INTERNATIONAL TELECOMMUNICATION UNION

CCITT X.722

THE INTERNATIONAL
TELEGRAPH AND TELEPHONE
CONSULTATIVE COMMITTEE

DATA COMMUNICATION NETWORKS

INFORMATION TECHNOLOGY – OPEN SYSTEMS INTERCONNECTION – STRUCTURE OF MANAGEMENT INFORMATION: GUIDELINES FOR THE DEFINITION OF MANAGED OBJECTS

Recommendation X.722

Geneva, 1992

Foreword

ITU (International Telecommunication Union) is the United Nations Specialized Agency in the field of telecommunications. The CCITT (the International Telegraph and Telephone Consultative Committee) is a permanent organ of the ITU representing some 166 member countries, 68 telecom operating entities, 163 scientific and industrial organizations and 39 international organizations and is the body which sets world telecommunications standards (Recommendations).

The approval of Recommendations by the members of CCITT is covered by the procedure laid down in CCITT Resolution No. 2 (Melbourne, 1988). In addition, the Plenary Assembly of CCITT, 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 CCITT's purview, the necessary standards are prepared on a collaborative basis with ISO and IEC. The text of CCITT Recommendation X.722 was approved on 17 January 1992. The identical text is also published as ISO/IEC International Standard 10165-4.

CCITT NOTE

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

INFORMATION NOTE

The following table gives a list of X.700 Series Recommendations which were developed in collaboration with the ISO/IEC and are identical to the corresponding International Standard. Cross-references to the corresponding ISO/IEC International Standard number and the short title of the Recommendation | International Standard are provided.

CCITT Recommendation ISO/IEC International Standard	Short Title
X.700 | 7498-4 (Note)	Management Framework
X.701 | 10040	System Management Overview
X.710 | 9595	Common Management Information Service Definition
X.711 | 9596-1	Common Management Information Protocol Specification
X.712 | 9596-2	CMIP PICS
X.720 | 10165-1	Management Information Model
X.721 | 10165-2	Definition of Management Information
X.722 | 10165-4	Guidelines for the Definition of Managed Objects
X.730 | 10164-1	Object Management Function
X.731 | 10164-2	State Management Function
X.732 | 10164-3	Attributes for Representing Relationships
X.733 | 10164-4	Alarm Reporting Function
X.734 | 10164-5	Event Management Function
X.735 | 10164-6	Log Control Function
X.736 | 10164-7	Security Alarm Reporting Function
X.740 | 10164-8	Security Audit Trail Function
Note – This Recommendation and International Standard are not identical, but are technically aligned.	

INTERNATIONAL STANDARD

CCITT RECOMMENDATION

INFORMATION TECHNOLOGY – OPEN SYSTEMS INTERCONNECTION – STRUCTURE OF MANAGEMENT INFORMATION : GUIDELINES FOR THE DEFINITION OF MANAGED OBJECTS

1 Scope

This Recommendation | International Standard provides developers of Recommendations and International Standards that contain managed object definitions with guidance that will

a) encourage consistency between managed object definitions;

b) ensure the development of such definitions in a manner compatible with the OSI management Recommendations and International Standards;

c) reduce duplication of effort in other working groups by identifying commonly useful documentation layouts, procedures and definitions.

To this end, this Recommendation | International Standard specifies

a) the relationships between the relevant OSI management Recommendations and International Standards and the definition of managed object classes, and how those Recommendations and International Standards should be used by managed object class definitions;

b) the appropriate methods to be adopted for the definition of managed object classes and their attributes, notifications, actions and behaviour, including

   1) a summary of aspects that shall be addressed in the definition;

   2) the notational tools that are recommended to be used in the definition;

   3) consistency guidelines that the definition may follow.

c) the relationship of managed object class definitions to management protocol, and what protocol-related definitions are required;

d) the recommended documentation structure for managed object class definitions.

This Recommendation | International Standard is applicable to the development of any Recommendation | International Standard which defines

a) management information which is to be transferred or manipulated by means of OSI management protocol;

b) the managed objects to which that information relates.

This Recommendation | International Standard does not specify or imply

a) any constraints on the development of managed object class definitions in terms of their functionality, the Recommendations | International Standards to which they relate, or the uses to which they are put in a particular management environment;

b) guidelines for the definition of resources; it provides guidelines only for the definition of the managed objects which provide the management view of resources.

2 Normative references

The following CCITT 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 International 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 International Standards indicated below. Members of IEC and ISO maintain registers of currently valid International Standards. The CCITT Secretariat maintains a list of the currently valid CCITT Recommendations.

2.1 Identical CCITT Recommendations | International Standards

– CCITT Recommendation X.660[1)] | ISO/IEC 9834-1[2)], *Information technology – Open Systems Interconnection – Procedures for the operation of OSI registration authorities – Part 1: General procedures.*

– CCITT Recommendation X.701 (1992) | ISO/IEC 10040 : 1992, *Information technology – Open Systems Interconnection – Systems management overview.*

– CCITT Recommendation X.720 (1992) | ISO/IEC 10165-1 : 1992, *Information technology – Open Systems Interconnection – Structure of management information : Management information model.*

– CCITT Recommendation X.721 (1992) | ISO/IEC 10165-2 : 1992, *Information technology – Open Systems Interconnection – Structure of management information : Definition of management information.*

– CCITT Recommendation X.732 (1992) | ISO/IEC 10164-3 : 1992, *Information technology – Open Systems Interconnection – Systems management : Attributes for Representing Relationships.*

– CCITT Recommendation X.733 (1992) | ISO/IEC 10164-4 : 1992, *Information technology – Open Systems Interconnection – Systems management : Alarm reporting function.*

2.2 Paired CCITT Recommendations | International Standards equivalent in technical content

– CCITT Recommendation X.200 (1988), *Reference Model of Open Systems Interconnection for CCITT Applications.*

ISO 7498 : *1984, Information processing systems – Open Systems Interconnection – Basic Reference Model.*

– CCITT Recommendation X.208 (1988), Specification of abstract syntax notation one *(ASN.1).*

ISO/IEC 8824 : 1990, *Information technology – Open Systems Interconnection – Specification of Abstract Syntax Notation One (ASN.1).*

– CCITT Recommendation X.501 (1989), *The Directory – Models.*

ISO/IEC 9594-2 : 1990, *Information technology – Open Systems Interconnection – The Directory – Part 2: Models.*

– CCITT Recommendation X.650 (1992), *Naming and Addressing for Open Systems Interconnection (OSI) for CCITT Applications.*

ISO 7498-3 : 1989, *Information processing systems – Open Systems Interconnection – Basic Reference Model – Part 3: Naming and Addressing.*

– CCITT Recommendation X.700[1)], *Management Framework Definition for Open Systems Interconnection (OSI) for CCITT Applications.*

ISO 7498-4 : 1989, *Information processing systems – Open Systems Interconnection – Basic Reference Model – Part 4: Management framework.*

[1)] Presently at state of draft Recommendation.

[2)] To be published.

[1)] Presently at state of draft Recommendation.

– CCITT Recommendation X.710 (1991), *Common Management Information Service Definition for CCITT Applications.*

  ISO/IEC 9595 : 1990, *Information technology – Open Systems Interconnection – Common management information service definition.*

– CCITT Recommendation X.711 (1991), *Common Management Information Protocol Specification for CCITT Applications.*

  ISO/IEC 9596-1 : 1991, *Information technology – Open Systems Interconnection – Common management information protocol – Part 1: Specification.*

3 Definitions

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

3.1 Basic reference model definitions

This Recommendation | International Standard makes use of the following terms defined in CCITT Rec. X.200 | ISO 7498:

a) (N)-connection;

b) (N)-entity;

c) (N)-layer;

d) (N)-service-access-point;

e) open system;

f) systems management.

3.2 Naming and addressing definitions

This Recommendation | International Standard makes use of the following term defined in CCITT Rec. X.650 | ISO 7498-3:

a) (N)-selector

3.3 Management framework definitions

This Recommendation | International Standard makes use of the following terms defined in CCITT Rec. X.700 | ISO/IEC 7498-4:

a) managed object;

b) (N)-layer operation.

3.4 Systems management overview definitions

This Recommendation | International Standard makes use of the following terms defined in CCITT Rec. X.701 | ISO/IEC 10040:

a) agent;

b) generic definitions;

c) managed object class;

d) management information;

e) manager;

f) (N)-layer management protocol;

g) notification;

h) notification type;

i) (systems management) operation;

j) systems management (application) protocol.

3.5 Management information model definitions

This Recommendation | International Standard makes use of the following terms defined in CCITT Rec. X.720 | ISO/IEC 10165-1:

a) action;

b) actual class;

c) attribute group;

d) attribute identifier;

e) attribute type;

f) attribute value set;

g) behaviour;

h) characteristic;

i) conditional package;

j) containment;

k) inheritance;

l) inheritance hierarchy;

m) initial value managed object;

n) instantiation;

o) mandatory package;

p) multiple inheritance;

q) name binding;

r) package;

s) parameter;

t) permitted value set;

u) relative distinguished name;

v) required value set;

w) specialization;

x) subclass;

y) subordinate object;

z) superclass;

aa) superior object.

3.6 CMIS definitions

This Recommendation | International Standard makes use of the following terms defined in CCITT Rec. X.710 | ISO/IEC 9595:

a) attribute;

b) Common Management Information Services.

3.7 ASN.1 definitions

This Recommendation | International Standard makes use of the following terms defined in CCITT Rec. X.208 | ISO/IEC 8824:

a) object identifier;

b) sequence type;

c) sequence-of type;

d) set type;

e) set-of type;

f) subtype;

g) type;

h) type reference name;

i) value reference name.

3.8 Additional definitions

3.8.1 managed object class definition: A set of attribute, operation, notification and behaviour definitions to which a managed object class name has been allocated, documented by the use of a managed object class template and one or more other templates, of the types defined in this Recommendation | International Standard, which are directly or indirectly referenced by the managed object class template. The definition of a managed object class includes all elements of definition inherited from the superclass(es) of the managed object class and all elements of definition that constitute specialization(s) of the superclass(es).

3.8.2 template: A standard format for the documentation of name bindings, and managed object class definitions and their components, such as packages, parameters, attributes, attribute groups, behaviour definitions, actions or notifications.

3.8.3 directory object class: An object class, as defined in CCITT Rec. X.501 | ISO/IEC 9594-2.

4 Abbreviations

ASN.1	Abstract Syntax Notation One
CMIP	Common Management Information Protocol
CMIS	Common Management Information Services
DMI	Definition of Management Information
IVMO	Initial Value Managed Object
MOCS	Managed Object Conformance Statement
(N)-SAP	(N)-service-access-point
OSI	Open Systems Interconnection
PDU	Protocol Data Unit
SAP	Service Access Point
SDU	Service Data Unit
SMI	Structure of Management Information
RDN	Relative Distinguished Name

5 Conventions

Distinctive typefaces are used throughout this Recommendation | International Standard where the text makes use of ASN.1 notation or `the notational tools defined in clause 8`.

No externally defined conventions are used in this Recommendation | International Standard.

6 Global issues

6.1 Relationship integrity

When defining managed object classes, it is important to consider situations where there are consistency requirements that will apply to instances of those classes, for example, situations where the behaviour of a managed object is constrained by rules that depend not only upon its own state but also upon the state of other managed objects in the system. Any such constraints must be expressed as behaviour associated with the managed object classes concerned.

A particular case where the definitions associated with the instantiation of a managed object shall explicitly define consistency rules is that of the Delete operation; for this operation, such consistency rules are specified in name binding(s) associated with the managed object class. The effect of a Delete operation shall be defined in such a way that it is clear under what circumstances deletion is permitted, and what the consequences of deletion are. In particular, the name binding shall specify whether deletion of an instance of the class is permitted when contained managed objects still exist, and what rules apply in cases where other (non-containment) relationships exist between the managed object being deleted and other managed objects, such as those that may exist as a consequence of the presence of relationship attributes as defined in CCITT Rec. X.732 | ISO/IEC 10164-3. The consistency rules that are applied for deletion shall be such that the delete operation cannot result in inconsistent relationships. As these consistency rules are specified as part of a name binding, the rules that apply to the deletion of a given managed object are established at the time that the managed object is instantiated.

6.2 Inherited characteristics

The process of inheritance results in the inclusion of all characteristics of the superclass(es) of a managed object class in the managed object class definition. This rule is applied recursively, terminating at the apex of the inheritance hierarchy, known as top. A given managed object class therefore includes all characteristics that are part of the definition of top, plus all characteristics that are added in the process of defining any subclasses of top that form part of the managed object class's inheritance hierarchy.

6.3 Optionality

In general, the provision of options in managed object class definitions is discouraged, on the grounds that interworking becomes more difficult as the number of options increases. As stated in CCITT Rec. X.720 | ISO/IEC 10165-1, the managed object class definition may include conditional packages which are present in an instance of the managed object class if the specified condition applies. It is the intention that the conditions applied to these packages should relate to standardized features of the resource to which the managed object class applies, or to optional management functions supported by the management system.

6.4 Registration

The process of defining managed object classes requires the assignment of globally unique identifiers, known as object identifiers, to various aspects of the managed object class, such as the managed object class name, attribute types, etc. The values of these identifiers are used in management protocols to uniquely identify aspects of managed objects and their associated attributes, operations and notifications. It is therefore a necessary precursor to the development of a managed object class definition that the standards body or organization concerned should identify or establish a suitable registration mechanism that is capable of issuing object identifier values for its use. CCITT Rec. X.208 | ISO/IEC 8824 specifies the structure of the object identifier and the values of the initial arcs; further information on the establishment of registration mechanisms and registration authorities may be found in CCITT Rec. X.660 | ISO/IEC 9834-1.

Once an item of management information has been assigned an object identifier value, it is a requirement that any revision of the definition of that item shall not alter the semantics of the information. In practice, this means that editorial changes to registered management information definitions are permissible, but the definitions shall not be changed in ways that would be visible in protocol.

All object identifier values registered in systems management Recommendations | International Standards are allocated under the arc

{joint-iso-ccitt ms(9)}

The allocation of arcs below **{joint-iso-ccitt ms(9)}** is defined by this Recommendation | International Standard. The arcs below **{joint-iso-ccitt ms(9)}** are allocated on a per-Systems-Management-standard basis, as shown in Table 1.

Table 1 – Allocation of arcs below {joint-iso-ccitt ms(9)}

Arc	**Standard**
`smo(0)`	Systems management overview, CCITT Rec. X.701 | ISO/IEC 10040
`cmip(1)`	Common management information protocol, CCITT Rec. X.711 | ISO/IEC 9596-1
`function(2)`	Systems management functions, CCITT Rec. X.7NN | ISO/IEC 10164-X, where X is the part number of the standard in the ISO/IEC numbering scheme, and X.7NN is the corresponding CCITT Recommendation number.
`smi(3)`	Structure of management information, CCITT Rec. X.72N | ISO/IEC 10165-X, where X is the part number of the standard in the ISO/IEC numbering scheme, and X.72N is the corresponding CCITT Recommendation number.

The allocation of arcs below this level is defined in 6.4.1 to 6.4.5. Further arcs required by existing or future systems management standards will be allocated, as required, by means of amendments to this Recommendation | International Standard.

NOTE – The scheme for the allocation of object identifier values described in this clause and its sub-clauses applies only to the allocation of object identifier values in the systems management standards developed jointly by ISO/IEC JTC1 SC21/WG4 and CCITT SGVII. It is necessary for other standards bodies or organizations that require to allocate object identifier values in the course of the development of management standards to establish their own allocation schemes under an appropriate registration authority. The structure adopted by the systems management standards activity may serve as a useful example of how a suitable allocation scheme may be established, but the final choice of allocation scheme is the responsibility of the organization concerned. In order to improve human readability of object identifier values, it is recommended that the name and number form for the representation of object identifier values, as defined in CCITT Rec. X.208 | ISO/IEC 8824, be used wherever possible.

6.4.1 Object identifier allocation for systems management overview

NOTE – The systems management overview is responsible for the allocation of these arcs; they are included for information only.

Below **{joint-iso-ccitt ms(9) smo(0)}**, the following arcs are allocated for the registration of application contexts, abstract syntaxes and ASN.1 module identifiers, as shown in Table 2.

Table 2 – Allocation of arcs below {joint-iso-ccitt ms(9) smo(0)}

Arc	**Purpose**
`applicationContext(0)`	Allocation of application context identifiers
`negotiationAbstractSyntax(1)`	Allocation of version identifiers to the negotiation abstract syntax
`asn1Modules(2)`	Allocation of ASN.1 module identifiers

Below **{joint-iso-ccitt ms(9) smo(0) applicationContext(0)}** further arcs are allocated, as specified by CCITT Rec. X.701 | ISO/IEC 10040, for the registration of particular application context identifiers, as shown in Table 3.

Table 3 – Allocation of arcs below {joint-iso-ccitt ms(9) smo(0) applicationContext(0)}

Arc	**Purpose**
`systems-management(2)`	Identifier of the Systems management application context

Below **{joint-iso-ccitt ms(9) smo(0) negotiationAbstractSyntax(1)}** further arcs are allocated, as specified by CCITT Rec. X.701 | ISO/IEC 10040, for the registration of particular versions of the negotiation abstract syntax, as shown in Table 4.

Table 4 – Allocation of arcs below {joint-iso-ccitt ms(9) smo(0) negotiationAbstractSyntax(1)}

Arc	**Purpose**
`version1(1)`	Identifies version 1 of the negotiation abstract syntax

Below **{joint-iso-ccitt ms(9) smo(0) asn1Modules(2)}** further arcs are allocated, as specified by CCITT Rec. X.701 | ISO/IEC 10040, for the registration of particular ASN.1 module identifiers, as shown in Table 5.

Table 5 – Allocation of arcs below {joint-iso-ccitt ms(9) smo(0) asn1Modules(2)}

Arc	**Purpose**
`negotiationDefinitions(0)`	Allocation of version identifiers to the ASN.1 module that contains the definitions associated with the negotiation abstract syntax

Below **{joint-iso-ccitt ms(9) smo(0) asn1Modules(2) negotiationDefinitions(0)}** further arcs are allocated, as specified by CCITT Rec. X.701 | ISO/IEC 10040, for the registration of particular versions of the ASN.1 module, as shown in Table 6.

Table 6 – Allocation of arcs below {joint-iso-ccitt ms(9) smo(0) asn1Modules(2) negotiationDefinitions(0)}

Arc	**Purpose**
`version1(1)`	Identifies Version 1 of the ASN.1 module that contains the definitions associated with the negotiation abstract syntax

6.4.2 Object identifier allocation for CMIP

NOTE – CMIP is responsible for the allocation of these arcs; they are included for information only. CMIP version 1 is obsolete and has been superseded by version 2. Version 1 was documented in ISO/IEC 9596, for which there is no corresponding CCITT Recommendation.

Below **{joint-iso-ccitt ms(9) cmip(1)}**, arcs are allocated for each version of CMIP, as described in 6.4.2.1 and 6.4.2.2.

6.4.2.1 CMIP version 1

Below **{joint-iso-ccitt ms(9) cmip(1)}**, arcs are allocated for version 1 of CMIP, as shown in Table 7.

Table 7 – Allocation of arcs below {joint-iso-ccitt ms(9) cmip(1)} for CMIP version 1

Arc	**Purpose**
`version1(1)`	Allocation of object identifiers for CMIP version 1

Below **{joint-iso-ccitt ms(9) cmip(1) version1(1)}**, arcs are allocated for the purposes described in ISO/IEC 9596, as shown in Table 8.

Table 8 – Allocation of arcs below {joint-iso-ccitt ms(9) cmip(1) version1(1)}

Arc
`aAssociateUserInfo(1)`
`aAbortUserInfo(2)`
`protocol(3)`
`abstractSyntax(4)`

6.4.2.2 CMIP version 2

Below **{joint-iso-ccitt ms(9) cmip(1)}**, arcs are allocated for version 2 of CMIP, as shown in Table 9.

Table 9 – Allocation of arcs below {joint-iso-ccitt ms(9) cmip(1)} for CMIP version 2

Arc	**Purpose**
`modules(0)`	Allocation of object identifiers for CMIP ASN.1 modules
`cmip-pci(1)`	Allocation of object identifiers for CMIP protocol control information

Below **{joint-iso-ccitt ms(9) cmip(1) modules(0)}**, arcs are allocated for the purposes described in CCITT Rec. X.711 | ISO/IEC 9596-1, as shown in Table 10.

Table 10 – Allocation of arcs below {joint-iso-ccitt ms(9) cmip(1) modules(0)}

Arc
`aAssociateUserInfo(1)`
`aAbortUserInfo(2)`
`protocol(3)`

Below **{joint-iso-ccitt ms(9) cmip(1) cmip-pci(1)}**, arcs are allocated for the purposes described in CCITT Rec. X.711 | ISO/IEC 9596-1, as shown in Table 11.

Table 11 – Allocation of arcs below {joint-iso-ccitt ms(9) cmip(1) cmip-pci(1)}

Arc
`reserved1(1)`
`reserved2(2)`
`reserved3(3)`
`abstractSyntax(4)`

6.4.3 Object identifier allocation for function standards

Below **{joint-iso-ccitt ms(9) function(2)}**, arcs are allocated to identify each function Recommendation | International Standard as shown in Table 12.

Table 12 – Allocation of arcs below {joint-iso-ccitt ms(9) function(2)}

Arc	**Standard**
`partX(X)`	CCITT Rec. X.7NN | ISO/IEC 10164-X where X is the part number of the Systems Management Function in the ISO/IEC numbering scheme, and X.7NN is the corresponding CCITT Recommendation number.

Below **{joint-iso-ccitt ms(9) function(2) partX(X)}**, arcs are allocated as shown in Table 13.

Table 13 – Allocation of arcs below {joint-iso-ccitt ms(9) function(2) partX(X)}

Arc	Purpose
`standardSpecificExtension(0)`	Standard-specific extensions to the allocation scheme
`functionalUnitPackage(1)`	Allocation of functional unit package identifiers
`asn1Module(2)`	Allocation of ASN.1 module identifiers
`managedObjectClass(3)`	Allocation of managed object class identifiers
`package(4)`	Allocation of package identifiers
`parameter(5)`	Allocation of parameter identifiers
`nameBinding(6)`	Allocation of name binding identifiers
`attribute(7)`	Allocation of attribute identifiers
`attributeGroup(8)`	Allocation of attribute group identifiers
`action(9)`	Allocation of action types
`notification(10)`	Allocation of notification types

Within each function Recommendation | International Standard, further arcs may be allocated below this level (e.g., to allocate particular attribute identifiers), as required by the function Recommendation | International Standard.

6.4.4 Object identifier allocation for SMI standards

Below **{joint-iso-ccitt ms(9) smi(3)}**, arcs are allocated to identify each SMI Recommendation | International Standard as shown in Table 14.

Table 14 – Allocation of arcs below {joint-iso-ccitt ms(9) smi(3)}

Arc	Standard
`partX(X)`	CCITT Rec. X.72N | ISO/IEC 10165-X where X is the part number of the standard in the ISO/IEC numbering scheme, and X.72N is the corresponding CCITT Recommendation number.

Below **{joint-iso-ccitt ms(9) smi(3) partX(X)}**, arcs are allocated as shown in Table 15.

Table 15 – Allocation of arcs below {joint-iso-ccitt ms(9) smi(3) partX(X)}

Arc	**Purpose**
`standardSpecificExtension(0)`	Standard-specific extensions to the allocation scheme
`asn1Module(2)`	Allocation of ASN.1 module identifiers
`managedObjectClass(3)`	Allocation of managed object class identifiers
`package(4)`	Allocation of package identifiers
`parameter(5)`	Allocation of parameter identifiers
`nameBinding(6)`	Allocation of name binding identifiers
`attribute(7)`	Allocation of attribute identifiers
`attributeGroup(8)`	Allocation of attribute group identifiers
`action(9)`	Allocation of action types
`notification(10)`	Allocation of notification types

Within each Recommendation | International Standard, further arcs may be allocated below this level (e.g., to allocate particular attribute identifiers), as required by that Recommendation | International Standard.

6.4.5 Object identifier allocation for actual class

The object identifier value

{joint-iso-ccitt ms(9) smi(3) part4(4) managedObjectClass(3) actualClass(42)}

is allocated by this Recommendation | International Standard to convey the semantics of *actual class* as defined in CCITT Rec. X.720 | ISO/IEC 10165-1. When used to specify the base managed object class in a CMIS operation service request, this object identifier value indicates that the recipient of the systems management operation shall respond as a member of its actual class.

6.5 Conformance

CCITT Rec. X.701 | ISO/IEC 10040 specifies general conformance related requirements for management information standards.

6.6 Complexity of managed object definitions

Through the modelling process, complexity in the managed object definitions should be minimized. In any case, management operations should be no more complex than the corresponding properties of the OSI entity involved.

6.7 Managed object creation and deletion

Creation and deletion of managed object instances may occur in the following ways:

– managed objects may be created and deleted by management protocol interactions. Create and delete operations are defined for this purpose, with associated semantics;

– managed objects may be created and deleted as a result of the operation of the resource to which they relate, typically a protocol machine. Create and delete operations would not be defined in this case. An example is the representation for management purposes of connections;

- managed objects may be created and deleted by other means. Create and Delete operations are not defined for them. An example is a managed object which is always created automatically when a piece of equipment is initialized, and which cannot be deleted through management.

The choice among these three methods for managed object creation may differ from the choice for managed object deletion.

In some cases, there may be only one means by which a managed object of a particular class may be created or deleted; in other cases, there may be the possibility to create or delete managed objects of a particular class by means of more than one of these mechanisms.

6.7.1 Initial value managed objects

When a managed object is created, it may be desirable to provide the capability to allocate default values to be used on creation which are themselves subject to modification by management operation. This may be achieved by the specification of an Initial Value Managed Object (IVMO), whose attributes are modifiable by management operation, and which is capable of providing default values for corresponding attributes on the creation of instances of another managed object class.

When a new managed object is created using an IVMO, the values of attributes in the IVMO are used as initial values of corresponding attributes in the new managed object. The managed object class definition may specify how the IVMO is selected. The specification of the IVMO shall define the circumstances in which it provides the initial values, how it provides the initial values, and to which attributes the initial values it provides are applicable.

When management operations are used to modify the attributes of an IVMO, attributes of managed objects previously created using that IVMO are not affected. Similarly, management operations performed upon the attributes of managed objects created using an IVMO do not affect the attributes of the IVMO.

6.7.2 Sources of initial attribute values

Initial attribute values for managed objects, used at creation time, are obtained from a number of sources, as defined in CCITT Rec. X.720 | ISO/IEC 10165-1. When an attribute represents a specific value that must be consistent with an underlying resource, this value forms the mandatory initial value used at creation time.

7 General principles for managed object definition

The general principles described here are intended for the guidance of managed object definers, and to promote consistency between managed object definitions; for this reason, managed object definers are encouraged to apply the guidance offered wherever it is applicable.

7.1 Commonality

Managed object definers should strive to identify and use as a basis

- common managed object classes, defined in CCITT Recommendations | International Standards;
- common managed object classes and other properties as defined in CCITT Rec. X.721 | ISO/IEC 10165-2.

Managed object definers should also strive to consider and re-use definitions originating in other groups in order to increase commonality of definitions. This objective may be met by development of models of things to be managed that are common to a number of groups of managed object definers.

7.2 What to manage

The definition of managed object classes and their components should fulfil clearly justified requirements related to particular management objectives. Such requirements are likely to include the management of peer protocol aspects of the operation of a layer or sub-layer and perceived problems not specifically reported by the underlying service provider across the service boundary (e.g., Quality Of Service provided by underlying service not meeting acceptable levels). It is important to retain justification for each management objective during the development of the management information definitions. Comments should be used to explain how each component of a management information definition. Comments should be used to explain how each component of a management information definition, (e.g., managed object classes, attributes, operations, notifications, etc.) relates to this justification.

Things of interest to management should be recorded via the managed objects that represent the resources where the things happen. That is, if a managed object has been defined representing a particular resource (for example, a connection), then information concerning that resource should be reflected through the corresponding managed object(s) and not elsewhere.

7.3 Structuring

A number of techniques are available for representing structure in managed objects, to reflect groupings of data or functionality. Each of these techniques has advantages and disadvantages; the choice of the most appropriate technique(s) for a particular specification requirement depends on a number of criteria, as described below.

The structuring techniques described in CCITT Rec. X.720 | ISO/IEC 10165-1 include

– attribute groups;

– subclasses (specialization);

– multiple inheritance;

– contained managed objects;

– packages.

Groupings of attributes, operations and notifications may be defined which are present or absent based on a standardized condition, such as the choice of particular options in a base standard. Such groupings of functionality are either present or absent as a whole. Groupings of functionality may arise as a consequence of option selection in the layer Recommendation | International Standard for the resource (e.g., provision of Transport Class 4), leading to additional management requirements or capability, or as a consequence of support for a defined management function (e.g., accounting). These groupings of functionality are defined using the conditional package techniques provided by the managed object class template.

One important criterion in the choice of structuring techniques is the static or dynamic presence of the grouping. If the presence of the grouping is fixed at specification time, then the use of attribute groups, subclasses, multiple inheritance, or contained managed objects may be appropriate. If the presence is fixed at implementation, installation or instantiation time, then the use of contained managed objects or conditional packages may be appropriate. If the presence of the grouping can change through the life of the containing/encapsulating managed objects, then the use of contained managed objects that are dynamically created and deleted may be appropriate.

Another criterion is whether there are multiple instances of the grouping within the managed object. Where this is the case, the use of contained managed objects is appropriate; otherwise, any of the five structuring techniques may be appropriate.

7.4 Managed objects

7.4.1 Instantiation of superclasses

Managed object classes that are never instantiated may be defined to provide a common base from which to specialize subclasses; for example, a generic virtual circuit managed object class may be defined, of which permanent and switched virtual circuits could be subclasses.

In some cases, particularly where subclasses are defined in order to revise a standard, there may be superclasses of which instances may be created.

7.4.2 Unrestricted superclasses

The rules for inheritance limit the ways in which the required value sets and permitted value sets of attributes of a managed object class can be modified when defining a subclass of that class. In the same way, the rules limit the ability to add parameters to actions and notifications. These limitations ensure that the subclass is compatible with the superclass.

For this reason, when defining a managed object class that is expected to be a superclass of subsequent managed object classes, it is useful to make provision for these kinds of extension. Although not all extensions can be anticipated and provided for, the following techniques allow for a wide variety of extensions while maintaining compatibility:

– define the syntax (type) of each attribute to include all values which might reasonably fit within the semantics of the attribute, even if some of those values are not immediately required or desired;

– provide extension capabilities in each action and notification definition;

– define an "unrestricted superclass" that includes these elements, without further restrictions, as the basis for defining more restricted subclasses. For attributes, this means an empty required value set and a permitted value set equal to the attribute syntax;

- define specific subclasses of this unrestricted superclass, which place the required restrictions on attributes, actions and notifications.

The managed object definer may choose to provide extension capabilities in only some of the attributes, actions and notifications of an unrestricted superclass.

7.5 Attributes

7.5.1 Attribute value sets

In some cases, options in the base Recommendation | International Standard permit the value set of an attribute to vary in accordance with implementation choices. A typical example of this would be where the base Recommendation | International Standard permits a wide range of packet sizes, but a conformant implementation of the Recommendation | International Standard may support a more limited range. Where this situation exists, the attribute behaviour definition shall identify what the possibilities are.

It may be necessary to define null values as permitted values in the value set of an attribute, or, in the case of attributes of an IVMO, it may be necessary to define attribute values to which particular semantics are attached, such as "create managed object with null as the value of the corresponding attribute" or "ignore this attribute as a source of an initial value". Techniques for defining such values include defining the abstract syntax as a choice type, where one choice defines the normal value set of the attribute and other choices define values to which particular semantics are attached.

Definition of the permitted value set of an attribute may be achieved in a number of ways, including

- defining the attribute value set statically, as part of the definition of the managed object class;
- defining a second attribute, whose value indicates the value set that the attribute may contain.

The former technique minimizes the number of attribute definitions associated with a managed object class; however, if a number of variants of the attribute are required, the latter technique avoids the necessity to define multiple subclasses to handle each possible value set variant.

7.5.2 Attribute types

Structured attributes, where the sequence type, sequence-of type or set type is used as the base type in attribute syntax definition, should only be used where it is not required to individually modify elements of the attribute, as these ASN.1 types correspond to single-valued attribute types. Where it is necessary to address a number of attributes together while maintaining the ability to manipulate each one individually, attribute groups may be defined, and if needed, action and behaviour definitions may be used to clarify any dependence between group members.

NOTE – This does not imply that there is a behaviour specification specific to the attribute group itself that does not also apply to the attributes when treated individually.

7.6 Attribute value relationships

The value of an attribute may be constrained according to some function of other attribute values. All relationships of this nature shall be identified.

Where an attribute value is constrained by other attributes in the *same* managed object, a synchronization requirement may exist for a single management operation, where failure to change the value of one or more of the related attributes would result in the related attributes having illegal values. If such a requirement exists, it shall be documented as part of the behaviour definition of the managed object class.

Where an attribute value is constrained by other attributes in *different* managed objects, if a synchronization requirement also exists, it shall be documented in behaviour associated with the managed object class. In this case, and where the managed objects are all in the same managed system and a single management operation can modify the attributes, the requirement is enforced by the use of the atomic cross-object synchronization capability of CMIS.

The general problem of synchronization across multiple management operations, across different attributes in different managed objects or across multiple managed systems cannot be enforced by means of current systems management protocol alone.

7.7 Modelling of SAPs

There is a general requirement to represent, as part of the managed object structure associated with the layers, the relationship between (N)-entities, (N)-selectors and (N+1)-entities. A number of possible solutions exist, for example

- model the relationship as information contained in managed objects of the (N+1)-layer;

- model the relationship as information contained in managed objects of the (N)-layer;
- model the relationship as information contained in managed objects that do not belong to either layer, i.e., managed objects that are common to all layers.

The practice recommended by this Recommendation | International Standard is that the second approach should be adopted. Specifically, (N)-SAPs should be represented by individual managed objects, which would have address (and other) information as attributes, together with relationship attributes pointing to the (N)- and (N+1)-entity managed objects associated with the (N)-SAPs. In order to enforce the consistency requirements on selectors necessary to make OSI addressing unambiguous, it is proposed that (N)-SAP managed objects should be **contained within** the managed objects corresponding to the (N)-entities to which they are bound.

NOTE – The consistency requirement referred to here is the requirement that the address of an (N)-entity combined with an (N)-selector is required to uniquely identify an (N+1) entity (or a set of (N+1)-entities of the same type). Given that this requirement amounts to a uniqueness requirement placed on the allocation of (N)-selector values to be used in the context of a given (N)-entity, maintenance of this consistency requirement may be more simply achieved if such selector information is maintained by the (N)-entity rather than the (N+1)-entities.

7.8 Statistics

7.8.1 Consistency

Managed object definers should strive to achieve consistency of statistics across layers, by adopting some principles based on CCITT Rec. X.200 | ISO 7498, in particular, the concept of recording information that is of interest to management via managed objects that represent the resources to which the information relates.

Candidate features of the (N)-layer for which statistics may be recorded are

- local errors;
- successful peer-to-peer exchanges;
- peer-to-peer failures;
- service rejections.

For example, applying the previously defined principles in this document to the definition of a Connection in CCITT Rec. X.200 | ISO 7498 leads to the identification of the following primary statistics:

- number of (N)-entity connections established with other (N)-layer peer entities;
- number of (N)-entity connection establishment local failures;
- number of (N)-entity connection establishment peer-to-peer negotiation failures;
- number of underlying service provider (N–1)-connection establishment rejections.

This set of statistics provide a consistent view of what is happening at each layer (in the connection-oriented case) without duplication of counters.

NOTE – Similar models are required for errors, disconnections, etc.

7.8.2 PDU counters

Managed object definers should specify counters of (N)-layer PDUs (and PDU octets) rather than SDUs (and SDU octets).

NOTE – It is thought likely that it is only necessary to count the number of PDU octets at a limited number of (N) -layers.

7.8.3 Overlaps

Managed object definers should strive to achieve consistency and avoid unnecessary duplication or overlap of statistics. For example, maintain a count of first PDU send requests and a count of retry PDU send requests, so that it will not be necessary to increment both counters at the same time. The sum of these counters yields the total PDUs sent.

7.8.4 Non-resettable counters

Non-resettable counters are recommended, as they permit multiple observers without the need for complex interlocking mechanisms associated with reset co-ordination.

7.8.5 Event counters

A count should be maintained for managed resource events that result in the issuing of a notification, as the generation of a CMIS M-EVENT-REPORT may be suppressed by the event forwarding discriminators.

7.9 Counters

To manage a counter, its modulus must be known, otherwise a manager cannot determine the value at which the counter "wraps". There are therefore at least four possibilities when defining counters:

- counters never wrap;
- the modulus is fixed as part of the managed object class definition;
- the modulus is defined in a related attribute;
- the modulus is defined on a per-implementation basis and is specified in the MOCS.

NOTE – In the managed object classes defined by ISO/IEC JTC1 for layers 1 to 4, the approach of using counters that never wrap has been adopted.

7.10 Timers

There are advantages to be gained in adhering to a common specification of the precision with which systems must store the values of timer attributes used in management communications. The relationship between these attribute values and the actual operation of the timers in the protocol is documented in behaviour statements.

NOTE – In the managed object classes defined by ISO/IEC JTC1 for layers 1 to 4, in order to encompass a large enough range without inordinate precision, a floating point representation is used to express timer values, with a mantissa length of 16 bits and an exponent length of up to 16 bits. (This does not imply a requirement that floating point *arithmetic* will be performed.) Systems are expected to be able to store values with this precision. Other constraints permitting, a request to set a timer attribute to that precision must be accepted.

7.11 Updating of attributes

Managed object class definers should ensure that, when defining attributes which can be updated both by management operation and by the normal operation of the resource, the effects of concurrent update are defined. In particular, the effect of a replace attribute value operation may be lost if the resource also updates the same attribute.

7.12 Precision of attributes

A managing system may attempt to set an attribute value to a higher precision than is supported by an agent system. Such higher precision values may be approximated to a neighbouring value of the defined precision.

7.13 Managed object identification

Each managed object class definition for which instances may exist shall include at least one attribute suitable for use as a naming attribute for the managed object. A suitable attribute is a mandatory attribute that is testable for equality, whose semantics permit its value to remain fixed for the lifetime of each managed object that uses it for naming, and whose identifier and value shall uniquely identify the managed object from all others named by the same superior object.

When a managed object is deleted, the value assigned to its naming attribute becomes available for re-use, to identify subsequent managed objects created within the same superior object.

If it is necessary to guarantee that one instance of a managed object class remains distinguishable from all other instances of that class after it has been deleted, it is necessary to define an additional attribute to be included in the managed object class definition - a *unique identification attribute* - whose semantics ensure unique identification is maintained over time. Managed object classes that do not have this requirement need not include a unique identification attribute in their definition.

The unique identification attribute shall be read-only and, when included in a managed object, be included in notifications emitted by that managed object.

7.14 Notifications

7.14.1 Denial of service

Notifications should not be issued regarding the withdrawal of an underlying service, as the managed object(s) representing the underlying service shall be responsible for reporting any abnormal reason for such termination. This is intended to inhibit an abnormal termination rippling up the layers and generating phantom notifications.

7.14.2 Information conservation

Notifications contain information associated with an event that may otherwise be lost by maintaining statistics only. For example

- the header field of a received PDU for which a protocol error has been detected;
- statistics for a connection that is about to be terminated, for a specific reason;
- timing of a sequence of particular events.

7.15 Use of operations

A managed object class definition should include appropriate operations. For invocations by the managed system, notifications are specified. For invocations by the managing system, operations are specified according to their direct effect on managed objects in the managed system, as follows:

a) if the direct effect is to create an instance of a managed object class, the Create operation is used. The Create operation is not used for complex activities which require the coordinated creation of many managed objects, when a managed object is created as a side effect of a change in another managed object, or when support managed objects are created due to a change in the state of another managed object;

b) if the direct effect is to delete a managed object, then the Delete operation is used;

c) if the direct effect is to set the values of attribute(s) of a managed object to a specified value(s), then the Replace attribute value operation is used;

d) if the direct effect is to set the values of attribute(s) to default value(s) (provided such defaults have been defined), then the Replace with default value operation is used;

e) if the direct effect is to add or remove members of set-valued attribute(s) of a managed object, then the Add member or Remove member operations are used;

f) if the direct effect is to retrieve attribute value(s) from a managed object, then the Get attribute value operation is used;

g) in other cases, e.g., where there is no direct effect, or the direct effect is a combination of those listed, or where there is some other effect on the object as a whole, then an Action operation is used. Examples of where an action operation is used include

   1) where it is not possible to define the required operation on a set of managed objects by means of scoping and filtering techniques in conjunction with Get attribute value, Replace attribute value, Replace with default value, Create, Delete, Add member or Remove member operations;

   2) where creation of more than one managed object as an atomic operation is required;

   3) where several objects without common attributes are to be affected;

   4) where the request or response for the operation contains information that is not modelled by the attributes of managed objects.

The concepts of direct effect and indirect effect are discussed in CCITT Rec. X.720 | ISO/IEC 10165-1.

8 Notational tools for managed object definition

8.1 Overview of notational tools

The templates defined in this clause provide a common set of tools and a common notation for the representation of various aspects of a managed object class definition and its associated naming structure. Formal definitions of each template are contained in 8.3 to 8.11; the syntactic conventions used in these formal definitions are specified in 8.2. These formal definitions define the constructs that each template shall or may contain and the order in which the constructs shall appear within each template. Examples of the use of these tools are contained in Annex A.

The structure and behaviour of a managed object class is primarily defined by means of the Managed Object Class template. This template identifies the inheritance relationships that exist between the managed object class and other managed object classes, and the packages of behaviour, attributes, notifications and operations that are included in the managed object class definition. In order to allow re-use of parts of this specification in the specification of other managed object classes, additional templates are defined to provide for the specification of attributes (individual and group), behaviour, actions, notifications, parameters and packages. These further templates are "called up" by other templates by means of the referencing mechanism defined in 8.2; this mechanism allows references to be made in one standard to specifications that are contained in other standards, hence allowing generic definitions to be made available for use in managed object class definitions in addition to local definitions. These further templates may also be included "in-line" if so desired.

The naming of a managed object class is defined by means of the Name Binding template. This template identifies the managed object class being named and defines a relative distinguished name that can be used to name instances of the class in the context of a particular superior class. This template also provides for the specification of relationships that exist between two object classes as a consequence of a particular name binding.

8.2 Conventions used in template definitions

The start of a template consists of a `template-label` and a `TEMPLATE-NAME`. A template contains one or more constructs, each of which is named by a `CONSTRUCT-NAME` and each may have a `construct-argument`. The `construct-argument` may in turn consist of a number of elements, as called for by the definition of the particular construct. Each instance of use of a template declares a unique `template-label` by which that instance may be referenced from other templates, and if the `REGISTERED AS` construct is present, assigns a value of an ASN.1 object identifier under which the instance has been registered. The semicolon character is used to mark the end of each construct (except `REGISTERED AS` and `DEFINED AS`) and to mark the end of a template.

In order to simplify the structure of the templates, for example where the same syntactic structure is repeated in a template definition, supporting syntactic definitions may be defined. If any such supporting syntactic definitions are required in order to complete the template definition, these are introduced by the keywords

```
supporting productions
```

at the end of the template definition, and consist of a number of productions of the form

```
<definition-label> -> <syntactic-definition>
```

The `definition-label` allows the definition to be referenced by the template definition or by other `supporting productions` and the `syntactic-definition` gives the expansion of the definition, using the syntactic conventions defined in the remainder of this sub-clause. In the case of a `syntactic-definition` that defines a number of alternative strings, references to the `supporting production` that contains it are assumed to be evaluated to a single string chosen from the list of alternatives.

The template definitions are based on the following syntactic conventions

a) all terminal symbols and keywords that form part of a template definition are case-sensitive;

b) where necessary in order to render the syntax of a template unambiguous, each element of a template shall be separated from adjacent elements by one or more delimiters. Valid delimiters are the space character, the end of a line, a blank line or a comment. One or more delimiters shall appear between

   1) a `template-label` and a `TEMPLATE-NAME`;

   2) a `TEMPLATE-NAME` and a `CONSTRUCT-NAME`;

   3) a `CONSTRUCT-NAME` and a `construct-argument`.

   One or more delimiters may be introduced between any other pair of elements within a template, and where a `construct-argument` consists of a number of distinct elements, delimiters may be introduced between them. Delimiters shall not otherwise be introduced within elements of the template unless the definition of the template explicitly permits their introduction;

c) spaces, blank lines, comments and the end of a line are significant only as delimiters;

d) a comment is introduced by the character pair

   ```
   --
   ```

   and is terminated either by the character pair

   ```
   --
   ```

   or by the end of a line, whichever occurs first. A comment is equivalent to a space for the purposes of interpreting the templates defined in this Recommendation | International Standard. Comments have no normative significance;

e) the character

   ```
   ;
   ```

   shall be used to mark the end of each construct within a template (except `REGISTERED AS` and `DEFINED AS`) and to mark the end of a template;

f) the notation used for representing object identifiers shall be the value notation defined in CCITT Rec. X.208 | ISO/IEC 8824 for representing object identifier values; i.e., the production

```
object-identifier -> <ObjectIdentifierValue>
```

is assumed to be a supporting production for all template definitions in this document, where `ObjectIdentifierValue` refers to the corresponding notation defined in CCITT Rec. X.208 | ISO/IEC 8824;

g) strings surrounded by

```
[ ]
```

delimit parts of a template definition that may either be present or absent in each instance of use of the template. If the closing brace is followed by an asterisk, i.e.,

```
[ ]*
```

the contents of the braces may appear zero or more times. The circumstances under which these parts of the definition may be omitted or repeated are dependent upon the definition of the template type;

h) strings surrounded by

```
<>
```

delimit strings that shall be replaced in each instance of use of a template. The structure and meaning of the replacement string is dependent upon the string type;

i) upper case strings denote Keywords which are required to be present in each instance of use of a template, unless they are enclosed in

```
[ ]
```

to indicate that their presence is optional;

j) the character

```
¦
```

is used as a delimiter for alternative strings in the `syntactic-definition` of a `supporting production`. Where a `supporting production` is used to define alternative strings, the opening delimiter of the first alternative is the symbol `->`, the character | is the closing and opening delimiter for subsequent alternatives, and the closing delimiter of the final alternative is the first end of line encountered following its opening delimiter;

k) a `template-label` shall be unique within the Recommendation | International Standard or document in which it is declared. In a multi-part Recommendation | International Standard or document where the individual parts are normally maintained and distributed separately, a `template-label` shall be unique within the part in which it is declared.

The requirement for uniqueness of `template-label`s is independent of the type of the template being labelled. For example, if the label `label1` is used in a document to label an instance of use of a template, it is not permitted to use the label `label1` in the same document to label another instance of use of a template of the same type or of a different type.

Where a `template-label` is declared in document A and referenced in document B, the reference in document B shall be prefixed by the globally unique name of document A. In the case of documents named by an internationally recognised naming authority, such as CCITT or ISO/IEC, the registered alphanumeric designator of the document shall be used as the identifier, such as `"CCITT Rec. X.722 (1992) | ISO/IEC 10165-4 : 1992"`. The format of this string shall be as specified by the naming authority concerned for references to its documents. Where the document referenced is jointly developed and published by CCITT and ISO/IEC, the designator of the document shall include both the CCITT and the ISO/IEC document designators separated by a "|", as shown in this example. Where a globally unique name is not already available, it is permissible to assign the value of an object identifier to the referenced document, and use this value as a globally unique document name. The syntax of a `template-label`, defined using the above notation, is as follows

```
[document-identifier :] <label-string>
document-identifier -> "<standard-name>" | object-identifier
```

A `label-string` may include any number of the following characters

1) upper or lower case alphabetic characters;

2) the digits `0-9`;

3) the character

```
-
```

4) the character

```
    /
```

in any order, commencing with a lower case alphabetic character, with the exception that the character pair

```
--
```

shall not appear in a label-string. For example, the following `template-label`

```
"CCITT Rec. X.722 (1992) | ISO/IEC 10165-4 : 1992":exampleObjectClass
```

constitutes a globally unique label for the definition of `exampleObjectClass` contained in Annex A.

Label references that are not prefixed by a `document-identifier` are assumed to be references to labels declared in the document in which the reference appears.

l) wherever a `template-label` is present in a template as a pointer to another template, it may be replaced by the entire text of the referenced template itself (including the template's label). This allows a template to include the templates it references (sub-templates) "in-line" while maintaining the ability for other templates to refer to the sub-templates so defined. In effect, the `supporting production`

```
template-label -> template-definition
```

is assumed for all instances of `template-label` and `template-definition`.

m) wherever it is necessary to refer from a template to an ASN.1 type or value definition, the name of the ASN.1 type or value is prefixed by the module name of the ASN.1 module which contains the type or value definition. The module name used is assumed to refer to an ASN.1 module that is contained in the same document that contains the template from which the type or value reference is made. The supporting productions

```
type-reference     -> <module-name>.<type-name>

value-reference    -> <module-name>.<value-name>
```

are therefore assumed for all template definitions that refer to ASN.1 types or values, where `module-name` is the name assigned to an ASN.1 module contained in the document that makes the reference, and `type-name` and `value-name` are names assigned to ASN.1 type or value definitions contained in that module. Where it is necessary to refer to type or value definitions contained in other documents, this may be achieved by means of a local ASN.1 module which makes use of the ASN.1 `IMPORTS` mechanism to import the appropriate type or value definitions.

n) wherever it is necessary to include text in a template, the text is included in the form of a string of characters, optionally starting and ending with a `text-delimiter` character chosen from the following characters

```
! " # $ % ^ & * ′ ‘ ~ ? @ \
```

If a `text-delimiter` character is used, the same character shall be used at the start and end of the string, and wherever that `text-delimiter` character appears in the body of the text string, it shall be replaced by two occurrences of that character. If a `text-delimiter` character is not used, then the text string shall not contain any punctuation character that is a valid successor to the text string in the template definition in which the text string is used.

The supporting productions

```
delimited-string ->      text-delimiter <text-string> text-delimiter |
                         <text-string>
text-delimiter ->  ! | " | # | $ | % | ^ | & | * | ′ | ‘ | ~ | ? | @ | \
```

are therefore assumed for all templates that permit the use of a `delimited-string`, where `text-string` is an arbitrary sequence of characters, and if the `text-delimiter` has been used, all occurrences of that delimiter character in `text-string` have been replaced by a pair of `text-delimiter` characters.

With the exception of the rules pertaining to the use of delimiters, the internal structure of a `text-string`, and in particular, the use of the comment structure defined in this Recommendation | International Standard, has no significance with respect to the provisions of this Recommendation | International Standard. The meaning of the entire `text-string` must therefore be assumed to carry normative significance, unless explicitly stated otherwise in the document in which the notation is used.

8.3 Managed object class template

8.3.1 Overview

The Managed Object Class template forms the basis of the formal definition of a managed object. Elements in the template allow the class to be placed at the appropriate node of the inheritance tree, the various characteristics of the class to be specified, and the behaviour of the class to be defined. The major elements of the definition are defined below.

8.3.1.1 Inheritance

Each managed object class defines the superclass(es) from which it has been derived. Characteristics of the superclass(es) are inherited by the subclass; the subclass definition may add to these characteristics (specialization) but may not remove any characteristics of the superclass. Ultimately, all classes are subclasses of top.

8.3.1.2 Mandatory packages

The managed object class definition includes the packages of behaviour, attributes, operations and notifications that provide a complete specification of the behaviour that characterizes all instances of the class.

8.3.1.3 Conditional packages

The managed object class definition includes the specification of packages of behaviour, attributes, operations and notifications that are present in instances of that class as a consequence of a specified condition.

8.3.1.4 Class naming

The managed object class definition shall include a class name which may be used to refer to the class in management protocol. This is achieved by registration of an object identifier value which identifies to the managed object class definition.

8.3.2 Template structure

```
<class-label> MANAGED OBJECT CLASS
           [DERIVED FROM          <class-label>       [,<class-label>]* ;
           ]
           [CHARACTERIZED BY      <package-label>     [,<package-label>]* ;
           ]
           [CONDITIONAL PACKAGES  <package-label>     PRESENT IF condition-definition
                                  [,<package-label>   PRESENT IF condition-definition]* ;
           ]
REGISTERED AS object-identifier ;

supporting productions

condition-definition -> delimited-string
```

8.3.3 Supporting definitions

8.3.3.1 `DERIVED FROM <class-label> [,<class-label>]*`

The `DERIVED FROM` construct shall be present in all managed object class definitions other than top. It is therefore the case that top is a superclass of all managed object classes other than itself.

The `class-label` identifies a managed object class from which the managed object class has been derived; i.e., a managed object class which is one of the managed object class's immediate superclasses. As multiple inheritance is permitted, a managed object class may have more than one immediate superclass.

The process of inheritance (specialization) requires all the characteristics of the superclass(es) to be included in the definition of the subclass.

Characteristics that are inherited from a superclass shall not be repeated in the documentation of the subclass unless one of the techniques described in this Recommendation | International Standard for extending or modifying a definition inherited from a superclass is being used. The `DERIVED FROM` construct is therefore presumed to automatically import all characteristics from the superclass definition(s). These characteristics may be augmented or modified by elements defined within the `CHARACTERIZED BY` and `CONDITIONAL PACKAGES` constructs.

NOTE 1 – A list of all managed object classes that the managed object class definition inherits characteristics from should be included, as a comment, in the documentation of the managed object class definition.

Where multiple inheritance results in the same element definition being multiply imported (as could happen, for example, if two superclass definitions include the same attribute), the subclass is assumed to contain a single copy of the definition concerned.

From the point of view of resolving any conflicts that may exist between the elements defined in the packages and conditional packages inherited or included in the managed object class definition by specialization, all packages that are to be included in a particular managed object class are treated identically. Each package defines a number of elements, which are treated as follows

a) `BEHAVIOUR`. Packages included in a subclass extend the inherited behaviour. The behaviour of a managed object class shall be expressed in a way that takes account of the possible presence or absence of conditional packages;

   NOTE 2 – In some circumstances, subclasses may be defined where no additional behaviour definitions are required over and above those that are inherited from the superclass(es).

b) `ATTRIBUTES`. Attributes may be specified in the packages included in the subclass definition. Where the `ATTRIBUTES` construct of a package identifies an attribute that is multiply defined in the managed object class, the following rules apply:

   1) a single attribute of that type shall be included in the instantiated managed object;

   2) the resultant `propertylist` is the logical OR of the `propertylist` included in the subclass and the inherited `propertylist`(s), with the exception of `PERMITTED VALUES`, where the instantiated permitted value set is the set intersection of all permitted value specifications for that attribute type, and `REQUIRED VALUES`, where the instantiated required value set is the intersection of the instantiated permitted value set with the set union of all required value specifications for that attribute type. For `DEFAULT VALUE` or `INITIAL VALUE`, if conflicting values are specified for the attribute in the collected definitions, then a package included in the subclass shall resolve the conflict;

   3) the parameters associated with a given attribute are the set union of all parameters associated with the Attribute template and all parameters associated with the attribute in any package that is instantiated.

   If the managed object class is intended to be instantiated, there shall be at least one attribute defined as part of the class definition, as it is necessary to identify an attribute that can be used to name instances of a managed object;

   NOTE 3 – Attributes used for naming may be chosen from any attributes that are part of the class definition. This includes all attributes that have been inherited from superclasses and all attributes that have been added to the class by the process of specialization.

c) `ATTRIBUTE GROUPS`. For an extensible attribute group, the membership of an attribute group in an instance of the subclass is the set union of all attributes defined in the Attribute Group template, plus all attributes added to that group in the superclass(es) or the subclass;

d) `ACTIONS`. Actions may be included in the subclass definition; these may be additional actions to those inherited from superclasses, or may include additional parameters for an inherited action. The parameters associated with a given action are the set union of all parameters associated with the Action template and all parameters associated with the action in any package that is instantiated;

e) `NOTIFICATIONS`. Notifications may be included in the subclass definition; these may be additional notifications to those inherited from superclasses, or may include additional parameters for an inherited notification. The parameters associated with a given notification are the set union of all parameters associated with the Notification template, and all parameters associated with the notification in any package that is instantiated.

If a package is included in a managed object class definition more than once, by inheritance and/or multiple inclusion in the Managed Object Class template, the resultant `condition-definition` associated with the package is the logical OR of all the `condition-definitions` in the collected set of definitions. For this purpose, packages included in CHARACTERIZED BY constructs (mandatory packages) are treated as if they had been included in a CONDITIONAL PACKAGES construct with a condition definition of `PRESENT IF !TRUE!`.

The characteristics within a package (mandatory or conditional) shall only depend on the characteristics of other conditional packages if the associated package conditions ensure that the required characteristics will be present in all managed objects where the first package is present.

8.3.3.2 `CHARACTERIZED BY <package-label> [,<package-label>]*`

This construct, if present, allows one or more mandatory packages of behaviour, attributes, operations and notifications to be included in the managed object class definition, in addition to those that are part of the definition as a result of the `DERIVED FROM` construct. The `package-label` identifies the package definition that is to be included. Specifying the label of a package that is also included in the managed object class definition as a conditional package makes that package mandatory in this managed object class and its subclasses.

8.3.3.3 `CONDITIONAL PACKAGES <package-label> PRESENT IF condition-definition [,<package-label> PRESENT IF condition-definition]*`

Present if one or more conditional packages are to be included in the class. The `package-label` identifies the package definition that is applicable. The `condition-definition` is a description of the condition which, if true, requires that the package be included in an instance of the class. No restriction is placed on the structure or character set used to represent the `condition-definition`. The condition shall meet the requirements for conditional packages presented in CCITT Rec. X.720 | ISO/IEC 10165-1. For example

```
CONDITIONAL PACKAGES class-4-attributes PRESENT IF
   'the corresponding protocol entity supports Class 4 operation as  specified
   in ISO/IEC XXXX' ;
```

would constitute a valid package declaration, provided that standard ISO/IEC XXXX defined Class 4 operation as a valid optional feature of the resource.

8.3.3.4 `REGISTERED AS object-identifier`

The `object-identifier` value provides a globally unique identifier for the object class definition. This value is used in management protocol when it is necessary to identify the object class.

8.4 Package template

8.4.1 Overview

This template allows a package consisting of a combination of behaviour definitions, attributes, attribute groups, operations, notifications and parameters to be defined for subsequent insertion into a Managed Object Class template under the `CHARACTERIZED BY` or `CONDITIONAL PACKAGES` constructs. The major elements of the definition are described below.

8.4.1.1 Behaviour

The package definition provides a complete specification of the behaviour that is included in the package. This includes

- the effect of the Operations upon a managed object, and the circumstances under which Notifications are generated;
- any constraints that are placed on operations in order to satisfy consistency rules, and in particular, the rules under which Creation and Deletion of managed objects may be performed and the consequences of these operations;
- a specification of how instances of a managed object class interact with other, related, managed objects of the same or of different class(es);
- identification of any attributes that correspond with information in notifications, if any. This includes identifying any mappings to particular notification fields that takes place or to the emission of a notification;
- specification of IVMO selection criteria, if any;
- a complete definition of any other aspects of the behaviour of the managed object class.

8.4.1.2 Contained attributes

The set of attributes that the package contains shall be specified.

8.4.1.3 Operations and Notifications

The package definition shall specify which notifications instances of the class that make use of this package shall be able to generate, which operations instances of the class shall be capable of performing, and in the case of attribute-related operations, which attributes shall be available to be operated upon. The package definition shall also specify any additional parameters that notifications and operations of instances of the managed object class that make use of this package shall be capable of carrying.

NOTES

1 The operations identified in the class definition are the operation types defined in CCITT Rec. X.720 | ISO/IEC 10165-1 (Get attribute value, Replace attribute value, Replace with default value, etc.). In the case of Actions and Notifications, further definitions are required in order to characterize their functionality, as described in 8.10 and 8.11. The Create and Delete operations are specified as part of the Name Binding template described in 8.6, as creation and deletion of a managed object is closely bound to the containment relationship between superior and subordinate objects, rather than to all instances of a managed object class.

2 Late binding, i.e. the allocation of additional parameters to actions or notifications of a managed object class, can be achieved by including in the managed object class a package that contains (only) the affected actions or notifications and their new parameters. The set union rule given in 8.3.3 for parameters of actions and notifications, means that the additional parameter(s) will be associated with the notification or action if the package is instantiated.

8.4.2 Template structure

```
<package-label>   PACKAGE
       [BEHAVIOUR         <behaviour-definition-label> [,<behaviour-definition-label>]* ;
       ]
       [ATTRIBUTES        <attribute-label> propertylist [<parameter-label>]*
                          [,<attribute-label> propertylist [<parameter-label>]*]* ;
       ]
       [ATTRIBUTE GROUPS  <group-label> [<attribute-label>]* [,<group-label>
                          [<attribute-label>]*]* ;
       ]
       [ACTIONS           <action-label> [<parameter-label>]* [,<action-label>
                          [<parameter-label>]*]* ;
       ]
       [NOTIFICATIONS <notification-label>   [<parameter-label>]*   [,<notification-label>
       [<parameter-label>]*]* ;
       ]
[REGISTERED AS object-identifier] ;

supporting productions
propertylist      ->  [REPLACE-WITH-DEFAULT]
                      [DEFAULT VALUE            value-specifier]
                      [INITIAL VALUE            value-specifier]
                      [PERMITTED VALUES         type-reference]
                      [REQUIRED VALUES          type-reference]
                      [get-replace]
                      [add-remove]
value-specifier   ->  value-reference | DERIVATION RULE <behaviour-definition-label>
get-replace->     GET | REPLACE | GET-REPLACE
add-remove        ->  ADD | REMOVE  | ADD-REMOVE
```

8.4.3 Supporting definitions

8.4.3.1 BEHAVIOUR <behaviour-definition-label> [,<behaviour-definition-label>]*

The BEHAVIOUR construct allows the behaviour (semantics) associated with the package to be completely described. This construct relates the external view of aspects of a managed object (its operations and notifications) to its internal operation. The behaviour-definition-label identifies an instance of use of the Behaviour template. In some circumstances, packages may be defined where no behaviour definitions are required.

8.4.3.2 ATTRIBUTES <attribute-label> propertylist [<parameter-label>]* [,<attribute-label> propertylist [<parameter-label>]*]*

This construct allows attributes to be included in the package definition. The propertylist that follows each attribute-label defines the set of operations that may be performed on the managed object with reference to the attribute, and defines any default, initial, permitted or required value(s) associated with the attribute.

The REPLACE-WITH-DEFAULT property is included if the attribute has a default value that may be set by means of the Replace with default value operation.

The DEFAULT VALUE property is included if the attribute has a default value, to be used to provide the value of the attribute in a Replace with default value operation, or to specify a default value that may be used for the attribute when the package is instantiated according to the rules defined in CCITT Rec. X.720 | ISO/IEC 10165-1. If a default value is unspecified and the REPLACE-WITH-DEFAULT property is present, the default value is determined by other means local to the managed system. The value may be specified either by means of a value-reference or by means of a DERIVATION RULE which specifies how the default value shall be determined.

The INITIAL VALUE property is included if the attribute has a mandatory initial value, to be used at create time to provide the initial value of the attribute. The value may be specified either by means of a value-reference or by means of a DERIVATION RULE which specifies how the initial value shall be determined.

If the `PERMITTED VALUES` property is present, the `type-reference` specifies any restrictions on the possible values that the attribute may take. The form of the specification referenced shall be a subtype of the attribute syntax type, defined using the ASN.1 subtype notation.

NOTE 1 – The `PERMITTED VALUES` construct is required only in attribute definitions where it is necessary to specify a restriction on the value set permitted by the specification of the attribute syntax, e.g., when modifying an existing attribute specification. Such restrictions on the value set of an attribute should only be made in circumstances where the restriction is based on a limitation inherent in the semantics of the attribute, rather than a restriction that is based on some arbitrary assumption as to what might constitute a reasonable value set.

If the `REQUIRED VALUES` property is present, the `type-reference` specifies any values that the attribute shall be capable of taking. The form of the specification referenced shall be a subtype of the attribute syntax type, defined using the ASN.1 subtype notation.

NOTE 2 – This property defines the value set required for conformance. For example, a modem managed object might have a data rate attribute with a permissible value set of 0 to 19,2K; however, conformance to the modem standard might require support for one particular data rate from the value set permitted. As with `PERMITTED VALUES`, such requirements on the supported value set of an attribute should only be made in circumstances where the requirement is based on a feature inherent in the semantics of the attribute, rather than a requirement that is based on some arbitrary assumption as to what might constitute a reasonable minimum value set.

If present, the `parameter-labels` identify managed object class specific error parameters associated with management operations on the attribute. These are reported as processing failures. The syntax of the error parameters are defined in the referenced templates.

8.4.3.3 `ATTRIBUTE GROUPS <group-label> [<attribute-label>]* [,<group-label> [<attribute-label>]*]*`

This construct allows a set of attribute groups to be identified as part of the package. In the case of extensible attribute groups, the original definition of an attribute group may be extended by the addition of further `attribute-label`s.

8.4.3.4 `ACTIONS <action-label> [<parameter-label>]* [,<action-label> [<parameter-label>]*]*`

If present, the `action-label`s identify the set of action definitions that are included in the package. The behaviour definitions shall specify the effect of these actions upon managed objects.

If present, the `parameter-label`s identify any managed object class specific action information or reply parameters, or any managed object class specific error parameters associated with the action. The syntax of the parameters are defined in the referenced templates.

8.4.3.5 `NOTIFICATIONS <notification-label> [<parameter-label>]* [,<notification-label> [<parameter-label>]*]*`

Present if any notifications are included in the package. The `notification-label`s identify the notification definitions that are applicable. The behaviour definitions shall specify the circumstances under which these notifications are generated by a managed object.

If present, the `parameter-label`s identify any managed object class specific notification information or reply parameters, or any managed object class specific error parameters associated with the notification. These may be additional parameters used, for example, to populate the Additional information field of notifications defined in CCITT Rec. X.733 | ISO/IEC 10164-4. The syntax of the parameters are defined in the referenced templates.

8.4.3.6 `REGISTERED AS object-identifier`

The `object-identifier` value, if present, provides a globally unique identifier for the package definition, and registers the grouping of behaviour, attributes, attribute groups, actions and notifications that the package defines. The `object-identifier` value is the value that is included in the Packages attribute of any instances of the managed object class that are created, in accordance with the rules stated in CCITT Rec. X.720 | ISO/IEC 10165-1. This construct is required in cases where the package is referenced by a `CONDITIONAL PACKAGES` construct in a Managed Object Class template.

8.5 Parameter template

8.5.1 Overview

This template permits the specification and registration of parameter syntaxes and associated behaviour that may be associated with particular attributes, operations and notifications within the Package, Attribute, Action and Notification templates defined in 8.4, 8.7, 8.10 and 8.11. The type specified in a Parameter template is used to fill in an `ANY DEFINED BY x` construct in a management PDU, where `x` is a field in the PDU that carries the object identifier assigned to the parameter. This mechanism is, for example, applicable to the definition of

- processing failures;
- parameters of action requests/responses;
- parameters of notification requests/responses.

Use of the template in each of these contexts is described in 8.5.3.

The major elements of the definition are described below.

8.5.1.1 Context definition

The template specifies the context in which the parameter is applicable, that is, it specifies that the parameter is carried in a particular field in a management PDU.

8.5.1.1.1 Action information/reply, event information/reply, specific error

When the context is unambiguously identified by the management PDU in which the parameter is carried, the context may be indicated by one of the five predefined keywords specified in 9.5.3. The context is unambiguously identified by the management PDU if and only if the `ANY DEFINED BY` construct appears in that PDU exactly once.

8.5.1.1.2 Context keyword

When the context is not unambiguously identified by the management PDU in which the parameter is carried, a context keyword shall be specified. The context keyword shall identify the field in the management PDU in which the parameter may be carried.

8.5.1.1.3 Usage in other templates

Table 16 shows where the Parameter template is referenced, with the allowed options.

When used as a qualifier in a package definition, the parameter may be "late bound" to the element that it qualifies, e.g., further parameters may be added to a previously defined notification at the time of package definition if the notification syntax is extensible.

8.5.1.2 Syntactic definition

The template permits an abstract syntax to be associated with the parameter.

8.5.1.3 Attribute reference

In place of an explicit syntactic definition and registration in the Parameter template, the template may specify these two items by referring to an Attribute template. The use of this construct has no effect on the meaning of the existing registered attribute.

8.5.1.4 Behaviour

The template defines any behaviour that applies to the use of the parameter.

Table 16 – Use of the Parameter template

Use	Possible contexts
ATTRIBUTES construct of Package template	SPECIFIC-ERROR
ACTIONS construct of Package template	context-keyword, SPECIFIC-ERROR, ACTION-INFO, ACTION-REPLY
NOTIFICATIONS construct of Package template	context-keyword, SPECIFIC-ERROR, EVENT-INFO, EVENT-REPLY
CREATE construct of Name Binding template	SPECIFIC-ERROR
DELETE construct of Name Binding template	SPECIFIC-ERROR
Attribute template	SPECIFIC-ERROR
Action template	context-keyword, SPECIFIC-ERROR, ACTION-INFO, ACTION-REPLY
Notification template	context-keyword, SPECIFIC-ERROR, EVENT-INFO, EVENT-REPLY

8.5.2 Template structure

```
<parameter-label> PARAMETER

       CONTEXT                  context-type ;

       syntax-or-attribute-choice ;

       [BEHAVIOUR               <behaviour-definition-label>
                                [,<behaviour-definition-label>]* ;

       ]

[REGISTERED AS object-identifier]  ;

supporting productions
context-type                  ->  context-keyword |
                                  ACTION-INFO |
                                  ACTION-REPLY |
                                  EVENT-INFO |
                                  EVENT-REPLY |
                                  SPECIFIC-ERROR
context-keyword               ->  type-reference.<identifier>
syntax-or-attribute-choice    ->  WITH SYNTAX   type-reference |
                                  ATTRIBUTE     <attribute-label>
```

8.5.3 Supporting definitions

8.5.3.1 `CONTEXT context-type`

This construct defines the context in which the parameter is applicable. The options are

- `context-keyword`: this option is a reference to a context defined externally to the template. The structure of the reference consists of a `type-reference` followed by an `identifier` which is the name of a field in the management PDU specified by the `type-reference`. It may therefore be used as a reference to a context defined in another document. This may be used to indicate, for example, that the parameter applies only to a specific field of the CMIS Event information parameter (e.g., CCITT Rec. X.733 | ISO/IEC 10164-4 Additional information field), or the CMIS Action reply parameter. If the parameter does not map to a specific field-name (e.g., Event information is defined to be a set of Parameter Identifier/Parameter Value pairs), one of the following, more general contexts can be specified;
- `ACTION-INFO`: this option defines the parameter as being applicable to the representation of a parameter that may be carried in the CMIS Action information parameter;
- `ACTION-REPLY`: this option defines the parameter as being applicable to the representation of a parameter that may be carried in a CMIS Action reply parameter;
- `EVENT-INFO`: this option defines the parameter as being applicable to the representation of a parameter that may be carried in the CMIS Event information parameter;
- `EVENT-REPLY`: this option defines the parameter as being applicable to the representation of a parameter that may be carried in the CMIS Event reply parameter;
- `SPECIFIC-ERROR`: this option defines the parameter as being applicable to the representation or generation of a CMIS processing failure error. When this option is used with parameters that apply to attributes, the managed object class definition shall specify whether, when this error occurs for a Replace attribute value or Replace with default value operation on one attribute, other attributes referenced in a single replace request are modified or not.

8.5.3.2 `WITH SYNTAX type-reference`

This construct, if present, identifies the ASN.1 type of the parameter, as carried in protocol.

8.5.3.3 `ATTRIBUTE <attribute-label>`

If present, this construct identifies an Attribute template whose syntax and object identifier are used as the syntax and object identifier of the parameter, respectively.

8.5.3.4 `BEHAVIOUR <behaviour-definition-label> [,<behaviour-definition-label>]*`

If present, this construct permits the specification of any behaviour or semantics associated with the parameter. If the `ATTRIBUTE` construct is used, this construct does not modify the behaviour of the attribute.

8.5.3.5 `REGISTERED AS object-identifier`

The `object-identifier` value, if present, provides a globally unique identifier for the parameter definition. This value is used in management protocol when it is necessary to identify the parameter. This construct shall be present if and only if the `WITH SYNTAX` construct is present.

8.6 Name binding template

8.6.1 Overview

This template allows alternative naming structures to be defined for managed objects of a given managed object class by means of name bindings. A name binding allows an attribute to be selected as the naming attribute that shall be used when a subordinate object which is an instance of a specified managed object class is named by a superior object which is an instance of a specified managed object class or other object class, such as a Directory object class.

If a given name binding is used, the attribute identified as the naming attribute shall be present in the subordinate object. The naming attribute is used to construct the relative distinguished name (RDN) of subordinate objects of that class. An RDN is constructed from the object identifier assigned to that attribute type and the value of the instance of the attribute. The Distinguished Name of the subordinate object is obtained by appending its RDN to the Distinguished Name of its superior object.

Name bindings are not considered to be part of the definition of either of the classes that they reference. A given subordinate managed object class may have more than one name binding associated with it. The set of name bindings defines the set of possible naming relationships with superior objects and the set of managed object classes from which subordinate objects may be instantiated.

A name binding may also be defined to apply to all subclasses of the specified superior object class or all subclasses of the specified subordinate object class, or both.

NOTE – Name bindings may be specified for a managed object class subsequent to the specification of the managed object class itself.

8.6.2 Template structure

```
<name-binding-label>      NAME BINDING

       SUBORDINATE OBJECT CLASS          <class-label> [AND SUBCLASSES];
       NAMED BY SUPERIOR OBJECT CLASS    <class-label> [AND SUBCLASSES];
       WITH ATTRIBUTE                    <attribute-label> ;
       [BEHAVIOUR                        <behaviour-definition-label>
                                         [,<behaviour-definition-label>]* ;
       ]
       [CREATE                           [create-modifier [,create-modifier]]
                                         [<parameter-label>]* ;
       ]
       [DELETE                           [delete-modifier] [<parameter-label>]* ;
       ]

REGISTERED AS object-identifier ;

supporting productions

create-modifier   ->      WITH-REFERENCE-OBJECT |
                          WITH-AUTOMATIC-INSTANCE-NAMING
delete-modifier   ->      ONLY-IF-NO-CONTAINED-OBJECTS |
                          DELETES-CONTAINED-OBJECTS
```

8.6.3 Supporting definitions

8.6.3.1 `SUBORDINATE OBJECT CLASS <class-label> [AND SUBCLASSES]`

This defines a managed object class whose instances may be named by instances of the object class defined by the `NAMED BY SUPERIOR OBJECT CLASS` construct. The name of an instance of this subordinate object class is constructed by concatenating the distinguished name of its superior object with the relative distinguished name of the subordinate object. If `AND SUBCLASSES` is specified, the name binding also applies to all subclasses of the specified managed object class.

8.6.3.2 `NAMED BY SUPERIOR OBJECT CLASS <class-label> [AND SUBCLASSES]`

This defines a managed object class or other object class, such as a Directory object class, whose instances may name instances of the managed object class defined by the `SUBORDINATE OBJECT CLASS` construct. If `AND SUBCLASSES` is specified, the name binding also applies to all subclasses of the specified object class.

8.6.3.3 `WITH ATTRIBUTE <attribute-label>`

This defines the attribute that shall be used, in the context of this name binding, to construct the relative distinguished name for instances of the managed object class defined by the `SUBORDINATE OBJECT CLASS` construct. Values of this attribute shall be represented by single-valued data types complying with the restrictions specified in CCITT Rec. X.720 | ISO/IEC 10165-1; if no suitable attribute is available for use as a naming attribute, managed object designers are encouraged to provide a naming attribute of type `GraphicString`.

8.6.3.4 `BEHAVIOUR <behaviour-definition-label> [,<behaviour-definition-label>]*`

If present, this construct permits any behavioural impact imposed as a consequence of the name binding to be defined. The `behaviour-definition-label` identifies the behaviour definition concerned.

NOTE 1 – This construct is intended for use as a means of describing behaviour specific to a name binding. Any behaviour that is applicable to all possible instances of a managed object class should be defined as part of the behaviour referenced by the Package template(s) that define the managed object class.

8.6.3.5 `CREATE [<create-modifier> [,<create-modifier>]] [<parameter-label>]*`

Present if it is permitted to create new instances of the managed object class referenced by the `SUBORDINATE OBJECT CLASS` construct in the context of this name binding, by means of systems management operation. The `create-modifier` values specify the options available on creation. The permitted `create-modifier` values are as follows:

- `WITH-REFERENCE-OBJECT`. If present, a reference managed object may be specified on creation as a source of default values and to specify choice of conditional packages;

- `WITH-AUTOMATIC-INSTANCE-NAMING`. If present, the Create request may omit to specify the instance name of the new managed object.

The behaviour definitions shall specify what course of action is taken when there is a choice of Name Bindings that may be applied to the new managed object.

The sources of initial attribute values used at managed object creation time, and their associated precedence rules, are defined in CCITT Rec. X.720 | ISO/IEC 10165-1.

If present, the `parameter-label`s identify name binding specific error parameters associated with the Create operation. These are reported as processing failures. The syntax of the error parameters are defined in the referenced templates.

8.6.3.6 `DELETE [<delete-modifier>] [<parameter-label>]*`

Present if it is permitted to delete instances of the managed object class referenced by the `SUBORDINATE OBJECT CLASS` construct in the context of this name binding. The `delete-modifier`, if present, indicates the behaviour of a managed object of that class if the managed object is deleted. The permitted `delete-modifiers` are as follows

- `ONLY-IF-NO-CONTAINED-OBJECTS`. If specified, any contained managed objects shall be explicitly deleted by management operations prior to deletion of the containing managed object, i.e., a Delete request will cause an error if there are contained managed objects;
- `DELETES-CONTAINED-OBJECTS`. If a Delete request is applied to a managed object for which the DELETES-CONTAINED-OBJECTS modifier is specified, the Delete request will fail if any directly or indirectly contained managed object has the ONLY-IF-NO-CONTAINED-OBJECTS modifier specified and also has a contained managed object; otherwise, a successful Delete request will also have the effect of deleting contained managed objects.

Other rules which describe the behaviour with respect to deletion of contained managed objects may be specified in the `BEHAVIOUR` construct.

NOTE 2 – Given that the `DELETES-CONTAINED-OBJECTS` modifier permits the deletion of a managed object regardless of whether it contains other managed objects, it is advisable to use the `ONLY-IF-NO-CONTAINED-OBJECTS` modifier if there is any doubt as to which modifier is appropriate.

If there are constraints on deletion relative to other relationships or conditions that are generic to the managed object class, these shall be specified as part of the behaviour of the managed object class.

If present, the `parameter-label`s identify name binding specific error parameters associated with the Delete operation. These are reported as processing failures. The syntax of the error parameters are defined in the referenced templates.

8.6.3.7 `REGISTERED AS object-identifier`

The `object-identifier` value provides a globally unique identifier for the name binding definition. This value is used to identify the name binding for the purposes of knowledge management.

8.7 Attribute template

8.7.1 Overview

This template is used to define individual attribute types. These definitions may be further combined by the attribute group template where attribute groups are required. The major elements of the definition are described below.

8.7.1.1 Derivation

The definition of an attribute type may modify or constrain the definition of another attribute type.

8.7.1.2 Attribute syntax

The definition of an attribute type shall include the definition of the syntax that shall be used to convey values of the attribute in management protocol. This definition is achieved by means of a reference to an ASN.1 Type Definition. The definition of an attribute syntax indicates whether the attribute value is a single- or set-valued attribute type. If the base type is SET OF, the attribute is a set-valued type, otherwise it is a single-valued type.

8.7.1.3 Value matching

The definition of an attribute type may include the valid ways in which the value of an instance of the type may be tested, i.e., whether the attribute may be tested for equality, magnitude, etc.. Value matching on some attribute types may require the specification of how a matching rule is defined to operate, as part of the attribute’s behaviour definition. The absence of any matching rules in the attribute definition implies that matching of values is undefined.

8.7.1.4 Behaviour

The attribute definition may include definition of attribute-specific behaviour; i.e., behaviour that applies to an attribute type regardless of which managed object class contains instances of the attribute type.

8.7.1.5 Attribute identifier

An object identifier value shall be allocated to each attribute that is to be included in the definition of a managed object class. This value is used in management protocol to identify the attribute.

8.7.1.6 Parameters

The attribute definition may identify attribute-specific error parameters associated with management operations on the attribute type. The referenced templates map these to processing failures.

8.7.2 Template Structure

```
<attribute-label> ATTRIBUTE
           derived-or-with-syntax-choice ;
           [MATCHES FOR    qualifier [, qualifier]* ;
           ]
           [BEHAVIOUR      <behaviour-definition-label> [,<behaviour-definition-label>]* ;
           ]
           [PARAMETERS     <parameter-label> [,<parameter-label>]* ;
           ]
[REGISTERED AS object-identifier] ;
```

```
supporting productions

qualifier                         ->     EQUALITY | ORDERING | SUBSTRINGS |
                                         SET-COMPARISON | SET-INTERSECTION
derived-or-with-syntax-choice     ->     DERIVED FROM <attribute-label> |
                                         WITH ATTRIBUTE SYNTAX type-reference
```

8.7.3 Supporting definitions

8.7.3.1 `DERIVED FROM <attribute-label>`

If this element is present, the attribute definition takes as a starting point all aspects of the definition referenced by `attribute-label`, including any that it may in turn have derived from other attribute definitions. The rules for interpreting the effect of presence of any of the other elements of the Attribute template under these circumstances are as follows:

- `MATCHES FOR`: The resultant set of matching rules shall be the logical OR of the matching rules specified by this construct with any derived matching rules;
- `BEHAVIOUR` is assumed to extend any derived behaviour definitions;
- `REGISTERED AS` is assumed to replace any registration derived from other definitions.

This derivation mechanism permits

- the definition of an attribute based on an existing attribute definition;
- the addition of further constraints to an existing attribute definition.

8.7.3.2 `WITH ATTRIBUTE SYNTAX type-reference`

This construct, present only if the `DERIVED FROM` construct is absent, identifies the ASN.1 data type that describes how instances of the attribute value are carried in protocol.

The ASN.1 data type also defines the data type of the attribute itself. If the base type of the syntax is the set-of type, the attribute is a set-valued attribute. All other ASN.1 data types, including the set type, sequence type and sequence-of type, define single-valued attribute types.

8.7.3.3 `MATCHES FOR qualifier [, qualifier]*`

This construct defines the types of test that may be applied to a value of the attribute as part of a Filter operation. Matching for the presence of an attribute is implicitly permitted for all attributes. For other types of matching, if this construct is not present, matching is undefined and is therefore not permitted on the attribute. The options are

- `EQUALITY`. If present, the attribute value may be tested for equality against a given value;
- `ORDERING`. If present, the attribute value may be tested against a given value in order to determine which has the greater value;

- SUBSTRINGS. If present, the attribute value may be tested against a given substring value in order to determine its presence or absence in the attribute value;
- SET-COMPARISON. If present, the attribute value may be tested against a given value in order to determine subset/superset relationships between the values;
- SET-INTERSECTION. If present, the attribute value may be tested against a given value in order to determine the presence or absence of a non-null set intersection between the two values.

8.7.3.4 BEHAVIOUR <behaviour-definition-label> [,<behaviour-definition-label>]*

Any behaviour that is generic to this attribute type may be defined by means of this behaviour construct. The behaviour definition shall include any additional specification that is required in order to define how the chosen set of matching rules are applied to the attribute definition. Behaviour that is specific to the managed object class is defined in the behaviour construct of the Package template.

8.7.3.5 PARAMETERS <parameter-label> [,<parameter-label>]*

The parameter-labels allow parameters to be associated with the behaviour of an attribute type, for the definition of processing failures. For example, an attribute type might exhibit a "constraint violation" error under certain conditions. A parameter giving information about such an error could be specified using a Parameter template with CONTEXT SPECIFIC-ERROR, and referenced from the Attribute template.

8.7.3.6 REGISTERED AS object-identifier

The object-identifier value provides a globally unique identifier for the attribute definition; this includes all elements referenced either directly or indirectly by the DERIVED FROM, WITH ATTRIBUTE SYNTAX, MATCHES FOR and BEHAVIOUR constructs, where present. This value is used in management protocol when it is necessary to identify the attribute type. If this construct is omitted, the attribute definition shall not be referenced in a managed object class definition. Where an attribute definition is derived from an existing attribute definition that includes a REGISTERED AS construct, the object-identifier value assigned to the existing definition is not a valid identifier for the derived definition. The REGISTERED AS construct shall therefore be included in the derived definition if it is to be referenced in a managed object class definition.

8.8 Attribute group template

8.8.1 Overview

This template allows attribute groupings to be defined; such groupings are applicable to situations where it is desirable to operate upon the collection of attributes that are members of the group. The behaviour definitions for a given managed object class define the meaning of Get attribute value and Replace with default value operations when applied to attribute groups. Each member of the group shall itself be defined as a single- or set-valued attribute type.

The Attribute Group template defines the minimum set of attributes that constitute the group and the object identifier value that is used to name the group. Each managed object class definition that references an attribute group may extend the membership of the group by adding new attributes to the group, unless the group has been defined to be of fixed membership; such extensions apply only to instances of the managed object class in which the extension is defined. The attributes identified in the Attribute Group template define the minimum membership of the group in all managed object class definitions that reference the group.

If an extensible attribute group is present in a managed object class definition, then all attributes defined for the group, either within the body of the Attribute Group template or added to the attribute group by the managed object class definition, shall be present in the package which references the group or in one of the mandatory packages of the managed object class.

If a fixed attribute group is present in a managed object class definition, then all attributes defined for the group shall be present in the package which references the group.

8.8.2 Template structure

```
<group-label> ATTRIBUTE GROUP
           [GROUP ELEMENTS      <attribute-label> [,<attribute-label>]* ;
           ]
           [FIXED ;
           ]
           [DESCRIPTION         delimited-string ;
           ]
REGISTERED AS object-identifier ;
```

8.8.3 Supporting definitions

8.8.3.1 `GROUP ELEMENTS <attribute-label> [,<attribute-label>]*`

When present, this construct defines the set of `attribute-labels` that identify the individual attributes that form those elements of the attribute group that shall be present in all instances of the attribute group, each of which shall be defined by means of the Attribute template. The behaviour definitions for a given managed object class define the meaning of Get attribute value and Replace with default value operations when applied to attribute groups.

NOTE 1 – This does not imply that there is a behaviour specification specific to the attribute group itself that does not also apply to the attributes when treated individually.

All attributes in a Group shall be members of the managed object class definition(s) which reference the group; i.e., each attribute that is a member of the group for a given managed object class shall be referenced by the Attributes construct in one or more of the packages that are referenced by the managed object class definition.

8.8.3.2 `FIXED`

When present, this construct indicates that the attribute group is defined to be of fixed membership.

8.8.3.3 `DESCRIPTION delimited-string`

This construct allows a description of the semantics of the grouping to be expressed; e.g., `'Group of all state attributes in the managed object'`. No restrictions are placed on the character set used to represent the `DESCRIPTION` and no further structure within the `DESCRIPTION` is defined.

This construct shall not be used as a means of defining behavioural aspects of the group or its members.

8.8.3.4 `REGISTERED AS object-identifier`

The `object-identifier` value provides a globally unique identifier for the attribute group definition. This value shall be used in management protocol when it is necessary to identify the attribute group. The attribute group identified by this `object-identifier` value in the context of a managed object includes any attributes defined in the body of the Attribute Group template, plus, for extensible attribute groups, any attributes added to the group as a consequence of the definition of the elements of the Managed Object Class template that applied to the instantiation of the managed object.

NOTE 2 – The Attribute Group template defines the set of attributes (which may be the empty set) that are always members of the group; in the case of an extensible attribute group, this set may be extended by the Attribute Groups construct in the Package template, in order to extend the set for the purposes of particular managed object class definitions. This technique may be appropriate where it is desirable to define an attribute group, members of which share some common semantic (such as "state attributes"), but where the number of attributes of that semantic type that may be present in a given managed object class is determined at instantiation time, or where different groupings of attributes with the same semantics are required in multiple managed object classes. In general, it is only possible to determine the composition of an extensible attribute group at managed object instantiation time, at which point it is known which packages, and therefore which attributes, have been instantiated.

8.9 Behaviour template

8.9.1 Overview

This template is used to define behavioural aspects of managed object classes, name bindings, parameters and attribute, action and notification types. The Behaviour template is intended to permit extension provisions, but behaviour specifications shall not change the semantics of previously-defined information. If information is left undefined, the behaviour definition shall be explicit about what is undefined.

NOTES

1 Behaviour templates should be used to convey semantics that are not completely described by other templates. Specifically, definers should not rely on labels to convey semantics.

2 A behaviour statement should be worded in terms of the managed object whose managed object class definition includes it.

8.9.2 Template structure

```
<behaviour-definition-label> BEHAVIOUR
          DEFINED AS     delimited-string ;
```

8.9.3 Supporting definitions

8.9.3.1 `DEFINED AS delimited-string`

The text contained in the `delimited-string` gives a definition of an aspect of the behaviour of a managed object class or associated name bindings, parameters, attributes, actions or notifications. This definition may be documented by use of natural language text or by the use of formal description techniques. The text may be a textual reference to a clause or sub-clause of a document or Recommendation | International Standard. No restrictions are placed on the character set used to represent the `delimited-string` and no further structure within the text is defined.

8.10 Action template

8.10.1 Overview

This template is used to define the behaviour and syntax associated with a particular Action type. Action types defined by means of this template may be carried by the M-ACTION service defined in CCITT Rec. X.710 | ISO/IEC 9595. The major elements of the definition are described below.

8.10.1.1 Behaviour

The definition of an Action type shall specify the functionality of the Action in terms of the effect that it has on the managed object class upon which it operates. Where the action may apply to more than one managed object class, the behavioural description should be limited to those features of the behaviour that are common to all managed object classes; further managed object class specific behaviour related to those Actions is described as part of the definition of the managed object class itself.

8.10.1.2 Mode of operation

The definition of an Action type shall indicate whether the action is always confirmed or whether it may be confirmed or unconfirmed at the manager's option.

8.10.1.3 Abstract syntax

The definition of the Action type shall specify any syntax that shall be used to convey the action information and action reply parameters of the M-ACTION service defined in CCITT Rec. X.710 | ISO/IEC 9595. These syntaxes are defined by means of ASN.1 data types.

NOTES

1 Unless it is specifically intended to preclude future extension of the arguments of an action, it is recommended that the information and reply syntaxes of the action should be defined in an extensible manner, by inclusion of the ASN.1 type `SET OF ManagementExtension` (as defined in CCITT Rec. X.721 | ISO/IEC 10165-2) as an optional field.

2 It is recommended that the base data type chosen for the information and reply syntax is the `SEQUENCE` type.

8.10.1.4 Action identifiers

The value of the object identifier associated with the Action type definition is used to identify the Action type in management protocol.

8.10.1.5 Parameters

The definition of the Action type may identify action information or action reply parameters, or specific error parameters associated with the Action type.

8.10.2 Template structure

```
<action-label> ACTION
           [BEHAVIOUR                    <behaviour-definition-label>
                                         [,<behaviour-definition-label>]* ;
           ]
           [MODE CONFIRMED ;
           ]
           [PARAMETERS                   <parameter-label> [,<parameter-label>]* ;
           ]
           [WITH INFORMATION SYNTAX      type-reference ;
           ]
           [WITH REPLY SYNTAX            type-reference ;
           ]
REGISTERED AS object-identifier ;
```

8.10.3 Supporting definitions

8.10.3.1 `BEHAVIOUR <behaviour-definition-label> [,<behaviour-definition-label>]*`

When present, this defines the behaviour of the Action, the parameters that shall be specified with the Action, the results that the Action may generate and their meaning. The `behaviour-definition-labels` reference behaviour descriptions defined by use of the Behaviour template.

8.10.3.2 `MODE CONFIRMED`

If present, the action shall operate in confirmed mode. If absent, the action may be confirmed or unconfirmed at the discretion of the manager.

8.10.3.3 `PARAMETERS <parameter-label> [,<parameter-label>]*`

The `parameter-labels` identify action information or action reply parameters, or processing failures associated with the Action type. See clause A.7 for an example.

8.10.3.4 `WITH INFORMATION SYNTAX type-reference`

If present, the `type-reference` identifies the ASN.1 data type that describes the structure of the action information parameter that is carried in management protocol. If absent, there is no action-specific information associated with the action invocation.

8.10.3.5 `WITH REPLY SYNTAX type-reference`

If present, the `type-reference` identifies the ASN.1 data type that describes the structure of the action reply parameter that is carried in management protocol. If absent, there is no action-specific information associated with the action reply.

8.10.3.6 `REGISTERED AS object-identifier`

The `object-identifier` value provides a globally unique identifier for the action type definition. This value is used in management protocol when it is necessary to identify the action type.

8.11 Notification template

8.11.1 Overview

This template is used to define the behaviour and syntax associated with a particular Notification type. Notification types defined by means of this template may be carried in event reports by the M-EVENT REPORT service defined in CCITT Rec. X.710 | ISO/IEC 9595. The major elements of the definition are described below.

8.11.1.1 Behaviour

The definition of a Notification type shall specify the circumstances under which a notification of the type is generated.

8.11.1.2 Abstract syntax

The definition of the Notification type shall specify any syntax that shall be used to convey the event information and event reply parameters of the M-EVENT REPORT service defined in CCITT Rec. X.710 | ISO/IEC 9595. The template also permits the allocation of attribute values to fields in the syntax.

NOTES

1 Unless it is specifically intended to preclude future extension of the arguments of a notification, it is recommended that the information and reply syntaxes of the notification should be defined in an extensible manner, by inclusion of the ASN.1 type `SET OF ManagementExtension` (as defined in CCITT Rec. X.721 | ISO/IEC 10165-2) as an optional field.

2 It is recommended that the base data type chosen for the information and reply syntax is the `SEQUENCE` type.

8.11.1.3 Notification naming

The value of the object identifier associated with the Notification definition is used to identify the Event type in management protocol.

8.11.1.4 Parameters

The definition of the Notification type may identify event information or event reply parameters, or specific error parameters associated with the Notification type.

8.11.2 Template structure

```
<notification-label> NOTIFICATION
           [BEHAVIOUR                   <behaviour-definition-label> [,<behaviour-
                                        definition-label>]* ;
           ]
           [PARAMETERS                  <parameter-label> [,<parameter-label>]* ;
           ]
           [WITH INFORMATION SYNTAX     type-reference
               [AND ATTRIBUTE IDS              <field-name> <attribute-label>
                                               [,<field-name> <attribute-label>]*
               ] ;
           ]
           [WITH REPLY SYNTAX           type-reference ;
           ]
REGISTERED AS object-identifier ;
```

8.11.3 Supporting definitions

8.11.3.1 BEHAVIOUR <behaviour-definition-label> [,<behaviour-definition-label>]*

If present, this construct defines the behaviour of the notification, the data that shall be specified with the notification, the results that the notification may generate and their meaning. The `behaviour-definition-label` references a behaviour description defined by use of the Behaviour template.

8.11.3.2 PARAMETERS <parameter-label> [,<parameter-label>]*

The `parameter-labels` identify event information or event reply parameters, or processing failures associated with the Notification type. See clause A.8 for an example.

8.11.3.3 WITH INFORMATION SYNTAX type-reference [AND ATTRIBUTE IDS <field-name> <attribute-label> [,<field-name> <attribute-label>]*]

If present, this construct identifies the ASN.1 data type that describes the structure of the notification information that is carried in management protocol, and permits the association of attribute identifiers with named fields in the abstract syntax. If absent, there is no notification-specific information associated with the notification invocation. If the `AND ATTRIBUTE IDS` option is present, the `field-name` shall be a label defined within the abstract syntax referenced by the `type-reference` that appears in the construct. The data type that is labelled by the `field-name` is used to carry values of the attribute referenced by `attribute-label`. The ASN.1 data type of the attribute shall be the same as the data type referenced by `field-name`.

No label within a `SET OF` or `SEQUENCE OF` type nor any label within the definition of such an element can be used as a field-name since labels within such repeating constructs cannot always refer unambiguously to a single instance of a data type. Similarly, no label of a component of a `CHOICE`, `SET` or `SEQUENCE` type nor any label within the definition of such a component can be used as a field-name if the labelled component appears multiple times within the type definition.

8.11.3.4 WITH REPLY SYNTAX type-reference

If present, this construct identifies the ASN.1 data type that describes the structure of the notification reply that is carried in management protocol. If absent, there is no notification-specific information associated with the notification reply.

The reply syntax is used where a notification is sent using the confirmed mode of the CMIS M-EVENT- REPORT service. Event confirmations are not returned to the managed object. The decision whether to send a notification in the confirmed or unconfirmed mode is a matter for the agent to determine based on policies associated with the manager. When the `WITH REPLY SYNTAX` construct is omitted from a notification definition but the notification is sent in the confirmed mode, the confirmation will not include reply information.

8.11.3.5 REGISTERED AS object-identifier

The `object-identifier` value provides a globally unique identifier for the notification type definition. This value is used in management protocol when it is necessary to identify the notification type.

ANNEX A

Examples of use of the guidelines

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

The examples shown in this Annex are intended to give examples of use of the template notation, not to provide definitions that are useful in real implementations. In particular, the behaviour definitions attached to these examples are, of necessity, somewhat artificial. Further examples of use of the notation that are of practical use in the development of managed object class definitions may be found in CCITT Rec. X.721 | ISO/IEC 10165-2.

A.1 Managed object class definition

```
exampleObjectClass MANAGED OBJECT CLASS
           DERIVED FROM                 "CCITT  Rec.  X.721  (1992)  |  ISO/IEC  10165-2  :
                                        1992":top ;
           CHARACTERIZED BY      examplePackage2 ;
           CONDITIONAL PACKAGES
              examplePackage1           PACKAGE
                 ACTIONS                qOSResetAction,
                                        activate ;
                 NOTIFICATIONS          communicationError ;
              REGISTERED AS      {joint-iso-ccitt   ms(9)   smi(3)   part4(4)   package(4)
                                 examplepack1(0)} ;
              PRESENT IF         !conformance class 2 of underlying resource implemented
                                 as described in ISO/IEC XXXX! ;

REGISTERED AS     {joint-iso-ccitt    ms(9)    smi(3)    part4(4)    managedObjectClass(3)
                  exampleclass(0)} ;
```

NOTE – This template uses the in-line documentation option for documenting the conditional package.

A.2 Name binding definition

```
exampleNameBinding NAME BINDING
           SUBORDINATE OBJECT CLASS     exampleObjectClass ;
           NAMED BY
           SUPERIOR OBJECT CLASS        "CCITT  Rec.  X.721  (1992)  |  ISO/IEC  10165-2  :
                                        1992":system ;
           WITH ATTRIBUTE        objectName ;
           BEHAVIOUR
              containmentBehaviour             BEHAVIOUR
                 DEFINED AS      !A maximum of 3 instances of exampleObjectClass may be
                                 contained in any instance of "CCITT Rec. X.721 | ISO/IEC
                                 10165-2":system.!
              ;
           ;
           CREATE   WITH-AUTOMATIC-INSTANCE-NAMING createErrorParameter ;
           DELETE   DELETES-CONTAINED-OBJECTS ;

REGISTERED AS {joint-iso-ccitt ms(9) smi(3) part4(4) nameBinding(6) examplenb(0)} ;
```

NOTE – This template uses the in-line documentation option for documenting the behaviour.

A.3 Parameter definitions

```
pDUHeader PARAMETER
           CONTEXT          EVENT-INFO;
           WITH SYNTAX      ParameterModule.PDUString;
           BEHAVIOUR
             pDUHeaderBehaviour BEHAVIOUR
                 DEFINED AS       !PDU header. Carried in the CMIP eventInfo field.!
             ;
           ;

REGISTERED AS {joint-iso-ccitt ms(9) smi(3) part4(4) parameter(5) pduheaderparam(0)};

createErrorParameter       PARAMETER
           CONTEXT          SPECIFIC-ERROR ;
           WITH SYNTAX      ParameterModule.ErrorInfo1 ;
           BEHAVIOUR
             createErrorBehaviour    BEHAVIOUR
                 DEFINED AS          !If    the    maximum    number    of    instances    of
                                     exampleObjectClass   exist   within   the   containing
```

```
                                             managed  object,  attempts  to  create  additional
                                             instances  will  result  in  the  return  of  a  CMIP
                                             Processing Failure error where the SpecificErrorInfo
                                             field is of the form
                                                SpecificErrorInfo ::= SEQUENCE {
                                                      errorid            OBJECT IDENTIFIER,
                                                      errorinfo    ANY DEFINED BY errorid }
                                             The OBJECT IDENTIFIER carried in errorid shall be the
                                             value  under  which  this  parameter  definition  is
                                             registered.  The type carried in errorinfo shall be
                                             the type identified by the WITH SYNTAX construct of
                                             this parameter definition.  The value carried by this
                                             type  indicates  the  number  of  instances  of  this
                                             managed  object  class  that  currently  exist  in  the
                                             containing managed object.!
                   ;
           ;
       REGISTERED AS {joint-iso-ccitt ms(9) smi(3) part4(4) parameter(5) createrror(1)} ;

serviceProviderErrorResponseReason PARAMETER
           CONTEXT        ACTION-REPLY;
           WITH SYNTAX    ParameterModule.ServiceProviderErrorResponseReason;
           BEHAVIOUR
             serviceProviderErrorResponseReasonBehaviour BEHAVIOUR
                  DEFINED AS    !Returned  in  the  responseParameters  field  of  the  CMIP
                                actionReplyInfo    if    responseCode    has    the    value
                                serviceProviderErrorResponse.!
             ;
           ;
REGISTERED AS {joint-iso-ccitt ms(9) smi(3) part4(4) parameter(5) sperrorrsp(2)};
```

NOTE – These templates use the in-line documentation option for documenting the behaviour.

A.4 Package definition

```
examplePackage2   PACKAGE
                  BEHAVIOUR       exampleClassBehaviour ;
                  ATTRIBUTES      objectName              GET ,
                                  qOS-Error-Cause         GET ,
                                  qOS-Error-Counter       PERMITTED VALUES
                                                          AttributeModule.QOSCounterRange
                                                          REQUIRED VALUES
                                                          AttributeModule.QOSCounterRange
                                                          GET ;
               ATTRIBUTE GROUPS   qOS-Group ;
               NOTIFICATIONS      protocolError;
REGISTERED AS {joint-iso-ccitt ms(9) smi(3) part4(4) package(4) examplepack2(1)} ;
```

NOTE – As this template is not used as a conditional package, the REGISTERED AS construct is not strictly necessary, but it is easier to include the registration at specification time than it would be to add it later if it became necessary in the future to use this package as a conditional package.

A.5 Attribute definitions

```
objectName ATTRIBUTE
           WITH ATTRIBUTE SYNTAX     AttributeModule.ObjectName ;
           MATCHES FOR               EQUALITY ;
REGISTERED AS {joint-iso-ccitt ms(9) smi(3) part4(4) attribute(7) objectname(0)} ;

qOS-Error-Cause ATTRIBUTE
           WITH ATTRIBUTE SYNTAX     AttributeModule.QOSErrorCause ;
           MATCHES FOR               EQUALITY ;
           BEHAVIOUR                 qOSErrorBehaviour ;
REGISTERED AS {joint-iso-ccitt ms(9) smi(3) part4(4) attribute(7) qoscause(1)} ;

qOS-Error-Counter ATTRIBUTE
           WITH ATTRIBUTE SYNTAX     AttributeModule.QOSErrorCounter ;
           MATCHES FOR               EQUALITY, ORDERING ;
           BEHAVIOUR                 qOSCounterBehaviour ;
REGISTERED AS {joint-iso-ccitt ms(9) smi(3) part4(4) attribute(7) qoscount(2)} ;
```

A.6 Attribute group definition

```
qOS-Group ATTRIBUTE GROUP
          GROUP ELEMENTS      qOS-Error-Cause, qOS-Error-Counter ;
          DESCRIPTION         !Attribute group that includes all QOS-related attributes
                              in a managed object class! ;
REGISTERED AS {joint-iso-ccitt ms(9) smi(3) part4(4) attributeGroup(8) qosgroup(0)} ;
```

A.7 Action definitions

```
qOSResetAction    ACTION
          BEHAVIOUR
            reset BEHAVIOUR
                 DEFINED AS   !<Definition of the reset behaviour and its effect on the
                              operation of the managed object, etc..>!
            ;
          ;
          MODE CONFIRMED ;
REGISTERED AS {joint-iso-ccitt ms(9) smi(3) part4(4) action(9) reset(0)} ;
```

NOTE – This action definition makes use of the in-line documentation option for the action Behaviour template. No abstract syntaxes are defined for invoke or reply.

```
activate ACTION
          BEHAVIOUR
            activateBehaviour BEHAVIOUR
                 DEFINED AS   !Enables the managed object for operation. If the action
                              succeeds, the value successResponse is returned in the
                              responseCode parameter of the CMIP actionReplyInfo. If the
                              action fails because of a problem with the
                              underlying service provider, responseCode is set to the
                              value serviceProviderErrorResponse and the parameter
                              serviceProviderErrorResponseReason returned to indicate
                              the cause of the problem.!
            ;
          ;
          MODE CONFIRMED;
          PARAMETERS          serviceProviderErrorResponseReason ;
          WITH REPLY SYNTAX   ActionModule.ActivateReply;
REGISTERED AS {joint-iso-ccitt ms(9) smi(3) part4(4) action(9) activate(1)} ;
```

A.8 Notification definitions

```
communicationError  NOTIFICATION
          BEHAVIOUR                communicationErrorBehaviour ;
          WITH INFORMATION SYNTAX  NotificationModule.ErrorInfo ;
          WITH REPLY SYNTAX        NotificationModule.ErrorResult ;
REGISTERED AS     {joint-iso-ccitt ms(9) smi(3) part4(4) notification(10) commerror(0)} ;

protocolError NOTIFICATION
          BEHAVIOUR
            protocolErrorBehaviour BEHAVIOUR
                 DEFINED AS     !Generated when a protocol entity receives a PDU which
                                is invalid or contains a protocol error. The
                                notification includes the header of the received PDU.!
            ;
          ;
          PARAMETERS               pDUHeader;
          WITH INFORMATION SYNTAXNotificationModule.ProtocolError;
REGISTERED AS {joint-iso-ccitt ms(9) smi(3) part4(4) notification(10) protoerror(1)};
```

NOTE – This template uses the in-line documentation option for documenting the behaviour.

A.9 Behaviour definitions

```
qOSCounterBehaviour BEHAVIOUR
           DEFINED AS      !The QOS Error Counter attribute is a wraparound counter that
                           is incremented by one for every occurrence of a QOS Error.  Its
                           value is a positive integer, whose range is specified in any
                           package that references this definition.  When the counter
                           reaches its maximum value, the next increment causes its value
                           to return to zero.! ;

qOSErrorBehaviour BEHAVIOUR
           DEFINED AS      !The QOS Error Cause attribute indicates the reason for a
                           failure in quality of service associated with the managed
                           object.
```

NOTE – The relationship between the permitted attribute values and the operation of the managed object itself are defined by the behaviour definitions associated with the managed object class definition. !;

```
communicationErrorBehaviour BEHAVIOUR
           DEFINED AS      !The CommunicationError notification is generated by the
                           managed object class when a communication error is detected
                           by the managed object. The notification may contain any
                           combination of the parameters Probable Cause, Severity, Trend
                           Indication, Backed Up Status, Diagnostic Info, Proposed Repair
                           Action, Threshold Info, State Change and Other Info.
```

NOTE – The precise definition of what constitutes a communication error and the parameter values that apply is managed object class specific. In a practical example, this Behaviour definition could, for example, refer to pieces of specification in a base standard in order to specify the behaviour. !;

```
exampleClassBehaviour BEHAVIOUR
           DEFINED AS      !<....Description of managed object class behaviour, including
                           - How its attributes attain particular values and what they mean,
                           - What circumstances cause notifications to be generated,
                           - Etc. >! ;
```

A.10 ASN.1 modules

```
AttributeModule {joint-iso-ccitt ms(9) smi(3) part4(4) asn1Module(2) attributes(0)}

DEFINITIONS ::= BEGIN

ObjectName ::= GraphicString

QOSErrorCause ::= INTEGER {
               responseTimeExcessive (0),
               queueSizeExceeded (1),
               bandwidthReduced (2),
               retransmissionRateExcessive (3) }

QOSErrorCounter ::= INTEGER

QOSCounterRange ::= QOSErrorCounter {0..4294967296}   -- Range is 32 bits

END

NotificationModule {joint-iso-ccitt ms(9) smi(3) part4(4) asn1Module(2) notifications(1)}
DEFINITIONS ::= BEGIN

IMPORTS

ProbableCause, PerceivedSeverity, TrendIndication, BackedUpStatus, ProposedRepairActions,
ThresholdInfo, ManagementExtension

FROM Attribute.ASN1Module {joint-iso-ccitt ms(9) smi(3) part2(2) asn1Module(2) 1} ;

ErrorInfo ::= SET{

          [0] ProbableCause            OPTIONAL,
          [1] PerceivedSeverity        OPTIONAL,
          [2] TrendIndication          OPTIONAL,
          [3] BackedUpStatus           OPTIONAL,
          [4] ProposedRepairActions    OPTIONAL,
          [5] ThresholdInfo            OPTIONAL,
          [6] OtherInfo                OPTIONAL }
```

```
ErrorResult ::= NULL

OtherInfo          ::= SET OF ManagementExtension

ProtocolError      ::= SET OF ManagementExtension

END

ActionModule {joint-iso-ccitt ms(9) smi(3) part4(4) asn1Module(2) actions(2)}
DEFINITIONS ::= BEGIN

IMPORTS

OperationalState, ManagementExtension

FROM Attribute.ASN1Module {joint-iso-ccitt ms(9) smi(3) part2(2) asn1Module(2) 1} ;

ActivateReply ::= SEQUENCE {
                    operationalStatus   [0] OperationalState,
                    responseCode        [1] INTEGER  {successResponse (0),
                                                     serviceProviderErrorResponse (1) },
                    responseParams      [2] SET OF ManagementExtension OPTIONAL }

END

ParameterModule {joint-iso-ccitt ms(9) smi(3) part4(4) asn1Module(2) parameters(3)}

DEFINITIONS ::= BEGIN

ErrorInfo1 ::= INTEGER

ServiceProviderErrorResponseReason ::= ENUMERATED {
             insufficientResources (0),
             providerDoesNotExist (1),
             providerNotAvailable (2),
             requiredServiceNotAvailable (3) }

PDUString ::= OCTETSTRING

END
```