INTERNATIONAL TELECOMMUNICATION UNION

CCITT

THE INTERNATIONAL
TELEGRAPH AND TELEPHONE
CONSULTATIVE COMMITTEE

X.501

(11/1988)

SERIES X: DATA COMMUNICATION NETWORKS:
DIRECTORY

THE DIRECTORY-MODELS

Reedition of CCITT Recommendation X.501 published in the Blue Book, Fascicle VIII.8 (1988)

NOTES

1 CCITT Recommendation X.501 was published in Fascicle VIII.8 of the *Blue Book*. This file is an extract from the *Blue Book.* While the presentation and layout of the text might be slightly different from the *Blue Book* version, the contents of the file are identical to the *Blue Book* version and copyright conditions remain unchanged (see below).

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

Recommendation X.501

THE DIRECTORY-MODELS [1)]

(Melbourne, 1988)

CONTENTS

0 *Introduction*

1 *Scope and field of application*

2 *References*

3 *Definitions*

4 *Abbreviations*

SECTION 1 – *Directory model*

5 *Directory model*

SECTION 2 – *Information model*

6 *Directory information base*

7 *Directory entries*

8 *Names*

9 *Directory schema*

SECTION 3 – *Security model*

10 *Security*

Annex A – The mathematics of trees

Annex B – Object identifier usage

Annex C – Information framework in ASN.1

Annex D – Alphabetical index of definitions

Annex E – Name design criteria

Annex F – Access control

1) Recommendations X.501 and ISO 9594-2, The Directory-Models were developed in close collaboration and are technically aligned.

0 Introduction

0.1 This document, together with the others of the series, has been produced to facilitate the interconnection of information processing systems to provide directory services. A set of such systems, together with the directory information which they hold, can be viewed as an integrated whole, called the Directory. The information held by the Directory, collectively known as the Directory Information Base (DIB), is typically used to facilitate communication between, with or about objects such as application entities, people, terminals and distribution lists.

0.2 The Directory plays a significant role in Open Systems Interconnection, whose aim is to allow, with a minimum of technical agreement outside of the interconnection standards themselves, the interconnection of information processing systems:

- from different manufacturers;
- under different managements;
- of different levels of complexity; and
- of different ages.

0.3 This Recommendation provides a number of different models for the Directory as a framework for the other Recommendations. The models are the overall (functional) model; the organizational model; the security model; and the information framework. The latter describes the manner in which the Directory organizes the information it holds. It describes, for example, how information about objects is grouped to form directory entries for those objects and how that information provides names for objects.

0.4 Annex A summarizes the mathematical terminology associated with tree structures.

0.5 Annex B summarizes the usage of ASN.1 object identifiers in this series of Recommendations.

0.6 Annex C provides the ASN.1 module which contains all of the definitions associated with the information framework.

0.7 Annex D lists alphabetically the terms defined in this document.

0.8 Annex E describes some criteria that can be considered in designing names.

0.9 Annex F describes guidelines for access control.

1 Scope and field of application

1.1 The models defined in this Recommendation provide a conceptual and terminological framework for the other Recommendations which define various aspects of the Directory.

1.2 The functional and organizational models define ways in which the Directory can be distributed, both functionally and administratively.

1.3 The security model defines the framework within which security features, such as access control, are provided in the Directory.

1.4 The information model describes the logical structure of the DIB. From this viewpoint, the fact that the Directory is distributed, rather than centralized, is not visible. The other Recommendations in the series make use of the concepts of the information framework. Specifically:

a) the service provided by the Directory is described (in Recommendation X.511) in terms of the concepts of the information framework: this allows the service provided to be somewhat independent of the physical distribution of the DIB;

b) the distributed operation of the Directory is specified (in Recommendation X.518) so as to provide that service, and therefore maintain that logical information structure, given that the DIB is in fact highly distributed.

2 References

Recommendation X.200 – Open Systems Interconnection – Basic Reference Model.

Recommendation X.500 – The Directory – Overview of Concepts, Models and Services.

Recommendation X.509 – The Directory – Authentication Framework.

Recommendation X.511 – The Directory – Access and System Services Definition.

Recommendation X.518 – The Directory – Procedures for Distributed Operation.

Recommendation X.519 – The Directory – Access and System Protocols Specification.

Recommendation X.520 – The Directory – Selected Attribute Types.

Recommendation X.521 – The Directory – Selected Object Classes.

3 Definitions

Definitions of terms are included at the beginning of individual clauses, as appropriate. An index of these terms is provided in AnnexD for easy reference.

4 Abbreviations

ADDMD	Administration Directory Management Domain
AVA	Attribute value assertion
DIB	Directory Information Base
DIT	Directory Information Tree
DMD	Directory Management Domain
DSA	Directory System Agent
DUA	Directory User Agent
PRDMD	Private Directory Management Domain
RDN	Relative distinguished name.

SECTION 1 – *Directory model*

5 Directory model

5.1 *Definitions*

a) *access point:* The point at which an abstract service is obtained.

b) *Administration Directory Management Domain (ADDMD):* A DMD which is managed by an Administration.

Note – The term "Administration" denotes a public telecommunications administration or other organization offering public telecommunications services;

c) *Administrative Authority:* An entity which has administrative control over all entries stored within a single Directory System Agent;

d) *The Directory:* A repository of information about objects and which provides directory services to its users which allow access to the information;

e) *Directory Management Domain (DMD):* A collection of one or more DSAs and zero or more DUAs which is managed by a single organization;

f) *Directory System Agent (DSA):* An OSI application process which is part of the Directory;

g) *(Directory) user:* The end user of the Directory, i.e. the entity or person which accesses the Directory;

h) *Directory User Agent (DUA):* An OSI application process which represents a user in accessing the Directory;

Note – DUAs may also provide a range of local facilities to assist users, compose queries and interpret the responses;

i) *Private Directory Management Domain (PRDMD):* A DMD which is managed by an organization other than an Administration.

5.2 *The Directory and its users*

5.2.1 A directory user (e.g. a person or an application process) obtains directory services by accessing the *Directory*. More precisely, it is a *Directory User Agent* (DUA), which actually accesses the Directory and interacts with it to obtain the service on behalf of a particular user. The Directory provides one or more *access points* at which such accesses can take place. These concepts are illustrated in Figure1/X.501.

5.2.2 The services provided by the Directory are defined in RecommendationX.511.

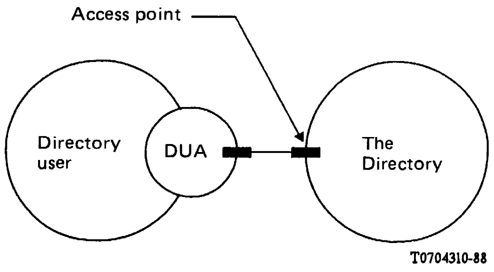


FIGURE 1/X.501

Access to the Directory

5.2.3 The Directory is a repository of information about objects, and the directory services it provides to its users are concerned with various kinds of access to this information. The information is collectively known as the *Directory Information Base* (DIB). A model for the DIB is defined in section 2 of this Recommendation.

5.2.4 A DUA is manifested as an application-process. Each DUA represents precisely one directory user.

Note 1 – Some open systems may provide a centralised DUA function retrieving information for the actual users (application-processes, persons, etc.). This is transparent to the Directory.

Note 2 – The DUA functions and a DSA (see § 5.3.1) can be within the same open system, and it is an implementation choice whether to make one or more DUAs visible within the OSI environment as application-entities.

Note 3 – A DUA will likely exhibit local behaviour and structure which is outside the scope of envisaged Recommendations. For example, a DUA which represents a human directory user may provide a range of local facilities to assist its user to compose queries and interpret the responses.

5.3 *Functional model*

5.3.1 The Directory is manifested as a set of one or more application-processes known as Directory System Agents (DSAs), each of which provides zero, one or more of the access points. This is illustrated in Figure 2/X.501. Where the Directory is composed of more than one DSA, it is said to be distributed. The procedures for the operation of the Directory when it is distributed are specified in Recommendation 518.

Note – A DSA will likely exhibit local behaviour and structure which is outside the scope of envisaged Recommendations. For example, a DSA which is responsible for holding some or all of the information in the DIB will normally do so by means of a database, the interface to which is a local matter.

5.3.2 A particular pair of application-processes which need to interact in the provision of directory services (either a DUA and a DSA, or two DSAs) may be located in different open systems. Such an interaction is carried out by means of OSI directory protocols, as specified in Recommendation 519.

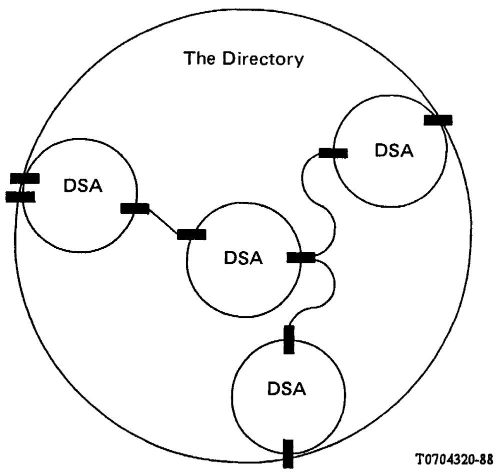


FIGURE 2/X.501

The Directory provided by multiple DSAs

5.4 *Organizational model*

5.4.1 A set of one or more DSAs and zero or more DUAs managed by a single organization may form a Directory Management Domain (DMD).

Note – The organization which manages a DMD may be an Administration (i.e. a public telecommunications administration or other organization offering public telecommunications services) in which case the DMD is said to be an Administration DMD (ADDMD); otherwise it is a Private DMD (PRDMD). It should be recognized that the provision of support for private directory systems by CCITT members falls within the framework of national regulations. Thus, the technical possibilities described may or may not be offered by an Administration which provides directory services. The internal operation and configuration of private DMDs is not within the scope of envisaged CCITT Recommendations.

5.4.2 Management of a DUA by a DMD implies an ongoing responsibility for service to that DUA, e.g.maintenance, or in some cases ownership, by the DMD.

5.4.3 The organization concerned may or may not elect to make use of this series of Recommendations to govern any interactions among DUAs and DSAs which are wholly within the DMD.

5.4.4 Each DSA is administered by an Administrative Authority. This entity has control over all object entries and alias entries stored by that DSA. This includes responsibilities for the Directory schema being used to guide the creation and modification of entries (see § 9). The structure and allocation of names is the responsibility of a naming authority [see § 8.1f)] and the role of the Administrative Authority is to implement these naming structures in the schema.

SECTION 2 – *Information model*

6 **Directory information base**

6.1 *Definitions*

a) *alias entry:* an entry of the class "alias" containing information used to provide an alternative name for an object;

b) *Directory Information Base (DIB):* the complete set of information to which the Directory provides access and which includes all of the pieces of information which can be read or manipulated using the operations of the Directory;

c) *Directory Information Tree (DIT):* the DIB considered as a tree, whose vertices (other than the root) are the Directory entries;

Note – The term DIT is used instead of DIB only in contexts where the tree structure of the information is relevant.

d) *(Directory) entry:* a part of the DIB which contains information about an object;

e) *immediate superior (noun)*: relative to a particular entry or object (it must be clear from the context which is intended) the immediately superior entry or object;

f) *immediately superior*

(entry): relative to a particular entry – an entry which is at the initial vertex of an arc in the DIT whose final vertex is that of the particular entry;

(object): relative to a particular object – an object whose object entry is the immediate superior of any of the entries (object or alias) for the second object;

g) *object (of interest):* anything in some "world", generally the world of telecommunications and information processing or some part thereof, which is identifiable (can be named), and which it is of interest to hold information on in the DIB;

h) *object class:* an identified family of objects (or conceivable objects) which share certain characteristics;

i) *object entry:* an entry which is the primary collection of information in the DIB about an object and which can therefore be said to represent that object in the DIB;

j) *subclass:* relative to a superclass – an object class derived from a superclass. The members of the subclass share all the characteristics of another object class (the superclass) and additional characteristics possessed by none of the members of that object class (the superclass);

k) *subordinate/inferior:* the converse of superior;

l) *superclass:* relative to a subclass – an object class from which a subclass is derived;

m) *superior:* (applying to entry or object) immediately superior, or superior to one which is immediately superior (recursively).

6.2 *Objects*

6.2.1 The purpose of the Directory is to hold, and provide access to, information about *objects of interest (objects)* which exist in some "world". An object can be anything in that world which is identifiable (can be named).

Note 1 – The "world" is generally that of telecommunications and information processing or some part thereof.

Note 2 – The objects known to the Directory may not correspond exactly with the set of "real" things in the world. For example, a real-world person may be regarded as two different objects, a business person and a residential person, as far as the Directory is concerned. The mapping is not defined in this Recommendation but is a matter for the users and providers of the Directory in the context of their applications.

6.2.2 The complete set of information to which the Directory provides access is known as the *Directory Information Base* (DIB). All of the pieces of information which can be read or manipulated by the operations of the Directory are considered to be included in the DIB.

6.2.3 An *object class* is an identified family of objects (or conceivable objects) which share certain characteristics. Every object belongs to at least one class. An object class may be a subclass of another object class, in which case the members of the former class (the subclass) are also considered to be members of the latter (the superclass). There may be subclasses of subclasses, etc. to an arbitrary depth.

6.3 *Directory entries*

6.3.1 The DIB is composed of *Directory entries (entries)* each containing information about (describing) a single object.

6.3.2 For any particular object there is precisely one *object entry*, this being the primary collection of information in the DIB about that object. The object entry is said to represent the object.

6.3.3 For any particular object there may, in addition to the object entry, be one or more alias entries for that object which are used to provide alternative names (see § 8.5).

6.3.4 The structure of directory entries is depicted in Figure3/X.501 and described in 7.2.

6.3.5 Each entry contains an indication of the object class and the superclasses of that object class with which the entry is associated. In the case of an object entry, this indicates the class(es) to which the object belongs. In the case of an alias entry, this indicates, by means of a special object class, "alias" (defined in § 9 .4.8.2), that it is in fact an alias entry, and may also indicate to which subclass(es) of the alias object class the entry belongs.

6.4 *The Directory information tree (DIT)*

6.4.1 In order to satisfy the requirements for the distribution and management of a potentially very large DIB, and to ensure that objects can be unambiguously named (see § 8) and their entries found, a flat structure of entries is not likely to be feasible. Accordingly, the hierarchical relationship commonly found among objects (e.g. a person works for a department, which belongs to an organization, which is headquartered in a country) can be exploited, by the arrangement of the entries into a tree, known as the *Directory Information Tree (DIT)*.

Note – An introduction to the concepts and terminology of tree structures can be found in AnnexA.

6.4.2 The component parts of the DIT have the following interpretations:

a) the vertices are the entries. Object entries may be either leaf or non-leaf vertices, whereas alias entries are always leaf vertices. The root is not an entry as such, but can, when convenient to do so (e.g. in the definitions of b) and c) below), can be viewed as a null object entry [see d) below];

b) the arcs define the relationship between vertices (and hence entries). An arc from vertex A to vertexB means that the entry at A is the *immediately superior entry (immediate superior)* of the entry at B, and conversely, that the entry at B is an *immediately subordinate entry (immediate subordinate)* of the entry at A. The *superior entries (superiors)* of a particular entry are its immediate superior together with its superiors (recursively). The *subordinate entries (subordinates)* of a particular entry are its immediate subordinates together with their subordinates (recursively);

c) the object represented by an entry is or is closely associated with the naming authority (see § 8) for its subordinates;

d) the root represents the highest level of naming authority for the DIB.

6.4.3 A superior/subordinate relationship between objects can be derived from that between entries. An object is an *immediately superior object (immediate superior)* of another object if and only if the object entry for the first object is the immediate superior of any of the entries for the second object. The terms *immediately subordinate object*, *immediate subordinate*, *superior* and *subordinate* (applied to objects) have their analogous meanings.

6.4.4 Permitted superior/subordinate relationships among objects are governed by the DIT structure definitions (see § 9.2).

7 Directory entries

7.1 *Definitions*

a) *attribute:* the information of a particular type concerning an object and appearing in an entry describing that object in the DIB;

b) *attribute type:* that component of an attribute which indicates the class of information given by that attribute;

c) *attribute value:* a particular instance of the class of information indicated by an attribute type;

d) *attribute value assertion:* a proposition, which may be true, false or undefined, concerning the values (or perhaps only the distinguished values) of an entry;

Note – In this document the notation "string1 = string2" is used to write down examples of attribute value assertions. In this notation, "string1" is an abbreviation for the "name" of the attribute type, and "string2" is a textual representation of suitable value. Although the attribute types in the examples are often based upon real types, such as those defined in Recommendation X.520 (e.g. "C" stands for "Country", CN for "Common Name"), this is not strictly necessary for the purposes of this document, as the Directory is usually unaware of the meanings of the attribute types in use.

e) *distinguished value:* an attribute value in an entry which has been designated to appear in the relative distinguished name of the entry.

7.2 *Overall structure*

7.2.1 As depicted in Figure 3/X.501, an entry consists of a set of *attributes*.

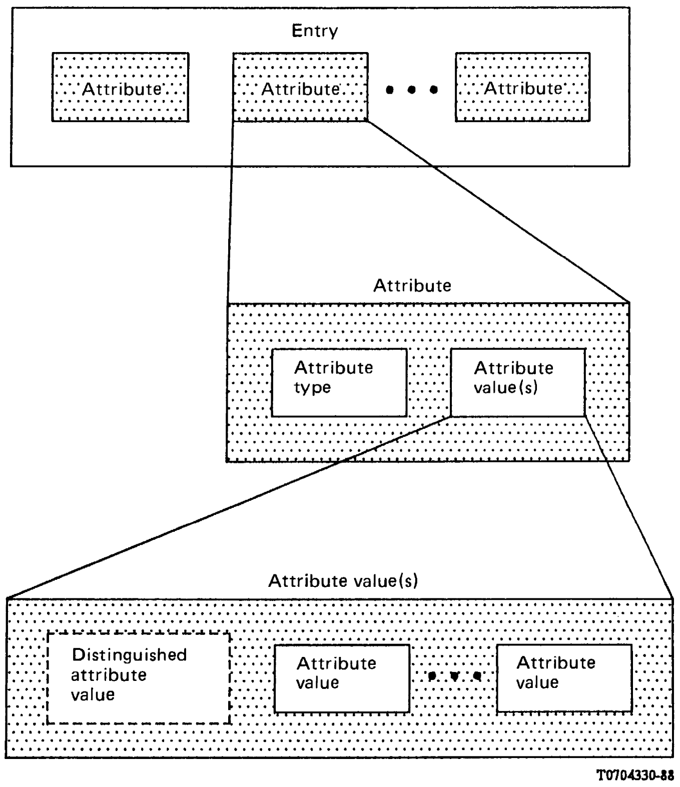


FIGURE 3/X.501

Structure of an entry

7.2.2 Each attribute provides a piece of information about, or describes a particular characteristic of, the object to which the entry corresponds.

Note – Examples of attributes which might be present in an entry include naming information such as the object's personal name, and addressing information, such as its telephone number.

7.2.3 An attribute consists of an *attribute type*, which identifies the class of information given by an attribute, and the corresponding *attribute value(s)*, which are the particular instances of that class appearing in the entry.

```
Attribute ::=
    SEQUENCE{
        type        Attribute Type
        values      SET OF AttributeValue
        -- at least one value is required --}
```

7.3 *Attribute types*

7.3.1 Some attribute types will be internationally standardized. Other attribute types will be defined by national administrative authorities and private organizations. This implies that a number of separate authorities will be responsible for assigning types in a way that ensures that each is distinct from all other assigned types. This is accomplished by identifying each attribute type with an object identifier when the type is defined (as described in § 9.5):

```
Attribute Type ::= OBJECT IDENTIFIER
```

7.3.2 All attributes in an entry must be of distinct attribute types.

7.3.3 There are a number of attribute types which the Directory knows about and uses for its own purposes. They include:

a) **ObjectClass**. An attribute of this type appears in every entry and indicates the object class and superclass(es) to which the object belongs.

b) **AliasedObjectName**. An attribute of this type appears in every alias entry and holds the distinguished name (see § 8.5) of the object which this alias entry describes.

These attributes are (partially) defined in § 9.5.4.

7.3.4 The types of attributes which must or which may appear within an entry (other than as mentioned in § 7.3.3) are governed by rules applying to the indicated object class(es).

7.4 *Attribute values*

7.4.1 Defining an attribute type (see § 9.5) also involves specifying the syntax, and hence data type, to which every value in such attributes must conform. This could be any data type:

```
AttributeValue ::=  ANY
```

7.4.2 At most one of the values of an attribute may be designated as a *distinguished value*, in which case the attribute value appears in the relative distinguished name (see § 8.3) of the entry.

7.4.3 An *attribute value assertion* (AVA) is a proposition, which may be true, false, or undefined, concerning the values (or perhaps only the distinguished values) of an entry. It involves an attribute type and an attribute value.

```
AttributeValueAssertion ::=
        SEQUENCE {AttributeType,AttributeValue}
```

and is:

a) undefined, if any of the following holds:

   i) the attribute type is unknown;

   ii) the attribute syntax for the type has no equality matching rule;

   iii) the value does not conform to the data type of the attribute syntax;

   Note – ii) and iii) normally indicate a faulty AVA; i), however, may occur as a local situation (e.g. a particular DSA has not registered that particular attribute type).

b) true, if the entry contains an attribute of that type, one of whose values matches that value (if the assertion is concerned only with distinguished values, then the matched value must be the distinguished one);

   Note – The matching of values is for equality and involves the matching rule associated with the attribute syntax.

c) false, otherwise.

8 Names

8.1 *Definitions*

a) *alias, alias name:* a name for an object, provided by the use of one or more alias entries in the DIT;

b) *dereferencing:* replacing the alias name for an object by the object's distinguished name;

c) *distinguished name (of an object):* one of the names of the object, formed from the sequence of the RDNs of the object entry and each of its superior entries;

d) *(directory) name:* a construct that singles out a particular object from all other objects. A name must be unambiguous (that is, denote just one object), however it need not be unique (that is, be the only name which unambiguously denotes the object);

e) *purported name:* a construct which is syntactically a name but which has not (yet) been shown to be a valid name;

f) *naming authority:* an authority responsible for the allocation of names. Each object whose object entry is located at a non-leaf vertex in the DIT is, or is closely associated with, a naming-authority;

g) *relative distinguished name (RDN):* a set of attribute value assertions, each of which is true, concerning the distinguished values of a particular entry.

8.2 *Names in general*

8.2.1 A *(directory) name* is a construct that identifies a particular object from among the set of all objects. A name must be unambiguous, that is, denote just one object. However, a name need not be unique, that is be the only name that unambiguously denotes the object.

8.2.2 Syntactically, each name for an object is an ordered sequence of relative distinguished names (see § 8.3).

```
NAME ::=
        CHOICE { --only one possibility for now--
                RDNSequence}

RDNSequence ::= SEQUENCE OF RelativeDistinguishedName
DistinguishedName ::= RDNSequence
```

Note – Names which are formed in other ways than as described herein are a possible future extension.

8.2.3 The null sequence is the name for the root of the tree.

8.2.4 Each initial subsequence of the name of an object is also the name of an object. The sequence of objects so identified, starting with the root and ending with the object being named, is such that each is the immediate superior of that which follows it in the sequence.

8.2.5 A *purported name* is a construct which is syntactically a name but which has not (yet) been shown to be a valid name.

8.3 *Relative distinguished names*

8.3.1 Each entry has a unique relative distinguished name (RDN). An RDN consists of a set of attribute value assertions, each of which is true, concerning the distinguished values of the entry.

```
RelativeDistinguishedName ::=
        SET OF AttributeValueAssertion
```

The set contains exactly one assertion about each distinguished value in the entry.

8.3.2 The RDNs of all of the entries with a particular immediate superior are distinct. It is the responsibility of the relevant naming authority for that entry to ensure that this is so by appropriately assigning distinguished attribute values.

Note – Frequently, an entry will contain a single distinguished value (and the RDN will thus comprise a single AVA); however, under certain circumstances (in order to differentiate), additional values (and hence AVAs) may be used.

8.3.3 The RDN for an entry is chosen when the entry is created. A single value instance of any attribute type may form part of the RDN, depending on the nature of the object class denoted. Allocation of RDNs is considered an administrative undertaking that may or may not require some negotiation between involved organizations or administrations. This Recommendation does not provide such a negotiation mechanism and makes no assumption as to how it is performed. The RDN may be modified if necessary by complete replacement.

Note – RDNs are intended to be long-lived so that the users of the Directory can store the distinguished names of objects (e.g. in the Directory itself) without concerns for their obsolescence. Thus RDNs should be changed cautiously.

8.4 *Distinguished names*

8.4.1 The distinguished name of a given object is defined as being the sequence of the RDNs of the entry which represents the object and those of all of its superior entries (in descending order). Because of the one to one correspondence between objects and object entries, the distinguished name of an object can be considered to also identify the object entry.

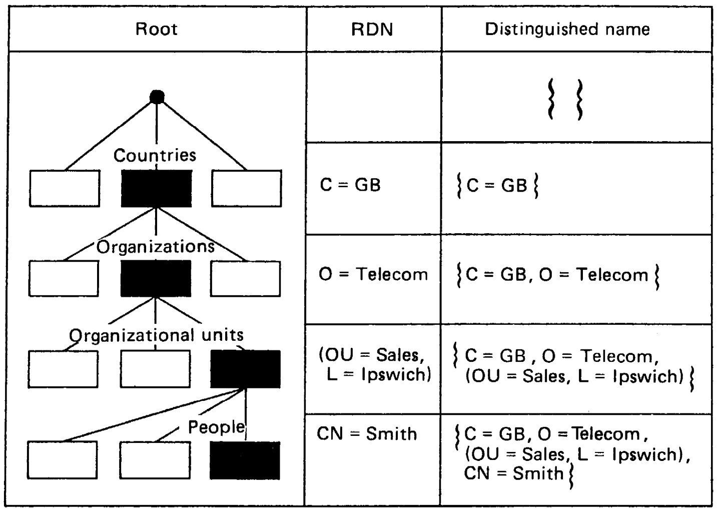
Note 1 – It is preferable that the distinguished names of objects which humans have to deal with be user-friendly.

Note 2 – ISO 7498/3 defines the concept of a primitive name. A distinguished name can be used as a primitive name for the object it identifies because: a)it is unambiguous, b) it is unique, and c) the semantics of its internal structure (a sequence of RDNs) need not (but of course may) be understood by the user of the Directory.

Note 3 – Because only the object entry and its superiors are involved, distinguished names of objects can never involve alias entries.

8.4.2 It proves convenient to define the "distinguished name" of the root and of an alias entry, although in neither case is the name also the distinguished name of an object. The distinguished name of the root is defined to be the null sequence. The distinguished name of an alias entry is defined to be the sequence of RDNs of the alias entry and those of all of its superior entries (in descending order).

8.4.3 An example which illustrates the concepts of RDN and distinguished name appears in Figure4/X.501.



Root	RDN	Distinguished name
		{ }
Countries	C = GB	{C = GB}
Organizations	O = Telecom	{C = GB, O = Telecom}
Organizational units	(OU = Sales, L = Ipswich)	{C = GB, O = Telecom, (OU = Sales, L = Ipswich)}
People	CN = Smith	{C = GB, O = Telecom, (OU = Sales, L = Ipswich), CN = Smith}

T0704340-88

FIGURE 4/X.501

Determination of distinguished names

8.5 *Alias names*

8.5.1 An *alias*, or an *alias name*, for an object is a name at least one of whose RDNs is that of an alias entry. Aliases permit object entries to achieve the effect of having multiple immediate superiors. Therefore, aliases provide a basis for alternative names.

8.5.2 Just as the distinguished name of an object expresses its principal relationship to some hierarchy of objects, so an alias expresses (in the general case) an alternative relationship to a different hierarchy of objects.

8.5.3 An object with an entry in the DIT may have zero or more aliases. It, therefore, follows that several alias entries may point to the same object entry. An alias entry may point to an object entry that is not a leaf entry. Only object entries may have aliases. Thus aliases of aliases are not permitted.

8.5.4 An alias entry shall have no subordinates, that is, an alias entry is a leaf entry.

8.5.5 The Directory makes use of the aliased object name attribute in an alias entry to identify and to find the corresponding object entry.

9 Directory schema

9.1 *Definitions*

a) *Directory Schema:* The set of rules and constraints concerning DIT structure, object class definitions, attribute types and syntaxes which characterize the DIB;

b) *DIT Structure Rule:* A rule, forming part of the Directory Schema which relates an object class (the subordinate) to another object class (the superior) and which allows an entry of the former class to be immediately subordinate to one of the latter classes in the DIT. The rule also governs the attribute type(s) permitted to appear in the (subordinate) entry's RDN, and may impose additional conditions. The schema may contain many such rules.

9.2 *Overview*

9.2.1 The Directory Schema is a set of definitions and constraints concerning the structure of the DIT and the possible ways entries are named, the information that can be held in an entry, and the attributes used to represent that information.

Note 1 – The schema enables the directory system to, for example:

– prevent the creation of subordinate entries of the wrong object-class (e.g. a country as a subordinate of a person);

– prevent the addition of attribute-types to an entry inappropriate to the object-class (e.g. a serial number to a person's entry);

– prevent the addition of an attribute value of a syntax not matching that defined for the attribute type (e.g. a printable string to a bit string).

Note 2 – Dynamic mechanisms for the management of the directory schema are not presently provided by this series of Recommendations.

9.2.2 Formally, the Directory Schema comprises a set of:

a) *DIT Structure* definitions (rules) that define the distinguished names that entries may have and the ways in which they may be related to one another through the DIT;

b) *Object Class* definitions that define the set of mandatory and optional attributes that must be present, and may be present, respectively, in an entry of a given class (see § 6.2.3 of this Recommendation);

c) *Attribute Type* definitions that identify the object identifier by which an attribute is known, its syntax, and whether it is permitted to have multiple values;

d) *Attribute Syntax* definitions that define for each attribute the underlying ASN.1 data type and matching rules.

Figure 5/X.501 summarizes the relationships between the schema definitions on the one side, and the DIT, directory entries, attributes, and attribute values on the other.

9.2.3 The Directory Schema is distributed, like the DIB itself. Each Administrative Authority establishes the part of the schema that will apply for those portions of the DIB administered by the authority.

Note – Distribution of schema information across DSAs managed by different Administrative Authorities is not supported by this series of Recommendations. Such distribution is handled administratively by bilateral agreements.

9.2.4 The specification of what is involved in the definition of DIT structure, object classes, attribute types and attribute syntaxes can be found in § 9.3 – § 9.6 respectively.

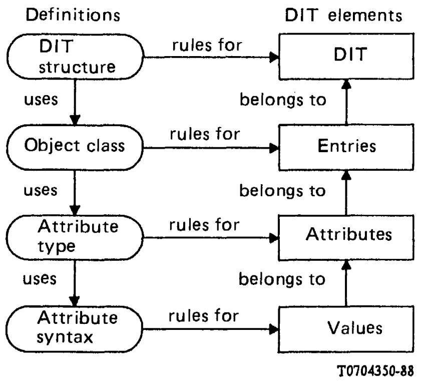


FIGURE 5/X.501

Overview of directory schema

9.3 *DIT structure definition*

9.3.1 A DIT Structure Rule defines the permitted hierarchical relationships between entries and their permitted RDNs. The definition of a DIT Structure Rule involves:

- identifying the subordinate and superior object classes;
- identifying the attribute types which may be involved in subordinate entries' RDNs; and
- (optionally) additional information.

9.3.2 The Directory permits an entry to stand in the relationship of immediate subordinate to another (its immediate superior) only if there exists a DIT Structure definition, contained in the schema (see § 9.2.3) applicable to the portion of the DIB that would contain the entry, for which:

- the entry is of the subordinate object class;
- the immediate superior of the entry is of the superior object class;
- the attribute type(s) forming the entry's RDN is (are) among those permitted;

and

- any conditions imposed by the additional information set element are satisfied.

Note 1 – Techniques for documenting DIT Structure or for representing structure rules in the DIB are not presently provided by this series of Recommendations.

Note 2 – If a DIT Structure Rule permits subordinates or superiors belonging to a particular class, it implicitly (unless explicitly overridden) also allows subordinates or superiors belonging to any object class derived from that class (see § 9.4).

9.3.3 The Directory enforces the defined structure rules at every entry in the DIT. Any attempt to modify the DIT in such a way as to violate the applicable structure rules fails.

9.3.4 A DIT Structure Rule in which an object class is the subordinate is termed a name binding for that object class.

9.3.5 For an object class to be represented by entries in a portion of the DIB, at least one name binding for that object class must be contained in the applicable part of the schema. The schema contains additional name bindings as required.

Note – It is conceivable that an object class, occurring in two distinct schemas, might have distinct name bindings in each schema.

9.4 *Object class definition*

9.4.1 The definition of an object-class involves:

a) optionally, assigning an object-identifier for the object-class;

b) indicating which classes this is to be a subclass of;

c) listing the mandatory attribute types that an entry of the object class must contain in addition to the mandatory attribute types of all its superclasses.

d) listing the optional attribute types that an entry of the object class may contain in addition to the optional attributes of all its superclasses.

Note – An object class without an assigned object identifier is intended for local use as a means of conveniently adding new attribute types to a pre-defined superclass. "This addition allows for a number of possibilities. For example, an Administrative Authority may define an unregistered Object Class so as to permit a user to add any registered attribute to the entry. The Administrative authority may limit the attributes for an entry for a particular object class to those on a locally held list. It may also make particular attributes mandatory for a particular object class, over and above those required by the registered object class definition."

9.4.2 There is one special object class, of which every other class is a subclass. This object class is called "Top" and is defined in § 9.4.8.1.

9.4.3 Every entry shall contain an attribute of type **ObjectClass** to identify the object class and superclasses to which the entry belongs. The definition of this attribute is given in § 9.5.4. The attribute is multivalued. There shall be one value of the attribute for the object class and each of its superclasses for which an object identifier is defined, except that the value of "**Top**" need not be present so long as some other value is present.

Note 1 – The requirement that the **ObjectClass** attribute be present in every entry is reflected in the definition of "**Top**".

Note 2 – Because an object class is considered to belong to all its superclasses, each member of the chain of superclasses up to Top is represented by a value in the object class attribute (and any value in the chain may be matched by a filter).

The **ObjectClass** attribute is managed by the Directory, i.e. it may not be modified by the user.

9.4.4 The Directory enforces the defined object class for every entry in the DIB. Any attempt to modify an entry that would violate the entry's object class definition fails.

Note – In particular, the Directory will prevent:

a) attribute types absent from the object class definition being added to an entry of that object class;

b) an entry being created with one or more absent mandatory attribute types for the object class of the entry;

c) a mandatory attribute type for the object class of the entry being deleted.

9.4.5 The special object class **Alias** is defined in § 9.4.8.2. Every alias entry shall have an object class which is a subclass of this class.

Note – The Directory's dereferencing of alias entries ensures that the values of the **ObjectClass** attribute of an alias entry are rarely seen. It is recommended that appropriate alias object classes be derived from "**Alias**" without assigning an object identifier.

9.4.6 The following ASN.1 macro may (but need not) be used to define an object class. The empty production for **SubclassOf** is permitted only in defining Top:

```
OBJECT-CLASS MACRO ::=
BEGIN

TYPENOTATION   ::=   SubclassOf
                     MandatoryAttributes
                     OptionalAttributes
VALUENOTATION ::=
        value(VALUE OBJECT IDENTIFIER)

SubclassOf ::=
        "SUBCLASS OF" Subclasses |
        empty

Subclasses ::=    Subclass | subclass ","
                  Subclasses
Subclass ::=      value (OBJECT-CLASS)

MandatoryAttributes ::=
        "MUST CONTAIN {"Attributes"}" | empty

OptionalAttributes ::=
        "MAY CONTAIN {"Attributes"}" | empty

Attributes       ::= AttributeTerm | AttributeTerm "," Attributes

AttributeTerm    ::= Attribute | AttributeSet

Attribute        ::= value(ATTRIBUTE)

AttributeSet     ::= value(ATTRIBUTE-SET)

END
```

The correspondence between the parts of the definition, as listed in § 9.4.1, and the various pieces of the notation introduced by the macro, is as follows:

a) the object identifier to the object class is the value supplied in the value assignment of the macro;

b) the superclasses of which this object class is a subclass are those identified by the **SubclassOf** production, i.e. that following "**SUBCLASS OF**";

c) the mandatory attributes are those identified by the list of object identifiers produced by the **MandatoryAttributes** production, i.e. those following "**MUST CONTAIN**";

d) the optional attributes are those identified by the list of object identifiers produced by the **OptionalAttributes** production, i.e. those following "**MAY CONTAIN**".

Note 1 – The object identifiers in c) and d) identify both individual attributes and sets of attributes (see 9.4.7). The effective list in both cases is the set union of these. If an attribute appears in both the mandatory set and the optional set, it shall be considered mandatory.

Note 2 – The macro is used in defining selected object classes in Recommendation X.521.

Should all of the pieces of notation introduced by the macro and described in b), c), and d) above be empty, the resulting notation ("**OBJECT-CLASS**") can be used to denote any possible object class.

9.4.7 An attribute set is a set of attributes identified by an object identifier. The definition of an attribute set involves:

a) assigning an object identifier to the set;

b) listing the object identifiers of the attributes and other attribute sets whose members together form the set.

The following ASN.1 macro may (but need not) be used to define a set of attributes for use with the **OBJECT-CLASS** macro:

```
ATTRIBUTE-SET-MACRO   ::=

          BEGIN

          TYPE NOTATION ::= "CONTAINS" {"Attributes"}" | empty

          VALUE NOTATION ::= value(VALUE OBJECT IDENTIFIER)
          Attributes ::=
                    AttributeTerm | AttributeTerm "," Attributes

          AttributeTerm   ::=  Attribute | AttributeSet

          Attribute       ::=  value(ATTRIBUTE)

          AttributeSet    ::=  value(ATTRIBUTE-SET)

          END
```

The correspondence between the parts of the definition of an attribute set and the notation introduced by the macro is as follows:

a) the object identifier assigned to the attribute set is the value supplied in the value assignment of the macro;

b) the set of attributes comprising the attribute set is that formed by the set union of the attributes and sets of attributes identified by the **Attributes** production, i.e. following "**CONTAINS**".

Should the "**empty**" alternative of the notation be selected, the resulting notation ("**ATTRIBUTE- SET**") can be used to denote any possible attribute set.

9.4.8 The object classes previously mentioned are defined in § 9.4.8.1, § 9.4.8.2.

Note – These are partial definitions: the object identifiers are actually allocated for these object classes in Recommendation X.521 so as to provide a single point of allocation of these object identifiers in this series of Recommendations.

9.4.8.1 The object class "**Top**" is defined as follows:

```
Top ::=
        OBJECT-CLASS
                MUST CONTAIN {ObjectClass}
```

9.4.8.2 The object class "**Alias**" is defined as follows:

```
Alias ::=
        OBJECT-CLASS
                SUBCLASS OF top
                MUST CONTAIN {aliasedObjectName}
```

Note 1 – The object class "**Alias**" does not specify appropriate attribute types for the RDN of an alias entry. Administrative Authorities may specify subclasses of the class "**Alias**" which specify useful attribute types for RDNs of alias entries (see Recommendation X.521).

Note 2 – Entries of a subclass of the class "**Alias**" are alias entries.

9.5 *Attribute type definition*

9.5.1 The definition of an attribute type involves:

a) assigning an object identifier to the attribute type:

b) indicating or defining the attribute syntax for the attribute type;

c) indicating whether an attribute of this type may have only one or may have more than one value (recur).

9.5.2 The Directory ensures that the indicated attribute syntax is used for every attribute of this type. The Directory also ensures that attributes of this type will have one and only one value in entries if attributes of this type are defined to have only one value.

9.5.3 The following ASN.1 macro may (but need not) be used to define an attribute type:

```
ATTRIBUTE MACRO         ::=
BEGIN

TYPENOTATION            ::= AttributeSyntax Multivalued | empty
VALUENOTATION           ::= value (VALUE OBJECT IDENTIFIER)

AttributeSyntax         ::=
        "WITH ATTRIBUTE-SYNTAX" SyntaxChoice
Multivalued             ::= "SINGLE VALUE"
                             | "MULTIVALUE" | empty
SyntaxChoice            ::= value(ATTRIBUTE-SYNTAX)
                        Constraint | type MatchTypes
Constraint              ::= "("ConstraintAlternative")" | empty
ConstraintAlternative   ::= StringConstraint | IntegerConstraint
StringConstraint        ::= "SIZE" "("SizeConstraint")"
SizeConstraint          ::= SingleValue | Range
SingleValue             ::= value(INTEGER)
Range                   ::= value(INTEGER) ".." value
                            (INTEGER)
IntegerConstraint       ::= Range
MatchTypes              ::= "MATCHES FOR" Matches | empty
Matches                 ::= Match Matches | Match
Match                   ::= "EQUALITY" | "SUBSTRINGS" |
                                "ORDERING"
END
```

The correspondence between the parts of the definition, as listed in § 9.5.1, and the various pieces of the notation introduced by the macro, is as follows:

a) the object identifier assigned to the attribute type is the value supplied in the value assignment of the MACRO;

b) the attribute syntax for the attribute type is that identified by the **AttributeSyntax** production. This either points to a separately defined attribute syntax, or explicitly defines an attribute syntax by giving its ASN.1 type and matching rules (see § 9.6). If a separately identified attribute syntax is employed, a size constraint for underlying string types or a value range for an underlying integer type may optionally be indicated;

c) the attribute is single valued if the **MultiValued** production is "**SINGLE VALUE**", and may have one or more values if it is "**MULTI VALUE**" or empty.

Note – The macro is used in defining selected attribute types in Recommendation X.520.

Should the "**empty**" alternative of the type notation be selected, the resulting notation ("**ATTRIBUTE**") can be used to denote any possible attribute type.

9.5.4 The attribute types identified in § 7.3.3 which are known to and used by the Directory for its own purposes are defined as follows:

```
ObjectClass ::= ATTRIBUTE
        WITH ATTRIBUTE-SYNTAX objectIdentifierSyntax

AliasedObjectName ::= ATTRIBUTE
        WITH ATTRIBUTE-SYNTAX distinguishedNameSyntax
        SINGLE VALUE
```

Note 1 – These are partial definitions: the object identifiers are actually allocated for these attribute types in Recommendation X.520 so as to provide a single point of allocation of these object identifiers in this series of Recommendations.

Note 2 – The attribute syntaxes referred to in these definitions are themselves defined in § 9.6.5.

9.6 *Attribute syntax definition*

9.6.1 The definition of an attribute syntax involves:

a) optionally, assigning an object identifier to the attribute syntax;

b) indicating the data type, in ASN.1 of the attribute syntax;

c) defining appropriate rules for matching a presented value with a target attribute value held in the DIB. None, some, or all of the following matching rules may be defined for a particular attribute syntax:

   i) equality. Applicable to any attribute syntax. The presented value must conform to the data type of the attribute syntax;

   ii) substrings. Applicable to any attribute syntax with a string data type. The presented value must be a sequence ("**SEQUENCE OF**"), each of whose elements conforms to the data type;

   iii) ordering. Applicable to any attribute syntax for which a rule can be defined that will allow a presented value to be described as less than, equal to, or greater than a target value. The presented value must conform to the data type of the attribute syntax.

9.6.2 If no equality matching rule is defined, the Directory:

a) treats values as attributes of this attribute syntax as having type **ANY**, i.e. the Directory does not check that those values conform with the data type indicated for the attribute syntax;

b) will not attempt to match presented values against target values of such an attribute type.

Note – It follows that the Directory will not permit such an attribute to be used in a distinguished name, nor allow for a specific value to be modified.

9.6.3 If an equality matching rule is defined, the Directory:

a) treats values of attributes of this attribute syntax as having type **ANY DEFINED BY** the data type indicated for the attribute syntax;

b) will only match according to the matching rules defined for that attribute syntax;

c) will only match a presented value of a suitable data type as specified in § 9.6.1 c).

9.6.4 The following ASN.1 macro may, but need not, be used to define attribute syntaxes:

```
ATTRIBUTE-SYNTAX MACRO ::=
BEGIN

TYPE NOTATION ::=     Syntax
                      MatchTypes | empty

VALUE NOTATION ::=
      value (VALUE OBJECT IDENTIFIER)

Syntax ::= type
MatchTypes ::= "MATCHES FOR" Matches | empty
Matches ::= Match Matches | Match
Match ::= "EQUALITY" | "SUBSTRINGS" | "ORDERING"
END
```

The correspondence between the parts of the definition, as listed in § 9.6.1, and the various pieces of the notation introduced by the macro, is as follows:

a) the object identifier assigned to the attribute syntax is a value supplied in the value assignment of the macro;

b) the data type of the attribute syntax is that identified by the **Syntax** production, i.e. that following macro name;

c) the defined matching rules are equality, if "**EQUALITY**" appears in the **MatchTypes** production, substrings if "**SUBSTRINGS**" appears, and ordering if "**ORDERING**" appears. If the production is empty, then no matching rules are defined.

Should the "**empty**" alternative of the notation be selected, the resulting notation ("**ATTRIBUTE- SYNTAX**") can be used to denote any possible attribute syntax.

Note 1 – No support is provided in the macro for actually defining the matching rules themselves: this must be done by natural language or by other means.

Note 2 – The macro is used in defining selected attribute syntaxes in Recommendation X.520.

9.6.5 The attribute syntaxes used in § 9.5.4 are defined in § 9.6.5.1 and 9.6.5.2.

Note – These are partial definitions: the object identifiers are actually allocated for these attribute syntaxes in Recommendation X.520 so as to provide a single point of allocation of these object identifiers in this series of Recommendations.

9.6.5.1 **ObjectIdentifierSyntax** is defined as follows:

```
ObjectIdentifierSyntax ::=
      ATTRIBUTE-SYNTAX
            OBJECT IDENTIFIER
            MATCHES FOR EQUALITY
```

The matching rule for equality is inherent in the definition of the ASN.1 type object identifier.

9.6.5.2 **DistinguishedNameSyntax** is defined as follows:

```
DistinguishedNameSyntax ::=
    ATTRIBUTE-SYNTAX
        DistinguishedName
        MATCHES FOR EQUALITY
```

A presented distinguished name value is equal to a target distinguished name value if and only if all of the following are true:

a) the number of RDNs in each is the same;

b) corresponding RDNs have the same number of AVAs;

c) corresponding AVAs (i.e. those with identical attribute types) have attribute values which match for equality (in such a match, the attribute values take the same roles – i.e. as presented or target value – as the distinguished name which contains them does in the overall match).

SECTION 3 – Security model

10 Security

10.1 The directory exists in an environment where various authorities provide access to their fragment of the DIB. Such access shall be in conformance to the security policy (see Recommendation X.509) of the security domain in which the fragment of the DIB exists.

10.2 Two specific components of a security policy are addressed here:

a) the definition of an authorization policy;

b) the definition of an authentication policy.

10.3 The definition of authorization in the context of the Directory includes the methods to:

a) specify access rights;

b) enforce access rights (access control);

c) maintain access rights.

10.4 The definition of authentication in the context of the Directory includes the methods to verify:

a) the identity of DSAs and directory users;

b) the identity of the origin of received information at an access point.

The integrity of received information is a local matter and shall be in conformance to the security policy in force.

10.5 This Recommendation does not define a Security Policy.

10.6 Annex F describes guidelines for specifying access rights.

10.7 Recommendation X.509 defines authentication procedures. The DAP and DSP may provide strong authentication of the initiator by the signing of the request, data integrity of the request by signing of the request, strong authentification of the responder and data integrity of the result by signing the result. The DAP may provide simple authentication between a DUA and a DSA. The DSP may provide simple authentication between two DSAs.

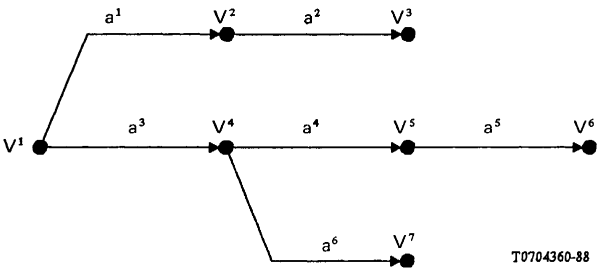
10.8 Administrative authorities of applications which make use of the Directory can use their own security policy. The directory can support applications by holding authentication information (e.g. distinguished names, passwords, certificates) about communication entities. This is further described in Recommendation X.509.

ANNEX A

(to Recommendation X.501)

The mathematics of trees

This Annex is not part of the standard.



A tree is a set of points, called *vertices*, and a set of directed lines, called *arcs*; each arc a leads from a vertex V to a vertex V′. For example, the tree in the Figure has seven vertices, labelled V^1 through V^7, and six arcs, labelled a^1 through a^6.

Two vertices V and V′ are said to be the *initial* and *final* vertices, respectively, of an arc a from V to V′. For example, V^2 and V^3 are the initial and final vertices, respectively, of arc a^2. Several different arcs may have the same initial vertex, but not the same final vertex. For example, arcs a^1 and a^3 have the same initial vertex, V^1, but no two arcs in the Figure have the same final vertex.

The vertex that is not the final vertex of any arc is often referred to as the root vertex, or even more informally as the "root" of the tree. For example, in the Figure, V^1 is the root.

A vertex that is not the initial vertex of any arc is often referred to informally as a *leaf vertex*, or even more informally, as a "leaf" of the tree graph. For example, vertices V^3, V^6, and V^7 are leaves.

An *oriented path* from a vertex V to a vertex V′ is a set of arcs (a^1, a^2, ..., a^n) ($n \geq 1$) such that V is the initial vertex of arc a^1, V′ is the final vertex of arc a^n, and the final vertex of arc a^k is also the initial vertex of arc a^{k+1} for $1 \leq k < n$. For example, the oriented path from vertex V^1 to vertex V^6 is the set of arcs (a^3, a^4, a^5). The term "path" should be understood to denote an oriented path from the root to a vertex.

ANNEX B

(to Recommendation X.501)

Object identifier usage

This Annex is part of the standard.

This Annex documents the upper reaches of the object identifier subtree in which all of the object identifiers assigned in this series of Recommendations reside. It does so by providing an ASN.1 module called "**UsefulDefinitions**" in which all non-leaf nodes in the subtree are assigned names.

```
UsefulDefinitions           {joint-iso-ccitt ds(5) modules(1)
                            usefulDefinitions(0)}
DEFINITIONS ::=
BEGIN

EXPORTS
    module, serviceElement, applicationContext, attributeType, attributeSyntax, objectClass,
    algorithm, abstractSyntax, attributeSet,

    usefulDefinitions, informationFramework, directoryAbstractService,
    directoryObjectIdentifiers, algorithmObjectIdentifiers, distributedOperations,
    protocolObjectIdentifiers, selectedAttributeTypes, selectedObjectClasses,
    authenticationFramework, upperBounds,
    dap,dsp

    id-ac, id-ase, id-as, id-ot, id-pt;

ds                    OBJECT IDENTIFIER ::=  {joint-iso-ccitt ds(5)}

-- categories of information object --

module                      OBJECT IDENTIFIER ::= {ds 1}
serviceElement              OBJECT IDENTIFIER ::= {ds 2}
applicationContext          OBJECT IDENTIFIER ::= {ds 3}
attributeType               OBJECT IDENTIFIER ::= {ds 4}
attributeSyntax             OBJECT IDENTIFIER ::= {ds 5}
objectClass                 OBJECT IDENTIFIER ::= {ds 6}
attributeSet                OBJECT IDENTIFIER ::= {ds 7}
algorithm                   OBJECT IDENTIFIER ::= {ds 8}
abstractSyntax              OBJECT IDENTIFIER ::= {ds 9}
object                      OBJECT IDENTIFIER ::= {ds 10}
port                        OBJECT IDENTIFIER ::= {ds 11}

-- modules --

usefulDefinitions                         OBJECT IDENTIFIER ::= {module 0}
informationFramework                      OBJECT IDENTIFIER ::= {module 1}
directoryAbstractService                  OBJECT IDENTIFIER ::= {module 2}
distributedOperations                     OBJECT IDENTIFIER ::= {module 3}
protocolObjectIdentifier                  OBJECT IDENTIFIER ::= {module 4}
selectedAttributeTypes                    OBJECT IDENTIFIER ::= {module 5}
selectedObjectClasses                     OBJECT IDENTIFIER ::= {module 6}
authenticationFramework                   OBJECT IDENTIFIER ::= {module 7}
algorithmObjectIdentifiers                OBJECT IDENTIFIER ::= {module 8}
directoryObjectIdentifiers                OBJECT IDENTIFIER ::= {module 9}
upperBounds                               OBJECT IDENTIFIER ::= {module 10}
dap                                       OBJECT IDENTIFIER ::= {module 11}
dsp                                       OBJECT IDENTIFIER ::= {module 12}
distributedDirectoryObjectIdentifiers     OBJECT IDENTIFIER ::= {module 13}

-- synonyms --

id-ac       OBJECT IDENTIFIER ::= applicationContext
id-ase      OBJECT IDENTIFIER ::= serviceElement
id-as       OBJECT IDENTIFIER ::= abstractSyntax
id-ot       OBJECT IDENTIFIER ::= object
id-pt       OBJECT IDENTIFIER ::= port

END
```

ANNEX C

(to Recommendation X.501)

Information framework in ASN.1

This Annex is part of the standard.

This Annex provides a summary of all of the ASN.1 type, value, and macro definitions contained in this Recommendation. The definitions form the ASN.1 module "**InformationFramework**".

```
InformationFramework     {joint-iso-ccitt ds(5) modules(1)
                          informationFramework(1)}
    DEFINITIONS ::=
    BEGIN

    EXPORTS
        Attribute, AttributeType, AttributeValue, AttributeValueAssertion,
        DistinguishedName, Name, RelativeDistinguishedName,
        OBJECT-CLASS,ATTRIBUTE,ATTRIBUTE-SET,ATTRIBUTE-SYNTAX,
        Top, Alias,
        ObjectClass, AliasedObjectName,
        ObjectIdentifierSyntax, DistinguishedNameSyntax;

    IMPORTS
        selectedAttributeTypes, selectedObjectClasses
            FROM      UsefulDefinitions {joint-iso-ccitt ds(5) modules(1)
                      usefulDefinitions(0)}
        top
            FROM SelectedObjectClasses selectedObjectClasses
        objectIdentifierSyntax, distinguishedNameSyntax, objectClass, aliasedObjectName
            FROM SelectedAttributeTypes selectedAttributeTypes;

    -- attribute data types --

    Attribute                 ::=  SEQUENCE{
                                       type          AttributeType
                                       values SET OF AttributeValue
                                       -- at least one value is required --}

    AttributeType             ::=  OBJECT IDENTIFIER

    AttributeValue            ::=  ANY

    AttributeValueAssertion        ::=  SEQUENCE {AttributeType, AttributeValue}

    -- naming data types --

    Name             ::=      CHOICE {-- only one possibility for now --
                                  RDNSequence}

    RDNSequence                    ::=     SEQUENCE OF
                                               RelativeDistinguishedName

    DistinguishedName              ::=     RDNSequence

    RelativeDistinguishedName      ::=     SET OF AttributeValueAssertion
```

-- macros --

```
OBJECT-CLASS MACRO     ::=
BEGIN

          TYPENOTATION        ::=     SubclassOf MandatoryAttributes
                                      OptionalAttributes
          VALUENOTATION       ::=     value (VALUE OBJECT IDENTIFIER)

          SubclassOf          ::=     "SUBCLASS OF" Subclasses | empty
          Subclasses          ::=     Subclass | Subclass "," Subclasses
          Subclass            ::=     value (OBJECT-CLASS)
          MandatoryAttributes ::=     "MUST CONTAIN {"Attributes"}" | empty
          OptionalAttributes  ::=     "MAY CONTAIN {"Attributes"}" | empty

          Attributes          ::=     AttributeTerm | AttributeTerm ","
                                      Attributes
          AttributeTerm       ::=     Attribute | AttributeSet
          Attribute         ::=       value(ATTRIBUTE)
          AttributeSet        ::=     value(ATTRIBUTE-SET)

END

ATTRIBUTE-SET-MACRO    ::=

          BEGIN

          TYPE NOTATION       ::=     "CONTAINS" "{Attributes"}" | empty

          VALUE NOTATION      ::=     value(VALUEOBJECTIDENTIFIER)

          Attributes          ::=     AttributeTerm | AttributeTerm "," Attributes

          AttributeTerm       ::=     Attribute | AttributeSet
          Attribute           ::=     value(ATTRIBUTE)
          AttributeSet        ::=     value(ATTRIBUTE-SET)

          END

ATTRIBUTE MACRO      ::=
BEGIN

TYPENOTATION         ::=  AttributeSyntax Multivalued | empty
VALUENOTATION        ::=  value(VALUE OBJECT IDENTIFIER)

AttributeSyntax      ::=  "WITH ATTRIBUTE-SYNTAX"  SyntaxChoice

Multivalued          ::=  "SINGLE VALUE" | "MULTI VALUE" | empty

SyntaxChoice         ::=  value(ATTRIBUTE-SYNTAX)
                          Constraint | type Match Types
Constraint           ::=  "("ConstraintAlternative")" | empty

ConstraintAlternative ::=  StringConstraint | IntegerConstraint

StringConstraint     ::=  "SIZE" "("SizeConstraint")"

SizeConstraint       ::=  SingleValue | Range

SingleValue          ::=  value(INTEGER)

Range                ::=  value(INTEGER) ".." value(INTEGER)
```

```
IntegerConstraint        ::=  Range

MatchTypes               ::=  "MATCHES FOR" Matches | empty

Matches                  ::=  Match Matches | Match

Match                    ::=  "EQUALITY" | "SUBSTRINGS" | "ORDERING"

END

ATTRIBUTE-SYNTAX MACRO ::=
BEGIN

      TYPENOTATION       ::=     Syntax MatchTypes | empty
      VALUENOTATION      ::=     value(VALUE OBJECT IDENTIFIER)

      Syntax             ::=     type

      MatchTypes         ::=     "MATCHES FOR "Matches | empty
      Matches            ::=     Match Matches | Match
      Match              ::=     "EQUALITY" | "SUBSTRINGS" | "ORDERING"

END
```

-- object classes --

```
Top       ::=       OBJECT-CLASS
                        MUST CONTAIN {objectClass}

Alias     ::=       OBJECT-CLASS
                        SUBCLASS OF top
                        MUST CONTAIN {aliasedObjectName}
```

-- attribute types --

```
ObjectClass        ::=  ATTRIBUTE
                             WITH ATTRIBUTE-SYNTAX objectIdentifierSyntax

AliasedObjectName ::= ATTRIBUTE
                             WITH ATTRIBUTE-SYNTAX distinguishedNameSyntax
                             SINGLE VALUE
```

-- attribute syntaxes --

```
ObjectIdentifierSyntax ::=
      ATTRIBUTE-SYNTAX
      OBJECT IDENTIFIER
      MATCHES FOR EQUALITY

DistinguishedNameSyntax ::=
      ATTRIBUTE-SYNTAX
      DistinguishedName
      MATCHES FOR EQUALITY

END
```

ANNEX D

(to Recommendation X.501)

Alphabetical index of definitions

This Annex is not part of the standard.

This Annex alphabetically lists all of the terms defined in this Recommendation together with a cross reference to the § in which they are defined.

A	access point	§ 5
	Administration Directory Management Domain	§ 5
	alias	§ 8
	alias entry	§ 6
	attribute	§ 7
	attribute type	§ 7
	attribute value	§ 7
	attribute value assertion	§ 7
D	the Directory	§ 5
	Directory entry	§ 6
	Directory Information Base (DIB)	§ 6
	Directory Information Tree (DIT)	§ 6
	Directory Management Domain (DMD)	§ 5
	Directory name	§ 8
	Directory schema	§ 9
	Directory System Agent (DSA)	§ 5
	Directory User Agent (DUA)	§ 5
	distinguished name	§ 8
	DIT Structure Rule	§ 9
E	entry	§ 6
I	immediate(ly) subordinate	§ 6
	immediate(ly) superior	§ 6
N	name	§ 8
	naming authority	§ 8
O	object (of interest)	§ 6
	object class	§ 6
	object entry	§ 6
P	Private Directory Management Domain	§ 5
	purported name	§ 8
R	relative distinguished name	§ 8
S	subordinate	§ 6
	superior	§ 6

ANNEX E

(to Recommendation X.501)

Name design criteria

This Annex is not part of the standard.

The information framework is very general, and allows for arbitrary variety of entries and attributes within the DIT. Since, as defined there, names are closely related to paths through the DIT, this means that arbitrary variety in names is possible. This section suggests criteria to be considered in the design of names. The appropriate criteria have been used in the design of the recommended name forms which are to be found in Recommendation X.521. It is suggested that the criteria also be used, where appropriate, in designing the names for objects to which the recommended name forms do not apply.

Presently, only one criterion is addressed: that of user-friendliness.

Note – Not all names need to be user-friendly.

E.1 *User-friendliness*

Names with which human beings must deal directly should be user-friendly. A user-friendly name is one that takes the human user's point of view, not the computer's. It is one that is easy for people to deduce, remember, and understand, rather than one that is easy for computers to interpret.

The goal of user-friendliness can be stated somewhat more precisely in terms of the following two principles:

– A human being usually should be able to correctly guess an object's user-friendly name on the basis of information about the object that he naturally possesses. For example, one should be able to guess a business person's name given only the information about her casually acquired through normal business association.

– When an object's name is ambiguously specified, the Directory should recognize the fact rather than conclude that the name identifies one particular object. For example, where two people have the same last name, the last name alone should be considered inadequate identification of either party.

The following subgoals follow from the goal of user-friendliness:

a) Names should not artificially remove natural ambiguities. For example, if two people share the last name "Jones", neither should be required to answer to "WJones" or "Jones2". Instead, the naming convention should provide a user-friendly means of discriminating between the entities. For example, it might require first name and middle initial in addition to last name.

b) Names should admit common abbreviations and common variations in spelling. For example, if one is employed by the Conway Steel Corporation and the name of one's employer figures in one's name, any of the names "Conway Steel Corporation", "Conway Steel Corp.", "Conway Steel", and "CSC" should suffice to identify the organization in question.

c) In certain cases, alias names can be used to direct the search for a particular entry, in order to be more user-friendly, or to reduce the scope of a search. The following example demonstrates the use of an alias name for such a purpose: as shown in Figure E-1/X.501, the branch office in Osaka can also be identified with the name {C = Japan, L = Osaka, O = ABC, OU = Osaka-branch}.

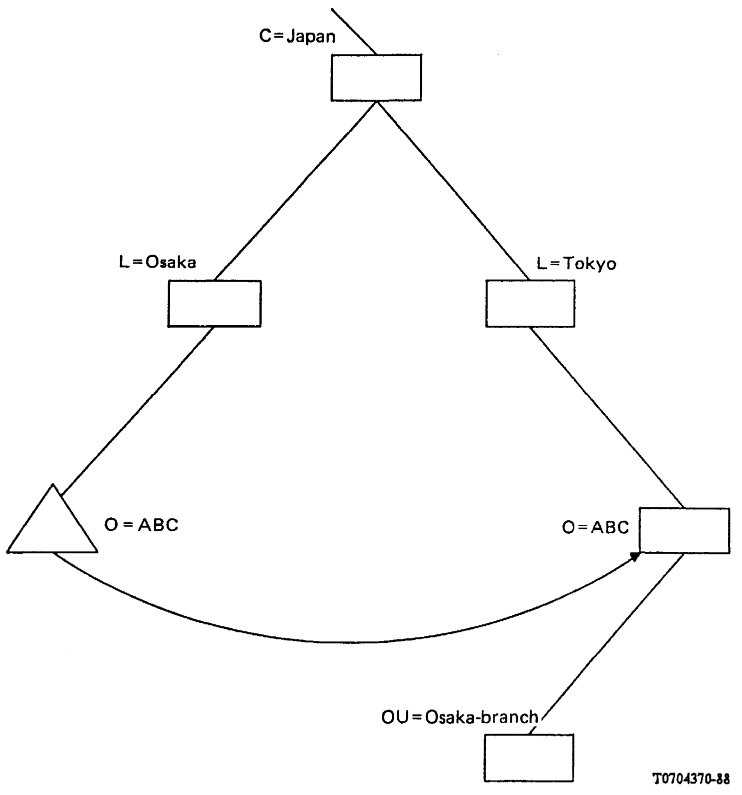


FIGURE E-1/X.501

Aliasing example

d) If names are multi-part, both the number of mandatory parts and the number of optional parts should be relatively small and thus easy to remember.

e) If names are multi-part, the precise order in which those parts appear should generally be immaterial.

f) User-friendly names should not involve computer addresses.

ANNEX F

(to Recommendation X.501)

Access control

This Annex is not part of the standard.

F.1 *Introduction*

Directory users are granted access to the information in the DIB on the basis of their access control rights in accordance with the access control policy in force protecting that information.

Access Control is left as a local matter in this series of Recommendations. However, it is recognized that implementations will need to introduce means of controlling access and that future versions of this series of Recommendations are likely to define standardized means of creating, maintaining and applying access control information. This Annex describes the principles underlying access control, and outlines two possible approaches to access control.

F.2 *Principles*

The two principles that will guide the establishment of procedures for managing access control are:

a) there must be means of protecting information in the Directory from unauthorized detection, examination, and modification, including protecting the DIT from unauthorized modification;

b) the information required to determine a user's rights to perform a given operation must be available to the DSA(s) involved in performing the operation in order to avoid further remote operations solely to determine these rights.

F.3 *Protected items*

These levels of protection are presently identified:

a) protection of an entire subtree of the DIT;

b) protection of an individual entry;

c) protection of an entire attribute within an entry;

d) protection of selected instances of attribute values.

F.4 *Access categories*

A need for at least five categories of access is envisaged. If access is not granted to a protected item in any category, then the directory in so far as is possible responds as though their protected item did not exist at all.

The categories of access are shown in Table F-1/X.501. The items column denotes whether the item that can be so protected is an entry (E), an attribute (A) or both (EA).

TABLE F-1/X.501

Access categories

Category	Items	Description
detect	A	Allows the protected item to be detected.
compare	A	Allows a presented value to be compared to the protected item.
read	A	Allows the protected item to be read.
modify	A	Allows the protected item to be updated.
add/delete	EA	Allows the creation and deletion of new components (attributes or attribute values) within the protected item.
naming	E	Allows the modification of the Relative Distinguished Name of, and the creation and deletion of, entries which are immediately subordinate to the protected entry.

F.5 *Determination of access rights*

One scheme for managing access control associates with every protected item, either explicitely or implicitely, a list of access rights. Each item in such a list pairs a set of users with a set of access categories.

Determining if a user is in one (or more) of the noted sets must be possible from the information supplied with the request – either from the authenticated identity and credentials of the user as supplied in BIND, or from information carried in the operation argument.

There at least two possibilities:

a) The sets are described in terms of the distinguished names of the users they identify either the distinguished name of the user or the distinguished name of a superior with a flag specifying that the entire subtree is included.

b) The sets give only a capability, and implicitly include all users having that capability. This scheme requires that such users' capability be available locally or else carried in the BIND or operation argument. The latter may require an extension to the currently defined protocols.

Printed in Switzerland

Geneva, 2008