



This electronic version (PDF) was scanned by the International Telecommunication Union (ITU) Library & Archives Service from an original paper document in the ITU Library & Archives collections.

La présente version électronique (PDF) a été numérisée par le Service de la bibliothèque et des archives de l'Union internationale des télécommunications (UIT) à partir d'un document papier original des collections de ce service.

Esta versión electrónica (PDF) ha sido escaneada por el Servicio de Biblioteca y Archivos de la Unión Internacional de Telecomunicaciones (UIT) a partir de un documento impreso original de las colecciones del Servicio de Biblioteca y Archivos de la UIT.

(ITU) للاتصالات الدولي الاتحاد في والمحفوظات المكتبة قسم أجراه الضوئي بالمسح تصوير نتاج (PDF) الإلكترونية النسخة هذه والمحفوظات المكتبة قسم في المتوفرة الوثائق ضمن أصلية ورقية وثيقة من نقلًا.

此电子版（PDF版本）由国际电信联盟（ITU）图书馆和档案室利用存于该处的纸质文件扫描提供。

Настоящий электронный вариант (PDF) был подготовлен в библиотечно-архивной службе Международного союза электросвязи путем сканирования исходного документа в бумажной форме из библиотечно-архивной службы МСЭ.



INTERNATIONAL TELECOMMUNICATION UNION

CCITT

THE INTERNATIONAL
TELEGRAPH AND TELEPHONE
CONSULTATIVE COMMITTEE

BLUE BOOK

VOLUME VIII – FASCICLE VIII.8

**DATA COMMUNICATION NETWORKS
DIRECTORY**

RECOMMENDATIONS X.500-X.521



IXTH PLENARY ASSEMBLY
MELBOURNE, 14-25 NOVEMBER 1988

Geneva 1989



INTERNATIONAL TELECOMMUNICATION UNION

CCITT

THE INTERNATIONAL
TELEGRAPH AND TELEPHONE
CONSULTATIVE COMMITTEE

BLUE BOOK

VOLUME VIII – FASCICLE VIII.8

DATA COMMUNICATION NETWORKS DIRECTORY

RECOMMENDATIONS X.500-X.521



IXTH PLENARY ASSEMBLY
MELBOURNE, 14-25 NOVEMBER 1988

Geneva 1989

ISBN 92-61-03731-3

**CONTENTS OF THE CCITT BOOK
APPLICABLE AFTER THE NINTH PLENARY ASSEMBLY (1988)**

BLUE BOOK

Volume I

- FASCICLE I.1 – Minutes and reports of the Plenary Assembly.
List of Study Groups and Questions under study.
- FASCICLE I.2 – Opinions and Resolutions.
Recommendations on the organization and working procedures of CCITT (Series A).
- FASCICLE I.3 – Terms and definitions. Abbreviations and acronyms. Recommendations on means of expression (Series B) and General telecommunications statistics (Series C).
- FASCICLE I.4 – Index of Blue Book.

Volume II

- FASCICLE II.1 – General tariff principles – Charging and accounting in international telecommunications services. Series D Recommendations (Study Group III).
- FASCICLE II.2 – Telephone network and ISDN – Operation, numbering, routing and mobile service. Recommendations E.100-E.333 (Study Group II).
- FASCICLE II.3 – Telephone network and ISDN – Quality of service, network management and traffic engineering. Recommendations E.401-E.880 (Study Group II).
- FASCICLE II.4 – Telegraph and mobile services – Operations and quality of service. Recommendations F.1-F.140 (Study Group I).
- FASCICLE II.5 – Telematic, data transmission and teleconference services – Operations and quality of service. Recommendations F.160-F.353, F.600, F.601, F.710-F.730 (Study Group I).
- FASCICLE II.6 – Message handling and directory services – Operations and definition of service. Recommendations F.400-F.422, F.500 (Study Group I).

Volume III

- FASCICLE III.1 – General characteristics of international telephone connections and circuits. Recommendations G.100-G.181 (Study Groups XII and XV).
- FASCICLE III.2 – International analogue carrier systems. Recommendations G.211-G.544 (Study Group XV).
- FASCICLE III.3 – Transmission media – Characteristics. Recommendations G.601-G.654 (Study Group XV).
- FASCICLE III.4 – General aspects of digital transmission systems; terminal equipments. Recommendations G.700-G.795 (Study Groups XV and XVIII).
- FASCICLE III.5 – Digital networks, digital sections and digital line systems. Recommendations G.801-G.961 (Study Groups XV and XVIII).

- FASCICLE III.6 – Line transmission of non-telephone signals. Transmission of sound-programme and television signals. Series H and J Recommendations (Study Group XV).
- FASCICLE III.7 – Integrated Services Digital Network (ISDN) – General structure and service capabilities. Recommendations I.110-I.257 (Study Group XVIII).
- FASCICLE III.8 – Integrated Services Digital Network (ISDN) – Overall network aspects and functions, ISDN user-network interfaces. Recommendations I.310-I.470 (Study Group XVIII).
- FASCICLE III.9 – Integrated Services Digital Network (ISDN) – Internetwork interfaces and maintenance principles. Recommendations I.500-I.605 (Study Group XVIII).

Volume IV

- FASCICLE IV.1 – General maintenance principles: maintenance of international transmission systems and telephone circuits. Recommendations M.10-M.782 (Study Group IV).
- FASCICLE IV.2 – Maintenance of international telegraph, phototelegraph and leased circuits. Maintenance of the international public telephone network. Maintenance of maritime satellite and data transmission systems. Recommendations M.800-M.1375 (Study Group IV).
- FASCICLE IV.3 – Maintenance of international sound-programme and television transmission circuits. Series N Recommendations (Study Group IV).
- FASCICLE IV.4 – Specifications for measuring equipment. Series O Recommendations (Study Group IV).

- Volume V** – Telephone transmission quality. Series P Recommendations (Study Group XII).

Volume VI

- FASCICLE VI.1 – General Recommendations on telephone switching and signalling. Functions and information flows for services in the ISDN. Supplements. Recommendations Q.1-Q.118 *bis* (Study Group XI).
- FASCICLE VI.2 – Specifications of Signalling Systems Nos. 4 and 5. Recommendations Q.120-Q.180 (Study Group XI).
- FASCICLE VI.3 – Specifications of Signalling System No. 6. Recommendations Q.251-Q.300 (Study Group XI).
- FASCICLE VI.4 – Specifications of Signalling Systems R1 and R2. Recommendations Q.310-Q.490 (Study Group XI).
- FASCICLE VI.5 – Digital local, transit, combined and international exchanges in integrated digital networks and mixed analogue-digital networks. Supplements. Recommendations Q.500-Q.554 (Study Group XI).
- FASCICLE VI.6 – Interworking of signalling systems. Recommendations Q.601-Q.699 (Study Group XI).
- FASCICLE VI.7 – Specifications of Signalling System No. 7. Recommendations Q.700-Q.716 (Study Group XI).
- FASCICLE VI.8 – Specifications of Signalling System No. 7. Recommendations Q.721-Q.766 (Study Group XI).
- FASCICLE VI.9 – Specifications of Signalling System No. 7. Recommendations Q.771-Q.795 (Study Group XI).
- FASCICLE VI.10 – Digital subscriber signalling system No. 1 (DSS 1), data link layer. Recommendations Q.920-Q.921 (Study Group XI).

- FASCICLE VI.11 – Digital subscriber signalling system No. 1 (DSS 1), network layer, user-network management. Recommendations Q.930-Q.940 (Study Group XI).
- FASCICLE VI.12 – Public land mobile network. Interworking with ISDN and PSTN. Recommendations Q.1000-Q.1032 (Study Group XI).
- FASCICLE VI.13 – Public land mobile network. Mobile application part and interfaces. Recommendations Q.1051-Q.1063 (Study Group XI).
- FASCICLE VI.14 – Interworking with satellite mobile systems. Recommendations Q.1100-Q.1152 (Study Group XI).

Volume VII

- FASCICLE VII.1 – Telegraph transmission. Series R Recommendations. Telegraph services terminal equipment. Series S Recommendations (Study Group IX).
- FASCICLE VII.2 – Telegraph switching. Series U Recommendations (Study Group IX).
- FASCICLE VII.3 – Terminal equipment and protocols for telematic services. Recommendations T.0-T.63 (Study Group VIII).
- FASCICLE VII.4 – Conformance testing procedures for the Teletex Recommendations. Recommendation T.64 (Study Group VIII).
- FASCICLE VII.5 – Terminal equipment and protocols for telematic services. Recommendations T.65-T.101, T.150-T.390 (Study Group VIII).
- FASCICLE VII.6 – Terminal equipment and protocols for telematic services. Recommendations T.400-T.418 (Study Group VIII).
- FASCICLE VII.7 – Terminal equipment and protocols for telematic services. Recommendations T.431-T.564 (Study Group VIII).

Volume VIII

- FASCICLE VIII.1 – Data communication over the telephone network. Series V Recommendations (Study Group XVII).
- FASCICLE VIII.2 – Data communication networks: services and facilities, interfaces. Recommendations X.1-X.32 (Study Group VII).
- FASCICLE VIII.3 – Data communication networks: transmission, signalling and switching, network aspects, maintenance and administrative arrangements. Recommendations X.40-X.181 (Study Group VII).
- FASCICLE VIII.4 – Data communication networks: Open Systems Interconnection (OSI) – Model and notation, service definition. Recommendations X.200-X.219 (Study Group VII).
- FASCICLE VIII.5 – Data communication networks: Open Systems Interconnection (OSI) – Protocol specifications, conformance testing. Recommendations X.220-X.290 (Study Group VII).
- FASCICLE VIII.6 – Data communication networks: interworking between networks, mobile data transmission systems, internetwork management. Recommendations X.300-X.370 (Study Group VII).
- FASCICLE VIII.7 – Data communication networks: message handling systems. Recommendations X.400-X.420 (Study Group VII).
- FASCICLE VIII.8 – Data communication networks: directory. Recommendations X.500-X.521 (Study Group VII).

Volume IX

- Protection against interference. Series K Recommendations (Study Group V). Construction, installation and protection of cable and other elements of outside plant. Series L Recommendations (Study Group VI).

Volume X

- FASCICLE X.1 – Functional Specification and Description Language (SDL). Criteria for using Formal Description Techniques (FDTs). Recommendation Z.100 and Annexes A, B, C and E, Recommendation Z.110 (Study Group X).
 - FASCICLE X.2 – Annex D to Recommendation Z.100: SDL user guidelines (Study Group X).
 - FASCICLE X.3 – Annex F.1 to Recommendation Z.100: SDL formal definition. Introduction (Study Group X).
 - FASCICLE X.4 – Annex F.2 to Recommendation Z.100: SDL formal definition. Static semantics (Study Group X).
 - FASCICLE X.5 – Annex F.3 to Recommendation Z.100: SDL formal definition. Dynamic semantics (Study Group X).
 - FASCICLE X.6 – CCITT High Level Language (CHILL). Recommendation Z.200 (Study Group X).
 - FASCICLE X.7 – Man-Machine Language (MML). Recommendations Z.301-Z.341 (Study Group X).
-

CONTENTS OF FASCICLE VIII.8 TO THE BLUE BOOK

Rec. No.		Page
X.500	The Directory - Overview of Concepts, Models and Services	3
X.501	The Directory-Models	19
X.509	The Directory - Authentication framework	48
X.511	The Directory - Abstract Service Definition	82
X.518	The Directory - Procedures for Distributed Operation	116
X.519	The Directory - Protocol Specifications	174
X.520	The Directory - Selected Attribute Types	189
X.521	The Directory - Selected Object Classes	212

PRELIMINARY NOTES

1 The questions entrusted to each Study Group for the Study Period 1989-1992 can be found in Contribution No. 1 to that Study Group.

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

3 Except where otherwise indicated, the status of annexes and appendices attached to the Series X Recommendations should be interpreted as follows:

- an *annex* to a Recommendation forms an integral part of the Recommendation;
- an *appendix* to a Recommendation does not form part of the Recommendation and only provides some complementary explanation of information.

4 The Series X Recommendations contained in this Fascicle were jointly developed in collaboration with the ISO/IEC. Cross-references between these Recommendations and the corresponding ISO/IEC standards are given in the table below.

CCITT Recommendation	ISO/IEC Standard
X.500	ISO 9594-1, Information processing systems - Open Systems Interconnection - The Directory - Part 1: Overview of concepts, models and service ^{a)}
X.501	ISO 9594-2, Information processing systems - Open Systems Interconnection - The Directory - Part 2: Models ^{a)}
X.509	ISO 9594-8, Information processing systems - Open Systems Interconnection - The Directory - Part 8: Authentication framework ^{a)}
X.511	ISO 9594-3, Information processing systems - Open Systems Interconnection - The Directory - Part 3: Abstract service definition ^{a)}
X.518	ISO 9594-4, Information processing systems - Open Systems Interconnection - The Directory - Part 4: Procedures for distributed operations ^{a)}
X.519	ISO 9594-5, Information processing systems - Open Systems Interconnection - The Directory - Part 5: Protocol specifications ^{a)}
X.520	ISO 9594-6, Information processing systems - Open Systems Interconnection - The Directory - Part 6: Selected attribute types ^{a)}
X.521	ISO 9594-7, Information processing systems - Open Systems Interconnection - The Directory - Part 7: Selected object classes ^{a)}

^{a)} Presently at the stage of Draft International Standard (DIS).

FASCICLE VIII.8

Recommendations X.500 to X.521

**DATA COMMUNICATION NETWORKS:
DIRECTORY**

PAGE INTENTIONALLY LEFT BLANK

PAGE LAISSEE EN BLANC INTENTIONNELLEMENT

THE DIRECTORY - OVERVIEW OF CONCEPTS, MODELS AND SERVICES ¹⁾

(Melbourne, 1988)

CONTENTS

0	<i>Introduction</i>
1	<i>Scope and field of application</i>
2	<i>References</i>
3	<i>Definitions</i>
	3.1 OSI reference model definitions
	3.2 Basic directory definitions
	3.3 Directory model definitions
	3.4 Distributed operation definitions
4	<i>Abbreviations</i>
5	<i>Overview of the directory</i>
6	<i>The directory information base (DIB)</i>
7	<i>The directory service</i>
	7.1 Introduction
	7.2 Service qualification
	7.3 Directory interrogation
	7.4 Directory modification
	7.5 Other outcomes
8	<i>The distributed directory</i>
	8.1 Functional model
	8.2 Organizational model
	8.3 Operation of the model
9	<i>Directory protocols</i>
 <i>Annex A - Applying the directory</i>	
	A.1 The directory environment
	A.2 Directory service characteristics
	A.3 Patterns of use of the directory
	A.4 Generic applications

¹⁾ Recommendation X.500 and ISO 9594-1, The Directory - Overview of Concepts, Models and Services, 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. The set of all 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 introduces and models the concepts of the Directory and of the DIB and overviews the services and capabilities which they provide. Other Recommendations make use of these models in defining the abstract service provided by the Directory, and in specifying the protocols through which this service can be obtained or propagated.

1 Scope and field of application

1.1 The Directory provides the directory capabilities required by OSI applications, OSI management processes, other OSI layer entities, and telecommunication services. Among the capabilities which it provides are those of "user-friendly naming" whereby objects can be referred to by names which are suitable for citing by human users (though not all objects need have user-friendly names); and "name-to-address mapping" which allows the binding between objects and their locations to be dynamic. The latter capability allows OSI networks, for example, to be "self-configuring" in the sense that addition, removal and the changes of object location do not affect OSI network operation.

1.2 The Directory is not intended to be a general-purpose data base system, although it may be built on such systems. It is assumed, for instance, that, as is typical with communications directories, there is a considerably higher frequency of "queries" than of updates. The rate of updates is expected to be governed by the dynamics of people and organizations, rather than, for example, the dynamics of networks. There is also no need for instantaneous global commitment of updates: transient conditions where both old and new versions of the same information are available, are quite acceptable.

1.3 It is a characteristic of the Directory that, except as a consequence of differing access rights or unpropagated updates, the results of directory queries will not be dependent on the identity or location of the enquirer. This characteristic renders the Directory unsuitable for some telecommunications applications, for example some types of routing.

2 References

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

Recommendation X.208 - Open Systems Interconnection - Specification of Abstract Syntax Notation One (ASN.1).

Recommendation X.501 - The Directory - Models.

Recommendation X.509 - The Directory - Authentication framework.

Recommendation X.511 - The Directory - Abstract Service Definition.

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

Recommendation X.519 - The Directory - Protocol Specifications.

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

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

Recommendation X.219 - Remote Operations - Model, Notation and Service Definition.

Recommendation X.229 - Remote Operations - Protocol Specification.

3 Definitions

The definitions contained in this § make use of the abbreviations defined in § 4.

3.1 *OSI reference model definitions*

This Recommendation is based on the concepts developed in Recommendation X.200, and makes use of the following terms therein defined:

- a) *application-entity*;
- b) *Application Layer*;
- c) *application process*;
- d) *application protocol data unit*;
- e) *application service element*.

3.2 *Basic directory definitions*

- a) The *Directory*: a collection of open systems cooperating to provide directory services;
- b) *Directory Information Base (DIB)*: the set of information managed by the Directory;
- c) (*Directory*) *user*: the end user of the Directory, i.e. the entity or person which accesses the Directory.

3.3 *Directory model definitions*

This Recommendation makes use of the following terms defined in Recommendation X.501.

- a) *Administration Directory Management Domain*;
- b) *alias*;
- c) *attribute*;
- d) *attribute type*;
- e) *attribute value*;
- f) *Directory Information Tree (DIT)*;
- g) *Directory Management Domain (DMD)*;
- h) *Directory System Agent (DSA)*;
- i) *Directory User Agent (DUA)*;
- j) *distinguished name*;
- k) *entry*;
- l) *name*;
- m) *object (of interest)*;
- n) *Private Directory Management Domain*;
- o) *relative distinguished name*;
- p) *root*;
- q) *schema*;
- r) *subordinate object*;
- s) *superior entry*;

- t) *superior object*;
- u) *tree*.

3.4 Distributed operation definitions

This Recommendation makes use of the following terms defined in Recommendation X.518:

- a) *chaining*;
- b) *multicasting*;
- c) *referral*.

4 Abbreviations

ADDMD	Administration Directory Management Domain
DAP	Directory Access Protocol
DIB	Directory Information Base
DIT	Directory Information Tree
DMD	Directory Management Domain
DSA	Directory System Agent
DSP	Directory System Protocol
DUA	Directory User Agent
OSI	Open Systems Interconnection
PRDMD	Private Directory Management Domain
RDN	Relative Distinguished Name

5 Overview of the Directory

5.1 The *Directory* is a collection of open systems which cooperate to hold a logical data base of information about a set of objects in the real world. The *users* of the Directory, including people and computer programs, can read or modify the information, or parts of it, subject to having permission to do so. Each user is represented in accessing the Directory by a Directory User Agent (DUA), which is considered to be an application-process. These concepts are illustrated in Figure 1/X.500.

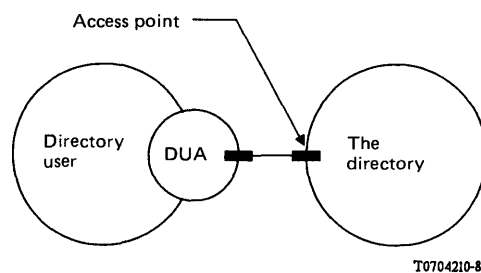


FIGURE 1/X.500

Access to the Directory

Note - This series of Recommendations refers to the Directory in the singular, and reflects the intention to create, through a single, unified, name space, one logical directory composed of many systems and serving many applications. Whether or not these systems choose to interwork will depend on the needs of the applications they support. Applications dealing with non-intersecting worlds of objects, may have no such need. The single name space facilitates later interworking should the needs change.

5.2 The information held in the Directory is collectively known as the *Directory Information Base* (DIB). Clause 6 of this Recommendation overviews its structure.

5.3 The Directory provides a well-defined set of access capabilities, known as the abstract service of the Directory, to its users. This service, which is overviewed in § 7 of this Recommendation provides a simple modification and retrieval capability. This can be built on with local DUA functions to provide the capabilities required by the end-users.

5.4 It is likely that the Directory will be distributed, perhaps widely distributed, both along functional and organizational lines. § 8 overviews the corresponding models of the Directory. These have been developed in order to provide a framework for the cooperation of the various components to provide an integrated whole.

5.5 The provision and consumption of the Directory services requires that the users (actually the DUAs) and the various functional components of the Directory should cooperate with one another. In many cases this will require cooperation between application processes in different open systems, which in turn requires standardized application protocols, overviewed in § 9, to govern this cooperation.

5.6 The Directory has been designed so as to support multiple applications, drawn from a wide range of possibilities. The nature of the application supported will govern which objects are listed in the Directory, which users will access the information, and which kinds of access they will carry out. Applications may be very specific, such as the provision of distribution lists for electronic mail, or generic, such as the "inter-personal communications directory" application. The Directory provides the opportunity to exploit commonalities among the applications:

- a single object may be relevant to more than one application; perhaps even the same piece of information about the same object may be so relevant.

To support this, a number of object classes and attribute types are defined, which will be useful across a range of applications. These definitions are contained in Recommendations X.520 and X.521:

- certain patterns of use of the Directory will be common across a range of applications: this area is overviewed further in Annex A.

6 The Directory Information Base (DIB)

Note - The DIB, and its structure, are defined in Recommendation X.501.

6.1 The DIB is made up of information about objects. It is composed of (*directory*) *entries*, each of which consists of a collection of information on one object. Each entry is made up of *attributes*, each with a type and one or more values. The types of attribute which are present in a particular entry are dependent on the *class* of object which the entry describes.

6.2 The entries of the DIB are arranged in the form of a tree, the Directory Information Tree (DIT) where the vertices represent the entries. Entries higher in the tree (nearer the root) will often represent objects such as countries or organizations while entries lower in the tree will represent people or application processes.

Note - The services defined in this Recommendation operate only on a tree-structured DIT. This Recommendation does not preclude the existence in the future of other structures (as the need arises).

6.3 Every entry has a *distinguished name*, which uniquely and unambiguously identifies the entry. These properties of the distinguished name are derived from the tree structure of the information. The distinguished name of an entry is made up of the distinguished name of its superior entry, together with specially nominated attribute values (the *distinguished* values) from the entry.

6.4 Some of the entries at the leaves of the tree are alias entries, while all other entries are object entries. Alias entries point to object entries, and provide the basis for alternative names for the corresponding objects.

6.5 The Directory enforces a set of rules to ensure that the DIB remains well-formed in the face of modifications over time. These rules, known as the *Directory schema*, prevent entries having the wrong types of attributes for its object class, attribute values being of the wrong form for the attribute type, and even entries having subordinate entries of the wrong class.

6.6 Figure 2/X.500 illustrates the above concepts of the DIT and its components.

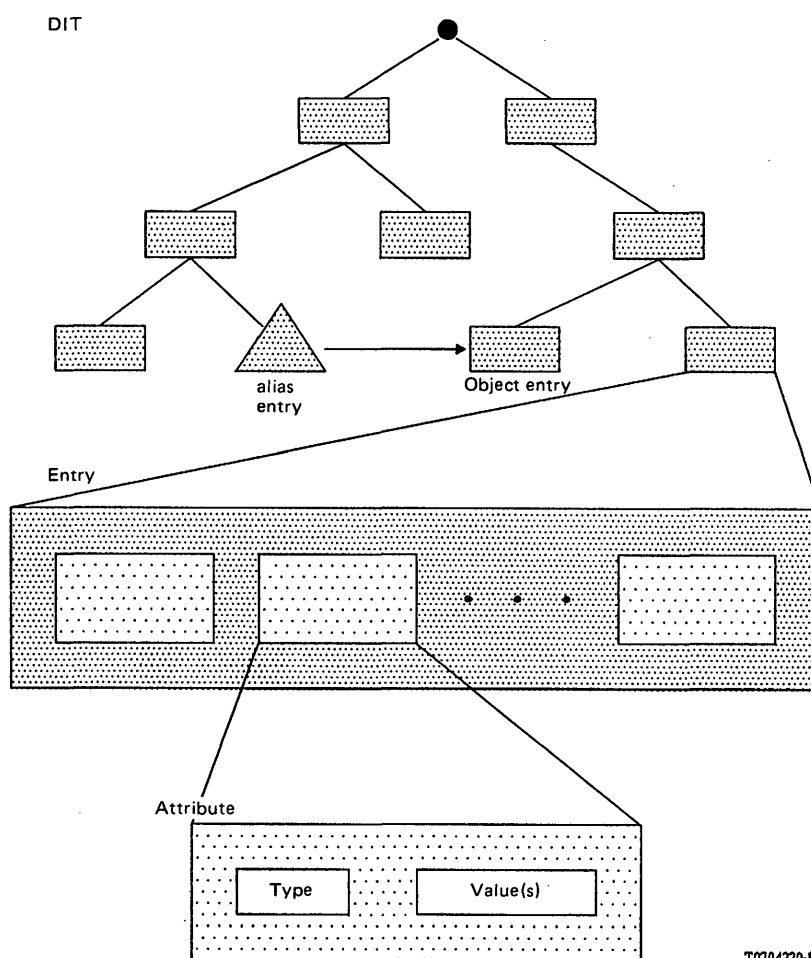


FIGURE 2/X.500

Structure of the DIT and of entries

6.7 Figure 3/X.500 gives a hypothetical example of a DIT. The tree provides examples of some of the types of attributes used to identify different objects. For example the name:

{C = GB, L = Winslow, O = Graphic Services, CN = Laser Printer}

identifies the application entity "Laser Printer" which has in its distinguished name the geographical attribute of Locality. The residential person John Jones, whose name is GB

{C = GB, L = Winslow, CN = John Jones}

has the same geographical attribute in his distinguished name.

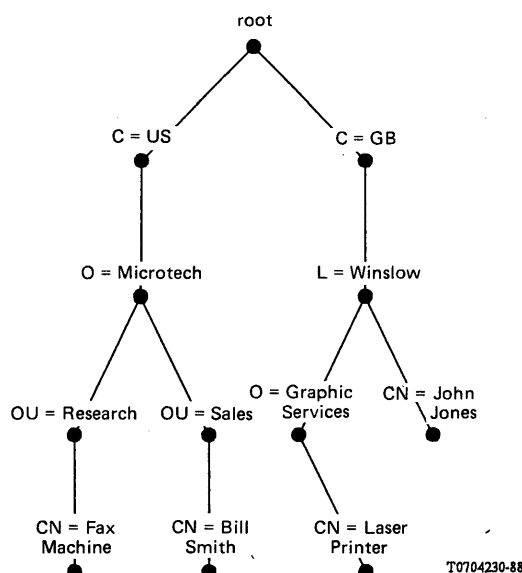


FIGURE 3/X.500

A Hypothetical directory information tree

6.8 The growth and form of the DIT, the definition of the Directory schema, and the selection of distinguished names for entries as they are added, is the responsibility of various authorities, whose hierarchical relationship is reflected in the shape of the tree. The authorities must ensure, for example, that all of the entries in their jurisdiction have unambiguous distinguished names, by carefully managing the attribute types and values which appear in those names. Responsibility is passed down the tree from superior to subordinate authorities, with control being exercised by means of the schema.

7 The Directory service

Note - The definition of the abstract service of the Directory can be found in Recommendation X.511.

7.1 Introduction

7.1.1 This § provides an overview of the service provided to users, as represented by their DUAs, by the Directory. All services are provided by the Directory in response to requests from DUAs. There are requests which allow interrogation of the Directory, as described in § 7.3, and those for modification, as described in § 7.4. In addition, requests for service can be qualified, as described in § 7.2. The Directory always reports the outcome of each request that is made of it. The form of the normal outcome is specific to the request, and is evident from the description of the request. Most abnormal outcomes are common to several requests. The possibilities are described in § 7.5.

7.1.2 A number of aspects of the eventual directory service are not presently provided by the standards specified in this series of Recommendations. The corresponding capabilities will, therefore, need to be provided as a local function until such time as a standardized solution is available. These capabilities include:

- addition and deletion of arbitrary entries, thus allowing a distributed Directory to be created;
- the management of access control (i.e. granting or withdrawing permission for a particular user to carry out a particular access on a particular piece of information);
- the management of the Directory schema;
- the management of knowledge information;
- the replication of parts of the DIB.

Note - This list is not necessarily exhaustive.

7.1.3 The Directory ensures that changes to the DIB, whether the result of a Directory service request, or by some other (local) means, result in a DIB which continues to obey the rules of the Directory schema.

7.1.4 A User and the Directory are bound together for a period of time at an access point to the Directory. At the time of binding, the User and the Directory optionally verify each other's identity.

7.2 *Service qualification*

7.2.1 *Service controls*

A number of controls can be applied to the various service requests, primarily to allow the user to impose limits on the use of resources which the Directory must not surpass. Controls are provided on, among other things: the amount of time, the size of the results, the scope of search the interaction modes, and on the priority of the request.

7.2.2 *Security parameters*

Each request may be accompanied by information in support of security mechanisms for protecting the Directory information. Such information may include the user's request for various kinds of protection; a digital signature of the request, together with information to assist the correct party to verify the signature.

7.2.3 *Filters*

A number of requests whose outcome involves information from or concerning a number of entries, may carry with them a filter. A filter expresses one or more conditions that an entry must satisfy in order to be returned as part of the outcome. This allows the set of entries returned to be reduced to only those relevant.

7.3 *Directory interrogation*

7.3.1 *Read*

A read request is aimed at a particular entry, and causes the values of some or all of the attributes of that entry to be returned. Where only some attributes are to be returned, the DUA supplies the list of attribute types of interest.

7.3.2 *Compare*

A compare request is aimed at a particular attribute of a particular entry, and causes the Directory to check whether a supplied value matches a value of that attribute.

Note - For example, this can be used to carry out password checking, where the password, held in the Directory, might be inaccessible for read, but accessible for compare.

7.3.3 *List*

A list request causes the Directory to return the list of immediate subordinates of a particular named entry in the DIT.

7.3.4 *Search*

A search request causes the Directory to return information from all of the entries within a certain portion of the DIT which satisfy some filter. The information returned from each entry consists of some or all of the attributes of that entry, as with read.

7.3.5 *Abandon*

An abandon request, as applied to an outstanding interrogation request, informs the Directory that the originator of the request is no longer interested in the request being carried out. The Directory may, for example, cease processing the request, and may discard any results so far achieved.

7.4 *Directory modification*

7.4.1 *Add entry*

An add entry request causes a new leaf entry (either an object entry, or an alias entry) to be added to the DIT.

Note - In its present form this service is intended to be used to add entries which will remain as leaves, such as entries for people or application entities, rather than to add whole subtrees by repeated applications of this service. It is envisaged that the service will be enhanced in the future to cater to the more general case.

7.4.2 *Remove entry*

A remove entry request causes a leaf entry to be removed from the DIT.

Note - As with add entry, this service is presently intended for operation on "true leaf" entries, and will be enhanced in the future for the general case.

7.4.3 *Modify entry*

A modify entry request causes the Directory to execute a sequence of changes to a particular entry. Either all of the changes are made, or none of them, and the DIB is always left in a state consistent with the schema. The changes allowed include the addition, removal, or replacement of attributes or attribute values.

7.4.4 *Modify relative distinguished name*

A modify relative distinguished name (RDN) request causes the relative distinguished name of a leaf entry (either an object entry or an alias entry) in the DIT to be modified by the nomination of different distinguished attribute values.

7.5 *Other outcomes*

7.5.1 *Errors*

Any service may fail, for example because of problems with the user supplied parameters, in which case an error is reported. Information is returned with the error, where possible, to assist in correcting the problem. However, in general, only the first error encountered by the Directory is reported. Besides the above-mentioned example of problems with the parameters supplied by the user (particularly invalid names for entries or invalid attribute types), errors may arise from violations of security policy, schema rules, and service controls.

7.5.2 *Referrals*

A service may fail because the particular access point to which the DUA is bound is not the most suitable for carrying out the request, e.g. because the information affected by the request is (logically) far away from the access point. In this case the Directory may return a referral, which suggests an alternative access point at which the DUA can make its request.

Note - The Directory and the DUA may each have a preference as to whether referrals are used, or whether the requests are *chained* (see § 8.3.3.2). The DUA can express its preference by means of service controls. The Directory makes the final decision as to which approach is used.

8 *The distributed Directory*

Note - the models of the directory are defined in Recommendation X.501 while the procedures for the operation of the distributed Directory are specified in Recommendation X.518.

8.1 *Functional model*

The functional model of the Directory is shown in Figure 4/X.500.

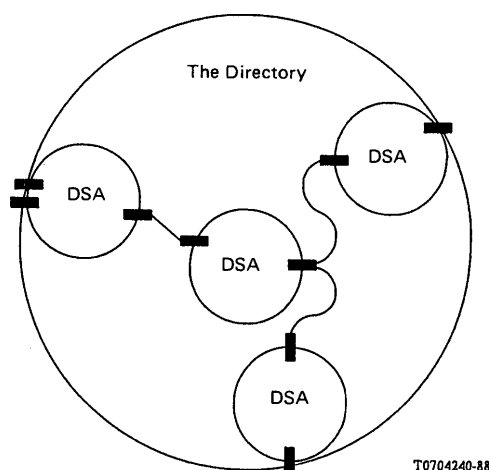


FIGURE 4/X.500

Functional model of the Directory

A *Directory System Agent* (DSA) is an OSI application process which is part of the Directory and whose role is to provide access to the DIB to DUAs and/or other DSAs. A DSA may use information stored in its local data base or interact with other DSAs to carry out requests. Alternatively, the DSA may direct a requestor to another DSA which can help carry out the request. Local data bases are entirely implementation dependent.

8.2 *Organizational model*

8.2.1 A set of one or more DSAs and zero or more DUAs managed by a single organization may form a *Directory Management Domain* (DMD). The organization concerned may or may not elect to make use of this series of Recommendations to govern the communications among the functional components within the DMD.

8.2.2 Subsequent Recommendations specify certain aspects of the behaviour of DSAs. For this purpose, a group of DSAs within one DMD may, at the option of the organization which manages the DMD, behave as a single DSA.

8.2.3 A DMD may be an *Administration DMD* (ADDMD), or a *Private DMD* (PRDMD), depending on whether or not it is being operated by a public telecommunications organization.

Note - 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.

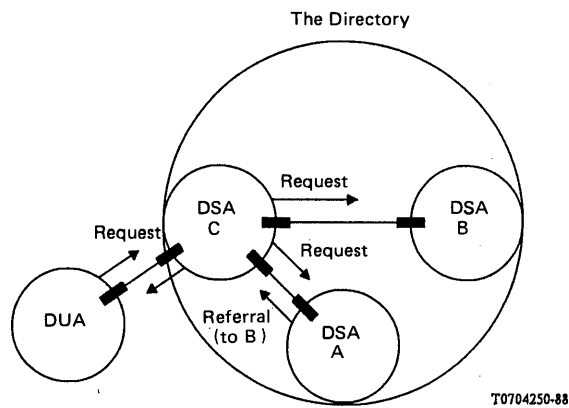
8.3 *Operation of the model*

8.3.1 The DUA interacts with the Directory by communicating with one or more DSAs. A DUA need not be bound to any particular DSA. It may interact directly with various DSAs to make requests. For some administrative reasons, it may not always be possible to interact directly with the DSA which needs to carry out the request, e.g. to return some directory information. It is also possible that the DUA can access the Directory through a single DSA. For this purpose, DSAs will need to interact with each other.

8.3.2 The DSA is concerned with carrying out the requests of DUAs and with obtaining the information where it does not have the necessary information. It may take the responsibility to obtain the information by interacting with other DSAs on behalf of the DUA.

8.3.3 A number of cases of request handling have been identified, as illustrated in Figures 5-7/X.500, and described below.

8.3.3.1 In Figure 5a/X.500, the DSA C receives a referral from DSA A and is responsible for either conveying the request to the DSA B (named in the referral from DSA A) or conveying the referral back to the originating DUA.



Note - If DSA C returns the referral to the DUA, the "request (to B)" will not occur. Similarly, if DSA C conveys the request to DSA B, it will not return a referral to the DUA.

FIGURE 5a/X.500

Referrals

In Figure 5b/X.500, the DUA receives the referral from DSA C and is responsible for reissuing the request directly to DSA A (named in the referral from DSA C).

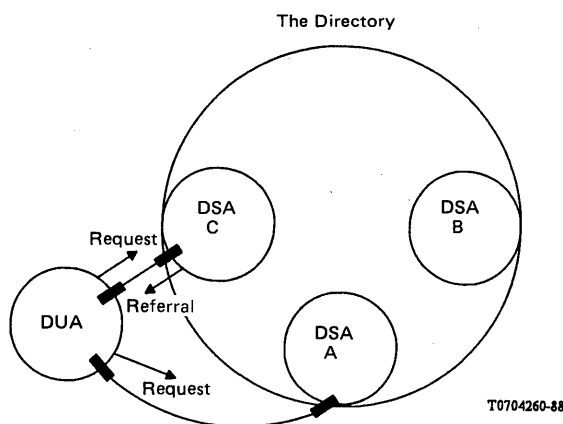


FIGURE 5b/X.500

Referrals

8.3.3.2 Figure 6/X.500 shows DSA chaining, whereby the request can be passed through several DSAs before the response is returned.

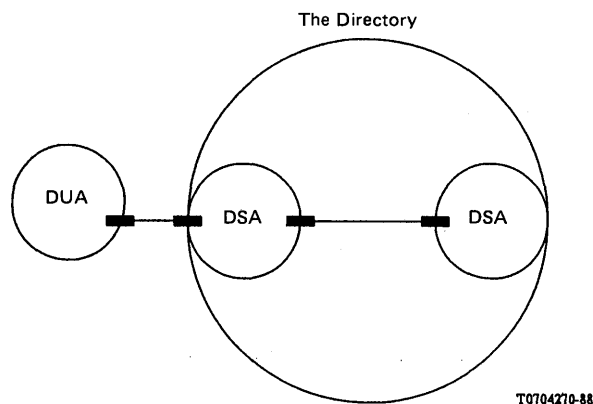


FIGURE 6/X.500

Chaining

8.3.3.3 Figure 7/X.500 shows multicasting, where the DSA associated with the DUA carries out the request by forwarding it to two or more other DSAs, the request to each DSA being identical.

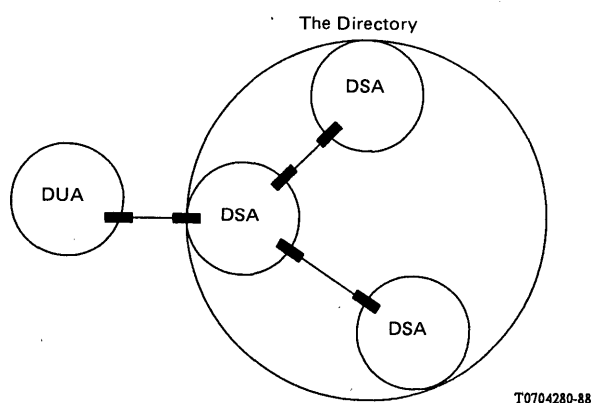


FIGURE 7/X.500

Multicasting

8.3.4 All of the approaches have their merits. For example, the approach in Figure 5/X.500 may be used where it is desirable to offload the burden from the local DSA. In other circumstances, a hybrid approach that combines a more elaborate set of functional interactions may be needed to satisfy the initiator's request, as illustrated in Figure 8/X.500.

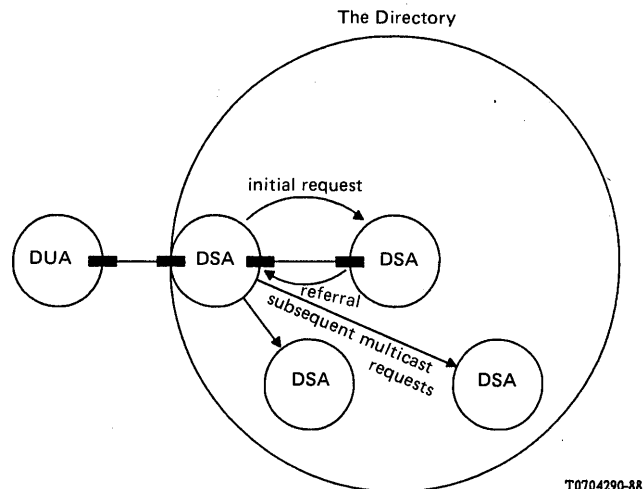


FIGURE 8/X.500

Mixed modes hybrid approach

9 Directory protocols

Note - The OSI application layer protocols defined to allow DUAs and DSAs in different open systems to cooperate are specified in Recommendation X.519.

9.1 There are two Directory protocols:

- the Directory Access Protocol (DAP), which defines the exchange of requests and outcomes between a DUA and a DSA;
- the Directory System Protocol (DSP), which defines the exchange of requests and outcomes between two DSAs.

9.2 Each protocol is defined by an application context, each containing a set of protocol elements. For example, the DAP contains protocol elements associated with interrogating and modifying the Directory.

9.3 Each application context is made up of application service elements. These application service elements are defined to use the Remote Operations Service (ROS) of Recommendation X.219 to structure and support their interactions. Thus the DAP and DSP are defined as sets of remote operations and errors using the ROS notation.

ANNEX A

(to Recommendation X.500)

Applying the Directory

This annex is not an integral part of this Recommendation.

A.1 The Directory environment

Note - In this §, the term "network" is used with its general meaning to denote the set of interlinked systems and processes relevant to any telecommunications service, not only one which relates to the OSI network layer.

The Directory exists in and provides services in the following environment:

- a) many telecommunications networks will be on a large scale, and will constantly undergo change:
 - 1) objects of various kinds will enter and leave the network without warning and may do so either singly or in groups;
 - 2) the connectivity of the objects (particularly network nodes) will change, owing to the addition or removal of paths between them;
 - 3) various characteristics of the objects, such as their addresses, availability, and physical locations, may change at any time;
- b) although the overall rate of changes is high, the useful lifetime of any particular object is not short. An object will typically be involved in communications much more frequently than it will change its address, availability, physical location, etc.;
- c) the objects involved in current telecommunications services are typically identified by numbers or other strings of symbols, selected for their ease of allocation or processing but not for ease of use by human beings.

A.2 *Directory service characteristics*

The need for directory capabilities arises from:

- a) the desire to isolate (as far as possible) the user of the network from the frequent changes to it. This can be accomplished by placing a "level of indirection" between the users and the objects with which they deal. This involves the users referring to objects by name, rather than by, for example, address. The Directory provides the necessary mapping service;
- b) the desire to provide a more "user-friendly" view of the network. For example, the use of aliases, the provision of "yellow-pages" (see A.3.5) etc., helps to relieve the burden of finding and using network information.

The Directory allows users to obtain a variety of information about the network, and provides for the maintenance, distribution and security of that information.

A.3 *Patterns of use of the directory*

Note - This subclause is concerned only with Directory retrieval: it is assumed that the Directory modification services are used solely to maintain the DIB in the form necessary for the application over time.

A.3.1 *Introduction*

The Directory service is defined in these standards in terms of particular requests that a DUA can make and the parameters of them. An application designer is likely, however, to think in more goal-oriented terms when considering the information retrieval requirements of the Directory in that application. Accordingly, this clause describes a number of high-level patterns of use of the Directory service that are likely to be relevant to many applications.

A.3.2 *Look-up*

The straight Directory look-up is likely to be the most frequent type of query of the Directory. It involves the DUA supplying the distinguished name of an object, together with an attribute type. The Directory will return any value(s) corresponding that attribute type. This is a generalization of the classic directory function, which is obtained when the attribute type requested corresponds to a particular type of address. Attribute types for various kinds of address are standardized, including OSI PSAP address, Message Handling O/R address, and telephone and telex numbers.

Look-up is supported by the read service, which also provides the following further generalizations:

- look-up can be based upon names other than the distinguished name of the object, e.g. aliases;
- the values from a number of attribute types can be requested with a single request: the extreme case being that the values of all attributes in the entry are to be returned.

A.3.3 User-friendly naming

Names can be given to objects in such a way as to maximize the chances that these names can be predicted (or perhaps remembered) by human users. Names which have this property would typically be made up of attributes which are somehow inherent to the object, rather than being fabricated for the purpose. The name of an object will be common among all of the applications which refer to it.

A.3.4 Browsing

In many human-oriented uses of the Directory, it may not be possible for the user (or DUA) to directly quote a name, user-friendly or otherwise, for the object about which information is sought. However, perhaps the user will "know it when he sees it". The browsing capability will allow a human user to wander about the DIB looking for the appropriate entries.

Browsing is accomplished by combinations of the list and search services, possibly in conjunction with read (although the search service includes the capability of read).

A.3.5 "Yellow Pages"

There are a variety of ways to provide a "Yellow Pages" type capability. The simplest is based upon filtering, using assertions about particular attributes whose values are the categories (e.g. the "Business Category" attribute type defined in Recommendation X.520). This approach does not require any special information being set-up in the DIT, except to ensure that the requisite attributes are present. However, in the general case, it may be expensive to search where there is a large population because filtering requires the generation of the universal set which is to be filtered.

An alternative approach is possible, based upon the setting up of special subtrees, whose naming structures are designed especially for "Yellow Pages" type searching. Shown in Figure A-1/X.500 is an example of a "Yellow Pages" subtree populated by alias entries only. In reality, the entries within the "Yellow Pages" subtrees may be a mixture of object and alias entries, so long as there exists only one object entry for each object stored in the Directory.

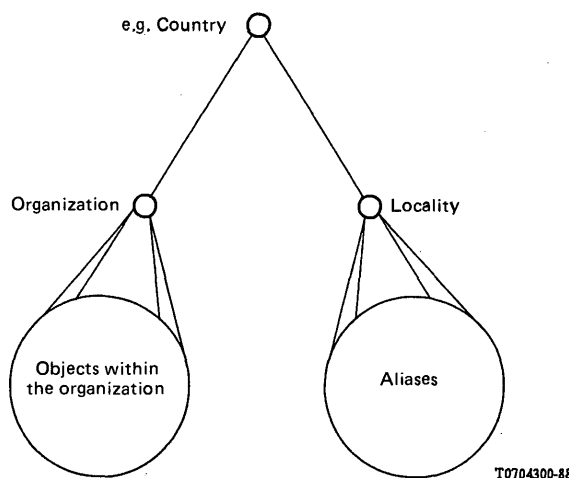


FIGURE A-1/X.500

An approach to "Yellow Pages"

A.3.6 Groups

A group is a set whose membership can change over time by explicit addition and removal of members. The group is an object, as are its members. The Directory can be requested to:

- indicate whether or not a particular object is a member of a group;
- list the membership of a group.

Groups are supported by having the entry for the group contain a multiple valued "Member" attribute (such an attribute type is defined in Recommendation X.520). The two capabilities mentioned can then be carried out by means of compare and read respectively.

A member of a group could itself be a group, if this is meaningful for the application. However, the necessary recursive verification and expansion services would have to be created by the DUA out of the non-recursive versions provided.

A.3.7 Authentication

Many applications require the objects taking part to offer some proof of their identity before they are permitted to carry out some action. The Directory provides support for this authentication process. (As a separate matter, the Directory itself requires its users to authenticate themselves, so as to support access control).

The more straightforward approach to authentication, called "simple authentication", is based upon the Directory holding a "User Password" attribute in the entry for any user that wishes to be able to authenticate itself to a Service. At the request of the Service, the Directory will confirm or deny that a particular value supplied is actually the user's password. This avoids the user needing a different password for every Service. In cases where the exchange of passwords in a local environment that uses simple authentication is considered to be inappropriate, the Directory optionally provides means to protect those passwords against replay or misuse by a one way function.

The more complex approach, called "strong authentication" is based upon public key cryptography, where the Directory acts as a repository of users' public encryption keys, suitably protected against tampering. The steps that users can take to obtain each other's public keys from the Directory, and then to authenticate with each other using them, are described in detail in Recommendation X.509.

A.4 Generic applications

A.4.1 Introduction

There are a number of generic applications which can be imagined as implicitly supported by the Directory: applications which are not specific to any particular telecommunications service. Two such applications are described herein: the inter-personal communications directory and the inter-system communications directory (for OSI).

Note - Authentication, described in the previous subclause as an "access pattern" could alternatively be thought of as a generic Directory application.

A.4.2 Inter-personal communications

The intent of this application is to provide humans or their agents with information on how to communicate with other humans, or groups thereof.

The following classes of objects are certainly involved: person, organizational role and group. Many other classes are involved too, perhaps in a less direct way, including: country, organization, organizational unit.

The attribute types concerned, other than those used in naming, are generally the addressing attributes. Typically the entry for a particular person will have the addresses corresponding to each of the communication methods by which that person can be reached, selected from an open-ended list which includes at least the following: telephony, electronic mail, telex, ISDN, physical delivery (e.g. the postal system), facsimile. In some cases, such as electronic mail, the entry will have some additional information such as the types of information which the user's equipment can handle. If authentication is to be supported, then User Password and/or Credentials will be needed.

The naming schemes used for the various object classes should be user-friendly, with aliases being set up as appropriate to provide alternative names, provide continuity after a name change, etc.

The following access patterns will be manifested in this application: look-up, user-friendly naming, browsing, "Yellow Pages", and groups. To varying degrees, authentication will also be used.

A.4.3 *Inter-system communications (for OSI)*

According to the OSI Reference Model, two Directory functions are required in OSI, one, operating in the application layer, which maps application-titles onto presentation-addresses, and one, in the network layer, which maps NSAP-addresses onto SNPA-addresses (SNPA = Subnetwork Point of Attachment).

Note - For the remainder of this §, only the application layer case is dealt with.

This function is carried out by consulting the Directory if the information required to accomplish the mapping is not conveniently available by other means.

The users are application-entities and the object classes of interest are also application-entities, or subclasses thereof.

The main attribute type concerned, other than those used for naming, is the presentation-address. Other attribute types, not viewed as necessary for the directory function itself, could support verifying or finding out the application entity type, or the lists of application contexts, abstract syntaxes, etc. supported. The authentication-related attribute types could also be relevant.

The main access pattern to be manifested will be look-up.

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*

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

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

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 Annex D 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 Figure 1/X.501.

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

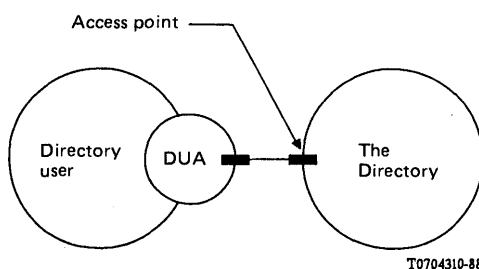


FIGURE 1/X.501

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 X.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 X.519.

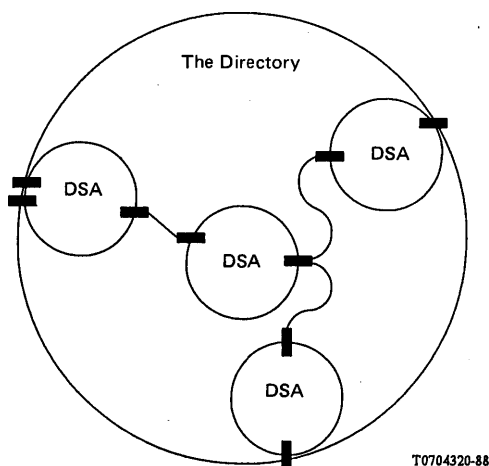


FIGURE 2/X.501

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 § 8.1 f)] 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 Figure 3/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 Annex A.

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 vertex B 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*.

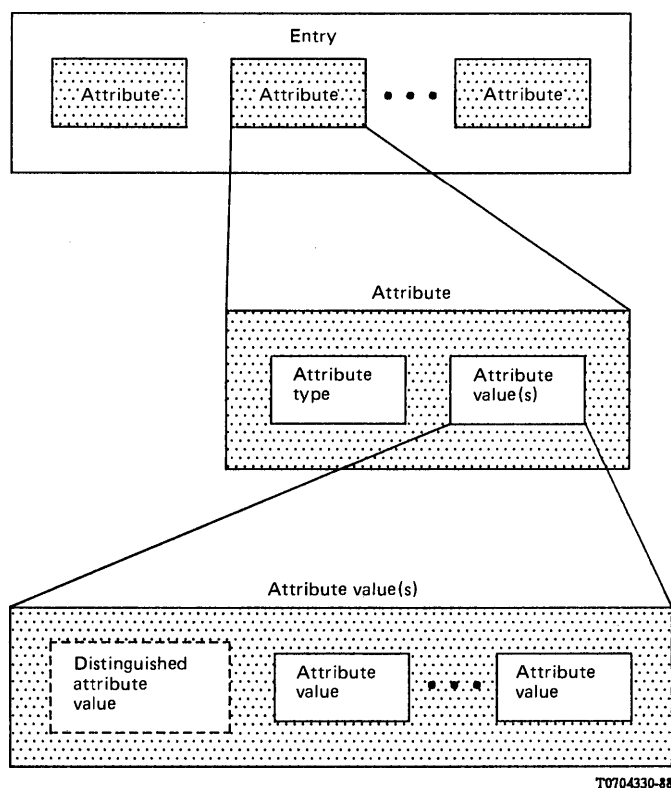


FIGURE 3/X.501

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.

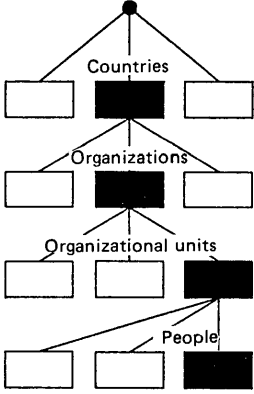
Note 1 - It is preferable that the distinguished names of objects which humans have to deal with be user-friendly.

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 Figure 4/X.501.

Root	RDN	Distinguished name
		{ }
	C = GB	{ C = GB }
	O = Telecom	{ C = GB, O = Telecom }
	(OU = Sales, L = Ipswich)	{ C = GB, O = Telecom, (OU = Sales, L = Ipswich) }
	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.

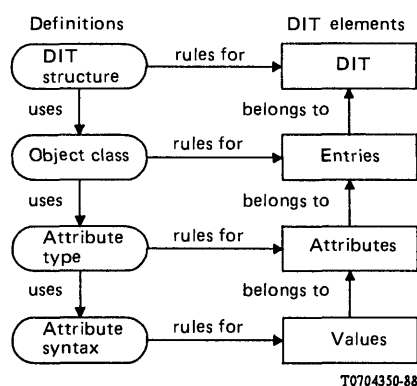


FIGURE 5/X.501

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"
 | "MULTI VALUE" | 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 authentication 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.

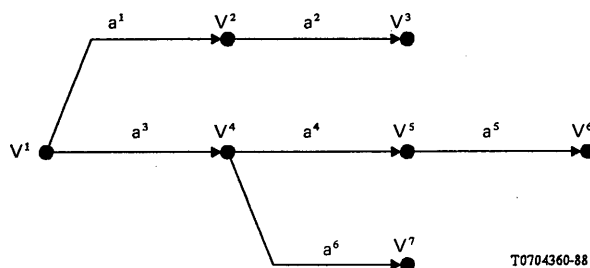
10.8 Administrative authorities of applications which make use of the Directory can use their own security policy. The directory can support applications by holding authentication information (e.g. distinguished names, passwords, certificates) about communication entities. This is further described in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

ANNEX A

(to Recommendation X.501)

The mathematics of trees

This Annex is not part of the standard.



T0704360-88

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, \dots, 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-se	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}.

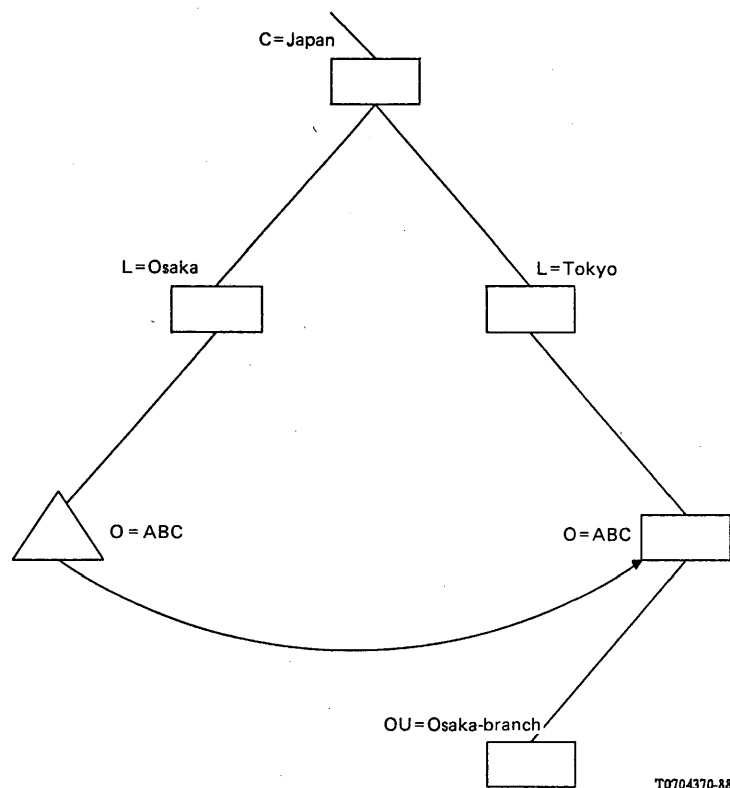


FIGURE E-1/X.501

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 explicitly or implicitly, 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 are 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.

Recommendation X.509

THE DIRECTORY - AUTHENTICATION FRAMEWORK ¹⁾

(Melbourne, 1988)

CONTENTS

- 0 *Introduction*
- 1 *Scope and field of application*
- 2 *References*
- 3 *Definitions*
- 4 *Notation and abbreviations*

SECTION 1 - *Simple authentication*

- 5 *Simple authentication procedure*

SECTION 2 - *Strong authentication*

- 6 *Basis of strong authentication*
- 7 *Obtaining a user's public key*

¹⁾ Recommendations X.509 and ISO 9594-8, Information Processing Systems - Open Systems Interconnection - The Directory - Authentication Framework, were developed in close collaboration and are technically aligned.

- 8 *Digital signatures*
- 9 *Strong authentication procedures*
- 10 *Management of keys and certificates*

Annex A - Security requirements

Annex B - An introduction to public key cryptography

Annex C - The RSA public key cryptosystem

Annex D - Hash functions

Annex E - Threats protected against by the strong authentication method

Annex F - Data confidentiality

Annex G - Authentication framework in ASN.1

Annex H - Reference definition of algorithm object identifiers

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. The set of all 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 OSI 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 Many applications have requirements for security to protect against threats to the communication of information. Some commonly known threats, together with the security services and mechanisms that can be used to protect against them, are briefly described in Annex A. Virtually all security services are dependent upon the identities of the communicating parties being reliably known, i.e. authentication.

0.4 This Recommendation defines a framework for the provision of authentication services by the Directory to its users. These users include the Directory itself, as well as other applications and services. The Directory can usefully be involved in meeting their needs for authentication and other security services because it is a natural place from which communicating parties can obtain authentication information of each other: knowledge which is the basis of authentication. The Directory is a natural place because it holds other information which is required for communication and obtained prior to communication taking place. Obtaining the authentication information of a potential communication partner from the Directory is, with this approach, similar to obtaining an address. Owing to the wide reach of the Directory for communications purposes, it is expected that this authentication framework will be widely used by a range of applications.

1 Scope and field of application

1.1 This Recommendation:

- specifies the form of authentication information held by the Directory;
- describes how authentication information may be obtained from the Directory;
- states the assumptions made about how this authentication information is formed and placed in the Directory;
- defines three ways in which applications may use this authentication information to perform authentication and describes how other security services may be supported by authentication.

1.2 This Recommendation describes two levels of authentication: simple authentication, using a password as a verification of claimed identity; and strong authentication, involving credentials formed using cryptographic techniques. While simple authentication offers some limited protection against unauthorized access, only strong authentication should be used as the basis for providing secure services. It is not intended to establish this as a general framework for authentication, but it can be of general use for applications which consider these techniques adequate.

1.3 Authentication (and other security services) can only be provided within the context of a defined security policy. It is a matter for users of an application to define their own security policy which may be constrained by the services provided by a standard.

1.4 It is a matter for standards defining applications which use the authentication framework to specify the protocol exchanges which need to be performed in order to achieve authentication based upon the authentication information obtained from the Directory. The protocol used by applications to obtain credentials from the Directory is the Directory Access Protocol (DAP), specified in Recommendation X.519.

1.5 The strong authentication method specified in this Recommendation is based upon public-key cryptosystems. It is a major advantage of such systems that user certificates may be held within the Directory as attributes, and may be freely communicated within the Directory System and obtained by users of the Directory in the same manner as other Directory information. The user certificates are assumed to be formed by "off-line" means, and placed in the Directory by their creator. The generation of user certificates is performed by some off-line Certification Authority which is completely separate from the DSAs in the Directory. In particular, no special requirements are placed upon Directory providers to store or communicate user certificates in a secure manner.

A brief introduction to public-key cryptography can be found in Annex B.

1.6 In general, the authentication framework is not dependent on the use of a particular cryptographic algorithm, provided it has the properties described in § 6.1. Potentially a number of different algorithms may be used. However, two users wishing to authenticate must support the same cryptographic algorithm for authentication to be performed correctly. Thus, within the context of a set of related applications, the choice of a single algorithm will serve to maximize the community of users able to authenticate and communicate securely. One example of a public key cryptographic algorithm can be found in Annex C.

1.7 Similarly, two users wishing to authenticate must support the same hash function (see § 3.3 f)) (used in forming credentials and authentication tokens). Again, in principle, a number of alternative hash functions could be used, at the cost of narrowing the communities of users able to authenticate. A brief introduction to hash functions together with one example hash function can be found in Annex D.

2 References

2.1 ISO 7498-2: Information Processing Systems - Open Systems Interconnection - Security Architecture.

3 Definitions

3.1 This Recommendation makes use of the following general security-related terms defined in Part 2 of the OSI Reference Model on Security:

- a) *asymmetric* (encipherment);
- b) *authentication exchange*;
- c) *authentication information*;
- d) *confidentiality*;
- e) *credentials*;
- f) *cryptography*;
- g) *data origin authentication*;
- h) *decipherment*;
- i) *encipherment*;
- j) *key*;
- k) *password*;
- l) *peer-entity authentication*;
- m) *symmetric* (encipherment).

3.2 The following terms used in this Recommendation are defined in Recommendation X.501:

- a) *attribute*;
- b) *Directory Information Base*;
- c) *Directory Information Tree*;
- d) *distinguished name*;
- e) *entry*;
- f) *object*;
- g) *root*.

3.3 The following specific terms are defined and used in this Recommendation:

- a) *authentication token (token)*: information conveyed during a strong authentication exchange, which can be used to authenticate its sender;
- b) *user certificate (certificate)*: the public keys of a user, together with some other information, rendered unforgeable by encipherment with the secret key of the certification authority which issued it;
- c) *certification authority*:^(CA) an authority trusted by one or more users to create and assign certificates. Optionally the certification authority may create the user's keys;
- d) *certification path*: an ordered sequence of certificates of objects in the DIT which, together with the public key of the initial object in the path, can be processed to obtain that of the final object in the path;
- e) *cryptographic system, cryptosystem*: a collection of transformations from plain text into ciphertext and vice versa, the particular transformation(s) to be used being selected by keys. The transformations are normally defined by a mathematical algorithm.
- f) *hash function*: a (mathematical) function which maps values from a large (possibly very large) domain into a smaller range. A "good" hash function is such that the results of applying the function to a (large) set of values in the domain will be evenly distributed (and apparently at random) over the range;

- g) *one-way function*: a (mathematical) function f which is easy to compute, but which for a general value y in the range, it is computationally difficult to find a value x in the domain such that $f(x) = y$. There may be a few values y for which finding x is not computationally difficult;
- h) *public key*: (in a public key cryptosystem) that key of a user's key pair which is publicly known;
- i) *private key (secret key - deprecated)*: (in a public key cryptosystem) that key of a user's key pair which is known only by that user;
- j) *simple authentication*: authentication by means of simple password arrangements;
- k) *security policy*: the set of rules laid down by the security authority governing the use and provision of security services and facilities;
- l) *strong authentication*: authentication by means of cryptographically derived credentials;
- m) *trust*: generally, an entity can be said to "trust" a second entity when it (the first entity) makes the assumption that the second entity will behave exactly as the first entity expects. This trust may apply only for some specific function. The key role of trust in the authentication framework is to describe the relationship between an authenticating entity and a certification authority; an authenticating entity must be certain that it can trust the certification authority to create only valid and reliable certificates;
- n) *certificate serial number*: an integer value, unique within the issuing CA, which is unambiguously associated with a certificate issued by that CA.

4 Notation and abbreviations

4.1 The notation used in this Recommendation is defined in Table 1/X.509 below.

Note - In the table, the symbols X , X_1 , X_2 etc. occur in place of the names of users, while the symbol I occurs in place of arbitrary information.

4.2 The following abbreviations are used in this Recommendation:

CA	Certification Authority
DIB	Directory Information Base
DIT	Directory Information Tree
PKCS	Public key cryptosystem.

TABLE 1/X.509

Notation

NOTATION	MEANING
X_p	Public key of a user X.
X_s	Secret key of X.
$X_p[I]$	Encipherment of some information, I, using the public key of X.
$X_s[I]$	Encipherment of I using the secret key of X.
$X\{I\}$	The signing of I by user X. It consists of I with an enciphered summary appended.
$CA(X)$	A certification authority of user X.
$CA^n(X)$	(where $n > 1$): $CA(CA(\dots n \text{ times } \dots (X)))$.
$X_1 \ll X_2 \gg$	The certificate of user X_2 issued by certification authority X_1 .
$X_1 \ll X_2 \gg X_2 \ll X_3 \gg$	A chain of certificates (can be of arbitrary length), where each item is the certificate for the certification authority which produced the next. It is functionally equivalent to the following certificate $X_1 \ll X_{n+1} \gg$. For example, possession of $A \ll B \gg B \ll C \gg$ provides the same capability as $A \ll C \gg$, namely the ability to find out C_p given A_p .
$X_{1p} \cdot X_1 \ll X_2 \gg$	<p>The operation of unwrapping a certificate (or certificate chain) to extract a public key. It is an infix operator, whose left operand is the public key of a certification authority, and whose right operand is a certificate issued by that certification authority. The outcome is the public key of the user whose certificate is the right operand. For example:</p> $A_p \cdot A \ll B \gg B \ll C \gg$ <p>denotes the operation of using the public key of A to obtain B's public key, B_p, from its certificate, followed by using B_p to unwrap C's certificate. The outcome of the operation is the public key of C, C_p.</p>
$A \rightarrow B$	A certification path from A to B, formed of a chain of certificates, starting with $CA(A) \ll CA^2(A) \gg$ and ending with $CA(B) \ll B \gg$.

SECTION 1 - *Simple authentication*

5 Simple authentication procedure

5.1 Simple authentication is intended to provide local authorization based upon a Distinguished Name of a user, bilaterally agreed (optional) password and a bilateral understanding of the means of using and handling this password within a single domain. Utilization of Simple Authentication is primarily intended for local use only, i.e. for peer entity authentication between one DUA and one DSA or between one DSA and one DSA. Simple authentication may be achieved by several means:

- a) the transfer of the user's distinguished name and (optional) password in the clear (non-protected) to the recipient for evaluation;
- b) the transfer of the user's distinguished name, password, and a random number and/or a timestamp, all of which are protected by applying a one-way function;
- c) the transfer of the protected information described in b) together with a random number and/or timestamp, all of which is protected by applying a one-way function.

Note 1 - There is no requirement that the one-way functions applied be different.

Note 2 - The signalling of procedures for protecting passwords may be a matter for extension to the Document.

5.2 Where passwords are not protected, a minimal degree of security is provided for preventing unauthorized access. It should not be considered a basis for secure services. Protecting the user's distinguished name and password provides greater degrees of security. The algorithms to be used for the protection mechanism are typically non-enciphering one-way functions that are very simple to implement.

5.3 The general procedure for achieving simple authentication is shown in Figure 1/X.509.

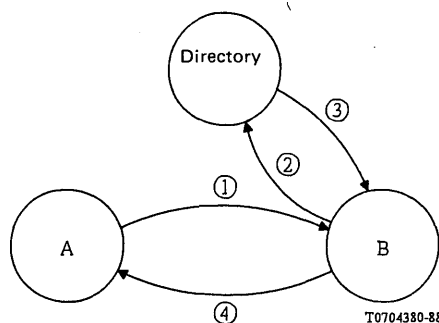


FIGURE 1/X.509

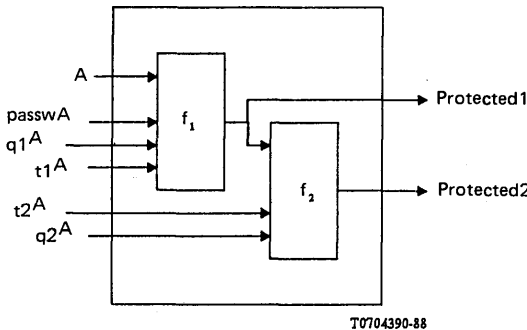
The unprotected simple authentication procedure

5.3.1 The following steps are involved:

- 1) an originating user A sends its distinguished name and password to a recipient user B;
- 2) B sends the purported distinguished name and password of A to the Directory, where the password is checked against that held as the **User Password** attribute within the directory entry for A (using the Compare operation of the Directory);
- 3) the Directory confirms (or denies) to B that the credentials are valid;
- 4) the success (or failure) of authentication may be conveyed to A.

5.3.2 The most basic form of simple authentication involves only step 1) and after B has checked the distinguished name and password, may include step 4).

5.4 Figure 2/X.509 illustrates two approaches by which protected identifying information may be generated. f_1 and f_2 are one-way functions (either identical or different) and the timestamps and random numbers are optional and subject to bilateral agreements.



- A = user's distinguished name
- t^A = timestamps
- $passw^A$ = password of A
- q^A = random numbers, optionally with a counter included

FIGURE 2/X.509

Protected simple authentication

5.4.1 Figure 3/X.509 illustrates the procedure for protected simple authentication.

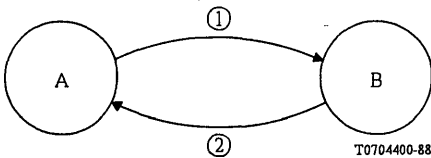


FIGURE 3/X.509

The protected simple authentication procedure

The following steps are involved (initially using f_1 only):

- 1) An originating user, User A, sends its protected identifying information (Authenticator1) to User B. Protection is achieved by applying the one-way function (f_1) of Figure 2/X.509, where the timestamp and/or random number (when used) is to minimize replay and to conceal the password.
 The protection of A's password is of the form:
 $Protected1 = f_1(t1^A, q1^A, passwA).$
 The information conveyed to B is of the form:
 $Authenticator1 = t1^A, q1^A, A, Protected1.$

B verifies the protected identifying information offered by A by generating (using the timestamp, distinguished name and, optionally, additional timestamp and/or random number provided by A, together with a local copy of A's password) a local protected copy of A's password (of the form of Protected1). B compares (for equality) the purported identifying information (Protected1) with the locally generated value).

- 2) B confirms (or denies) to A the verification of the protected identifying information.

5.4.2 The procedure of § 5.4.1 can be modified to afford greater protection (using f_1 and f_2).

The main differences are as follows:

- 1) A sends its (additionally) protected identifying information (Authenticator2) to B. Additional protection is achieved by applying a further one-way function, f_2 , as illustrated in Figure 2/X.509. The further protection is of the form:

$$\text{Protected2} = f_2(t2^A, q2^A, \text{Protected1}).$$

The information conveyed to B is of the form:

$$\text{Authenticator2} = t1^A, t2^A, q1^A, q2^A, A, \text{Protected2}.$$

For comparison, B generates a local value of A's additionally protected password and compares it (for equality) with that of Protected2. (Similar in principle to step 1) of § 5.4.1.)

- 2) B confirms (or denies) to A the verification of the protected identifying information.

Note - The procedures defined in this § are specified in terms of A and B. As applied to the Directory (specified in Recommendation X.511 and X.518), A could be a DUA binding to a DSA, B; alternatively A could be a DSA binding to another DSA, B.

5.5 A User Password attribute type contains the password of an object. An attribute value for the user password is a string specified by the object.

**UserPassword ::= ATTRIBUTE
WITH ATTRIBUTE-SYNTAX
OCTET STRING (SIZE (0..ub-user-password))
MATCHES FOR EQUALITY**

5.6 The following ASN.1 macro may be used to define the data type arising from applying a one-way function to a given other data type.

PROTECTED MACRO ::= SIGNATURE

SECTION 2 - *Strong authentication*

6 Basis of strong authentication

6.1 The approach to strong authentication taken in this Recommendation makes use of the properties of a family of cryptographic systems, known as public-key cryptosystems (PKCS). These cryptosystems, also described as asymmetric, involve a pair of keys, one secret and one public, rather than a single key as in conventional cryptographic systems. Annex B gives a brief introduction to these cryptosystems and the properties which make them useful in authentication. For a PKCS to be usable in this authentication framework, at the present time, it must have the property that both keys in the key pair can be used for encipherment with the secret key being used to decipher if the public key was used, and the public key being used to decipher if the secret key was used. In other words $X_p \cdot X_s = X_s \cdot X_p$ where X_p/X_s are encipherment/decipherment functions using the public/secret keys of user X.

Note - Alternative types of PKCS, i.e., ones which do not require the property of permutability and that can be supported without great modification to this Recommendation, are a possible future extension.

6.2 This authentication framework does not mandate a particular cryptosystem for use. It is intended that the framework will be applicable to any suitable public key cryptosystem, and will thus support changes to the methods used as a result of future advances in cryptography, mathematical techniques or computational capabilities. However, two users wishing to authenticate must support the same cryptographic algorithm for authentication to be performed correctly. Thus, within the context of a set of related applications, the choice of a single algorithm will serve to maximize the community of users able to authenticate and communicate securely. One example of a cryptographic algorithm can be found in Annex C.

6.3 Authentication relies on each user possessing a unique distinguished name. The allocation of distinguished names is the responsibility of the Naming Authorities. Each user must therefore trust the Naming Authorities not to issue duplicate distinguished names.

6.4 Each user is identified by its possession of its secret key. A second user is able to determine if a communication partner is in possession of the secret key, and can use this to corroborate that the communication partner is in fact the user. The validity of this corroboration depends on the secret key remaining confidential to the user.

6.5 For a user to determine that a communication partner is in possession of another user's secret key, it must itself be in possession of that user's public key. Whilst obtaining the value of this public key from the user's entry in the Directory is straightforward, verifying its correctness is more problematic. There are many possible ways for doing this: § 7 describes a process whereby a user's public key can be checked by reference to the Directory. This process can only operate if there is an unbroken chain of trusted points in the Directory between the users requiring to authenticate. Such a chain can be constructed by identifying a common point of trust. This common point of trust must be linked to each user by an unbroken chain of trusted points.

7 Obtaining a user's public key

7.1 In order for a user to trust the authentication procedure, it must obtain the other user's public key from a source that it trusts. Such a source, called a certification authority (CA), uses the public key algorithm to certify the public key, producing a *certificate*. The certificate, the form of which is specified in § 7.2 has the following properties:

- any user with access to the public key of the certification authority can recover the public key which was certified;
- no party other than the certification authority can modify the certificate without this being detected (certificates are unforgeable).

Because certificates are unforgeable, they can be published by being placed in the Directory, without the need for the latter to make special efforts to protect them.

Note - Although the CAs are unambiguously defined by a distinguished name in the DIT, this does not imply that there is any relationship between the organization of the CAs and the DIT.

7.2 A certification authority produces the certificate of a user by signing (see § 8) a collection of information, including the user's distinguished name and public key. Specifically, the certificate of a user with distinguished name A, produced by the certification authority CA, has the following form:

$$CA\langle\langle A \rangle\rangle = CA \{SN, AI, CA, A, Ap, T^A\}$$

where SN is the serial number of the certificate, AI is the identifier of the algorithm used to sign the certificate, and T^A indicates the period of validity of the certificate, and consists of two dates, the first and last on which the certificate is valid. Since T^A is assumed to be changed in

periods not less than 24 hours, it is expected that systems would use Coordinated Universal Time as a reference time base. The signature in the certificate can be checked for validity by any user with knowledge of CAP. The following ASN.1 data type can be used to represent certificates.

```

Certificate ::= SIGNED SEQUENCE{
    version          [0]Version DEFAULT 1988,
    serialNumber     SerialNumber,
    signature        AlgorithmIdentifier
    issuer           Name
    validity         Validity,
    subject          Name,
    subjectPublicKeyInfo SubjectPublicKeyInfo}

Version          ::= INTEGER { 1988(0)}
SerialNumber     ::= INTEGER

Validity         ::= SEQUENCE{
    notBefore       UTCTime,
    notAfter        UTCTime}

SubjectPublicKeyInfo ::= SEQUENCE{
    algorithm       AlgorithmIdentifier
    subjectKey      BIT STRING}

AlgorithmIdentifier ::= SEQUENCE{
    algorithm       OBJECT IDENTIFIER
    parameters     ANY DEFINED BY algorithm
                  OPTIONAL}

```

7.3 The directory entry of each user, A, who is participating in strong authentication, contains the certificate(s) of A. Such a certificate is generated by a Certification Authority of A which is an entity in the DIT. A Certification Authority of A, which may not be unique, is denoted CA(A), or simply CA if A is understood. The public key of A can thus be discovered by any user knowing the public key of CA. Discovering public keys is thus recursive.

7.4 If user A, trying to obtain the public key of user B, has already obtained the public key of CA(B), then the process is complete. In order to enable A to obtain the public key of CA(B), the directory entry of each Certification Authority, X, contains a number of certificates. These certificates are of two types. First there are forward certificates of X generated by other Certification Authorities. Second there are reverse certificates generated by X itself which are the certified public keys of other certification authorities. The existence of these certificates enables users to construct certification paths from one point to another.

7.5 A list of certificates needed to allow a particular user to obtain the public key of another, is known as a *certification path*. Each item in the list is a certificate of the certification authority of the next item in the list. A certification path from A to B (denoted $A \rightarrow B$):

- starts with a certificate produced by CA(A), namely $CA(A) \ll X^1 \gg$ for some entity X^1 ;
- continues with further certificates $X^i \ll X^{i+1} \gg$;
- ends with the certificate of B.

A certification path logically forms an unbroken chain of trusted points in the Directory Information Tree between two users wishing to authenticate. The precise method employed by users A and B to obtain certification paths $A \rightarrow B$ and $B \rightarrow A$ may vary. One way to facilitate this is to arrange a hierarchy of CAs, which may or may not coincide with all or part of the DIT hierarchy. The benefit of this is that users who have CAs in the hierarchy may establish a certification path between them using the Directory without any prior information. In order to allow for this each CA may store one certificate and one reverse certificate designated as corresponding to its superior CA.

7.6 Certificates are held within directory entries as attributes of type **UserCertificate**, **CACertificate** and **CrossCertificatePair**. These attribute types are known to the Directory. These attributes can be operated on using the same protocol operations as other attributes. The definition of these types may be found in § 3.3 of this Recommendation, the specification of these attribute types is as follows:

```

UserCertificate      ::=  ATTRIBUTE
                        WITH ATTRIBUTE-SYNTAX Certificate
CACertificate        ::=  ATTRIBUTE
                        WITH ATTRIBUTE-SYNTAX Certificate
CrossCertificatePair ::=  ATTRIBUTE
                        WITH ATTRIBUTE-SYNTAX CertificatePair
CertificatePair      ::=
    SEQUENCE{
        forward [o] Certificate OPTIONAL
        reverse [1] Certificate OPTIONAL
        -- at least one must be present --
    }

```

A user may obtain one or more certificates from one or more Certification Authorities. Each certificate bears the name of the Certification Authority which issued it.

7.7 In the general case, before users can mutually authenticate, the Directory must supply the complete certification and return certification paths. However, in practice, the amount of information which must be obtained from the Directory can be reduced for a particular instance of authentication by:

- a) if the two users that want to authenticate are served by the same certification authority, then the certification path becomes trivial, and the users unwrap each other's certificates directly;
- b) if the CAs of the users are arranged in a hierarchy, a user could store the public keys, certificates and reverse certificates of all certification authorities between the user and the root of the DIT. Typically, this would involve the user in knowing the public keys and certificates of only three or four certification authorities. The user would then only require to obtain the certification paths from the common point of trust;
- c) if a user frequently communicates with users certified by a particular other CA, that user could learn the certification path to that CA and the return certification path from that CA, making it necessary only to obtain the certificate of the other user itself from the directory;
- d) certification authorities can cross-certify one another by bilateral agreement. The result is to shorten the certification path;
- e) if two users have communicated before and have learned one another's certificates, they are able to authenticate without any recourse to the Directory.

In any case, having learned each other's certificates from the certification path, the users must check the validity of the received certificates.

7.8 (Example). Figure 4/X.509 illustrates a hypothetical example of a DIT fragment, where the CAs form a hierarchy. Besides the information shown at the CAs, we assume that each user knows the public key of its certification authority, and its own public and secret keys.

7.8.1 If the CAs of the users are arranged in a hierarchy, A can acquire the following certificates from the directory to establish a certification path to B:

$X \ll W \gg, W \ll V \gg, V \ll Y \gg, Y \ll Z \gg, Z \ll B \gg$

When A has obtained these certificates, it can unwrap the certification path in sequence to yield the contents of the certificate of B, including Bp:

$Bp = Xp \cdot X \ll W \gg W \ll V \gg V \ll Y \gg Y \ll Z \gg Z \ll B \gg$

In general, A also has to acquire the following certificates from the directory to establish the return certification path from B to A:

$Z \ll Y \gg, Y \ll V \gg, V \ll W \gg, W \ll X \gg, X \ll A \gg.$

When B receives these certificates from A, it can unwrap the return certification path in sequence to yield the contents of the certificate of A, including A_p :

$$A_p = Z_p \cdot Z\langle\langle Y \rangle\rangle Y\langle\langle V \rangle\rangle V\langle\langle W \rangle\rangle W\langle\langle X \rangle\rangle X\langle\langle A \rangle\rangle.$$

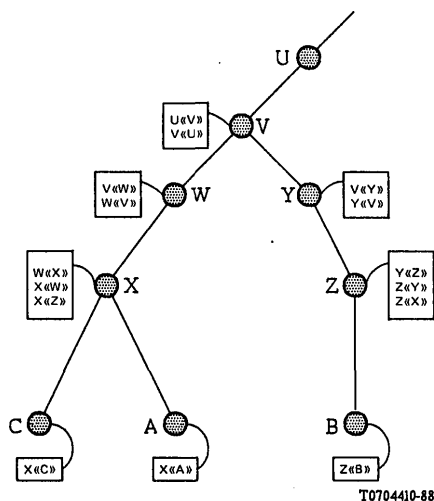


FIGURE 4/X.509

CA hierarchy - a hypothetical example

7.8.2 Applying the optimizations of § 7.7:

- a) taking A and C, for example: both know X_p , so that A simply has to directly acquire the certificate of C. Unwrapping the certification path reduces to:

$$C_p = X_p \cdot X\langle\langle C \rangle\rangle$$

and unwrapping the return certification Path reduces to:

$$A_p = X_p \cdot X\langle\langle A \rangle\rangle$$

- b) assuming that A would thus know $W\langle\langle X \rangle\rangle$, W_p , $V\langle\langle W \rangle\rangle$, V_p , $U\langle\langle V \rangle\rangle$, U_p , etc., reduces the information which A has to obtain from the directory to form the certification path to:

$$V\langle\langle Y \rangle\rangle, Y\langle\langle Z \rangle\rangle, Z\langle\langle B \rangle\rangle$$

and the information which A has to obtain from the directory to form the return certification path to:

$$Z\langle\langle Y \rangle\rangle, Y\langle\langle V \rangle\rangle$$

- c) assuming that A frequently communicates with users certified by Z, it can learn (in addition to the public keys learned in b) above) $V\langle\langle Y \rangle\rangle$, $Y\langle\langle V \rangle\rangle$, $Y\langle\langle Z \rangle\rangle$, and $Z\langle\langle Y \rangle\rangle$. To communicate with B, it need therefore only obtain $Z\langle\langle B \rangle\rangle$ from the directory;
- d) assuming that users certified by X and Z frequently communicate, then $X\langle\langle Z \rangle\rangle$ would be held in the directory entry for X, and vice versa (this is shown in Figure 4/X.509). If A wants to authenticate to B, A need only obtain:

$$X\langle\langle Z \rangle\rangle, Z\langle\langle B \rangle\rangle$$

to form the certification path, and

$$Z\langle\langle X \rangle\rangle$$

to form the return certification path;

- e) assuming users A and C have communicated before and have learned one another's certificates, they may use each other's public key directly, i.e.

$$C_p = X_p \cdot X_{\langle\langle C \rangle\rangle}$$

and

$$A_p = X_p \cdot X_{\langle\langle A \rangle\rangle}.$$

7.8.3 In the more general case the Certification Authorities do not relate in a hierarchical manner. Referring to the hypothetical example in Figure 5/X.509, suppose a user D, certified by U, wishes to authenticate to user E, certified by W. The directory entry of user D will hold the certificate $U_{\langle\langle D \rangle\rangle}$ and the entry of user E will hold the certificate $W_{\langle\langle E \rangle\rangle}$.

Let V be a CA with whom CAs U and W have at some previous time exchanged public keys in a trusted way. As a result, certificates $U_{\langle\langle V \rangle\rangle}$, $V_{\langle\langle U \rangle\rangle}$, $W_{\langle\langle V \rangle\rangle}$ and $V_{\langle\langle W \rangle\rangle}$ have been generated and stored in the Directory. Assume $U_{\langle\langle V \rangle\rangle}$ and $W_{\langle\langle V \rangle\rangle}$ are stored in the entry of V, $V_{\langle\langle U \rangle\rangle}$ is stored in U's entry, and $V_{\langle\langle W \rangle\rangle}$ is stored in W's entry.

User D must find a certification path to E. Various strategies could be used. One such strategy would be to regard the users and CAs as nodes, and the certificates as arcs in a directed graph, in these terms, D has to perform a search in the graph to find a path from U to E, one such being $U_{\langle\langle V \rangle\rangle}$, $V_{\langle\langle W \rangle\rangle}$, $W_{\langle\langle E \rangle\rangle}$. When this path has been discovered, the reverse path $W_{\langle\langle V \rangle\rangle}$, $V_{\langle\langle U \rangle\rangle}$, $U_{\langle\langle D \rangle\rangle}$ can also be constructed.

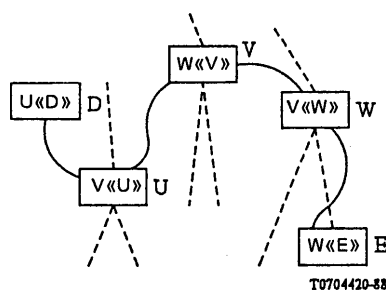


FIGURE 5/X.509

Non-hierarchical certification path - an example

8 Digital signatures

This section is not intended to specify a standard for digital signatures in general, but to specify the means by which the tokens are signed in the Directory.

8.1 Information (info) is signed by appending to it an enciphered summary of the information. The summary is produced by means of a one-way hash function, while the enciphering is carried out using the secret key of the signer (see Figure 6/X.509). Thus

$$X\{\text{Info}\} = \text{Info}, X_s[h(\text{Info})]$$

Note - The encipherment using the secret key ensures that the signature cannot be forged. The one-way nature of the hash function ensures that false information, generated so as to have the same hash result (and thus signature), cannot be substituted.

8.2 The recipient of signed information verifies the signature by:

- applying the one-way hash function to the information;
- comparing the result with that obtained by deciphering the signature using the public key of the signer.

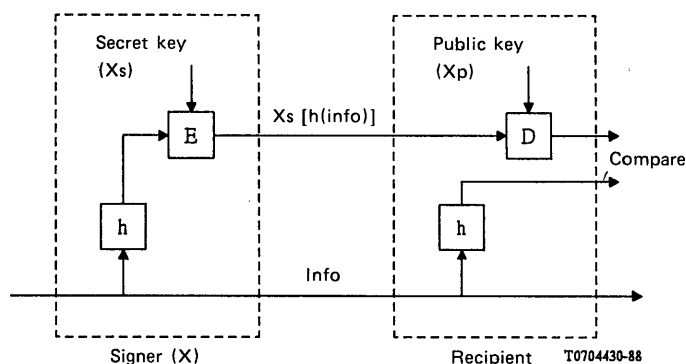


FIGURE 6/X.509

Digital signatures

8.3 This authentication framework does not mandate a single one-way hash function for use in signing. It is intended that the framework will be applicable to any suitable hash function, and will thus support changes to the methods used as a result of future advances in cryptography, mathematical techniques or computational capabilities. However, two users wishing to authenticate must support the same hash function for authentication to be performed correctly. Thus, within the context of a set of related applications, the choice of a single function will serve to maximize the community of users able to authenticate and communicate securely. An example hash function is specified in Annex D.

The signed information includes indicators that identify the hashing algorithm and the encryption algorithm used to compute the digital signature.

8.4 The encipherment of some data items may be described using the following ASN.1 MACRO:

```

ENCRYPTED MACRO ::=
BEGIN
TYPE NOTATION      ::= type (ToBeEnciphered)
VALUE NOTATION     ::= value (VALUE BIT STRING)
END

```

The value of the bit string is generated by taking the octets which form the complete encoding (using the ASN.1 Basic Encoding Rules) of the value of the **ToBeEnciphered** type and applying an encipherment procedure to those octets.

Note 1 - The encryption procedure requires agreement on the algorithm to be applied, including any parameters of the algorithm, such as any necessary keys, initialization values, and padding instructions. It is the responsibility of the encryption procedures to specify the means by which synchronization of the sender and receiver of data is achieved, which may include information in the bits to be transmitted.

Note 2 - The encryption procedure is required to take as input a string of octets and to generate a single string of bits as its result.

Note 3 - Mechanisms for secure agreement on the encryption algorithm and its parameters by the sender and receiver of data are outside the scope of this Recommendation.

8.5 In the case where a signature must be appended to a data type, the following ASN.1 macro may be used to define the data type resulting from applying a signature to the given data type.


```

SIGNED MACRO ::=
BEGIN

TYPE NOTATION ::= type (ToBeSigned)

VALUE NOTATION ::= value (VALUE

    SEQUENCE{
        ToBeSigned,
        AlgorithmIdentifier,
        -- of the algorithm used to compute
        -- the signature
        ENCRYPTED OCTET STRING
        -- where the octet string is the result
        -- of the hashing of the value of
        -- 'ToBeSigned' --}

END -- of SIGNED.    )

```

8.6 In the case where only the signature is required, the following ASN.1 macro may be used to define the data type resulting from applying a signature to the given data type.

```

SIGNATURE MACRO ::=
BEGIN

TYPE NOTATION ::= type (OfSignature)

VALUE NOTATION ::= value (VALUE

    SEQUENCE{
        AlgorithmIdentifier,
        -- of the algorithm used to compute
        -- the signature
        ENCRYPTED OCTET STRING
        -- where the octet string is a function (e.g. a
        -- compressed or hashed version) of the
        -- value 'OfSignature', which may include
        -- the identifier of the algorithm used to
        -- compute the signature --}

END -- of SIGNATURE.    )

```

8.7 In order to enable the validation of **SIGNED** and **SIGNATURE** types in a distributed environment, a distinguished encoding is required. A distinguished encoding of a **SIGNED** or **SIGNATURE** data value shall be obtained by applying the Basic Encoding Rules defined in Recommendation X.209 with the following restrictions:

- a) the definite form of length encoding shall be used, encoded in the minimum number of octets;
- b) for string types, the constructed form of encoding shall not be used;
- c) if the value of a type is its default value, it shall be absent;
- d) the components of a Set type shall be encoded in ascending order of their tag value;
- e) the components of a Set-of type shall be encoded in ascending order of their octet value;
- f) if the value of a Boolean type is true, the encoding shall have its contents octet set to 'FF'₁₆ ;
- g) each unused bits in the final octet of the encoding of a BitString value, if there are any, shall be set to zero;
- h) the encoding of a Real type shall be such that bases 8, 10 and 16 shall not be used, and the binary scaling factor shall be zero.

9 Strong authentication procedures

9.1 Overview

9.1.1 The basic approach to authentication has been outlined above, namely the corroboration of identity by demonstrating possession of a secret key. However, many authentication procedures employing this approach are possible. In general it is the business of a specific application to determine the appropriate procedures, so as to meet the security policy of the application. This clause describes three particular authentication procedures, which may be found useful across a range of applications.

Note - This Recommendation does not specify the procedures to the detail required for implementation. However, additional standards could be envisaged which would do so, either in an application-specific or in a general-purpose way.

9.1.2 The three procedures involve different numbers of exchanges of authentication information, and consequently provide different types of assurance to their participants. Specifically,

- a) one way authentication, described in § 9.2, involves a single transfer of information from one user (A) intended for another (B), and establishes the following:
 - the identity of A, and that the authentication token actually was generated by A;
 - the identity of B, and that the authentication token actually was intended to be sent to B;
 - the integrity and "originality" (the property of not having been sent two or more times) of the authentication token being transferred.

The latter properties can also be established for arbitrary additional data accompanying the transfer;

- b) two-way authentication, described in § 9.3, involves, in addition, a reply from B to A.

It establishes, in addition, the following:

- that the authentication token generated in the reply actually was generated by B and was intended to be sent to A;
 - the integrity and originality of the authentication token sent in the reply;
 - (optionally) the mutual secrecy of part of the tokens;
- c) three-way authentication, described in § 9.4, involves, in addition, a further transfer from A to B. It establishes the same properties as the two-way authentication, but does so without the need for association time stamp checking.

In each case where Strong Authentication is to take place, A must obtain the public key of B, and the return certification path from B to A, prior to any exchange of information. This may involve access to the Directory, as described in § 7 above. Any such access is not mentioned again in the description of the procedures below.

The checking of timestamps as mentioned in the following sections only applies when either synchronized clocks are used in a local environment, or if clocks are logically synchronized by bilateral agreements. In any case, it is recommended that Coordinated Universal Time be used.

For each of the three authentication procedures described below, it is assumed that party A has checked the validity of all of the certificates in the certification path.

9.2 One-way authentication

The following steps are involved, as depicted in Figure 7/X.509.

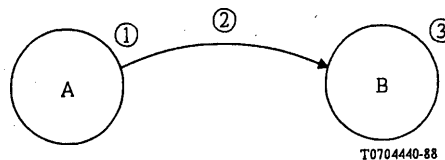


FIGURE 7/X.509

One-way authentication

- 1) A generates r^A , a non-repeating number, which is used to detect replay attacks and to prevent forgery.
- 2) A sends the following message to B:

$B \rightarrow A, A\{t^A, r^A, B\}$

where t^A is a timestamp. t^A consists of one or two dates: the generation time of the token (which is optional) and the expire date. Alternatively, if data origin authentication of "sgnData" is to be provided by the digital signature:

$B \rightarrow A, A\{t^A, r^A, B, \text{sgnData}\}$

In cases where information is to be conveyed which will subsequently be used as a secret key (this information is referred to as "encData"):

$B \rightarrow A, A\{t^A, r^A, B, \text{sgnData}, Bp[\text{encData}]\}$.

The use of "encData" as a secret key implies that it must be chosen carefully, e.g. to be a strong key for whatever cryptosystem is used as indicated in the "sgnData" field of the token.

- 3) B carries out the following actions:
 - a) obtains A_p from $B \rightarrow A$, checking that A's certificate has not expired;
 - b) verifies the signature, and thus the integrity of the signed information;
 - c) checks that B itself is the intended recipient;
 - d) checks that the timestamp is "current";
 - e) optionally, checks that r^A has not been replayed. This could, for example, be achieved by having r^A include a sequential part that is checked by a local implementation for its value uniqueness.

r^A is valid until the expire date indicated by t^A , r^A is always accompanied by a sequential part, which indicates that A will not repeat the token during the timerange t^A and therefore that checking of the value of r^A itself is not required.

In any case it is reasonable for party B to store the sequential part together with timestamp t^A in the clear and together with the hashed part of the token during timerange t^A .

9.3 Two-way authentication

The following steps are involved, as depicted in Figure 8/X.509.

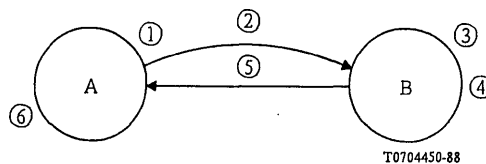


FIGURE 8/X.509

Two-way authentication

- 1) as for § 9.2.
- 2) as for § 9.2.
- 3) as for § 9.2.
- 4) B generates r^B , a non-repeating number, used for similar purpose(s) to r^A .
- 5) B sends the following authentication token to A:

$B \{t^B, r^B, A, r^A\}$

where t^B is a timestamp defined in the same way as t^A .

Alternatively, if data origin authentication of "sgnData" is to be provided by the digital signature:

$B \{t^B, r^B, A, r^A, \text{sgnData}\}$

In cases where information is to be conveyed which will subsequently be used as a secret key (this information is referred to as "encData"):

$B \{t^B, r^B, A, r^A, \text{sgnData}, \text{Ap}[\text{encData}]\}$.

The use of "encData" as a secret key implies that it must be chosen carefully, e.g. to be a strong key for whatever cryptosystem is used as indicated in the "sgnData" field of the token.

- 6) A carries out the following actions:
 - a) verifies the signature, and thus the integrity of the signed information;
 - b) checks that A is the intended recipient;
 - c) checks that the timestamp t^B is "current";
 - d) optionally, checks that r^B has not been replayed [see § 9.2 step 3) e)].

9.4 Three-way authentication

The following steps are involved, as depicted in Figure 9/X.509.

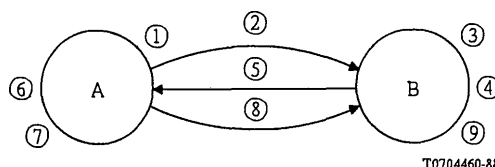


FIGURE 9/X.509

Three-way authentication

- 1) as for § 9.3.
- 2) as for § 9.3. Timestamp t^A may be zero.
- 3) as for § 9.3, except that the timestamp need not be checked.
- 4) as for § 9.3.
- 5) as for § 9.3. Timestamp t^B may be zero.
- 6) as for § 9.3, except that the timestamp need not be checked.
- 7) A checks that the received r^A is identical to the r^A which was sent.
- 8) A sends the following authentication token to B:
 $A\{r^B\}$.
- 9) B carries out the following actions:
 - a) checks the signature and thus the integrity of the signed information;
 - b) checks that the received r^B is identical to the r^B which was sent by B.

10 Management of keys and certificates

10.1 Generation of key pairs

10.1.1 The overall security management policy of an implementation will define the lifecycle of key pairs, and is thus outside the scope of the authentication framework. However, it is vital to the overall security that all secret keys remain known only to the user to whom they belong.

Key data is not easy for a human user to remember, so a suitable method for storing it in a convenient transportable manner must be employed. One possible mechanism would be to use a "Smart Card". This would hold the secret and (optionally) public keys of the user, the user's certificate, and a copy of the certification authority's public key. The use of this card must additionally be secured by e.g. at least use of a PIN (Personal Identification Number), increasing the security of the system by requiring the user to possess the card and to know how to access it. The exact method chosen for storing such data, however, is beyond the scope of this Recommendation.

10.1.2 There are three ways in which a user's key pair may be produced, as described in § 10.1.2.1 to § 10.1.2.3.

10.1.2.1 The user generates its own key pair. This method has the advantage that a user's secret key is never released to another entity, but requires a certain level of competence by the user as described in Annex C.

10.1.2.2 The key pair is generated by a third party. The third party must release the secret key to the user in a physically secure manner, then actively destroy all information relating to the creation of the key pair plus the keys themselves. Suitable physical security measures must be employed to ensure that the third party and the data operations are free from tampering.

10.1.2.3 The key pair is generated by the CA. This is a special case of § 10.1.2.2, and the considerations there apply.

Note - The certification authority already exhibits trusted functionality with respect to the user, and will be subject to the necessary physical security measures. This method has the advantage of not requiring secure data transfer to the CA for certification.

10.1.2.4 The cryptosystem in use imposes particular (technical) constraints on key generation.

10.2 Management of certificates

10.2.1 A certificate associates the public key and unique distinguished name of the user it describes. Thus:

- a) a certification authority must be satisfied of the identity of a user before creating a certificate for it;

- b) a certification authority must not issue certificates for two users with the same name.

10.2.2 The production of a certificate occurs off-line and must not be performed with an automatic query/response mechanism. The advantage of this certification is that because the secret key of the certification authority, CAs, is never known except in the isolated and physically secure CA, the CA secret key may then only be learnt by an attack on CA itself, making compromise unlikely.

10.2.3 It is important that the transfer of information to the certification authority is not compromised, and suitable physical security measures must be taken. In this regard:

- a) it would be a serious breach of security if the CA issued a certificate for a user with a public key that had been tampered with;
- b) if the means of generation of key pairs of § 10.1.2.3 is employed, no secure transfer is needed;
- c) if the means of generation of key pairs of § 10.1.2.1 or of § 10.1.2.2 is employed, the user may use different methods (on-line or off-line) to communicate its public key to the CA in a secure manner. On-line methods may provide some additional flexibility for remote operations performed between the user and the CA.

10.2.4 A certificate is a publicly available piece of information, and no specific security measures need to be employed with respect to its transportation to the Directory. As it is produced by an off-line certification authority on behalf of a user who will be given a copy of it, the user need only store this information in its directory entry on a subsequent access to the Directory. Alternatively the CA could lodge the certificate for the user, in which case this agent must be given suitable access rights.

10.2.5 Certificates will have a lifetime associated with them, at the end of which they expire. In order to provide continuity of service, the CA shall ensure timely availability of replacement certificates to supersede expired/expiring certificates. This has a number of aspects, as described in §§ 10.2.5.1 and 10.2.5.2.

10.2.5.1 Validity of certificates may be designed so that each becomes valid at the time of expiry of its predecessor, or an overlap may be allowed. The latter prevents the CA from having to install and distribute a large number of certificates that may run out at the same expiration date.

10.2.5.2 Expired certificates will normally be removed from the Directory. It is a matter for the security policy and responsibility of the CA to keep old certificates for a period of time if a non repudiation of data service is provided.

10.2.6 Certificates may be revoked prior to their expiration time, e.g. if the user's secret key is assumed to be compromised, or the user is no longer to be certified by the CA, or if the CA's certificate is assumed to be compromised. This has a number of aspects, as described in §§ 10.2.6.1-10.2.6.4.

10.2.6.1 The revocation of a user certificate or CA certificate shall be made known by the CA, and a new certificate shall be made available, if appropriate. The CA may then inform the owner of the certificate about its revocation by some off-line procedure.

10.2.6.2 The CA shall maintain:

- a) a time-stamped list of the certificates it issued which have been revoked;
- b) a time-stamped list of revoked certificates of all CAs known to the CA, certified by the CA.

Both certified lists shall exist, even if empty.

10.2.6.3 The maintenance of Directory entries affected by the CA's revocation lists is the responsibility of the Directory and its users, acting in accordance with the security policy. For example, the user may modify its object entry by replacing the old certificate with a new one. The latter will then be used to authenticate the user to the Directory.

10.2.6.4 The revocation lists ("black-lists") are held within entries as attributes of types "CertificateRevocationList" and "AuthorityRevocationList". These attributes can be operated on using the same operations as other attributes. These attribute types are defined as follows:

```
CertificateRevocationList ::= ATTRIBUTE
    WITH ATTRIBUTE-SYNTAX CertificateList
AuthorityRevocationList  ::= ATTRIBUTE
    WITH ATTRIBUTE-SYNTAX CertificateList
CertificateList           ::= SIGNED SEQUENCE{
    signature AlgorithmIdentifier,
    issuer Name,
    lastUpdate UTCTime,
    revokedCertificates
        SIGNED SEQUENCE OF SEQUENCE{
            signature AlgorithmIdentifier,
            issuer Name, CertificateSerialNumber subject,
            revocationDate UTCTime}
        OPTIONAL}
```

Note 1 - The checking of the entire list of certificates is a local matter.

Note 2 - If a non-repudiation of data service is dependent on keys provided by the CA the service must ensure that all relevant keys of the CA (revoked or expired) and the timestamped revocation lists are archived and certified by a current authority.

ANNEX A

(to Recommendation X.509)

Security requirements

This Annex does not form an integral part of this Recommendation.

[Additional material relevant to this topic can be found in OSI 7498 - Information Processing Systems - OSI Reference Model - Part 2, Security Architecture.]

Many OSI applications, CCITT-defined services and non-CCITT-defined services will have requirements for security. Such requirements derive from the need to protect the transfer of information from a range of potential threats.

A.1 Threats

Some commonly known threats are:

- a) *identity interception*: the identity of one or more of the users involved in a communication is observed for misuse;
- b) *masquerade*: the pretense by a user to be a different user in order to gain access to information or to acquire additional privileges;
- c) *replay*: the recording and subsequent replay of a communication at some later date;
- d) *data interception*: the observation of user data during a communication by an unauthorized user;
- e) *manipulation*: the replacement, insertion, deletion or misordering of user data during a communication by an unauthorized user;

- f) *repudiation*: the denial by a user of having participated in part or all of a communication;
- g) *denial of service*: the prevention or interruption of a communication or the delay of time-critical operations;

Note - This security threat is a more general one and depends on the individual application or on the intention of the unauthorized disruption and is therefore not explicitly within the scope of the authentication framework.

- h) *mis-routing*: the mis-routing of a communication path intended for one user to another;

Note - Mis-routing will naturally occur in OSI layers 1 - 3. Therefore mis-routing is outside of the scope of the authentication framework. However, it may be possible to avoid the consequences of mis-routing by using appropriate security services as provided within the authentication framework.

- i) *traffic analysis*: the observation of information about a communication between users (e.g. absence/presence, frequency, direction, sequence, type, amount, etc.).

Note - Traffic analysis threats are naturally not restricted to a certain OSI layer. Therefore traffic analysis is generally outside the scope of the authentication framework. However, traffic analysis can be partially protected against by generating additional unintelligible traffic (traffic padding), using enciphered or random data.

A.2 Security services

In order to protect against perceived threats, various security services need to be provided. Security services as provided by the authentication framework are performed by means of the security mechanisms described in A.3 of this Annex.

- a) *peer entity authentication*: this service provides corroboration that a user in a certain instance of communication is the one claimed. Two different peer entity authentication services may be requested:
 - *single entity authentication* (either *data origin* entity authentication or *data recipient* entity authentication);
 - *mutual authentication*, where both users communicating authenticate each other.

When requesting a peer entity authentication service, the two users agree whether their identities will be protected or not.

The peer entity authentication service is supported by the authentication framework. It can be used to protect against masquerade and replay, concerning the user's identities;

- b) *access control*: this service can be used to protect against the unauthorized use of resources. The access control service is provided by the Directory or another application and is therefore not a concern of the authentication framework;
- c) *data confidentiality*: this service can be used to provide for protection of data from unauthorized disclosure. The data confidentiality service is supported by the authentication framework. It can be used to protect against data interception;
- d) *data integrity*: this service provides proof of the integrity of data in a communication. The data integrity service is supported by the authentication framework. It can be used to detect and protect against manipulation;
- e) *non-repudiation*: this service provides proof of the integrity and origin of data - both in an unforgeable relationship - which can be verified by any third party at any time.

A.3 Security mechanisms

The security mechanisms outlined here perform the security services described in A.2.

a) *authentication exchange*: there are two grades of authentication framework:

- *simple authentication*: relies on the originator supplying its name and password, which are checked by the recipient;
- *strong authentication*: relies on the use of cryptographic techniques to protect the exchange of validating information. In the authentication framework, strong authentication is based upon an asymmetric scheme.

The authentication exchange mechanism is used to support the peer entity authentication service;

b) *encipherment*: the authentication framework envisages the encipherment of data during transfer. Either asymmetric or symmetric schemes may be used. The necessary key exchange for either case is performed either within a preceding authentication exchange or off-line any time before the intended communication. The latter case is outside the scope of the authentication framework. The encipherment mechanism supports the data confidentiality service;

c) *data integrity*: this mechanism involves the encipherment of a compressed string of the relevant data to be transferred. Together with the plain data, this message is sent to the recipient. The recipient repeats the compressing and subsequent encipherment of the plain data and compares the results with that created by the originator to prove integrity.

The data integrity mechanism can be provided by encipherment of the compressed plain data by either an asymmetric scheme or a symmetric scheme. (With the symmetric scheme, compression and encipherment of data might be processed simultaneously.) The mechanism is not explicitly provided by the authentication framework. However it is fully provided as a part of the digital signature mechanism (see below) using an asymmetric scheme.

The data integrity mechanism supports the data integrity service. It also partially supports the non-repudiation service (that service also needs the digital signature mechanism for its requirement to be fully met);

d) *digital signature*: this mechanism involves the encipherment, by the originator's secret key, of a compressed string of the relevant data to be transferred. The digital signature together with the plain data is sent to the recipient. Similarly to the case of the data integrity mechanism, this message is processed by the recipient to prove integrity. The digital signature mechanism also proves the authenticity of the originator and the unambiguous relationship between the originator and the data that was transferred.

The authentication framework supports the digital signature mechanism using an asymmetric scheme.

The digital signature mechanism supports the data integrity service and also supports the non-repudiation service.

A.4 Threats protected against by the security services

The table at the end of this Annex indicates the security threats which each security service can protect against. The presence of an asterisk (*) indicates that a certain security service affords protection against a certain threat.

A.5 Negotiation of security services and mechanisms

The provision of security features during an instance of communication requires the negotiation of the context in which security services are required. This entails agreement on the type of security mechanisms and security parameters that are necessary to provide such security services. The procedures required for negotiating mechanisms and parameters can either be carried out as an integral part of the normal connection establishment procedure or as a separate process. The precise details of these procedures for negotiation are not specified in this Annex.

SERVICES

THREATS	Entity Authentication	Data Confidentiality	Data Integrity	Non- Repudiation
Identity Interception	* (if req'd)			
Data interception		*		
Masquerade	*			
Replay	* (identity)		* (data)	*
Manipulation			*	*
Repudiation				*

ANNEX B

(to Recommendation X.509)

An introduction to public key cryptography

This Annex does not form an integral part of this Recommendation.

In conventional cryptographic systems, the key used to encipher information by the originator of a secret message is the same as that used to decipher the message by the legitimate recipient.

In public key cryptosystems (PKCS), however, keys come in pairs, one key of which is used for enciphering and the other for deciphering. Each key pair is associated with a particular user X. One of the keys, known as the public key (X_p) is publicly known, and can be used by any user to encipher data. Only X, who possesses the complementary secret key (X_s) may decipher the data. (This is represented notationally by $D = X_s[X_p[D]]$.) It is computationally infeasible to derive the secret key from knowledge of the public key. Any user can thus communicate a piece of information which only X can find out, by enciphering it under X_p . By extension, two users can communicate in secret, by using each other's public key to encipher the data, as shown in Figure B-1/X.509.

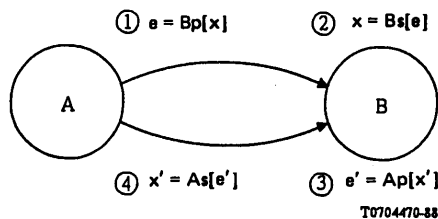


FIGURE B-1/X.509

Use of a PKCS to exchange secret information

User A has public key A_p and secret key A_s , and user B has another set of keys, B_p and B_s . A and B both know the public keys of each other, but are unaware of the secret key of the other party. A and B may therefore exchange secret information with one another using the following steps (illustrated in Figure B-1/X.509):

- 1) A wishes to send some secret information x to B. A therefore enciphers x under B's enciphering key and sends the enciphered information e to B. This is represented by:

$$e = B_p[x].$$
- 2) B may now decipher this encipherment e to obtain the information x by using the secret decipherment key B_s . Note that B is the only possessor of B_s , and because this key may never be disclosed or sent, it is impossible for any other party to obtain the information x . The possession of B_s determines the identity of B. The decipherment operation is represented by:

$$x = B_s[e], \text{ or } x = B_s[B_p[x]].$$
- 3) B may now similarly send some secret information, x' , to A, under A's enciphering key, A_p :

$$e' = A_p[x'].$$
- 4) A obtains x' by deciphering e' :

$$x' = A_s[e'], \text{ or } x' = A_s[A_p[x']].$$

By this means, A and B have exchanged secret information x and x' . This information may not be obtained by anyone other than A and B, providing that their secret keys are not revealed.

Such an exchange can, as well as transferring secret information between the parties, serve to verify their identities. Specifically, A and B are identified by their possession of the secret deciphering keys, A_s and B_s respectively. A may determine if B is in possession of the secret deciphering key, B_s , by having returned part of his information x in B's message x' . This indicates to A that communication is taking place with the possessor of B_s . B may similarly test the identity of A.

It is a property of some PKCS that the steps of decipherment and encipherment can be reversed, as in $D = X_p[X_s[D]]$. This allows a piece of information which could only have been originated by X, to be readable by any user (who has possession of X_p). This can therefore be used in the certifying of the source of information, and is the basis for digital signatures. Only PKCS which have this (permutability) property are suitable for use in this authentication framework. One such algorithm is described in Annex C.

For further information, see:

DIFFIE, W. and HELLMAN, M. E. (November 1976) - New Directions in Cryptography, *IEEE Transactions on Information Theory*, IT-22, No. 6.

ANNEX C

(to Recommendation X.509)

The RSA public key cryptosystem

This Annex does not form an integral part of this Recommendation.

Note - The cryptosystem specified in this Annex, which was invented by R. L. Rivest, A. Shamir and L. Adleman, is widely known as "RSA".

C.1 *Scope and field of application*

It is beyond the scope of this paper to discuss RSA fully. However, a brief description is given on the method, which relies on the use of modular exponentiation.

C.2 *References*

For further information, see:

1) General

RIVEST, R. L., SHAMIR, A. and ADLEMAN, L. (February 1978) - A Method for Obtaining Digital Signatures and Public-key Cryptosystems, *Communications of the ACM*, 21, 2, 120-126.

2) Key Generation Reference

GORDON, J. - Strong RSA Keys, *Electronics Letters*, 20, 5, 514-516.

3) Decipherment Reference

QUISQUATER, J. J. and COUVREUR, C. (October 14, 1982) - Fast Decipherment Algorithm for RSA Public-key Cryptosystems, *Electronics Letters*, 18, 21, 905-907.

C.3 *Definitions*

- a) *public key*: the pair of parameters consisting of the Public Exponent and the Arithmetic Modulus;

Note - The ASN.1 data element **subjectPublicKey** defined as **BIT STRING** (see Annex G), should be interpreted in the case of RSA as being of type:

SEQUENCE {INTEGER,INTEGER}

where the first integer is the Arithmetic Modulus and the second is the Public Exponent. The sequence is represented by means of the ASN.1 Basic Encoding Rules.

- b) *secret key*: the pair of parameters consisting of the Secret Exponent and the Arithmetic Modulus.

C.4 *Symbols and abbreviations*

X,Y data blocks which are arithmetically less than the modulus

n the Arithmetic Modulus

e the Public Exponent

d the Secret Exponent

p,q the prime numbers whose product forms the Arithmetic Modulus (n).

Note - While the prime numbers are preferably two in number, the use of a Modulus with three- or more prime factors is not precluded.

mod n arithmetic modulo n.

C.5 Description

This asymmetric algorithm uses the power function for transformation of data blocks such that:

$$Y = X^e \bmod n \quad \text{with } 0 \leq X < n$$

$$X = Y^d \bmod n \quad 0 \leq Y < n$$

which may be satisfied, for example, by

$$ed \bmod \text{lcm}(p-1, q-1) = 1,$$

$$ed \bmod (p-1)(q-1) = 1$$

To effect this process, a data block must be interpreted as an integer. This is accomplished by considering the entire data block to be an ordered sequence of bits (of length l , say). The integer is then formed as the sum of the bits after giving a weight of 2^{l-1} to the first bit and dividing the weight by 2 for each subsequent bit (the last bit has a weight of 1).

The data block length should be the largest number of octets containing fewer bits than the modulus. Incomplete blocks should be padded in any way desired. Any number of blocks of additional padding may be added.

C.6 Security requirements

C.6.1 Key lengths

It is recognized that the acceptable key length is likely to change with time, subject to the cost and availability of hardware, the time taken, advances in techniques and the level of security required. It is recommended that a value for the length of n of 512 bits be adopted initially, but subject to *further study*.

C.6.2 Key generation

The security of RSA relies on the difficulty of factorizing n . There are many algorithms for performing this operation, and in order to thwart the use of any currently known technique, the values p and q must be chosen carefully, according to the following rules [e.g. see Reference 2), Section C.2]:

- a) they should be chosen randomly;
- b) they should be large;
- c) they should be prime;
- d) $|p-q|$ should be large;
- e) $(p+1)$ must possess a large prime factor;
- f) $(q+1)$ must possess a large prime factor;
- g) $(p-1)$ must possess a large prime factor, say r ;
- h) $(q-1)$ must possess a large prime factor, say s ;
- i) $(r-1)$ must possess a large prime factor;
- j) $(s-1)$ must possess a large prime factor.

After generating the public and secret keys, e.g. " X_p " and " X_s " as defined in § 3.3 and § 4.1 of this Recommendation which consist of d , e and n , the values p and q together with all other data produced such as the product $(p-1)(q-1)$ and the large prime factors should preferably be destroyed. However, keeping p and q locally can improve throughput in decryption by two to four times. The decision to keep p and q is considered to be a local matter [Reference 3)].

It must be ensured that $e > \log_2(n)$ in order to prevent attack by taking the e 'th root mod n to disclose the plaintext.

C.7 Public exponent

The Public Exponent (e) could be common to the whole environment, in order to minimize the length of that part of the public key that actually has to be distributed, in order to reduce transmission capacity and complexity of transformation (see Note 1).

Exponent e should be large enough but such that exponentiation can be performed efficiently with regard to processing time and storage capacity. If a fixed public exponent e is desired, there are notable merits for the use of the Fermat Number F_4 (see Note 2).

$$F_4 = 2^{2^4} + 1$$

= 65537 decimal, and

= 1 0000 0000 0000 0001 binary.

Note 1 - Although both Modulus n and Exponent e are public, the Modulus should not be the part which is common to a group of users. Knowledge of Modulus " n ", Public Exponent " e " and Secret Exponent " d " is sufficient to determine the factorization of " n ". Therefore if the modulus was common, everyone could deduce its factors, thereby finding everyone else's secret exponent.

Note 2 - The fixed exponent should be large and prime but it should also provide efficient processing. Fermat Number F_4 meets these requirements, e.g. authentication takes only 17 multiplications and is on the average 30 times faster than decipherment.

C.8 Conformance

Whilst this Annex specifies an algorithm for the public and secret functions, it does not define the method whereby the calculations are carried out; therefore there may be different products which comply with this Annex and are mutually compatible.

ANNEX D

(to Recommendation X.509)

Hash functions

This Annex does not form an integral part of this Recommendation.

D.1 Requirements for hash functions

To use a hash function as a secure one-way function, it must not be possible to obtain easily the same hash result from different combinations of the input message.

A strong hash function will meet the following requirements:

- a) the hash function must be one-way, i.e. given any possible hash result it must be computationally infeasible to construct an input message which hashes to this result;
- b) the hash function must be collision-free, i.e. it must be computationally infeasible to construct two distinct input messages which hash to the same result.

D.2 Description of a hash function

The following hash function ("square-mod n ") performs the compression of the data on a block by block basis.

Hashing is done in three major steps:

- 1) The string of data to be hashed is divided into blocks B of equal length. This length is determined by the characteristics of the asymmetric cryptosystem used for signing. With the RSA cryptosystem, this length (in octets) is the largest integer, l , such that, with modulus n , $16 \leq l < \log_2 n$.

- 2) For non-invertibility reasons each octet of the block is split in half. Each of the halves is headed ("padded") by binary ones. By this zoning, stiffness or redundancy is introduced that increases the non-invertibility property of the hash function considerably. Each block generated in step 1 is spread to the length of the modulus n .
- 3) Each block resulting from step 2 is added to the previous block modulo 2, squared, and reduced modulo n , until all m blocks are processed.

The result is thus the value H_m , where

$$H_0 = 0$$

$$H_i = (H_{i-1} \oplus B_i)^2 \bmod n, \text{ for } 1 \leq i \leq m$$

If the last block of the data to be hashed is incomplete, it is padded with "1"s.

ANNEX E

(to Recommendation X.509)

Threats protected against by the strong authentication method

This Annex does not form an integral part of this Recommendation.

The strong authentication method described in this Recommendation offers protection against the threats as described in Annex A for strong authentication.

In addition, there is a range of potential threats that are specific to the strong authentication method itself. These are:

Compromise of the user's secret key - one of the basic principles of strong authentication is that the user's secret key remain secure. A number of practical methods are available for the user to hold his secret key in a manner that provides adequate security. The consequences of the compromise are limited to subversion of communication involving that user.

Compromise of the CA's secret key - that the secret key of a CA remain secure is also a basic principle of strong authentication. Physical security and "need to know" methods apply. The consequences of the compromise are limited to subversion of communication involving any user certified by that CA.

Misleading CA into producing an invalid certificate - the fact that CAs are off-line affords some protection. The onus is on the CA to check that purported strong credentials are valid before creating a certificate. The consequences of the compromise are limited to subversion of communication involving the user for whom the certificate was created, and anyone impacted by the invalid certificate.

Collusion between a rogue CA and user - such a collusive attack will defeat the method. This would constitute a betrayal of the trust placed in the CA. The consequences of a rogue CA are limited to subversion of communication involving any user certified by that CA.

Forging of a certificate - the strong authentication method protects against the forging of a certificate by having the CA sign it. The method depends on maintaining the secrecy of the CA's secret key.

Forging of a token - the strong authentication method protects against the forging of a token by having the sender sign it. The method depends on maintaining the secrecy of the sender's secret key.

Replay of a token - the one- and two-way authentication methods protect against the replay of a token by the inclusion of a timestamp in the token. The three-way method does so by checking the random numbers.

Attack on the cryptographic system - the likelihood of effective cryptanalysis of the system, based on advances in computational number theory and leading to the need for a greater key length are reasonably predictable.

ANNEX F

(to Recommendation X.509)

Data confidentiality

This Annex does not form an integral part of this Recommendation.

F.1 Introduction

The process of data confidentiality can be initiated after the necessary keys for encipherment have been exchanged. This might be provided by a preceding authentication exchange as described in § 9 or by some other key exchange process, the latter being outside the scope of this document.

Data confidentiality can be provided either by the application of an asymmetric or symmetric enciphering scheme.

F.2 Data confidentiality by asymmetric encipherment

In this case Data Confidentiality is performed by means of an originator enciphering the data to be sent using the intended recipient's public key; the recipient will then decipher it using its secret key.

F.3 Data confidentiality by symmetric encipherment

In this case Data Confidentiality is achieved by the use of a symmetric enciphering algorithm. Its choice is outside the scope of the authentication framework.

Where an authentication exchange according to § 9 has been carried out by the two parties involved, then a key for the usage of a symmetric algorithm can be derived. Choosing secret keys depends on the transformation to be used. The parties must be sure that they are strong keys. This Recommendation does not specify how this choice is made, although clearly this would need to be agreed by the parties concerned, or specified in other standards.

ANNEX G

(to Recommendation X.509)

Authentication framework in ASN.1

This Annex is part of the Recommendation.

This Annex includes all of the ASN.1 type, macro and value definitions contained in this Recommendation in the form of the ASN.1 module, "AuthenticationFramework".

AuthenticationFramework {joint-iso-ccitt ds(5) modules(1)
authenticationFramework(7)}

DEFINITIONS ::=
BEGIN

EXPORTS AlgorithmIdentifier, AuthorityRevocationList, CACertificate, Certificate,
 Certificates, CertificationPath, CertificateRevocationList, UserCertificate,
 CrossCertificatePair, UserPassword, ALGORITHM,
 ENCRYPTED, PROTECTED, SIGNATURE, SIGNED;


```

IMPORTS
    informationFramework, selectedAttributeTypes, upperBounds
    FROM UsefulDefinitions {joint-iso-ccitt ds(5)modules(1)
        usefulDefinitions(0)}
    Name, ATTRIBUTE,ATTRIBUTE-SYNTAX
    FROM InformationFramework informationFramework

ub-user-passwordFROM Upper Bounds upperBounds;

-- types

Certificate ::= SIGNED SEQUENCE{
    version                [0] Version DEFAULT 1988,
    serialNumber            SerialNumber,
    signature               AlgorithmIdentifier,
    issuer                  Name,
    validity                Validity,
    subject                 Name,
    subjectPublicKeyInfo    SubjectPublicKeyInfo}

Version ::= INTEGER { 1988(0)}
SerialNumber ::= INTEGER

Validity ::= SEQUENCE{
    notBefore              UTCTime
    notAfter               UTCTime}

SubjectPublicKeyInfo ::= SEQUENCE{
    algorithm               AlgorithmIdentifier
    subjectPublicKey        BIT STRING}

AlgorithmIdentifier ::= SEQUENCE{
    algorithm               OBJECT IDENTIFIER,
    parameters             ANY DEFINED BY algorithm OPTIONAL}

Certificates ::= SEQUENCE{
    certificate             Certificate,
    certificationPath       ForwardCertificationPath OPTIONAL}

ForwardCertificationPath ::= SEQUENCE OF CrossCertificates

CertificationPath ::= SEQUENCE{
    userCertificate         Certificate,
    theCACertificates       SEQUENCE OF CertificatePair
                                OPTIONAL}

CrossCertificates ::= SET OF Certificate

CertificateList ::= SIGNED SEQUENCE{
    signature               AlgorithmIdentifier,
    issuer                  Name,
    lastUpdate              UTCTime,
    revokedCertificates     SIGNEDSEQUENCE OF SEQUENCE{
        signature           AlgorithmIdentifier,
        issuer               Name,
        userCertificate      SerialNumber,
        revocationDate       UTCTime}
                                OPTIONAL}

CertificatePair ::= SEQUENCE{
    forward [0]             Certificate OPTIONAL,
    reverse [1]             Certificate OPTIONAL
    -- at least one of the pair must be present --}

--attribute types

UserCertificate ::= ATTRIBUTE
    WITH ATTRIBUTE-SYNTAXCertificate

CACertificate ::= ATTRIBUTE
    WITH ATTRIBUTE-SYNTAXCertificate

```

```

CrossCertificatePair      ::=  ATTRIBUTE
                             WITH ATTRIBUTE-SYNTAXCertificatePair

CertificateRevocationList ::=  ATTRIBUTE
                             WITH ATTRIBUTE-SYNTAXCertificateList

AuthorityRevocationList  ::=  ATTRIBUTE
                             WITH ATTRIBUTE-SYNTAXCertificateList

UserPassword              ::=  ATTRIBUTE
                             WITH ATTRIBUTE-SYNTAX
                             OCTETSTRING(SIZE(0...ub-user-password))
                             MATCHES FOR EQUALITY

```

-- macros

```

ALGORITHM MACRO          ::=
BEGIN
TYPE NOTATION            ::=  "PARAMETER" type
VALUE NOTATION           ::=  value(VALUE OBJECT IDENTIFIER)
END -- of ALGORITHM

```

```

ENCRYPTED MACRO          ::=
BEGIN
TYPE NOTATION           ::=  type (ToBeEnciphered)
VALUENOTATION           ::=  value (VALUE BIT STRING
-- the value of the bit string is generated by
-- taking the octets which form the complete
-- encoding (using the ASN.1 Basic Encoding Rules)
-- of the value of the ToBeEnciphered type and
-- applying an encipherment procedure to those octets--

```

END

```

SIGNED MACRO            ::=
BEGIN
TYPE NOTATION           ::=  type (ToBeSigned)
VALUE NOTATION          ::=  value(VALUE

```

```

SEQUENCE{
  ToBeSigned,
  AlgorithmIdentifier, -- of the algorithm used to generate the signature
  ENCRYPTED OCTET STRING
  -- where the octet string is the result
  -- of the hashing of the value of
  -- "ToBeSigned" --}
)

```

END -- of SIGNED

```

SIGNATURE MACRO        ::=
BEGIN
TYPE NOTATION          ::=  type (OfSignature)
VALUE NOTATION         ::=  value(VALUE

```

```

  SEQUENCE{
    AlgorithmIdentifier,
    -- of the algorithm used to compute the signature
    ENCRYPTED OCTET STRING
    -- where the octet string is a function (e.g. a compressed or hashed version)
    -- of the value "OfSignature", which may include the identifier of the
    -- algorithm used to compute the signature --}
  )

```

END -- of SIGNATURE

```

PROTECTED MACRO        ::=  SIGNATURE
END -- of Authentication Framework Definitions

```

ANNEX H

(to Recommendation X.509)

Reference Definition of algorithm object identifiers

This Annex is not an integral part of the Recommendation.

This Annex defines object identifiers assigned to authentication and encryption algorithms, in the absence of a formal register. It is intended to make use of such a register as it becomes available. The definitions take the form of the ASN.1 module, *AlgorithmObjectIdentifiers*.

```
AlgorithmObjectIdentifiers  {joint-iso-ccitt ds(5) modules(1)
                             algorithmObjectIdentifiers(8)}
DEFINITIONS ::=
BEGIN

EXPORTS
    encryptionAlgorithm, hashAlgorithm, signatureAlgorithm,
    rsa, squareMod-n, sqMod-nWithRSA;

IMPORTS
    algorithm, authenticationFramework
        FROM UsefulDefinitions(joint-iso-ccitt ds(5)modules(1)
                                usefulDefinitions(0))

    ALGORITHM FROM AuthenticationFramework authenticationFramework;

-- categories of object identifier

encryptionAlgorithm OBJECT IDENTIFIER ::= {algorithm 1}

hashAlgorithm OBJECT IDENTIFIER      ::= {algorithm 2}

signatureAlgorithm OBJECT IDENTIFIER  ::= {algorithm 3}

-- algorithms

rsa    ALGORITHM
      PARAMETER KeySize
      ::= {encryptionAlgorithm 1}

KeySize ::= INTEGER

sqMod-n ALGORITHM
      PARAMETER BlockSize
      ::= {hashAlgorithm 1}

BlockSize ::= INTEGER

sqMod-nWithRSA ALGORITHM
      PARAMETER KeyAndBlockSize
      ::= {signatureAlgorithm 1}

KeyAndBlockSize ::= INTEGER

END -- of Algorithm Object Identifier Definitions
```

LA GUIA - DEFINICION DEL SERVICIO ABSTRACTO ¹⁾

(Melbourne, 1988)

INDICE

- 0 Introducción
- 1 Alcance y campo de aplicación

SECCION 1 - *Generalidades*

- 2 Referencias
- 3 Definiciones
- 4 Abreviaturas
- 5 Convenios de descripción

SECCION 2 - *Servicio abstracto*

- 6 Visión de conjunto del servicio de Guía
- 7 Tipos de información
- 8 Operaciones de vincular y desvincular
- 9 Operaciones de lectura de la guía
- 10 Operaciones de búsqueda en la guía
- 11 Operación de modificación de la guía
- 12 Errores

Anexo A - Servicio abstracto en NSA.1

Anexo B - Identificadores de objeto de guía

¹⁾ La Recomendación X.511 y la norma ISO 9594-3, Information Processing Systems - Open Systems Interconnection - The Directory - Abstract Service Definition (Sistemas de procesamiento de información - Interconexión de sistemas abiertos - La guía - Definición del servicio abstracto) se redactaron en estrecha colaboración y están técnicamente alineadas.

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. The set of all 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 defines the capabilities provided by the Directory to its users.

0.4 Annex A provides the ASN.1 module which contains all the definitions associated with the abstract service.

1 Scope and field of application

1.1 This Recommendation defines in an abstract way the externally visible service provided by the Directory.

1.2 This Recommendation does not specify individual implementation or products.

SECTION 1 - General

2 References

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

Recommendation X.208 - Specification of Abstract Syntax Notation One (ASN.1).

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

Recommendation X.501 - The Directory - Models.

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

Recommendation X.519 - The Directory - Protocol Specifications.

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

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

Recommendation X.509 - The Directory - Authentication Framework.

Recommendation X.219 - Remote Operations - Model, Notation and Service Definition.

Recommendation X.229 - Remote Operations - Protocol Specification.

Recommendation X.407 - Abstract Service Definition Conventions.

3 Definitions

3.1 Basic Directory definitions

This Recommendation makes use of the following terms defined in Recommendation X.500:

- a) *Directory*;
- b) *Directory Information Base (DIB)*;
- c) *(Directory) User*.

3.2 Directory model definitions

This Recommendation makes use of the following terms defined in Recommendation X.501:

- a) *Directory System Agent*;
- b) *Directory User Agent*.

3.3 Directory information base definitions

This Recommendation makes use of the following terms defined in Recommendation X.501:

- a) *alias entry*;
- b) *Directory Information Tree*;
- c) *(Directory) entry*;
- d) *immediate superior*;
- e) *immediately superior entry/object*;
- f) *object*;
- g) *object class*;
- h) *object entry*;
- i) *subordinate*;
- j) *superior*.

3.4 Directory entry definitions

This Recommendation makes use of the following terms defined in Recommendation X.501:

- a) *attribute*;
- b) *attribute type*;
- c) *attribute value*;
- d) *attribute value assertion*.

3.5 Name definitions

This Recommendation makes use of the following terms defined in Recommendation X.501:

- a) *alias, alias name*;
- b) *distinguished name*;
- c) *(directory) name*;
- d) *purported name*;
- e) *relative distinguished name*.

3.6 Distributed operations definitions

This Recommendation makes use of the following terms defined in Recommendation X.518:

- a) *chaining*;
- b) *referral*.

3.7 Abstract service definitions

This Recommendation defines the following terms:

- a) *filter*: an assertion about the presence or value of certain attributes of an entry in order to limit the scope of a search;
- b) *service controls*: parameters conveyed as part of an abstract-operation which constrain various aspects of its performance;
- c) *originator*: the user that originated an operation.

4 Abbreviations

This Recommendation makes use of the following abbreviations:

AVA	Attribute Value Assertion
DIB	Directory Information Base
DIT	Directory Information Tree
DMD	Directory Management Domain
DSA	Directory System Agent
DUA	Directory User Agent
RDN	Relative Distinguished Name

5 Conventions

This Recommendation makes use of the abstract service definition conventions defined in Recommendation X.407.

SECTION 2 - Abstract service

6 Overview of the directory service

6.1 As described in Recommendation X.501 the services of the Directory are provided through access points to DUAs, each acting on behalf of a user. These concepts are depicted in Figure 1/X.511.

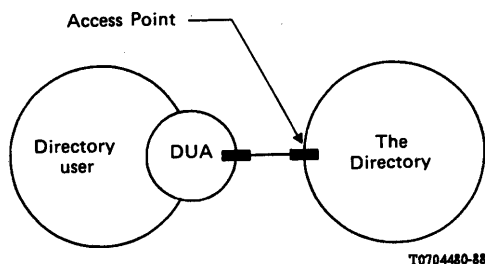


FIGURE 1/X.511

Access to the Directory

6.2 In principle, access points to the Directory may be of different types, providing different combinations of services. It is valuable to consider the Directory as an *object*, supporting a number of types of *port*. Each port defines a particular kind of interaction which the Directory can participate in with a DUA. Each access point corresponds to a particular combination of port types.

6.3 Using the notation defined in Recommendation X.407 the Directory can be defined as follows:

```
directory
  OBJECT
    PORTS { readPort [S],
             searchPort [S],
             modifyPort [S]}
  ::= id-ot-directory
```

The Directory supplies operations via: Read Ports, which support reading information from a particular named entry in the DIB; Search Ports, which allow more "exploration" of the DIB; and Modify Ports, which enable the modification of entries in the DIB.

Note - It is intended that in the future there may be other types of Directory port.

6.4 Similarly, a DUA (from the viewpoint of the Directory) can be defined as follows:

```
dua
  OBJECT
    PORTS { readPort [C],
             searchPort [C],
             modifyPort [C]}
  ::= id-ot-dua
```

The DUA consumes the services provided by the Directory.

6.5 The ports cited from §§ 6.2 to 6.4 can be defined as follows:

```
readPort
  PORT
    CONSUMER INVOKES {
      Read, Compare, Abandon}
  ::= id-pt-search

searchPort
  PORT
    CONSUMER INVOKES {
      List, Search}
  ::= id-pt-search

modifyPort
  PORT
    CONSUMER INVOKES {
      AddEntry, RemoveEntry,
      ModifyEntry, ModifyRDN}
  ::= id-pt-modify
```

6.6 The operations from the **readPort**, **searchPort** and the **modifyPort** are defined in §§ 9, 10, and 11 respectively.

6.7 These ports are used only as a method of structuring the description of the Directory service. Conformance to the Directory operations is specified in Recommendation X.519.

7 Information types

7.1 Introduction

7.1.1 This paragraph identifies, and in some cases defines, a number of information types which are subsequently used in the definition of Directory operations. The information types concerned are those which are common to more than one operation, are likely to be in the future, or which are sufficiently complex or self-contained as to merit being defined separately from the operation which uses them.

7.1.2 Several of the information types used in the definition of the Directory service are actually defined elsewhere. Paragraph 7.2 identifies types and indicates the source of their definition. Each of the remaining §§ (7.3 to 7.10) identifies and defines an information type.

7.2 Information types defined elsewhere

7.2.1 The following information types are defined in Recommendation X.501:

- a) **Attribute**;
- b) **AttributeType**;
- c) **AttributeValue**;
- d) **AttributeValueAssertion**;
- e) **DistinguishedName**;
- f) **Name**;
- g) **RelativeDistinguishedName**.

7.2.2 The following information type is defined in Recommendation X.520:

- a) **PresentationAddress**.

7.2.3 The following information types are defined in Recommendation X.509:

- a) **Certificate**;
- b) **SIGNED**;
- c) **CertificationPath**.

7.2.4 The following information type is defined in Recommendation X.219:

- a) **InvokeID**.

7.2.5 The following information types are defined in Recommendation X.518:

- a) **OperationProgress**;
- b) **ContinuationReference**.

7.3 Common arguments

7.3.1 The **CommonArguments** information may be present to qualify the invocation of each operation that the Directory can perform.

```
CommonArguments ::= SET {  
    [30] ServiceControls DEFAULT { },  
    [29] SecurityParameters DEFAULT { },  
    requestor [28] DistinguishedName  
        OPTIONAL,  
    [27] OperationProgress DEFAULT notStarted,  
    aliasedRDNs [26] INTEGER OPTIONAL,  
    extensions [25] SET OF EXTENSION OPTIONAL}
```

```
Extension ::= SET {  
    identifier [0] INTEGER,  
    critical [1] BOOLEAN DEFAULT FALSE,  
    item [2] ANY DEFINED BY identifier}
```

7.3.2 The various components have the meanings as defined in §§ 7.3.2.1 to 7.3.2.4.

7.3.2.1 The **ServiceControls** component is specified in § 7.5. Its absence is deemed equivalent to there being an empty set of controls.

7.3.2.2 The **SecurityParameters** component is specified in § 7.9. Its absence is deemed equivalent to there being an empty set of security parameters.

7.3.2.3 The **requestor DistinguishedName** identifies the originator of a particular abstract-operation. It holds the name of the user as identified at the time of binding to the Directory. It may be required when the request is to be signed (see § 7.10), and shall hold the name of the user who initiated the request.

7.3.2.4 The **OperationProgress** defines the role that the DSA is to play in the distributed evaluation of the request. It is more fully defined in Recommendation X.518.

7.3.2.5 The **aliasedRDNs** component indicates to the DSA that the object component of the operation was created by the dereferencing of an alias on an earlier operation attempt. The integer value indicates the number of RDNs in the object that came from dereferencing the alias. (The value would have been set in the referral response of the previous operation.)

7.3.2.6 The **extensions** component provides a mechanism to express standardized extensions to the form of the argument of a Directory abstract-operation.

Note - The form of the result of such an extended abstract-operation is identical to that of the non-extended version. (Nonetheless, the result of a particular extended abstract-operation may differ from its non-extended counterpart).

The subcomponents are as defined in §§ 7.3.2.6.1 to 7.3.2.6.3.

7.3.2.6.1 The **identifier** serves to identify a particular extension. Values of this component shall be assigned only by future versions of this series of Recommendations.

7.3.2.6.2 The **critical** subcomponent allows the originator of the extended abstract-operation to indicate that the performance of only the extended form of the abstract-operation is acceptable (i.e. that the non-extended form is not acceptable). In this case the extension is a *critical extension*. If the Directory, or some part of it, is unable to perform a critical extension it returns an indication of **unavailableCriticalExtension** (as a **ServiceError** or **PartialOutcomeQualifier**). If the Directory is unable to perform an extension which is not critical, it ignores the presence of the extension.

7.3.2.6.3 The **item** subcomponent provides the information needed for the Directory to perform the extended form of the abstract-operation.

7.4 Common results

7.4.1 The **CommonResults** information should be present to qualify the result of each retrieval operation that the Directory can perform.

```
CommonResults ::= SET {  
    [30] SecurityParameters OPTIONAL,  
    performer [29] DistinguishedName  
        OPTIONAL,  
    aliasDereferenced [28] BOOLEAN  
        DEFAULT FALSE}
```

7.4.2 The various components have the meanings as defined in §§ 7.4.2.1 to 7.4.2.3.

7.4.2.1 The **SecurityParameters** component is specified in § 7.9. Its absence is deemed equivalent to there being an empty set of security parameters.

7.4.2.2 The **performer DistinguishedName** identifies the performer of a particular operation. It may be required when the result is to be signed (see § 7.10), and shall hold the name of the DSA which signed the result.

7.4.2.3 The **aliasDereferenced** Component is set to **TRUE** when the purported name of an object or base object which is the target of the operation included on alias which was dereferenced.

7.5 Service controls

7.5.1 A **ServiceControls** parameter contains the controls, if any, that are to direct or constrain the provision of the service.

```
ServiceControls ::= SET {  
    options [0] BIT STRING {  
        preferChaining(0)  
        chainingProhibited (1),  
        localScope (2),  
        dontUseCopy (3),  
        dontDereferenceAliases(4))  
        DEFAULT {},  
    priority [1] INTEGER {  
        low (0),  
        medium (1),  
        high (2) } DEFAULT medium,
```

timeLimit [2] INTEGER OPTIONAL,
sizeLimit [3] INTEGER OPTIONAL,
scopeOfReferral [4] INTEGER {
 dmd(0),
 country(1)}
 OPTIONAL }

7.5.2 The various components have the meanings as defined in §§ 7.5.2.1 to 7.5.2.5.

7.5.2.1 The **options** component contains a number of indications, each of which, if set, asserts the condition suggested. Thus:

- a) **preferChaining** indicates that the preference is that chaining, rather than referrals, be used to provide the service. The Directory is not obliged to follow this preference;
- b) **chainingProhibited** indicates that chaining, and other methods of distributing the request around the Directory, are prohibited;
- c) **localScope** indicates that the operation is to be limited to a local scope. The definition of this option is itself a local matter. For example, within a single DSA or a single DMD;
- d) **dontUseCopy** indicates that copied information (as defined in Recommendation X.518) shall not be used to provide the service;
- e) **dontDereferenceAliases** indicate that any alias used to identify the entry affected by an operation is not to be dereferenced;

Note - This is necessary to allow reference to an alias entry itself rather than the aliased entry, e.g. in order to read the alias entry.

If this component is omitted, the following are assumed: no preference for chaining but chaining not prohibited, no limit on the scope of the operation, use of copy permitted, and aliases will be dereferenced (except for modify operations where aliases will never be dereferenced).

7.5.2.2 The **priority** (low, medium or high) at which the service is to be provided. Note that this is not a guaranteed service in that Directory, as a whole, does not implement queuing. There is no relationship implied with the use of "priorities" in underlying layers.

7.5.2.3 The **timeLimit** indicates the maximum elapsed time, in seconds, within which the service shall be provided. If the constraint cannot be met, an error is reported. If this component is omitted, no time limit is implied. In the case of time limit exceeded on a **List** or **Search**, the result is an arbitrary selection of the accumulated results.

Note - This component does not imply the length of time spent processing the request during the elapsed time: any number of DSAs may be involved in processing the request during the elapsed time.

7.5.2.4 The **sizeLimit** is only applicable to **List** and **Search** operations. It indicates the maximum number of objects to be returned. In the case of size limit exceeded, the results of **List** and **Search** may be an arbitrary selection of the accumulated results, equal in number to the size limit. Any further results shall be discarded.

7.5.2.5 The **scopeOfReferral** indicates the scope to which a referral returned by a DSA should be relevant. Depending on whether the value **dmd** or **country** are selected, only referrals to other DSAs within the selected scope will be returned.

This applies to the referrals in both a **ReferralError** and the **unexplored** parameter of **List** and **Search** results.

7.5.3 Certain combinations of **priority**, **timeLimit**, and **sizeLimit** may result in conflicts. For example, a short time limit could conflict with low priority; a high size limit could conflict with a low time limit, etc.

7.6 Entry information selection

7.6.1 An **EntryInformationSelection** parameter indicates what information is being requested from an entry in a retrieval service.

```

EntryInformationSelection ::= SET {
    attributeTypes
        CHOICE {
            allAttributes [0] NULL,
            select [1]] SET OF AttributeType
            -- empty set implies no attributes
            -- are requested --}
        DEFAULT allAttributes NULL,

    InfoTypes [2] INTEGER {
        attributeTypesOnly (0),
        attributeTypesAndValues (1) }
        DEFAULT attributeTypesAndValues }

```

7.6.2 The various components have the meanings as defined in §§ 7.6.2.1 and 7.6.2.2.

7.6.2.1 The **attributeTypes** component specifies the set of attributes about which information is requested:

- a) if the **select** option is chosen, then the attributes involved are listed. If the list is empty, then no attributes will be returned. Information about a selected attribute shall be returned if the attribute is present. An **AttributeError** with the **noSuchAttribute** problem shall only be returned if none of the attributes selected is present;
- b) if the **allAttributes** option is selected, then information is requested about all attributes in the entry.

Attribute information is only returned if access rights are sufficient. A **SecurityError** (with an **insufficientAccessRights** problem) will only be returned in the case where access rights preclude the reading of all attribute values requested.

7.6.2.2 The **infoTypes** component specifies whether both attribute type and attribute value information (the default) or attribute type information only is requested. If the **attributeTypes** component (§ 7.6.2.1) is such as to request no attributes, then this component is not meaningful.

7.7 Entry information

7.7.1 An **EntryInformation** parameter conveys selected information from an entry.

```

EntryInformation ::= SEQUENCE {
    DistinguishedName,
    fromEntry BOOLEAN DEFAULT TRUE,
    SET OF CHOICE {
        AttributeType,
        Attribute} OPTIONAL }

```

7.7.2 The **DistinguishedName** of the entry is always included.

7.7.3 The **fromEntry** parameter indicates whether the information was obtained from the entry (**TRUE**) or a copy of the entry (**FALSE**).

7.7.4 A set of **AttributeTypes** or **Attributes** are included, if relevant, each of which may be alone or accompanied by one or more attribute values.

7.8 Filter

7.8.1 A **Filter** parameter applies a test that is either satisfied or not by a particular entry. The filter is expressed in terms of assertions about the presence or value of certain attributes of the entry, and is satisfied if and only if it evaluates to **TRUE**.

Note - A Filter may be **TRUE**, **FALSE**, or undefined.

```

Filter ::= CHOICE {
    item [0] FilterItem,
    and [1] SET OF Filter,
    or [2] SET OF Filter,
    not [3] Filter }

```

```

FilterItem ::= CHOICE {
    equality      [0]  AttributeValueAssertion,
    substrings   [1]  SEQUENCE {
        type      AttributeType,
        strings   SEQUENCE OF CHOICE {
            initial [0]  AttributeValue,
            any     [1]  AttributeValue,
            final   [2]  AttributeValue}},
    greaterOrEqual [2]  AttributeValueAssertion,
    lessOrEqual    [3]  AttributeValueAssertion,
    present        [4]  AttributeType,
    approximateMatch [5] AttributeValueAssertion }

```

7.8.2 A **Filter** is either a **FilterItem** (see § 7.8.3), or an expression involving simpler **Filters** composed together using the logical operators **and**, **or**, and **not**. The **Filter** is undefined if it is a **FilterItem** which is undefined, or if it involves one or more simpler **Filters**, all of which are undefined. Otherwise, where the **Filter** is:

- a) an **item**, it is **TRUE** if and only if the corresponding **FilterItem** is **TRUE**;
- b) an **and**, it is **TRUE** unless any of the nested **Filters** is **FALSE**;
Note - Thus, if there are no nested **Filters** the **and** evaluates to **TRUE**.
- c) an **or**, it is **FALSE** unless any of the nested **Filters** is **TRUE**;
Note - Thus, if there are no nested **Filters** the **or** evaluates to **FALSE**.
- d) a **not**, it is **TRUE** if and only if the nested **Filter** is **FALSE**.

7.8.3 A **FilterItem** is an assertion about the presence or value(s) of an attribute of a particular type in the entry under test. Each such assertion is **TRUE**, **FALSE**, or undefined.

7.8.3.1 Every **FilterItem** includes an **AttributeType** which identifies the particular attribute concerned.

7.8.3.2 Any assertion about the value of such an attribute is only defined if the **AttributeType** is known, and the purported **AttributeValue(s)** conforms to the attribute syntax defined for that attribute type.

Note 1 - Where these conditions are not met the **FilterItem** is undefined.

Note 2 - Access control restrictions may require that the **FilterItem** be considered undefined.

7.8.3.3 Assertions about the value of an attribute are evaluated using the matching rules associated with the attribute syntax defined for that attribute type. A matching rule not defined for a particular attribute syntax cannot be used to make assertions about that attribute.

Note - Where this condition is not met, the **FilterItem** is undefined.

7.8.3.4 A **FilterItem** may be undefined (as described in §§ 7.8.3.2 and 7.8.3.3 above). Otherwise, where the **FilterItem** asserts:

- a) **equality**, it is **TRUE** if and only if there is a value of the attribute which is equal to that asserted;
- b) **substrings**, it is **TRUE** if and only if there is a value of the attribute in which the specified substrings appear in the given order. The substrings shall be non-overlapping, and may (but need not) be separated from the ends of the attribute value and from one another by zero or more string elements.

If **initial** is present, the substring shall match the initial substring of the attribute value; if **final** is present, the substring shall match the final substring of the attribute value; if **any** is present, the substring may match any substring in the attribute value;

- c) **greaterOrEqual**, it is **TRUE** if and only if the relative ordering (as defined by the appropriate ordering algorithm) places the supplied value before or equal to any value of the attribute;
- d) **lessOrEqual**, it is **TRUE** if and only if the relative ordering (as defined by the appropriate ordering algorithm) places the supplied value after or equal to any value of the attribute;
- e) **present**, it is **TRUE** if and only if such an attribute is present in the entry;
- f) **approximateMatch**, it is **TRUE** if and only if there is a value of the attribute which matches that which is asserted by some locally-defined approximate matching algorithm (e.g. spelling variations, phonetic match, etc.). There are no specific guidelines for approximate matching in this version of the Recommendation. If approximate matching is not supported, this **FilterItem** should be treated as a match for **equality**.

7.9 Security Parameters

7.9.1 The **SecurityParameters** govern the operation of various security features associated with a Directory operation.

Note - These parameters are conveyed from sender to recipient. Where the parameters appear in the argument of an abstract-operation the requestor is the sender, and the performer is the recipient. In a result, the roles are reversed.

```

SecurityParameters ::= SET {
    certification-path [0]
    CertificationPath OPTIONAL,
    name [1] DistinguishedName
    OPTIONAL,
    time [2] UTCTime OPTIONAL,
    random [3] BIT STRING OPTIONAL,
    target [4] ProtectionRequest OPTIONAL
}

ProtectionRequest ::= INTEGER {
    none(0),
    signed(1)}

```

7.9.2 The various components have the meanings as defined in §§ 7.9.2.1 to 7.9.2.5.

7.9.2.1 The **CertificationPath** component consists of the sender's certificate, and, optionally, a sequence of certificate pairs. The certificate is used to associate the sender's public key and distinguished name, and may be used to verify the signature on the argument or result. This parameter shall be present if the argument or result is signed. The sequence of certification pairs consists of certification authority cross certificates. It is used to enable the sender's certificate to be validated. It is not required if the recipient shares the same certification authority as the sender. If the recipient requires a valid set of certificate pairs, and this parameter is not present, whether the recipient rejects the signature on the argument or result, or attempts to generate the certification path, is a local matter.

7.9.2.2 The **name** is the distinguished name of the first intended recipient of the argument or result. For example, if a DUA generates a signed argument, the name is the distinguished name of the DSA to which the operation is submitted.

7.9.2.3 The **time** is the intended expiry time for the validity of the signature, when signed arguments are used. It is used in conjunction with the random number to enable the detection of replay attacks.

7.9.2.4 The **random** component is a number which should be different for each unexpired token. It is used in conjunction with the time parameter to enable the detection of replay attacks when the argument or result has been signed.

7.9.2.5 The **target ProtectionRequest** may appear only in the request for an operation to be carried out, and indicates the requestor's preference regarding the degree of protection to be provided to the result. Two levels are provided: **none** (no protection requested), and **signed** (the Directory is requested to sign the result, the default). The degree of protection actually provided to the result is indicated by the form of result and may be equal to or lower than that requested, based on the limitations of the Directory.

7.10 OPTIONALLY-SIGNED

7.10.1 An **OPTIONALLY-SIGNED** information type is one whose values may, at the option of the generator, be accompanied by their digital signature. This capability is specified by means of the following macro:

```
OPTIONALLY-SIGNED MACRO ::=
BEGIN
TYPE NOTATION ::= type (Type)
VALUE NOTATION ::= value (VALUE
CHOICE { Type, SIGNED Type))
END
```

7.10.2 The **SIGNED** macro, which describes the form of the signed form of the information, is specified in Recommendation X.509.

8 Bind and unbind operations

The **DirectoryBind** and **DirectoryUnbind** operations, defined in § 8.1 and § 8.2 respectively, are used by the DUA at the beginning and end of a particular period of accessing the Directory.

8.1 Directory bind

8.1.1 A **DirectoryBind** operation is used at the beginning of a period of accessing the Directory.

```
DirectoryBind ::= ABSTRACT-BIND
TO { readPort, searchPort, modifyPort }
BIND
ARGUMENT DirectoryBindArgument
RESULT DirectoryBindResult
BIND-ERROR DirectoryBindError

DirectoryBindArgument ::= SET {
credentials [0] Credentials OPTIONAL,
versions [1] Versions DEFAULT
v1988}

Credentials ::= CHOICE {
simple [0] SimpleCredentials,
strong [1] StrongCredentials,
externalProcedure [2] EXTERNAL }

SimpleCredentials ::= SEQUENCE {
name [0] DistinguishedName,
validity [1] SET {
time1 [0] UTCTime OPTIONAL,
Time2 [1] UTCTime OPTIONAL,
random1 [2] BIT STRING OPTIONAL,
random2 [3] BIT STRING OPTIONAL } OPTIONAL,
-- in most instances the argument for
-- time and random are relevant in
-- dialogues employing protected password
-- mechanisms and derive their meaning
-- as per bilateral agreements
password [2] OCTET STRING OPTIONAL }
-- the value could be an unprotected
-- password or Protected1 or Protected2
-- as specified in Recommendation X.509.

StrongCredentials ::= SET {
certification-path[0] CertificationPath
OPTIONAL,
bind-token [1] Token }
```

```

Token      ::= SIGNED SEQUENCE {
    algorithm [0] AlgorithmIdentifier,
    name      [1] DistinguishedName,
    time      [2] UTCTime,
    random    [3] BIT STRING }

Versions   ::= BIT STRING {v1988(0)}

DirectoryBindResult ::= DirectoryBindArgument

DirectoryBindError ::= SET {
    versions [0] Versions DEFAULT v1988,
    CHOICE {
        serviceError [1] ServiceProblem
        securityError [2] SecurityProblem
    }
}

```

8.1.2 The various arguments have the meanings as defined in §§ 8.1.2.1 to 8.1.2.2.

8.1.2.1 The **Credentials** of the **DirectoryBindArgument** allow the Directory to establish the identity of the user. They may be either **simple**, **strong** (as described in Recommendation X.509) or externally defined (**externalProcedure**).

8.1.2.1.1 **SimpleCredentials** consist of a name (always the distinguished name of an object) and (optionally) a password. This provides a limited degree of security. If the password is protected as described in § 5 of Recommendation X.509, then **SimpleCredentials** includes name, password and (optionally) time and/or random numbers which are used to detect replay. In some instances a protected password may be checked by an object which knows the password only after locally regenerating the protection to its own copy of the password and computing the result with the value in the bind argument (password). In other instances a direct compare may be possible.

8.1.2.1.2 **StrongCredentials** consist of a bind token and, optionally, a certificate and sequence of certification-authority cross-certificate (as defined in Recommendation X.509). This enables the Directory to authenticate the identity of the request establishing the association, and vice versa.

The arguments of the bind token are used as follows: **algorithm** is the identifier of the algorithm employed to sign the information; **name** is the name of the intended recipient. The **time** parameter contains the expiry time of the token. The **random** number is a number which should be different for each unexpired token, and may be used by the recipient to detect replay attacks.

8.1.2.1.3 If **externalProcedure** is used then the semantics of the authentication scheme being used is outside the scope of the Directory document.

8.1.2.2 The **Versions** argument of the **DirectoryBindArgument** identifies the versions of the service which the DUA is prepared to participate in. For this version of the protocol the value shall be set to v1988(0).

8.1.2.3 Migration to future versions of the Directory should be facilitated by:

- a) any elements of **DirectoryBindArgument** other than those defined in this Recommendation shall be accepted and ignored;
- b) additional options for named bits of **DirectoryBindArgument** (e.g. **Versions**) not defined shall be accepted and ignored.

8.1.3 Should the bind request succeed, a result will be returned. The result parameters have the meanings as defined in §§ 8.1.3.1 and 8.1.3.2.

8.1.3.1 The **Credentials** of the **DirectoryBindResult** allow the user to establish the identity of the DSA. They allow information identifying the DSA (that is directly providing the Directory service) to be conveyed to the DUA. They shall be of the same form (i.e. **CHOICE**) as those supplied by the user.

8.1.3.2 The **Versions** parameter of the **DirectoryBindResult** indicates which of the versions of the service requested by the DUA is actually going to be provided by this DSA.

8.1.4 Should the bind request fail, a bind error will be returned as defined in §§ 8.1.4.1 and 8.1.4.2.

8.1.4.1 The **Versions** parameter of the **DirectoryBindError** indicates which versions are supported by this DSA.

8.1.4.2 A **securityError** or **serviceError** shall be supplied as follows:

- **securityError** **inappropriateAuthentication**
 invalidCredentials
- **serviceError** **unavailable.**

8.2 *Directory unbind*

8.2.1 A **DirectoryUnbind** operation is used at the end of a period of accessing the Directory.

DirectoryUnbind ::= **ABSTRACT-UNBIND**
FROM {**readPort**, **searchPort**, **modifyPort** }

8.2.2 The **DirectoryUnbind** has no arguments.

9 **Directory read operations**

There are two "read-like" operations: **Read** and **Compare**, defined in §§ 9.1 and 9.2, respectively. The **Abandon** operation, defined in § 9.3, is grouped with the **Read** operations for convenience.

9.1 *Read*

9.1.1 A **Read** operation is used to extract information from an explicitly identified entry. It may also be used to verify a distinguished name. The arguments of the operation may optionally be signed (see § 7.10) by the requestor. If so requested, the Directory may sign the result.

Read ::= **ABSTRACT-OPERATION**
ARGUMENT **ReadArgument**
RESULT **ReadResult**
ERRORS {
 AttributeError, **NameError**,
 ServiceError, **Referral**, **Abandoned**,
 SecurityError }

ReadArgument ::= **OPTIONALLY-SIGNED SET** {
 object [0] **Name**,
 selection [1] **Selection F¹³ EntryInformationSelection**
 DEFAULT {}
 COMPONENTS OF CommonArguments }

ReadResult ::= **OPTIONALLY-SIGNED SET** {
 entry [0] **EntryInformation**,
 COMPONENTS OF CommonResults }

9.1.2 The various arguments have the meanings as defined in §§ 9.1.2.1 to 9.1.2.3.

9.1.2.1 The **object** argument identifies the object entry from which the information is requested. Should the Name involve one or more aliases, they are dereferenced (unless this is prohibited by the relevant service controls).

9.1.2.2 The **selection** argument indicates what information from the entry is requested (see § 7.6).

9.1.2.3 The **CommonArguments** (see § 7.3) include a specification of the service controls applying to the request. For the purposes of this operation the **sizeLimit** component is not relevant and is ignored if provided.

9.1.3 Should the request succeed, the result will be returned. The result parameters have the meanings as defined in § 9.1.3.1 and § 7.4.

9.1.3.1 The **entry** result parameter holds the requested information (see § 7.7).

9.1.4 Should the request fail, one of the listed errors will be reported. If none of the attributes explicitly listed in **selection** can be returned, then an **AttributeError** with problem **noSuchAttribute** will be reported. The circumstances under which other errors will be reported are defined in § 12.

9.2 Compare

9.2.1 A **Compare** operation is used to compare a value (which is supplied as an argument of the request) with the value(s) of a particular attribute type in a particular object entry. The arguments of the operation may optionally be signed (see § 7.10) by the requestor. If so requested, the Directory may sign the result.

```
Compare ::= ABSTRACT-OPERATION
  ARGUMENT CompareArgument
  RESULT CompareResult
  ERRORS {
    AttributeError, NameError,
    ServiceError, Referral, Abandoned,
    SecurityError }

CompareArgument ::= OPTIONALLY-SIGNED
SET {
  object      [0] Name,
  purported   [1] AttributeValueAssertion,
  COMPONENTS OF CommonArguments }

CompareResult  ::= OPTIONALLY-SIGNED
SET {
  DistinguishedName OPTIONAL,
  matched          [0] BOOLEAN,
  from Entry      [1] BOOLEAN DEFAULT TRUE,
  COMPONENTS OF CommonResults }
```

9.2.2 The various arguments have the meanings as defined in §§ 9.2.2.1 to 9.2.2.3.

9.2.2.1 The **object** argument is the name of the particular object entry concerned. Should the **Name** involve one or more aliases, they are dereferenced (unless prohibited by the relevant service control).

9.2.2.2 The **purported** argument identifies the attribute type and the value to be compared with that in the entry.

9.2.2.3 The **CommonArguments** (see § 7.3) specify the service controls applying to the request. For the purposes of this operation the **sizeLimit** component is not relevant and is ignored, if provided.

9.2.3 Should the request succeed (i.e. the comparison is actually carried out), the result will be returned. The result parameters have the meanings as described in § 9.2.3.1, § 9.2.3.2 and § 7.4.

9.2.3.1 The **DistinguishedName** is present if an alias was dereferenced and represents the distinguished name of the object itself.

9.2.3.2 The **matched** result parameter, holds the result of the comparison. The parameter takes the value **TRUE** if the values were compared and matched, and **FALSE** if they did not.

9.2.3.3 If **fromEntry** is **TRUE** the information was compared against the entry; if **FALSE** some of the information was compared against a copy.

9.2.4 Should the request fail, one of the listed errors will be reported. The circumstances under which the particular errors will be reported are defined in § 12.

9.3 Abandon

9.3.1 Operations that interrogate the Directory may be abandoned using the **Abandon** operation if the user is no longer interested in the result.

```
Abandon ::= ABSTRACT-OPERATION
  ARGUMENT AbandonArgument
  RESULT AbandonResult
  ERRORS {AbandonFailed}

AbandonArgument ::= SEQUENCE {
  InvokeID [0] InvokeID}

AbandonResult ::= NULL
```

9.3.2 There is a single argument, the **InvokeID** which identifies the operation that is to be abandoned. The value of the **invokeID** is the same **invokeID** which was used to invoke the operation which is to be abandoned.

9.3.3 Should the request succeed, a result will be returned, although no information will be conveyed with it. The original operation will fail with an **Abandoned** error.

9.3.4 Should the request fail, the **AbandonFailed** error will be reported. This error is described in § 12.3.

9.3.5 **Abandon** is only applicable to interrogation operations, i.e., **Read**, **Compare**, **List** and **Search**.

9.3.6 A DSA may abandon an operation locally. If the DSA has chained or multicasted the operation to other DSAs, it may in turn request them to abandon the operation. A DSA may choose not to abandon the operation and shall then return the **AbandonFailed** error.

10 Directory search operations

There are two "search-like" operations: **List** and **Search**, defined in § 10.1 and § 10.2 respectively.

10.1 List

10.1.1 A **List** operation is used to obtain a list of the immediate subordinates of an explicitly identified entry. Under some circumstances, the list returned may be incomplete. The arguments of the operation may optionally be signed (see § 7.10) by the requestor. If so requested, the Directory may sign the result.

```
List ::= ABSTRACT-OPERATION
        ARGUMENT      ListArgument
        RESULT        ListResult
        ERRORS {
            NameError
            ServiceError, Referral, Abandoned,
            SecurityError }

List Argument ::= OPTIONALLY-SIGNED SET {
    object [0]      Name,
    COMPONENTS OF CommonArguments }

ListResult ::= OPTIONALLY-SIGNED
CHOICE {
    listInfo SET {
        DistinguishedName OPTIONAL,
        subordinates [1]    SET OF SEQUENCE {
            RelativeDistinguishedName,
            aliasEntry [0]    BOOLEAN DEFAULT FALSE
            fromEntry [1]    BOOLEAN DEFAULT TRUE},
        partialOutcomeQualifier [2]
            PartialOutcomeQualifier OPTIONAL
        COMPONENTS OF CommonResults },
    uncorrelatedListInfo [0] SET OF
        ListResult }

PartialOutcomeQualifier ::= SET {
    limitProblem [0]    LimitProblem
        OPTIONAL,
    unexplored [1]    SET OF
        ContinuationReference OPTIONAL,
    unavailableCriticalExtensions [2] BOOLEAN DEFAULT FALSE }

LimitProblem ::= INTEGER {
    timeLimitExceeded (0),
    sizeLimitExceeded (1),
    administrativeLimitExceeded (2) }
```

10.1.2 The various arguments have the meanings as defined in § 10.1.2.1 and § 7.3.

10.1.2.1 The **object** argument identifies the object entry (or possibly the root) whose immediate subordinates are to be listed. Should the **Name** involve one or more aliases, they are dereferenced (unless prohibited by the relevant service control).

10.1.3 The request succeeds if the object is located regardless of whether there is any subordinate information to return. The result parameters have the meanings as defined in §§ 10.1.3.1 to 10.1.3.4 and § 7.4.

10.1.3.1 The **DistinguishedName** is present if an alias was dereferenced. It represents the distinguished name of the object itself.

10.1.3.2 The **subordinates** parameter conveys the information on the immediate subordinate, if any, of the named entry. Should any of the subordinate entries be aliases, they will not be dereferenced.

10.1.3.2.1 The **RelativeDistinguishedName** is that of the subordinate.

10.1.3.2.2 The **fromEntry** parameter indicates whether the information was obtained from the entry (**TRUE**) or a copy of the entry (**FALSE**).

10.1.3.2.3 The **aliasEntry** parameter indicates whether the subordinate entry is an alias entry (**TRUE**) or not (**FALSE**).

10.1.3.3 The **PartialOutcomeQualifier** consists of three subcomponents as defined in §§ 10.1.3.3.1 to 10.1.3.3.3. This parameter shall be present whenever the result is incomplete.

10.1.3.3.1 The **LimitProblem** parameter indicates whether the time limit, the size limit, or an administrative limit has been exceeded. The results being returned are those which were available when the limit was reached.

10.1.3.3.2 The **unexplored** parameter shall be present if regions of the DIT were not explored. Its information allows the DUA to continue the processing of the **List** operation by contacting other access points if it so chooses. The parameter consists of a set (possibly empty) of **ContinuationReferences**, each consisting of the name of a base object from which the operation may be progressed, an appropriate value of **OperationProgress**, and a set of access points from which the request may be further progressed. The **ContinuationReferences** that are returned shall be within the scope of referral requested in the operation service control.

10.1.3.3.3 The **unavailableCriticalExtensions** parameter indicates, if present, that one or more critical extensions were unavailable in some part of the Directory.

10.1.3.4 When the DUA has requested a protection request of **signed**, the **uncorrelatedListInfo** parameter may comprise a number of sets of result parameters originating from and signed by different components of the Directory. If no DSA in the chain can correlate all the results, the DUA must assemble the actual result from the various pieces.

10.1.4 Should the request fail, one of the listed errors will be reported. The circumstances under which the particular errors will be reported are defined in § 12.

10.2 Search

10.2.1 A **Search** operation is used to search a portion of the DIT for entries of interest and to return selected information from those entries. The arguments of the operation may optionally be signed (see § 7.10) by the requestor. If so requested, the Directory may sign the result.

```
Search ::= ABSTRACT-OPERATION
          ARGUMENT SearchArgument
          RESULT SearchResult
          ERRORS {
              AttributeError, NameError,
              ServiceError, Referral, Abandoned,
              SecurityError }

```

```
SearchArgument ::= OPTIONALLY-SIGNED
SET {
    baseObject      [0] Name,
    subset          [1] INTEGER {
        baseObject (0),
        oneLevel(1),
        wholeSubtree(2)) DEFAULT baseObject,
    filter          [2] Filter DEFAULT and {}.

```

searchAliases [3] BOOLEAN DEFAULT TRUE,
 selection [4] EntryInformationSelection DEFAULT {}
 COMPONENTS OF CommonArguments }

SearchResult ::= OPTIONALLY-SIGNED
 CHOICE {
 searchInfo SET {
 DistinguishedName OPTIONAL,
 entries [0] SET OF EntryInformation,
 partialOutcomeQualifier
 [2]PartialOutcomeQualifier OPTIONAL,
 COMPONENTS OF CommonResults },
 uncorrelatedSearchInfo [0] SET OF
 SearchResult }

10.2.2 The various arguments have the meanings as defined in §§ 10.2.2.1 to 10.2.2.3, § 10.2.2.5, and § 7.3.

10.2.2.1 The **baseObject** argument identifies the object entry (or possibly the root) relative to which the search is to take place.

10.2.2.2 The **subset** argument indicates whether the search is to be applied to:

- a) the **baseObject** only;
- b) the immediate subordinates of the base object only (**oneLevel**);
- c) the base object and all its subordinates (**wholeSubtree**).

10.2.2.3 The **filter** argument is used to eliminate entries from the search space which are not of interest. Information will only be returned on entries which satisfy the filter (see § 7.8).

10.2.2.4 Aliases shall be dereferenced while locating the base object, subject to the setting of the **dontDereferenceAliasesServiceControl**. Aliases among the subordinates of the base object shall be dereferenced during the search, subject to the setting of the **searchAliases** parameter. If the **searchAliases** parameter is **TRUE**, aliases shall be dereferenced, if the parameter is **FALSE**, aliases shall not be dereferenced. If the **searchAliases** parameter is **TRUE**, the search shall continue in the subtree of the aliased object.

10.2.2.5 The **selection** argument indicates what information from the entries is requested (see § 7.6).

10.2.3 The request succeeds if the base object is located, regardless of whether there are any subordinates to return.

Note - As a corollary to this, the outcome of an (unfiltered) **Search** applied to a single entry may not be identical to a **Read** which seeks to interrogate the same set of attributes of the entry. This is because the latter will return an **AttributeError** if none of the selected attributes exist in the entry.

The result parameters have the meanings as defined in §§ 10.2.3.1 to 10.2.3.4 and § 7.3.

10.2.3.1 The **DistinguishedName** is present if an alias was dereferenced, and represents the distinguished name of the base object.

10.2.3.2 The **entries** parameter conveys the requested information from each entry (zero or more) which satisfied the filter (see § 7.5).

10.2.3.3 The **PartialOutcomeQualifier** consists of two subcomponents as described for the **List** operation in § 10.1.3.4.

10.2.3.4 The **uncorrelatedSearchInfo** parameter is as described for **uncorrelatedListInfo** in § 10.1.3.4.

10.2.4 Should the request fail, one of the listed errors will be reported. The circumstances under which the particular errors will be reported are defined in § 12.

11 Directory modify operations

There are four operations to modify the Directory: **AddEntry**, **RemoveEntry**, **ModifyEntry** and **ModifyRDN** defined in §§ 11.1 to 11.4 respectively.

Note 1 - Each of these abstract-operations identifies the target entry by means of its *distinguished name*.

Note 2 - The success of **AddEntry**, **RemoveEntry**, and **ModifyRDN** operations will be dependent on the physical distribution of the DIB across the Directory. Failure will be reported with an **UpdateError** and problem affects **MultipleDSAs**. See Recommendation X.518.

11.1 Add entry

11.1.1 An **AddEntry** operation is used to add a leaf entry (either an object entry, or an alias entry) to the DIT. The arguments of the operation may optionally be signed (see § 7.10) by the requestor.

```
AddEntry ::= ABSTRACT-OPERATION
  ARGUMENT  AddEntryArgument
  RESULT    AddEntryResult
  ERRORS {
    AttributeError, NameError,
    ServiceError, Referral, SecurityError,
    UpdateError }

AddEntryArgument ::= OPTIONALLY-SIGNED
SET {
  object      [0]  DistinguishedName,
  entry       [1]  SET OF Attribute,
  COMPONENTS OF CommonArguments }

AddEntryResult ::= NULL
```

11.1.2 The various arguments have the meanings as defined in §§ 11.1.2.1 to 11.1.2.3.

11.1.2.1 The **object** argument identifies the entry to be added. Its immediate superior, which must already exist for the operation to succeed, can be determined by removing the last RDN component (which belongs to the entry to be created).

11.1.2.2 The **entry** argument contains the attribute information which, together with that from the RDN, constitutes the entry to be created. The Directory shall ensure that the entry conforms to the Directory schema. Where the entry being created is an alias, no check is made to ensure that the **aliasedObjectName** attribute points to a valid entry.

11.1.2.3 The **CommonArguments** (see § 7.3) include a specification of the service controls applying to the request. For the purposes of this operation the **dontDereferenceAlias** option and the **sizeLimit** component are not relevant and are ignored if provided. Aliases are never dereferenced by this operation.

11.1.3 Should the request succeed, a result will be returned, although no information will be conveyed with it.

11.1.4 Should the request fail, one of the listed errors will be reported. The circumstances under which the particular errors will be reported are defined in § 12.

11.2 Remove Entry

11.2.1 A **RemoveEntry** operation is used to remove a leaf entry (either an object entry or an alias entry) from the DIT. The arguments of the operation may optionally be signed (see § 7.10) by the requestor.

```
RemoveEntry ::= ABSTRACT-OPERATION
  ARGUMENT  RemoveEntryArgument
  RESULT    RemoveEntryResult
  ERRORS {
    NameError,
    ServiceError, Referral, SecurityError,
    UpdateError}
```

RemoveEntryArgument ::= OPTIONALLY-SIGNED SET {
 object [0] **DistinguishedName,**
 COMPONENTS OF CommonArguments }

RemoveEntryResult ::= NULL

11.2.2 The various arguments have the meanings as defined in §§ 11.2.2.1 and 11.2.2.2.

11.2.2.1 The **object** argument identifies the entry to be deleted. Aliases in the name will not be dereferenced.

11.2.2.2 The **CommonArguments** (see § 7.3) include a specification of the service controls applying to the request. For the purposes of this operation the **dontDereferenceAlias** option and the **sizeLimit** component are not relevant and are ignored if provided. Aliases are never dereferenced by this operation.

11.2.3 Should the request succeed, a result will be returned, although no information will be conveyed with it.

11.2.4 Should the request fail, one of the listed errors will be reported. The circumstances under which the particular errors will be reported are defined in § 12.

11.3 *Modify Entry*

11.3.1 The **ModifyEntry** operation is used to perform a series of one or more of the following modifications to a single entry:

- a) add a new attribute;
- b) remove an attribute;
- c) add attribute values;
- d) remove attribute values;
- e) replace attribute values;
- f) modify an alias.

The arguments of the operation may optionally be signed (see § 7.10) by the requestor.

ModifyEntry ::= **ABSTRACT-OPERATION**
 ARGUMENT **ModifyEntryArgument**
 RESULT **ModifyEntryResult**
 ERRORS {
 AttributeError, NameError,
 ServiceError, Referral, SecurityError,
 UpdateError }

ModifyEntryArgument ::= **OPTIONALLY-SIGNED SET {**
 object [0] **DistinguishedName,**
 changes [1] **SEQUENCE OF EntryModification,**
 COMPONENTS OF CommonArguments }

ModifyEntryResult ::= NULL

EntryModification ::= CHOICE {
 addAttribute [0] **Attribute,**
 removeAttribute [1] **AttributeType,**
 addValues [2] **Attribute,**
 removeValues [3] **Attribute }**

11.3.2 The various arguments have the meanings as defined in §§ 11.3.2.1 and 11.3.2.2.

11.3.2.1 The **object** argument identifies the entry to which the modifications should be applied. Any aliases in the name will not be dereferenced.

11.3.2.2 The **changes** argument defines a sequence of modifications, which are applied in the order specified. If any of the individual modifications fails, then an **AttributeError** is generated and the entry left in the state it was prior to the operation. That is, the operation is atomic. The end result of the sequence of modifications shall not violate the Directory schema. However, it is possible, and sometimes necessary, for the individual **EntryModification** changes to appear to do so. The following types of modification may occur:

- a) **addAttribute**: This identifies a new attribute to be added to the entry, which is fully specified by the argument. Any attempt to add an already existing attribute results in an **AttributeError**;
- b) **removeAttribute**: The argument identifies (by its type) an attribute to be removed from the entry. Any attempt to remove a non-existing attribute results in an **AttributeError**;

Note - This operation is not allowed if the attribute type is present in the RDN.

- c) **addValues**: This identifies an attribute by the attribute type in the argument, and specifies one or more attribute values to be added to the attribute. An attempt to add an already existing value results in an error. An attempt to add a value to a non-existent type results in an error;
- d) **removeValues**: This identifies an attribute by the attribute type in the argument and specifies one or more attribute values to be removed from the attribute. If the values are not present in the attribute, this results in an **AttributeError**. If an attempt is made to modify the object class attribute, an update error is returned.

Note - This operation is now allowed if one of the values is present in the RDN.

Values may be replaced by a combination of **addValues** and **removeValues** in a single **ModifyEntry** operation.

11.3.2.3 The **CommonArguments** (see § 7.3) include a specification of the service controls applying to the request. For the purposes of this operation the **dontDereferenceAlias** option and the **sizeLimit** component are not relevant and are ignored if provided. Aliases are never dereferenced by this operation.

11.3.3 Should the request succeed, a result will be returned although no information will be conveyed with it.

11.3.4 Should the request fail, one of the listed errors will be reported. The circumstances under which the particular errors will be reported are defined in § 12.

11.4 *Modify RDN*

11.4.1 The **ModifyRDN** operation is used to change the Relative Distinguished Name of a leaf entry (either an object entry or an alias entry) in the DIT. The arguments of the operation may optionally be signed (see § 7.10) by the requestor.

```

ModifyRDN ::=      ABSTRACT-OPERATION
                    ARGUMENT ModifyRDNArgument
                    RESULT   ModifyRDNResult
                    ERRORS {
                        NameError,
                        ServiceError, Referral, SecurityError,
                        UpdateError }

ModifyRDNArgument ::= OPTIONALLY-SIGNED SET {
    object          [0]   DistinguishedName,
    newRDN          [1]   RelativeDistinguishedName,
    deleteOldRDN   [2]   BOOLEAN DEFAULT FALSE,
    COMPONENTS OF CommonArguments }

ModifyRDNResult   ::=      NULL

```

11.4.2 The various parameters have the meanings as defined in §§ 11.4.2.1 to 11.4.2.5.

11.4.2.1 The **object** argument identifies the entry whose Relative Distinguished Name is to be modified. Aliases in the name will not be dereferenced. The immediate superior entry shall not have any Non-Specific Subordinate References (see Recommendation X.518).

11.4.2.2 The **newRDN** argument specifies the new RDN of the entry.

11.4.2.3 If an attribute value in the new RDN does not already exist in the entry (either as part of the old RDN or as a non-distinguished value) it is added. If it cannot be added, an error is returned.

11.4.2.4 If the **deleteOldRDN** flag is set, all attribute values in the old RDN which are not in the new RDN are deleted. If this flag is not set, the old values should remain in the entry (not as a part of the RDN). The flag shall be set where a single value attribute in the RDN has its value changed by the operation. If this operation removes the last attribute value of an attribute, that attribute shall be deleted.

11.4.2.5 The **Common Arguments** (see § 7.3) include a specification of the service controls applying to the request. For the purposes of this operation the **dontDereferenceAlias** option and the **sizeLimit** component are not relevant and are ignored if provided. Aliases are never dereferenced by this operation.

11.4.3 Should the request succeed, a result will be returned, although no information will be conveyed with it.

11.4.4 Should the request fail, one of the listed errors will be reported. The circumstances under which the particular errors will be returned are defined in § 12.

11.4.5 As defined in this Recommendation this operation may only be used on a leaf entry.

12 Errors

12.1 Error Precedence

12.1.1 The Directory does not continue to perform an operation beyond the point at which it determines that an error is to be reported.

Note 1 - An implication of this rule is that the first error encountered can differ for repeated instances of the same query, as there is not a specific logical order in which to process a given query. For example, DSAs may be searched in different orders.

Note 2 - The rules of error precedence specified here apply only to the abstract service provided by the Directory as a whole. Different rules apply when the internal structure of the Directory is taken into account.

12.1.2 Should the Directory simultaneously detect more than one error, the following list determines which error is reported. An error higher in the list has a higher logical precedence than one below it and is the error which is reported.

- a) **NameError**
- b) **UpdateError**
- c) **AttributeError**
- d) **SecurityError**
- e) **ServiceError**.

12.1.3 The following errors do not present any precedence conflicts:

- a) **AbandonFailed**, because it is specific to one operation, **Abandon**, which can encounter no other error;
- b) **Abandoned**, which is not reported if an **Abandon** operation is received simultaneously with the detection of an error. In this case an **AbandonFailed** error, reporting the problem **tooLate** is reported along with the report of the actual error encountered;
- c) **Referral**, which is not a "real" error, only an indication that the Directory has detected that the DUA must present its request to another access point.

12.2 Abandoned

12.2.1 This outcome may be reported for any outstanding directory enquiry operation (i.e. **Read**, **Search**, **Compare**, **List**) if the DUA invokes an **Abandon** operation with the appropriate **InvokeID**.

Abandoned ::= **ABSTRACT-ERROR** -- *not literally an "error"*

12.2.2 There are no parameters associated with this error.

12.3 Abandon Failed

12.3.1 The **AbandonFailed** error reports a problem encountered during an attempt to abandon an operation.

AbandonFailed ::= ABSTRACT-ERROR
PARAMETER SET {
 problem [0] **AbandonProblem**,
 operation [1] **InvokeID**}

AbandonProblem ::= INTEGER
 noSuchOperation (1),
 tooLate (2),
 cannotAbandon (3) }

12.3.2 The various parameters have the meanings as defined in §§ 12.3.2.1 and 12.3.2.2.

12.3.2.1 The particular **problem** encountered is specified. Any of the following problems may be indicated:

- a) **noSuchOperation**, when the Directory has no knowledge of the operation which is to be abandoned (this could be because no such invoke took place or because the Directory has forgotten about it);
- b) **tooLate**, when the Directory has already responded to the operation;
- c) **cannotAbandon**, when an attempt has been made to abandon an operation for which this is prohibited (e.g. modify), or the abandon could not be performed.

12.3.2.2 The identification of the particular **operation** (invocation) to be abandoned.

12.4 *Attribute Error*

12.4.1 An **AttributeError** reports an attribute-related problem.

AttributeError ::= ABSTRACT-ERROR
PARAMETER SET {
 object [0] **Name**,
 problems [1] **SET OF SEQUENCE {**
 problem [0] **AttributeProblem**,
 type [1] **AttributeType**,
 value [2] **AttributeValue**
 OPTIONAL }}

AttributeProblem ::= INTEGER {
 noSuchAttributeOrValue (1),
 InvalidAttributeSyntax (2),
 undefinedAttributeType (3),
 InappropriateMatching (4),
 constraintViolation (5)
 attributeOrValueAlreadyExists (6) }

12.4.2 The various parameters have the meanings as described in §§ 12.4.2.1 and 12.4.2.2.

12.4.2.1 The **object** parameter identifies the entry to which the operation was being applied when the error occurred.

12.4.2.2 One or more **problems** may be specified. Each **problem** identified below is accompanied by an indication of the attribute **type**, and if necessary to avoid ambiguity, the **value**, which caused the problem:

- a) **noSuchAttributeOrValue**: The named entry lacks one of the attributes or attribute values specified as an argument of the operation;
- b) **invalidAttributeSyntax**: A purported attribute value, specified as an argument of the operation, does not conform to the attribute syntax of the attribute type;
- c) **undefinedAttributeType**: An undefined attribute type was provided as an argument to the operation. This error may occur only in relation to **Add**, **Remove**, **Modify** or **ModifyRDN** operations;
- d) **inappropriateMatching**: An attempt was made, e.g. in a filter, to use a matching rule not defined for the attribute type concerned;
- e) **constraintViolation**: An attribute or attribute value supplied in the argument of abstract-operation does not conform to the constraints imposed by Recommendation X.501 or by the attribute definition (e.g. the value exceeds the maximum size allowed);

- f) **attributeOrValueAlreadyExists**: An attempt was made to add an attribute which already existed in the entry, or a value which already existed in the attribute.

12.5 Name Error

12.5.1 A **NameError** reports a problem related to the name provided as an argument to an operation.

```
NameError ::= ABSTRACT-ERROR  
  PARAMETER SET {  
    problem [0] NameProblem,  
    matched [1] Name}
```

```
NameProblem ::= INTEGER {  
  noSuchObject (1),  
  aliasProblem (2),  
  invalidAttributeSyntax (3),  
  aliasDereferencingProblem (4) }
```

12.5.2 The various parameters have the meanings as described in §§ 12.5.2.1 and 12.5.2.2.

12.5.2.1 The particular **problem** encountered. Any of the following problems may be indicated:

- noSuchObject**: The name supplied does not match the name of any object;
- aliasProblem**: An alias has been dereferenced which names no object;
- invalidAttributeSyntax**: An attribute type and its accompanying attribute value in AVA in the name are incompatible;
- aliasDereferencingProblem**: An alias was encountered in a situation where it was not allowed.

12.5.2.2 The **matched** parameter contains the name of the lowest entry (object or alias) in the DIT that was matched and is a truncated form of the name provided or, if an alias has been dereferenced, of the resulting name.

Note - If there is a problem with the attribute types and/or values in the name offered in a directory operation argument, this is reported via a **NameError** (with problem **invalidAttributeSyntax**) rather than as an **AttributeError** or an **UpdateError**.

12.6 Referral

12.6.1 A **Referral** redirects the service-user to one or more access points better equipped to carry out the requested operation.

```
Referral ::= ABSTRACT-ERROR -- not literally an "error"  
  PARAMETER SET {  
    candidate [0] ContinuationReference }
```

12.6.2 The error has a single parameter which contains a **ContinuationReference** which can be used to progress the operation (see Recommendation X.518).

12.7 Security Error

12.7.1 A **SecurityError** reports a problem in carrying out an operation for security reasons.

```
SecurityError ::= ABSTRACT-ERROR  
  PARAMETER SET {  
    problem [0] SecurityProblem }
```

```
SecurityProblem ::= INTEGER {  
  InappropriateAuthentication (1),  
  InvalidCredentials (2),  
  InsufficientAccessRights (3),  
  InvalidSignature (4),  
  protectionRequired (5),  
  noInformation (6) }
```

12.7.2 The error has a single parameter, which reports the particular **problem** encountered. The following problems may be indicated:

- inappropriateAuthentication**: The level of security associated with the requestor's credentials is inconsistent with the level of protection requested, e.g. simple credentials were supplied while strong credentials were required;

- b) **invalidCredentials**: The supplied credentials were invalid;
- c) **insufficientAccessRights**: The requestor does not have the right to carry out the requested operation;
- d) **invalidSignature**: The signature of the request was found to be invalid;
- e) **protectionRequired**: The Directory was unwilling to carry out the requested operation because the argument was not signed;
- f) **noInformation**: The requested operation produced a security error for which no information is available.

12.8 Service Error

12.8.1 A **ServiceError** reports a problem related to the provision of the service.

```
ServiceError ::= ABSTRACT-ERROR
  PARAMETER SET {
    problem [0] ServiceProblem },
ServiceProblem ::= INTEGER {
  busy (1),
  unavailable (2),
  unwillingToPerform (3),
  chainingRequired (4),
  unableToProceed (5),
  invalidReference (6),
  timeLimitExceeded (7),
  administrativeLimitExceeded (8),
  loopDetected (9),
  unavailableCriticalExtension (10),
  outOfScope (11),
  ditError (12) }
```

12.8.2 The error has a single parameter, which reports the particular **problem** encountered. The following problems may be indicated:

- a) **busy**: The Directory, or some part of it, is presently too busy to perform the requested operation, but may be able to do so after a short while;
- b) **unavailable**: The Directory, or some part of it, is currently unavailable;
- c) **unwillingToPerform**: The Directory, or some part of it, is not prepared to execute this request, e.g. because it would lead to excessive consumption of resources or violate the policy of an Administrative Authority involved;
- d) **chainingRequired**: The Directory is unable to accomplish the request other than by chaining, however chaining was prohibited by means of the **chainingProhibited** service control option;
- e) **unableToProceed**: The DSA returning this error did not have administrative authority for the appropriate naming context and as a consequence was not able to participate in name resolution;
- f) **invalidReference**: The DSA was unable to perform the request as directed by the DUA (in **OperationProgress**). This may have arisen due to using an invalid referral;
- g) **timeLimitExceeded**: The Directory has reached the limit of time set by the user in a service control. No partial results are available to return to the user;
- h) **administrativeLimitExceeded**: The Directory has reached some limit set by an administrative authority, and no partial results are available to return to the user;
- i) **loopDetected**: The Directory is unable to accomplish the request due to an internal loop;
- j) **unavailableCriticalExtension**: The Directory was unable to execute the request because one or more critical extensions were not available;

- k) **outOfScope**: No referrals were available within the requested scope;
- l) **ditError**: The Directory is unable to accomplish the request due to a DIT consistency problem.

12.9 Update Error

12.9.1 An **UpdateError** reports problems related to attempts to add, delete, or modify information in the DIB.

```
UpdateError ::= ABSTRACT-ERROR
  PARAMETER SET {
    problem [0] UpdateProblem }
```

```
UpdateProblem ::= INTEGER {
  namingViolation (1),
  objectClassViolation (2),
  notAllowedOnNonLeaf (3),
  notAllowedOnRDN (4),
  entryAlreadyExists (5),
  affectsMultipleDSAs (6),
  objectClassModificationProhibited (7) }
```

12.9.2 The error has a single **problem** parameter, which reports the particular **problem** encountered. The following problems may be indicated:

- a) **namingViolation**: The attempted addition or modification would violate the structure rules of the DIT as defined in the Directory schema and Recommendation X.501. That is, it would place an entry as the subordinate of an alias entry, or in a region of the DIT not permitted to a member of its object class or would define an RDN for an entry to include a forbidden attribute type;
- b) **objectClassViolation**: The attempted update would produce an entry inconsistent with the definition provided by its object class or with the definitions of Recommendation X.501 as they pertain to object classes;
- c) **notAllowedOnNonLeaf**: The attempted operation is only allowed on leaf entries of the DIT;
- d) **notAllowedOnRDN**: The attempted operation would affect the RDN (e.g. removal of an attribute which is a part of the RDN);
- e) **entryAlreadyExists**: An attempted AddEntry operation names an entry which already exists;
- f) **affectsMultipleDSAs**: An attempted update would need to operate on multiple DSAs, which is not permitted;
- g) **objectClassModificationProhibited**: An operation attempted to modify the object class attribute.

Note - The **UpdateError** is not used to report problems with attribute types, values or constraint violations encountered in an **AddEntry**, **RemoveEntry**, **ModifyEntry** or **ModifyRDN** operation. Such problems are reported via an **AttributeError**.

ANNEX A

(to Recommendation X.511)

Abstract service in ASN.1

This Annex is part of the standard.

This Annex includes all of the ASN.1 type, value and macro definitions contained in this Recommendation in the form of the ASN.1 module **DirectoryAbstractService**.

```

DirectoryAbstractService {joint-ISO-CCITT ds(5) modules(1) directoryAbstractService(2)}
DEFINITIONS ::=
BEGIN
EXPORTS
    directory, readPort, searchPort, modifyPort,
    DirectoryBind, DirectoryBindArgument,
    DirectoryUnbind,
    Read, ReadArgument, ReadResult,
    Abandon, AbandonArgument, AbandonResult,
    Compare, CompareArgument, CompareResult,
    List, ListArgument, ListResult,
    Search, SearchArgument, SearchResult,
    AddEntry, AddEntryArgument, AddEntryResult,
    RemoveEntry, RemoveEntryArgument, RemoveEntryResult,
    ModifyEntry, ModifyEntryArgument, ModifyEntryResult,
    ModifyRDN, ModifyRDNArgument, ModifyRDNResult,
    Abandoned, AbandonFailed, AttributeError, NameError,
    Referral, SecurityError, ServiceError, UpdateError,
    SecurityParameters;

IMPORTS
    informationFramework, authenticationFramework,
        distributedOperations, directoryObjectIdentifiers
    FROM UsefulDefinitions {joint-iso-ccitt ds(5) modules(1)
        usefulDefinitions(0)}

OBJECT, PORT, ABSTRACT-BIND, ABSTRACT-UNBIND,
ABSTRACT-OPERATION, ABSTRACT-ERROR
    FROM AbstractServiceNotation {joint-iso-ccitt mhs-motis(6)
        asdc(2) modules(0) notation(1) }

Attribute, AttributeType, AttributeValue, AttributeValueAssertion,
DistinguishedName, Name, RelativeDistinguishedName
    FROM InformationFramework InformationFramework

id-ot-directory, id-ot-dua, id-pt-read, id-pt-search, id-pt-modify
    FROM DirectoryObjectIdentifiers directoryObjectIdentifiers

ContinuationReference, OperationProgress
    FROM DistributedOperations distributedOperations

Certificate, CertificationPath, SIGNED,
PROTECTED, AlgorithmIdentifier
    FROM AuthenticationFramework authenticationFramework

InvokeID,
    FROM Remote-Operations-Notation {joint-iso-ccitt
        remoteOperations(4) notation(0)};

-- macro for representing optional signing --
OPTIONALLY-SIGNED MACRO ::=
BEGIN
    TYPE NOTATION ::= type (Type)
    VALUE NOTATION ::= value (VALUE CHOICE { Type, SIGNED Type})
END

-- objects and ports --

directory
    OBJECT
        PORTS { readPort [S],
            searchPort [S],
            modifyPort [S]}

::= id-ot-directory

```

```

dua
    OBJECT
        PORTS { readPort [C],
                  searchPort [C]
                  modifyPort [C]}

::= id-ot-dua

readPort
    PORT
        CONSUMER INVOKES {
            Read, Compare, Abandon}

::= id-pt-read

searchPort
    PORT
        CONSUMER INVOKES {
            List, Search }

::= id-pt-search

modifyPort
    PORT
        CONSUMER INVOKES {
            AddEntry, RemoveEntry,
            ModifyEntry, ModifyRDN}

::= id-pt-modify

-- bind and unbind --

DirectoryBind ::= ABSTRACT-BIND
    TO { readPort, searchPort, modifyPort }
    BIND
    ARGUMENT DirectoryBindArgument
    RESULT DirectoryBindResult
    BIND-ERROR DirectoryBindError

DirectoryBindArgument ::= SET {
    credentials [0] Credentials OPTIONAL,
    versions [1] Versions DEFAULT v1988}

Credentials ::= CHOICE {
    simple [0] SimpleCredentials,
    strong [1] StrongCredentials,
    externalProcedure [2] EXTERNAL }

SimpleCredentials ::= SEQUENCE {
    name [0] DistinguishedName,
    validity [1] SET {
        time1 [0] UTCTime OPTIONAL,
        time2 [1] UTCTime OPTIONAL,
        random1 [2] BIT STRING OPTIONAL,
        random2 [3] BIT STRING OPTIONAL }
        OPTIONAL,
    password [2] OCTET STRING OPTIONAL }

StrongCredentials ::= SET {
    certification-path [0] CertificationPath OPTIONAL,
    bind-token [1] Token }

Token ::= SIGNED SEQUENCE {
    algorithm [0] AlgorithmIdentifier
    name [1] DistinguishedName,
    time [2] UTCTime,
    random [3] BIT STRING }

Versions ::= BIT STRING (v1988(0))

DirectoryBindResult ::= DirectoryBindArgument

```

```

DirectoryBindError ::= SET {
    versions [0] Versions DEFAULT v1988,
    CHOICE {
        serviceError [1] ServiceProblem,
        securityError [2] SecurityProblem }}

DirectoryUnbind ::= ABSTRACT-UNBIND
    FROM {readPort, searchPort, modifyPort }

-- operations, arguments, and results --

Read ::= ABSTRACT-OPERATION
    ARGUMENT ReadArgument
    RESULT    ReadResult
    ERRORS {
        AttributeError, NameError,
        ServiceError, Referral, Abandoned,
        SecurityError }

ReadArgument ::= OPTIONALLY-SIGNED SET {
    object      [0] Name,
    selection    [1] EntryInformationSelection
                DEFAULT {},
    COMPONENTS OF CommonArguments }

ReadResult ::= OPTIONALLY-SIGNED SET {
    entry [0] EntryInformation,
    COMPONENTS OF CommonResults }

Compare ::= ABSTRACT-OPERATION
    ARGUMENT CompareArgument
    RESULT    CompareResult
    ERRORS {
        AttributeError, NameError,
        ServiceError, Referral, Abandoned,
        SecurityError }

CompareArgument ::= OPTIONALLY-SIGNED SET {
    object      [0] Name,
    purported    [1] AttributeValueAssertion,
    COMPONENTS OF CommonArguments }

CompareResult ::= OPTIONALLY-SIGNED SET {
    DistinguishedName OPTIONAL,
    matched [0] BOOLEAN,
    fromEntry [1] BOOLEAN DEFAULT TRUE,
    COMPONENTS OF CommonResults }

Abandon ::= ABSTRACT-OPERATION
    ARGUMENT AbandonArgument
    RESULT AbandonResult
    ERRORS {AbandonFailed}

AbandonArgument ::= SEQUENCE {
    InvokeID [0] InvokeID}

AbandonResult ::= NULL

List ::= ABSTRACT-OPERATION
    ARGUMENT ListArgument
    RESULT    ListResult
    ERRORS {
        AttributeError, NameError,
        ServiceError, Referral, Abandoned,
        SecurityError }

ListArgument ::= OPTIONALLY-SIGNED SET {
    object [0] Name,
    COMPONENTS OF CommonArguments }

```



```

ListResult ::= OPTIONALLY-SIGNED CHOICE{
  listInfo SET {
    DistinguishedName OPTIONAL
    subordinates [1] SET OF SEQUENCE {
      RelativeDistinguishedName,
      aliasEntry [0] BOOLEAN DEFAULT FALSE,
      fromEntry [1] BOOLEAN DEFAULT TRUE },
      partialOutcomeQualifier [2] PartialOutcomeQualifier
      OPTIONAL,
      COMPONENTS OF CommonResults},
  uncorrelatedListInfo [0] SET OF
    ListResult }

PartialOutcomeQualifier ::= SET {
  limitProblem [0] LimitProblem OPTIONAL,
  unexplored [1] SET OF
    ContinuationReference OPTIONAL,
  unavailableCriticalExtensions [2] BOOLEAN DEFAULT FALSE }

LimitProblem ::= INTEGER {
  timeLimitExceeded (0),
  sizeLimitExceeded (1),
  administrativeLimitExceeded (2) }

Search ::= ABSTRACT-OPERATION
  ARGUMENT SearchArgument
  RESULT SearchResult
  ERRORS {
    AttributeError, NameError,
    ServiceError, Referral, Abandoned,
    SecurityError }

SearchArgument ::= OPTIONALLY-SIGNED SET {
  baseObject [0] Name,
  subset [1] INTEGER {
    baseObject(0),
    oneLevel(1),
    wholeSubtree(2)) DEFAULT baseObject,
  filter [2] Filter DEFAULT and {},
  searchAliases [3] BOOLEAN DEFAULT TRUE,
  selection [4] EntryInformationSelection DEFAULT {},
  COMPONENTS OF CommonArguments }

SearchResult ::= OPTIONALLY-SIGNED
  CHOICE {
    searchInfo SET {
      DistinguishedName OPTIONAL,
      entries [0] SET OF EntryInformation,
      partialOutcomeQualifier
        [2] partialOutcomeQualifier OPTIONAL,
      COMPONENTS OF CommonResults },
    uncorrelatedSearchInfo [0] SET OF
      SearchResult }

AddEntry ::= ABSTRACT-OPERATION
  ARGUMENT AddEntryArgument
  RESULT AddEntryResult
  ERRORS {
    AttributeError, NameError,
    ServiceError, Referral, SecurityError
    UpdateError }

AddEntryArgument ::= OPTIONALLY-SIGNED SET {
  object [0] DistinguishedName,
  entry [1] SET OF Attribute,
  COMPONENTS OF CommonArguments}

AddEntryResult ::= NULL

```

```

RemoveEntry ::= ABSTRACT-OPERATION
    ARGUMENT RemoveEntryArgument
    RESULT    RemoveEntryResult
    ERRORS {
        NameError,
        ServiceError, Referral, SecurityError,
        UpdateError}

RemoveEntryArgument ::= OPTIONALLY-SIGNED SET {
    object [0] DistinguishedName,
    COMPONENTS OF CommonArguments }

RemoveEntryResult ::= NULL

ModifyEntry ::= ABSTRACT-OPERATION
    ARGUMENT ModifyEntryArgument
    RESULT    ModifyEntryResult
    ERRORS {
        AttributeError, NameError,
        ServiceError, Referral, SecurityError,
        UpdateError}

ModifyEntryArgument ::= OPTIONALLY-SIGNED SET {
    object [0] DistinguishedName,
    changes [1] SEQUENCE OF EntryModification,
    COMPONENTS OF CommonArguments }

ModifyEntryResult ::= NULL

EntryModification ::= CHOICE {
    addAttribute [0] Attribute,
    removeAttribute [1] AttributeType,
    addValues [2] Attribute,
    removeValues [3] Attribute}

ModifyRDN ::= ABSTRACT-OPERATION
    ARGUMENT ModifyRDNArgument
    RESULT    ModifyRDNResult
    ERRORS {
        NameError,
        ServiceError, Referral, SecurityError,
        UpdateError }

ModifyRDNArgument ::= OPTIONALLY-SIGNED SET {
    object [0] DistinguishedName,
    newRDN [1] RelativeDistinguishedName,
    deleteoldRDN [2] BOOLEAN DEFAULT FALSE,
    COMPONENTS OF CommonArguments }

ModifyRDNResult ::= NULL

-- errors and parameters --

Abandoned ::= ABSTRACT-ERROR -- not literally an "error"

AbandonFailed ::= ABSTRACT-ERROR
    PARAMETER SET {
        problem [0] AbandonProblem,
        operation [1] InvokeID}

AbandonProblem ::= INTEGER {
    noSuchOperation (1),
    tooLate (2),
    cannotAbandon (3)}

```

AttributeError ::= ABSTRACT-ERROR
 PARAMETER SET {
 object [0] Name,
 problems [1] SET OF SEQUENCE {
 problem [0] AttributeProblem,
 type [1] AttributeType,
 value [2] AttributeValue OPTIONAL }}}

AttributeProblem ::= INTEGER {
 noSuchAttributeOrValue (1),
 invalidAttributeSyntax (2),
 undefinedAttributeType (3),
 inappropriateMatching (4),
 constraintViolation (5),
 attributeOrValueAlreadyExists (6) }

NameError ::= ABSTRACT-ERROR
 PARAMETER SET {
 problem [0] NameProblem,
 matched [1] Name}

NameProblem ::= INTEGER {
 noSuchObject (1),
 aliasProblem (2),
 invalidAttributeSyntax (3),
 aliasDereferencingProblem (4)}

Referral ::= ABSTRACT-ERROR -- *not literally an "error"*
 PARAMETER SET {
 candidate [0] ContinuationReference}

SecurityError ::= ABSTRACT-ERROR
 PARAMETER SET {
 problem [0] SecurityProblem }

SecurityProblem ::= INTEGER {
 inappropriateAuthentication (1),
 invalidCredentials (2),
 insufficientAccessRights (3),
 invalidSignature (4),
 protectionRequired (5),
 noInformation (6) }

ServiceError ::= ABSTRACT-ERROR
 PARAMETER SET {
 problem [0] ServiceProblem }

ServiceProblem ::= INTEGER {
 busy (1),
 unavailable (2),
 unwillingToPerform (3),
 chainingRequired (4),
 unableToProceed (5),
 invalidReference (6),
 timeLimitExceeded (7),
 administrativeLimitExceeded (8),
 loopDetected (9),
 unavailableCriticalExtension (10),
 outOfScope (11),
 ditError (12) }

UpdateError ::= ABSTRACT-ERROR
 PARAMETER SET {
 problem [0] UpdateProblem }

```

UpdateProblem ::= INTEGER {
    namingViolation (1),
    objectClassViolation (2),
    notAllowedOnNonLeaf (3),
    notAllowedOnRDN (4),
    entryAlreadyExists (5),
    affectsMultipleDSAs (6),
    objectClassModificationProhibited (7))

-- common arguments/results --

CommonArguments ::= SET {
    [30] ServiceControls DEFAULT {},
    [29] SecurityParameters DEFAULT {},
    requestor [28] DistinguishedName OPTIONAL,
    [27] OperationProgress DEFAULT notStarted,
    aliasedRDNs [26] INTEGER OPTIONAL,
    extensions [25] SET OF Extension OPTIONAL }

Extension ::= SET {
    identifier [0] INTEGER,
    critical [1] BOOLEAN DEFAULT FALSE,
    item [2] ANY DEFINED BY identifier }

CommonResults ::= SET {
    [30] SecurityParameters OPTIONAL,
    performer [29] DistinguishedName OPTIONAL,
    aliasDereferenced [28] BOOLEAN DEFAULT FALSE}

-- common data types --

ServiceControls ::= SET {
    options [0] BIT STRING {
        preferChaining (0),
        chainingProhibited (1),
        localScope (2),
        dontUseCopy (3),
        dontDereferenceAliases(4)}
        DEFAULT {},

    priority [1] INTEGER {
        low (0),
        medium (1),
        high (2) } DEFAULT medium,

    timeLimit [2] INTEGER OPTIONAL,
    sizeLimit [3] INTEGER OPTIONAL,
    scopeOfReferral [4] INTEGER {
        dmd(0),
        country(1)}
        OPTIONAL }

EntryInformationSelection ::= SET {
    attributeTypes
        CHOICE {
            allAttributes [0] NULL,
            select [1] SET OF AttributeType
            -- empty set implies no attributes
            -- are requested --}
            DEFAULT allAttributes NULL,

    infoTypes [2] INTEGER {
        attributeTypesOnly (0),
        attributeTypesAndValues (1) } DEFAULT
        attributeTypesandValues }

```

```

EntryInformation ::= SEQUENCE {
    DistinguishedName,
    fromEntry BOOLEAN DEFAULT TRUE,
    SET OF CHOICE {
        AttributeType,
        Attribute} OPTIONAL }

Filter ::= CHOICE {
    item [0] FilterItem,
    and [1] SET OF Filter,
    or [2] SET OF Filter,
    not [3] Filter }

FilterItem ::= CHOICE {
    equality [0] AttributeValueAssertion,
    substrings [1] SEQUENCE {
        type AttributeType,
        strings SEQUENCE OF CHOICE {
            initial [0] AttributeValue,
            any [1] AttributeValue,
            final [2] AttributeValue}},
    greaterOrEqual [2] AttributeValueAssertion,
    lessOrEqual [3] AttributeValueAssertion,
    present [4] AttributeType,
    approximateMatch [5] AttributeValueAssertion }

SecurityParameters ::= SET {
    certification-Path [0] CertificationPath OPTIONAL,
    name [1] DistinguishedName OPTIONAL,
    time [2] UTCTime OPTIONAL,
    random [3] BIT STRING OPTIONAL,
    target [4] ProtectionRequest OPTIONAL }

ProtectionRequest ::= INTEGER {
    none(0),
    signed (1)}

```

ANNEX B

(to Recommendation X.511)

Directory object identifiers

This Annex is part of the standard.

This Annex includes all of the ASN.1 object identifiers contained in this Recommendation in the form of the ASN.1 module "DirectoryObjectIdentifiers".

```

DirectoryObjectIdentifiers {joint-ISO-CCITT ds(5) modules(1)
    directoryObjectIdentifiers(9)}

DEFINITIONS ::=
BEGIN

EXPORTS
    id-ot-directory, id-ot-dua, id-pt-read, id-pt-search, id-pt-modify;

IMPORTS
    id-ot, id-pt
    FROM UsefulDefinitions {joint-iso-ccitt ds(5) modules(1),
        usefulDefinitions(0)};

```

-- *Objects* --

id-ot-directory OBJECT IDENTIFIER ::= {id-ot 1}

id-ot-dua OBJECT IDENTIFIER ::= {id-ot 2}

-- *Port Types* --

id-pt-read OBJECT IDENTIFIER ::= {id-pt 1}

id-pt-search OBJECT IDENTIFIER ::= {id-pt 2}

id-pt-modify OBJECT IDENTIFIER ::= {id-pt 3}

END

Recommendation X.518

THE DIRECTORY - PROCEDURES FOR DISTRIBUTED OPERATION ¹⁾

(Melbourne, 1988)

CONTENTS

SECTION 1 - *Introduction*

- 0 *Introduction*
- 1 *Scope and field of application*
- 2 *References*
- 3 *Definitions*
- 4 *Abbreviations*
- 5 *Notation*

SECTION 2 - *Overview*

- 6 *Overview*

SECTION 3 - *Distributed directory models*

- 7 *Distributed directory system model*
- 8 *DSA interactions model*
 - 8.1 *Chaining*
 - 8.2 *Multicasting*
 - 8.3 *Referral*
 - 8.4 *Mode determination*
- 9 *Directory distribution*

¹⁾ Recommendation X.518 and ISO 9594-4, Information Processing Systems - Open Systems Interconnection - The Directory - Procedures for Distributed Operation, were developed in close collaboration and are technically aligned.

- 10 *Knowledge*
 - 10.1 Minimal knowledge references
 - 10.2 Root context
 - 10.3 Knowledge references
 - 10.4 Knowledge administration

SECTION 4 - *DSA abstract service*

- 11 *Overview of DSA abstract service*
- 12 *Information types*
 - 12.1 Introduction
 - 12.2 Information types defined elsewhere
 - 12.3 Chaining arguments
 - 12.4 Chaining results
 - 12.5 Operation progress
 - 12.6 Trace information
 - 12.7 Reference type
 - 12.8 Access point
 - 12.9 Continuation reference
- 13 *Abstract-bind and abstract-unbind*
 - 13.1 DSA bind
 - 13.2 Directory unbind
- 14 *Chained abstract-operations*
- 15 *Chained abstract-errors*
 - 15.1 Introduction
 - 15.2 DSA referral

SECTION 5 - *Distributed operations procedures*

- 16 *Introduction*
 - 16.1 Scope and limits
 - 16.2 Conceptual model
 - 16.3 Individual and cooperative operation of DSAs
- 17 *Distributed directory behaviour*
 - 17.1 Cooperative fulfillment of operations
 - 17.2 Phases of operation processing
 - 17.3 Managing distributed operations
 - 17.4 Other considerations for distributed operation
 - 17.5 Authentication of distributed operations
- 18 *DSA behaviour*
 - 18.1 Introduction
 - 18.2 Overview of the DSA behaviour
 - 18.3 Specific operations
 - 18.4 Operation dispatcher
 - 18.5 Looping
 - 18.6 Name resolution procedure
 - 18.7 Object evaluation procedures
 - 18.8 Result merging procedure
 - 18.9 Procedures for distributed authentication

Annex A - ASN.1 for distributed operations

Annex B - Modelling of knowledge

Annex C - Distributed use of authentication

Annex D - Distributed directory object identifiers

SECTION 1 - Introduction

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. The set of all 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 OSI 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 specifies the procedures by which the distributed components of the Directory interwork in order to provide a consistent service to its users.

1 Scope and field of application

1.1 This Recommendation specifies the behaviour of DSAs taking part in the distributed Directory application. The allowed behaviour has been designed so as to ensure a consistent service given a wide distribution of the DIB across many DSAs.

1.2 The Directory is not intended to be a general purpose database system, although it may be built on such systems. It is assumed that there is a considerably higher frequency of queries than of updates.

2 References

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

Recommendation X.208 - Open Systems Interconnection - Specification of Abstract Syntax Notation (ASN.1)

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

Recommendation X.501 - The Directory - Models

Recommendation X.511 - The Directory - Abstract Service Definition

Recommendation X.519 - The Directory - Protocol Specifications

Recommendation X.520 - The Directory - Selected Attribute Types

Recommendation X.521 - The Directory - Selected Object Classes

Recommendation X.407 - Message Handling Systems - Abstract Service Definition Conventions

3 Definitions

The definitions contained in this paragraph make use of the abbreviations defined in § 4.

3.1 OSI Reference Model Definitions

This Recommendation makes use of the following term defined in X.200:

- a) *application entity title*.

3.2 *Basic Directory Definitions*

This Recommendation makes use of the following terms defined in Recommendation X.500:

- a) *(the) Directory*;
- b) *Directory Information Base*.

3.3 *Directory Model Definitions*

This Recommendation makes use of the following terms defined in Recommendation X.501:

- a) *access point*;
- b) *alias*;
- c) *distinguished name*;
- d) *Directory Information Tree*;
- e) *Directory System Agent*;
- f) *Directory User Agent*;
- g) *relative distinguished name*.

3.4 *Abstract Service Definition Conventions*

This Recommendation makes use of the following terms defined in X.407:

- a) *abstract error*;
- b) *abstract operation*;
- c) *result*.

3.5 *Distributed Operation Definitions*

This Recommendation makes use of the following terms, as defined here:

- a) *chaining*: a mode of interaction optionally used by a DSA which cannot perform an operation itself. The DSA chains by invoking an operation of another DSA and then relaying the outcome to the original requestor;
- b) *context prefix*: the sequence of RDNs leading from the Root of the DIT to the initial vertex of a naming context, corresponds to the distinguished name of that vertex;
- c) *cross reference*: a knowledge reference containing information about the DSA that holds an entry. This is used for optimisation. The entry need have no superior or subordinate relationship;
- d) *DIB fragment*: the portion of the DIB that is held by one DSA, comprising one or more naming contexts;
- e) *distributed name resolution*: the process by which name resolution is performed in more than one DSA;
- f) *internal reference*: a knowledge reference containing an internal pointer to an entry held in the same DSA;
- g) *knowledge information*: the information which a particular DSA has about the entries it holds and how to locate other entries in the directory;
- h) *knowledge reference*: knowledge which associates, either directly or indirectly, a DIT entry with the DSA in which it is located;
- i) *knowledge tree*: the conceptual model of the knowledge information that a DSA holds to enable it to perform distributed name resolution;
- j) *multicasting*: a mode of interaction which may optionally be used by a DSA which cannot perform an operation itself. The DSA *multicasts* the operation, i.e. invokes the same operation of several other DSAs (in series or in parallel) and passes an appropriate outcome to the original requestor;
- k) *name resolution*: the process of locating an entry by sequentially matching each RDN in a purported name to a vertex of the DIT;
- l) *naming context*: a partial sub-tree of the DIT which starts at a vertex and extends downwards to leaf and/or non-leaf vertices. Such vertices constitute the border of the

naming context. Non-leaf vertices belonging to the border denote the start of further naming contexts;

- m) *non-specific subordinate reference*: a knowledge reference that holds information about the DSA that holds one or more unspecified subordinate entries;
- n) *operation progress*: a set of values which denotes the extent to which name resolution has taken place;
- o) *reference path*: a continuous sequence of knowledge references;
- p) *referral*: an outcome which can be returned by a DSA which cannot perform an operation itself, and which identifies one or more other DSAs more able to perform the operation;
- q) *request decomposition*: decomposition of a request into subrequests each accomplishing a part of the distributed operation;
- r) *root context*: the naming context for the vertex whose name comprises the empty sequence of RDNs;
- s) *subordinate reference*: a knowledge reference containing information about the DSA that holds a specific subordinate entry;
- t) *subrequest*: a request generated by request decomposition;
- u) *superior reference*: a knowledge reference containing information about the DSA that holds a superior entry.

4 Abbreviations

The following abbreviations are used in this Recommendation:

DIB	Directory Information Base
DIT	Directory Information Tree
DSA	Directory System Agent
DUA	Directory User Agent
RDN	Relative Distinguished Name

5 Notation

The notation used in this paragraph is defined as follows:

- a) the data syntax notation, encoding and macro notation are defined in Recommendation X.208;
- b) the notations for abstract models and abstract services are defined in Recommendation X.407.

SECTION 2 - Overview

6 Overview

The Directory Abstract Service allows the interrogation, retrieval and modification of Directory information in the DIB. This service is described in terms of the abstract Directory object as specified in Recommendation X.511.

Necessarily, the specification of the abstract Directory object does not in any way address the physical realization of the Directory, in particular it does not address the specification of Directory System Agents (DSA) within which the DIB is stored and managed, and through which the service is provided. Furthermore, it does not consider whether the DIB is centralized, i.e. contained within a single DSA, or distributed over a number of DSAs. Consequently, the requirements for DSAs to have knowledge of, navigate to, and cooperate with other DSAs, in order to support the abstract service in a distributed environment, is also not covered by the service description.

This Recommendation specifies the refinement of the abstract Directory object, the refinement being expressed in terms of a set of one or more DSA objects which collectively constitute the distributed directory service. Inherent in this is the identification and specification of the DSA ports that are internal to the Directory object. For each such port, this Recommendation specifies the associated abstract services and its procedures.

In addition, this Recommendation specifies the permissible ways in which the DIB may be distributed over one or more DSAs. For the limiting case where the DIB is contained within a single DSA, the Directory is in fact centralized; for the case where the DIB is distributed over two or more DSAs, knowledge and navigation mechanisms are specified which ensure that the whole of the DIB is potentially accessible from all DSAs that hold constituent entries.

Additionally, request handling interactions are specified that enable particular operational characteristics of the Directory to be controlled by its users. In particular, the user has control over whether a DSA, responding to a directory enquiry pertaining to information held in other DSA(s), has the option of interrogating the other DSA(s) directly (chaining/multicasting) or, whether it should respond with information about other DSA(s) which could further progress the enquiry (referral).

Generally, the decision by a DSA to chain/multicast or refer is determined by the service controls set by the user, and by the DSA's own administrative, operational, or technical circumstances.

Recognizing that, in general, the Directory will be distributed, that directory enquiries will be satisfied by an arbitrary number of cooperating DSAs which may arbitrarily chain/multicast or refer according to the above criteria, this Recommendation specifies the appropriate procedures to be effected by DSAs in responding to distributed directory enquiries. These procedures will ensure that users of the distributed Directory service perceive it to be both user-friendly and consistent.

SECTION 3 - *Distributed directory models*

7 Distributed directory system model

The Directory abstract service as defined in Recommendation X.511 models the directory as an object which provides a set of directory services to its users. The services of the directory are modelled in terms of ports, where each port provides a particular set of directory services. Users of the directory access its services through an access point. The directory may have one or more access points and each access point is characterized by the services it provides and the mode of interaction used to provide these services.

This paragraph addresses the internal structure of the directory object, identifying its constituent objects and their ports, and thereby facilitates the specification of a distributed directory service.

Figure 1/X.518 illustrates the distributed directory which will be used as the basis for specifying the distributed aspects of the directory. It illustrates the directory object as comprising a set of one or more DSA-objects.

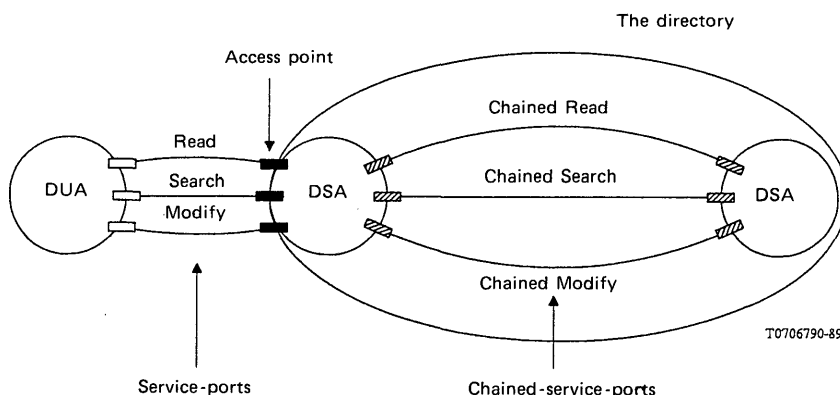


FIGURE 1/X.518

Objects of the distributed directory model

DSA objects are specified in detail in the subsequent clauses of this Recommendation. This clause merely states a number of their characteristics in order to serve as an introduction and to establish the relationship between this Recommendation and other Recommendations.

DSA objects are defined in order that distribution of the DIB can be accommodated and that a number of physically distributed DSAs can interact in a prescribed, cooperative manner to provide directory services to the users of the directory (DUAs).

DSA objects, like the Directory object, are characterized by their externally visible ports. The ports associated with a DSA-object are of two types: service-ports and chained-service-ports.

The service-ports of a DSA object are identical to those of the Directory object, namely, **read**, **search** and **modify**. Figure 1/X.518 illustrates that the service-ports associated with a DSA object constitute an access-point through which directory services are made available.

The detailed specification of the **read**, **search**, and **modify** service-ports of the DSA object can be found in Recommendation X.511. (The protocol specification for the corresponding OSI application service elements, as derived from these port definitions, can be found in Recommendation X.519.)

In addition to the service-ports of the DSA object which accommodate access to the Directory object, a second set of ports are defined, the chained-service-ports. These permit inter-DSA communication in order that the Directory abstract service can be realized in a distributed environment.

The chained-service-ports and the operations provided through them are in direct correspondence to the similarly named service-ports, and are, respectively, **chainedRead**, **chainedSearch**, and **chainedModify**.

The process of specifying the constituent objects of a more abstract object is termed "refinement". The specification of the refinement of the Directory object into its component parts (the DSAs), and the specification of the abstract service provided by each of them (the DSA Abstract Service) is contained in Section Four of this Recommendation. The protocol specification of the corresponding OSI application service elements, as derived from the chained port definitions, can be found in Recommendation X.519.

8 DSA interactions model

A basic characteristic of the Directory is that, given a distributed DIB, a user should potentially be able to have any service request satisfied (subject to security, access control and administrator policies) irrespective of the access point at which the request originates. In accommodating this requirement it is necessary that any DSA involved in satisfying a particular service request have some knowledge (as specified in § 10 of this Recommendation) of where the requested information is located and either return this knowledge to the requestor or attempt to have the request satisfied on its behalf. (The requestor may either be a DUA or another DSA: in the latter case both DSAs must have a chained port.)

Three modes of DSA interaction are defined to meet these requirements, namely "chaining", "multicasting", and "referral". "Chaining" and "multicasting" are defined to meet the latter of the above requirements whilst referrals address the former.

8.1 Chaining

This mode of interaction (depicted in Figure 2/X.518) may be used by one DSA, to pass on a request to another DSA when the former has knowledge about naming contexts held by the latter. Chaining may be used to contact a single DSA pointed to in a cross reference, a subordinate reference, or a superior reference. Multicasting is a form of chaining, described in § 8.2.

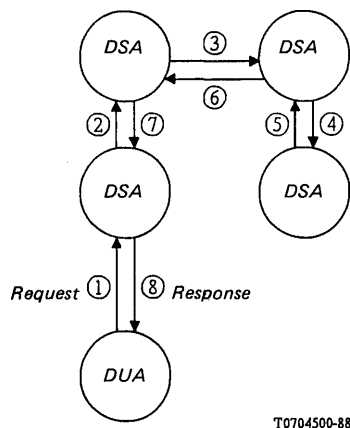


FIGURE 2/X.518

Chaining mode

Note - In Figure 2/X.518, the order of interactions is defined by the numbers associated with the interaction lines.

8.2 Multicasting

This mode of interaction (depicted in Figures 3a/X.518 and 3b/X.518) may be used by a DSA, to chain an identical request in parallel (a) or sequential (b) to one or more DSAs, when the former does not know the complete naming contexts held by the other DSAs. Multicasting is only used by a DSA to contact other DSAs pointed to in a non-specific subordinate reference. Each of the DSAs is passed the identical request. Normally, during name resolution, only one of the DSAs will be able to continue processing the remote operation, all of the others returning the **unableToProceed ServiceError**. However, during the evaluation phase of search and list operations, all DSAs in a non-specific subordinate reference should be able to continue processing the request.

Note - In Figures 3a/X.518 and 3b/X.518, the order of interactions is defined by the numbers associated with the interaction lines.

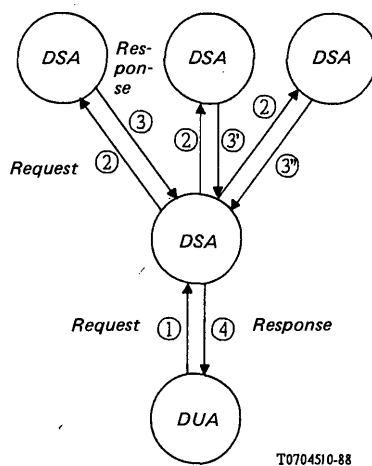


FIGURE 3a/X.518

Multicasting mode

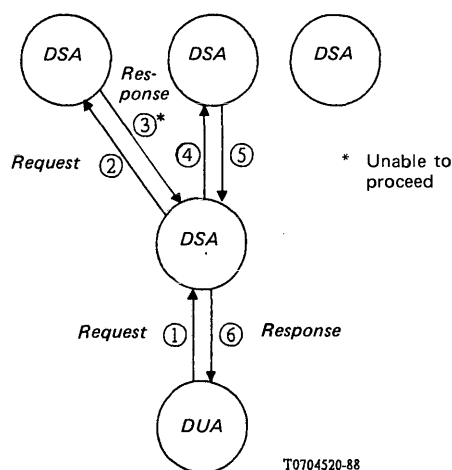


FIGURE 3b/X.518

Multicasting mode

8.3 Referral

A referral (depicted in Figures 4a/X.518 and 4b/X.518) is returned by a DSA in its response to a request which it had been requested to perform, either by a DUA, or by another DSA (in which case both DSAs must have a chained-service port). The referral may constitute the whole response (in which case it is categorized as an error) or just part of the response. The referral contains a knowledge reference, which may be either a superior, subordinate, cross or non-specific subordinate reference.

The DSA (Figure 4a/X.518) receiving the referral may use the knowledge reference contained therein, to subsequently chain or multicast (depending upon the type of reference) the original operation to other DSAs. Alternatively, a DSA receiving a referral, may in turn pass the referral back in its response. A DUA (Figure 4b/X.518) receiving a referral may use it to contact one or more other DSAs to progress the request.

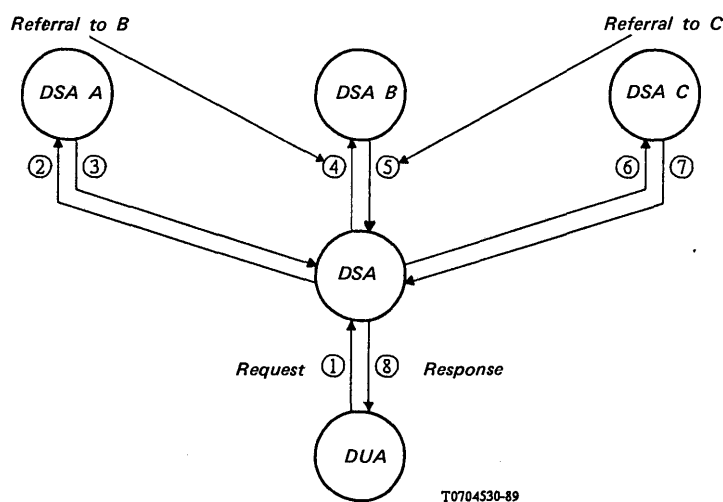


FIGURE 4a/X.518

Referral mode - DSA with chained port

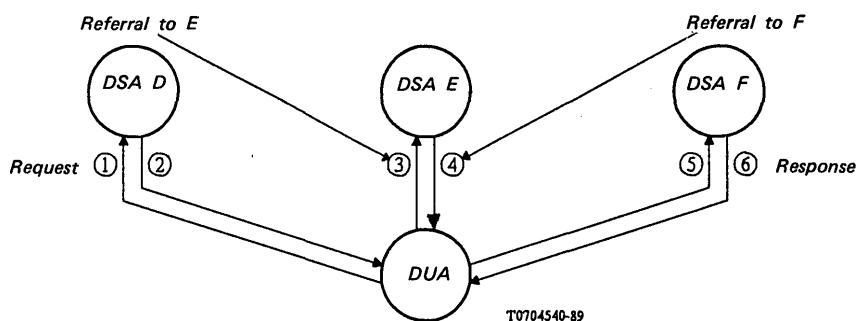


FIGURE 4b/X.518

Referral mode - DUA requests DSAs with no chained ports

Note - In Figures 4a/X.518 and 4b/X.518, the order of interactions is defined by the numbers associated with the interaction lines.

8.4 Mode determination

If a DSA cannot itself fully resolve a request, it must chain/multicast the request (or a request formed by decomposing the original one), to another DSA, unless:

- a) chaining is prohibited by the user via the service controls, in which case the DSA must return a referral or a **chainingRequired ServiceError** (at its choice), or
- b) the DSA has administrative, operational, or technical reasons for preferring not to chain, in which case the DSA must return a referral.

Note 1 - A "technical reason" for not chaining/multicasting is that the DSA identified in the knowledge reference has no chained service ports.

Note 2 - If the **localScope** service control is set, then the DSA (or DMD) must either resolve the request or return an error.

Note 3 - If the user prefers referrals, the user should set **chainingProhibited**.

9 Directory distribution

This paragraph defines the principles according to which the DIB can be distributed.

Each entry within the DIB is administered by one, and only one, DSA's Administrator who is said to have administrative authority for that entry. Maintenance and management of an entry must take place in a DSA administered by the administrative authority for the entry.

Although the Directory does not provide any support for the replication of entries, it is nevertheless possible to realize replication in two ways:

- Copies of an entry may be stored in other DSA(s) through bilateral agreement. The means by which these copies are maintained and managed is a function of the bilateral agreement and is not defined in this Recommendation.
- Copies of an entry may be acquired by storing (locally and dynamically) a copy of an entry which results from a request.

Note - The acquisition of cache entries is subject to access control.

The originator of the request is informed (via **fromCopy**) as to whether information returned in response to a request is from a replicated entry or not. A service control, **dontUseCopy**, is defined which allows the user to prohibit the use of replicated entries.

Each DSA within the Directory holds a *fragment* of the DIB. The DIB fragment held by a DSA is described in terms of the DIT and comprises one or more naming contexts. A *naming context* is a

partial subtree of the DIT defined as starting at a vertex and extending downwards to leaf and/or non-leaf vertices. Such vertices constitute the border of the naming context. Subordinates of the non-leaf vertices belonging to the border denote the start of further naming contexts.

It is possible for a DSA's administrator to have administrative authority for several disjoint naming contexts. For every naming context for which a DSA has administrative authority, it must logically hold the sequence of RDNs which lead from the root of the DIT to the initial vertex of the subtree comprising the naming context. This sequence of RDNs is called the *context prefix*.

A DSA's administrator may delegate administrative authority for any immediate subordinates of any entry held locally to another DSA. A DSA that delegated authority is called a *superior DSA* and the context that holds the superior entry of one for which the administrative authority was delegated, is called the *superior naming context*. Delegation of administrative authority begins with the root and proceeds downwards in the DIT; that is, it can only occur from an entry to its subordinates.

Figure 5/X.518 illustrates a hypothetical DIT logically partitioned into five naming contexts (named A, B, C, D and E), which are physically distributed over three DSAs (DSA1, DSA2, and DSA3).

From the example it can be seen that the naming contexts held by particular DSAs may be configured so as to meet a wide range of operational requirements. Certain DSAs may be configured to hold those entries that represent higher level naming domains within some logical part(s) of the DIB, the organizational structure of a large company say, but not necessarily all the subordinate entries. Alternatively, DSAs may be configured to hold only those naming contexts representing primarily leaf entries.

From the above definitions, the limiting case for a naming context can be either a single entry or the whole of the DIT.

Whilst the logical to physical mapping of the DIT onto DSAs is potentially arbitrary, the task of information location and management is simplified if the DSAs are configured to hold a small number of naming contexts.

In order for a DUA to begin processing a request it must hold some information, specifically the presentation address, about at least one DSA that it can contact initially. How it acquires and holds this information is a local matter.

During the process of modification of entries it is possible that the directory may become inconsistent. This will be particularly likely if modification involves aliases or aliased objects which may be in different DSAs. The inconsistency must be corrected by specific administrator action, for example to delete aliases if the corresponding aliased objects have been deleted. The Directory continues to operate during this period of inconsistency.

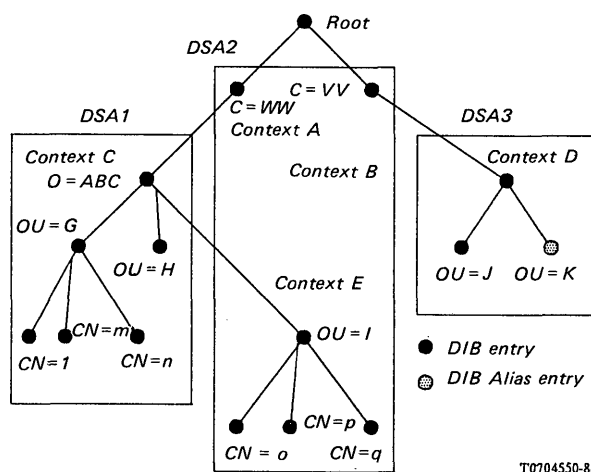


FIGURE 5/X.518
Hypothetical DIT

Note - The Root is not held by any DSA, however some indication must exist at the local level to distinguish those vertices (e.g. C = VV, C = WW) which are immediate subordinates of the Root.

10 Knowledge

The DIB is potentially distributed across multiple DSAs with each DSA holding a DIB fragment; the principles that govern distribution of the DIB are specified in § 9 of this Recommendation.

It is a requirement of the Directory that, for particular modes of user interaction, the distribution of the directory be rendered transparent, thereby giving the effect that the whole of the DIB appears to be within each and every DSA.

In order to support the operational requirements described above, it is necessary that each DSA holding a fragment of the DIB be able to identify and optionally interact with other fragments of the DIB held by other DSAs.

This paragraph defines knowledge as the basis for the mapping of a name to its location within a fragment of the DIT.

Conceptually DSAs hold two types of information:

- a) Directory Information;
- b) Knowledge Information.

Directory Information is the collection of entries comprising the Naming Context(s) for which the Administrator of a particular DSA has Administrative Authority.

Knowledge Information embodies the Naming Context(s) held by a particular DSA and denotes how these fit into the overall DIT hierarchy. Name Resolution, the process of locating the DSA which has Administrative Authority for a particular entry given that entry's name, is based on knowledge information.

A *Context Prefix* is the sequence of RDNs leading from the Root of the DIT to the initial vertex of a naming context and corresponds to the distinguished name of that vertex.

A *Naming Context* comprises a collection of knowledge references and a Context Prefix. A Naming Context must contain exactly the following knowledge references:

- All the internal references which define the internal structure of the portion of the DIT included in the Naming Context.
- All the subordinate and non-specific subordinate references to other Naming Contexts.

10.1 Minimal knowledge references

It is a property of the Directory that each entry can be accessed independently of where a request is generated.

To accomplish this, each DSA shall at least maintain the following knowledge references:

- *subordinate references* as defined in § 10.3.2 and/or *non-specific subordinate references* as defined in § 10.3.5; and
- *superior references* as defined in § 10.3.3.

It is then possible to establish a *reference path*, as a continuous sequence of knowledge references, to all naming contexts within the Directory.

Optionally, *cross references*, as defined in § 10.3.4 may form part of a reference path to optimize performance.

10.2 Root context

Because of the autonomy of the different countries or global organizations, there is likely to be no "single" DSA which holds the root context. The functionality of a "root-DSA" concerning the name resolution process has to be provided by those DSAs which have administrative authority for naming contexts that are immediately subordinate to the root. These DSAs are called *First Level DSAs*. Each First Level DSA must be able to simulate the functionality of the "root-DSA". This

requires full knowledge about the root naming context. The root context is replicated onto each First Level DSA and therefore has to be administered commonly by the autonomous first level administrative authorities. Administration procedures have to be determined by multilateral agreements outside the scope of this Recommendation.

- Each first level DSA shall hold the root context, which implies a reference path to each other first level DSA.
- Each non-first level DSA shall have a superior reference, which implies a reference path to any arbitrary first level DSA.

10.3 *Knowledge references*

The knowledge possessed by a DSA is defined in terms of a set of one or more knowledge references where each reference associates, either directly or indirectly, entries of the DIB with DSAs which hold those entries.

To be able to fulfill the requirements to reach every DIB entry from any DSA, every DSA is required to have knowledge about the entries which it itself holds, and about subordinates and possibly superiors thereof. This gives rise to the following types of knowledge references:

- Internal references
- Subordinate references
- Superior references
- Non-specific subordinate references.

Additionally, for optimization purposes the following type of optional reference is defined:

- Cross references

In the event that the set of knowledge references associated with a particular DSA contain only internal references, the DSA has no knowledge of other DSAs and the DIB is therefore centralized.

10.3.1 *Internal references*

An *internal reference* consists of:

- the RDN corresponding to a DIB entry;
- an internal pointer to where the entry is stored in the local DIB. (The specification of this pointer is outside the scope of this Recommendation.)

All entries for which a particular DSA has Administrative Authority are represented by internal references in the knowledge information of that DSA.

10.3.2 *Subordinate references*

A *subordinate reference* consists of:

- an RDN corresponding to an immediate subordinate DIB entry;
- the Access Point of the DSA to which Administrative Authority for that entry was delegated.

All subordinate entries held by another DSA to which this DSA has delegated Administrative Authority, must be represented by subordinate references (or non-specific subordinate references as described in § 10.3.5).

10.3.3 *Superior references*

A *superior reference* consists of:

- the Access Point of a DSA.

Each non-first level DSA maintains precisely one superior reference. The superior reference shall form part of a reference path to the root. Unless some method outside of the standard is employed to ensure this, for example within a DMD, this shall be accomplished by referring to a DSA which holds a naming context whose context prefix has less RDNs than the context prefix with fewest RDNs held by this DSA.

If a new non-first level DSA is introduced, it must have a minimal initial knowledge, which is represented by the superior reference. Any further knowledge will be added by subordinate references

or cross references (as described in § 10.3.4). If a new first level DSA is introduced, it must acquire the root context and advise all other first level DSAs. How this is accomplished is outside the scope of this Recommendation.

10.3.4 *Cross reference*

A *cross reference* consists of:

- a Context Prefix;
- the Access Point of a DSA which has Administrative Authority for that Naming Context.

This type of reference is optional and serves to optimize Name Resolution. A DSA may hold any number (including zero) of cross references.

10.3.5 *Non-specific subordinate references*

A *non-specific subordinate reference* consists of:

- The Access Point of a DSA which holds one or more immediately subordinate Naming Contexts.

This type of reference is optional, to allow for the case in which a DSA is known to contain some subordinate entries but the specific RDNs of those entries is not known.

For each naming context which it holds, a DSA may hold any number (including zero) of non-specific subordinate references, which will be evaluated if all specific internal and subordinate references have been pursued. DSAs accessed via a non-specific reference must be able to resolve the request directly (either success or failure). In the event of failure a **ServiceError** reporting a problem of **unableToProceed** is returned to the requestor.

10.4 *Knowledge administration*

To operate a widely distributed Directory with an acceptable degree of consistency and performance, procedures are required to maintain and extend the knowledge held by each DSA. The same procedures are appropriate for creating initial knowledge.

Knowledge can be maintained by:

- a) The DSA or its administrative authority propagating changes of knowledge to those DSAs holding all kinds of references to it, whenever changes at that DSA cause the references to become invalid. This is the only way superior, subordinate and non-specific subordinate references can be maintained.
- b) DSAs requesting and obtaining cross references to improve the performance through ordinary directory operations.

This Recommendation does not define any procedures for propagating knowledge changes as described in a). Bilateral agreements must be established locally for this.

10.4.1 *Requesting cross reference*

To improve the performance of the Directory System, the local set of cross references can be expanded using ordinary Directory operations. If a DSA has a chained port it may request another DSA (which also must have a chained port) to return those knowledge references which contain information about the location of naming contexts related to the target object name of an ordinary Directory operation.

If the **returnCrossReference** component of the **ChainingArgument** is set to **TRUE**, the **crossReference** component of the **ChainingResult** may be present, consisting of a sequence of cross reference items.

If a DSA is not able to chain a request to the next DSA a referral is returned to the originating DSA. If the **returnCrossReference** component of the chaining argument was **TRUE**, the referral may contain additionally the context prefix of the naming context which the referral refers to. The **contextPrefix** component is absent if the referral is based on a non-specific subordinate reference. The cross reference returned by a referral is only based on knowledge held by the DSA which generated the referral.

In both cases (chaining result and referral) an administrative authority through its DSA may elect to ignore the request for returning cross references.

10.4.2 *Knowledge inconsistencies*

The Directory has to support consistency-checking mechanisms to guarantee a certain degree of knowledge consistency.

10.4.2.1 *Detection of knowledge inconsistencies*

The kind of inconsistency and its detection varies for the different types of knowledge references.

- *Cross and subordinate references:*

This type of reference is invalid if the referenced DSA does not have a local naming context with the context prefix contained in the reference. This inconsistency will be detected during the determination of the initial naming context of the name resolution process by the operation progress and reference type components of the **ChainingArgument**.

- *Non-specific Subordinate-references:*

This type of reference is invalid if the referenced DSA does not have a local naming context whose immediately superior context prefix is contained in the reference, i.e. the reference contains that DSA's local context prefix minus the last RDN. The consistency check is applied as above.

- *Superior references:*

An invalid superior reference is one which does not form part of a reference path to the root. The maintenance of superior references must be done by external means and is outside the scope of this Recommendation.

Note - It is not always possible to detect an invalid superior reference.

10.4.2.2 *Reporting of knowledge inconsistencies*

If chaining is used in performing a Directory request, all knowledge inconsistencies will be detected by the DSA which holds the invalid knowledge reference, through receiving a **ServiceError** with problem of **invalidReference**.

If a DSA returns a referral which is based on an invalid knowledge reference, the requestor will be returned a **ServiceError** with problem of **invalidReference** if it uses the referral. How the error condition will be propagated to the DSA which stores the invalid reference is not within the scope of this Recommendation.

10.4.2.3 *Treatment of inconsistent knowledge references*

After a DSA has detected an invalid reference it should try to re-establish knowledge consistency. For example, this can be done by simply deleting an invalid cross reference or by replacing it with a correct one which can be obtained using the **requestCrossReferences** mechanisms.

The way in which a DSA actually handles invalid references is a local matter, and outside the scope of this Recommendation.

SECTION 4 - *DSA abstract service*

11 **Overview of DSA abstract service**

11.1 The abstract service of the directory is fully described in Recommendation X.511. When such a service is provided in a distributed environment, as modelled in § 7 of this Recommendation, it can be regarded as being provided by means of a set of DSAs. This is illustrated in Figure 1/X.518.

11.2 To describe this model, the refinement of the **directory** object into its component **dsa** objects can be expressed as:

```

DirectoryRefinement ::= REFINE directory AS
dsa                RECURRING
    readPort       [S] VISIBLE
    searchPort      [S] VISIBLE
    modifyPort      [S] VISIBLE
    chainedReadPort PAIRED with dsa
    chainedSearchPort PAIRED with dsa
    chainedModifyPort PAIRED with dsa

```

11.3 The **dsa** object itself can be defined as follows:

```

dsa OBJECT
    PORTS { readPort      [S],
            searchPort    [S],
            modifyPort     [S],
            chainedReadPort,
            chainedSearchPort,
            chainedModifyPort}
::= id-ot-dsa

```

The DSA supplies Read, Search and Modify ports, thus making visible those services to the users of the directory object, namely the DUAs. In addition, a DSA supports "chained" versions of these ports, namely Chained Read, Chained Search, and Chained Modify, which allow DSAs to propagate requests for those services to other DSAs.

11.4 The ports cited from §§ 11.2 and 11.3 (excluding those which are defined in Recommendation X.511) are defined as follows:

```

chainedReadPort    PORT
    ABSTRACT OPERATIONS {
        ChainedRead, ChainedCompare,
        ChainedAbandon}
    ::= id-pt-chained-read

chainedSearchPort  PORT
    ABSTRACT OPERATIONS {
        ChainedList, ChainedSearch}
    ::= id-pt-chained-search

chainedModifyPort  PORT
    ABSTRACT OPERATIONS {
        ChainedAddEntry,
        ChainedRemoveEntry,
        ChainedModifyEntry,
        ChainedModifyRDN}
    ::= id-pt-chained-modify

```

12 Information types

12.1 Introduction

12.1.1 This paragraph identifies, and in some cases defines, a number of information types which are subsequently used in the definition of various of the operations of the DSA abstract service. The information types concerned are those which are common to more than one operation, are likely to be in the future, or which are sufficiently complex or self-contained as to merit being defined separately from the operation which uses them.

12.1.2 Several of the information types used in the definition of the DSA abstract service are actually defined elsewhere. § 12.2 identifies these types and indicates the source of their definition. Each of the remaining (§§ 12.3 to 12.9) identifies and defines an information type.

12.2 Information types defined elsewhere

12.2.1 The following information types are defined in Recommendation X.501:

- a) **aliasedObjectName;**

- b) **DistinguishedName;**
- c) **Name;**
- d) **RelativeDistinguishedName.**

12.2.2 The following information types are defined in Recommendation X.511:

(Abstract-bind)

- a) **DirectoryBind;**

(Abstract-operations)

- b) **Abandon;**

(Abstract-errors)

- c) **Abandoned;**
- d) **AttributeError;**
- e) **NameError;**
- f) **SecurityError;**
- g) **ServiceError;**
- h) **UpdateError;**

(Macro)

- i) **OPTIONALLY-SIGNED;**

(Data Type)

- j) **SecurityParameters.**

12.2.3 The following information type is defined in Recommendation X.520:

- a) **PresentationAddress.**

12.3 Chaining arguments

12.3.1 The **ChainingArguments** are present in each Chained abstract-operation, to convey to a DSA the information needed to successfully perform its part of the overall task:

```
ChainingArguments ::= SET {
    originator           [0] DistinguishedName OPTIONAL,
    targetObject         [1] DistinguishedName OPTIONAL,
    operationProgress    [2] OperationProgress DEFAULT {notStarted},
    traceInformation     [3] TraceInformation,
    aliasDereferenced    [4] BOOLEAN DEFAULT FALSE,
    aliasedRDNs          [5] INTEGER OPTIONAL,
    -- absent unless aliasDereferenced is TRUE
    returnCrossRefs      [6] BOOLEAN DEFAULT FALSE,
    referenceType        [7] ReferenceType DEFAULT superior,
    info                 [8] DomainInfo OPTIONAL,
    timeLimit            [9] UTCTime OPTIONAL,
    [10] SecurityParameters DEFAULT {}
}
```

12.3.2 The various components have the meanings as defined in §§ 12.3.2.1 to 12.3.2.11.

12.3.2.1 The **originator** component conveys the name of the (ultimate) originator of the request, unless already specified in the security parameters. If **requestor** is present in **CommonArguments**, this argument may be omitted.

12.3.2.2 The **targetObject** component conveys the name of the object whose directory entry is being routed to. The role of this object depends on the particular abstract-operation concerned: it may be the object whose entry is to be operated on, or which is to be the base object for a request or sub-request involving multiple objects (e.g. **ChainedList** or **ChainedSearch**). This component may be omitted only if it would have had the same value as the base object parameter in **XArgument** (see § 14.3.1), in which case its implied value is that value.

12.3.2.3 The **operationProgress** component is used to inform the DSA of the progress of the operation, and hence of the role which it is expected to play in its overall performance. The information conveyed in this component is specified in § 12.5.

12.3.2.4 The **traceInformation** component is used to prevent looping among DSAs when chaining is in operation. A DSA adds a new element to trace information prior to chaining an operation to another DSA. On being requested to perform an operation, a DSA checks, by examination of the trace information, that the operation has not formed a loop. The information conveyed in this component is specified in § 12.6.

12.3.2.5 The **aliasDereferenced** component is a Boolean value which is used to indicate whether or not one or more alias entries have so far been encountered and dereferenced during the course of distributed name resolution. The default value of **FALSE** indicates that no alias entry has been dereferenced.

12.3.2.6 The **aliasedRDNs** component indicates how many of the RDNs in the **targetObjectName** have been generated from the **aliasedObjectName** attributes of one (or more) alias entries. The integer value is set whenever an alias entry is encountered and dereferenced. This component shall be present if and only if the **aliasDereferenced** component is **TRUE**.

12.3.2.7 The **returnCrossRefs** component is a Boolean value which indicates whether or not knowledge references, used during the course of performing a distributed operation, are requested to be passed back to the initial DSA, as cross references, along with a result or referral. The default value of **FALSE** indicates that such knowledge references are not to be returned.

12.3.2.8 The **referenceType** component indicates, to the DSA being asked to perform the abstract-operation, what type of knowledge was used to route the request to it. The DSA may therefore be able to detect errors in the knowledge held by the invoker. If such an error is detected it shall be indicated by a **ServiceError** with the **invalidReference** problem. **ReferenceType** is described fully in § 12.7.

Note - If the **referenceType** is missing, then the value **superior** shall be assumed.

12.3.2.9 The **info** component is used to convey DMD-specific information among DSAs which are involved in the processing of a common request. This component is of type **DomainInfo**, which is of unrestricted type:

DomainInfo ::= ANY

12.3.2.10 The **timeLimit** component, if present, indicates the time by which the operation is to be completed.

12.3.2.11 The **SecurityParameters** component is specified in Recommendation X.511. Its absence is deemed equivalent to there being an empty set of security parameters.

12.4 Chaining results

12.4.1 The **ChainingResults** are present in the result of each abstract-operation and provide feedback to the DSA which invoked the abstract-operation.

ChainingResults ::= SET {
 Info [0] **DomainInfo** OPTIONAL,
 crossReferences [1] **SEQUENCE OF CrossReference** OPTIONAL,
 [2] **SecurityParameters** DEFAULT {}

12.4.2 The various components have the meanings as defined in §§ 12.4.2.1 to 12.4.2.3.

12.4.2.1 The **info** component is used to convey DMD-specific information among DSAs which are involved in the processing of a common request. This component is of type **DomainInfo**, which is of unrestricted type.

12.4.2.2 The **crossReferences** component is not present in the **ChainingResults** unless the **returnCrossRefs** component of the corresponding request had the value **TRUE**. This component consists of a sequence of **CrossReference** items, each of which contains a **contextPrefix** and an **accessPoint** descriptor (see § 12.8).

CrossReference ::= SET{
 contextPrefix [0] **DistinguishedName**,
 accessPoint [1] **AccessPoint**}

A **CrossReference** may be added by a DSA when it matches part of the **targetObject** argument of an abstract-operation with one of its context prefixes. The administrative authority of a DSA may have a policy not to return such knowledge, and will in this case not add an item to the sequence.

12.4.2.3 The **SecurityParameters** component is specified in Recommendation X.511. Its absence is deemed equivalent to there being an empty set of security parameters.

12.5 Operation progress

12.5.1 An **OperationProgress** value describes the state of progress in the performance of an abstract-operation which several DSAs must participate in.

```

OperationProgress ::= SET {
    nameResolutionPhase [0]
        ENUMERATED {
            notStarted (1),
            proceeding (2),
            completed (3)},
    nextRDNTToBeResolved [1]
        INTEGER OPTIONAL}

```

12.5.2 The various components have the meanings as defined in §§ 12.5.2.1 and 12.5.2.2.

12.5.2.1 The **nameResolutionPhase** component indicates what phase has been reached in handling the **targetObject** name of an operation. Where this indicates that name resolution has **notStarted**, then a DSA has not hitherto been reached with a naming context containing the initial RDN(s) of the name. If name resolution is **proceeding**, then the initial part of the name has been recognized, though the DSA holding the target object has not yet been reached. The **nextRDNTToBeResolved** indicates how much of the name has already been recognized (§ 12.5.2.2). If name resolution is **completed**, then the DSA holding the target object has been reached, and performance of the operation proper is proceeding.

12.5.2.2 The **nextRDNTToBeResolved** indicates to the DSA which of the RDNs in the **targetObject** name is the next to be resolved. It takes the form of an integer in the range one to the number of RDNs in the name. This component is only present if the **nameResolutionPhase** component has the value **proceeding**.

12.6 Trace information

12.6.1 A **TraceInformation** value carries forward a record of the DSAs which have been involved in the performance of an operation. It is used to detect the existence of, or avoid, loops which might arise from inconsistent knowledge or from the presence of alias loops in the DIT.

```

TraceInformation ::= SEQUENCE OF TraceItem
TraceItem ::= SET {
    dsa [0] Name,
    targetObject [1] Name OPTIONAL,
    operationProgress [2] OperationProgress }

```

12.6.2 Each DSA which is propagating an operation to another adds a new item to the trace information. Each such **TraceItem** contains:

- a) the **Name** of the dsa which is adding the item;
- b) the **targetObject Name** which the DSA adding the item received on the incoming request. This parameter is omitted if the query being chained came from a DUA (in which case its implied value is the **object** or **baseObject** in **XOperation**), or if its value is the same as the (actual or implied) **targetObject** in the **ChainingArgument** of the outgoing request;
- c) the **operationProgress** which the DSA adding the item received on the incoming request.

12.7 Reference type

12.7.1 A **ReferenceType** value indicates one of the various kinds of reference defined in § 10.

```

ReferenceType ::=
    ENUMERATED {
        superior (1),
        subordinate (2),
        cross (3),
        nonSpecificSubordinate (4)}

```


12.8 Access point

12.8.1 An **AccessPoint** value identifies a particular point at which access to the Directory, specifically to a DSA, can occur. The access point has a **Name**, that of the DSA concerned, and a **PresentationAddress**, to be used in OSI communications to that DSA.

```
AccessPoint ::= SET {  
    ae-title      [0] Name,  
    address       [1] PresentationAddress }
```

12.9 Continuation reference

12.9.1 A **ContinuationReference** describes how the performance of all or part of an abstract-operation can be continued at a different DSA or DSAs. It is typically returned as a referral when the DSA involved is unable or unwilling to propagate the request itself.

```
ContinuationReference ::= SET {  
    targetObject      [0] Name,  
    aliasedRDNs       [1] INTEGER OPTIONAL,  
    operationProgress [2] OperationProgress,  
    rdnsResolved      [3] INTEGER OPTIONAL,  
    referenceType     [4] ReferenceType OPTIONAL,  
    -- only present in the DSP  
    accessPoints      [5] SET OF AccessPoint}
```

12.9.2 The various components have the meanings as defined in §§ 12.9.2.1 to 12.9.2.6.

12.9.2.1 The **targetObject Name** which is proposed to be used in continuing the operation. This might be different from the **targetObject Name** received on the incoming request if, for example, an alias has been dereferenced, or the base object in a search has been located.

12.9.2.2 The **aliasedRDNs** component indicates how many (if any) of the RDNs in the target object name have been produced by dereferencing an alias. The argument is only present if an alias has been dereferenced.

12.9.2.3 The **operationProgress** which has been achieved, and which will govern the further performance of the abstract-operation by the DSAs named, should the DSA or DUA receiving the **ContinuationReference** follow it up.

12.9.2.4 The **rdnsResolved** component value, (which need only be present if some of the RDNs in the name have not been the subject of full name resolution, but have been assumed to be correct from a cross reference) indicates how many RDNs have actually been resolved, using internal references only.

12.9.2.5 The **referenceType** component, which is only present in the DSA abstract service, indicates what type of knowledge was used in generating this continuation.

12.9.2.6 The **accessPoints** component indicates the access points which are to be followed up to achieve this continuation. Where Nonspecific Subordinate References are involved there may be more than one **AccessPoint** listed, and each should be followed up, e.g. by multicasting.

13 Abstract-bind and abstract-unbind

DSABind and **DSAUnbind**, respectively, are used by a DSA at the beginning and at the end of a period accessing another DSA.

13.1 DSA bind

13.1.1 A **DSABind** abstract-bind-operation is used by a DSA to bind its **chainedRead**, **chainedSearch**, and **chainedModify** ports to those of another DSA.

```
DSABind      ::= ABSTRACT-BIND  
TO           {chainedRead,  
              chainedSearch,  
              chainedModify}  
  
DirectoryBind
```

13.1.2 The components of the **DSABind** are identical to their counterparts in the **DirectoryBind** (see Recommendation X.511) with the following differences.

13.1.2.1 The **Credentials** of the **DirectoryBindArgument** allows information identifying the AE-Title of the initiating DSA to be sent to the responding DSA. The AE-Title must be in the form of a Directory Distinguished Name.

13.1.2.2 The **Credentials** of the **DirectoryBindResult** allows information identifying the AE-Title of the responding DSA to be sent to the initiating DSA. The AE-Title must be in the form of Distinguished Name.

13.2 DSA unbind

13.2.1 A **DSAUnbind** operation is used to unbind the Chained Read, Chained Search and Chained Modify ports of a pair of DSAs.

DSAUnbind	::=	ABSTRACT-UNBIND
FROM		{ chainedRead ,
		chainedSearch ,
		chainedModify }

13.2.2 There are no arguments, results or errors.

14 Chained abstract-operations

14.1 Corresponding to each of the ports of the Directory abstract service is a port of the DSA which allows the abstract service to be provided by cooperating DSAs. The abstract-operations in the corresponding ports are also in one-to-one correspondence. The names of the ports and the abstract-operations have been chosen to reflect this correspondence, with the port or abstract-operation in the DSA abstract service being formed from that of the Directory abstract service by prefixing the word "Chained". The resulting ports and abstract-operations are as follows:

ChainedReadPort:	ChainedRead ,
	ChainedCompare ,
	ChainedAbandon
ChainedSearchPort:	ChainedList ,
	ChainedSearch
ChainedModifyPort:	ChainedAddEntry ,
	ChainedRemoveEntry ,
	ChainedModifyEntry ,
	ChainedModifyRDN

14.2 The arguments, results, and errors of the chained abstract-operation are, with one exception, formed systematically from the arguments, results, and errors of the corresponding abstract-operations in the Directory abstract service (as described in § 14.3). The one exception is the **ChainedAbandon** abstract-operation, which is syntactically equivalent to its Directory abstract-service counterpart (described in § 14.4).

14.3 A **ChainedX** abstract-operation is used to propagate between DSAs a request which (normally) originated as a DUA invoking an **X** abstract-operation at a DSA, that DSA having elected to chain it. The arguments of the abstract-operation may optionally be signed by the invoker, and, if so requested, the performing DSA may sign the results.

14.3.1 The systematic derivation of a Chained abstract-operation **ChainedX** from its counterpart **X** is as follows:

given:

X	::=	ABSTRACT-OPERATION
		ARGUMENT XArgument
		RESULT XResult
		ERRORS {..., Referral,...}

the Chained abstract-operation is derived as:

```
ChainedX ::=  
  ABSTRACT-OPERATION  
    ARGUMENT OPTIONALLY-SIGNED SET{  
      ChainingArgument,  
      [0] XArgument}  
    RESULT OPTIONALLY-SIGNED SET{  
      ChainingResult,  
      [0] XResult}  
    ERRORS {...,DsaReferral,...}
```

Note - The definitive specification of the DSA abstract service in Annex A applies this derivation in full to the Chained abstract-operations.

14.3.2 The arguments of the derived abstract-operation have the meanings as described in §§ 14.3.2.1 and 14.3.2.2.

14.3.2.1 The **ChainingArgument** contains that information, over and above the original DUA-supplied arguments, which is needed in order for the performing DSA to carry out the operation. This information type is defined in § 12.3.

14.3.2.2 The **XArgument** contains the original DUA-supplied arguments, as specified in the appropriate clause of Recommendation X.511.

14.3.3 Should the request succeed, the result will be returned. The result parameters have the meanings as described in §§ 14.3.3.1 and 14.3.3.2.

14.3.3.1 The **ChainingResult** contains the information, over and above that to be supplied to the originating DUA, which may be needed by previous DSAs in a chain. This information type is defined in § 12.4.

14.3.3.2 The **XResult** contains the result which is being returned by the performer of this abstract-operation, and which is intended to be passed back in the result to the originating DUA. This information is as specified in the appropriate clause of Recommendation X.511.

14.3.4 Should the request fail, one of the listed errors will be returned. The set of errors which may be reported are as described for the corresponding abstract-operation in Recommendation X.511, except that **DSAReferral** is returned instead of **Referral**. The various errors are defined or referenced in § 15.

14.4 A **ChainedAbandon** abstract-operation is used by one DSA to indicate to another that it is no longer interested in having a previously invoked chained operation performed. This may be for any of a number of reasons, of which the following are examples:

- a) the operation which led to the DSA originally chaining has itself been abandoned, or has implicitly been aborted by the breakdown of an association;
- b) the DSA has obtained the necessary information in another way, e.g. from a faster responding DSA involved in a multicast.

A DSA is never obliged to issue a **ChainedAbandon**, or indeed to actually abandon an operation if requested to do so.

If **ChainedAbandon** actually succeeds in stopping the performance of an operation, then a result will be returned, and the subject operation will return an **Abandoned** abstract-error. If the **ChainedAbandon** does not succeed in stopping the operation, then it itself will return an **AbandonFailed** error.

15 Chained abstract-errors

15.1 Introduction

15.1.1 For the most part, the same abstract-errors can be returned in the DSA abstract service which can be returned in the Directory abstract-service. The exceptions are that the **DSAReferral** "error" is returned (see § 15.2), instead of **Referral**, and the following service problems have the same abstract syntax but different semantics.

- a) **invalidReference.**
- b) **loopDetected.**

15.1.2 The precedence of the abstract-errors which may occur is as for their precedence in the Directory abstract service, as specified in Recommendation X.511.

15.2 *DSA Referral*

15.2.1 The **DSAReferral** abstract-error is generated by a DSA when, for whatever reason, it doesn't wish to continue performing an abstract-operation by chaining or multicasting the abstract-operation to one or more other DSAs. The circumstances where it may return a referral are described in § 8.4.

```

DSAReferral ::=
  ABSTRACT-ERROR
    PARAMETER SET{
      [0] ContinuationReference,
      contextPrefix [1] DistinguishedName OPTIONAL }

```

15.2.2 The various parameters have the meanings as described in §§ 15.2.2.1 and 15.2.2.2.

15.2.2.1 The **ContinuationReference** contains the information needed by the invoker to propagate an appropriate further request, perhaps to another DSA. This information type is specified in § 12.9.

15.2.2.2 If the **returnCrossRefs** component of the **ChainingArguments** for this abstract-operation had the value **TRUE**, and the referral is being based upon a subordinate or cross-reference, then the **contextPrefix** parameter may optionally be included. The administrative authority of any DSA will decide which knowledge references, if any, can be returned in this manner (the others, for example, may be confidential to that DSA).

SECTION 5 - *Distributed operations procedures*

16 **Introduction**

16.1 *Scope and limits*

This paragraph specifies the procedures for distributed operation of the Directory which are performed by DSAs. Each DSA individually performs the procedures described below: the collective action of all DSAs produces the full set of services provided to users by the Directory.

The description of procedures for a single DSA is based on the models in §§ 7 to 10 of this Recommendation.

It should be noted that the model and procedures are included for expositional purposes only and are not intended to constrain or govern the implementation of an actual DSA.

This paragraph is divided into three sub-paragraphs: this introduction, a conceptual model for describing directory behaviour and an introduction of both DSA-Centred and Operation-Centred models of DSA operations.

16.2 *Conceptual model*

The complexity of the Directory's distributed operation gives rise to a need for conceptual modelling using both narrative and pictorial descriptive techniques. However, neither the narrative nor graphic diagrams should be construed as a formal description of distributed directory operation.

16.3 *Individual and cooperative operation of DSAs*

The model views DSA operation from two separate perspectives, which, taken together, provide a complete, operational picture of the Directory.

- a) **DSA-Centred Perspective.** In this perspective the set of procedures that support the directory is described from the viewpoint of a single DSA. This makes it possible to provide a definitive specification of each procedure and to fully account for their interrelationships and overall control structure. § 18 describes the DSA procedures from a DSA-centred perspective.
- b) **Operation-Centred Perspective.** The DSA-centred view provides complete detail but makes it difficult to understand the structure of individual operations, which may undergo processing by multiple DSAs. Consequently § 17 adopts a primarily operation-centred view to introduce the processing phases applicable to each.

To support the distributed operation of the directory, each DSA must perform actions needed to realize the intent of each operation and additional actions needed to distribute that realization across multiple DSAs. § 17 explores the distinction between these two kinds of actions. In § 18 both kinds of actions are specified in detail.

17 Distributed directory behaviour

17.1 Cooperative fulfillment of operations

Each DSA is equipped with procedures capable of completely fulfilling all Directory operations. In the case that a DSA contains the entire DIB all operations are, in fact, completely carried out within that DSA. In the case that the DIB is distributed across multiple DSAs the completion of a typical operation is fragmented, with just a portion of that operation carried out in each of potentially many cooperating DSAs.

In the distributed environment, the typical DSA sees each operation as a transitory event; the operation is invoked by a DUA or some other DSA; the DSA carries out processing on the object and then directs it toward another DSA for further processing.

An alternate view considers the total processing experienced by an operation during its fulfillment by multiple, cooperating DSAs. This perspective reveals the common processing phases that apply to all operations.

17.2 Phases of operation processing

Every Directory operation may be thought of as comprising three distinct phases:

- a) the **Name Resolution phase** - in which the name of the object on whose entry a particular operation is to be performed is used to locate the DSA which holds the entry;
- b) the **Evaluation phase** - in which the operation specified by a particular directory request (e.g. read) is actually performed;
- c) the **Results Merging phase** - in which the results of a specified operation are returned to the requesting DUA. If a chaining mode of interaction was chosen, the Results Merging phase may involve several DSAs, each of which chained the original request or sub-request (as defined in § 17.3.1 Request Decomposition) to another DSA during either or both of the preceding phases.

In the case of the operations **Read**, **Compare**, **List**, **Search**, and **ModifyEntry**, name resolution takes place on the object name provided in the argument of the operation. In the case of **AddEntry**, **RemoveEntry**, and **ModifyRDN**, name resolution takes place on the name of the immediately superior object (derived by removing the final RDN from the name provided in the operation argument).

An operation on a particular entry may initially be directed at any DSA in the Directory. That DSA used its knowledge, possibly in conjunction with other DSAs to process the operation through the three phases.

17.2.1 Name resolution phase

Name Resolution is the process of sequentially matching each RDN in a purported Name to an arc (or vertex) of the DIT, beginning logically at the Root and progressing downwards in the DIT. However, because the DIT is distributed between arbitrarily many DSAs, each DSA may only be able to perform a fraction of the name resolution process. A given DSA performs its part of the Name Resolution process by traversing its local knowledge. This process is described in § 18.6 and the

accompanying diagrams (Figures 11/X.518 to 13/X.518). When a DSA reaches the border of its naming context, it will know from the knowledge information contained therein, whether the resolution can be continued by another DSA or whether the name is erroneous.

17.2.2 *Evaluation phase*

When the name resolution phase has been completed, the actual operation required (e.g. read or search) is performed.

Operations that involve a single entry - **Read**, **Compare**, **AddEntry**, **RemoveEntry**, **ModifyRDN** and **ModifyEntry** - can be carried out entirely within the DSA in which that entry has been located. **AddEntry**, **RemoveEntry** and **ModifyRDN** may affect knowledge in more than one DSA. See § 18.7.1.

Operations that involve multiple entries - **List** and **Search** - need to locate subordinates of the target, which may or may not reside in the same DSA. If they do not all reside in the same DSA, operations need to be directed to the DSAs specified in the subordinate references to complete the evaluation process.

17.2.3 *Results merging phase*

The results merging phase is entered once some of the results of the evaluation phase are available.

In those cases where the operation affected only a single entry, the result of the operation can simply be returned to the requesting DUA. In those cases where the operation has affected multiple entries on multiple DSAs, results need to be combined.

The permissible responses returned to a requestor after results merging include:

- a) a complete result of the operation;
- b) a result which is not complete because some parts of the DIT remain unexplored (applies to **List** and **Search** only). Such a *partial result* may include continuation references for those parts of the DIT not explored;
- c) an error (a referral being a special case);
- d) and if the requestor was a DSA, a **ChainingResult**.

17.3 *Managing distributed operations*

Information is included in the argument of each abstract-operation which a DSA may be asked to perform indicating the progress of each operation as it traverses various of the DSAs of the Directory. This makes it possible for each DSA to perform the appropriate aspect of the processing required, and to record the completion of that aspect before directing the operation outward toward further DSAs.

Additional procedures are included in the DSA to physically distribute the operations and support other needs arising from their distribution.

17.3.1 *Request decomposition*

Request decomposition is a process performed internally by a DSA prior to communication with one or more other DSAs. A request is decomposed into several sub-requests such that each of the latter accomplishes a part of the original task. Request decomposition can be used, for example, in the search operation, after the base object has been found. After decomposition, each of the sub-requests may then be chained or multicast to other DSAs, to continue the task.

17.3.2 *DSA as Request responder*

A DSA that receives a request can check the progress of that request using the **Operation Progress** parameter. This will determine whether the operation is still in the name resolution phase or has reached the evaluation phase, and what portion of the operation the DSA should attempt to satisfy. If the DSA cannot fully satisfy the request it must either pass the operation on to one or more DSAs which can help to fulfill the request (by chaining or multicasting) or return a referral to another DSA or terminate the request with an error.

17.3.3 Completion of operations

Each DSA that has initiated an operation or propagated an operation to one or more other DSAs must keep track of that operation's existence until each of the other DSAs has returned a result or error, or the operation's maximum time limit has expired. This requirement applies to all operations, propagation modes and processing phases. It ensures the orderly closing down of distributed operations that have propagated out into the Directory.

17.4 Other considerations for distributed operation

17.4.1 Request validation

On receipt of a directory operation a DSA must initially validate the operation to ensure that it can be progressed. Circumstances such as loops within the DIT caused by inappropriate use of aliases or the use of erroneous knowledge may cause operations to be sent to DSAs that cannot be processed.

In the simple case these erroneous circumstances are adequately handled by name resolution procedures as described in § 18. However, where circumstances cause operations to loop (as described in § 17.4.3) name resolution alone is inadequate.

The request validation actions ensures that a loop is detected before any attempt is made to progress an operation through the erroneous data caused by the loop. The detection process is carried out by the loop detection procedure specified in § 18.5.1.

Where security procedures are in force request validation also verifies the identity of the requesting DSA or DUA, and the validity of the request.

17.4.2 State and trace information

The progression of an operation within the directory and the presence of loop conditions are determined by an operation's "state", where state is defined to be the following:

- the name of the DSA currently processing the operation;
- the name of the **targetObject** as contained within the argument of the operation;
- the **operationProgress** as contained within the argument of the operation and as defined in § 12.5.

In addition to the current state of an operation, a DSA also needs to know all previous states for that operation. These are recorded in the **traceInformation** argument and conveyed with the operation.

The **traceInformation** argument forms the basis of loop avoidance/detection strategies as specified in § 17.4.3.

17.4.3 Looping

Within the context of a particular directory operation a loop occurs if at any time the operation returns to a previous state (as defined above). Looping is managed using the **traceInformation** argument. Two strategies are defined to handle loops. In *loop detection* a DSA determines whether a loop has occurred in an incoming operation and, if so returns an error. In *loop avoidance* a DSA determines whether an operation, if forwarded, would yield a loop.

17.4.4 Service controls

Some service controls need special consideration in the distributed environment in order that the operation is processed the way that was requested.

- a) **chainingProhibited**: A DSA consults this service control when determining the mode of propagation of an operation. If it is set then the DSA always uses referral mode. If, however, it is not set, the DSA can choose whether to use chaining or referral depending on its capabilities.
- b) **timeLimit**: A DSA needs to take account of this service control to ensure that the time limit is not exceeded in that DSA. A DSA requested to perform an operation by a DUA, initially heeds the **timeLimit** expressed by the DUA as the available elapsed time in seconds for completion of the operation. If chaining is required, the **timeLimit** is included in the chaining argument to be passed to the next DSA(s). In this case the same value of the limit is used for each chained request, and is the (UTC) time by which the

operation must be completed to meet the originally specified constraint. On receiving a chaining argument with a **timeLimit** specified, the receiving DSA respects this limit.

- c) **sizeLimit**: A DSA needs to take account of this service control to ensure that the list of results does not exceed the size specified. The limit, as included in the common argument of the original request, is conveyed unchanged as the request is chained/multicast. If request decomposition is required, the same value is included in the argument to be passed to the next DSA: that is, the full limit is used for each sub-request. When the results are returned the requestor DSA resolves the multiple results and applies the limit to the total to ensure that only the requested numbers are returned. If the limit has been exceeded, this is indicated in the reply.
- d) **Priority**: In all modes of propagation, each DSA is responsible for ensuring that the processing of operations is ordered so as to support this service control if present.
- e) **localScope**: The operation is limited to a locally defined scope and cannot be propagated by any of the modes.
- f) **scopeOfReferral**: If the DSA returns a referral or partial result to a **List** or **Search** operation, then the embedded **ContinuationReferences** shall be within the requested scope.

All other service controls need to be respected, but their use does not require any special consideration in the distributed environment.

17.4.5 Extensions

17.4.5.1 If a DSA encounters an extended abstract-operation in the name resolution phase of processing and determines that the abstract-operation should be chained to one or more other DSAs, it shall include unchanged in the chained abstract-operation any extensions present.

Note - An Administrative Authority may determine that it is appropriate to return a **ServiceError** with problem **unwillingToPerform** if it does not wish to propagate an extension.

17.4.5.2 If a DSA encounters an extension in the execution phase of processing, two possibilities may arise. If the extension is not critical, the DSA shall ignore the extension. If the extension is critical, the DSA shall return a **ServiceError** with problem **unavailableCriticalExtension**.

A critical extension to a multiple object operation may result in both results and service errors of this variety. A DSA merging such results and errors shall discard these service errors and employ the **unavailableCriticalExtension** component of **PartialOutcomeQualifier** as described in § 10.1.1 of Recommendation X.511.

17.4.6 Alias Dereferencing

Alias dereferencing is the process of creating a new target object name, by replacing the alias entry distinguished name part of the original target object name with the Aliased Object Name attribute value from the alias entry. The object name in the operation is not affected by alias dereferencing.

17.5 Authentication of distributed operations

Users of the Directory together with administrative authorities that provide directory services may, at their discretion, require that directory operations be authenticated. For any particular directory operation the nature of the authentication process will depend upon the security policy in force.

Two sets of authentication procedures are available which collectively enable a range of authentication requirements to be met. One set of procedures are those provided by Bind: these facilitate authentication between two directory application-entities for the purposes of establishing an association. The Bind procedures accommodate a range of authentication exchanges from a simple exchange of identities to strong authentication.

In addition to the peer entity authentication of an association as provided by Bind, additional procedures are defined within the directory to enable individual operations to be authenticated. Two distinct sets of directory authentication procedures are defined. One facilitates originator authentication services, which address the authentication, by a DSA, of the initiator of the original service request. The second set facilitates results authentication services which address the authentication, by an initiator, of any results that are returned.

For originator authentication two procedures are defined, one based upon a simple exchange of identities, termed identity based authentication, and one based upon digital signature techniques, termed signature based authentication. The former of these procedures is rudimentary in nature since the identity exchange is based upon the exchange of distinguished names which are transmitted in the clear.

For authentication of results a single results authentication procedure is defined, based upon digital signature techniques; due to the generally complex nature of results collation a simpler, identity-based procedure is not defined.

Authentication of error responses is not supported by these procedures.

The services described above are to be considered as augmenting those provided by the Bind service; Bind procedures are assumed to have been effected successfully prior to authentication of directory operations.

The procedures to be effected by a DSA in providing originator and results authentication are specified in § 18.9.

18 DSA behaviour

18.1 Introduction

Corresponding to each operation invoked by a requestor (e.g. DUA or DSA) the performing DSA must behave in accordance with well-defined procedures so that an appropriate response will be returned deterministically. This paragraph specifies the allowed behaviour by modelling a DSA in terms of processes implementing a particular collection of procedures. It is important to realize that a DSA need conform only to the externally visible behaviour implied by these procedures, and not to the procedures themselves.

18.2 Overview of the DSA behaviour

The behaviour of the distributed Directory as a whole is the sum of the behaviour of its cooperating DSAs. Each of these DSAs can be viewed as a process, supported internally by a set of procedures.

Figure 6/X.518 illustrates the internal view of the DSA behaviour.

The Operation Dispatcher is the main controlling procedure in a DSA. It guides each operation through the three phases of processing described in § 17.2.

The procedures which support the Operation Dispatcher are: Name Resolution, Find Naming Context, Local Name Resolution, Evaluation, Single Object Evaluation, Multiple Object Evaluation, and Result Merging. The relationships among these procedures are shown graphically in Figure 6/X.518.

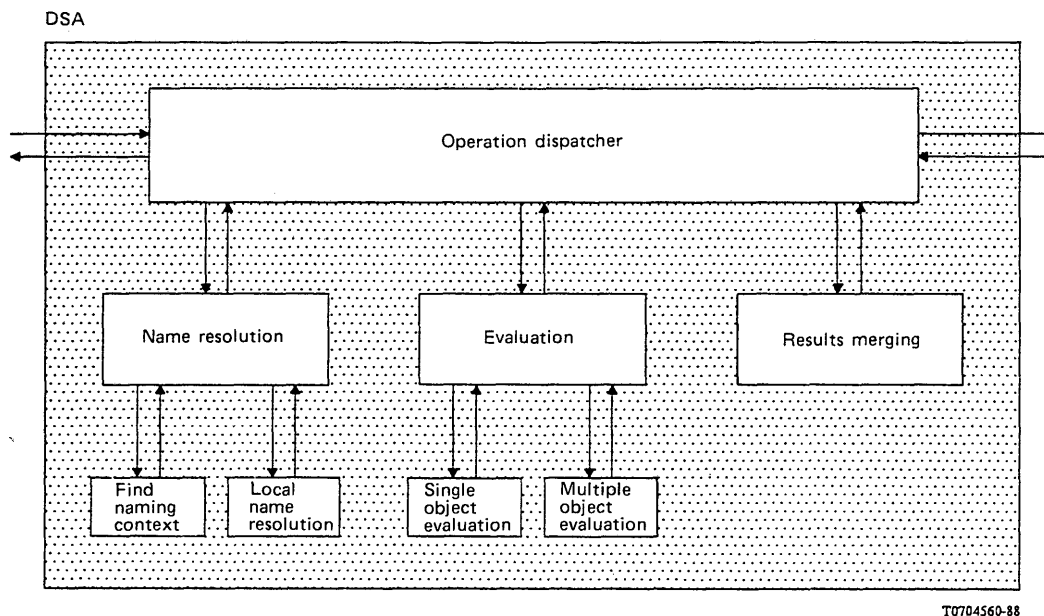


FIGURE 6/X.518

DSA Behaviour - Internal view

18.2.1 The operation dispatcher

Upon initially receiving an operation, the Operation Dispatcher validates it, checking for loop or authentication errors. If none is found, it calls Name Resolution, which returns either a Found indication, a Reference, or an error indication. References are handled by a referral or by a Chain or Multicast action, Found indications by calling the Evaluation procedure, which actually performs the intended operation. Once returned, internal or external results are collated by Results Merging, and, in the absence of errors, returned to the calling DUA or DSA.

18.2.2 Name resolution

Name Resolution calls Find Naming Context. If the returned context is local, then Local Name resolution is called, otherwise Name Resolution returns a reference or an error and terminates. If Local Name Resolution encounters an alias, it is dereferenced (if permitted) and Name Resolution repeats the analysis from the beginning. Otherwise Local Name Resolution returns a Found indication, an error or a Referral, which is passed back to the Operation Dispatcher.

18.2.3 Find naming context

Find Naming Context attempts to match the Purported Name against Context Prefixes. If none matches, then Find Naming Context attempts to identify a cross or superior reference. If a context prefix is matched, Find Naming Context returns a cross reference relating downwards in the DIT, or an indication that a suitable naming context was found locally, and sets NameResolutionPhase to "proceeding".

18.2.4 Local Name Resolution

The Local Name Resolution procedure attempts to match RDNs in the Purported Name internally until it can return a Found indication. If unable to match all RDNs internally, it attempts to identify first specific, then non-specific subordinate references, and return these to Name Resolution. If an alias is encountered, and dereferencing is allowed by the service controls, a dereferenced alias indication is returned. If dereferencing is not allowed, a Found indication is returned if and only if all RDNs had matched at the time the alias was encountered, otherwise a **nameError** is returned.

18.2.5 *Evaluation*

The Evaluation procedure actually executes the requested Directory operation against the target object. Depending on the type of operation, Single Object Evaluation or Multiple Object Evaluation is invoked.

18.2.6 *Single object evaluation*

Single object evaluation is invoked for **Read**, **Compare**, **AddEntry**, **RemoveEntry**, **ModifyEntry**, and **ModifyRDN**. It is in this procedure that attributes are actually retrieved, checked, or changed.

18.2.7 *Multiple object evaluation*

The Multiple Object Evaluation procedure is invoked for the **Search** and **List** operations to check filters, retrieve results, and if necessary, dispatch sub-requests.

18.2.8 *Result merging*

The Results Merging procedure collates results or errors received from other DSAs with locally retrieved results.

18.3 *Specific operations*

The operations fall into three categories of operations (in each case the operation and its Chained counterpart are both in the same category).

- a) Single-Object Operations: **Read**, **Compare**, **AddEntry**, **ModifyEntry**, **ModifyRDN**, **RemoveEntry**.
- b) Multiple-Object Operations: **List**, **Search**.
- c) Abandon Operation, i.e. **Abandon**.

The handling of these categories are described in §§ 18.3.1 to 18.3.3 respectively. Since there is considerable similarity between the way that a DSA behaves in performing an operation of a service-port and in performing its counterpart chained operation of a chained service-port, there is a single description applying to both, with exceptions to this rule being noted.

18.3.1 *Single-object operations*

Single-object operations are those which affect a single entry, and which therefore can be carried out entirely within the DSA which contains the entry on which the operation is to be performed. Such operations can be commonly described by the following sequence of events:

- 1) Activate the Operation Dispatcher.
- 2) Perform Name Resolution to locate the object whose name was specified as the argument of the operation.
- 3) Perform the single-object evaluation procedure.
- 4) Service controls, such as time limit, should be checked during the course of the operation to enforce the constraints specified by the user.
- 5) Return the results to the DUA or DSA which forwarded the request.

18.3.2 *Multiple-object operations*

Multiple-object operations are those which affect several entries which may or may not be co-located in the same DSA. Such operations may thus entail a cooperative effort by several DSAs to locate and operate on all the entries affected by the requested operation. The common behaviour of such operations can be summarized as follows:

- 1) Activate the Operation Dispatcher.
- 2) Perform the Name Resolution procedures to locate the object whose name was specified as the argument of operation.
- 3) Once the target object of the operation has been located, perform the multiple-object evaluation procedures.

- 4) If request decomposition has taken place in one of the multiple-object evaluation procedures and sub-requests have been chained/multicast, the Operation Dispatcher maintains the current local results, waits for chained responses, and activates Results Merging.
- 5) Service Controls such as time limit, size limit should be checked during the course of the operation to remain within the constraints specified in the common argument.
- 6) Return the results or errors to the DUA or DSA which forwarded the request.

18.3.3 *Abandon operation*

On receipt of an abandon operation, a DSA determines whether it can abandon the specified operation, and, if so, abandons it and returns a result (the operation that was abandoned returns an **Abandoned** error). If it cannot abandon the specified operation, it returns an **AbandonFailed** error.

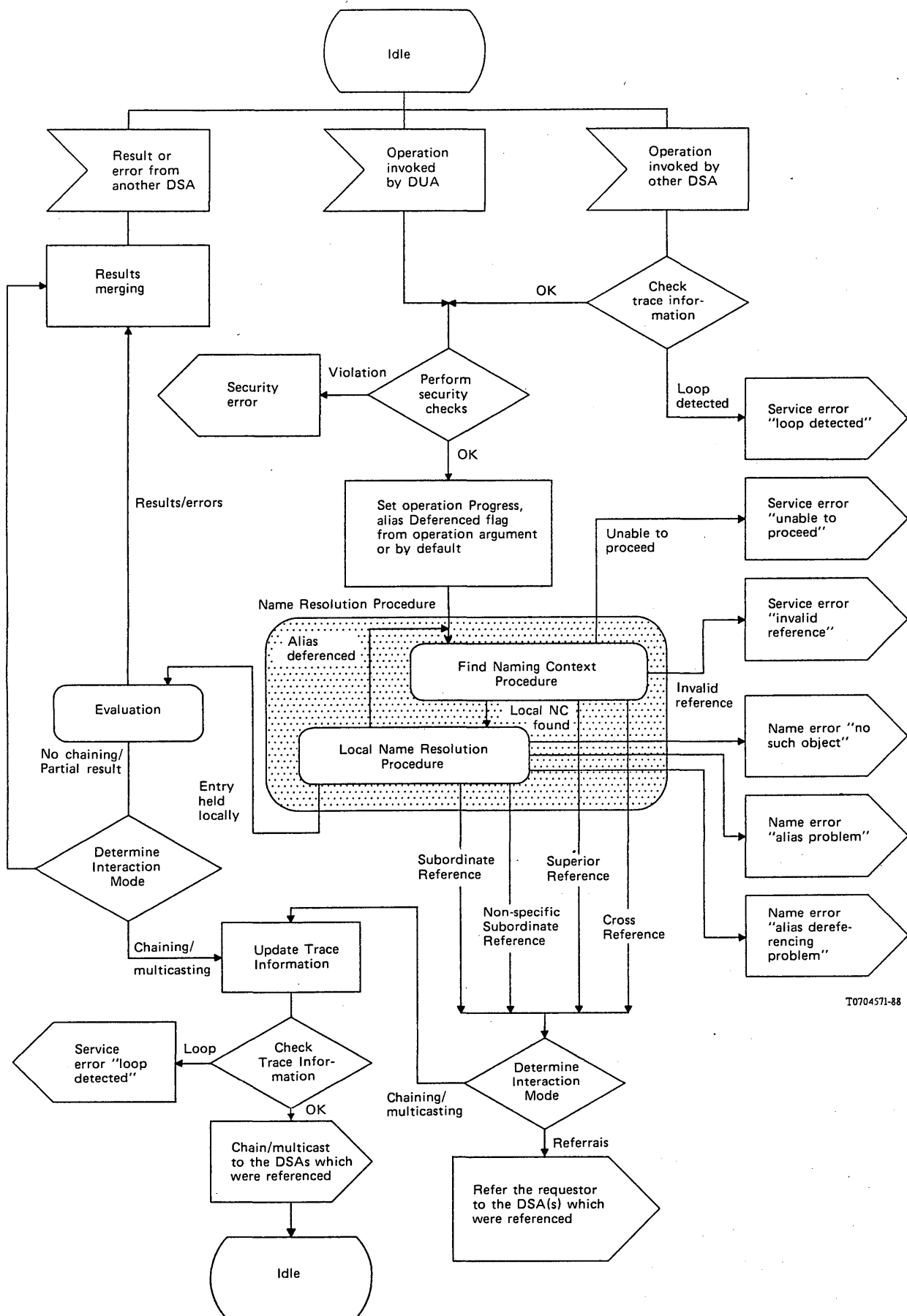
The following specifies the procedure specific to the **Abandon** operation.

- 1) Locate the operation whose invoke identifier is specified as the argument of the **Abandon** operation.
- 2) Optionally compose request(s) with the proper invoke-id to abandon any outstanding chained/multicast operations to other DSAs.
- 3) Optionally, the abandon operation is performed locally as defined in Recommendation X.511.
- 4) Return result or error to the DUA or DSA which forwarded the request.

18.4 *Operation dispatcher*

18.4.1 *Introduction*

The Operation Dispatcher utilizes the Name Resolution described in § 18.6 of this Recommendation and all the interactions (i.e. DSA to DSA or DUA to DSA) necessary to locate target entries in a distributed directory environment. Figure 7/X.518 shows a detailed diagram describing the Operation Dispatcher. The algorithm is summarized below.



T0704571-88

FIGURE 7/X.518

Operation Dispatcher

18.4.2 *Implicit actions*

18.4.2.1 *Security*

It should be noted that although the checking of signatures is not explicitly included in this algorithm, this action is always the first step when a signed operation, result or error arrives to the DSA.

Note - This does not include embedded signatures.

Should the signature be invalid, or absent in a case when it should be present, a **SecurityError** is returned. All processing of the operation is terminated and the operation dispatcher goes to its idle state.

The signing of an operation result if required is likewise an implicit last step before sending it off.

18.4.2.2 *ServiceControls*

Although the **ServiceControls** are not explicitly mentioned, they are respected. For example, the checking of the **timeLimit** of an arriving operation and the checking of **sizeLimit** before sending a result are regarded as mandatory. These are discussed in § 17.4.4.

18.4.2.3 *TraceInformation*

TraceInformation is always updated with the state it arrived to the DSA in, before including it in the **ChainingArguments**. That is not explicitly stated in the text below.

18.4.3 *Arguments*

Chaining arguments for the particular operation.

18.4.4 *Results*

Chaining results for the particular operation.

18.4.5 *Errors*

Any error defined in this Recommendation.

18.4.6 *Algorithm*

1) Receive operation.

If the operation originates from another DSA it will comprise the chaining arguments, including: **operationProgress**, **aliasDereferenced**, **aliasedRDNs**, **targetObject Name** and **traceInformation** as well as the parameters contained in the original operation.

If the operation originates from a DUA it will not contain the **aliasDereferenced** indication: thus adopt the value of **FALSE**. The argument also does not include any **TraceInformation**, so no loop checking needs to be performed. Set **targetObject Name** to the name of the target object for the operation (see § 17.2). Other chaining arguments are set according to the parameters in the DAP operation. **Originator** is set to the name of the user.

2) If the operation came from a DSA, check the trace information for loops (activate Loop Detection). If a loop is detected, return **ServiceError** with a problem of **loopDetected** and terminate the processing.

3) Perform security checks to the operation (originating either from a DUA or a DSA). If there is a violation, a **SecurityError** is returned. Otherwise, set **operationProgress** and **aliasDereferenced** according to the operation argument or by default.

4) Perform the Name Resolution Procedure.

The Name Resolution Procedure will return a found indication, a remote reference, or an error indication.

5) One of the following errors may be raised:

ServiceError (UnableToProceed) - if a DSA determines that it was forwarded an operation pertaining to information which it does not hold.

ServiceError (invalidReference) - if a DSA determines that an invalid knowledge reference was used.

NameError (noSuchObject) - if the purported name specified in the operation request is determined to be invalid.

NameError (aliasProblem) - if an alias has been dereferenced which names no object.

Name Error (aliasDereferencingProblem) - if an alias was encountered in a situation where it is not allowed.

On receipt of any one of these errors, the Operation Dispatcher terminates and an error is returned to the DSA or DUA which originated the distributed operation.

- 6) If Found is returned, activate the Evaluation Procedure.
- 7) If a remote reference is returned (whether from Name Resolution or Evaluation) it may be any one of the following: a cross reference, a subordinate reference, a superior reference or a non-specific subordinate reference.

If any such reference is returned it signifies that the Name Resolution or Evaluation cannot be completed in this DSA, but must involve the DSA identified in the reference.

The Operation Dispatcher then checks for referral or chaining mode.

- 8) If the referral mode or interaction has been selected, then, subject to **scopeOfReferral**, either the information contained in the returned reference will be returned to the originating DUA or DSA as a referral, or **outOfScope ServiceError** will be returned. The processing of this operation will then terminate.

Note - If **returnCrossRefs** is true and reference is not a non-specific subordinate reference or superior reference and, in addition, the administrative authority is willing to provide knowledge, then the context prefix in the referral can be set.

- 9) If the chaining mode of interaction has been selected, the operation is forwarded to the DSA specified in the reference. In the case of a non-specific subordinate reference, the operation must be forwarded to each DSA whose name was attained as part of a non-specific subordinate reference. Such forwarding may be accomplished either by multicasting or by sequentially chaining the operation.
- 10) Perform Loop Avoidance for each operation to be sent. If the avoidance turns out to be not applicable or no loop is detected, assign values to the chaining arguments, including an updated version of **traceInformation**, and send the operations.

If no operations were sent (because of looping problems), return a **serviceError** (with problem of **loopDetected**) and terminate the processing of this operation.

Note - If the decomposed operation was aborted because of loop avoidance in this step it is a local matter whether to return a partial result or to abort the whole operation and return an error. If the latter is chosen then return **ServiceError** (with problem **loopDetected**) and terminate processing.

- 11) Wait for the responses then perform the Results Merging procedure.

18.5 Looping

Within the context of a particular directory operation a loop occurs if at any time the operation returns to a previous state (as defined in § 17.4.2). This does not mean that an operation cannot be processed multiple times by a particular DSA. However, it does mean that the DSA will not process the same operation in the same state multiple times.

Looping is managed using the **traceInformation** argument as defined in § 12.6. Two strategies are defined to determine loops: loop detection and loop avoidance, described in §§ 18.5.1 and 18.5.2 respectively.

18.5.1 *Loop detection*

Loop detection requires that a DSA, when receiving an incoming operation, determines whether the current state of the operation appears in the sequence of previous states recorded in the **traceInformation** argument for that operation. If it does, the operation is looping and a **ServiceError** (with problem of **loopDetected**) is returned. Otherwise the DSA continues processing the operation according to the procedures specified in § 18.4.

18.5.2 *Loop avoidance*

Loop avoidance requires that a DSA, immediately prior to forwarding an operation to another DSA (as part of a chaining, multicasting, or request decomposition procedure), determines whether the consequential state of the operation (if known) appears on the sequence of previous states recorded in the trace-information argument for the original incoming operation. The consequential state is the value of **TraceItem** which will be added to **TraceInformation** by the receiving DSA.

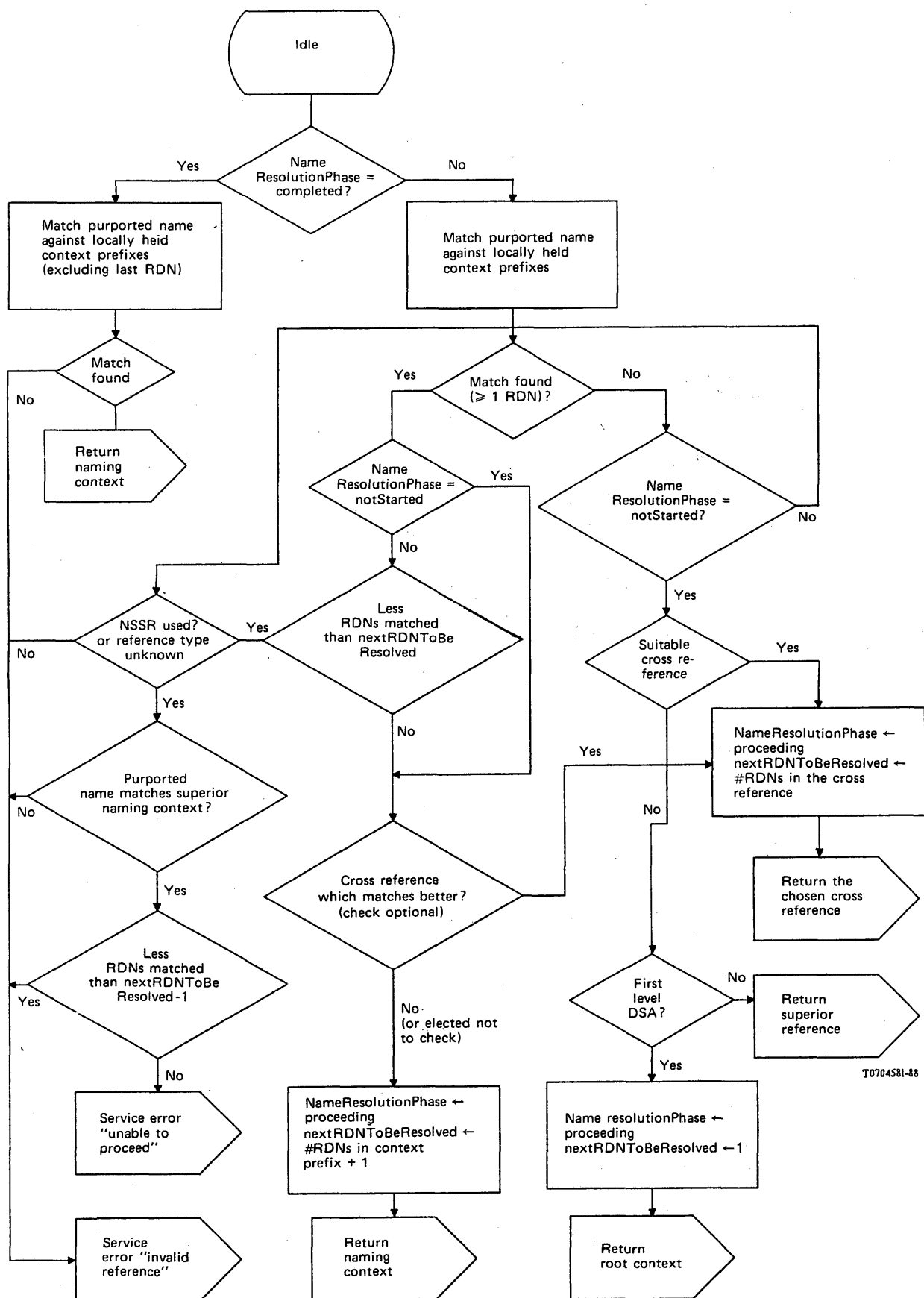
In the event that the original incoming operation was to a service-port (rather than a chained-service-port) there will be no trace information and the loop avoidance procedure will not be relevant.

If the consequential state of the operation is known and does appear within the **traceInformation**, the operation, if invoked, would cause a loop. Under this circumstance the appropriate response to the original operation is a **ServiceError** (with problem of **loopDetected**).

18.6 *Name resolution procedure*

This paragraph describes in detail the Name Resolution procedure, its input and output parameters, and its possible error conditions. Figure 7/X.518 shows the overall procedure in the form of a diagram. The Name Resolution procedure calls two component procedures:

- 1) Find Naming Context (Figure 8/X.518).



T0704581-88

FIGURE 8/X.518

Find Naming Context

2) Local Name Resolution (Figure 9/X.518).

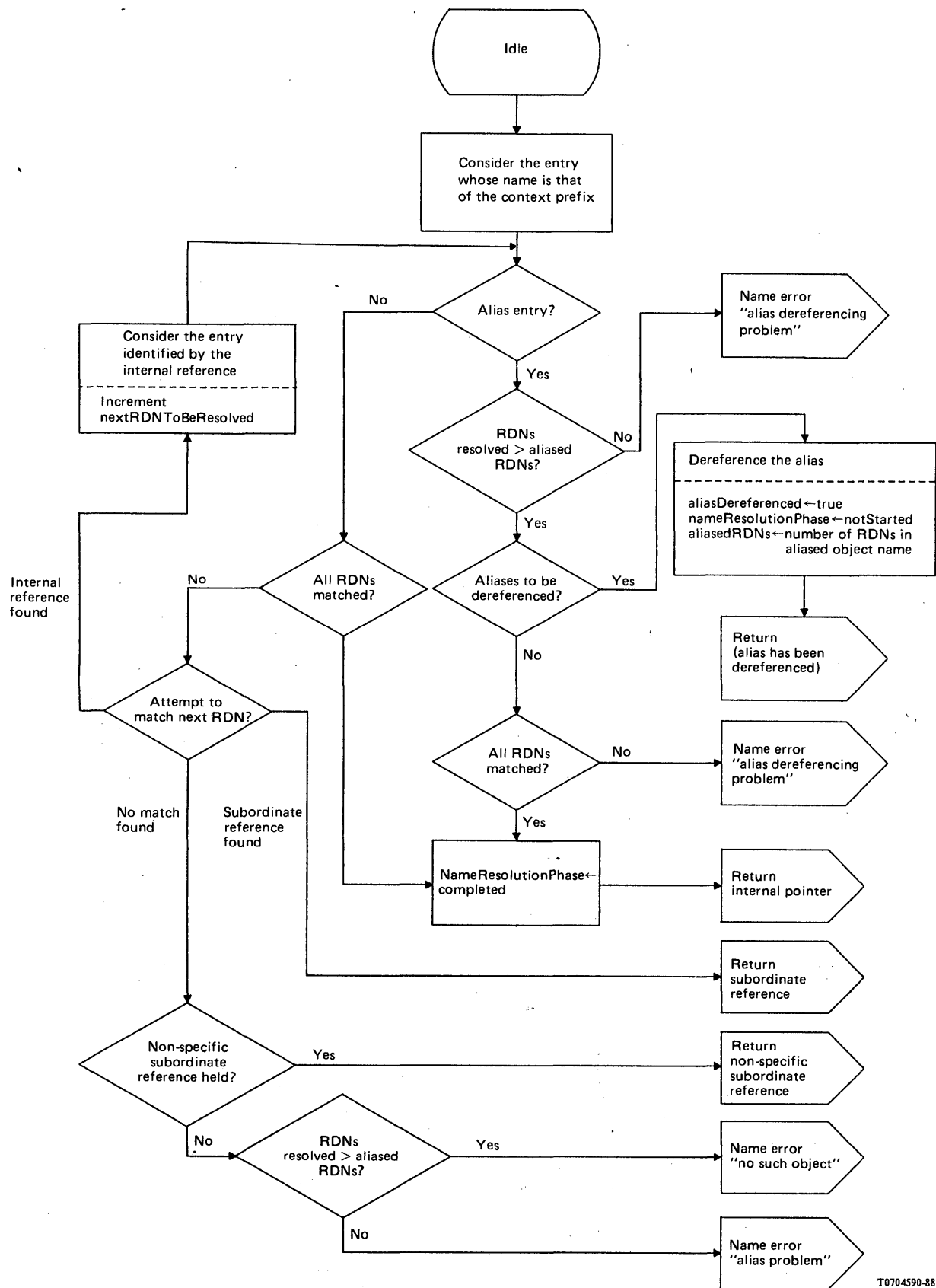


FIGURE 9/X.518

Local Name Resolution

The Name Resolution procedure conveys back to the Operation Dispatcher the results of the above mentioned component procedures, except in the following two cases. The first one is when the Find Naming Context procedure identifies a suitable context which has to be further examined, and returns the local naming context. The second case is when the Local Name Resolution procedure indicates that it has dereferenced an alias. In the former case, the Name Resolution procedure calls the Local Name Resolution procedure. In the latter case, the Name Resolution procedure is reactivated with the new target object name.

18.6.1 *Arguments*

The procedure makes use of the following arguments:

- the target object name (the purported name);
- operation progress;
- the value of the **dontDereferenceAliases** service control;
- the value of the **aliasedRDNs** parameter;
- the value of the **aliasDereferenced** parameter.

18.6.2 *Results*

There are two cases of successful outcome.

The first of these returns:

- a reference;
- operation progress (updated appropriately);
- **aliasDereferenced** indication and, optionally, **aliasedRDNs**.

The second of these returns:

- an indication that the naming context was found (together with the local pointer to the entry);
- operation progress (updated appropriately);
- **aliasDereferenced** indication and, optionally, **aliasedRDNs**.

18.6.3 *Errors*

One of the following errors may be returned:

- **ServiceError (unableToProceed)**;
- **ServiceError (invalidReference)**;
- **NameError (aliasProblem, noSuchObject or aliasDereferencingProblem)**.

18.6.4 *Procedure*

- 1) Activate the Find Naming Context procedure.
- 2) Wait for response from Find Naming Context procedure.
- 3) Receive returned results or error, i.e. Local Naming Context Found, Remote Reference, Unable to Proceed Error, Name Error, or invalidReference.
- 4) Perform functions based on returned results or error.
 - a) If the local naming context has been found, activate the Local Name Resolution procedure. This procedure may return an Internal Reference Found, a Remote Reference, an Alias Dereference, or a NameError. Each of these causes the Name Resolution to be terminated with the outcome reported, except that if an alias has been dereferenced, the procedure is restarted at step 1).
 - b) Any other outcome is passed back to the Operation Dispatcher.

18.6.5 Find naming context procedure

18.6.5.1 Introduction

Figure 8/X.518 shows this procedure in the form of a diagram. Below is a textual description. In this it is assumed that the current value of Operation Progress is always returned upon exit of the procedure.

18.6.5.2 Arguments

The procedure makes use of the following arguments:

- the target object name (the purported name);
- operation progress.

18.6.5.3 Results

There are two cases of successful outcome.

The first of these returns:

- a reference;
- operation progress (updated appropriately).

The second of these returns:

- an indication that a suitable naming context was found locally;
- operation progress (updated appropriately).

18.6.5.4 Errors

One of the following errors may be returned:

- **ServiceError (unableToProceed)**;
- **ServiceError (invalidReference)**.

18.6.5.5 Procedure

- 1) If **nameResolutionPhase** is set to **completed** on entry, attempt to match the purported name against the context prefixes of the superior naming contexts of all the locally held naming contexts. If a match is found, return all the appropriate locally held naming contexts. If no match is found, return an **invalidReference ServiceError**.
- 2) If **nameResolutionPhase** is not set to **completed**, attempt to match context prefixes against a sequence of one or more RDNs in the initial portion of the purported name. For a match to be found, all RDNs in a context prefix must be matched. The context prefixes used are those of Naming Contexts for which this DSA has administrative authority. In case of multiple matches the one with the maximum number of matched RDNs is chosen.

If a match is found, execute (3).

If a match is not found, execute (5).

- 3) If **nameResolutionPhase** is **notStarted**, execute (4). If the number of RDNs in the initial portion of the purported name, matched as described in (2) above, is greater or equal to the **nextRDNTToBeResolved** component of **OperationProgress**, then execute (4), otherwise execute (9).
- 4) The **nextRDNTToBeResolved** is set to the number of matched RDNs plus 1 and the **nameResolutionPhase** is set to **Proceeding**. The context is returned and this procedure terminated.

As a performance enhancement, the DSA may optionally match the purported name against the cross references held by the DSA. If more RDNs are matched against a cross reference than against the locally held context prefixes, then execute step (7).

Note - The Name Resolution procedure will in case of this outcome call the Local Name Resolution.

- 5) If no match was found, the value of the **nameResolutionPhase** is checked. If the **nameResolutionPhase** is **notStarted**, execute (6).

If the **nameResolutionPhase** is **proceeding** or **completed**, then execute (9).

- 6) Using Cross Reference context prefixes, attempt to match against a sequence of one or more RDNs in the initial portion of the purported name. In case of multiple matches, the one with the maximum number of matched RDNs is chosen.
- 7) If a match was found to a cross reference, set the **nextRDNTToBeResolved** to the number of RDNs in the chosen cross reference. The cross reference is returned and this procedure is terminated.
- 8) If no match was found to a cross reference, determine if the DSA is a first level DSA. If not, it will have a superior reference. Return this and terminate the procedure.

If the DSA is a first level DSA, set **nextRDNTToBeResolved** to one, and **nameResolutionPhase** to **proceeding**. Return the root naming context and terminate the procedure.

- 9) Check the value of the **referenceType** component of the **ChainingArgument**. If a non-specific subordinate reference was used, or the request came from a DUA, execute (10); otherwise, return **ServiceError** with **invalidReference** problem and terminate the procedure.
- 10) Compare the initial portion of the purported name to the context prefixes (minus their last RDN) of the locally held naming contexts. This effectively is a comparison to some of the naming contexts of the immediate superior to this DSA.

If there is no match, return **ServiceError** with **invalidReference** problem and terminate the procedure.

If a match is found, and the number of RDNs matched is less than in **nextRDNTToBeResolved** - 1, return **ServiceError** with **invalidReference** problem; otherwise, return **ServiceError** **unableToProceed** problem. Terminate the procedure.

18.6.6 Local Name Resolution

18.6.6.1 Introduction

The Local Name Resolution matches RDNs in the purported name against internal knowledge references. It returns Found, Remote Reference, Alias Dereferenced, or Error indication.

Figure 9/X.518 shows this procedure in the form of a diagram. Below is a textual description.

18.6.6.2 Arguments

The procedure makes use of the following arguments:

- internal reference to naming context (with pointer to the entry whose name is the same as the context prefix);
- the target object name (the purported name);
- operation progress;
- the value of the **dontDereferenceAliases** service control;
- the value of the **aliasedRDNs** parameter;
- the value of the **aliasDereferenced** parameter.

18.6.6.3 Results

There are three cases of successful outcome.

The first of these returns:

- a reference;

- operation progress (updated appropriately).

The second of these returns:

- an indication that the entry was found locally;
- operation progress (updated appropriately).

The third of these returns:

- an indication that an alias was dereferenced;
- operation progress (set back to "not started").

18.6.6.4 *Errors*

One of the following errors may be returned:

- name error.

18.6.6.5 *Procedure*

The naming context returned by FindNaming Context will point to the entry of the root of the subtree. In the case of the root context, the entry is only a null entry.

- 1) If the internal reference is for an alias entry, execute step (7), otherwise step (2).
- 2) If all the RDNs in the purported name have been matched, then the target entry has been found. Set **nameResolutionPhase** to **completed**. An internal pointer is returned and the procedure terminated.

Otherwise step (3) should be executed.

Note - The matching could be attained with the context prefix on its own, or with the context prefix plus successive RDNs contained in internal references in the knowledge tree.

- 3) If an internal reference entry is found subordinate to the current entry in the knowledge tree which matches the next RDN in the purported name, then increment the **nextRDNTToBeResolved**, set current entry to subordinate entry, and execute step (1) of this procedure again.
- 4) If the current entry has a subordinate reference whose RDN matches the next one in the purported name, return it and terminate the procedure.
- 5) If there are any non-specific subordinate references, subordinate to the current entry in the knowledge tree, return them as references and terminate the procedure.
- 6) If an internal reference, subordinate reference, or non-specific subordinate reference is not found, then check the number of RDNs in the purported name that have been matched. If more RDNs have been matched than in the **aliasedRDNs** component of **ChainingArgument**, then return **NameError** with **noSuchObject** problem. If less RDNs have been matched, then return **NameError** with **aliasProblem**.
- 7) If the number of RDNs in the purported name that have been matched is less than or equal to the **aliasedRDNs** component of **ChainingArgument** (if any), then the previous alias that was dereferenced (if any) points to another alias. If so, return **NameError** with **aliasDereferencingProblem**.
- 8) If the **aliasedRDNs** component is missing, or if the number of RDNs matched is greater than **aliasedRDNs** component of **ChainingArgument**, then check the **dontDereferenceAlias** service control. If aliases can be dereferenced, then execute step (9), otherwise step (10).
- 9) Dereference the alias. Set **nameResolutionPhase** of **OperationProgress** to **notStarted**. Set **aliasDereferenced** component of **ChainingArgument** to **TRUE**, and **aliasedRDNs** to the number of RDNs in the **aliasedObjectName** attribute of the alias entry. Set **targetObject** to the new name. Terminate the procedure. (The process of Name Resolution will be restarted.)

- 10) If all the RDNs in the purported name have been matched, execute step (2). Otherwise, return **NameError** with **aliasDereferencingProblem**.

18.7 Object evaluation procedures

The object evaluation procedures specified comprise two categories of procedures:

- a) single-object evaluation procedure;
- b) multiple-object evaluation procedures.

Figure 10/X.518 shows the object evaluation procedure.

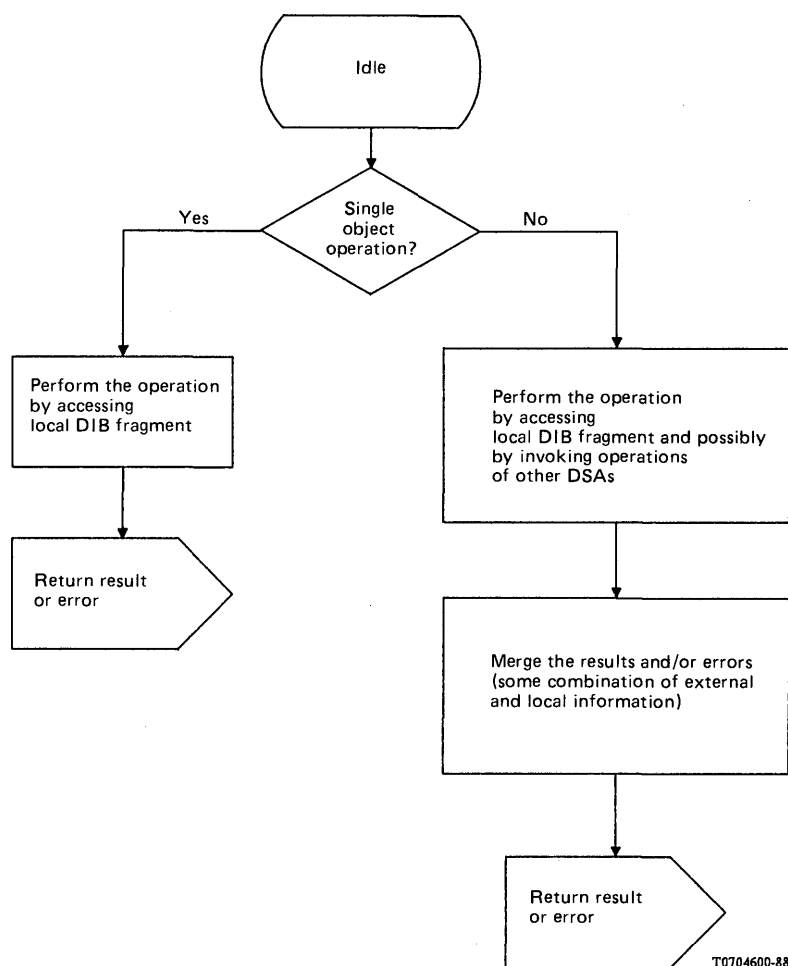


FIGURE 10/X.518

Evaluation and result merging

18.7.1 Single-object evaluation procedures

Single-object evaluation procedures, which are common to the class of operations concerned with accessing a single object are carried out directly, with the result or error being returned to the invoker.

These operations comprise **Read**, **Compare**, **AddEntry**, **RemoveEntry**, **ModifyEntry** and **ModifyRDN**, and their Chained counterparts.

The action required on the entry is as described in the appropriate paragraph of Recommendation X.511.

AddEntry, **RemoveEntry**, and **ModifyRDN** operations affect knowledge. If the immediate superior of the entry is in a different DSA, correct external knowledge references shall be maintained. How this is done is outside the scope of this Recommendation.

How the DSA is chosen to contain the entry created by **AddEntry** is outside the scope of this Recommendation.

If the immediate superior of an entry to be created by **AddEntry** or modified by **ModifyRDN** has non-specific subordinate references, procedures outside the scope of this Recommendation shall be followed to ensure that no two entries have the same distinguished name.

Requests which cannot be satisfied under these conditions shall fail with an **UpdateError** with problem **affectsMultipleDSAs**.

18.7.2 *Multiple-object evaluation procedures*

Multiple-object evaluation procedures, which are common to the class of operations concerned with accessing multiple objects, are specified in the following subparagraphs.

These operations comprise **List** and **Search**, and their Chained counterparts.

18.7.2.1 *List*

This paragraph specifies the evaluation procedure specific to **List** and **ChainedList**. (In what follows the term "List" applies to both.)

18.7.2.1.1 *List procedure (I)*

This procedure applies where the **List** request has **nameResolutionPhase** component of **OperationProgress** set to **notStarted** or **proceeding** and where the DSA, after performing Name Resolution finds that it holds the base object.

The base object will be denoted by "e".

- 1) Get each locally held immediate subordinate of e to form a local set of results. Set **aliasEntry** and **fromEntry** in **ListResult** as appropriate.
- 2) Get the set of non-specific subordinate references and subordinate references to DSAs which hold immediate subordinates of "e".
- 3) Pass the subrequest with base object = e, and **OperationProgress** set to **completed** to the Operation Dispatcher which subsequently forwards it to each DSA which holds immediate subordinates of e.

Note - If the DSA holds subordinate references with an indication of whether or not the subordinate entry are aliases, and the **dontUseCopy** is **FALSE**, then this step can be omitted for those entries. The information about the subordinates is available directly.

18.7.2.1.2 *List Procedure (II)*

This procedure applies to a **List** request with the **nameResolutionPhase** component of **OperationProgress** set to **completed**.

The base object will be denoted by "e".

- 1) Get each locally held immediate subordinate of e to form a local set of results. Set **aliasEntry** and **fromEntry** in **ListResult** as appropriate.
- 2) Pass the results to the Operation Dispatcher which forwards them to the requesting DUA or DSA.

18.7.2.2 *Search*

This paragraph specifies the evaluation procedure specific to **Search** and **ChainedSearch**. (In what follows the term "Search" applies to both.)

Note that two circumstances exist, requiring two separate procedures. The first procedure (§ 18.7.2.2.1) applies when the DSA executing the Search contains the **targetObject** as a local entry. The second procedure (§ 18.7.2.2.2) applies when the DSA executing the Search does not hold the **targetObject**, but only subordinates of the **targetObject**.

18.7.2.2.1 *Search procedure (I)*

This procedure applies to a **Search** request with the **nameResolutionPhase** component of **OperationProgress** set to **notStarted** or **proceeding** and where the DSA, after performing Name Resolution, determines that it holds the target object.

The base object will be denoted by "e".

- 1) If the **subset** argument is **baseObject** or **wholeSubtree**, then apply the filter argument specified in the Search to the entry e, to form a set of local results. Return the results for Results Merging. If the **subset** argument is **baseObject**, terminate the procedure, otherwise continue at (2).

- 2) If the **subset** argument is **oneLevel** or **wholeSubtree** form a set E from the locally-held immediate subordinates of e, except that:

If aliases are to be dereferenced, i.e. the **searchAliases** parameter is **TRUE**, then any alias entries that are found are handled in paragraph 5) below and do not contribute to these results.

Apply the filter arguments to E to give a filtered subset E'gE; return this set E' of local results for Results Merging.

- 3) Other subordinates of e may reside in other DSAs, and if so will be referenced as subordinate or non-specific subordinate references. For each DSA which is so referenced, prepare a new Search with **targetObject** = e, and with **nameResolutionPhase** of **OperationProgress** set to **completed**. Return each Search subrequest to the Operation Dispatcher for forwarding. If any error result is returned from a subrequest, it is ignored, as if no subrequest had been sent.
- 4) If the **subset** argument is **oneLevel**, the Search is now complete so terminate the procedure.

If the **subset** argument is **wholeSubtree**, then:

if the set E from paragraph (2) is empty, then the whole subtree held in this DSA has been searched, so terminate the procedure;

otherwise continue processing as follows:

let each entry that was in set E be denoted by e. Repeat the Search procedure from paragraph (2), for each entry e.

- 5) If aliases are to be dereferenced, any alias entries found in step (2) are placed in set D. For each entry, d in D, dereference the alias, and formulate a new Search with **nameResolutionPhase** set to **notStarted**, and **targetObject** created from the **aliasedObjectName** attribute and the old **targetObject** name.

If the **subset** argument was **oneLevel**, set it to **baseObject** in the new subrequest, otherwise set it to **wholeSubtree**.

If any error result is returned from the subrequest, it is ignored, as if no subrequest had been made.

18.7.2.2.2 Search Procedure (II)

This procedure applies to a Search request with the **nameResolutionPhase** component of **OperationProgress** set to **completed**.

The target object will be denoted by "e".

For each locally held immediate subordinate e' of e, formulate a new request with **targetObject** = e'. If the **subset** argument was **oneLevel**, set it to **baseObject**, otherwise leave it as **wholeSubtree**. Now carry out the procedure defined in steps (1) to (5) in § 18.7.2.2.1. If there are no such subordinates, return **unableToProceed ServiceError**.

18.8 Result merging procedure

This procedure is called when external results and/or errors are present. There might also be one internal result. All results and errors are assumed to be held within the DSA until the procedure completes.

The external information could be due to chaining, multicasting or request decomposition.

In the case of chaining there will be a single result or error. In the case of multicasting there might be either no result, one result or several identical results. In addition, there may be some errors. If there is more than one result, all but one of them are arbitrarily discarded. A result is always returned in preference to an error. If there are no results, an error is returned, with the following exceptions:

- i) If **invalidReference** was returned, the reference is marked as such, and the DSA may either use an appropriate alternate external reference to continue the request, or return **ditError** to the requestor. (The handling of invalid external references is beyond the scope of this Recommendation.)
- ii) In the case of multicasting, **unableToProceed** errors should be ignored, unless all responses are of this type in which case **NameError noSuchObject** should be returned to the responder. If at least one result is returned, then all errors can be ignored.
- iii) In the case of referrals, these need not be treated as errors, and may be acted upon.

If the merging is required due to a request decomposition, the merging amounts to forming the union of the results.

In the case of decomposition, when there are both results and errors to be merged, an incomplete result is returned to the requestor.

A DSA might at this stage choose to extract referrals from the incoming results and errors that should be merged. It might then decide to explore all or some of these further, in which case operations are chained. The old result will have to be saved and later merged with the results or errors produced by the chaining.

The handling of signatures which may be present with the results being returned is specified in § 18.9.2 below.

18.9 Procedures for distributed authentication

This paragraph specifies the procedures necessary to support the directory distributed authentication services. These services, and hence the procedures, are categorized as:

- originator authentication, which is supported in either an unprotected (simple identity based) or secure (based upon digital signatures) form; and
- results authentication which is similarly protected (again based upon digital signatures).

18.9.1 Originator authentication

18.9.1.1 Identity based authentication

The identity based authentication service enables DSAs to authenticate the original requestor of information for the purpose of effecting local access controls. DSAs wishing to exploit this service must adopt the following procedure:

- for a DSA requiring to authenticate a DAP request, the DSA acquires the distinguished name of the requestor through the Bind procedures at the time a DUA association (DUA or DSA) is established. Successful conclusion of these procedures does not in any way prejudice the level of authentication that may subsequently be required for processing operations using that association;
- the DSA with which the DUA association exists must insert the requestor's distinguished name in the initiator field of the ChainingArgument for all subsequent chained operations to other DSAs;
- a DSA, on receiving a chained-operation, may satisfy that operation, or not, depending upon the determination of access rights (a locally defined mechanism). If the outcome is not satisfactory a **SecurityError** may be returned with **SecurityProblem** set to **insufficientAccessRights**.

18.9.1.2 Signature-based originator authentication

This signature-based originator authentication service enables a DSA to authenticate (in a secure manner) the originator of a particular service request. The procedures to be effected by a DSA in realizing this service are described in this paragraph.

The signature-based authentication service is invoked by a DUA using the **SIGNED** variant of an optionally-signed service request.

A DSA, on receiving a signed request from another DSA, shall remove that DSA's signature prior to processing the operation. Assuming the result of any signature verification proves to be satisfactory, the DSA will continue to progress the operation. If, during processing, the DSA requires to perform chaining, multicasting or request decomposition, the argument set for each associated chained operation shall be constructed as follows:

- the DSA forms an argument set which may be optionally signed; the argument set comprises the incoming signed argument set together with a modified **ChainingArgument**.

In the event that the DSA is able to contribute information to the response, originator authentication, based upon the signed service request, may be used for the determination of access rights to that information.

If a DSA receives an unsigned service request for information which will only be released subject to originator authentication, a **SecurityError** will be returned with **SecurityProblem** set to **protectionRequired**.

18.9.2 Results authentication

This service is provided to enable requestors of directory operations (either DUA or DSAs) to verify (in a secure manner using digital signature techniques) the source of results. The results authentication service may be requested irrespective of whether originator authentication is to be used.

The results authentication service is initiated using the **signed** value of the **protectionRequest** component as contained within the argument set of directory operations; a DSA receiving an operation with this option selected may then optionally sign any subsequent results. The **signed** option in the **protectionRequest** serves as an indication, to the DSA, of the requestor's preference; the DSA may, or may not, actually sign any subsequent results.

In the case where a DSA performs chaining, multicasting or request decomposition of such a request, the DSA has a number of options in terms of the form of results sent back to the requestor, namely:

- a) return a composite response (signed or unsigned) to the requestor;
- b) return a set of two or more uncollated partial responses (signed or unsigned) to the requestor; within this set zero or more members may be signed and zero or one unsigned. In the event that an unsigned partial result is present, this member may in fact be a collation of one or more unsigned partial responses which have been received from other DSAs, contributed by this DSA, or both.

ANNEX A

(to Recommendation X.518)

ASN.1 for distributed operations

This Annex is part of the Recommendation.

This Annex includes all of the ASN.1 type, value and macro definitions contained in this Recommendation in the form of the ASN.1 module **Distributed Operations**.

```
DistributedOperations {joint-iso-ccitt ds(5) modules(1) distributedOperations(3)}  
DEFINITIONS ::=   
BEGIN
```

EXPORTS

DirectoryRefinement, chainedReadPort, chainedSearchPort, chainedModifyPort,
DSABind, DSABindArgument,
DSAUnbind,
ChainedRead, ChainedCompare, ChainedAbandon,
ChainedList, ChainedSearch,
ChainedAddEntry, ChainedRemoveEntry,
ChainedModifyEntry, ChainedModifyRDN,
DsaReferral, ContinuationReference;

IMPORTS

InformationFramework, abstractService, distributedOperations,
directoryObjectIdentifiers, selectedAttributeTypes
FROM UsefulDefinitions {joint-iso-ccitt ds(5) modules(1) usefulDefinitions(0)}

DistinguishedName, Name, RelativeDistinguishedName
FROM InformationFramework informationFramework

id-ot-dsa, id-pt-chained-read, id-pt-chained-search, id-pt-chained-modify
FROM DistributedDirectoryObjectIdentifiers, distributedDirectoryObjectIdentifiers

PresentationAddress
FROM SelectedAttributeTypes selectedAttributeTypes

directory, readPort, searchPort, modifyPort
DirectoryBind,
ReadArgument, ReadResult,
CompareArgument, CompareResult,
Abandon
ListArgument, ListResult,
SearchArgument, SearchResult,
AddEntryArgument, AddEntryResult,
RemoveEntryArgument, RemoveEntryResult,
ModifyEntryArgument, ModifyEntryResult,
ModifyRDNArgument, ModifyRDNResult,
Abandoned, AttributeError, NameError, ServiceError, SecurityError, UpdateError
OPTIONALLY-SIGNED, SecurityParameters
FROM DirectoryAbstractService directoryAbstractService

-- objects and ports --

DirectoryRefinement ::= REFINE directory AS

dsa RECURRING
readPort [S] VISIBLE
searchPort [S] VISIBLE
modifyPort [S] VISIBLE
chainedReadPort PAIRED WITH dsa
chainedSearchPort PAIRED WITH dsa
chainedModifyPort PAIRED WITH dsa

dsa OBJECT

PORTS { readPort [S],
searchPort [S],
modifyPort [S],
chainedReadPort,
chainedSearchPort
chainedModifyPort}

::= id-ot-dsa

chainedReadPort PORT

ABSTRACT OPERATIONS {
ChainedRead, ChainedCompare,
ChainedAbandon}
::= id-pt-chained-read

chainedSearchPort PORT

ABSTRACT OPERATIONS {
ChainedList, ChainedSearch}
::= id-pt-chained-search

```

chainedModifyPort PORT
  ABSTRACT-OPERATIONS {
    ChainedAddEntry, ChainedRemoveEntry,
    ChainedModifyEntry, ChainedModifyRDN}
  ::= id-pt-chained-modify

DSABind ::= ABSTRACT-BIND
  TO {chainedRead,
    chainedSearch,
    chainedModify}
  DirectoryBind

DSAUnbind::= UNBIND
  FROM {chainedRead,
    chainedSearch,
    chainedModify}

-- operations, arguments and results --

ChainedRead ::=
  ABSTRACT-OPERATION
    ARGUMENT      OPTIONALLY-SIGNED    SET{
      ChainingArgument,
      [0] ReadArgument}
    RESULT        OPTIONALLY-SIGNED    SET{
      ChainingResult,
      [0] ReadResult}
    ERRORS {
      DsaReferral, Abandoned, AttributeError, NameError,
      ServiceError, SecurityError}

ChainedCompare ::=
  ABSTRACT-OPERATION
    ARGUMENT      OPTIONALLY-SIGNED    SET{
      ChainingArgument,
      [0] CompareArgument}
    RESULT        OPTIONALLY-SIGNED    SET{
      ChainingResult,
      [0] CompareResult}
    ERRORS {
      DsaReferral, Abandoned, AttributeError, NameError,
      ServiceError, SecurityError}

ChainedAbandon ::= Abandon

ChainedList ::=
  ABSTRACT-OPERATION
    ARGUMENT      OPTIONALLY-SIGNED    SET{
      ChainingArgument,
      [0] ListArgument}
    RESULT        OPTIONALLY-SIGNED    SET{
      ChainingResult,
      [0] ListResult}
    ERRORS {
      DsaReferral, Abandoned, AttributeError, NameError,
      ServiceError, SecurityError }

ChainedSearch ::=
  ABSTRACT-OPERATION
    ARGUMENT      OPTIONALLY-SIGNED    SET{
      ChainingArgument,
      [0] SearchArgument}
    RESULT        OPTIONALLY-SIGNED    SET{
      ChainingResult,
      [0] SearchResult}
    ERRORS {
      DsaReferral, Abandoned, AttributeError, NameError,
      ServiceError, SecurityError}

```

```

ChainedAddEntry ::=
  ABSTRACT-OPERATION
    ARGUMENT      OPTIONALLY-SIGNED  SET{
                                         ChainingArgument,
                                         [0] AddEntryArgument}
    RESULT        OPTIONALLY-SIGNED  SET{
                                         ChainingResult,
                                         [0] AddEntryResult}
    ERRORS {
      DsaReferral, Abandoned, AttributeError, NameError,
      ServiceError, SecurityError, UpdateError}

ChainedRemoveEntry ::=
  ABSTRACT-OPERATION
    ARGUMENT      OPTIONALLY-SIGNED  SET{
                                         ChainingArgument,
                                         [0] RemoveEntryArgument}
    RESULT        OPTIONALLY-SIGNED  SET{
                                         ChainingResult,
                                         [0] RemoveEntryResult}
    ERRORS {
      DsaReferral, Abandoned, NameError,
      ServiceError, SecurityError, UpdateError}

ChainedModifyEntry ::=
  ABSTRACT-OPERATION
    ARGUMENT      OPTIONALLY-SIGNED  SET{
                                         ChainingArgument,
                                         [0] ModifyEntryArgument}
    RESULT        OPTIONALLY-SIGNED  SET{
                                         ChainingResult,
                                         [0] ModifyEntryResult}
    ERRORS {
      DsaReferral, Abandoned, AttributeError, NameError,
      ServiceError, SecurityError, UpdateError}

ChainedModifyRDN ::=
  ABSTRACT-OPERATION
    ARGUMENT      OPTIONALLY-SIGNED  SET{
                                         ChainingArgument,
                                         [0] ModifyRDNArgument}
    RESULT        OPTIONALLY-SIGNED  SET{
                                         ChainingResult,
                                         [0] ModifyRDNResult}
    ERRORS {
      DsaReferral, Abandoned, NameError,
      ServiceError, SecurityError, UpdateError}

-- errors and parameters --

DSASReferral ::=
  ABSTRACT-ERROR
    PARAMETER SET {
      [0] ContinuationReference,
      contextPrefix [1] DistinguishedName OPTIONAL}

-- common arguments/results --

ChainingArguments ::=
  SET {
    originator      [0] DistinguishedName OPTIONAL,
    targetObject    [1] DistinguishedName OPTIONAL,
    operationProgress [2] OperationProgress DEFAULT {notStarted}
    traceInformation [3] TraceInformation,
    aliasDereferenced [4] BOOLEAN DEFAULT FALSE,
    aliasedRDNs     [5] INTEGER OPTIONAL,
    -- absent unless aliasDereferenced is TRUE
  }

```

returnCrossRefs	[6]	BOOLEAN DEFAULT FALSE,
referenceType	[7]	ReferenceType DEFAULT superior,
info	[8]	Domaininfo OPTIONAL,
timeLimit	[9]	UTCTime OPTIONAL,
	[10]	SecurityParameters DEFAULT { }
ChainingResults	::=	SET {
info	[0]	Domaininfo OPTIONAL,
crossReferences	[1]	SEQUENCE OF CrossReference OPTIONAL,
	[2]	SecurityParameters DEFAULT { }
CrossReference	::=	SET {
contextPrefix	[0]	DistinguishedName,
accessPoint	[1]	AccessPoint)
ReferenceType	::=	ENUMERATED {
superior		(1),
subordinate		(2),
cross		(3),
nonSpecificSubordinate		(4)}
TraceInformation	::=	SEQUENCE OF
SEQUENCE {		
targetObject		Name,
dsa Name,		
OperationProgress}		
OperationProgress	::=	SET {
nameResolutionPhase	[0]	ENUMERATED {
		notStarted (1),
		proceeding (2),
		completed (3)},
nextRDNTToBeResolved	[1]	INTEGER OPTIONAL}
DomainInfo	::=	ANY
ContinuationReference	::=	SET {
targetObject	[0]	Name,
aliasedRDNs	[1]	INTEGER OPTIONAL,
operationProgress	[2]	OperationProgress
rdnsResolved	[3]	INTEGER OPTIONAL,
referenceType	[4]	ReferenceType OPTIONAL,
		-- only present in the DSP --
accessPoints	[5]	SET OF AccessPoint }
AccessPoint	::=	SET {
ae-title	[0]	Name,
address	[1]	PresentationAddress }

ANNEX B

(to Recommendation X.518)

Modelling of knowledge

This Annex is not part of the Recommendation.

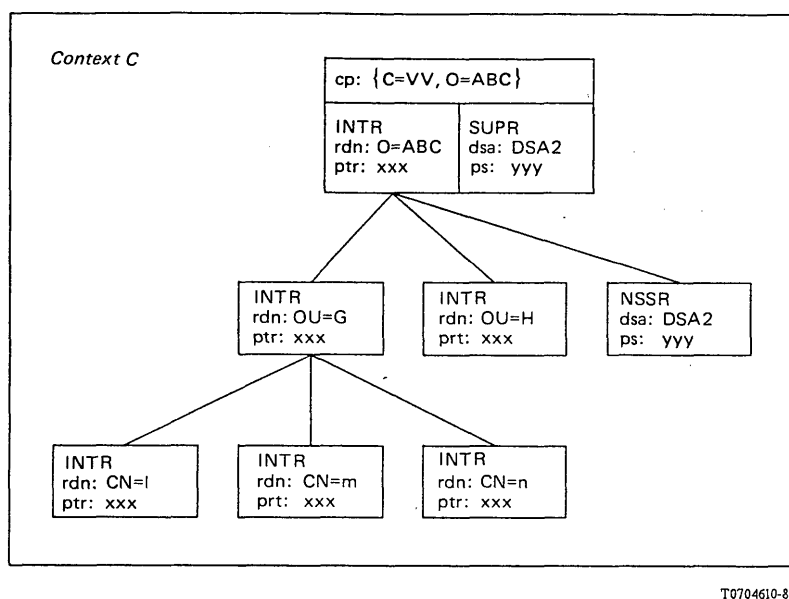
B.1 Example of knowledge modelling

The following example illustrates the knowledge information that would have to be maintained by the DSAs shown in Figure 5/X.518 (§ 9). Figure 5/X.518 depicts a hypothetical DIT logically partitioned into five Naming Contexts (A, B, C, D and E) and physically distributed over three DSAs (DSA1, DSA2, DSA3). In the example, DSA1 holds context C, DSA2 holds contexts A, B, and E, and DSA3 holds context D.

The following abbreviations have been used in Figures B-1/X.518 to B-3/X.518.

SUPR: superior reference
 SUBR: subordinate reference
 INTR: internal reference
 NSSR: non-specific subordinate reference
 CROSSR: cross reference
 DSA_n: Distinguished Name of DSA_n
 PS: Presentation Address
 CP: context prefix
 RDN: Relative Distinguished name
 DSA: Distinguished name of a DSA
 PTR: Pointer
 AON: Aliased Object Name.

Note - The following figures are intended only to provide a pictorial example of the concepts defined in this paragraph. How knowledge information is actually stored and managed in a particular DSA implementation is a *local matter* and is outside the scope of this Recommendation.



T0704610-88

FIGURE B-1/X.518

, Knowledge information for DSA1

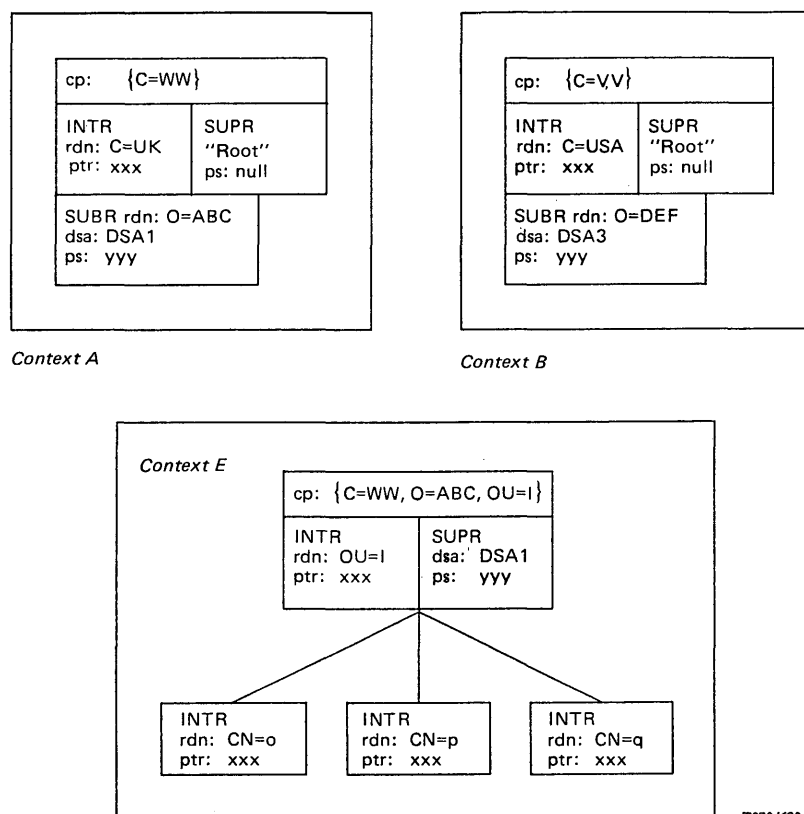
Figure B-1/X.518 illustrates the knowledge information that must be held by DSA1. This must include the following context prefixes and set of references:

Context Prefixes: {C=WW, O=ABC}, context C.
 Cross References: { }
 Superior References: {DSA2, presentation address of DSA2}
 Internal References
 for Context C: {C=WW, O=ABC},
 {OU=G}, {OU=H}
 {OU=G, CN=l},
 {OU=G, CN=m},
 {OU=G, CN=n}.

Subordinate References: { }

Non-specific subordinate

References: {DSA2, presentation address of DSA2}.



T0704620-88

FIGURE B-2/X.518

Knowledge information for DSA2

Figure B-2/X.518 illustrates the knowledge information that must be held by DSA2. This must include the following context prefixes and set of references:

Context Prefixes: {C=WW}, context A
{C=VV}, context B
{C=WW, O=ABC, OU=I}, context E.

Cross References: { }

Superior References: { }

Internal References
for Context A: {C=WW}

Internal References
for Context B: {C=VV}

Internal References
for Context E: {C=WW, O=ABC, OU=I},
{CN=o},
{CN=p},
{CN=q}.

Subordinate References
for Context A: {C=WW, O=ABC}

Subordinate References
for Context B: {C=VV, O=DEF}

Non-specific subordinate
References: { }

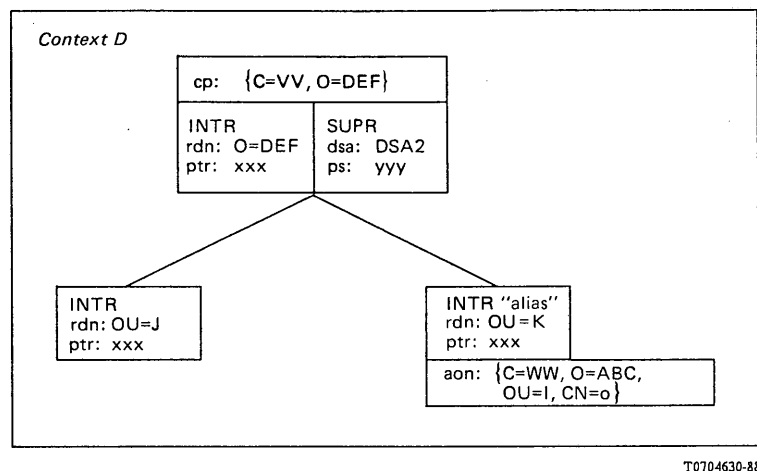


FIGURE B-3/X.518

Knowledge information for DSA3

Figure B-3/X.518 illustrates the knowledge information that must be held by DSA3. This must include the following context prefixes and set of references:

Context Prefixes:	{C=VV, O=DEF}, context D
Cross References:	{{C=WW, O=ABC, OU=H}, DSA1, presentation address of DSA1} (not shown in the figure above)
Superior References:	{DSA2, presentation address of DSA2}
Internal References for Context D:	{DSA1, presentation address of DSA1} {C=VV, O=DEF}, {OU=J}, {OU=K} alias for {C=WW, O=ABC, OU=I, CN=o} (alias information is not part of the knowledge)
Subordinate References:	{ }
Non-specific subordinate References:	{ }

B.2 Example of distributed name resolution

The following is an example of how Distributed Name Resolution is used to process different directory requests. The example is based on the hypothetical DIT shown in Figure 5/X.518 (§ 9) and the corresponding DSA configuration(s) shown in Figures B-1/X.518 to B-3/X.518 (Annex B).

Assuming a chaining mode of propagating, the following requests addressed to DSA1 would be processed as follows:

- 1) A request with distinguished name {C=WW, O=ABC, OU=G, CN=1}
 - Will match context prefix {C=WW, O=ABC} of context C for which DSA1 has administrative authority. Therefore, name resolution will begin in DSA1 with context C.
 - Name resolution will proceed downwards in context C successfully matching each remaining RDN, until CN=1 is located.

- 2) A request with distinguished name {C=WW, O=JPR}
 - Will not match any context prefix held by DSA1, therefore DSA1 will use its superior reference to forward the request to its superior DSA, DSA2.
 - In DSA2, the request will match context prefix {C=WW} and name resolution will begin in DSA2 with context A.
 - Name resolution will not find a subordinate of C=WW to match RDN O=JPR, therefore the request will fail and the name will be determined to have been invalid (i.e. reference a non-existent object).
- 3) A request with distinguished name {C=VV, O=DEF, OU=K}
 - Will not match any context prefix held by DSA1.
 - DSA1 will therefore forward the request to its superior DSA, DSA2.
 - The request will match context prefix {C=VV} of context B held by DSA2. Therefore, name resolution will begin in DSA2 with context B.
 - As name resolution attempts to match O=DEF, it will find a subordinate reference indicating that {C=VV, O=DEF} is the start of a new context held in DSA3.
 - Name resolution will continue in DSA3 until {C=VV, O=DEF, CN=K} is located.
 - Assuming that aliases are to be dereferenced, a new name will be constructed using the aliased name contained in the entry {C=VV, O=DEF, CN=K}. The resulting new name will be: {C=WW, O=ABC, OU=I, CN=o}.
 - DSA3 will resume processing of the request using the new name obtained by dereferencing.

ANNEX C

(to Recommendation X.518)

Distributed use of authentication

This Annex is not part of the Recommendation.

C.1 *Summary*

The security model is defined in § 10 of Recommendation X.501. The following is a summary of the main points of the model.

- a) Simple Authentication of the operation initiator is not supported in the DSP.
- b) Strong Authentication, by the signing of the request and of the result, is supported in the DSP.
- c) Encryption of the request, or of the result, is not supported in the DSP.
- d) Authentication of errors, including referrals, is not supported in the DSP.

This Annex describes how b) above is realized in the distributed Directory. It makes use of terminology and notation defined in Recommendation X.509.

C.2 *Simple authentication*

The DUA will be authenticated as part of the Bind Operation of the DAP. Thereafter, only the name of the DUA will be carried in the DSP, in the initiator field of the Chaining Argument.

C.3 Distributed authentication model

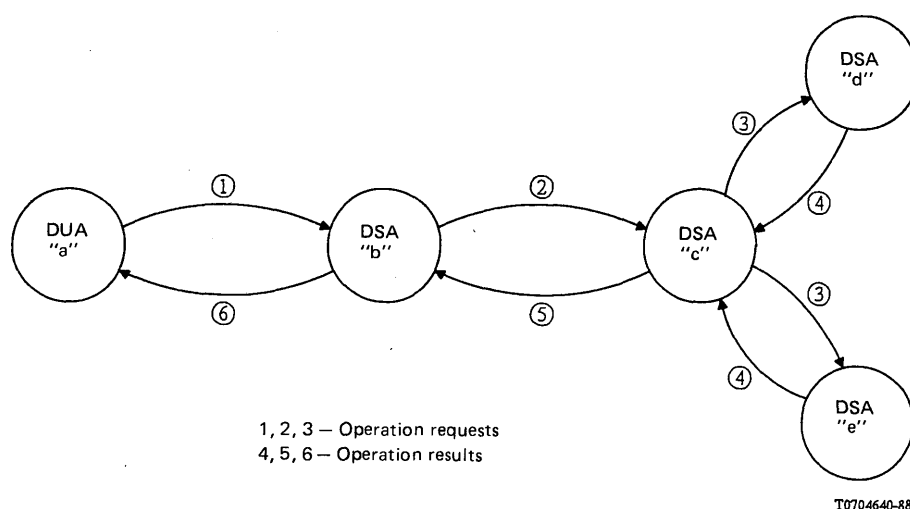


FIGURE C-1/X.518

Distributed authentication model

Figure C-1/X.518 illustrates the model to be used to specify the distributed authentication procedures. The model identifies the sequence of information flows for the general case of a list or search operation. The operation is considered as originating from DUA "a" citing a target object which resides in DSA "c"; in performing the operation, DSAs "b", "c", "d" and "e" are to be involved.

DUA "a" initially contacts any DSA (DSA "b") which does not hold the target object, but which is able to navigate, via chaining, to the DSA (DSA "c") holding the target object. If all the DSAs were operating in referral mode, then the model would be significantly simplified, and each DUA/DSA exchange would equate, in authentication terms, to the interaction between DUA "a" and DSA "b".

C.4 DUA to DSA

Originator authentication is realized as a consequence of exchange (1). In Figure C-1/X.518 the authentication procedure is as follows:

Let

OA = the Operation Argument i.e. Search, Read, Compare etc. Argument as defined in Part 3.

and

a(OA) = the Operation Argument signed by DUA "a".

Authentication will be determined by verification of the signature.

C.5 Transference from the DAP to the DSP

This procedure is effected by DSA "b" in Figure C-1/X.518 and represents the transference of the signed identity of the initiator from the DAP to the DSP.

DSA "b" formulates the appropriate Chaining Argument as described in § 12.3 of this Recommendation and combines it with the Operation Argument from the DAP thus forming a Chained Operation, i.e. Chained Read, Search, List etc. of the DSP. The Chained Operation so formed will be signed prior to passing it to other DSAs (DSA "c" in Figure C-1/X.518). The data structure can be represented as:

$b\{ChA, a(OA)\}$ = the Chained Operation signed by
DSA b

where

ChA = Chaining Argument.

Authentication information carried in the DSP between two DSAs [labelled exchange (2) in Figure C-1/X.518] therefore comprises two parts:

- the Operation Argument, signed by the initiator, which allows authentication of the initiator;
- the Chained Operation, signed by the sending DSA, which allows authentication of the sending DSA.

C.6 Chaining through intermediate DSAs

This procedure would be effected by DSA "c" in the model depicted in Figure C-1/X.518. DSA "c" will discard the signature provided by the sending DSA (DSA "b" in Figure C-1/X.518), and will modify the Chaining Argument, as described in § 12.3 of this Recommendation. DSA "c" shall then combine the modified Chaining Argument with the signed Operation Argument, and sign the result to create a modified signed Chained Operation. This can be represented by:

$c\{ChA', a\{OA\}\}$ = the Chained Operation signed by DSA "c"

where

ChA' = modified Chaining Argument.

The modified Chained Operation is represented in Figure C-1/X.518 by exchange (3). Depending upon the nature of the operation, and upon the type of knowledge held, DSA "c" may perform request decomposition prior to chaining or multicasting any resultant operation(s). This has been represented in Figure C-1/X.518 by DSA "c" sending operations to DSA "d" and DSA "e"; in each case the authentication procedure is identical.

C.7 Results authentication

The results authentication service is requested by an initiator of a directory operation using the **signed** option within the **protectionRequest SecurityParameter**. In providing a response to such a request a DSA may optionally decide whether or not to sign any or all of the results; the results authentication service does not provide for the authentication of error responses.

Within the context of a particular DSA processing results from an arbitrary number of DSAs (each of which are associated with a particular service request) the following distinct cases are possible:

- the DSA provides a complete set of results for an operation without the need to perform any collating function (represented by DSA "d" and DSA "e" in Figure C-1/X.518);
- the DSA collates local results (sourced by this DSA) with the results from one or more other DSAs (represented by DSA "c" in Figure C-1/X.518);
- the DSA chains a result from a DSA to either another DSA or a DUA and does not contribute to the result set as it does so (represented by DSA "b" in Figure C-1/X.518).

C.7.1 DSA results - no collation

This paragraph addresses the role of a DSA in being the sole source of results to a particular operation request, i.e. the DSA has no collation function to perform. The paragraph considers the case for both the DSP and the DAP.

C.7.1.1 DSP

The DSA can choose to perform either of the following procedures:

- return the results unsigned, this can be represented by:
 ChR, OR = Chained Operation Result (unsigned)
 where
 ChR = Chaining Results
 OR = Operation Result;
- sign only the Operation Result, this can be represented by:
 $ChR, d(OR)$ = Operation Result signed by DSA "d";
- sign only the Chained Operation Result, which can be represented as:

d (ChR, OR) = Chained Operation Result signed by DSA "d"

- sign both the Operation Result and the Chained Operation Result, which can be represented by:

d{ChR, D{OR}} = Operation Result and Chained Operation Result
signed by DSA "d".

Note - For the case where the Operation Result is signed, the signed result will be carried back to the initiator; for the case where the Chained Operation Result has been signed, the receiving DSA will have to discard the signature in order to modify the Chaining Results argument prior to forwarding the Chained Operation Result.

C.7.1.2 DAP

This is fully described in Recommendation X.511, a summary is reproduced here for completeness.

The DSA can choose to either return the results unsigned, which can be represented by:

OR = Operation Result

or, signed, which can be represented by:

d{OR} = Operation Result signed by DSA "d".

C.7.2 DSA results - collation included

This paragraph addresses the role of a DSA in returning the result of particular service requests where collation and integration of results from other DSAs is a necessary prerequisite. The paragraph considers the case for both the DSP and the DAP.

C.7.2.1 DSP

Recognizing that zero or more results received from other DSAs may be signed, this procedure enables a DSA to collate and integrate the results and sign zero or more constituent parts of the composite result and optionally, sign the composite result as a whole.

C.7.2.1.1 Production of the chaining results argument

This procedure requires that a DSA (represented by DSA "c" in Figure C-1/X.518) remove all of the Chained Operation Result signatures from the results received from external DSAs (DSA "d" and DSA "e" in Figure C-1/X.518). DSA "c" then possesses a set of unsigned Chaining results, a set of signed Operation Results, and a set of unsigned Operation Results.

All the Chaining Results are manipulated as described in § 12.4 of this Recommendation to create a single modified Chaining Result, denoted by:

i) ChR' = modified Chaining Results.

C.7.2.1.2 Unsigned locally derived result

If the DSA does not wish to sign the locally generated results, the set of unsigned Operation Results are merged with the local result to form a modified set of Operation Results, denoted by:

OR' = Merged Operation Result.

The complete set of Operation Results is then the union of the set of externally signed Operation Results denoted by:

d{OR}, e{OR} ...

and the Merged Operation Result, collectively denoted by:

(ii) OR', d{OR}, e{OR} ... = Operation Result.

C.7.2.1.3 Signed locally derived result

If the DSA does wish to sign the locally generated results, then the externally generated set of unsigned Operation Results are first merged together. The complete set of Operation Results is

then the union of the locally signed set of Operation Results denoted by $C\{OR\}$, the merged set of externally unsigned Operation Results denoted by, OR'' , and the set of externally signed Operation Results denoted by:

$d\{OR\}$, $e\{OR\}$, ..., which are collectively denoted as:

(iii) $c\{OR\}$, OR'' , $d\{OR\}$, $e\{OR\}$, ... = Operation Result.

C.7.2.1.4 *Unsigned chained operation result*

If the DSA does not wish to sign the Chained Operation Result, then the latter will comprise the Chaining Results (identified in (i) above) added to the Operation Result identified in either (ii) or (iii) above, collectively, these are denoted by:

either:

ChR' , OR' , $d\{OR\}$, $e\{OR\}$, ... = Chained Operation Result (unsigned).

or,

ChR' , $c\{OR\}$, OR'' , $d\{OR\}$, $e\{OR\}$, ... = Chained Operation Result (unsigned) and Operation Result signed by DSA "c".

C.7.2.1.5 *Signed chained operation result*

If the DSA does wish to sign the Chained Operation Result, then the result will comprise the Chaining Results (identified in (i) above) added to the Operation Result (identified in either (ii) or (iii) above), collectively denoted as:

either:

$c\{ChR', OR', d\{OR\}, e\{OR\}, \dots\}$ = Chained Operation Result signed by DSA "c"

or,

$c\{ChR', c\{OR\}, OR'', d\{OR\}, e\{OR\}, \dots\}$ = Chained Operation Result and Operation Result signed by DSA "c".

C.7.2.2 *DAP*

The procedure is very similar to that described in § C.7.2.1, with the exception that the Chaining Results argument is not passed in the DAP.

C.7.3 *DSA chained results*

This paragraph addresses the procedures to be effected by a DSA in chaining an operation result back to the requestor, DSA or DUA, within the DSP and DAP respectively.

C.7.3.1 *DSP*

The DSA initially removes the signature (if one exists) from the Chained Operation Result. It then manipulates the Chaining Results argument as described in this Recommendation, to produce a modified Chaining Results argument. The latter is then merged back with the Operation Result argument to produce a modified Chained Operation Result. Finally, the DSA may optionally sign the Chained Operation Result before passing it to the next DSA in the chain.

C.7.3.2 *DAP*

A DSA (represented by DSA "b" in Figure C-1/X.518) first removes the signature (if one exists) from the Chained Operation Result. It then analyses and discards the Chaining Results argument and, finally, it optionally signs the remaining Operation Result argument before passing the result to the DUA.

ANNEX D

(to Recommendation X.518)

Distributed directory object identifiers

This Annex is part of the Recommendation.

This Annex includes all of the ASN.1 object identifiers contained in this Recommendation in the form of the ASN.1 module **DistributedDirectoryObjectIdentifiers**.

```

DistributedDirectoryObjectIdentifiers      {joint-iso-ccitt ds(5) modules(1)
distributedDirectoryObjectIdentifiers(13)}

DEFINITION::=
BEGIN

EXPORTS
    id-ot-dsa, id-pt-chainedRead, id-pt-chainedSearch, id-pt-chainedModify;

IMPORTS
    id-ot, id-pt
    FROM      UsefulDefinitions {joint-iso-ccitt ds(5) modules(1) usefulDefinitions(0)};

-- objects --

id-ot-dsa  OBJECT IDENTIFIER ::= {id-ot 3}

-- part types --

id-pt-chainedRead  OBJECT IDENTIFIER ::= {id-pt 4}
id-pt-chainedSearch OBJECT IDENTIFIER ::= {id-pt 5}
id-pt-chainedModify OBJECT IDENTIFIER ::= {id-pt 6}

END

```

Recommendation X.519

THE DIRECTORY - PROTOCOL SPECIFICATIONS¹⁾

(Melbourne, 1988)

CONTENTS

0	<i>Introduction</i>
1	<i>Scope</i>
2	<i>References</i>
3	<i>Definitions</i>
	3.1 OSI Reference Model Definitions
	3.2 Basic Directory Definitions
	3.3 Distributed Operation Definitions
4	<i>Abbreviations</i>
5	<i>Conventions</i>

¹⁾ Recommendation X.519 and ISO 9594-5, The Directory - Protocol Specifications, were developed in close collaboration and are technically aligned.

6 *Protocol Overview*

- 6.1 Directory Protocol Model
- 6.2 Directory Access Protocol
- 6.3 Directory System Protocol
- 6.4 Use of Underlying Services

7 *Directory Protocol Abstract Syntax*

- 7.1 Abstract Syntaxes
- 7.2 Directory Application Service Elements
- 7.3 Directory Application Contexts
- 7.4 Errors

8 *Mapping onto Used Services*

- 8.1 Mapping onto ACSE
- 8.2 Mapping onto ROSE

9 *Conformance*

- 9.1 Conformance by DUAs
- 9.2 Conformance by DSAs

Annex A - DAP in ASN.1

Annex B - DSP in ASN.1

Annex C - Reference Definition of Protocol Object Identifiers

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. The set of all 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 specifies the application service elements and application contexts for two protocols - the Directory Access Protocol (DAP) and the Directory System Protocol (DSP). The DAP provides for access to the Directory to retrieve or modify Directory information. The DSP provides for the chaining of requests to retrieve or modify Directory information to other parts of the distributed Directory System where the information may be held.

1 **Scope**

This Recommendation specifies the Directory Access Protocol and the Directory System Protocol, fulfilling the abstract services specified in Recommendations X.511 and X.518.

2 References

Recommendation X.200 - Open Systems Interconnection - Basic Reference Model
Recommendation X.208 - Open Systems Interconnection - Specification of Abstract Syntax Notation (ASN.1)
Recommendation X.209 - Open Systems Interconnection - Specification of Basic Encoding rules for Abstract Syntax Notation One (ASN.1)
Recommendation X.500 - The Directory - Overview of Concepts, Models and Services
Recommendation X.501 - The Directory - Information Framework
Recommendation X.511 - The Directory - Abstract Service Definition
Recommendation X.518 - The Directory - Procedures for Distributed Operation
Recommendation X.520 - The Directory - Selected Attribute Types
Recommendation X.521 - The Directory - Selected Object Classes
Recommendation X.219 - Remote Operations - Model, Notation and Service Definition
Recommendation X.229 - Remote Operations - Protocol Specification
Recommendation X.217 - Open Systems Interconnection - Association Control: Service Definition
Recommendation X.227 - Open Systems Interconnection - Association Control: Protocol Specification
Recommendation X.216 - Open Systems Interconnection - Presentation Layer Service Definition.

3 Definitions

The definitions contained in this paragraph make use of the abbreviations defined in § 4.

3.1 *OSI Reference Model definitions*

This Recommendation is based on the concepts developed in Recommendation X.200 and makes use of the following terms defined therein:

- a) *application-service-element*;
- b) *application-protocol-control-information*;
- c) *application-control-data-unit*;
- d) *application-context*;
- e) *application-entity*;
- f) *abstract-syntax*.

3.2 *Basic Directory definitions*

This Recommendation makes use of the following terms defined in Recommendation X.501:

- a) *the Directory*;
- b) *(Directory) user*;
- c) *Directory System Agent (DSA)*;
- d) *Directory User Agent (DUA)*.

3.3 *Distributed Operation definitions*

This Recommendation makes use of the following terms defined in Recommendation X.518:

- a) *chaining*;
- b) *referral*.

4 Abbreviations

The following abbreviations are used in this Recommendation:

AC	Application Context
ACSE	Association Control Service Element
AE	Application Entity
APCI	Application Protocol Control Information
APDU	Application Protocol Data Unit
ASE	Application Service Element
DAP	Directory Access Protocol
DSA	Directory System Agent
DSP	Directory System Protocol
DUA	Directory User Agent
ROSE	Remote Operations Service Element.

5 Conventions

The Recommendation makes use of the following conventions:

- the abstract syntax definitions in § 7 are defined using the abstract syntax notation defined in Recommendation X.208;
- the remote operation macros (RO-notation), and the application-service-element and application-context macros are defined in Recommendation X.219;
- the words of defined terms and the names and values of service parameters and protocol fields, unless they are proper names, begin with a lower-case letter and are linked by a hyphen thus: defined-term. Proper names begin with an upper case letter and are not linked by a hyphen thus: Proper Name.

6 Protocol Overview

6.1 Directory Protocol Model

Recommendation X.511 defines the abstract service between a DUA and the Directory to support a user accessing Directory services. The Directory is further modelled as being represented by a DSA which supports the particular access point concerned. Recommendation X.518 defines the interactions between a pair of DSAs within the Directory to support user requests which are chained. These concepts are illustrated in Figure 1/X.519.

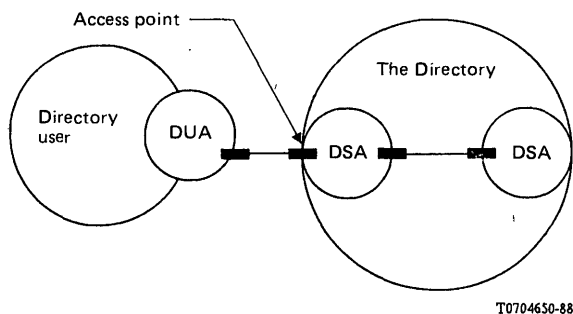


FIGURE 1/X.519

Directory interactions

When a DUA is in a different open system from a DSA with which it is interacting, these interactions are supported by the Directory Access Protocol (DAP), which is an OSI application layer protocol. Similarly, when a pair of DSAs which are interacting are in different open systems, the interactions are supported by the Directory System Protocol (DSP), which is also in the application layer.

Both the DAP and the DSP are protocols to provide communication between a pair of application processes. In the OSI environment this is represented as communication between a pair of application-entities (AEs) using the presentation service. The function of an AE is provided by a set of application-service-elements (ASEs). The interaction between AEs is described in terms of their use of the services provided by the ASEs. The two ASEs common to both of the directory protocols are summarized in this paragraph.

The Remote Operations Service Element (ROSE) supports the request/reply paradigm of the abstract operation that occurs at the ports in the abstract model. The Directory ASEs provide the mapping function of the abstract-syntax notation of the directory abstract-service onto the services provided by the ROSE.

The Association Control Service Element (ACSE) supports the establishment and release of an application-association between a pair of AEs. Associations between a DUA and a DSA may be established only by the DUA. Only the initiator of an established association can release it.

6.2 *Directory Access Protocol*

The Directory Access Protocol (DAP) is used to realise the Directory Abstract Service. It comprises three directory specific ASEs in addition to ROSE and ACSE. These are: **readASE**, **searchASE**, and **modifyASE**. They correspond to the **readPort**, **searchPort**, and **modifyPort** of the abstract service. The **directoryAccessAC** application context identifies the combination of: **readASE**, **searchASE**, and **modifyASE**, **aCSE**, **rOSE**.

6.3 *Directory System Protocol*

The Directory System Protocol (DSP) is used to realise the functionality of distributed operation described in Recommendation X.518. It comprises three directory specific ASEs in addition to ROSE and ACSE. These are: **chainedReadASE**, **chainedSearchASE**, and **chainedModifyASE**. They correspond to the **chainedReadPort**, **chainedSearchPort**, and **chainedModifyPort** of the abstract service. The **directorySystemAC** application context identifies the combination of: **chainedReadASE**, **chainedSearchASE**, and **chainedModifyASE**, **aCSE**, **rOSE**.

6.4 *Use of Underlying Services*

The DAP and DSP protocols make use of underlying services as described below.

6.4.1 *Use of ROSE services*

The Remote Operations Service Element (ROSE) is defined in Recommendation X.219.

The ROSE supports the request/reply paradigm of remote operations.

The Directory ASEs are users of the RO-INVOKE, RO-RESULT, RO-ERROR, RO-REJECT-U and RO-REJECT-P services of the ROSE.

The remote operations of the DAP and the DSP are Class 2 (asynchronous) operations. Note that as the DUA is a consumer of the DAP it may choose to operate in a synchronous manner.

DAP uses Association Class 1. This means that the DSA cannot invoke operations on the DUA. DSP uses Association Class 3. This means that the responding DSA can invoke operations on the initiating DSA and vice versa.

6.4.2 *Use of ACSE services*

The Association Control Service Element (ACSE) is defined in Recommendation X.217.

The ACSE provides for the control (establishment, release, abort) of application-associations between AEs.

The Directory Bind and Directory Unbind (or DSA Bind and DSA Unbind) are the sole users of the A-ASSOCIATE and A-RELEASE services of the ACSE in normal mode. The application-process is the user of the A-ABORT and A-P-ABORT services of the ACSE.

6.4.3 Use of the Presentation Service

The presentation-service is defined in Recommendation X.216.

The Presentation Layer coordinates the representation (syntax) of the Application Layer semantics that are to be exchanged.

In normal mode, a different presentation-context is used for each abstract-syntax included in the application-context.

The ACSE is the sole user of the P-CONNECT, P-RELEASE, P-U-ABORT and P-P-ABORT services of the presentation-service.

The ROSE is a user of the P-DATA service of the presentation-service.

6.4.4 Use of Lower Layer Services

The session-service is defined in Recommendation X.215. The Session Layer structures the dialogue of the flow of information between the end-systems.

The Kernel and Duplex functional units of the session-service are used by the Presentation Layer.

The transport-service is defined in Recommendation X.214. The Transport Layer provides for the end-to-end transparent transfer of data over the underlying network connection.

The choice of the class of transport-service used by the Session Layer depends on the requirements for multiplexing and error recovery. Support for Transport Class 0 (non-multiplexing) is mandatory. Transport Expedited Service is not used.

Support for other classes is optional. A multiplexing class may be used to multiplex the DAP or DSP and other protocols over the same network connection. An error recovery class may be chosen over a network connection with an unacceptable residual error rate.

An underlying network supporting the OSI network-service defined in Recommendation X.213 is assumed.

A network-address is as defined in Recommendation X.121, Recommendations E.163/E.164, or Recommendation X.200 (OSI NSAP-address).

7 Directory Protocol Abstract Syntax

7.1 Abstract Syntaxes

The Directory ASEs specified in §§ 7.2.1, 7.2.3 and 7.2.5 share a single abstract syntax, **id-as-directory-AccessAS**. Those specified in §§ 7.2.2, 7.2.4 and 7.2.6 also share a single abstract syntax **id-as-directorySystemAS**. In each case, this defines application-protocol-control-information (APCI) which, when used in conjunction with the ROSE, defines a set of APDUs. The Directory APDUs are defined by the abstract-syntax of the Directory ASEs and ROSE. These plus the abstract-syntax of ACSE form the complete definition of APDUs used during a Directory association.

The ACSE abstract-syntax **id-as-acse** is needed to establish the associations.

These abstract syntaxes shall (as a minimum) be encoded according to the ASN.1 Basic Encoding Rules.

7.2 Directory Application Service Elements

This paragraph specifies the ASEs which are used as "building blocks" in the construction of the various Directory application contexts in § 7.3.

Note - These ASEs are used for the construction of the application contexts defined in this Recommendation. They are not intended to allow for claims of conformance to individual, or other combinations of, ASEs.

7.2.1 *Read ASE*

The **readASE** supports the abstract-operations of the **readPort**, namely **Read**, **Compare**, and **Abandon**, as defined in Recommendation X.511.

```
readASE
  APPLICATION-SERVICE-ELEMENT
    CONSUMER INVOKES
      {read, compare, abandon}
    ::= id-ase-readASE
read      Read      ::= 1
compare   Compare   ::= 2
abandon   Abandon   ::= 3
```

7.2.2 *Chained Read ASE*

The **chainedReadASE** supports the abstract-operation of the **ChainedReadPort**, i.e. **ChainedRead**, **ChainedCompare** and **ChainedAbandon**, as defined in Recommendation X.518.

```
chainedReadASE
  APPLICATION-SERVICE-ELEMENT
    OPERATIONS {
      chainedRead,
      chainedCompare
      chainedAbandon}
    ::= id-ase-chainedReadASE
chainedRead      ChainedRead      ::= 1
chainedCompare   ChainedCompare   ::= 2
chainedAbandon   ChainedAbandon   ::= 3
```

7.2.3 *Search ASE*

The **searchASE** supports the abstract-operations of the **SearchPort**, namely **List** and **Search**, as defined in Recommendation X.511.

```
searchASE
  APPLICATION-SERVICE-ELEMENT
    CONSUMER INVOKES { list, search}
    ::= id-ase-searchASE
list      List      ::= 4
search    Search    ::= 5
```

7.2.4 *Chained Search ASE*

The **chainedSearchASE** supports the abstract-operations of the **ChainedSearchPort**, namely **ChainedList** and **ChainedSearch**, as defined in Recommendation X.518.

```
chainedSearchASE
  APPLICATION-SERVICE-ELEMENT
    OPERATIONS {
      chainedList, chainedSearch}
    ::= id-ase-chainedSearchASE
chainedList      ChainedList      ::= 4
chainedSearch    ChainedSearch    ::= 5
```

7.2.5 *Modify ASE*

The **modifyASE** supports the abstract-operations of the **ModifyPort**, namely **AddEntry**, **RemoveEntry**, **ModifyEntry**, and **ModifyRDN**, as defined in Recommendation X.511.

```

modifyASE
  APPLICATION-SERVICE-ELEMENT
    CONSUMER INVOKES
      {addEntry, removeEntry,
       modifyEntry, modifyRDN}
  ::= id-ase-modifyASE
addEntry      AddEntry      ::= 6
removeEntry   RemoveEntry   ::= 7
modifyEntry   ModifyEntry    ::= 8
modifyRDN     ModifyRDN     ::= 9

```

7.2.6 Chained Modify ASE

The **chainedModifyASE** supports the abstract-operations of the **ChainedModifyPort**, namely **ChainedAddEntry**, **ChainedRemoveEntry**, **ChainedModifyEntry** and **ChainedModifyRDN**, as defined in Recommendation X.518.

```

chainedModifyASE
  APPLICATION-SERVICE-ELEMENT
    OPERATIONS
      {chainedAddEntry,
       chainedRemoveEntry,
       chainedModifyEntry,
       chainedModifyRDN}
  ::= id-ase-chainedModifyASE
chainedAddEntry      ChainedAddEntry      ::= 6
chainedRemoveEntry   ChainedRemoveEntry    ::= 7
chainedModifyEntry   ChainedModifyEntry    ::= 8
chainedModifyRDN     ChainedModifyRDN     ::= 9

```

7.3 Directory Application Contexts

7.3.1 Directory Access Application Context

The **directoryAccessAC** allows the DUA to access the operations of the following ASEs: **readASE**, **searchASE**, **modifyASE**.

```

directoryAccessAC
  APPLICATION-CONTEXT
    APPLICATION SERVICE ELEMENTS
      {aCSE}
    BIND DirectoryBind
    UNBIND DirectoryUnbind
    REMOTE OPERATIONS {rOSE}
    INITIATOR CONSUMER OF {
      readASE,
      searchASE,
      modifyASE}
    ABSTRACT SYNTAXES {
      id-as-acse,
      id-as-directoryAccessAS}
  ::= id-ac-directoryAccessAC

```

7.3.2 Directory System Application Context

The **directorySystemAC** allows DSAs to communicate for the purpose of chaining operations.

```

directorySystemAC
  APPLICATION-CONTEXT
    APPLICATION SERVICE ELEMENTS
      {aCSE}
    BIND DSABind
    UNBIND DSAUnbind

```

REMOTE OPERATIONS {rOSE}
 OPERATIONS OF
 {chainedReadASE,
 chainedSearchASE,
 chainedModifyASE}
 ABSTRACT SYNTAXES {
 id-as-acse,
 id-as-directorySystemAS}
 ::= id-ac-directorySystemAC

7.4 Errors

Corresponding to each abstract-error defined in the Abstract Service is an error value which may be conveyed by the protocol. The assignments follow:

abandoned	Abandoned	::= 5
attributeError	AttributeError	::= 1
nameError	NameError	::= 2
referral	Referral	::= 4
securityError	SecurityError	::= 6
serviceError	ServiceError	::= 3
updateError	UpdateError	::= 8
dSAReferral	DSAReferral	::= 9
abandonFailed	AbandonFailed	::= 7

8 Mapping onto Used Services

This paragraph defines the mapping of the DAP and DSP onto the used services.

8.1 Mapping onto ACSE

This paragraph defines the mapping of the abstract-bind (**DirectoryBind** or **DSABind**) and abstract-unbind (**DirectoryUnbind** or **DSAUnbind**) services onto the services of the ACSE. The ACSE is defined in Recommendation X.217.

8.1.1 Abstract-bind onto A-ASSOCIATE

The abstract-bind service is mapped onto the A-ASSOCIATE service of the ACSE. The use of the parameters of the A-ASSOCIATE service is qualified in the following subparagraphs.

8.1.1.1 Mode

This parameter shall be supplied by the initiator of the association in the A-ASSOCIATE request primitive, and shall have the value "normal mode".

8.1.1.2 Application Context Name

The initiator of the association shall propose either the **directoryAccessAC** or the **directorySystemAC** application-context.

8.1.1.3 User information

The mapping of the bind-operation of the abstract-bind service onto the User Information parameters of the A-ASSOCIATE request primitive is defined in Recommendation X.219.

8.1.1.4 Presentation Context Definition List

The initiator of the association shall supply the Presentation Context Definition List in the A-ASSOCIATE request primitive which shall contain the ACSE abstract-syntax (**id-as-acse**) and either the DAP abstract-syntax (**id-as-directoryAccessAS**) or the DSP abstract-syntax (**id-as-directorySystemAS**).

8.1.1.5 *Quality of service*

This parameter shall be supplied by the initiator of the association in the A-ASSOCIATE request primitive, and by the responder of the association in the A-ASSOCIATE response primitive. The parameters "Extended Control" and "Optimized Dialogue Transfer" shall be set to "feature not desired". The remaining parameters shall be such that default values are used.

8.1.1.6 *Session requirements*

This parameter shall be set by the initiator of the association in the A-ASSOCIATE request primitive, and by the responder of the association in the A-ASSOCIATE response primitive. The parameter shall be set to specify the following functional units:

- a) Kernel;
- b) Duplex.

8.1.1.7 *Application Entity Title and Presentation Address*

These parameters shall be supplied by the initiator and the responder of the association (Application Entity Title is optionally supplied). For a DUA establishing an association for an initial request, these parameters are obtained from locally held information.

For a DUA (or DSA) establishing an association with a DSA to which it has been referred, these parameters are obtained from the **AccessPoint** value of a **ContinuationReference**. For a DSA establishing an association, this parameter is obtained from its Knowledge Information, i.e. an external reference.

8.1.2 *Abstract-unbind onto A-RELEASE*

The abstract-unbind service is mapped onto the A-RELEASE service of the ACSE. The use of the parameters of the A-RELEASE service is qualified in the following subparagraph.

8.1.2.1 *Result*

This parameter shall have the value "affirmative".

8.1.3 *Use of A-ABORT and A-P-ABORT services*

The application-process is the user of the A-ABORT and A-P-ABORT services of the ACSE.

8.2 *Mapping onto ROSE*

The Directory ASE services are mapped onto the RO-INVOKE, RO-RESULT, RO-ERROR, RO-REJECT-U and RO-REJECT-P services of the ROSE. The mapping of the abstract-syntax notation of the Directory ASEs onto the ROSE services is as defined in Recommendation X.219.

9 **Conformance**

This paragraph defines the requirements for conformance to this Recommendation.

9.1 *Conformance by DUAs*

A DUA implementation claiming conformance to this Recommendation shall satisfy the requirements specified in §§ 9.1.1 to 9.1.3.

9.1.1 *Statement requirements*

The following shall be stated:

- a) the operations of the **directoryAccessAC** application-context that the DUA is capable of invoking for which conformance is claimed; and
- b) the security-level(s) for which conformance is claimed (none, simple, strong).

9.1.2 *Static requirements*

A DUA shall:

- a) have the capability of supporting the **directoryAccessAC** application-context as defined by its abstract syntax in § 7.

9.1.3 *Dynamic requirements*

A DUA shall:

- a) conform to the mapping onto used services defined in § 8.

9.2 *Conformance by DSAs*

A DSA implementation claiming conformance to this Recommendation shall satisfy the requirements specified in §§ 9.2.1 to 9.2.3.

9.2.1 *Statement requirements*

The following shall be stated:

- a) the application-contexts for which conformance is claimed: **directoryAccessAC**, **directorySystemAC**, or both. If a DSA is such that knowledge of it has been disseminated causing knowledge references to the DSA to be held by other DSA(s) outside of its own DMD, then it shall claim conformance to the **directorySystemAC**;

Note - An application context shall not be divided, except as stated herein: in particular, conformance may not be claimed to particular ports or operations.

- b) whether or not the DSA is capable of acting as a first-level DSA, as defined in Recommendation X.518;
- c) if conformance is claimed to the **directorySystemAC** application-context, whether or not the chained mode of operation is supported, as defined in Recommendation X.518;
- d) the security-level(s) for which conformance is claimed (none, simple, strong);
- e) the selected attribute types defined in Recommendation X.520 and any other attribute types, for which conformance is claimed; and
- f) the selected object classes defined in Recommendation X.521 and any other object classes, for which conformance is claimed.

9.2.2 *Static requirements*

A DSA shall:

- a) have the capability of supporting the application-contexts for which conformance is claimed as defined by their abstract syntax in § 7;
- b) have the capability of supporting the information framework defined by its abstract syntax in Recommendation X.501;
- c) conform to the minimal knowledge requirements defined in Recommendation X.518;
- d) if conformance is claimed as a first-level DSA, conform to the requirements for support of the root context, as defined in Recommendation X.518;
- e) have the capability of supporting the attribute types for which conformance is claimed as defined by their abstract syntaxes; and
- f) have the capability of supporting the object classes for which conformance is claimed, as defined by their abstract syntaxes.

9.2.3 Dynamic requirements

A DSA shall:

- a) conform to the mapping onto used services defined in § 8 of this Recommendation;
- b) conform to the procedures for distributed operation of the Directory related to referrals, as defined in Recommendation X.518;
- c) if conformance is claimed to the **directoryAccessAC** application-context, conform to the procedures of Recommendation X.518 as they relate to the referral mode of the DAP;
- d) if conformance is claimed to the **directorySystemAC** application-context, conform to the referral mode of interaction, as defined in Recommendation X.518;
- e) if conformance is claimed to the chained mode of interaction, conform to the chained mode of interaction, as defined in Recommendation X.518.

Note - Only in this case is it necessary for a DSA to be capable of invoking operations using the **directorySystemAC**.

ANNEX A

(to Recommendation X.519)

DAP in ASN.1

This Annex is part of the Recommendation.

This Annex includes all of the ASN.1 type and value definitions contained in this Recommendation in the form of the ASN.1 module, **DirectoryAccessProtocol**.

DirectoryAccessProtocol {joint-iso-ccitt ds(5) modules(1) dap(11)}

DEFINITIONS ::=

BEGIN

EXPORTS

directoryAccessAC, readASE, searchASE, modifyASE;

IMPORTS

abstractService

FROM UsefulDefinitions

{joint-iso-ccitt ds(5) modules(1) usefulDefinitions(0)}

APPLICATION-SERVICE-ELEMENT, APPLICATION-CONTEXT, aCSE

FROM Remote-Operations-Notation-extension

{joint-iso-ccitt remoteOperations(4) notation-extension(2)}

id-ac-directoryAccessAC, id-ase-readASE, id-ase-searchASE,

id-ase-modifyASE, id-as-directoryAccessAS, id-as-acse

FROM ProtocolObjectIdentifiers

{joint-iso-ccitt ds(5) modules(1)

protocolObjectIdentifiers(4)}

DirectoryBind, DirectoryUnbind, Read, Compare, Abandon, List,

Search, AddEntry, RemoveEntry, ModifyEntry, ModifyRDN, Abandoned, AbandonFailed,

AttributeError, NameError, Referral, SecurityError, ServiceError,

UpdateError

FROM DirectoryAbstractService

directoryAbstractService;

-- Application Contexts --

directoryAccessAC

APPLICATION-CONTEXT

APPLICATION SERVICE ELEMENTS {aCSE}

BIND DirectoryBind

```

UNBIND DirectoryUnbind
  REMOTE OPERATIONS {rOSE}
    INITIATOR CONSUMER OF {readASE, searchASE, modifyASE}
  ABSTRACT SYNTAXES {
    id-as-acse, id-as-directoryAccessAS}
  ::= id-ac-directoryAccessAC

-- Read ASE --

readASE
  APPLICATION-SERVICE-ELEMENT
  CONSUMER INVOKES {read, compare, abandon}
  ::= id-ase-readASE

-- Search ASE --

searchASE
  APPLICATION-SERVICE-ELEMENT
  CONSUMER INVOKES {list, search}
  ::= id-ase-searchASE

-- Modify ASE --

modifyASE
  APPLICATION-SERVICE-ELEMENT
  CONSUMER INVOKES
    {addEntry, removeEntry,
     modifyEntry, modifyRDN}
  ::= id-ase-modifyASE

-- Remote Operations --

read          Read          ::= 1
compare       Compare       ::= 2
abandon       Abandon       ::= 3
list          List          ::= 4
search        Search        ::= 5
addEntry      AddEntry      ::= 6
removeEntry   RemoveEntry   ::= 7
modifyEntry   ModifyEntry   ::= 8
modifyRDN     ModifyRDN     ::= 9

-- Remote Errors --

attributeError  AttributeError  ::= 1
nameError       NameError       ::= 2
serviceError    ServiceError    ::= 3
referral        Referral        ::= 4
abandoned       Abandoned       ::= 5
securityError   SecurityError   ::= 6
abandonFailed   AbandonFailed   ::= 7
updateError     UpdateError     ::= 8
END

```

ANNEX B

(to Recommendation X.519)

DSP in ASN.1

This Annex is part of the Recommendation.

This Annex includes all of the ASN.1 type and value definitions contained in this Recommendation in the form of the ASN.1 module, **DirectorySystemProtocol**.

```
DirectorySystemProtocol {joint-iso-ccitt ds(5) modules(1) dsp(12)}
DEFINITIONS ::=
BEGIN
EXPORTS
    directorySystemAC, chainedReadASE, chainedSearchASE, chainedModifyASE;
IMPORTS
    distributedOperations, directoryAbstractService
    FROM      UsefulDefinitions
              {joint-iso-ccitt ds(5) modules(1) usefulDefinitions(0)}

    APPLICATION-SERVICE-ELEMENT, APPLICATION-CONTEXT, aCSE
    FROM      Remote-Operations-Notation-extension
              {joint-iso-ccitt remoteOperations(4) notation-extension(2)}

    id-ac-directorySystemAC, id-ase-chainedReadASE,
    id-ase-chainedSearchASE, id-ase-chainedModifyASE,
    id-as-directorySystemAS, id-as-acse;
    FROM      ProtocolObjectIdentifiers
              {joint-iso-ccitt ds(5) modules(1)
               protocolObjectIdentifiers(4)}

    Abandoned, AttributeError, AbandonFailed,
    NameError, DSAReferral, SecurityError, ServiceError, UpdateError
    FROM      DirectoryAbstractService directoryAbstractService

    DSABind, DSAUnbind,
    ChainedRead, ChainedCompare, ChainedAbandon,
    ChainedList, ChainedSearch,
    ChainedAddEntry, ChainedRemoveEntry, ChainedModifyEntry,
    ChainedModifyRDN, DSAReferral,
    FROM      DistributedOperations
              distributedOperations;

-- Application Contexts --
directorySystemAC
    APPLICATION-CONTEXT
        APPLICATION SERVICE ELEMENTS {aCSE}
        BIND DSABind
        UNBIND DSAUnbind
        REMOTE OPERATIONS {rOSE}
        OPERATIONS OF {
            chainedReadASE, chainedSearchASE, chainedModifyASE}
        ABSTRACT SYNTAXES {
            id-as-acse, id-as-directorySystemAS}
    ::= {id-ac-directorySystemAC}

-- Chained Read ASE --
chainedReadASE
    APPLICATION-SERVICE-ELEMENT
        OPERATIONS {chainedRead, chainedCompare, chainedAbandon}
    ::= id-ase-chainedReadASE
```

```

-- Chained Search ASE --
chainedSearchASE
  APPLICATION-SERVICE-ELEMENT
    OPERATIONS {chainedList, chainedSearch}
    ::= id-ase-chainedSearchASE

-- Chained Modify ASE --
chainedModifyASE
  APPLICATION-SERVICE-ELEMENT
    OPERATIONS
      {chainedAddEntry, chainedRemoveEntry,
       chainedModifyEntry, chainedModifyRDN}
    ::= id-ase-chainedModifyASE

-- Remote Operations --
chainedRead          ChainedRead          ::= 1
chainedCompare      ChainedCompare        ::= 2
chainedAbandon      ChainedAbandon        ::= 3
chainedlist         ChainedList          ::= 4
chainedSearch       ChainedSearch        ::= 5
chainedAddEntry     ChainedAddEntry       ::= 6
chainedRemoveEntry  ChainedRemoveEntry    ::= 7
chainedModifyEntry  ChainedModifyEntry    ::= 8
chainedModifyRDN    ChainedModifyRDN     ::= 9

-- Remote Errors --
attributeError      AttributeError       ::= 1
nameError           NameError           ::= 2
serviceError        ServiceError        ::= 3
abandoned          Abandoned           ::= 5
securityError       SecurityError       ::= 6
abandonFailed       AbandonFailed       ::= 7
updateError         UpdateError         ::= 8
dsaReferral         DSAReferral        ::= 9

END

```

ANNEX C

(to Recommendation X.519)

Reference definition of protocol object identifiers

This Annex is part of the Recommendation.

This Annex includes all of the ASN.1 Object Identifiers assigned in this Recommendation in the form of ASN.1 module, **ProtocolObjectIdentifiers**.

```

ProtocolObjectIdentifiers {joint-iso-ccitt ds(5) modules(1) protocolObjectIdentifiers(4)}
DEFINITIONS ::=
BEGIN

```

EXPORTS

id-ac-directoryAccessAC, id-ac-directorySystemAC, id-ase-readASE, id-ase-searchASE,
id-ase-modifyASE, id-ase-chainedReadASE,
id-ase-chainedSearchASE, id-ase-chainedModifyASE, id-as-acse,
id-as-directoryAccessAS, id-as-directorySystemsAS;

IMPORTS

id-ac, id-ase, id-as
FROM UsefulDefinitions
{joint-iso-ccitt ds(5) modules(1) usefulDefinitions(0)};

-- Application Contexts --

id-ac-directoryAccessAC OBJECT IDENTIFIER ::= {id-ac 1}

id-ac-directorySystemAC OBJECT IDENTIFIER ::= {id-ac 2}

-- ASEs --

id-ase-readASE OBJECT IDENTIFIER ::= {id-ase 1}

id-ase-searchASE OBJECT IDENTIFIER ::= {id-ase 2}

id-ase-modifyASE OBJECT IDENTIFIER ::= {id-ase 3}

id-ase-chainedReadASE OBJECT IDENTIFIER ::= {id-ase 4}

id-ase-chainedSearchASE OBJECT IDENTIFIER ::= {id-ase 5}

id-ase-chainedModifyASE OBJECT IDENTIFIER ::= {id-ase 6}

-- ASs --

id-as-directoryAccessAS OBJECT IDENTIFIER ::= {id-as 1}

id-as-directorySystemAS OBJECT IDENTIFIER ::= {id-as 2}

id-as-acse OBJECT IDENTIFIER ::= {joint-iso-ccitt association-control(2) abstract-syntax(1) apdus(0) version1(1)}

END

Recommendation X.520

THE DIRECTORY - SELECTED ATTRIBUTE TYPES ¹⁾

(Melbourne, 1988)

CONTENTS

- 0 Introduction
- 1 Scope and field of application
- 2 References
- 3 Definitions
- 4 Notation

¹⁾ Recommendation X.520 and ISO 9594-6, Information Processing Systems - Open Systems Interconnection - The Directory - Selected attribute types, were developed in close collaboration and are technically aligned.

SECTION 1 - *Selected Attribute Types*

5 *Definition of Selected Attribute Types*

- 5.1 System Attribute Types
- 5.2 Labelling Attribute Types
- 5.3 Geographical Attribute Types
- 5.4 Organizational Attribute Types
- 5.5 Explanatory Attribute Types
- 5.6 Postal Addressing Attribute Types
- 5.7 Telecommunications Addressing Attribute Types
- 5.8 Preferences Attribute Types
- 5.9 OSI Application Attribute Types
- 5.10 Relational Attribute Types
- 5.11 Security Attribute Types

SECTION 2 - *Attribute Syntaxes*

6 *Definition of Attribute Syntaxes*

- 6.1 Attribute Syntaxes Used by the Directory
- 6.2 String Attribute Syntaxes
- 6.3 Miscellaneous Attribute Syntaxes

Annex A - Selected Attribute Types in ASN.1

Annex B - Index of Attribute Types and Syntaxes

Annex C - Upper Bounds

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. The set of all 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 defines a number of attribute types which may be found useful across a range of applications of the Directory. One particular use for many of the attributes defined herein is in the formation of names, particularly for the classes of object defined in Recommendation X.521. This Recommendation also defines a number of standard attribute syntaxes.

0.4 Annex A, which is part of this Recommendation, provides the ASN.1 notation for the complete module which defines the attributes and attribute syntaxes.

0.5 Annex B, which is not part of this Recommendation, provides an alphabetical index of attribute types, for easy reference.

1 Scope and field of application

1.1 This Recommendation defines a number of attribute types which may be found useful across a range of applications of the Directory.

1.2 Attribute types (and attribute syntaxes) fall into three categories, as described in §§ 1.2.1 to 1.2.3.

1.2.1 Some attribute types (syntaxes) are used by a wide variety of applications or are understood and/or used by the Directory itself.

Note - It is recommended that an attribute type (syntax) defined in this document be used, in preference to the generation of a new one, whenever it is appropriate for the application.

1.2.2 Some attribute types (syntaxes) are internationally-standardized, but are application-specific. These are defined in the standards associated with the application concerned.

1.2.3 Any administrative authority can define its own attribute types (syntaxes) for any purpose. These are not internationally standardized, and are available to others beyond the administrative authority which created them only by bilateral agreement.

2 References

ISO 3166 - Codes for the representation of names of countries

Recommendation X.121 - International numbering plan for public data networks

Recommendation X.208 - Open Systems Interconnection - Specification of Abstract Syntax Notation (ASN.1) (see also ISO 8824)

Recommendation X.501 - The Directory - Models (see also ISO 9594-2)

Recommendation X.521 - The Directory - Selected Object Classes (see also ISO 9594-7)

Recommendation E.123 - Notation for National and International Telephone Numbers

3 Definitions

This Recommendation makes use of the following definitions from Recommendation X.501:

- a) *attribute type*;
- b) *attribute syntax*;
- c) *object class*.

4 Notation

Attribute types and attribute syntaxes are defined in this document by the use of special notation, defined as ASN.1 macros in Recommendation X.501. There are two such macros, **ATTRIBUTE** and **ATTRIBUTE-SYNTAX**.

Two "generic" object identifiers (**attributeType** and **attributeSyntax**) are used in defining the object identifiers being allocated to attribute types and attribute syntaxes respectively. Their definitions can be found in Annex B of Recommendation X.501.

Examples of the use of the attribute types are described using an informal notation, where attribute type and value pairs are represented by an acronym for the attribute type, followed by an equals sign ("="), followed by the example value for the attribute.

SECTION 1 - *Selected Attribute Types*

5 Definition of Selected Attribute Types

This Recommendation defines a number of attribute types which may be found useful across a range of applications of the Directory.

5.1 *System Attribute Types*

These attribute types are concerned with information about objects known to the Directory.

5.1.1 *Object Class*

The *Object Class* attribute type, which is known to the Directory, is specified, except for the allocation of an object identifier, in Recommendation X.501.

objectClass ObjectClass ::= {attributeType 0}

5.1.2 *Aliased Object Name*

This attribute type is defined, except for the allocation of an object identifier, in Recommendation X.501.

aliasedObjectName AliasedObjectName ::= {attributeType 1}

5.1.3 *Knowledge information*

The *Knowledge Information* attribute type specifies a human readable accumulated description of knowledge mastered by a specific DSA.

**knowledgeInformation ATTRIBUTE
WITH ATTRIBUTE-SYNTAX caseIgnoreStringSyntax
::= {attributeType 2}**

5.2 *Labelling Attribute Types*

These attribute types are concerned with information about objects which has been explicitly associated with the objects by a labelling process.

5.2.1 *Common Name*

The *Common Name* attribute type specifies an identifier of an object. A Common Name is not a directory name; it is a (possibly ambiguous) name by which the object is commonly known in some limited scope (such as an organization) and conforms to the naming conventions of the country or culture with which it is associated.

An attribute value for common name is a string chosen either by the person or organization it describes or the organization responsible for the object it describes for devices and application entities. For example, a typical name of a person in an English-speaking country comprises a personal title (e.g. Mr, Ms, Dr, Professor, Sir, Lord), a first name, middle name(s), last name, generational qualifier (if any, e.g. Jr.) and decorations and awards (if any, e.g. QC).

Examples:

CN = "Mr. Robin Lachlan McLeod BSc(Hons) CEng MIEE"

CN = "Divisional Coordination Committee"

CN = "High Speed Modem".

Any variants should be associated with the named object as separate and alternative attribute values.

Other common variants should also be admitted, e.g. use of a middle name as a preferred first name; use of "Bill" in place of "William", etc.

**commonName ATTRIBUTE
WITH ATTRIBUTE-SYNTAX
caseIgnoreStringSyntax
(SIZE(1..ub-common-name))
::= {attributeType 3}**

5.2.2 *Surname*

The *Surname* attribute type specifies the linguistic construct which normally is inherited by an individual from the individual's parent or assumed by marriage, and by which the individual is commonly known.

An attribute value for Surname is a string, e.g. "McLeod".

```
surname ATTRIBUTE  
WITH ATTRIBUTE-SYNTAX  
  caseIgnoreStringSyntax  
    (SIZE(1..ub-surname))  
::= {attributeType 4}
```

5.2.3 *Serial Number*

The *Serial Number* attribute type specifies an identifier, the serial number of a device.

An attribute value for Serial Number is a printable string.

```
serialNumber ATTRIBUTE  
WITH ATTRIBUTE-SYNTAX  
  printableStringSyntax  
    (SIZE(1..ub-serial-number))  
::= {attributeType 5}
```

5.3 *Geographical Attribute Types*

These attribute types are concerned with geographical positions or regions with which objects are associated.

5.3.1 *Country Name*

The *Country Name* attribute type specifies a country. When used as a component of a directory name, it identifies the country in which the named object is physically located or with which it is associated in some other important way.

An attribute value for country name is a string chosen from ISO 3166.

```
countryName ATTRIBUTE  
WITH ATTRIBUTE-SYNTAX  
  PrintableString (SIZE(2)) - IS 3166 codes only  
MATCHES FOR EQUALITY  
SINGLE VALUE  
::= {attributeType 6}
```

The matching rule for values of this type is the same as that for **caseIgnoreStringSyntax**.

5.3.2 *Locality Name*

The *Locality Name* attribute type specifies a locality. When used as a component of a directory name, it identifies a geographical area or locality in which the named object is physically located or with which it is associated in some other important way.

An attribute value for Locality Name is a string, e.g. L = "Edinburgh".

```
localityName ATTRIBUTE  
WITH ATTRIBUTE-SYNTAX  
  caseIgnoreStringSyntax  
    (SIZE(1..ub-locality-name))  
::= {attributeType 7}
```

5.3.3 *State or Province Name*

The *State or Province Name* attribute type specifies a state or province. When used as a component of a directory name, it identifies a geographical subdivision in which the named object is physically located or with which it is associated in some other important way.

An attribute value for State or Province Name is a string, e.g. S = "Ohio".

```
stateOrProvinceName ATTRIBUTE  
WITH ATTRIBUTE-SYNTAX  
  caseIgnoreStringSyntax  
    (SIZE(1..ub-state-name))  
::= {attributeType 8}
```

5.3.4 *Street Address*

The *Street Address* attribute type specifies a site for the local distribution and physical delivery in a postal address, i.e. the street name, place, avenue, and the house number. When used as a component of a directory name, it identifies the street address at which the named object is located or with which it is associated in some other important way.

An attribute value for Street Address is a string, e.g. "Arnulfstraße 60".

streetAddress ATTRIBUTE
WITH ATTRIBUTE-SYNTAX
 caseIgnoreStringSyntax
 (SIZE(1..ub-street-address))
::= {attributeType 9}

5.4 *Organizational Attribute Types*

These attribute types are concerned with organizations and can be used to describe objects in terms of organizations with which they are associated.

5.4.1 *OrganizationName*

The *Organization Name* attribute type specifies an organization. When used as a component of a directory name it identifies an organization with which the named object is affiliated.

An attribute value for OrganizationName is a string chosen by the organization (e.g. O = "Scottish Telecommunications plc"). Any variants should be associated with the named Organization as separate and alternative attribute values.

organizationName ATTRIBUTE
WITH ATTRIBUTE-SYNTAX
 caseIgnoreStringSyntax
 (SIZE(1..ub-organization-name))
::= {attributeType 10}

5.4.2 *Organizational Unit Name*

The *Organizational Unit Name* attribute type specifies an organizational unit. When used as a component of a directory name it identifies an organizational unit with which the named object is affiliated.

The designated organizational unit is understood to be part of an organization designated by an OrganizationName attribute.

It follows that if an Organizational Unit Name attribute is used in a directory name, it must be associated with an OrganizationName attribute.

An attribute value for Organizational Unit Name is a string chosen by the organization of which it is part (e.g. OU = "Technology Division"). Note that the commonly used abbreviation "TD" would be a separate and alternative attribute value.

Examples:

O = "Scottel", OU = "TD"

organizationalUnitName ATTRIBUTE
WITH ATTRIBUTE-SYNTAX
 caseIgnoreStringSyntax
 (SIZE(1..ub-organizational-unit-name))
::= {attributeType 11}

5.4.3 *Title*

The *Title* attribute type specifies the designated position or function of the object within an organization.

An attribute value for Title is a string.

Example:

T = "Manager,Distributed Applications"

title ATTRIBUTE
WITH ATTRIBUTE-SYNTAX
caseIgnoreStringSyntax
(SIZE(1..ub-title))
::= {attributeType 12}

5.5 Explanatory Attribute Types

These attribute types are concerned with explanations (e.g. in a natural language) of something about an object.

5.5.1 Description

The *Description* attribute type specifies text which describes the associated object.

For example, the object "Standards Interest" might have the associated description "distribution list for exchange of information about intra-company standards development".

An attribute value for Description is a string.

description ATTRIBUTE
WITH ATTRIBUTE-SYNTAX
caseIgnoreStringSyntax
(SIZE(1..ub-description))
::= {attributeType 13}

5.5.2 Search Guide

The *Search Guide* attribute type specifies information of suggested search criteria which may be included in some entries expected to be a convenient base-object for the search operation, e.g. country or organization.

Search criteria consists of an optional identifier for the class of object sought and combinations of attribute types and logical operators to be used in the construction of a filter. It is possible to specify for each search criteria item the matching level, e.g. approximate match.

The Search Guide attribute may recur to reflect the various types of requests, e.g. search for a Residential Person or an Organizational Person, which may be fulfilled from the given base-object where the Search Guide is read.

searchGuide ATTRIBUTE
WITH ATTRIBUTE-SYNTAX
Guide
::= {attributeType 14}

Guide ::= SET {
objectClass [0] OBJECT-CLASS OPTIONAL,
criteria [1] Criteria}

Criteria ::=
CHOICE{
Type [0] CriteriaItem,
and [1] SET OF Criteria,
or [2] SET OF Criteria,
not [3] Criteria

CriteriaItem
CHOICE {
equality [0] AttributeType,
substrings [1] AttributeType,
greaterOrEqual [2] AttributeType,
lessOrEqual [3] AttributeType,
approximateMatch [4] AttributeType}

Example: The following is a potential value of the Search Guide attribute that could be stored in entries of object-class Locality to indicate how entries of object-class Residential Person might be found.

```
residential-person-guide Guide ::= {
  objectClass residentialPerson,
  criteria and {
    type substrings commonName,
    type substrings streetAddress }}
```

The construction of a Filter from this value of Guide is straightforward.

Step (1) produces the intermediate Filter value:

```
intermediate-filter Filter ::= and {
  item substrings {
    type commonName,
    strings {any T61String "Dubois" }}, - value supplied for Common Name

  item substrings {
    type streetAddress
    strings {any T61String "Hugo" }}} - value supplied for Street Address
```

Step (2) produces a filter for matching Residential Person entries in the subtree:

```
residential-person-filter Filter ::= {
  and {
    item equality {
      objectClass,
      OBJECT-CLASS residentialPerson },
    intermediate-filter }}
```

5.5.3 Business Category

The *Business Category* attribute type specifies information concerning the occupation of some common objects, e.g. people. For example, this attribute provides the facility to interrogate the Directory about people sharing the same occupation.

```
businessCategory ATTRIBUTE
  WITH ATTRIBUTE-SYNTAX
    caseIgnoreStringSyntax
    (SIZE(1..ub-business-category))
  ::= {attributeType 15})
```

5.6 Postal Addressing Attribute Types

These attribute types are concerned with information required for physical postal delivery to an object.

5.6.1 Postal address

The *Postal Address* attribute type specifies the address information required for the physical delivery of postal messages by the postal authority to the named object.

An attribute value for Postal Address will be typically composed of selected attributes from MHS Unformatted Postal O/R Address version 1 according to Recommendation F.401 and limited to 6 lines of 30 characters each, including a Postal Country Name. Normally the information contained in such an address could include an addressee's name, street address, city, state or province, postal code and possibly a Post Office Box number depending on the specific requirements of the named object.

```
postalAddress ATTRIBUTE
  WITH ATTRIBUTE-SYNTAX PostalAddress
  MATCHES FOR EQUALITY
  ::= {attributeType 16}
```

```
PostalAddress ::= SEQUENCE SIZE(1..ub-postal-line) OF CHOICE {
  T61String (SIZE(1..ub-postal-string)),
  PrintableString (SIZE(1..ub-postal-string))}
```

The matching rule for values of this type is the same as that for *caseIgnoreListSyntax*.

5.6.2 *Postal Code*

The *Postal Code* attribute type specifies the postal code of the named object. If this attribute value is present it will be part of the object's postal address.

An attribute value for Postal Code is a string.

```
postalCode ATTRIBUTE
  WITH ATTRIBUTE-SYNTAX
    caseIgnoreStringSyntax
      (SIZE(1..ub-postal-code))
  ::= {attributeType 17}
```

5.6.3 *Post Office Box*

The *Post Office Box* attribute type specifies the Post Office Box by which the object will receive physical postal delivery. If present, the attribute value is part of the object's postal address.

```
postOfficeBox ATTRIBUTE
  WITH ATTRIBUTE-SYNTAX
    caseIgnoreStringSyntax
      (SIZE(1..ub-post-office-box))
  ::= {attributeType 18}
```

5.6.4 *Physical Delivery Office Name*

The *Physical Delivery Office Name* attribute type specifies the name of the city, village, etc. where a physical delivery office is situated.

An attribute value for Physical Delivery Office Name is a string.

```
physicalDeliveryOfficeName ATTRIBUTE
  WITH ATTRIBUTE-SYNTAX
    caseIgnoreStringSyntax
      (SIZE(1..ub-physical-office-name))
  ::= {attributeType 19}
```

5.7 *Telecommunications Addressing Attribute Types*

These attribute types are concerned with addressing information needed to communicate with the object using telecommunication means.

5.7.1 *Telephone Number*

The *Telephone Number* attribute type specifies a telephone number associated with an object.

An attribute value for Telephone Number is a string that complies with the internationally agreed format for showing international telephone numbers. Recommendation E.123 (e.g. "+44 582 10101").

```
telephoneNumber ATTRIBUTE
  WITH ATTRIBUTE-SYNTAX
    telephoneNumberSyntax
  ::= {attributeType 20}
```

5.7.2 *Telex Number*

The *Telex Number* attribute type specifies the telex number, country code, and answerback code of a telex terminal associated with an object.

```
telexNumber ATTRIBUTE
  WITH ATTRIBUTE-SYNTAX TelexNumber
  ::= {attributeType 21}
```

```

TelexNumber ::= SEQUENCE{
    telexNumber PrintableString,
    (SIZE(1..ub-telex-number)),
    countryCode PrintableString,
    (SIZE(1..ub-country-code)),
    answerback PrintableString,
    (SIZE(1..ub-answerback)))

```

5.7.3 Teletex Terminal Identifier

The *Teletex Terminal Identifier* attribute type specifies the Teletex terminal identifier (and optionally parameters) for a teletex terminal associated with an object.

An attribute value for Teletex Terminal Identifier is a string which complies with CCITT Recommendation F.200 and an optional set whose components are according to Recommendation T.62.

```

teletexTerminalIdentifier ATTRIBUTE
WITH ATTRIBUTE-SYNTAX
    TeletexTerminalIdentifier
::= {attributeType 22}

```

```

TeletexTerminalIdentifier ::= SEQUENCE {
    teletexTerminal PrintableString
    (SIZE(1..ub-teletex-terminal-id)),
    parameters TeletexNonBasicParameters
    OPTIONAL}

```

5.7.4 Facsimile Telephone Number

The *Facsimile Telephone Number* attribute type specifies a telephone number for a facsimile terminal (and optionally its parameters) associated with an object.

An attribute value for the facsimile telephone number is a string that complies with the internationally agreed format for showing international telephone numbers, Recommendation E.1xx (e.g. "+81 3 347 7418") and an optional bit string (formatted according to Recommendation T.30).

```

facsimileTelephoneNumber ATTRIBUTE
WITH ATTRIBUTE-SYNTAX
    Facsimile TelephoneNumber
::= {attributeType 23}

```

```

FacsimileTelephoneNumber ::= SEQUENCE{
    telephoneNumber PrintableString
    (SIZE(1..ub-telephone-number)),
    parameters G3FacsimileNonBasicParameters
    OPTIONAL}

```

5.7.5 X.121 Address

The *X.121 Address* attribute type specifies an address as defined by CCITT Recommendation X.121 associated with an object.

```

x121Address ATTRIBUTE
WITH ATTRIBUTE-SYNTAX
    NumericString
    (SIZE(1..ub-x121-address))
MATCHES FOR EQUALITY SUBSTRINGS
::= {attributeType 24}

```

The matching rules for values of this type are the same as those for **numericStringSyntax**.

5.7.6 International ISDN Number

The *International ISDN Number* attribute type specifies an International ISDN Number associated with an object.

An attribute value for International ISDN Number is a string which complies with the internationally agreed format for ISDN addresses given in CCITT Recommendation E.164.

internationalISDNNumber ATTRIBUTE
WITH ATTRIBUTE-SYNTAX
NumericString
(SIZE(1..ub-isdn-address))
::= {attributeType 25}

The matching rule for values of this type is the same as that for **numericStringSyntax**.

5.7.7 Registered Address

The *Registered Address* attribute type specifies a mnemonic for an address associated with an object at a particular city location. The mnemonic is registered in the country in which the city is located and is used in the provision of the Public Telegram Service (according to Recommendation F.1).

registeredAddress ATTRIBUTE
WITH ATTRIBUTE-SYNTAX PostalAddress
::= {attributeType 26}.

5.7.8 Destination Indicator

The *Destination Indicator* attribute type specifies (according to Recommendations F.1 and F.3) the country and city associated with the object (the addressee) needed to provide the Public Telegram Service.

An attribute value for Destination Indicator is a string.

destinationIndicator ATTRIBUTE
WITH ATTRIBUTE-SYNTAX
PrintableString
(SIZE(1..ub-destination-indicator))
- alphabetical characters only
MATCHES FOR EQUALITY SUBSTRINGS
::= {attributeType 27}

The matching rules for values of this type are the same as those for **caseIgnoreStringSyntax**.

5.8 Preference Attribute Types

These attribute types are concerned with the preferences of an object.

5.8.1 Preferred Delivery Method

The *Preferred Delivery Method* attribute type specifies the object's priority order regarding the method to be used for communicating with it.

preferredDeliveryMethod ATTRIBUTE
WITH ATTRIBUTE-SYNTAX
SEQUENCE OF INTEGER {
 any-delivery-method (0),
 mhs-delivery (1),
 physical-delivery (2),
 telex-delivery (3),
 teletex-delivery (4),
 g3-facsimile-delivery (5),
 g4-facsimile-delivery (6),
 ia5-terminal-delivery (7),
 videotex-delivery (8),
 telephone-delivery (9)}
SINGLE VALUE
::= {attributeType 28}

5.9 OSI Application Attribute Types

These attribute types are concerned with information regarding objects in the OSI Application Layer.

5.9.1 Presentation Address

The *Presentation Address* attribute type specifies a presentation address associated with an object representing an OSI application entity.

An attribute value for Presentation Address is a presentation address as defined in Recommendation X.200.

**presentationAddress ATTRIBUTE
WITH ATTRIBUTE-SYNTAX
PresentationAddress
MATCHES FOR EQUALITY
SINGLE VALUE
::= {attributeType 29}**

PresentationAddress	::=	SEQUENCE
pSelector	[0]	OCTET STRING OPTIONAL,
sSelector	[1]	OCTET STRING OPTIONAL,
tSelector	[2]	OCTET STRING OPTIONAL,
nAddresses	[3]	SET SIZE(1..MAX) OF OCTET STRING)

The matching rule for values of this type is that a presented Presentation Address matches a stored one if and only if the selectors are equal and the presented nAddresses are a subset of the stored ones.

5.9.2 Supported Application Context

The *Supported Application Context* attribute type specifies the object identifier(s) of application context(s) that the object (an OSI application entity) supports.

**supportedApplicationContext ATTRIBUTE
WITH ATTRIBUTE-SYNTAX
objectIdentifierSyntax
::= {attributeType 30}**

5.10 Relational Attribute Types

These attribute types are concerned with information regarding the objects which are related to a particular object in certain ways.

5.10.1 Member

The *Member* attribute type specifies a group of names associated with the object.

An attribute value for Member is a distinguished name.

**member ATTRIBUTE
WITH ATTRIBUTE-SYNTAX
distinguishedNameSyntax
::= {attributeType 31}**

5.10.2 Owner

The *Owner* attribute type specifies the name of some object which has some responsibility for the associated object.

An attribute value for Owner is a distinguished name (which could represent a group of names) and can recur.

**owner ATTRIBUTE
WITH ATTRIBUTE-SYNTAX
distinguishedNameSyntax
::= {attributeType 32}**

5.10.3 *Role Occupant*

The *Role Occupant* attribute type specifies the name of an object which fulfills an organizational role.

An attribute value for Role Occupant is a distinguished name.

```
roleOccupant ATTRIBUTE
  WITH ATTRIBUTE-SYNTAX
    distinguishedNameSyntax
  ::= {attributeType 33}
```

5.10.4 *See Also*

The *See Also* attribute type specifies names of other Directory objects which may be other aspects (in some sense) of the same real world object.

An attribute value for See Also is a distinguished name.

```
seeAlso ATTRIBUTE
  WITH ATTRIBUTE-SYNTAX
    distinguishedNameSyntax
  ::= {attributeType 34}
```

5.11 *Security Attribute Types*

These attribute types are concerned with the security or security privileges of an object. These attribute types are specified, except for the allocation of an object identifier, in Recommendation X.509.

5.11.1 *User Password*

```
userPassword UserPassword
  ::= {attributeType 35}
```

5.11.2 *User Certificate*

```
userCertificate UserCertificate
  ::= {attributeType 36}
```

5.11.3 *CA Certificate*

```
cACertificate CACertificate
  ::= {attributeType 37}
```

5.11.4 *Authority Revocation List*

```
authorityRevocationList AuthorityRevocationList
  ::= {attributeType 38}
```

5.11.5 *Certificate Revocation List*

```
certificateRevocationList CertificateRevocationList
  ::= {attributeType 39}
```

5.11.6 *Cross Certificate Pair*

```
crossCertificatePair CrossCertificatePair
  ::= {attributeType 40}
```

SECTION 2 - *Attribute Syntaxes*

6 *Definition of Attribute Syntaxes*

6.1 *Attribute Syntaxes used by the Directory*

6.1.1 *Undefined*

The *Undefined* attribute syntax is intended for attributes whose values are not expected to be compared by the Directory.

Specifying this attribute syntax for an attribute is equivalent to specifying the data type **ANY** and no matching rules in the **ATTRIBUTE** macro for the attribute.

```
undefined ATTRIBUTE-SYNTAX
  ANY
  ::= {attributeSyntax 0}
```

6.1.2 *Distinguished Name*

The *Distinguished Name* attribute syntax is intended for attributes whose values are distinguished names. It is defined, except for the allocation of an object identifier, in Recommendation X.501.

```
distinguishedNameSyntax DistinguishedNameSyntax
  ::= {attributeSyntax 1}
```

6.1.3 *Object Identifier*

The *Object Identifier* attribute syntax is intended for attributes whose values are object identifiers. It is defined, except for the allocation of an object identifier, in Recommendation X.501.

```
objectIdentifierSyntax ObjectIdentifierSyntax
  ::= {attributeSyntax 2}
```

6.2 *String Attribute Syntaxes*

In the syntaxes specified in §§ 6.2.1 to 6.2.4, the following spaces are regarded as not significant:

- leading spaces (i.e. those preceding the first printing character);
- trailing spaces (i.e. those following the last printing character);
- multiple consecutive internal spaces (these are taken as equivalent to a single space character).

Attributes conforming to these syntaxes shall be matched in a form which omits those spaces which are not significant according to these rules.

6.2.1 *Case Exact String*

The *Case Exact String* attribute syntax is intended for attributes whose values are strings (either T.61 Strings or Printable Strings), where the case (upper or lower) is significant for comparison purposes (e.g. "Dundee" and "DUNDEE" do not match).

```
caseExactString ATTRIBUTE-SYNTAX
  CHOICE {T61String, PrintableString}
  MATCHES FOR EQUALITY SUBSTRINGS
  ::= {attributeSyntax 3}
```

For two strings having this syntax to match for equality, the strings must be the same length and corresponding characters must be identical. A Printable String can be compared with a T.61 String: where the corresponding characters are both in the Printable String character set then comparison proceeds as normal. However if the character in the T.61 String is not in the Printable String character set then matching fails.

6.2.2 *Case Ignore String*

The *Case Ignore String* attribute syntax is intended for attributes whose values are strings (either T.61 Strings or Printable Strings), but where the case (upper or lower) is not significant for comparison purposes (e.g. "Dundee" and "DUNDEE" match).

caseIgnoreStringSyntax ATTRIBUTE-SYNTAX
CHOICE {T61String, PrintableString}
MATCHES FOR EQUALITY SUBSTRINGS
::= {attributeSyntax 4}

The rules for matching are identical to those for the Case Exact String attribute syntax, except that characters that differ only in their case are considered identical.

6.2.3 Printable String

The *Printable String* attribute syntax is intended for attributes whose values are Printable Strings.

printableStringSyntax ATTRIBUTE-SYNTAX
PrintableString
MATCHES FOR EQUALITY SUBSTRINGS
::= {attributeSyntax 5}

The rules for matching are identical to those for the Case Exact String attribute syntax.

6.2.4 Numeric String

The *Numeric String* attribute syntax is intended for attributes whose values are Numeric Strings.

numericStringSyntax ATTRIBUTE-SYNTAX
NumericString
MATCHES FOR EQUALITY SUBSTRINGS
::= {attributeSyntax 6}

The rules for matching are identical to those for the Case Exact String attribute syntax, except that all space characters are skipped during comparison.

6.2.5 Case Ignore List

The *Case Ignore List* attribute syntax is intended for attributes whose values are sequences of strings (either T.61 Strings or Printable Strings), but where the case (upper or lower) is not significant for comparison purposes.

caseIgnoreListSyntax ATTRIBUTE-SYNTAX
SEQUENCE OF
CHOICE {T61String, PrintableString}
MATCHES FOR EQUALITY SUBSTRINGS
::= {attributeSyntax 7}

Two Case Ignore Lists match for equality if and only if the number of strings in each is the same, and corresponding strings match. The latter matching is as for Case Ignore String attribute syntax (§ 6.1.3).

6.3 Miscellaneous Attribute Syntaxes

6.3.1 Boolean

The *Boolean* attribute syntax is intended for attributes whose values are Boolean (i.e. represent true or false).

booleanSyntax ATTRIBUTE-SYNTAX
BOOLEAN
MATCHES FOR EQUALITY
::= {attributeSyntax 8}

Two attribute values of this syntax match for equality if they are both true or both false.

6.3.2 Integer

The *Integer* attribute syntax is intended for attributes whose values are integers.

integerSyntax ATTRIBUTE-SYNTAX
INTEGER
MATCHES FOR EQUALITY ORDERING
::= {attributeSyntax 9}

Two attribute values of this syntax match for equality if the integers are the same. The ordering rules for integers apply.

6.3.3 Octet String

The *Octet String* attribute syntax is intended for attributes whose values are Octet Strings.

octetStringSyntax ATTRIBUTE-SYNTAX
OCTET STRING
MATCHES FOR EQUALITY SUBSTRINGS ORDERING
::= {attributeSyntax 10}

For two strings having this attribute syntax to match, the strings must be the same length and corresponding octets must be identical. Ordering is determined by the ordering relation between the first octets to differ on comparing the strings from the beginning.

6.3.4 UTC Time

The *UTC Time* attribute syntax is intended for attributes whose values represent absolute time.

uTCTimeSyntax ATTRIBUTE-SYNTAX
UTCTime
MATCHES FOR EQUALITY ORDERING
::= {attributeSyntax 11}

Two attribute values of this syntax match for equality if they represent the same time. An earlier time is considered "less" than a later time.

6.3.5 Telephone Number

The *Telephone Number* attribute syntax is intended for attributes whose values are telephone numbers.

telephoneNumberSyntax ATTRIBUTE-SYNTAX
PrintableString
(SIZE{1..ub-telephone-number})
MATCHES FOR EQUALITY SUBSTRINGS
::= {attributeSyntax 12}

The rules for matching are identical to those for the Case Exact attribute syntax, except that all space and "-" characters are skipped during the comparison.

ANNEX A

(to Recommendation X.520)

Selected Attribute Types in ASN.1

This Annex is part of the Recommendation.

This Annex includes all of the ASN.1 type and value definitions contained in this Recommendation in the form of the ASN.1 module **SelectedAttributeTypes**.

```
SelectedAttributeTypes      {joint-iso-ccitt ds(5) modules(1)
                             selectedAttributeTypes(5)}

DEFINITIONS ::=
BEGIN
-- Exports everything --
IMPORTS
    informationFramework, authenticationFramework, attributeType,
    upperBounds
    FROM UsefulDefinitions      {joint-ISO-CCITT ds(5) modules(1)
                                usefulDefinitions(0) },
    ATTRIBUTE, ATTRIBUTE-SYNTAX, AttributeType, OBJECT-CLASS,
    ObjectClass, AliasedObjectName,
    DistinguishedNameSyntax, ObjectIdentifierSyntax
    FROM InformationFramework informationFramework
    G3FacsimileNonBasicParameters,
    TeletexNonBasicParameters
    FROM MTSAbstractService {joint-ISO-CCITT mhs-motis(6)
                             mts(3) modules(0) mts-abstract-service(1)}
    UserCertificate, CACertificate, CrossCertificatePair, CertificateRevocationList,
    AuthorityRevocationList, UserPassword
    FROM AuthenticationFramework, authenticationFramework
    ub-answerback,
    ub-common-name, ub-surname, ub-serial-number,
    ub-locality-name, ub-state-name,
    ub-street-address, ub-organization-name,
    ub-organizational-unit-name, ub-title,
    ub-description, ub-business-category, ub-postal-line,
    ub-postal-string, ub-postal-code, ub-post-office-box,
    ub-physical-office-name, ub-telex-number,
    ub-country-code, ub-teletex-terminal-id,
    ub-telephone-number, ub-x121-address,
    ub-international-isdn-number, ub-destination-indicator,
    ub-user-password
    FROM UpperBounds upperBounds;

-- attribute types --
objectClass ObjectClass ::= {attributeType 0}

aliasedObjectName AliasedObjectName ::= {attributeType 1}

knowledgeInformation ATTRIBUTE
    WITH ATTRIBUTE-SYNTAX caseIgnoreStringSyntax
    ::= {attributeType 2}

commonName ATTRIBUTE
    WITH ATTRIBUTE-SYNTAX
        caseIgnoreStringSyntax
        (SIZE(1..ub-common-name))
    ::= {attributeType 3}

surname ATTRIBUTE
    WITH ATTRIBUTE-SYNTAX
        caseIgnoreStringSyntax
        (SIZE(1..ub-surname))
    ::= {attributeType 4}

serialNumber ATTRIBUTE
    WITH ATTRIBUTE-SYNTAX
        printableStringSyntax
        (SIZE(1..ub-serial-number))
    ::= {attributeType 5}
```

```

countryName ATTRIBUTE
  WITH ATTRIBUTE-SYNTAX
    PrintableString (SIZE(2)) -- IS 3166 codes only
  MATCHES FOR EQUALITY
  SINGLE VALUE
  ::= {attributeType 6}

localityName ATTRIBUTE
  WITH ATTRIBUTE-SYNTAX
    caseIgnoreStringSyntax
      (SIZE(1..ub-locality-name))
  ::= {attributeType 7}

stateOrProvinceName ATTRIBUTE
  WITH ATTRIBUTE-SYNTAX
    caseIgnoreStringSyntax
      (SIZE(1..ub-state-name))
  ::= {attributeType 8}

streetAddress ATTRIBUTE
  WITH ATTRIBUTE-SYNTAX
    caseIgnoreStringSyntax
      (SIZE(1..ub-street-address))
  ::= {attributeType 9}

organizationName ATTRIBUTE
  WITH ATTRIBUTE-SYNTAX
    caseIgnoreStringSyntax
      (SIZE(1..ub-organization-Name))
  ::= {attributeType 10}

organizationalUnitName ATTRIBUTE
  WITH ATTRIBUTE-SYNTAX
    caseIgnoreStringSyntax
      (SIZE(1..ub-organizational-unit-name))
  ::= {attributeType 11}

title ATTRIBUTE
  WITH ATTRIBUTE-SYNTAX
    caseIgnoreStringSyntax
      (SIZE(1..ub-title))
  ::= {attributeType 12}

description ATTRIBUTE
  WITH ATTRIBUTE-SYNTAX
    caseIgnoreStringSyntax
      (SIZE(1..ub-description))
  ::= {attributeType 13}

searchGuide ATTRIBUTE
  WITH ATTRIBUTE-SYNTAX
    Criteria
  ::= {attributeType 14}

Guide ::= SET {
  objectClass [0] OBJECT-CLASS OPTIONAL,
  criteria [1] Criteria }

Criteria ::=
  CHOICE {
    type [0] CriteriaItem,
    and [1] SET OF Criteria
    or [2] SET OF Criteria
    not [3] Criteria}

CriteriaItem ::=
  CHOICE {
    equality [0] AttributeType
    substrings [1] AttributeType
    greaterOrEqual [2] AttributeType
    lessOrEqual [3] AttributeType
    approximateMatch [4] AttributeType
  }

```


businessCategory ATTRIBUTE
 WITH ATTRIBUTE-SYNTAX
 caseIgnoreStringSyntax
 (SIZE(1..ub-business-category))
 ::= {attributeType 15}

postalAddress ATTRIBUTE
 WITH ATTRIBUTE-SYNTAX PostalAddress
 MATCHES FOR EQUALITY
 ::= {attributeType 16}

PostalAddress ::= SEQUENCE SIZE(1..ub-postal-line) OF
CHOICE {
 T61String (SIZE(1..ub-postal-string)),
 PrintableString (SIZE(1..ub-postal-string))
}

postalCode ATTRIBUTE
 WITH ATTRIBUTE-SYNTAX
 caseIgnoreStringSyntax
 (SIZE(1..ub-postal-code))
 ::= {attributeType 17}

postOfficeBox ATTRIBUTE
 WITH ATTRIBUTE-SYNTAX
 caseIgnoreStringSyntax
 (SIZE(1..ub-post-office-box))
 ::= {attributeType 18}

physicalDeliveryOfficeName ATTRIBUTE
 WITH ATTRIBUTE-SYNTAX
 caseIgnoreStringSyntax
 (SIZE(1..ub-physical-office-name))
 ::= {attributeType 19}

telephoneNumber ATTRIBUTE
 WITH ATTRIBUTE-SYNTAX
 telephoneNumberSyntax
 ::= {attributeType 20}

telexNumber ATTRIBUTE
 WITH ATTRIBUTE-SYNTAX TelexNumber
 ::= {attributeType 21}

TelexNumber ::= SEQUENCE {
 telexNumber PrintableString
 (SIZE(1..ub-telex-number)),
 countryCode PrintableString,
 (SIZE(1..ub-country-code)),
 answerback PrintableString
 (SIZE(1..ub-answerback))
}

teletexTerminalIdentifier ATTRIBUTE
 WITH ATTRIBUTE-SYNTAX
 TeletexTerminalIdentifier
 ::= {attributeType 22}

TeletexTerminalIdentifier ::= SEQUENCE {
 teletexTerminalPrintableString
 (SIZE(1..ub-teletex-terminal-id)),
 parameters TeletexNonBasicParameters
 OPTIONAL
}

facsimileTelephoneNumber ATTRIBUTE
 WITH ATTRIBUTE-SYNTAX
 FacsimileTelephoneNumber
 ::= {attributeType 23}

FacsimileTelephoneNumber ::= SEQUENCE {
 telephoneNumber PrintableString
 (SIZE(1..ub-telephone-number)),
 parameters G3FacsimileNonBasicParameters OPTIONAL
}

x121Address ATTRIBUTE
WITH ATTRIBUTE-SYNTAX
 NumericString
 (SIZE(1..ub-x121-address))
MATCHES FOR EQUALITY SUBSTRINGS
::= {attributeType 24}

internationalISDNNumber ATTRIBUTE
WITH ATTRIBUTE-SYNTAX
 NumericString
 (SIZE(1..ub-isdn-address))
::= {attributeType 25}

registeredAddress ATTRIBUTE
WITH ATTRIBUTE-SYNTAX PostalAddress
::= {attributeType 26}

destinationIndicator ATTRIBUTE
WITH ATTRIBUTE-SYNTAX
 PrintableString
 (SIZE(1..ub-destination-indicator))
- alphabetical characters only
MATCHES FOR EQUALITY SUBSTRINGS
::= {attributeType 27}

preferredDeliveryMethod ATTRIBUTE
WITH ATTRIBUTE-SYNTAX
 SEQUENCE OF INTEGER {
 any-delivery-method (0),
 mhs-delivery (1),
 physical-delivery (2),
 telex-delivery (3),
 teletex-delivery (4),
 g3-facsimile-delivery (5),
 g4-facsimile-delivery (6),
 ia5-terminal-delivery (7),
 videotex-delivery (8),
 telephone-delivery (9))
 SINGLE VALUE
::= {attributeType 28}

presentationAddress ATTRIBUTE
WITH ATTRIBUTE-SYNTAX
 PresentationAddress
MATCHES FOR EQUALITY
 SINGLE VALUE
::= {attributeType 29}

PresentationAddress ::= SEQUENCE {
 pSelector [0] OCTET STRING OPTIONAL,
 sSelector [1] OCTET STRING OPTIONAL,
 tSelector [2] OCTET STRING OPTIONAL,
 nAddresses [3] SET SIZE(1..MAX) OF OCTET STRING}

supportedApplicationContext ATTRIBUTE
WITH ATTRIBUTE-SYNTAX
 objectIdentifierSyntax
::= {attributeType 30}

member ATTRIBUTE
WITH ATTRIBUTE-SYNTAX
 distinguishedNameSyntax
::= {attributeType 31}

owner ATTRIBUTE
WITH ATTRIBUTE-SYNTAX
 distinguishedNameSyntax
::= {attributeType 32}

```

roleOccupant ATTRIBUTE
  WITH ATTRIBUTE-SYNTAX
    distinguishedNameSyntax
    ::= {attributeType 33}

seeAlso ATTRIBUTE
  WITH ATTRIBUTE-SYNTAX
    distinguishedNameSyntax
    ::= {attributeType 34}

userPassword UserPassword
    ::= {attributeType 35}

userCertificate UserCertificate
    ::= {attributeType 36}

cACertificate CACertificate
    ::= {attributeType 37}

authorityRevocationList AuthorityRevocationList
    ::= {attributeType 38}

certificateRevocationList CertificateRevocationList
    ::= {attributeType 39}

CrossCertificatePair CrossCertificatePair
    ::= {attributeType 40}

-- attribute syntaxes --

undefined ATTRIBUTE-SYNTAX
  ANY
  ::= {attributeSyntax 0}

distinguishedNameSyntax DistinguishedNameSyntax
    ::= {attributeSyntax 1}

objectIdentifierSyntax ObjectIdentifierSyntax
    ::= {attributeSyntax 2}

caseExactStringSyntax ATTRIBUTE-SYNTAX
  CHOICE {T61String, PrintableString}
  MATCHES FOR EQUALITY SUBSTRINGS
  ::= {attributeSyntax 3}

caseIgnoreSyntax ATTRIBUTE-SYNTAX
  CHOICE {T61String, PrintableString}
  MATCHES FOR EQUALITY SUBSTRINGS
  ::= {attributeSyntax 4}

printableStringSyntax ATTRIBUTE-SYNTAX
  PrintableString
  MATCHES FOR EQUALITY SUBSTRINGS
  ::= {attributeSyntax 5}

numericStringSyntax ATTRIBUTE-SYNTAX
  NumericString
  MATCHES FOR EQUALITY SUBSTRINGS
  ::= {attributeSyntax 6}

caseIgnoreListSyntax ATTRIBUTE-SYNTAX
  SEQUENCE OF
    CHOICE {T61String, PrintableString}
  MATCHES FOR EQUALITY SUBSTRINGS
  ::= {attributeSyntax 7}

booleanSyntax ATTRIBUTE-SYNTAX
  BOOLEAN
  MATCHES FOR EQUALITY
  ::= {attributeSyntax 8}

```

integerSyntax ATTRIBUTE-SYNTAX

INTEGER

MATCHES FOR EQUALITY ORDERING

::= {attributeSyntax 9}

octetStringSyntax ATTRIBUTE-SYNTAX

OCTET STRING

MATCHES FOR EQUALITY SUBSTRINGS ORDERING

::= {attributeSyntax 10}

uTCTimeSyntax ATTRIBUTE-SYNTAX

UTCTime

MATCHES FOR EQUALITY ORDERING

::= {attributeSyntax 11}

telephoneNumberSyntax ATTRIBUTE-SYNTAX

PrintableString

(SIZE(1..ub-telephone-number))

MATCHES FOR EQUALITY SUBSTRINGS

::= {attributeSyntax 12}

ANNEX B

(to Recommendation X.520)

Index of Attribute Types and Syntaxes

ATTRIBUTE TYPES

A	Aliased Object Name *	§ 5.1.2
	Authority Revocation List	§ 5.11.4
B	Business Category	§ 5.5.3
C	CA Certificate	§ 5.11.3
	Certificate Revocation List	§ 5.11.5
	Common Name	§ 5.2.1
	Country Name	§ 5.3.1
	Cross Certificate Pair	§ 5.11.6
D	Description	§ 5.5.1
	Destination Indicator	§ 5.7.8
F	Facsimile Telephone Number	§ 5.7.4
I	International ISDN Number	§ 5.7.6
K	Knowledge Information	§ 5.1.3
L	Locality Name	§ 5.3.2
M	Member	§ 5.10.1
O	Object Class *	§ 5.1.1
	Organization Name	§ 5.4.1
	Organizational Unit Name	§ 5.4.2
	Owner	§ 5.10.2

ATTRIBUTE SYNTAXES

B	Boolean	§ 6.3.1
C	Case Exact String	§ 6.2.1
	Case Ignore List	§ 6.2.5
	Case Ignore String	§ 6.2.3
D	Distinguished Name *	§ 6.1.2
I	Integer	§ 6.3.2
N	Numeric String	§ 6.2.4
O	Object Identifier *	§ 6.1.3
	Object String	§ 6.3.2
P	Printable String	§ 6.2.3
T	Telephone Number	§ 6.3.5
U	UTC Time	§ 6.3.4
	Undefined	§ 6.1.1

* Known to and used by the Directory itself.

ATTRIBUTE TYPES

P	Physical Delivery Office Name	§ 5.6.4
	Post Office Box	§ 5.6.3
	Postal Address	§ 5.6.1
	Postal Code	§ 5.6.2
	Preferred Delivery Method	§ 5.8.1
	Presentation Address	§ 5.9.1
R	Registered Address	§ 5.7.7
	Role Occupant	§ 5.10.3
S	Search Guide	§ 5.5.2
	See Also	§ 5.10.4
	Serial Number	§ 5.2.3
	State or Province Name	§ 5.3.2
	Street Address	§ 5.3.4
	Supported Application Context	§ 5.9.2
	Surname	§ 5.2.2
T	Telephone Number	§ 5.7.1
	Teletex Terminal Identifier	§ 5.7.3
	Telex Number	§ 5.7.2
	Title	§ 5.4.3
U	User Certificate	§ 5.11.2
	User Password	§ 5.11.1
X	X.121 Address	§ 5.7.5

ANNEX C

(to Recommendation X.520)

Upper Bounds

This Annex is part of the Recommendation.

UpperBounds {joint-ISO-CCITT ds(5) modules(1)
upperBounds(10)}

DEFINITIONS ::=
BEGIN

-- Exports everything --

ub-answerback	INTEGER ::= 8
ub-common-name	INTEGER ::= 64
ub-surname	INTEGER ::= 64
ub-serial-number	INTEGER ::= 64
ub-locality-name	INTEGER ::= 128
ub-state-name	INTEGER ::= 128
ub-street-address	INTEGER ::= 128
ub-organization-name	INTEGER ::= 64
ub-organizational-unit-name	INTEGER ::= 64
ub-title	INTEGER ::= 64

ub-description	INTEGER ::= 1024
ub-business-category	INTEGER ::= 128
ub-postal-line	INTEGER ::= 6
ub-postal-string	INTEGER ::= 30
ub-postal-code	INTEGER ::= 40
ub-post-office-box	INTEGER ::= 40
ub-physical-office-name	INTEGER ::= 128
ub-telex-number	INTEGER ::= 14
ub-country-code	INTEGER ::= 4
ub-teletex-terminal-id	INTEGER ::= 24
ub-telephone-number	INTEGER ::= 32
ub-x121-address	INTEGER ::= 15
ub-international-isdn-number	INTEGER ::= 16
ub-destination-indicator	INTEGER ::= 128
ub-user-password	INTEGER ::= 128

END

Recommendation X.521

THE DIRECTORY - SELECTED OBJECT CLASSES ¹⁾

(Melbourne, 1988)

CONTENTS

- 0 *Introduction*
- 1 *Scope and field of application*
- 2 *References*
- 3 *Definitions and abbreviations*
 - 3.1 OSI Reference Model Definitions
 - 3.2 Directory Model Definitions
- 4 *Notation*

¹⁾ Recommendation X.521 and ISO 9594-7, Information Processing Systems - Open Systems Interconnection - The Directory - Selected object classes, were developed in close collaboration and are technically aligned.

SECTION 1 - *Selected Object Classes*

5 *Definitions of Useful Attribute Sets*

- 5.1 Telecommunication Attribute Set
- 5.2 Postal Attribute Set
- 5.3 Locale Attribute Set
- 5.4 Organizational Attribute Set

6 *Definition of Selected Object Classes*

- 6.1 Top
- 6.2 Alias
- 6.3 Country
- 6.4 Locality
- 6.5 Organization
- 6.6 Organizational Unit
- 6.7 Person
- 6.8 Organizational Person
- 6.9 Organizational Role
- 6.10 Group of Names
- 6.11 Residential Person
- 6.12 Application Process
- 6.13 Application Entity
- 6.14 DSA
- 6.15 Device
- 6.16 Strong Authentication User
- 6.17 Certification Authority

Annex A - Selected Object Classes in ASN.1

Annex B - Suggested Name Forms and DIT Structures

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. The set of all 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 defines (in section one) a number of attribute sets and object classes which may be found useful across a range of applications of the Directory.

0.4 Annex A, which is a part of the standard, provides an ASN.1 module containing all of the type and value definitions which appear in this document.

0.5 Annex B, which is not part of the Recommendation provides some common naming and structure rules which may or may not be used by Administrative authorities.

1 Scope and field of application

1.1 This Recommendation defines a number of selected attribute sets and object classes which may be found useful across a range of applications of the Directory. The definition of an attribute set involves identifying the attributes that it contains, and facilitates the definition of object classes. The definition of an object class involves optionally allocating an Object Identifier to it, and listing a number of attribute types which are relevant to objects of that class. These definitions are used by the administrative authority which is responsible for the management of the Directory information.

1.2 Any Administrative Authority can define its own object classes and subclasses for any purpose.

Note 1 - These definitions may or may not use the notation specified in Recommendation X.501.

Note 2 - It is recommended that an object class defined in this document, or a subclass derived from one, be used in preference to the generation of a new one, whenever the semantics is appropriate for the application.

1.3 Administrative authorities may support some or all the selected object classes, and may also add object classes.

All Administrative authorities shall support the object classes which the directory uses for its own purpose (the top, alias and DSA object classes).

2 References

Recommendation X.200 - Open Systems Interconnection - Basic Reference Model (see also ISO 7498)

Recommendation X.500 - The Directory - Overview of Concepts, Models and Services (see also ISO 9594-1)

Recommendation X.501 - The Directory - Models (see also ISO 9594-2)

3 Definitions and abbreviations

3.1 OSI Reference Model Definitions

This Recommendation makes use of the following definitions from Recommendation X.200:

- a) *application-entity*;
- b) *application-process*.

3.2 Directory Model Definitions

This Recommendation makes use of the following definitions from Recommendation X.501.

- a) *attribute*;
- b) *attribute type*;
- c) *Directory Information Tree (DIT)*;
- d) *Directory System Agent (DSA)*;
- e) *attribute set*;
- f) *entry*;
- g) *name*;
- h) *object class*;
- i) *subclass*.

4 Notation

Object classes are defined in this document by the use of special notation, defined as an ASN.1 macro, **OBJECT-CLASS**, in Recommendation X.501. One "generic" object identifier (**objectClass**) is used in specifying the object identifiers being allocated to object classes. Its definition can be found in Annex B of the same Recommendation.

Attribute sets are defined in this document by the use of special notation, defined as an ASN.1 macro **ATTRIBUTE-SET**, in Recommendation X.501. One "generic" object identifier (**attributeSet**) is used in specifying the object identifiers being allocated to attribute set definitions. Its definition can be found in Annex B of the same Recommendation.

SECTION 1 - *Selected Object Classes*

5 Definition of Useful Attribute Sets

5.1 *Telecommunication Attribute Set*

This set of attributes is used to define those which are commonly used for business communications.

```
telecommunicationAttributeSet ATTRIBUTE-SET
CONTAINS {
    facsimileTelephoneNumber,
    iSDNAddress,
    telephoneNumber,
    teletexTerminalIdentifier,
    telexNumber, X121Address,
    preferredDeliveryMethod,
    destinationIndicator,
    registeredAddress}
::= {attributeSet 0}
```

5.2 *Postal Attribute Set*

This set of attributes is used to define those which are directly associated with postal delivery.

```
postalAttributeSet ATTRIBUTE-SET
CONTAINS {
    physicalDeliveryOfficeName,
    postalAddress,
    postalCode,
    postOfficeBox,
    streetAddress}
::= {attributeSet 1}
```

5.3 *Locale Attribute Set*

This set of attributes is used to define those which are commonly used for search purposes to indicate the locale of an object.

```
localeAttributeSet ATTRIBUTE-SET
CONTAINS {
    localityName,
    stateOrProvinceName,
    streetAddress}
::= {attributeSet 2}
```

5.4 *Organizational Attribute Set*

This set of attributes is used to define the attributes that an organization or organizational unit may typically possess.

```
organizationalAttributeSet ATTRIBUTE-SET
CONTAINS {
    description,
    localeAttributeSet,
    postalAttributeSet,
    telecommunicationAttributeSet,
    businessCategory,
    seeAlso,
    searchGuide,
    userPassword}
::= {attributeSet 3}
```

6 Definition of Selected Object Classes

6.1 *Top*

The *top* object class, of which every other object class is a subclass, is defined, except for the allocation of an object identifier, in Recommendation X.501.

top Top ::= {objectClass 0}

6.2 *Alias*

The *alias* object class, from which classes for alias entries may be derived, is defined, except for the allocation of an object identifier, in Recommendation X.501.

alias Alias ::= {objectClass 1}

6.3 *Country*

A *Country* object class is used to define country entries in the DIT.

country OBJECT-CLASS
SUBCLASS OF top
MUST CONTAIN {
 countryName}
MAY CONTAIN {
 description,
 searchGuide}
::= {objectClass 2}

6.4 *Locality*

The *Locality* object class is used to define locality in the DIT.

locality OBJECT-CLASS
SUBCLASS OF top
MAY CONTAIN {
 description,
 localityName,
 stateOrProvinceName,
 searchGuide,
 seeAlso,
 streetAddress}
::= {objectClass 3}

At least one of Locality Name or State or Province Name must be present.

6.5 *Organization*

The *Organization* object class is used to define organization entries in the DIT.

organization OBJECT-CLASS
SUBCLASS OF top
MUST CONTAIN {
 organizationName}
MAY CONTAIN {
 organizationalAttributeSet}
::= {objectClass 4}

6.6 *Organizational Unit*

The *Organizational Unit* object class is used to define entries representing subdivisions or organizations.

organizationalUnit OBJECT-CLASS
SUBCLASS OF top
MUST CONTAIN {
 organizationalUnitName}
MAY CONTAIN {
 organizationalAttributeSet}
::= {objectClass 5}

6.7 *Person*

The *Person* object class is used to define entries representing people generically.

```
person OBJECT-CLASS
  SUBCLASS OF top
  MUST CONTAIN {
    commonName,
    surname}
  MAY CONTAIN {
    description,
    seeAlso,
    telephoneNumber,
    userPassword}
  ::= {objectClass 6}
```

6.8 *Organizational Person*

The *Organizational Person* object class is used to define entries representing people employed by, or in some other important way associated with, an organization.

```
organizationalPerson OBJECT-CLASS
  SUBCLASS OF person
  MAY CONTAIN {
    localeAttributeSet,
    organizationalUnitName,
    postalAttributeSet,
    telecommunicationAttributeSet,
    title}
  ::= {objectClass 7}
```

6.9 *Organizational Rôle*

The *Organizational Rôle* object class is used to define entries representing an organizational role, i.e. a position or rôle within an organization. An organizational rôle is normally considered to be filled by a particular organizational person. Over its lifetime, however, an organizational rôle may be filled by a number of different organizational people in succession. In general, an organizational rôle may be filled by a person or a non-human entity.

```
organizationalRole OBJECT-CLASS
  SUBCLASS OF top
  MUST CONTAIN {
    commonName}
  MAY CONTAIN {
    description,
    localeAttributeSet,
    organizationalUnitName,
    postalAttributeSet,
    preferredDeliveryMethod,
    roleOccupant,
    seeAlso,
    telecommunicationAttributeSet}
  ::= {objectClass 8}
```

6.10 *Group of Names*

The *Group of Names* object class is used to define entries representing an unordered set of names which represent individual objects or other groups of names. The membership of a group is static; that is, it is explicitly modified by administrative action, rather than dynamically determined each time the group is referred to.

The membership of a group can be reduced to a set of individual object's names by replacing each group with its membership. This process could be carried out recursively until all constituent group names have been eliminated, and only the names of individual objects remain.

```
groupOfNames OBJECT-CLASS
  SUBCLASS OF top
  MUST CONTAIN {
    commonName,
    member}
  MAY CONTAIN {
    description,
    organizationName,
    organizationalUnitName,
    owner,
    seeAlso,
    businessCategory}
  ::= {objectClass 9}
```

6.11 Residential Person

The *Residential Person* object class is used to define entries representing a person in the residential environment.

```
residentialPerson OBJECT-CLASS
  SUBCLASS OF person
  MUST CONTAIN {
    localityName}
  MAY CONTAIN {
    localeAttributeSet,
    postalAttributeSet,
    preferredDeliveryMethod,
    telecommunicationAttributeSet,
    businessCategory}
  ::= {objectClass 10}
```

6.12 Application Process

The *Application Process* object class is used to define entries representing application processes. An application process is an element within a real open system which performs the information processing for a particular application (see Recommendation X.200).

```
applicationProcess OBJECT-CLASS
  SUBCLASS OF top
  MUST CONTAIN {
    commonName}
  MAY CONTAIN {
    description,
    localityName,
    organizationalUnitName,
    seeAlso}
  ::= {objectClass 11}
```

6.13 Application Entity

The *Application Entity* object class is used to define entries representing application entities. An application entity consists of those aspects of an application-process pertinent to OSI.

```
applicationEntity OBJECT-CLASS
  SUBCLASS OF top
  MUST CONTAIN {
    commonName,
    presentationAddress}
  MAY CONTAIN {
    description,
    localityName,
```

```

        organizationName,
        organizationalUnitName,
        seeAlso,
        supportedApplicationContext}
 ::= {objectClass 12}

```

Note - If Application Entity is represented as a Directory object that is distinct from an Application Process, the **commonName** attribute is used to carry the value of Application Entity Qualifier.

6.14 DSA

The *DSA* object class is used to define entries representing DSAs. A DSA is as defined in Recommendation X.501.

```

dSA OBJECT-CLASS
  SUBCLASS OF applicationEntity
  MAY CONTAIN {
    knowledgeInformation}
 ::= {objectClass 13}

```

6.15 Device

The *Device* object class is used to define entries representing devices. A device is a physical unit which can communicate, such as a modem, disk drive, etc.

```

device OBJECT-CLASS
  SUBCLASS OF top
  MUST CONTAIN {
    commonName}
  MAY CONTAIN {
    description,
    localityName,
    organizationName,
    organizationalUnitName,
    owner,
    seeAlso,
    serialNumber}
 ::= {objectClass 14}

```

Note - At least one of **localityName**, **serialNumber**, **owner**, should be included. The choice is dependent on device type.

6.16 Strong Authentication User

The *Strong Authentication User* object class is used in defining entries for objects which participate in strong authentication, as defined in Recommendation X.509.

```

strongAuthenticationUser OBJECT-CLASS
  SUBCLASS OF top
  MUST CONTAIN {userCertificate}
 ::= {objectClass 15}

```

6.17 Certification Authority

The *Certification Authority* object class is used in defining entries for objects which act as certification authorities, as defined in Recommendation X.509.

```

certificationAuthority OBJECT-CLASS
  SUBCLASS OF top
  MUST CONTAIN {
    cACertificate,
    certificateRevocationList,
    authorityRevocationList }
  MAY CONTAIN {crossCertificatePair}
 ::= {objectClass 16}

```

ANNEX A

(to Recommendation X.521)

Selected Object Classes in ASN.1

This Annex includes all of the ASN.1 type and value definitions contained in this Recommendation in the form of the ASN.1 module, **SelectedObjectClasses**.

```
SelectedObjectClasses    {joint-ISO-CCITT ds(5) modules(1)
                          selectedObjectClasses(6)}

DEFINITIONS ::=
BEGIN
-- exports everything
IMPORTS
    objectClass, attributeSet, informationFramework, selectedAttributeTypes
    FROM UsefulDefinitions {joint-iso-ccitt ds(5) modules(1) usefulDefinitions(0)}
OBJECT-CLASS, ATTRIBUTE-SET, Top, Alias
    FROM InformationFramework informationFramework
authorityRevocationList, businessCategory, CACertificate, certificateRevocationList,
commonName, countryName, description, destinationIndicator, facsimileTelephoneNumber,
internationalISDNNumber, knowledgeInformation, localityName, member, organizationName,
organizationalUnitName, owner, physicalDeliveryOfficeName, postOfficeBox, postalAddress,
postalCode, preferredDeliveryMethod, presentationAddress, registeredAddress,
roleOccupant, searchGuide, seeAlso, serialNumber, stateOrProvinceName, streetAddress,
supportedApplicationContext, surname, telephoneNumber, teletexTerminalIdentifier,
telexNumber, title, userCertificate, userPassword, x121Address
    FROM SelectedAttributeTypes selectedAttributeTypes;
telecommunicationAttributeSet ATTRIBUTE-SET
    CONTAINS {
        facsimileTelephoneNumber,
        iSDNAddress,
        telephoneNumber,
        teletexTerminalIdentifier,
        telexNumber,
        x121Address, preferredDeliveryMethod, destinationIndicator,
        registeredAddress}
    ::= {attributeSet 0}
postalAttributeSet ATTRIBUTE-SET
    CONTAINS {
        physicalDeliveryOfficeName,
        postalAddress,
        postalCode,
        postOfficeBox,
        streetAddress}
    ::= {attributeSet 1}
localeAttributeSet ATTRIBUTE-SET
    CONTAINS {
        localityName,
        stateOrProvinceName,
        streetAddress}
    ::= {attributeSet 2}
organizationalAttributeSet ATTRIBUTE-SET
    CONTAINS {
        description,
        localeAttributeSet,
        postalAttributeSet,
        telecommunicationAttributeSet,
        businessCategory,
```

```

        seeAlso,
        searchGuide,
        userPassword}
        ::= {attributeSet 3}

top      Top ::= {objectClass 0}

alias    Alias ::= {objectClass 1}

country OBJECT-CLASS
  SUBCLASS OF top
  MUST CONTAIN {
    countryName}
  MAY CONTAIN {
    description,
    searchGuide}
  ::= {objectClass 2}

locality OBJECT-CLASS
  SUBCLASS OF top
  MAY CONTAIN {
    description,
    localityName,
    stateOrProvinceName,
    searchGuide,
    seeAlso,
    streetAddress}
  ::= {objectClass 3}

organization OBJECT-CLASS
  SUBCLASS OF top
  MUST CONTAIN {
    organizationName}
  MAY CONTAIN {
    organizationalAttributeSet}
  ::= {objectClass 4}

organizationalUnit OBJECT-CLASS
  SUBCLASS OF top
  MUST CONTAIN {
    organizationalUnitName}
  MAY CONTAIN {
    organizationalAttributeSet}
  ::= {objectClass 5}

person OBJECT-CLASS
  SUBCLASS OF top
  MUST CONTAIN {
    commonName,
    surname}
  MAY CONTAIN {
    description,
    seeAlso,
    telephoneNumber,
    userPassword}
  ::= {objectClass 6}

organizationalPerson OBJECT-CLASS
  SUBCLASS OF person
  MAY CONTAIN {
    localeAttributeSet,
    organizationalUnitName,
    postalAttributeSet,
    telecommunicationAttributeSet,
    title}
  ::= {objectClass 7}

```

organizationalRole OBJECT-CLASS
SUBCLASS OF top
MUST CONTAIN {
 commonName}
MAY CONTAIN {
 description,
 localeAttributeSet,
 organizationalUnitName,
 postalAttributeSet,
 preferredDeliveryMethod,
 roleOccupant,
 seeAlso,
 telecommunicationAttributeSet}
::= {objectClass 8}

groupOfNames OBJECT-CLASS
SUBCLASS OF top
MUST CONTAIN {
 commonName,
 member}
MAY CONTAIN {
 description,
 organizationName,
 organizationalUnitName,
 owner,
 seeAlso,
 businessCategory}
::= {objectClass 9}

residentialPerson OBJECT-CLASS
SUBCLASS OF person
MUST CONTAIN {
 localityName}
MAY CONTAIN {
 localeAttributeSet,
 postalAttributeSet,
 preferredDeliveryMethod,
 telecommunicationAttributeSet,
 businessCategory}
::= {objectClass 10}

applicationProcess OBJECT-CLASS
SUBCLASS OF top
MUST CONTAIN {
 commonName}
MAY CONTAIN {
 description,
 localityName,
 organizationalUnitName,
 seeAlso}
::= {objectClass 11}

applicationEntity OBJECT-CLASS
SUBCLASS OF top
MUST CONTAIN {
 commonName,
 presentationAddress}
MAY CONTAIN {
 description,
 localityName,
 organizationName,
 organizationalUnitName,
 seeAlso,
 supportedApplicationContext}
::= {objectClass 12}


```

dSA OBJECT-CLASS
  SUBCLASS OF applicationEntity
  MAY CONTAIN {
    knowledgeInformation}
  ::= {objectClass 13}

device OBJECT-CLASS
  SUBCLASS OF top
  MUST CONTAIN {
    commonName}
  MAY CONTAIN {
    description,
    localityName,
    organizationName,
    organizationalUnitName,
    owner,
    seeAlso,
    serialNumber}
  ::= {objectClass 14}

strongAuthenticationUser OBJECT-CLASS
  SUBCLASS OF top
  MUST CONTAIN {
    userCertificate}
  ::= {objectClass 15}

certificationAuthority OBJECT-CLASS
  SUBCLASS OF top
  MUST CONTAIN {
    cACertificate,
    certificateRevocationList,
    authorityRevocationList}
  MAY CONTAIN {
    crossCertificatePair}
  ::= {objectClass 16}

END

```

ANNEX B

(to Recommendation X.521)

Suggested Name Forms and DIT Structures

This Annex is not part of this Recommendation.

This Annex suggests some common naming practices and DIT structures that may or may not be used by an Administrative authority. Naming practices and DIT structure definitions for an object class include specification of the attributes used for naming and which object classes its superior entry or its subordinate entry in the DIT can have. All entries of a given object class must include at least the attributes used for naming. Users of the Directory should be informed of the suggested name forms to be able to predict names of objects with which they communicate. The following paragraphs suggest naming and structure rules for some object classes.

The structure rules are depicted in Figure B-1/X.521.

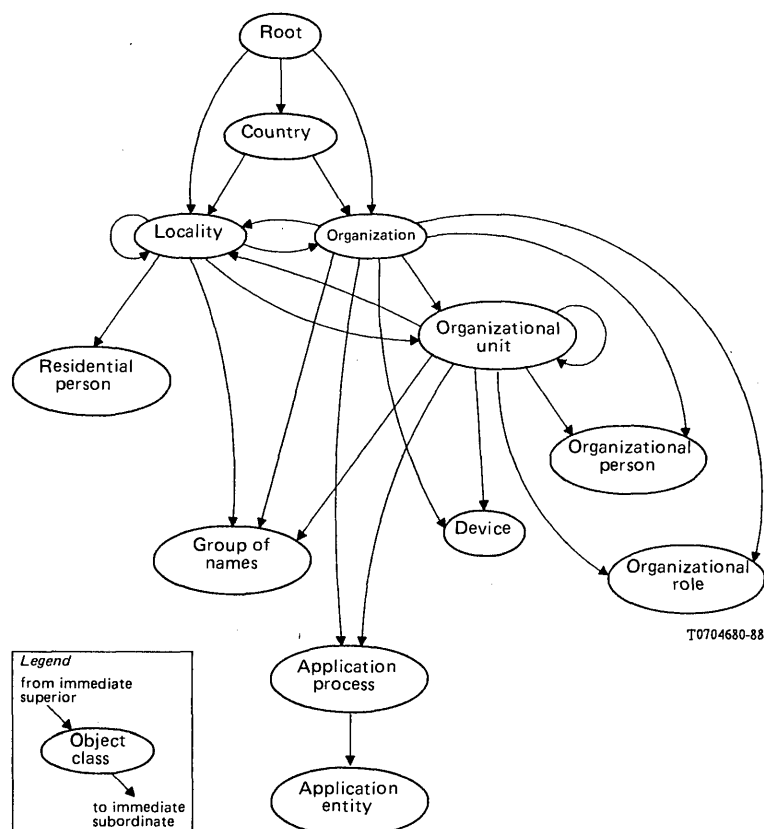


FIGURE B-1/X.521

Suggested DIT structure

B.1 Country

Attribute **countryName** is used for naming.

The Root is the immediate superior to entries of object class **country**.

B.2 Organization

Attribute **organizationName** is used for naming.

The Root, **country** or **locality** can be immediate superior to entries of object class **organization**.

Note - When the organization is directly under the root, this denotes an international organization. The values of the **organizationName** attribute for international organizations must all be distinct.

B.3 Locality

Attribute **localityName** or **stateOrprovinceName** is used for naming.

The Root, **country**, **locality**, **organization** or **organizationalUnit** can be immediate superior to entries of object class **locality**.

B.4 Organizational Unit

Attribute **organizationalUnitName** is used for naming.

organization, **organizationalUnit** or **locality** can be immediate superior to entries of object class **organizationalUnit**.

B.5 *Organizational Person*

Attribute **commonName** and optionally **organizationalUnitName** is used for naming.

organization or **organizationalUnit** can be immediate superior to entries of object class **organizationalPerson**.

Note - There are two ways that an **organizationalUnitName** attribute may be acquired in names: by having an **organizationalUnit** object as superior or by having such an attribute directly.

B.6 *Organizational Rôle*

Attribute **commonName** is used for naming.

organization or **organizationalUnit** can be immediate superior to entries of object class **organizationalRole**.

Note - There are two ways that an **organizationalUnitName** attribute may be acquired in names: by having an **organizationalUnit** object as superior or by having such an attribute directly.

B.7 *Group of Names*

Attribute **commonName** is used for naming.

locality, **organization** or **organizationalUnit** can be immediate superior to entries of object class **groupOfNames**.

Note - There are two ways that an **organizationalUnitName** attribute may be acquired in names: by having an **organizationalUnit** object as superior or by having such an attribute directly.

B.8 *Residential Person*

Attribute **commonName** and optionally **streetAddress** is used for naming.

locality is the immediate superior to entries of object class **residentialPerson**.

B.9 *Application Entity*

Attribute **commonName** is used for naming. The **commonName** should contain an application-entity qualifier (see Recommendation X.200).

applicationProcess is the immediate superior to entries of object class **applicationEntity**.

B.10 *Device*

Attribute **commonName** is used for naming.

organization or **organizationalUnit** can be immediate superior to entries of object class **device**.

Note - There are two ways that an **organizationalUnitName** attribute may be acquired in names: by having an **organizationalUnit** object as superior or by having such an attribute directly.

B.11 *Application Process*

Attribute **commonName** is used for naming.

organization or **organizationalUnit** can be immediate superior to entries of object class **applicationProcess**.

Note 1 - How **commonName** should be chosen for an Application Entity is documented in Recommendation X.200.

Note 2 - There are two ways that an **organizationalUnitName** attribute may be acquired in names: by having an **organizationalUnit** object as superior or by having such an attribute directly.

