation apply when the "FieldName" references a type field:

a) This notation shall not be used directly or indirectly in the definition of the type of a value or value set field of an information object class.

b) This notation has an indeterminate tag and thus cannot be used where a tag distinct from that of some other type is required.

NOTE 1 – This restriction can normally be avoided by (explicitly) tagging the type.

c) This notation shall not be implicitly tagged.

NOTE 2 – The reason for this is that when this open type is restricted to a particular type that type may be a choice type.

d) Encoding rules are required to encode the value assigned to a component defined in this way in such a way that a receiver can successfully determine the abstract values corresponding to all other parts of the construction in which the component is embedded without any knowledge of the actual type of this component.

NOTE 3 – This "Type" construct will commonly be constrained by use of an information object set and the "AtNotation", as specified in ITU-T Rec. X.682 | ISO/IEC 8824-3, clause 10. Users of ASN.1 are, however, cautioned that use of this notation without the application of a constraint can lead to ambiguity in implementation requirements, and should normally be avoided.

14.3 For a fixed-type value or a fixed type value set field, the notation denotes the "Type" that appears in the specification of that field in the definition of the information object class.

14.4 For a variable-type value or a variable-type value set field, the notation defines an open type. Its use is subject to the same restrictions as specified in 14.2.

14.5 This notation is not permitted if the field is an object field or an object set field.

14.6 The notation for defining a value of this type shall be "ObjectClassFieldValue":

```
ObjectClassFieldValue ::=
    OpenTypeFieldVal |
    FixedTypeFieldVal

OpenTypeFieldVal ::= Type ":" Value

FixedTypeFieldVal ::= BuiltinValue | ReferencedValue
```

14.7 For a fixed-type value or value set field used in "ObjectClassFieldType", the "FixedTypeFieldVal" shall be used, and shall be a value of the "Type" specified in the definition of the information object class.

14.8 For a type field or a variable-type value or value set field used in "ObjectClassFieldType", the "OpenTypeFieldVal" shall be used. The "Type" in the "OpenTypeFieldVal" shall be any ASN.1 type, and the "Value" shall be any value of that type.

14.9 Example usage of "ObjectClassFieldType"

Each of the following examples is based on the example in 10.3 and shows (a) a possible "ObjectClassFieldType", (b) the type to which the example type (a) is equivalent (when used unconstrained), and (c) the notation for an example value of that type.

1 (a) OPERATION.&operationCode
(b) INTEGER
(c) 7

2 (a) OPERATION.&ArgumentType
(b) *open type*
(c) Matrix:
{{1, 0, 0, 0},
{0, 1, 0, 0},
{0, 0, 1, 0},
{0, 0, 0, 1}}

3 (a) OPERATION.&Linked.&Linked.&Errors.&errorCode
(b) INTEGER
(c) 1

4 (a) OPERATION.&Linked.&ArgumentType
(b) *open type*
(c) UniversalString:{planckConstant, " and ", hamiltonOperator}

15 Information from objects

15.1 Information from the column of the associated table for an object or an object set can be referenced by the various cases of the "InformationFromObjects" notation.

```
InformationFromObjects ::=
    ValueFromObject          |
    ValueSetFromObjects      |
    TypeFromObject           |
    ObjectFromObject         |
    ObjectSetFromObjects

ValueFromObject ::=
    ReferencedObjects
    "."
    FieldName

ValueSetFromObjects ::=
    ReferencedObjects
    "."
    FieldName
```

TypeFromObject ::=
ReferencedObjects
"."
FieldName

ObjectFromObject ::=
ReferencedObjects
"."
FieldName

ObjectSetFromObjects ::=
ReferencedObjects
"."
FieldName

ReferencedObjects ::=
DefinedObject | ParameterizedObject |
DefinedObjectSet | ParameterizedObjectSet

15.2 This notation references the total contents of the referenced column of the associated table for the "ReferencedObjects".

15.3 Depending on the form of the "ReferencedObjects" and the "FieldName" this notation can denote a value, a value set, a type, an object, or an object set. These five cases are denoted by the constructs "ValueFromObject", "ValueSetFromObjects", "TypeFromObject", "ObjectFromObject", and "ObjectSetFromObjects" respectively. Each of these constructs is a special case of "InformationFromObjects".

15.4 The "InformationFromObjects" production can be divided into two parts. The first part is formed by deleting the final (or only) "PrimitiveFieldName" and its preceding period. If the first part denotes an object or an object set, then subclauses 15.5-15.9 apply. Otherwise the notation is illegal. The second part is the final (or only) "PrimitiveFieldName".

NOTE – (Tutorial) Given the following definition:

obj.&a.&b.&c.&d

The first part in the definition is obj.&a.&b.&c and the second part is &d.

15.5 The first column of Table 1 indicates the first part defined in 15.4. The second column indicates the second part defined in 15.4. The third column indicates which (if any) of the five cases of "InformationFromObjects" (listed in 15.3) applies.

Table 1 – Permissable cases of "InformationFromObjects"

The first part of InformationFromObjects	The second part of InformationFromObjects	Construct
object	fixed-type value field	"ValueFromObject"
	variable-type value field	"ValueFromObject"
	fixed-type value set field	"ValueSetFromObjects"
	variable-type value set field	"ValueSetFromObjects"
	type field	"TypeFromObject"
	object field	"ObjectFromObject"
	object set field	"ObjectSetFromObject"
object set	fixed-type value field	"ValueSetFromObjects"
	variable-type value field	not permitted
	fixed-type value set field	"ValueSetFromObjects"
	variable-type value set field	not permitted
	type field	not permitted
	object field	"ObjectSetFromObjects"
	object set field	"ObjectSetFromObjects"

15.6 If object sets are involved and the final "PrimitiveFieldName" identifies a fixed-type value set field, then "ValueSetFromObjects" is the union of the selected value sets.

15.7 If object sets are involved and the final "PrimitiveFieldName" identifies an object set field, then "ObjectSetFromObjects" is the union of the selected object sets.

15.8 As shown in Table 1, the notation is not permitted if an object set is involved and the final "PrimitiveFieldName" identifies a variable-type value or value set field or a type field.

15.9 Use of this notation is not permitted if all cells in the column being referenced are empty, except where it is used to directly define a field of an information object which is "OPTIONAL" (or "DEFAULT"), which results in the field becoming empty (or defaults).

15.10 Example information from objects

Given the definitions in the examples of 11.9, 11.10 and 12.4, the following constructs (in the left column) are valid, and can be used as equivalent to the expression in the right column.

"ValueFromObject"

invertMatrix.&operationCode	7
determinantIsZero.&errorCode	1

"TypeFromObject"

invertMatrix.&ArgumentType	Matrix

"ValueSetFromObjects"

invertMatrix.&Errors.&errorCode	{ 1 }
MatrixOperations.&operationCode	{7 | *and others* }

"ObjectSetFromObjects"

invertMatrix.&Errors	{determinantIsZero}
MatrixOperations.&Errors	{determinantIsZero | *and others* }

Annex A

The TYPE-IDENTIFIER information object class

(This annex forms an integral part of this Recommendation | International Standard)

A.1 This annex specifies a useful information object class, with class reference TYPE-IDENTIFIER.

NOTE – This information object class is the simplest useful class, having just two fields, an identifier field of type OBJECT IDENTIFIER, and a type field which defines the ASN.1 type for carrying all information concerning any particular object in the class. It is defined in this Recommendation | International Standard because of the widespread use of information objects of this form.

A.2 The TYPE-IDENTIFIER information object class is defined as:

```
TYPE-IDENTIFIER ::= CLASS
{
    &id OBJECT IDENTIFIER UNIQUE,
    &Type
}
WITH SYNTAX {&Type IDENTIFIED BY &id}
```

A.3 This class is defined as a "useful" information object class, and is available in any module without the necessity for importing it.

A.4 Example

The body of an MHS communication can be defined as:

```
MHS-BODY-CLASS ::= TYPE-IDENTIFIER

g4FaxBody MHS-BODY-CLASS ::=
    {BIT STRING IDENTIFIED BY {mhsbody 3}}
```

A protocol designer would typically define a component to carry an MHS-BODY-CLASS by specifying the type "INSTANCE OF MHS-BODY-CLASS" defined in C.9.

Annex B

Abstract syntax definitions

(This annex forms an integral part of this Recommendation | International Standard)

B.1 This annex specifies a useful information object class, ABSTRACT-SYNTAX, for defining abstract syntaxes.

NOTE – It is recommended that an instance of this information object class be defined whenever an abstract syntax is defined as the values of a single ASN.1 type.

B.2 The ABSTRACT-SYNTAX information object class is defined as:

```
ABSTRACT-SYNTAX ::= CLASS
{
    &id         OBJECT IDENTIFIER UNIQUE,
    &Type
}
WITH SYNTAX { &Type IDENTIFIED BY &id }
```

The &id field of each ABSTRACT-SYNTAX is the abstract syntax name, while the &Type field contains the single ASN.1 type whose values make up the abstract syntax.

B.3 This information object class is defined as being "useful" because it is of general utility, and is available in any module without the necessity for importing it.

B.4 Example

If an ASN.1 type has been defined called XXX-PDU, then an abstract syntax can be specified which contains all the values of XXX-PDU by the notation:

```
xxx-Abstract-Syntax ABSTRACT-SYNTAX ::=
    { XXX-PDU IDENTIFIED BY {xxx 5} }
```

See ITU-T Rec. X.680 | ISO/IEC 8824-1, subclause F.3, for a detailed example of use of the ABSTRACT-SYNTAX information object class.

B.5 It will frequently be the case that an abstract syntax will be defined in terms of a parameterized type (as defined in ITU-T Rec. X.683 | ISO/IEC 8824-4), for example with parameters providing bounds on some components of the protocol. Such parameters, subject to restrictions specified in ITU-T Rec. X.683 | ISO/IEC 8824-4, clause 10, may be resolved at the time of abstract syntax definition, or may be carried forward as parameters of the abstract syntax.

Annex C

The instance-of type

(This annex forms an integral part of this Recommendation | International Standard)

C.1 This annex specifies type and value notation for the instance-of types (see 3.4.12). Such types are capable of carrying any value from any information object in an information object class defined to be of class TYPE-IDENTIFIER (see Annex A) using an information object class assignment (the information object class reference is specified as part of this notation).

C.2 The "InstanceOfType" notation is referenced in ITU-T Rec. X.680 | ISO/IEC 8824-1, subclause 14.2, as one of the notations that produce a "Type", and is defined as:

InstanceOfType ::= INSTANCE OF DefinedObjectClass

NOTE – ITU-T Rec. X.682 | ISO/IEC 8824-3, clause 10, specifies the way in which this type can be constrained by applying a "table constraint", restricting the values of the type to those representing some specific information object set of the class.

C.3 This notation specifies a type which carries the &id field (an OBJECT IDENTIFIER) and a value of the &Type field from any instance of the "DefinedObjectClass".

NOTE – This construct will normally be constrained by an object set which will usually be (but is not necessarily) a dummy reference name as defined in ITU-T Rec. X.683 | ISO/IEC 8824-4, subclauses 8.3-8.11, with the actual object set defined elsewhere.

C.4 All instance-of types have a tag which is universal class, number 8.

NOTE – This is the same universal tag as for external type, and use of the instance-of type can be bit-compatible with the external type when the basic encoding rules for ASN.1 are in use.

C.5 The instance-of type has an associated sequence type which is used for defining values and subtypes of the instance-of type.

NOTE – Where this type is constrained by the constraint notation of ITU-T Rec. X.682 | ISO/IEC 8824-3, the associated sequence type is also constrained. The constraints on the associated sequence type resulting from a constraint on the instance-of type are specified in ITU-T Rec. X.682 | ISO/IEC 8824-3, Annex A.

C.6 The associated sequence type is assumed to be defined within an environment in which "EXPLICIT TAGS" is in force.

C.7 The associated sequence type shall be:

```
SEQUENCE
{
    type-id     <DefinedObjectClass>.&id,
    value       [0] <DefinedObjectClass>.&Type
}
```

where "<DefinedObjectClass>" is replaced by the particular "DefinedObjectClass" used in the "InstanceOfType" notation.

C.8 The value notation "InstanceOfValue" for an "InstanceOfType" notation shall be the value notation for the associated sequence type.

InstanceOfValue ::= Value

C.9 Example

An example, building on the example given in A.4, is as follows:

The type

INSTANCE OF MHS-BODY-CLASS

has an associated sequence type of

```
SEQUENCE
{
    type-id     MHS-BODY-CLASS.&id,
    value       [0]  MHS-BODY-CLASS.&Type
}
```

An example of the application of a table constraint to this type can be found in ITU-T Rec. X.682 | ISO/IEC 8824-3, Annex A.

Annex D

Examples

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

D.1 Example usage of simplified OPERATION class

Given the following simple definition of the OPERATION and ERROR information object classes:

```
OPERATION ::= CLASS
{
    &ArgumentType       OPTIONAL,
    &ResultType         OPTIONAL,
    &Errors             ERROR OPTIONAL,
    &Linked             OPERATION OPTIONAL,
    &resultReturned     BOOLEAN DEFAULT TRUE,
    &operationCode      INTEGER UNIQUE
}

WITH SYNTAX
{
    [ARGUMENT           &ArgumentType]
    [RESULT             &ResultType]
    [RETURN RESULT      &resultReturned]
    [ERRORS             &Errors]
    [LINKED             &Linked]
    CODE                &operationCode
}

ERROR ::= CLASS
{
    &ParameterType      OPTIONAL,
    &errorCode          INTEGER UNIQUE
}

WITH SYNTAX
{
    [PARAMETER          &ParameterType]
    CODE                &errorCode
}
```

We can define the following object set that contains two OPERATION objects:

```
My-Operations OPERATION ::= { operationA | operationB }

operationA OPERATION ::= {
    ARGUMENT            INTEGER
    ERRORS              { { PARAMETER INTEGER CODE 1000 } | { CODE 1001 } }
    CODE                1
}

operationB OPERATION ::= {
    ARGUMENT            IA5String
    RESULT              BOOLEAN
    ERRORS              { { CODE 1002 } | { PARAMETER IA5String CODE 1003 } }
    CODE                2
}
```

Extraction of the set of the ERROR objects from the object set above is done as follows:

```
My-OperationErrors ERROR ::= { My-Operations.&Errors }
```

The resulting object set is:

```
My-OperationErrors ERROR ::= {
    { PARAMETER INTEGER  CODE 1000 } |
    { CODE 1001 } |
    { CODE 1002 } |
    { PARAMETER IA5String  CODE 1003 }
}
```

Extraction of the set of error codes of the errors of the operations is done as follows:

```
My-OperationErrorCodes INTEGER ::= { My-Operations.&Errors.&errorCode }
```

The resulting value set is:

```
My-OperationErrorCodes INTEGER ::= { 1000 | 1001 | 1002 | 1003 }
```

D.2 Example usage of "ObjectClassFieldType"

The "ObjectClassFieldType" can be used in specification of types, for example:

```
-- "ObjectClassFieldType"s are extracted from this class.
-- Only the first five fields can be used in the extraction.

EXAMPLE-CLASS ::= CLASS {
    &TypeField                          OPTIONAL,
    &fixedTypeValueField                INTEGER   OPTIONAL,
    &variableTypeValueField   &TypeField        OPTIONAL,
    &FixedTypeValueSetField   INTEGER           OPTIONAL,
    &VariableTypeValueSetField &TypeField       OPTIONAL,
    &objectField              SIMPLE-CLASS      OPTIONAL,
    &ObjectSetField           SIMPLE-CLASS      OPTIONAL
}

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]
    [OBJECT-FIELD                               &objectField]
    [OBJECT-SET-FIELD                           &ObjectSetField]
}

SIMPLE-CLASS ::= CLASS {
    &value    INTEGER
}

WITH SYNTAX {
    &value
}
-- This type contains components which are specified using "ObjectClassFieldType" notation.
-- In case of type fields and variable-type value and value set fields the resulting
-- component type is an open type. In case of fixed-type value and value set fields the
-- resulting component type is INTEGER.  NOTE: Constraints are omitted from all the following
-- uses of "ObjectClassFieldType"; you normally will use constraints when referencing an
-- "ObjectClassFieldType".

ExampleType ::= SEQUENCE {
    openTypeComponent1        EXAMPLE-CLASS.&TypeField,
    integerComponent1         EXAMPLE-CLASS.&fixedTypeValueField,
    openTypeComponent2        EXAMPLE-CLASS.&variableTypeValueField,
    integerComponent2         EXAMPLE-CLASS.&FixedTypeValueSetField,
    openTypeComponent3        EXAMPLE-CLASS.&VariableTypeValueSetField
}
```

```
exampleValue ExampleType ::= {
    openTypeComponent1          BOOLEAN : TRUE,
    integerComponent1           123,
    openTypeComponent2          IA5String : "abcdef",
    integerComponent2           456,
    openTypeComponent3          BIT STRING : '0101010101'B
}
```

D.3 Illustrate usage of objects and object sets

The following uses the object class defined in D.2.

```
objectA EXAMPLE-CLASS ::= {
    FIXED-TYPE-VALUE-FIELD          123
    FIXED-TYPE-VALUE-SET-FIELD      { 1 | 2 | 3 }
    OBJECT-FIELD                    { 1 }
    OBJECT-SET-FIELD                { { 2 } | { 3 } }
}

objectB EXAMPLE-CLASS ::= {
    TYPE-FIELD                              IA5String
    FIXED-TYPE-VALUE-FIELD                  456
    VARIABLE-TYPE-VALUE-FIELD               "abc"
    VARIABLE-TYPE-VALUE-SET-FIELD           { "d" | "e" | "f" }
}
-- The following object set contains two defined objects and one builtin object.

ObjectSet EXAMPLE-CLASS ::= {
    objectA |
    objectB |
    {
        TYPE-FIELD                          INTEGER
        FIXED-TYPE-VALUE-FIELD              789
        VARIABLE-TYPE-VALUE-SET-FIELD       { 4 | 5 | 6 }
    }
}
-- The following definitions extract information from the objects and the object set.

integerValue INTEGER ::= objectA.&fixedTypeValueField

stringValue IA5String ::= objectB.&variableTypeValueField

IntegerValueSetFromObjectA INTEGER ::= { objectA.&FixedTypeValueSetField }

StringValueSet IA5String ::= { objectB.&VariableTypeValueSetField }

StringType ::= objectB.&TypeField

objectFromObjectA SIMPLE-CLASS ::= objectA.&objectField

ObjectSetFromObjectA SIMPLE-CLASS ::= { objectA.&ObjectSetField }

SetOfValuesInObjectSet INTEGER ::= { ObjectSet.&fixedTypeValueField }

SetOfValueSetsInObjectSet INTEGER ::= { ObjectSet.&FixedTypeValueSetField }

SetOfObjectsInObjectSet SIMPLE-CLASS ::= { ObjectSet.&objectField }

SetOfObjectSetsInObjectSet SIMPLE-CLASS ::= { ObjectSet.&ObjectSetField }
```

Annex E

Summary of the notation

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

The following items are defined in clause 7:

objectclassreference
objectreference
objectsetreference
typefieldreference
valuefieldreference
valuesetfieldreference
objectsetfieldreference
word
CLASS
INSTANCE
SYNTAX
UNIQUE

The following items are defined in ITU-T Rec. X.680 | ISO/IEC 8824-1 and used in this Recommendation | International Standard:

empty
modulereference
"::="
"{"
"}"
","
"."
"["
"]"
":"
DEFAULT
OF
OPTIONAL
WITH

The following productions are defined in ITU-T Rec. X.680 | ISO/IEC 8824-1 and are used in this Recommendation | International Standard:

ElementSetSpec
Type
Value
ValueSet

The following productions are defined in ITU-T Rec. X.683 | ISO/IEC 8824-4 and are used in this Recommendation | International Standard:

ParameterizedObjectClass
ParameterizedObjectSet
ParameterizedObject

The following productions are defined in this Recommendation | International Standard:

```
DefinedObjectClass ::=
        ExternalObjectClassReference | objectclassreference | UsefulObjectClassReference

ExternalObjectClassReference ::= modulereference "." objectclassreference

UsefulObjectClassReference ::=
    TYPE-IDENTIFIER |
    ABSTRACT-SYNTAX

ObjectClassAssignment ::= objectclassreference "::=" ObjectClass

ObjectClass ::= DefinedObjectClass | ObjectClassDefn | ParameterizedObjectClass

ObjectClassDefn ::= CLASS "{" FieldSpec "," + "}" WithSyntaxSpec?
```

```
FieldSpec ::=
        TypeFieldSpec |
        FixedTypeValueFieldSpec |
        VariableTypeValueFieldSpec |
        FixedTypeValueSetFieldSpec |
        VariableTypeValueSetFieldSpec |
        ObjectFieldSpec |
        ObjectSetFieldSpec

PrimitiveFieldName ::=
        typefieldreference |
        valuefieldreference |
        valuesetfieldreference |
        objectfieldreference |
        objectsetfieldreference

FieldName ::= PrimitiveFieldName "." +

TypeFieldSpec ::= typefieldreference TypeOptionalitySpec?

TypeOptionalitySpec ::= OPTIONAL | DEFAULT Type

FixedTypeValueFieldSpec ::= valuefieldreference Type UNIQUE? ValueOptionalitySpec?

ValueOptionalitySpec ::= OPTIONAL | DEFAULT Value

VariableTypeValueFieldSpec ::= valuefieldreference FieldName ValueOptionalitySpec?

FixedTypeValueSetFieldSpec ::= valuesetfieldreference Type ValueSetOptionalitySpec?

ValueSetOptionalitySpec ::= OPTIONAL | DEFAULT ValueSet

VariableTypeValueSetFieldSpec ::= valuesetfieldreference FieldName ValueSetOptionalitySpec?

ObjectFieldSpec ::= objectfieldreference DefinedObjectClass ObjectOptionalitySpec?

ObjectOptionalitySpec ::= OPTIONAL | DEFAULT Object

ObjectSetFieldSpec ::= objectsetfieldreference DefinedObjectClass ObjectSetOptionalitySpec?

ObjectSetOptionalitySpec ::= OPTIONAL | DEFAULT ObjectSet

WithSyntaxSpec ::= WITH SYNTAX SyntaxList

SyntaxList ::= "{" TokenOrGroupSpec empty * "}"

TokenOrGroupSpec ::= RequiredToken | OptionalGroup

OptionalGroup ::= "[" TokenOrGroupSpec empty + "]"

RequiredToken ::= Literal | PrimitiveFieldName

Literal ::= word | ","

DefinedObject ::= ExternalObjectReference | objectreference

ExternalObjectReference ::= modulereference "." objectreference

ObjectAssignment ::= objectreference DefinedObjectClass "::=" Object

Object ::= DefinedObject | ObjectDefn | ObjectFromObject |  ParameterizedObject

ObjectDefn ::= DefaultSyntax | DefinedSyntax

DefaultSyntax ::= "{" FieldSetting "," * "}"

FieldSetting ::= PrimitiveFieldName Setting

DefinedSyntax ::= "{" DefinedSyntaxToken empty * "}"

DefinedSyntaxToken ::= Literal | Setting

Setting ::= Type | Value | ValueSet | Object | ObjectSet

DefinedObjectSet ::= ExternalObjectSetReference | objectsetreference

ExternalObjectSetReference ::= modulereference "." objectsetreference
```

ObjectSetAssignment ::= objectsetreference DefinedObjectClass "::=" ObjectSet

ObjectSet ::= "{" ElementSetSpec "}"

ObjectSetElements ::=
Object | DefinedObjectSet | ObjectSetFromObjects | ParameterizedObjectSet

ObjectClassFieldType ::= DefinedObjectClass "." FieldName

ObjectClassFieldValue ::= OpenTypeFieldVal | FixedTypeFieldVal

OpenTypeFieldVal ::= Type ":" Value

FixedTypeFieldVal ::= Value

InformationFromObjects ::= ValueFromObject | ValueSetFromObjects |
TypeFromObject | ObjectFromObject | ObjectSetFromObjects

ReferencedObjects ::=
DefinedObject | ParameterizedObject |
DefinedObjectSet | ParameterizedObjectSet

ValueFromObject ::= ReferencedObjects "." FieldName

ValueSetFromObjects ::= ReferencedObjects "." FieldName

TypeFromObject ::= ReferencedObjects "." FieldName

ObjectFromObject ::= ReferencedObjects "." FieldName

ObjectSetFromObjects ::= ReferencedObjects "." FieldName

InstanceOfType ::= INSTANCE OF DefinedObjectClass

InstanceOfValue ::= Value