INTERNATIONAL TELECOMMUNICATION UNION

ITU-T X.511

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

(11/93)

DATA NETWORK AND OPEN SYSTEM COMMUNICATIONS

DIRECTORY

INFORMATION TECHNOLOGY – OPEN SYSTEMS INTERCONNECTION – THE DIRECTORY: ABSTRACT SERVICE DEFINITION

ITU-T Recommendation X.511
Superseded by a more recent version

(Previously "CCITT Recommendation")

FOREWORD

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

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

In some areas of information technology which fall within ITU-T's purview, the necessary standards are prepared on a collaborative basis with ISO and IEC. The text of ITU-T Recommendation X.511 was approved on 16th of November 1993. The identical text is also published as ISO/IEC International Standard 9594-3.

NOTE

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

ITU-T X-SERIES RECOMMENDATIONS
DATA NETWORKS AND OPEN SYSTEM COMMUNICATIONS
(February 1994)

ORGANIZATION OF X-SERIES RECOMMENDATIONS

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

Contents

Summary

This Recommendation | International Standard defines in an abstract way the externally visible service provided by the Directory, including bind and unbind operations, read operations, search operations, modify operations and errors.

Introduction

This Recommendation | International Standard, together with the other Recommendations | International Standards, 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.

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.

This Recommendation | International Standard defines the capabilities provided by the Directory to its users.

This second edition technically revises and enhances, but does not replace, the first edition of this Recommendation | International Standard. Implementations may still claim conformance to the first edition.

This second edition specifies version 1 of the Directory service and protocols. The first edition also specifies version 1. Differences between the services and between the protocols defined in the two editions are accommodated using the rules of extensibility defined in this edition of X.519 | ISO/IEC 9594-5.

Annex A, which is an integral part of this Recommendation | International Standard, provides the ASN.1 module for the directory abstract service.

Annex B, which is an integral part of this Recommendation | International Standard, provides charts that describe the semantics associated with Basic Access Control as it applies to the processing of a Directory operation.

Annex C, which is not an integral part of this Recommendation | International Standard, lists the amendments and defect reports that have been incorporated to form this edition of this Recommendation | International Standard.

INTERNATIONAL STANDARD

ITU-T RECOMMENDATION

INFORMATION TECHNOLOGY – OPEN SYSTEMS INTERCONNECTION – THE DIRECTORY: ABSTRACT SERVICE DEFINITION

1 Scope

This Recommendation | International Standard defines in an abstract way the externally visible service provided by the Directory.

This Recommendation | International Standard does not specify individual implementations or products.

2 Normative references

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

2.1 Identical Recommendations | International Standards

- ITU-T Recommendation X.500 (1993) | ISO/IEC 9594-1:1994, *Information technology – Open Systems Interconnection – The Directory: Overview of concepts, models and services*.
- ITU-T Recommendation X.501 (1993) | ISO/IEC 9594-2:1994, *Information technology – Open Systems Interconnection –The Directory: Models*.
- ITU-T Recommendation X.518 (1993) | ISO/IEC 9594-4:1994, *Information technology – Open Systems Interconnection –The Directory: Procedures for distributed operation*.
- ITU-T Recommendation X.519 (1993) | ISO/IEC 9594-5:1994, *Information technology – Open Systems Interconnection –The Directory: Protocol specifications*.
- ITU-T Recommendation X.520 (1993) | ISO/IEC 9594-6:1994, *Information technology – Open Systems Interconnection –The Directory: Selected attribute types*.
- ITU-T Recommendation X.521 (1993) | ISO/IEC 9594-7:1994, *Information technology – Open Systems Interconnection –The Directory: Selected object classes*.
- ITU-T Recommendation X.509 (1993) | ISO/IEC 9594-8:1994, *Information technology – Open Systems Interconnection –The Directory: Authentication framework*.
- ITU-T Recommendation X.525 (1993) | ISO/IEC 9594-9:1994, *Information technology – Open Systems Interconnection – The Directory: Replication*
- ITU-T Recommendation X.680 (1994) | ISO/IEC 8824-1:1994, *Information technology – Abstract Syntax Notation One (ASN.1): Specification of basic notation*.
- ITU-T Recommendation X.681 (1994) | ISO/IEC 8824-2:1994, *Information technology – Abstract Syntax Notation One (ASN.1): Information object specification*.
- ITU-T Recommendation X.682 (1994) | ISO/IEC 8824-3:1994, *Information technology – Abstract Syntax Notation One (ASN.1): Constraint specification*.
- ITU-T Recommendation X.683 (1994) | ISO/IEC 8824-4:1994, *Information technology – Abstract Syntax Notation One (ASN.1): Parameterization of ASN.1 specifications*.

- ITU-T Recommendation X.880 (1994) | ISO/IEC 13712-1:1994, *Information technology – Remote Operations: Concepts, model and notation*.
- ITU-T Recommendation X.881 (1994) | ISO/IEC 13712-2:1994, *Information technology – Remote Operations: OSI realizations – Remote Operations Service Element (ROSE) service definition*.

2.2 Paired Recommendations | International Standards equivalent in technical content

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

  ISO 7498:1984/Corr.1:1988, Information Processing Systems – Open Systems Interconnection – Basic Reference Model.

3 Definitions

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

3.1 Basic Directory definitions

The following terms are defined in ITU-T Rec. X.500 | ISO/IEC 9594-1:

a) *Directory;*

b) *Directory Information Base;*

c) *(Directory) User.*

3.2 Directory model definitions

The following terms aree defined in ITU-T Rec. X.501 | ISO/IEC 9594-2:

a) *Directory System Agent;*

b) *Directory User Agent.*

3.3 Directory Information Base definitions

The following terms are defined in ITU-T Rec. X.501 | ISO/IEC 9594-2:

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

The following terms are defined in ITU-T Rec. X.501 | ISO/IEC 9594-2:

a) *attribute;*

b) *attribute type;*

c) *attribute value;*

d) *attribute value assertion;*

e) *operational attribute;*

f) *user attribute;*

g) *matching rule.*

3.5 Name definitions

The following terms are defined in ITU-T Rec. X.501 | ISO/IEC 9594-2:

a) *alias, alias name;*

b) *distinguished name;*

c) *(directory) name;*

d) *purported name;*

e) *relative distinguished name.*

3.6 Distributed operations definitions

The following terms are defined in ITU-T Rec. X.518 | ISO/IEC 9594-4:

a) *chaining;*

b) *referral.*

3.7 Abstract Service definitions

The following terms are defined in this Recomendation | International Standard:

3.7.1 **filter**: An assertion about the presence or value of certain attributes of an entry in order to limit the scope of a search.

3.7.2 **originator**: The user that originated an operation.

3.7.3 **service controls**: Parameters conveyed as part of an operation which constrain various aspects of its performance.

4 Abbreviations

For the purposes of this Recommendation | International Standard, the following abbreviations apply:

AVA Attribute Value Assertion

DIB Directory Information Base

DIT Directory Information Tree

DSA Directory System Agent

DUA Directory User Agent

DMD Directory Management Domain

RDN Relative Distinguished Name

5 Conventions

With minor exceptions this Directory Specification has been prepared according to the “Presentation of ITU-T | ISO/IEC common text” guidelines in the Guide for ITU-TS and ISO/IEC JTC 1 Cooperation, March 1993.

The term “Directory Specification” (as in “this Directory Specification”) shall be taken to mean ITU-T Rec. X.511 | ISO/IEC 9594-3. The term “Directory Specifications” shall be taken to mean the X.500-Series Recommendations and all parts of ISO/IEC 9594.

This Directory Specification uses the term “1988 edition systems” to refer to systems conforming to the previous (1988) edition of the Directory Specifications, i.e. the 1988 edition of the series of ITU-T X.500 Recommendations and the ISO/IEC 9594:1990 edition. Systems conforming to the current Directory Specifications are referred to as “1993 edition systems”.

If the items in a list are numbered (as opposed to using "–" or letters), then the items shall be considered steps in a procedure.

This Directory Specification defines directory operations using the Remote Operation notation defined in ITU-T Rec. X.880 | ISO/IEC 9072-1.

6 Overview of the Directory Service

As described in ITU-T Rec. X.501 | ISO/IEC 9594-2, 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. Through an access point the Directory provides service to its users by means of a number of Directory operations.

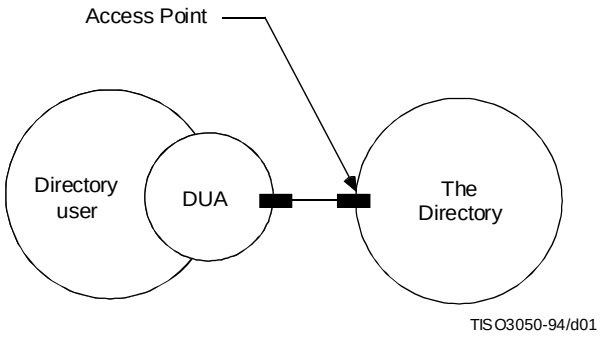

Figure 1 – Access to the Directory

The Directory operations are of three different kinds:

a) Directory Read operations, which interrogate a single Directory entry;

b) Directory Search operations, which interrogate potentially several Directory entries; and

c) Directory Modify operations.

The Directory Read operations, the Directory Search operations and the Directory Modify operations are specified in clauses 9, 10, and 11, respectively. Conformance to Directory operations is specified in ITU-T Rec. X.519 | ISO/IEC 9594-5.

7 Information types and common procedures

7.1 Introduction

This clause 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.

Several of the information types used in the definition of the Directory service are actually defined elsewhere. Subclause 7.2 identifies these types and indicates the source of their definition. Each of the remaining subclauses (7.3 through 7.11) identifies and defines an information type.

This clause also specifies some common elements of procedure that apply to most or all of the Directory operations.

7.2 Information types defined elsewhere

The following information types are defined in ITU-T Rec. X.501 | ISO/IEC 9594-2:

a) **Attribute**;

b) **AttributeType**;

c) **AttributeValue**;

d) **AttributeValueAssertion**;

e) **DistinguishedName**;

f) **Name**;

g) **RelativeDistinguishedName**.

The following information type is defined in ITU-T Rec. X.520 | ISO/IEC 9594-6:

– **PresentationAddress**.

The following information types are defined in ITU-T Rec. X.509 | ISO/IEC 9594-8:

a) **Certificate**;

b) **SIGNED**;

c) **CertificationPath**.

The following information types are defined in ITU-T Rec. X.880 | ISO/IEC 9072-1:

– **InvokeID**.

The following information types are defined in ITU-T Rec. X.518 | ISO/IEC 9594-4:

a) **OperationProgress**;

b) **ContinuationReference**.

7.3 Common arguments

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

```
CommonArguments ::= SET {
    serviceControls [30] ServiceControls DEFAULT {},
    securityParameters [29] SecurityParameters OPTIONAL,
    requestor [28] DistinguishedName OPTIONAL,
    operationProgress [27] OperationProgress
                DEFAULT { nameResolutionPhase notStarted },
    aliasedRDNs [26] INTEGER OPTIONAL,
    criticalExtensions [25] BIT STRING OPTIONAL,
    referenceType [24] ReferenceType OPTIONAL,
    entryOnly [23] BOOLEAN DEFAULT TRUE,
    exclusions [22] Exclusions OPTIONAL,
    nameResolveOnMaster
                [21] BOOLEAN DEFAULT FALSE }
```

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

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

The **requestor** Distinguished Name identifies the originator of a particular 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.

The **operationProgress**, **referenceType**, **entryOnly**, **exclusions**, and **nameResolveOnMaster** components are defined in ITU-T Rec. X.518 | ISO/IEC 9594-4. They are supplied by a DUA only when acting on a continuation reference returned by a DSA in response to an earlier operation, and their values are copied by the DUA from the continuation reference.

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 name that came from dereferencing the alias. (The value would have been set in the referral response of the previous operation.)

NOTE – This component is provided for compatibility with 1988 edition implementations of the Directory. DUAs (and DSAs) implemented according to later editons of the Directory specifications shall always omit this parameter from the **CommonArguments** of a subsequent request. In this way, the Directory will not signal an error if aliases dereference to further aliases.

7.3.1 Critical extensions

The **criticalExtensions** component provides a mechanism to list a set of extensions which are critical to the performance of a Directory operation. If the originator of the extended operation wishes to indicate that the operation shall be performed with one or more extensions (i.e. that performing the operation without these extensions is not acceptable), it does so by setting the **criticalExtensions** bit(s) which corresponds to the extension(s). 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.

This document defines a number of extensions which are available to 1993 edition implementations of the Directory. The extensions take such forms as additional numbered bits in a BIT STRING, or additional components of a SET or SEQUENCE, and are ignored by 1988 edition systems. Each such extension is assigned an integer *identifier*, which is the number of the bit which may be set in **criticalExtensions**. If the criticality of an extension is defined to be critical, the DUA shall set the corresponding bit in **criticalExtensions**. If the defined criticality is non-critical, the DUA may or may not set the corresponding bit in **criticalExtensions**.

The extensions, their identifiers, the operations in which they are permitted, the recommended criticality, and the clauses in which they are defined are shown in Table 1.

Table 1 – Extensions

Extension	Identifier	Operations	Criticality	Defined (subclauses)
subentries	1	all	non-critical	7.5
copyShallDo	2	Read, Compare, List, Search	non-critical	7.5
attribute size limit	3	Read, Search	non-critical	7.5
extraAttributes	4	Read, Search	non-critical	7.6
modifyRightsRequest	5	Read	non-critical	9.1
pagedResultsRequest	6	List, Search	non-critical	10.1
matchedValuesOnly	7	Search	non-critical	10.2
extendedFilter	8	Search	non-critical	10.2
targetSystem	9	AddEntry	critical	11.1
useAliasOnUpdate	10	AddEntry, RemoveEntry, ModifyEntry	critical	11.1
newSuperior	11	ModifyDN	critical	11.4

7.4 Common results

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

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

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

The **performer** Distinguished Name 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.

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 any aliases which were dereferenced.

7.5 Service controls

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),
        subentries (5),
        copyShallDo (6) } 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,
    attributeSizeLimit [5] INTEGER OPTIONAL }
```

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 ITU-T Rec. X.518 | ISO/IEC 9594-4 shall not be used to provide the service.

e) **dontDereferenceAliases** indicates that any alias used to identify the entry affected by an operation is not to be dereferenced.

NOTE 1 – 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.

f) **subentries** indicates that a **Search** or **List** operation is to access subentries only; normal entries become inaccessible – i.e. the Directory behaves as though normal entries do not exist. If this service control is not set, then the operation accesses normal entries only and subentries become inaccessible. The service control is ignored for operations other than **Search** or **List**.

NOTE 2 – The effects of subentries on access control, schema, and collective attributes are still observed even if subentries are inaccessible.

NOTE 3 – If this service control is set, normal entries may still be specified as the base object of an operation.

g) **copyShallDo** indicates that if the Directory is able to partly but not fully satisfy a query at a copy of an entry, it shall not chain the query. It is meaningful only if **dontUseCopy** is not set. If **copyShallDo** is not set, the Directory will use shadow data only if it is sufficiently complete to allow the operation to be fully satisfied at the copy. A query may be only partly satisfied because some of the requested attributes are missing in the shadow copy, or because the DSA holding the shadowed data does not support the requested matching rules on that data. If **copyShallDo** is set and the Directory is not able to fully satisfy a query, it shall set **incompleteEntry** in the the returned entry information.

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, aliases shall be dereferenced (except for modify operations for which alias dereferencing is not supported), subentries are not accessible, and operations not fully satisfiable with shadowed data are subject to further chaining.

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

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 4 – 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.

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.

The **scopeOfReferral** indicates the scope to which a referral returned by a DSA should be relevant. Depending on whether the values **dmd** or **country** are selected, only referrals to other DSAs within the selected scope shall be returned. This applies to the referrals in both a **Referral** error and the **unexplored** parameter of List and Search results.

The **attributeSizeLimit** indicates the largest size of any attribute (i.e. the type and all its values) that is included in returned entry information. If an attribute exceeds this limit, all of its values are omitted from the returned entry information and **incompleteEntry** is set in the returned entry information. The size of an attribute is taken to be its size in octets in the local concrete syntax of the DSA holding the data. Because of different ways applications store the data, the limit is imprecise. If this parameter is not specified, no limit is implied.

NOTE 5 – Attribute values returned as part of an entry's Distinguished Name are exempt from this limit.

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

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

```
EntryInformationSelection ::= SET {
    attributes CHOICE {
        allUserAttributes [0] NULL,
        select [1] SET OF AttributeType
        -- empty set implies no attributes are requested -- } DEFAULT allUserAttributes : NULL,
    infoTypes [2] INTEGER {
        attributeTypesOnly (0),
        attributeTypesAndValues (1) } DEFAULT attributeTypesAndValues,
    extraAttributes CHOICE {
        allOperationalAttributes [3] NULL,
        select [4] SET OF AttributeType } OPTIONAL }
```

The **attributes** component specifies the user and operational 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 shall be returned. Information about a selected attribute shall be returned if the attribute is present. An **AttributeError** with the **noSuchAttributeOrValue** problem shall only be returned if none of the attributes selected is present.

b) If the **allUserAttributes** option is selected, then information is requested about all user attributes in the entry.

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

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

The **extraAttributes** component specifies a set of additional user and operational attributes about which information is requested. If the **allOperationalAttributes** option is chosen, then information is requested about all directory operational attributes in the entry. If the **select** option is chosen, then information about the listed attributes is requested.

NOTE – This component may be used to request information about, for example, specific operational attributes when **attributes** is set to **allUserAttributes**, or about all operational attributes. If the same attribute is listed or implied in both **attributes** and **extraAttributes**, it is treated as though it has been requested only once.

A request for a particular attribute is always treated as a request for the attribute and all *subtypes* of that attribute (except for requests processed by1988-edition systems).

In responding to a request for attribute information, the Directory treats all *collective attributes* of an entry as if they were actual user attributes of the entry, i.e. they are selected like other user attributes and are merged into the returned entry information. A request for **allUserAttributes** requests all collective attributes of the entry as well as ordinary attributes of the entry. An attribute is a collective attribute of an entry if all of the following are true:

a) it is located in a subentry whose subtree specification includes the entry;

b) it is not excluded by the presence in the entry of a **collectiveExclusions** attribute value equal to the collective attribute type; and

c) it is permitted by the content rule for the structural object class for the entry.

7.7 Entry information

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

```
EntryInformation    ::=    SEQUENCE {
    name                        Name,
    fromEntry                   BOOLEAN DEFAULT TRUE,
    information                 SET OF CHOICE {
        attributeType               AttributeType,
        attribute                   Attribute } OPTIONAL,
    incompleteEntry     [3]     BOOLEAN DEFAULT FALSE -- not in 1988-edition systems -- }
```

The **Name** parameter indicates the Distinguished Name of the entry or the name of an alias to the entry. The Distinguished Name of the entry is returned whenever permitted by the access control policy. If access is allowed to the attributes of the entry but not to its Distinguished Name, the Directory may return either an error or the name of a valid alias to the entry.

NOTES

1 If the entry was located using an alias, then that alias is known to be a valid alias. Otherwise, how it is ensured that the alias is valid is outside the scope of these Directory Specifications.

2 Where a particular component of the Directory has a choice of alias names available to it for return, it is recommended that where possible it choose the same alias name for repeated requests by the same requestor, in order to provide a consistent service.

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

The **information** parameter is included if any attribute information from the entry is being returned, and contains a set of **attributeType**s and **attribute**s, as appropriate.

The **incompleteEntry** parameter is included and set to **TRUE** whenever the returned entry information is incomplete in relation to the user's request, e.g. because attributes or attribute values are omitted for reasons of access control (and their existence is permitted to be disclosed), the presence of incomplete shadow information together with **copyShallDo**, or because the **attributeSizeLimit** has been exceeded. It is not set to **TRUE** because an alias name has been returned instead of the Distinguished Name.

7.8 Filter

7.8.1 Filter

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                            ATTRIBUTE.&id({SupportedAttributes}),
                strings                         SEQUENCE OF CHOICE {
                        initial                         [0] ATTRIBUTE.&Type
                                                            ({SupportedAttributes}{@substrings.type}),
                        any                             [1] ATTRIBUTE.&Type
                                                            ({SupportedAttributes}{@substrings.type}),
                        final                           [2] ATTRIBUTE.&Type
                                                            ({SupportedAttributes}{@substrings.type})}},
        greaterOrEqual          [2]     AttributeValueAssertion,
        lessOrEqual             [3]     AttributeValueAssertion,
        present                 [4]     AttributeType,
        approximateMatch        [5]     AttributeValueAssertion,
        extensibleMatch         [6]     MatchingRuleAssertion }

MatchingRuleAssertion ::= SEQUENCE {
        matchingRule            [1]     SET SIZE (1..MAX) OF MATCHING-RULE.&id,
        type                    [2]     AttributeType  OPTIONAL,
        matchValue              [3]     MATCHING-RULE.&AssertionType ( CONSTRAINED BY {
                – matchValue must be a value of  type specified by the &AssertionType field of
                -- one of the MATCHING-RULE information objects identified by matchingRule -- } ),
        dnAttributes            [4]     BOOLEAN DEFAULT FALSE }
```

A **Filter** is either a **FilterItem** (see 7.8.2), or an expression involving simpler filters composed together with the logical operators **and**, **or**, and **not**.

A **Filter** which is a **FilterItem** has the value of the **FilterItem** (i.e. TRUE, FALSE, or undefined).

A **Filter** which is the **and** of a set of filters is TRUE if the set is empty or if each filter is TRUE; it is FALSE if at least one filter is FALSE; otherwise it is undefined (i.e. if at least one filter is undefined and no filters are FALSE).

A **Filter** which is the **or** of a set of filters is FALSE if the set is empty or if each filter is FALSE; it is TRUE if at least one filter is TRUE; otherwise it is undefined (i.e. if at least one filter is undefined and no filters are TRUE).

A **Filter** which is the **not** of a filter is TRUE if the filter is FALSE; FALSE if it is TRUE; and undefined if it is undefined.

7.8.2 Filter item

A **FilterItem** is an assertion about the presence or value(s) of attributes in the entry under test. An assertion about a particular attribute type is also satisfied if the entry contains a subtype of the attribute and the assertion is TRUE for the subtype, or if there is a collective attribute of the entry (see 7.6) for which the assertion is TRUE. Each assertion is TRUE, FALSE, or undefined.

Every **FilterItem** includes or implies one or more **AttributeType**s which identifies the particular attribute(s) concerned.

Any assertion about the values of such an attribute is only defined if the **AttributeType** is known by the evaluating mechanism, the purported **AttributeValue**(s) conforms to the attribute syntax defined for that attribute type, the implied or indicated matching rule is applicable to that attribute type, and (when used) a presented **matchValue** conforms to the syntax defined for the indicated matching rules.

NOTE 1 – Where these conditions are not met the **FilterItem** is undefined.

NOTE 2 – Access control restrictions may affect the evaluation of the **FilterItem**.

Attribute value assertions in filter items are evaluated using the matching rules defined for that attribute type. Matching rule assertions are evaluated as specified in their definition. A matching rule defined for a particular syntax can only be used to make assertions about attributes of that syntax or subtypes of that syntax.

A **FilterItem** may be undefined (as described above). Otherwise, where the **FilterItem** asserts:

a) **equality** – It is TRUE if and only if there is a value of the attribute or one of its subtypes for which the **equality** matching rule applied to that value and the presented value returns TRUE.

b) **substrings** – It is TRUE if and only if there is a value of the attribute or one of its subtypes for which the **substring** matching rule applied to that value and the presented value in **strings** returns TRUE. See ITU-T Rec. X.520 | ISO/IEC 9594-6 for a description of the semantics of the presented value.

c) **greaterOrEqual** – It is TRUE if and only if there is a value of the attribute or one of its subtypes for which the **ordering** matching rule applied to that value and the presented value returns FALSE. I.E., there is a value of the attribute which is *greater than or equal to* the presented value.

d) **lessOrEqual** – It is TRUE if and only if there is a value of the attribute or one of its subtypes for which either the **equality** matching rule *or* the **ordering** matching rule applied to that value and the presented value returns TRUE, i.e. there is a value of the attribute which is *less than or equal to* the presented value.

e) **present** – It is TRUE if and only if the attribute or one of its subtypes is present in the entry.

f) **approximateMatch** – It is TRUE if and only if there is a value of the attribute or one of its subtypes for which some locally-defined approximate matching algorithm (e.g. spelling variations, phonetic match, etc.) returns TRUE. There are no specific guidelines for approximate matching in this edition of this Directory Specification. If approximate matching is not supported, this **FilterItem** should be treated as a match for **equality**.

g) **extensibleMatch** – It is TRUE if and only if there is a value of the attribute with the indicated **type** or one of its subtypes for which the matching rule specified in **matchingRule** applied to that value and the presented value **matchValue** returns TRUE.

If several matching rules are given, the way in which these rules are combined into a new rule is unspecified (it is a locally-defined algorithm, which reflects the semantics of the constituent matching rules – e.g. **phonetic** + **keyword** match).

If **type** is omitted, the match is made against all attribute types which are compatible with that matching rule. If **dnAttributes** is TRUE, the attributes of the Distinguished Name of the entry are used in addition to those of the entry in evaluating the match.

If an **extensibleMatch** is requested in a **filter** (rather than an **extendedFilter**), the **extendedFilter** bit in the **criticalExtensions** parameter in **CommonArguments** shall be set, indicating that the extension is critical.

NOTE 3 – An **extensibleMatch** is not permitted for 1988-edition systems.

7.9 Paged results

A **PagedResultsRequest** parameter is used by the DUA to request that the results of a list or search operation be returned to it "page-by-page": It requests the DSA to return only a subset – a *page* – of the results of the operation, in particular the next **pageSize** subordinates or entries, and to return a **queryReference** which can be used to request the next set of results on a follow-up query. It shall not be used if results are to be signed, and is not supported by 1988-edition systems. Although a DUA may request **pagedResults**, a DSA is permitted to ignore the request and return its results in the normal manner.

```
PagedResultsRequest ::= CHOICE {
    newRequest          SEQUENCE {
        pageSize                    INTEGER,
        sortKeys                    SEQUENCE OF SortKey OPTIONAL,
        reverse                [1]  BOOLEAN DEFAULT FALSE,
        unmerged               [2]  BOOLEAN DEFAULT FALSE },
    queryReference      OCTET STRING }

SortKey ::= SEQUENCE {
    type                AttributeType,
    orderingRule        MATCHING-RULE.&id OPTIONAL }
```

For a new list or search operation, the **PagedResultsRequest** is set to **newRequest**, which consists of the following parameters:

a) The **pageSize** parameter specifies the maximum number of subordinates or entries to return in the results. The DSA shall return up to but not more than the requested number of entries. The **sizeLimit**, if any, is ignored.

b) The **sortKeys** parameter specifies a sequence of attribute types with optional ordering matching rules to use as sort keys for sorting the returned entries prior to return to the DUA. The entries are sorted according to their values of the **type** attribute of the first **SortKey** in the sequence, and in the event of multiple entries having the same sort position, of the next **SortKey** in the sequence, and so on.

For a particular **SortKey**, the DSA uses the **orderingRule** matching rule if it is present, otherwise the **ordering** matching rule of the attribute if one is defined; it ignores the sort key if none is defined. If the attribute type is multivalued, the "least" value is used; if the attribute type is missing from the returned results, it is regarded as "greater" than all other matched values. A DSA is permitted to support only certain sort key sequences (thus, a DSA that holds and returns its data in the internal order "alphabetic by surname" will be able to comply with only one sort key sequence). If it cannot support the requested sequence, it shall use a default sort sequence.

c) If the **reverse** parameter is TRUE then the DSA will return the sorted results in reverse order (i.e. from "greatest" to "least" – if the attribute type is multivalued, the "greatest" is used; if the attribute type is missing from the returned results, it is regarded as "less" than all other matched values). If it is false, the DSA returns them in forward order. If no **sortKeys** parameter is specified, this parameter is ignored.

d) If the **unmerged** parameter is TRUE and the DSA must merge results from a number of other DSAs, it shall return all the data from one DSA (in sort order) before returning data from the next DSA. If the parameter is false, the DSA shall collect the results from all other DSAs and sort the merged data before returning any of it. If no **sortKeys** parameter is specified, this parameter is ignored.

For a followup request, i.e. to request the next set of paged results, the DUA makes the same list or search request as before, but sets **PagedResultsRequest** to **queryReference**, with the value of this parameter the same as that returned in the **PartialOutcomeQualifier** of the previous results. The DUA has no understanding of the **queryReference**, which is available to a DSA to use as it wishes to record context information for the query. The DSA uses this information to determine which results to return next.

NOTES

1 If the DIB changes between search requests, the DUA may not see the effects of these changes. This is implementation dependent.

2 A query-reference may remain valid even if a DUA begins a new list or search operation. A DUA may request paged results with several queries and then return to an earlier query and request the next page of results using the query-reference supplied for it. The number of "active" query references to which a DUA can return is a local DSA implementation option, as is the lifetime of those query-references.

3 Paged results are not supported in the Directory System Protocol. Paged results are provided entirely by the DSA to which the DUA has connected.

7.10 Security parameters

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 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) }
```

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.

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.

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.

The **random** number 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.

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, the default), and **signed** (the Directory is requested to sign the result). 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.

7.11 Common elements of procedure for basic-access-control

This subclause defines the elements of procedure that are common to all abstract service operations when **basic-access-control** is in effect.

7.11.1 Alias dereferencing

If, in the process of locating a target object entry (identified in the argument of an abstract service operation), alias dereferencing is required, no specific permissions are necessary for alias dereferencing to take place. However, if alias dereferencing would result in a **ContinuationReference** being returned (i.e. in a **Referral**), the following sequence of access controls applies. These access controls shall also be applied to a referral that is received in a response from another DSA. That is, the DSA shall police all referrals whether they were generated locally or remotely.

1) *Read* permission is required to the alias entry. If permission is not granted, the operation fails in accordance to the procedure described in 7.11.3.

2) *Read* permission is required to the **AliasedObjectName** attribute and to the single value that it contains. If permission is not granted the operation fails and the error **NameError** with problem **aliasDereferencingProblem** shall be returned. The **matched** element shall contain the name of the alias entry.

NOTE – In addition to the access controls described above, security policy may prevent the disclosure of knowledge information which would otherwise be conveyed as a **ContinuationReference** in **Referral**. If such a policy is in effect and if a DUA constrains the service by specifying **chainingProhibited** the Directory may return a **ServiceError** with problem **chainingRequired**. Otherwise, a **SecurityError** with problem **insufficientAccessRights** or **noInformation** shall be returned.

7.11.2 Return of NameError

If, while performing an abstract service operation, the specified target object (alias or entry) – e.g. the Name of an entry to be read or the **baseObject** in a **Search** – could not be found, a **NameError** with problem **noSuchObject** shall be returned. The **matched** element shall either contain the name of the next superior entry to which *DiscloseOnError* permission is granted, or the name of the DIT root (i.e. an empty **RDNSequence**).

NOTE – The second alternative may be taken by a DSA which does not have access to all superior entries.

7.11.3 Non-disclosure of the existence of an entry

If, while performing an abstract service operation, the necessary entry level permission is not granted to the specified target object entry – e.g. the entry to be read – the operation fails and the error returned is one of: if *DiscloseOnError* permission is granted to the target entry, a **SecurityError** with problem **insufficientAccessRights** or **noInformation** shall be returned; otherwise, a **NameError** with problem **noSuchObject** shall be returned. The **matched** element shall either contain the name of the next superior entry to which *DiscloseOnError* permission is granted, or the name of the DIT root (i.e. an empty **RDNSequence**).

NOTE – The second alternative may be taken by a DSA which does not have access to all superior entries.

Additionally, whenever the Directory detects an operational error (including a **Referral**) it shall ensure that in returning that error it does not compromise the existence of the named target entry and any of its superiors. For example, before returning a **ServiceError** with problem **timeLimitExceeded** or an **UpdateError** with problem **notAllowedOnNonLeaf**, the Directory verifies that *discloseOnError* permission is granted to the target entry. If it is not, the procedure described in the paragraph above shall be followed.

7.11.4 Return of Distinguished Name

In a Compare, List, or Search operation, *ReturnDN* permission is required to the **object** (or **baseObject**) entry if as a result of dereferencing an alias, the object's distinguished name is to be returned in the **name** parameter of the operation result (see 9.2.3). If this permission is not granted, the Directory shall return an alias name for the entry instead, as described in 7.7, or shall omit the name parameter altogether.

In a Read or Search operation, *ReturnDN* permission is required to an entry in order to return its distinguished name in **EntryInformation**. If this permission is not granted, the Directory shall return the name of an alias instead, as described in 7.7, or if no alias name is available shall fail the operation with a **NameError** (in the case of Read) or omit the entry from the results (in the case of Search).

If the user supplied alias name is returned in the result, then the **aliasDeferenced** flag of **CommonResults** shall not be set to **TRUE**.

7.12 Optionally-signed parameters

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 type:

```
OPTIONALLY-SIGNED {Type}    ::=    CHOICE {
        unsigned        Type,
        signed          SIGNED {Type}}
```

The **SIGNED** type, which describes the form of the signed form of the information, is specified in ITU-T Rec. X.509 | ISO/IEC 9594-8.

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 Directory Bind syntax

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

```
directoryBind       OPERATION    ::=    {
        ARGUMENT            DirectoryBindArgument
        RESULT              DirectoryBindResult
        ERRORS              {directoryBindError }}

DirectoryBindArgument       ::=         SET {
        credentials         [0]         Credentials OPTIONAL,
        versions            [1]         Versions DEFAULT {v1}}

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]         CHOICE {
                unprotected                     OCTET STRING,
                protected                       SIGNATURE {OCTET STRING} } OPTIONAL}

StrongCredentials    ::=    SET {
        certification-path  [0]         CertificationPath OPTIONAL,
        bind-token          [1]         Token,
        name                [2]         DistinguishedName OPTIONAL }

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

Versions    ::=     BIT STRING {v1(0)}

DirectoryBindResult         ::=         DirectoryBindArgument
```

```
directoryBindError  ERROR       ::=     {
    PARAMETER       SET {
        versions        [0]     Versions DEFAULT {v1},
        error                   CHOICE {
        serviceError            [1]     ServiceProblem,
        securityError           [2]     SecurityProblem }}}
```

8.1.2 Directory Bind arguments

The **credentials** argument of the **DirectoryBindArgument** allows the Directory to establish the identity of the user. The credentials may be **simple**, or **strong** or externally defined (**externalProcedure**) (as described in ITU-T Rec. X.509 | ISO/IEC 9594-8.

If **simple** is used, it consists of a **name** (always the distinguished name of an object), an optional **validity**, and an optional **password**. This provides a limited degree of security. The **password** may be **unprotected**, or it may be **protected** (either Protected1 or Protected2) as described in clause 5 of ITU-T Rec. X.509 | ISO/IEC 9594-8. The **validity** supplies **time1**, **time2**, **random1** and **random2** arguments, which derive their meaning by bilateral agreement, and which may be 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 comparing the result with the value in the bind argument (**password**). In other instances a direct comparison may be possible.

If **strong** is used, it consists of a **bind-token**, and, optionally, a **certification-path** (certificate and sequence of certification-authority cross-certificates, as defined in ITU-T Rec. X.509 | ISO/IEC 9594-8) and the **name** of the requestor. This enables the Directory to authenticate the identity of the requestor 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 this 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.

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

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 v1(0).

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

a) Any elements of **DirectoryBindArgument** other than those defined in this Directory Specification 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 Directory Bind results

Should the bind request succeed, a result shall be returned.

The **credentials** argument of the **DirectoryBindResult** allows the user to establish the identity of the Directory. It allows information identifying the DSA (that is directly providing the Directory service) to be conveyed to the DUA. It shall be of the same form (i.e. **CHOICE**) as that supplied by the user.

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 the DSA.

8.1.4 Directory Bind errors

Should the bind request fail, a bind error shall be returned.

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

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

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

8.2 Directory Unbind

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

```
directoryUnbind    OPERATION    ::=    emptyUnbind
```

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 these operations for convenience.

9.1 Read

9.1.1 Read syntax

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    OPERATION  ::=    {
        ARGUMENT           ReadArgument
        RESULT             ReadResult
        ERRORS             { attributeError | nameError | serviceError | referral | abandoned |
                           securityError }
        CODE               id-opcode-read }

ReadArgument       ::=    OPTIONALLY-SIGNED { SET {
        object             [0]    Name,
        selection          [1]    EntryInformationSelection  DEFAULT { },
        modifyRightsRequest
                           [2]    BOOLEAN DEFAULT FALSE,
        COMPONENTS OF             CommonArguments }}

ReadResult    ::=    OPTIONALLY-SIGNED { SET {
        entry              [0]    EntryInformation,
        modifyRights       [1]    ModifyRights OPTIONAL,
        COMPONENTS OF             CommonResults }}

ModifyRights       ::=    SET OF SEQUENCE {
        item               CHOICE {
                entry      [0]    NULL,
                attribute  [1]    AttributeType,
                value      [2]    AttributeValueAssertion },
        permission         [3]    BIT STRING { add (0), remove (1), rename (2) , move(3) }}
```

9.1.2 Read arguments

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

The **selection** argument indicates what information from the entry is requested (see 7.6). However, it should not be assumed that the attributes returned are the same as or limited to those requested.

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.

The **modifyRightsRequest** argument is used to request return of the requestor's modification rights to the entry and its attributes.

9.1.3 Read results

Should the request succeed, the result shall be returned.

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

The **modifyRights** parameter is present if it was requested via the **modifyRightsRequest** argument, and the user has modification privileges to some or all of the requested entry information, and the return of this information is permitted by the local security policy. If returned, the modification rights of the requestor are returned for the entry and for the attributes specified in the **selection** argument. The parameter contains the following:

– An element of the SET is returned for the **entry**; for each user **attribute** requested which the user has the right to add or remove; and for each returned attribute **value** for which the user's rights to add or remove it differ from those of the corresponding attribute.

– The returned **permission** indicates what operations or actions on the entry by the user would succeed. In the case of an entry, **remove** indicates that a **RemoveEntry** operation would succeed; **rename** indicates that a **ModifyDN** operation with the **newSuperior** parameter absent would succeed; and **move** that a **ModifyDN** operation with the **newSuperior** parameter present and an unchanged RDN would succeed.

  In the case of attributes and values, **add** indicates that a **ModifyEntry** operation that adds the attribute or value would succeed; and **remove** indicates that a **ModifyEntry** operation that removes the attribute or value would succeed.

  NOTE – An operation to move an entry to a new superior may also depend on permissions associated with the new superior (as for example with **basic-access-control**). These are ignored when determining **permission**.

9.1.4 Read errors

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

9.1.5 Read operation decision points for basic-access-control

If **basic-access-control** is in effect for the entry being read, the following sequence of access controls applies.

1) *Read* permission is required to the entry being read. If permission is not granted, the operation fails in accordance with 7.11.3.

2) If the **infoTypes** element of **selection** specifies that attribute types only are to be returned, then for each attribute type that is to be returned, *Read* permission is required. If permission is not granted, the attribute type is omitted from the **ReadResult**. If as a consequence of applying these controls no attribute information is returned, the entire operation fails in accordance with 9.1.5.1.

3) If the **infoTypes** element of **selection** specifies that attribute types and values are to be returned, then for each attribute type and for each value that is to be returned, *Read* permission is required. If permission to an attribute type is not granted, the attribute is omitted from **ReadResult**. If permission to an attribute value is not granted, the value is omitted from its corresponding attribute. In the event that permission is not granted to any of the values within the attribute, an **Attribute** element containing an empty **SET OF AttributeValue** is returned. If as a consequence of applying these controls no attribute information is returned, the entire operation fails in accordance with 9.1.5.1.

9.1.5.1 Error returns

If the operation fails as defined in 9.1.5 items 2) or 3), the valid error returns are one of:

a) If an open-ended option was specified (i.e. **allUserAttributes** or **allOperationalAttributes**), a **SecurityError** with problem **insufficientAccessRights** or **noInformation** shall be returned.

b) Otherwise, if a **select** option was specified (in **attributes** and/or in **extraAttributes**), then if the *DiscloseOnError* permission is granted to any of the selected attributes a **SecurityError** with problem **insufficientAccessRights** or **noInformation** shall be returned. Otherwise, an **AttributeError** with problem **noSuchAttributeOrValue** shall be returned.

9.1.5.2 Non-disclosure of incomplete results

If an incomplete result is being returned in **EntryInformation**, i.e. some of the attributes or attribute values have been omitted because of the applicable access controls, the **incompleteEntry** element shall be set to TRUE if *DiscloseOnError* permission is granted to at least one attribute type withheld from the result, or at least one attribute value withheld from the result (for which attribute type *Read* permission was granted).

9.2 Compare

9.2.1 Compare syntax

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 OPERATION ::= {
        ARGUMENT        CompareArgument
        RESULT          CompareResult
        ERRORS          { attributeError | nameError | serviceError | referral | abandoned |
                        securityError }
        CODE            id-opcode-compare }

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

CompareResult ::=       OPTIONALLY-SIGNED { SET {
        name                Name  OPTIONAL,
        matched         [0] BOOLEAN,
        fromEntry       [1] BOOLEAN DEFAULT TRUE,
        matchedSubtype  [2] AttributeType OPTIONAL,
        COMPONENTS OF       CommonResults }}
```

9.2.2 Compare arguments

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

The **purported** argument identifies the attribute type and value to be compared with that in the entry. The comparison is TRUE if the entry holds the purported attribute type or one of its subtypes, or there is a collective attribute of the entry which is the purported attribute type or one of its subtypes (see 7.6), and if there is a value of that attribute which matches the purported value using the attribute's **equality** matching rule.

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 Compare results

Should the request succeed (i.e. the comparison is actually carried out), the result shall be returned.

The **name** is the distinguished name of the entry or an alias name of the entry, as described in 7.7. It is present only if an alias has been dereferenced and the name to be returned differs from the **object** name supplied in the operation argument.

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.

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

The **matchedSubtype** parameter is present only if the result of the match was TRUE and if the match succeeded because a subtype of the purported attribute was matched. It contains the matched subtype. If more than one such subtype is available, the one highest in the hierarchy is returned.

9.2.4 Compare errors

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

9.2.5 Compare operation decision points for basic-access-control

If **basic-access-control** is in effect for the entry being compared, the following sequence of access controls applies.

1) *Read* permission is required to the entry to be compared. If permission is not granted, the operation fails in accordance with 7.11.3.
2) *Compare* permission is required to the attribute being compared. If permission is not granted, the operation fails in accordance to 9.2.5.1**.**

3) If there exists a value within the attribute being compared that matches the **purported** argument and for which *Compare* permission is granted, the operation returns the value TRUE in the **matched** result parameter of the **CompareResult**. Otherwise, the operation returns the value FALSE.

9.2.5.1 Error returns

If the operation fails as defined in 9.2.5 item 2), the valid error returns are one of: if the *DiscloseOnError* permission is granted to the attribute being compared, a **SecurityError** with problem **insufficientAccessRights** or **noInformation** shall be returned; otherwise, an **AttributeError** with problem **noSuchAttributeOrValue** shall be returned.

9.3 Abandon

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

```
abandon     OPERATION     ::=     {
        ARGUMENT              AbandonArgument
        RESULT                AbandonResult
        ERRORS                { abandonFailed }
        CODE                  id-opcode-abandon }

AbandonArgument               ::=     SEQUENCE {
        invokeID              [0]     InvokeId}

AbandonResult       ::=       NULL
```

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.

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

Should the request fail, the **AbandonFailed** error shall be reported. As a local matter, a DSA may choose not to abandon the operation and shall then return the **AbandonFailed** error. This error is described in 12.3.

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

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.

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 List syntax

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    OPERATION  ::=     {
        ARGUMENT              ListArgument
        RESULT                ListResult
        ERRORS                { nameError | serviceError | referral | abandoned | securityError }
        CODE                  id-opcode-list }

ListArgument          ::=     OPTIONALLY-SIGNED { SET {
        object                [0]     Name,
        pagedResults          [1]     PagedResultsRequest OPTIONAL,
        COMPONENTS OF                 CommonArguments }}
```

```
ListResult          ::=  OPTIONALLY-SIGNED { CHOICE {
    listInfo                    SET {
        name                                Name  OPTIONAL,
        subordinates            [1]         SET OF SEQUENCE {
            rdn                                     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,
    unknownErrors           [3]     SET OF ABSTRACT-SYNTAX.&Type  OPTIONAL,
    queryReference          [4]     OCTET STRING  OPTIONAL }

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

10.1.2 List arguments

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).

The **pagedResults** argument is used to request that results of the operation be returned page-by-page, as described in 7.9.

10.1.3 List results

The request succeeds if the **object** is located, regardless of whether there is any subordinate information to return.

The **name** is the distinguished name of the entry or an alias name of the entry, as described in 7.7. It is present only if an alias has been dereferenced and the name to be returned differs from the **baseObject** name supplied in the operation argument.

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

The **rdn** parameter is the relative distinguished name of the subordinate.

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

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

The **partialOutcomeQualifier** consists of five subcomponents as described below. This parameter shall be present whenever the result is incomplete because of a time limit, size limit, or administrative limit problem, because regions of the DIT were not explored, because some critical extensions were unavailable, because an unknown error was received, or because paged results are being returned.

a) 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.

b) 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 **ContinuationReference**s, 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 **ContinuationReference**s that are returned shall be within the scope of referral requested in the operation service control.

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

d) The **unknownErrors** parameter is used to return unknown error types or parameters received from other DSAs in the processing of the operation. Each member of the SET contains one such unknown error. See ITU-T Rec. X.519 | ISO/IEC 9594-5 7.5.2.4.

e) The **queryReference** parameter shall be present when the DUA has requested paged results and the DSA has not returned all the available results. See 7.9

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 List errors

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

10.1.5 List operation decision points for basic-access-control

If **basic-access-control** is in effect for the portion of the DIB where the **List** operation is being performed, the following sequence of access controls applies.

1) No specific permission is required to the entry identified by the **object** argument.

2) For each immediate subordinate for which a **RelativeDistinguishedName** is to be returned in **subordinates**, *Browse* and *ReturnDN* permissions are required to that entry. Entries for which these permissions are not granted are ignored. If as a consequence of applying these controls no subordinate information (excluding any **ContinuationReferences** in **PartialOutcomeQualifier**) is returned and if *DiscloseOnError* permission is not granted to the entry identified by the **object** argument, the operation fails and a **NameError** with problem **noSuchObject** shall be returned. The **matched** element shall either contain the name of the next superior entry to which *DiscloseOnError* permission is granted, or the name of the DIT root (i.e. an empty **RDNSequence**). Otherwise, the operation succeeds but no subordinate information (excluding any **ContinuationReferences** in **PartialOutcomeQualifier**) is conveyed with it.

NOTE 1 – In the case of a **NameError** being returned, the empty **RDNSequence** may be used by a DSA which does not have access to all superior entries.

NOTE 2 – Security policy may prevent the disclosure of subordinate information which would otherwise be conveyed as **ContinuationReferences** in **PartialOutcomeQualifier**. If such a policy is in effect and if a DUA constrains the service by specifying **chainingProhibited** the Directory may return a **ServiceError** with problem **chainingRequired**. Otherwise, the procedure described in item 2) above is followed.

NOTE 3 – Security policy may prevent the Directory from indicating that a listed subordinate entry is an alias entry. For example, if the DUA is not granted *Read* access to the alias entry, its **ObjectClass** attribute and the value **alias** that it contains the Directory may omit the **aliasEntry** component of **subordinates** from the **ListResult** or set it to FALSE.

NOTE 4 – If *DiscloseOnError* permission is not granted to the entry identified by the **object** argument, a **partialOutcomeQualifier** indicating a **limitProblem** or **unavailableCriticalExtensions** should not be returned as it may compromise the security of this entry.

10.2 Search

10.2.1 Search syntax

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          OPERATION   ::=     {
        ARGUMENT            SearchArgument
        RESULT              SearchResult
        ERRORS              { attributeError | nameError | serviceError | referral | abandoned |
                            securityError }
        CODE                id-opcode-search }

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 { },
        pagedResults            [5]     PagedResultsRequest OPTIONAL,
        matchedValuesOnly
                                [6]     BOOLEAN DEFAULT FALSE,
        extendedFilter          [7]     Filter OPTIONAL,
        COMPONENTS OF                   CommonArguments }}

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

10.2.2 Search arguments

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

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**).

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

NOTE 1 – If the filter is overspecified, it may eliminate all entries from the search result, even though there are candidate entries matching portions of the filter. The user must simplify the filter and try again. The Directory provides no support for identifying these entries, or for identifying the changes that should be made to the filter.

Aliases shall be dereferenced while locating the base object, subject to the setting of the **dontDereferenceAliases** service control. 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 entry.

The **selection** argument indicates what information from the entries is requested (see 7.6). However, it should not be assumed that the attributes returned are the same as or limited to those requested.

The **pagedResults** argument is used to request that results of the operation be returned page-by-page, as described in 7.9.

The **matchedValuesOnly** argument indicates that certain attribute values are to be omitted from the returned entry information. Specifically, where an attribute to be returned is multivalued, and some but not all of the values of that attribute contributed to the search filter returning TRUE via filter items other than **equality** or **present**, then the values that did not so contribute are omitted from the returned entry information.

The **extendedFilter** argument is used in mixed version environments to specify an alternative filter to that described above. When this argument is present, the **filter** argument (if any) shall be ignored by 1993-edition systems. The **extendedFilter** is always ignored by 1988-edition systems.

NOTE 2 – By including both filters, a DUA can specify one filter to be used by 1988-edition systems and a different filter to be used by 1993-edition systems in the distributed processing of the Search request. 1988-edition systems do not support attribute polymorphism or matching rule assertions.

10.2.3 Search results

The request succeeds if the **baseObject** is located, regardless of whether there are any subordinates to return.

NOTE 1 – 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 shall return an **AttributeError** if none of the selected attributes exist in the entry.

The **name** is the distinguished name of the entry or an alias name of the entry, as described in 7.7. It is present only if an alias has been dereferenced and the name to be returned differs from the **baseObject** name supplied in the operation argument.

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

The **partialOutcomeQualifier** is as described in 10.1.3.

NOTE 2 – Where returned entry information is incomplete for a particular entry, it is indicated via the **incompleteEntry** parameter in the returned entry information.

The **uncorrelatedSearchInfo** parameter is as described for **uncorrelatedListInfo** in 10.1.3.

10.2.4 Search errors

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

10.2.5 Search operation decision points for basic-access-control

If **basic-access-control** is in effect for the portion of the DIT to be searched, the following sequence of access controls applies:

1) No specific permission is required to the entry identified by the **baseObject** argument.

   NOTE 1 – If the **baseObject** is within the scope of the **SearchArgument** (i.e. when the **subset** argument specifies **baseObject** or **wholeSubtree**) the access controls specified in items 2) through 4) apply.

2) For each entry within the scope of the **SearchArgument** which is to be a candidate for consideration, *Browse* permission is required. Entries for which this permission is not granted are ignored.

3) The **filter** argument is applied to each entry left to be considered after taking item 2) into account, in accordance with the following:

   a) For each **FilterItem** which specifies an attribute, *FilterMatch* permission for the attribute type is required before the **FilterItem** can be evaluated as either TRUE or FALSE. A **FilterItem** for which this permission is not granted evaluates as undefined.

   b) For each **FilterItem** which additionally specifies an attribute value, *FilterMatch* permission is required for each stored attribute value which is to be considered for the purposes of matching. If there is a value which both matches the **FilterItem** and for which permission is granted, the **FilterItem** evaluates to **TRUE**, otherwise it evaluates to **FALSE**.

4) Once the procedures defined in 2) and 3) have been applied, the entry is either selected or discarded. If as a consequence of applying these controls to the entire scoped subtree no entries have been selected (excluding any **ContinuationReferences** in **partialOutcomeQualifier**) and if *DiscloseOnError* permission is not granted to the entry identified by the **baseObject** argument, the operation fails and a **NameError** with problem **noSuchObject** shall be returned. The **matched** element shall either contain the name of the next superior entry to which *DiscloseOnError* permission is granted, or the name of the DIT root (i.e. an empty **RDNSequence**). Otherwise, the operation succeeds but no subordinate information is conveyed with it.

   NOTE 2 – In the case of a **NameError** being returned, the empty **RDNSequence** may be used by a DSA which does not have access to all superior entries.

   NOTE 3 – Security policy may prevent the disclosure of knowledge information which would otherwise be conveyed as **ContinuationReferences** in **partialOutcomeQualifier**. If such a policy is in effect and if a DUA constrains the service by specifying **chainingProhibited** the Directory may return a **ServiceError** with problem **chainingRequired**. Otherwise, the **ContinuationReference** is omitted from **partialOutcomeQualifier**.

5) Otherwise, for each selected entry the information returned is as follows:

   a) If the **infoTypes** element of **selection** specifies that attribute types only are to be returned, then for each attribute type that is to be returned, *Read* permission is required. If permission is not granted, the attribute type is omitted from **EntryInformation**. If as a consequence of applying these controls no attribute type information is selected, the **EntryInformation** element is returned but no attribute type information is conveyed with it (i.e. the **SET OF CHOICE** element is omitted or empty).

   b) If the **infoTypes** element of **selection** specifies that attribute types and values are to be returned, then for each attribute type and for each value that is to be returned, *Read* permission is required. If permission to an attribute type is not granted, the attribute is omitted from **EntryInformation**. If permission to an attribute value is not granted, the value is omitted from its corresponding attribute. In the event that permission is not granted to any of the values within the attribute, an **Attribute** element containing an empty **SET OF AttributeValue** is returned. If as a consequence of applying these controls no attribute information is selected, the **EntryInformation** element is returned but no attribute information is conveyed with it (i.e. the **SET OF CHOICE** element is omitted or empty).

      NOTE 4 – If *DiscloseOnError* permission is not granted to the entry identified by the **baseObject** argument, a **partialOutcomeQualifier** indicating a **limitProblem** or **unavailableCriticalExtensions** should not be returned as it may compromise the security of this entry.

10.2.5.1 Alias dereferencing during Search

No specific permissions are necessary for alias dereferencing to take place in the course of a **Search** operation (subject to the **searchAliases** parameter being set to TRUE). However, for each alias entry encountered, if alias dereferencing would result in a **ContinuationReference** being returned in **partialOutcomeQualifier**, the following access controls apply: *Read* permission is required to the alias entry, the **AliasedObjectName** attribute and to the single value that it contains. If any of these permissions is not granted, the **ContinuationReference** shall be omitted from **partialOutcomeQualifier**. These access controls shall also be applied to a **continuationReference** that is received in a response from another DSA. That is, the DSA shall police all **continuationReference**s whether they were generated locally or not.

NOTE – In addition to the access controls described above, security policy may prevent the disclosure of knowledge information which would otherwise be conveyed as **ContinuationReferences** in **partialOutcomeQualifier**. If such a policy is in effect and if a DUA constrains the service by specifying **chainingProhibited** the Directory may return a **ServiceError** with problem **chainingRequired**. Otherwise, the **ContinuationReference** is omitted from **partialOutcomeQualifier**.

10.2.5.2 Non-disclosure of incomplete results

If an incomplete result is being returned in **EntryInformation**, i.e. some of the attributes or attribute values have been omitted because of the applicable access controls, the **incompleteEntry** element shall be set to TRUE if *DiscloseOnError* permission is granted to at least one attribute type withheld from the result, or at least one attribute value withheld from the result (for which attribute type *Read* permission was granted).

11 Directory Modify operations

There are four operations to modify the Directory: **addEntry**, **removeEntry**, **modifyEntry**, and **modifyDN** defined in 11.1 through 11.4, respectively.

NOTES

1 Each of these operations identifies the target entry by means of its distinguished name.

2 The success of **AddEntry**, **RemoveEntry**, and **ModifyDN** operations may depend on the physical distribution of the DIB across the Directory. Failure shall be reported with an **UpdateError** and problem **affectsMultipleDSAs**. See ITU-T Rec. X.518 | ISO/IEC 9594-4.

3 In the event of failure of the underlying communications mechanism, the outcome of the operations is undetermined. The user must use Directory interrogation operations to check whether the attempted modification operation succeeded or not.

11.1 Add Entry

11.1.1 Add Entry syntax

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      OPERATION    ::=     {
      ARGUMENT             AddEntryArgument
      RESULT               AddEntryResult
      ERRORS               { attributeError | nameError | serviceError | referral | securityError |
                           updateError }
      CODE                 id-opcode-addEntry }

AddEntryArgument           ::= OPTIONALLY-SIGNED { SET {
      object               [0]     Name,
      entry                [1]     SET OF Attribute,
      targetSystem         [2]     AccessPoint  OPTIONAL,
      COMPONENTS OF                CommonArguments}}

AddEntryResult      ::=    NULL
```

11.1.2 Add Entry arguments

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

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.

The **targetSystem** argument indicates the DSA to hold the new entry. If this argument is absent, it shall be taken to mean the same DSA as holds the superior of the new object. If the argument is present, it shall be the DSA with the specified **AccessPoint**. The parameter shall be absent when subentries are to be added.

If the argument is present, the **targetSystem** bit in the **criticalExtensions** parameter in **CommonArguments** shall be set, indicating that this extension is critical.

NOTE 1 – If the choice of indicated or implied DSA conflicts with local adminstrative policy, the operation is not performed and an error is returned.

The **CommonArguments** (see 7.3) include a specification of the service controls applying to the request. The **dontDereferenceAlias** option is ignored (and treated as set) unless the **useAliasOnUpdate** critical extension bit is set in **criticalExtensions**. Thus aliases are dereferenced by this operation only if **dontDereferenceAlias** is not set and **useAliasOnUpdate** is set. The **sizeLimit** component is ignored if provided.

NOTE 2 – Update operations that involve dereferencing of an alias name will always fail if they encounter 1988-edition DSAs.

11.1.3 Add Entry results

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

11.1.4 Add Entry errors

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

11.1.5 Add operation decision points for basic-access-control

If **basic-access-control** is in effect for the entry being added, the following sequence of access controls applies:

1) No specific permission is required to the immediate superior of the entry identified by the **object** argument.

   NOTE 1 – Security policy may prevent Directory users from adding entries across DSA boundaries (e.g. using the **targetSystem** argument). In this event, an appropriate **NameError**, **ServiceError**, **SecurityError** or **UpdateError** may be returned provided that it does not compromise the existence of the immediate superior entry. If it does (i.e. *DiscloseOnError* is not granted to the superior entry), the procedure defined in 7.11.3 shall be followed with respect to the superior entry.

2) If an entry already exists with a distinguished name equal to the **object** argument, the operation fails in accordance with 11.1.5.1, item a).

3) *Add* permission is required for the new entry being added. If this permission is not granted, the operation fails in accordance with 11.1.5.1, item b).

   NOTE 2 – The *Add* permission must be provided as prescriptive ACI.

4) For each attribute type and for each value that is to be added, *Add* permission is required. If any permission is absent, the operation fails in accordance with 11.1.5.1, item c).

11.1.5.1 Error returns

If the operation fails as defined in 11.1.5, the following procedure applies:

a) If the operation fails as defined in 11.1.5 item 2), the valid error returns are one of: if *DiscloseOnError* or *Add* permission is granted to the existing entry, an **UpdateError** with problem **entryAlreadyExists** shall be returned. Otherwise, the procedure described in 7.11.3 is followed with respect to the entry being added.

b) If the operation fails as defined in 11.1.5 item 3), the procedure described in 7.11.3 is followed with respect to the entry being added.

c) If the operation fails as defined in 11.1.5 item 4), the valid error return is **SecurityError** with problem **insufficientAccessRights** or **noInformation**.

11.2 Remove Entry

11.2.1 Remove Entry syntax

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          OPERATION    ::=     {
        ARGUMENT             RemoveEntryArgument
        RESULT               RemoveEntryResult
        ERRORS               { nameError | serviceError | referral | securityError | updateError }
        CODE                 id-opcode-removeEntry }

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

RemoveEntryResult    ::=          NULL
```

11.2.2 Remove Entry arguments

The **object** argument identifies the entry to be deleted.

The **CommonArguments** (see 7.3) include a specification of the service controls applying to the request. The **dontDereferenceAlias** option is ignored (and treated as set) unless the **useAliasOnUpdate** critical extension bit is set in **criticalExtensions**. Thus aliases are dereferenced by this operation only if **dontDereferenceAlias** is not set and **useAliasOnUpdate** is set. The **sizeLimit** component is ignored if provided.

NOTE – Update operations that involve dereferencing of an alias name will always fail if they encounter 1988-edition DSAs.

11.2.3 Remove Entry results

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

11.2.4 Remove Entry errors

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

11.2.5 Remove Entry operation decision points for basic-access-control

If **basic-access-control** is in effect for the entry being removed, the following access controls apply:

– *Remove* permission is required for the entry being removed. If this permission is not granted, the operation fails in accordance with 7.11.3.

NOTE – No specific permissions are required for any of the attributes and attribute values present within the entry being removed.

11.3 Modify Entry

11.3.1 Modify Entry syntax

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          OPERATION    ::=     {
        ARGUMENT             ModifyEntryArgument
        RESULT               ModifyEntryResult
        ERRORS               { attributeError | nameError | serviceError | referral | securityError |
                             updateError }
        CODE                 id-opcode-modifyEntry }

ModifyEntryArgument  ::=          OPTIONALLY-SIGNED { SET {
        object               [0]  Name,
        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 ModifyEntry arguments

The **object** argument identifies the entry to which the modifications should be applied.

The **changes** argument defines a sequence of modifications that 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 1 – 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**.

NOTE 2 – This operation is not 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.

The **CommonArguments** (see 7.3) include a specification of the service controls applying to the request. The **dontDereferenceAlias** option is ignored (and treated as set) unless the **useAliasOnUpdate** critical extension bit is set in **criticalExtensions**. Thus aliases are dereferenced by this operation only if **dontDereferenceAlias** is not set and **useAliasOnUpdate** is set. The **sizeLimit** component is ignored if provided.

NOTE 3 – Update operations that involve dereferencing of an alias name will always fail if they encounter 1988-edition DSAs.

The operation may be used to modify directory operational attributes. Only those directory operational attributes which are not classified **noUserModification** (and to which the user has effective modification access rights) may be modified.

NOTE 4 – Whether or not user modification is permitted, the Directory may change the values of directory operational attributes as a side effect of other Directory operations.

The operation may be used to modify collective attributes only if the service control **subentries** is TRUE and if the **object** is the subentry actually holding the collective attribute(s) to be modified.

NOTE 5 – Caution should therefore be exercised when modifying the information returned on reading an entry: some of the information may be from collective attributes, and cannot be modified in an operation directed at the entry itself. For example, it is not possible to delete a collective attribute from an (ordinary) entry via a **removeAttribute** entry modification to the entry (an **attributeError** with problem **noSuchAttributeOrValue** would be returned).

The operation may be used to modify an entry's Object Class attribute value if the values specify auxiliary object classes. However, an attempt to change an Object Class value which specifies an entry's structural object class shall result in an **updateError** with problem **objectClassModificationProhibited**. Any modification to auxiliary object classes shall leave the superclass chains consistent and correct with the resultant object class definition.

11.3.3 Modify Entry results

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

11.3.4 Modify Entry errors

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

11.3.5 Modify Entry operation decision points for basic-access-control

If **basic-access-control** is in effect for the entry being modified, the following sequence of access controls applies:

1) *Modify* permission is required for the entry being modified. If this permission is not granted, the operation fails in accordance with 7.11.3.

2) For each of the specified **EntryModification** arguments applied in sequence, the following permissions are required:

   i) *Add* permission for the attribute type and for each of the values specified in an **addAttribute** parameter. If these permissions are not granted or the attribute already exists, the operation fails in accordance with 11.3.5.1, item a).

   ii) *Remove* permission for the attribute type specified in a **removeAttribute** parameter. If this permission is not granted, the operation fails in accordance with 11.3.5.1, item b).

   NOTE 1 – No specific permissions are required for any of the attribute values present within the attribute being removed.

   iii) *Add* permission on each of the attribute values specified in an **addValues** parameter. If these permissions are not granted or any of the attribute values already exist, the operation fails in accordance with 11.3.5.1, item c).

   iv) *Remove* permission on each of the values specified in a **removeValues** parameter. If these permissions are not granted, the operation fails in accordance with 11.3.5.1, item d).

   NOTE 2 – If the end result of a **removeValues** modification is to remove the last value of an attribute (which causes the attribute itself to be removed), *Remove* permission is also required on the specified attribute type.

11.3.5.1 Error returns

If the operation fails as defined in 11.3.5, the following procedure applies:

a) If the operation fails as defined in 11.3.5 item 2), subitem i), the valid error returns are one of: if the attribute already exists and *discloseOnError* or *add* is granted to that attribute, an **AttributeError** with problem **attributeOrValueAlreadyExists** shall be returned; otherwise a **SecurityError** with problem **insufficientAccessRights** or **noInformation.** shall be returned.

b) If the operation fails as defined in 11.3.5 item 2), subitem ii), the valid error returns are one of: if *DiscloseOnError* permission is granted to the attribute being removed and the attribute exists, a **SecurityError** with problem **insufficientAccessRights** or **noInformation** shall be returned; otherwise, an **AttributeError** with problem **noSuchAttributeOrValue** shall be returned.

c) If the operation fails as defined in 11.3.5 item 2), subitem iii), the valid error returns are one of: if an attribute value already exists and *discloseOnError* or *add* is granted to that attribute value, an **AttributeError** with problem **attributeOrValueAlreadyExists** shall be returned; otherwise, *discloseOnError* permission at the attribute level must be verified. If *discloseOnError* is granted to the attribute, a **SecurityError** with problem **insufficientAccessRights** or **noInformation**. shall be returned; otherwise, an **AttributeError** with problem **noSuchAttributeOrValue** shall be returned.

d) If the operation fails as defined in 11.3.5 item 2), subitem iv), the valid error returns are one of: if *DiscloseOnError* permission is granted to any of the attribute values being removed, a **SecurityError** with problem **insufficientAccessRights** or **noInformation** shall be returned; otherwise, an **AttributeError** with problem **noSuchAttributeOrValue** shall be returned.

11.4 Modify DN

11.4.1 Modify DN syntax

The **modifyDN** operation is used to change the Relative Distinguished Name of an entry, move an entry to a new superior in the DIT, or do both. It may be used with object entries or alias entries. If the entry has subordinates, then all subordinates are renamed or moved accordingly (i.e. the subtree remains intact). The arguments of the operation may optionally be signed (see 7.10) by the requestor.

NOTES

1 1988-edition systems may use the operation only to change the Relative Distinguished Name of a leaf entry.

2 1993-edition systems may use the operation to move entries to a new superior only if the old superior, the new superior, the entry, and all its subordinates are in the one DSA.

3 The operation does not move entries to a new DSA; all entries remain in the original DSA.

4 The operation either succeeds or fails in its entirety; it shall not fail with some entries moved and some not moved. No intermediate states of the operation shall be externally visible to users of the Directory.

5 Some offline activity may be required following this operation to preserve consistency, for example to update attributes in any entries that hold Distinguished Name values that refer to the renamed or moved entry(ies).

6 The **modifyTimeStamp** attribute is not updated for entries subordinate to the renamed or moved entry.

```
modifyDN    OPERATION    ::=    {
        ARGUMENT             ModifyDNArgument
        RESULT               ModifyDNResult
        ERRORS               { nameError | serviceError | referral | securityError | updateError }
        CODE                 id-opcode-modifyDN }

ModifyDNArgument             ::=    OPTIONALLY-SIGNED { SET {
        object               [0]    DistinguishedName,
        newRDN               [1]    RelativeDistinguishedName,
        deleteOldRDN         [2]    BOOLEAN DEFAULT FALSE,
        newSuperior          [3]    DistinguishedName OPTIONAL,
        COMPONENTS OF               CommonArguments }}

ModifyDNResult       ::=    NULL
```

11.4.2 Modify DN arguments

The **object** argument identifies the entry whose Distinguished Name is to be modified. Aliases in the name shall not be dereferenced.

The **newRDN** argument specifies the new RDN of the entry. If the operation moves the entry to a new superior without changing its RDN, the old RDN is supplied for this parameter.

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.

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.

The **newSuperior** argument, if present, specifies that the entry is to be moved to a new superior in the DIT. The entry becomes an immediate subordinate of the entry with the indicated Distinguished Name, which must be an already existing object entry. The new superior shall not be the entry itself or any of its subordinates, or an alias, or such that the moved entry violates any DIT structure rules. It is possible that entries *subordinate* to the moved entry may violate the active subschema, in which case it is the responsibility of the Subschema Administrative Authority to make subsequent adjustments to these entries to make them consistent with the subschema, as described in ITU-T Rec. X.501 | ISO/IEC 9594-2, clause 13.

If the argument is present, the **newSuperior** bit in the **criticalExtensions** parameter in **CommonArguments** shall be set, indicating that this extension is critical.

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.

11.4.3 Modify DN results

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

11.4.4 Modify DN errors

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

11.4.5 ModifyDN decision points for basic-access-control

If **basic-access-control** is in effect for the entry being renamed, the following access controls apply.

– If the effect of the operation is to change the RDN of the entry, *Rename* permission is required for the entry being renamed (considered with its original name). If this permission is not granted, the operation fails in accordance with 11.4.5.1.

– If the effect of the operation is to move an entry to a new superior in the DIT, *Export* permission is required for the entry being considered with its original name, and *Import* permission is required for the entry being considered with its new name. If either of these permissions is not granted, the operation fails in accordance with 11.4.5.1.

NOTES

1 The *Import* permission must be provided as prescriptive ACI.

2 No additional permissions are required even if, as a result of modifying the last RDN of the name, a new distinguished value needs to be added or an old one removed.

11.4.5.1 Error returns

If the operation fails as defined in 11.4.5, the procedure described in 7.11.3 is followed with respect to the entry being renamed (considered with its original name).

12 Errors

12.1 Error precedence

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

NOTES

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.

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.

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**.

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

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       ERROR       ::=     { -- not literally an "error"
    CODE                id-errcode-abandoned }
```

There are no parameters associated with this error.

12.3 Abandon Failed

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

```
abandonFailed    ERROR       ::=         {
        PARAMETER       SET {
                problem         [0]     AbandonProblem,
                operation       [1]     InvokeId}
        CODE                    id-errcode-abandonFailed }

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

The various parameters have the following meanings.

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.

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

12.4 Attribute Error

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

```
attributeError      ERROR       ::=     {
        PARAMETER       SET {
                object                  [0]     Name,
                problems                [1]     SET OF SEQUENCE {
                        problem                         [0]     AttributeProblem,
                        type                            [1]     AttributeType,
                        value                           [2]     AttributeValue  OPTIONAL }}
        CODE                    id-errcode-attributeError }

AttributeProblem    ::=     INTEGER {
        noSuchAttributeOrValue          (1),
        invalidAttributeSyntax          (2),
        undefinedAttributeType          (3),
        inappropriateMatching           (4),
        constraintViolation             (5),
        attributeOrValueAlreadyExists   (6) }
```

The various parameters have the following meanings.

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

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 **AddEntry** or **ModifyEntry** 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 value supplied in the argument of an operation does not conform to the constraints imposed by ITU-T Rec. X.501 | ISO/IEC 9594-2 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

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

```
nameError          ERROR      ::=    {
     PARAMETER          SET {
          problem                [0]    NameProblem,
          matched                [1]    Name }
     CODE               id-errcode-nameError}

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

The various parameters have the following meanings.

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

a) **noSuchObject** – The name supplied does not match the name of any object.

b) **aliasProblem** – An alias has been dereferenced which names no object.

c) **invalidAttributeSyntax** – An attribute type and its accompanying attribute value in an AVA in the name are incompatible.

d) **aliasDereferencingProblem** – An alias was encountered in a situation where it was not allowed or where access was denied.

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

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

```
referral       ERROR      ::=    { -- not literally an "error"
     PARAMETER          SET {
          candidate              [0]    ContinuationReference }
     CODE               id-errcode-referral }
```

The error has a single parameter which contains a **ContinuationReference** which can be used to progress the operation (see ITU-T Rec. X.518 | ISO/IEC 9594-4).

12.7 Security Error

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

```
securityError      ERROR      ::=    {
     PARAMETER          SET {
          problem                [0]    SecurityProblem }
     CODE               id-errcode-securityError }

SecurityProblem    ::=    INTEGER {
     inappropriateAuthentication         (1),
     invalidCredentials                  (2),
     insufficientAccessRights            (3),
     invalidSignature                    (4),
     protectionRequired                  (5),
     noInformation                       (6) }
```

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

a) **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

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

```
serviceError        ERROR       ::=     {
    PARAMETER       SET {
        problem             [0]     ServiceProblem}
    CODE                id-errcode-serviceError }

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),
    invalidQueryReference               (13) }
```

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 violates 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, (via 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 this request due to an internal loop.

j) **unavailableCriticalExtension** – The Directory was unable to satisfy 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.

m) **invalidQueryReference** – The parameters of the requested operation are invalid. This problem is reported if the **queryReference** in paged results is invalid.

NOTE – This problem is not supported by 1988 edition systems.

12.9 Update Error

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

```
updateError     ERROR     ::=     {
        PARAMETER     SET {
                problem          [0]     UpdateProblem }
        CODE          id-errcode-updateError }

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

The error has a single 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 ITU-T Recommendation X.501 | ISO/IEC 9594-2. 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 rules for entry content; for example, its object class definition, the DIT content rules, or with the definitions of ITU-T Rec. X.501 | ISO/IEC 9594-2 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** or **ModifyDN** operation names an entry which already exists.

f) **affectsMultipleDSAs** – An attempted update would need to operate on multiple DSAs where this operation is not permitted.

g) **objectClassModificationProhibited** – An operation attempted to modify the structural object class of an entry.

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

Annex A

Abstract Service in ASN.1

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

This annex includes all of the ASN.1 type, value and information object definitions contained in this Directory Specification in the form of the ASN.1 module **DirectoryAbstractService**.

```
DirectoryAbstractService {joint-iso-ccitt ds(5) module(1) directoryAbstractService(2) 2}
DEFINITIONS ::=
BEGIN

-- EXPORTS All --

-- The types and values defined in this module are exported for use in the other ASN.1 modules contained
-- within the Directory Specifications, and for the use of other applications which will use them to access
-- Directory services. Other applications may use them for their own purposes, but this will not constrain
-- extensions and modifications needed to maintain or improve the Directory service.

IMPORTS
        informationFramework, distributedOperations, authenticationFramework, dap
                FROM UsefulDefinitions {joint-iso-ccitt ds(5) module(1) usefulDefinitions(0) 2}

        Attribute, AttributeType, AttributeValue, AttributeValueAssertion, DistinguishedName, Name,
        RelativeDistinguishedName, SupportedAttributes, ATTRIBUTE, MATCHING-RULE
                FROM  InformationFramework informationFramework

        OperationProgress, ReferenceType, Exclusions, AccessPoint, ContinuationReference
                FROM  DistributedOperations  distributedOperations

        CertificationPath, SIGNED {}, SIGNATURE {}, AlgorithmIdentifier
                FROM  AuthenticationFramework authenticationFramework

        id-opcode-read, id-opcode-compare, id-opcode-abandon, id-opcode-list, id-opcode-search,
        id-opcode-addEntry, id-opcode-removeEntry,  id-opcode-modifyEntry,  id-opcode-modifyDN,
        id-errcode-abandoned, id-errcode-abandonFailed, id-errcode-attributeError,
        id-errcode-nameError, id-errcode-referral, id-errcode-securityError, id-errcode-serviceError,
        id-errcode-updateError
                FROM DirectoryAccessProtocol dap

        OPERATION, ERROR
                FROM   Remote-Operations-Information-Objects {joint-iso-ccitt remote-operations(4)
                        informationObjects(5) version1(0) }

        emptyUnbind
                FROM Remote-Operations-Useful-Definitions {joint-iso-ccitt remote-operations(4)
                        useful-definitions(7) version1(0)}

        InvokeId
                FROM Remote-Operations-Generic-ROS-PDUs {joint-iso-ccitt remote-operations(4)
                        generic-ROS-PDUs(6) version1(0)}    ;

-- Parameterized type for representing optional signing --

OPTIONALLY-SIGNED {Type}        ::=     CHOICE {
        unsigned        Type,
        signed          SIGNED {Type}}

-- Common data types --

CommonArguments         ::=     SET {
        serviceControls         [30]    ServiceControls DEFAULT {},
        securityParameters      [29]    SecurityParameters OPTIONAL,
        requestor               [28]    DistinguishedName OPTIONAL,
        operationProgress       [27]    OperationProgress
                                                DEFAULT { nameResolutionPhase notStarted },
        aliasedRDNs             [26]    INTEGER OPTIONAL,
        criticalExtensions      [25]    BIT STRING  OPTIONAL,
```

```
        referenceType           [24]    ReferenceType  OPTIONAL,
        entryOnly               [23]    BOOLEAN DEFAULT TRUE,
        exclusions              [22]    Exclusions  OPTIONAL,
        nameResolveOnMaster
                                [21]    BOOLEAN DEFAULT FALSE }

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

ServiceControls     ::=     SET {
        options                 [0]     BIT STRING {
                preferChaining                  (0),
                chainingProhibited              (1),
                localScope                      (2),
                dontUseCopy             (3),
                dontDereferenceAliases          (4),
                subentries                      (5),
                copyShallDo                     (6) }  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,
        attributeSizeLimit      [5]     INTEGER  OPTIONAL }

EntryInformationSelection   ::=     SET {
        attributes      CHOICE {
                allUserAttributes               [0]     NULL,
                select                  [1]     SET OF AttributeType
                -- empty set implies no attributes are requested --  } DEFAULT allUserAttributes : NULL,
        infoTypes                               [2]     INTEGER {
                attributeTypesOnly                      (0),
                attributeTypesAndValues                 (1) } DEFAULT attributeTypesAndValues,
        extraAttributes         CHOICE {
                allOperationalAttributes        [3]     NULL,
                select                          [4]     SET OF AttributeType } OPTIONAL  }

EntryInformation    ::=     SEQUENCE {
        name                    Name,
        fromEntry                       BOOLEAN DEFAULT TRUE,
        information                     SET OF CHOICE {
                attributeType                   AttributeType,
                attribute                       Attribute } OPTIONAL,
        incompleteEntry         [3]     BOOLEAN DEFAULT FALSE -- not in 1988-edition systems -- }

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                            ATTRIBUTE.&id ({SupportedAttributes}),
                strings                         SEQUENCE OF CHOICE {
                        initial                         [0]  ATTRIBUTE.&Type
                                                                ({SupportedAttributes}{@substrings.type}),
                        any                             [1]  ATTRIBUTE.&Type
                                                                ({SupportedAttributes}{@substrings.type}),
                        final                           [2]  ATTRIBUTE.&Type
                                                                ({SupportedAttributes}{@substrings.type})}},
        greaterOrEqual          [2]     AttributeValueAssertion,
        lessOrEqual             [3]     AttributeValueAssertion,
        present                 [4]     AttributeType,
        approximateMatch        [5]     AttributeValueAssertion,
        extensibleMatch         [6]     MatchingRuleAssertion }
```

```
MatchingRuleAssertion ::= SEQUENCE {
        matchingRule        [1]     SET SIZE (1..MAX) OF MATCHING-RULE.&id,
        type                [2]     AttributeType  OPTIONAL,
        matchValue          [3]     MATCHING-RULE.&AssertionType ( CONSTRAINED BY {
                -- matchValue must be a value of  type specified by the &AssertionType field of
                -- one of the MATCHING-RULE information objects identified by matchingRule -- } ),
        dnAttributes        [4]     BOOLEAN DEFAULT FALSE }

PagedResultsRequest         ::=     CHOICE {
        newRequest                  SEQUENCE {
                pageSize                        INTEGER,
                sortKeys                        SEQUENCE OF SortKey OPTIONAL,
                reverse                 [1]     BOOLEAN DEFAULT FALSE,
                unmerged                [2]     BOOLEAN DEFAULT FALSE },
        queryReference              OCTET STRING }

SortKey     ::=     SEQUENCE {
        type                        AttributeType,
        orderingRule            MATCHING-RULE.&id OPTIONAL }

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) }
```

-- Bind and unbind operations --

```
directoryBind       OPERATION ::=   {
        ARGUMENT            DirectoryBindArgument
        RESULT              DirectoryBindResult
        ERRORS              { directoryBindError } }

DirectoryBindArgument       ::=     SET {
        credentials         [0]     Credentials OPTIONAL,
        versions            [1]     Versions DEFAULT {v1}}

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]     CHOICE {
                unprotected                 OCTET STRING,
                protected                   SIGNATURE {OCTET STRING} } OPTIONAL}

StrongCredentials   ::=     SET {
        certification-path  [0]     CertificationPath OPTIONAL,
        bind-token          [1]     Token,
        name                [2]     DistinguishedName OPTIONAL }

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

Versions    ::=     BIT STRING {v1(0)}

DirectoryBindResult         ::=     DirectoryBindArgument
```

```
directoryBindError   ERROR        ::=     {
        PARAMETER SET {
                versions          [0]     Versions DEFAULT {v1},
                error                     CHOICE {
                serviceError      1]      ServiceProblem,
                securityError     [2]     SecurityProblem }}}

directoryUnbind      OPERATION ::=     emptyUnbind
```

-- Operations, arguments, and results --

```
read    OPERATION  ::=     {
        ARGUMENT           ReadArgument
        RESULT             ReadResult
        ERRORS             { attributeError | nameError | serviceError | referral | abandoned |
                           securityError }
        CODE          id-opcode-read }

ReadArgument       ::=     OPTIONALLY-SIGNED { SET {
        object             [0]     Name,
        selection          [1]     EntryInformationSelection  DEFAULT { },
        modifyRightsRequest
                           [2]     BOOLEAN DEFAULT FALSE,
        COMPONENTS OF              CommonArguments }}

ReadResult  ::=    OPTIONALLY-SIGNED { SET {
        entry              [0]     EntryInformation,
        modifyRights       [1]     ModifyRights OPTIONAL,
        COMPONENTS OF              CommonResults }}

ModifyRights       ::=     SET OF SEQUENCE {
        item               CHOICE {
                entry              [0]     NULL,
                attribute          [1]     AttributeType,
                value              [2]     AttributeValueAssertion },
        permission         [3]     BIT STRING { add (0), remove (1), rename (2) , move(3) }}

compare     OPERATION  ::=     {
        ARGUMENT           CompareArgument
        RESULT             CompareResult
        ERRORS             { attributeError | nameError | serviceError | referral | abandoned |
                           securityError }
        CODE          id-opcode-compare }

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

CompareResult      ::=     OPTIONALLY-SIGNED { SET {
        name               Name  OPTIONAL,
        matched            [0]     BOOLEAN,
        fromEntry          [1]     BOOLEAN DEFAULT TRUE,
        matchedSubtype     [2]     AttributeType OPTIONAL,
        COMPONENTS OF              CommonResults }}

abandon     OPERATION  ::=     {
        ARGUMENT           AbandonArgument
        RESULT             AbandonResult
        ERRORS             { abandonFailed }
        CODE          id-opcode-abandon }

AbandonArgument    ::=     SEQUENCE {
        invokeID           [0]      InvokeId }

AbandonResult      ::=     NULL

list    OPERATION  ::=     {
        ARGUMENT           ListArgument
        RESULT             ListResult
        ERRORS             { nameError | serviceError | referral | abandoned | securityError }
        CODE          id-opcode-list }
```

```
ListArgument ::= OPTIONALLY-SIGNED { SET {
    object [0] Name,
    pagedResults [1] PagedResultsRequest OPTIONAL,
    COMPONENTS OF CommonArguments }}

ListResult ::= OPTIONALLY-SIGNED { CHOICE {
    listInfo SET {
        name Name OPTIONAL,
        subordinates [1] SET OF SEQUENCE {
            rdn 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,
    unknownErrors [3] SET OF ABSTRACT-SYNTAX.&Type OPTIONAL,
    queryReference [4] OCTET STRING OPTIONAL }

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

search OPERATION ::= {
    ARGUMENT SearchArgument
    RESULT SearchResult
    ERRORS { attributeError | nameError | serviceError | referral | abandoned |
        securityError }
    CODE id-opcode-search }

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 { },
    pagedResults [5] PagedResultsRequest OPTIONAL,
    matchedValuesOnly
        [6] BOOLEAN DEFAULT FALSE,
    extendedFilter [7] Filter OPTIONAL,
    COMPONENTS OF CommonArguments }}

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

addEntry OPERATION ::= {
    ARGUMENT AddEntryArgument
    RESULT AddEntryResult
    ERRORS { attributeError | nameError | serviceError | referral | securityError |
        updateError }
    CODE id-opcode-addEntry }

AddEntryArgument ::= OPTIONALLY-SIGNED { SET {
    object [0] Name,
    entry [1] SET OF Attribute,
    targetSystem [2] AccessPoint OPTIONAL,
    COMPONENTS OF CommonArguments}}
```

```
AddEntryResult      ::=     NULL

removeEntry         OPERATION ::=   {
        ARGUMENT            RemoveEntryArgument
        RESULT              RemoveEntryResult
        ERRORS              { nameError | serviceError | referral | securityError |  updateError }
        CODE        id-opcode-removeEntry }

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

RemoveEntryResult       ::=     NULL

modifyEntry OPERATION   ::=     {
        ARGUMENT            ModifyEntryArgument
        RESULT              ModifyEntryResult
        ERRORS              { attributeError | nameError | serviceError | referral | securityError |
                            updateError }
        CODE        id-opcode-modifyEntry }

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

ModifyEntryResult   ::=     NULL

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

modifyDN    OPERATION   ::=     {
        ARGUMENT            ModifyDNArgument
        RESULT              ModifyDNResult
        ERRORS              { nameError | serviceError | referral | securityError |  updateError }
        CODE        id-opcode-modifyDN }

ModifyDNArgument    ::=     OPTIONALLY-SIGNED { SET {
        object      [0]     DistinguishedName,
        newRDN              [1]     RelativeDistinguishedName,
        deleteOldRDN        [2]     BOOLEAN DEFAULT FALSE,
        newSuperior [3]     DistinguishedName OPTIONAL,
        COMPONENTS OF               CommonArguments }}

ModifyDNResult      ::=     NULL
```

-- Errors and parameters --

```
abandoned           ERROR       ::=     { -- not literally an "error"
        CODE        id-errcode-abandoned }

abandonFailed       ERROR       ::=     {
        PARAMETER SET {
                problem         [0]     AbandonProblem,
                operation       [1]     InvokeId }
        CODE        id-errcode-abandonFailed }

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

attributeError      ERROR       ::=     {
        PARAMETER SET {
                object                  [0]     Name,
                problems                [1]     SET OF SEQUENCE {
                        problem                         [0]     AttributeProblem,
                        type                            [1]     AttributeType,
                        value                           [2]     AttributeValue   OPTIONAL }}
        CODE        id-errcode-attributeError }

AttributeProblem    ::=     INTEGER {
        noSuchAttributeOrValue          (1),
        invalidAttributeSyntax          (2),
        undefinedAttributeType          (3),
```

```
        inappropriateMatching               (4),
        constraintViolation                 (5),
        attributeOrValueAlreadyExists       (6)}

nameError           ERROR       ::=     {
        PARAMETER SET {
                problem             [0]     NameProblem,
                matched             [1]     Name }
        CODE        id-errcode-nameError}

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

referral        ERROR       ::=     { -- not literally an "error"
        PARAMETER SET {
                candidate           [0]     ContinuationReference }
        CODE        id-errcode-referral }

securityError       ERROR       ::=     {
        PARAMETER SET {
                problem             [0]     SecurityProblem }
        CODE        id-errcode-securityError }

SecurityProblem     ::=     INTEGER {
        inappropriateAuthentication         (1),
        invalidCredentials                  (2),
        insufficientAccessRights            (3),
        invalidSignature                    (4),
        protectionRequired                  (5),
        noInformation                       (6)}

serviceError ERROR          ::=     {
        PARAMETER SET {
                problem             [0]     ServiceProblem}
        CODE        id-errcode-serviceError }

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),
        invalidQueryReference               (13) }

updateError ERROR           ::=     {
        PARAMETER SET {
                problem             [0]     UpdateProblem }
        CODE        id-errcode-updateError }

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

END
```

Annex B

Operational semantics for Basic Access Control

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

This annex contains a number of charts that describe the semantics associated with Basic Access Control as it applies to the processing of a Directory operation (see Figures B.1 to B.16).

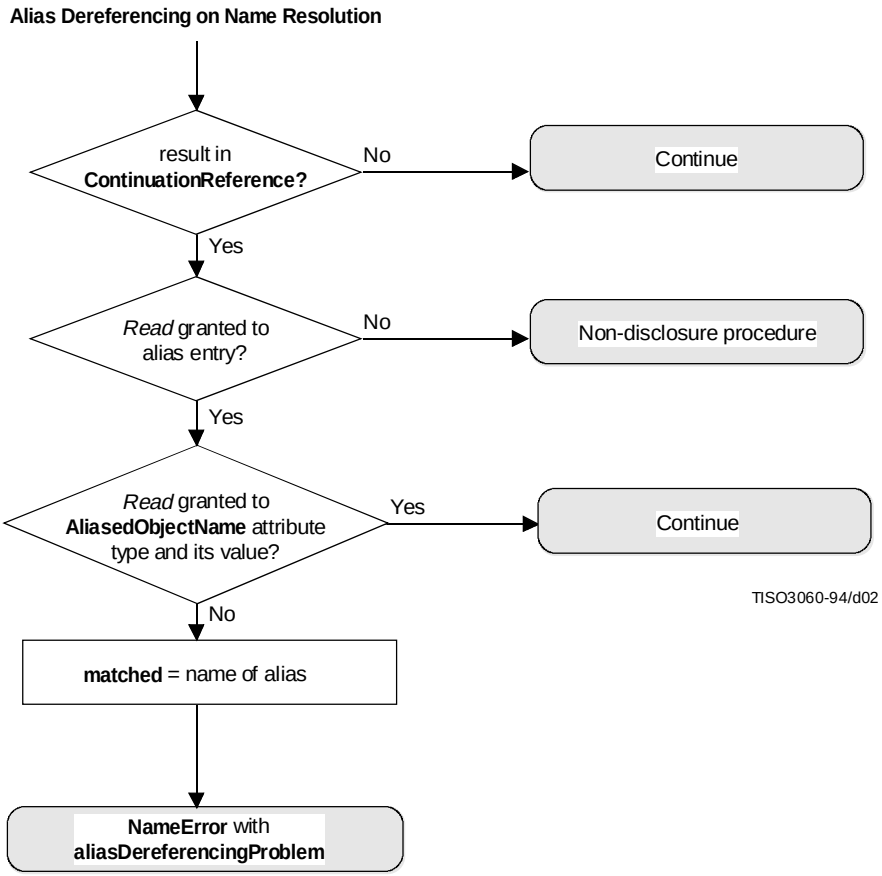

Figure B.1 – Alias Dereferencing in Name Resolution

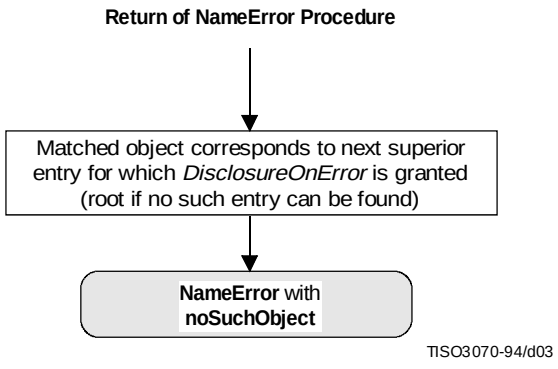

Figure B.2 – Return of Name Error

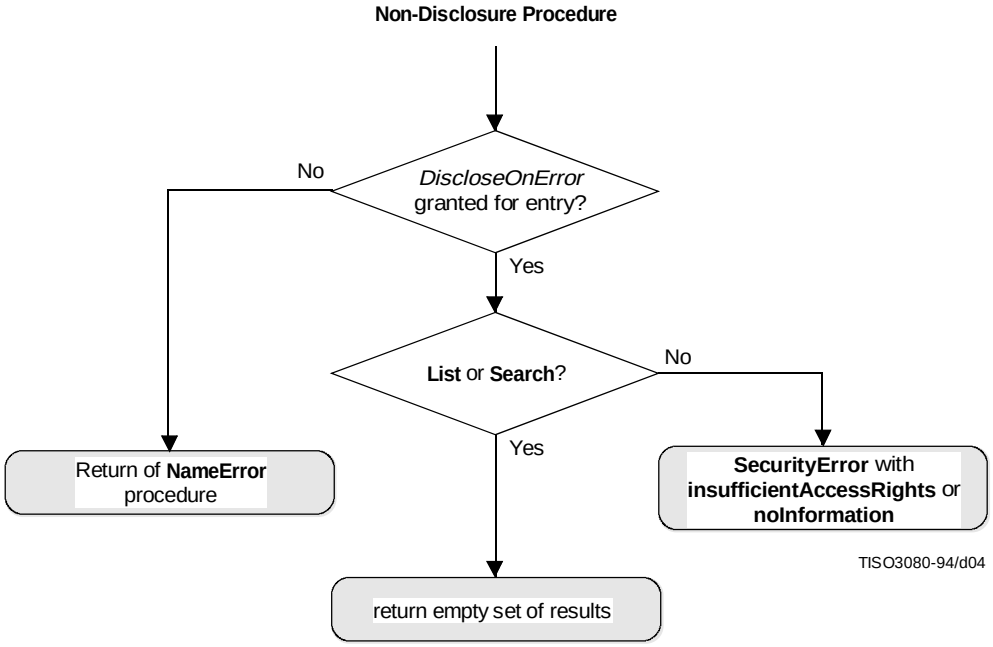

Figure B.3 – Non-Disclosure of the Existence of an Entry

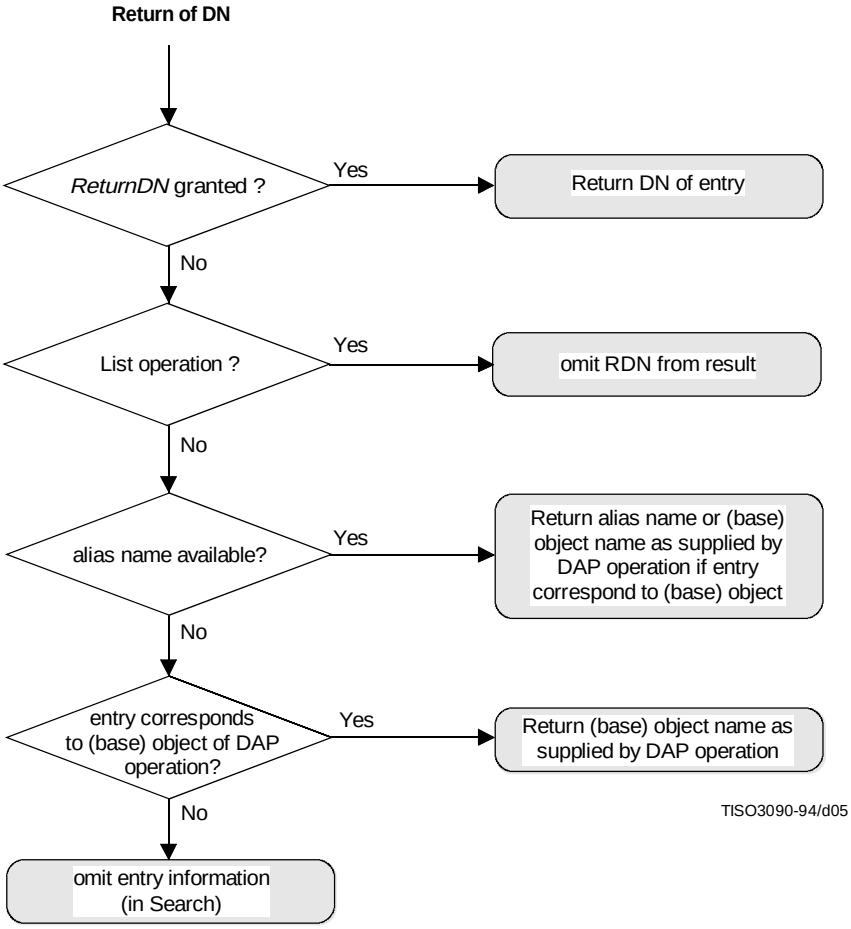

Figure B.4 – Return of Distinguished Name

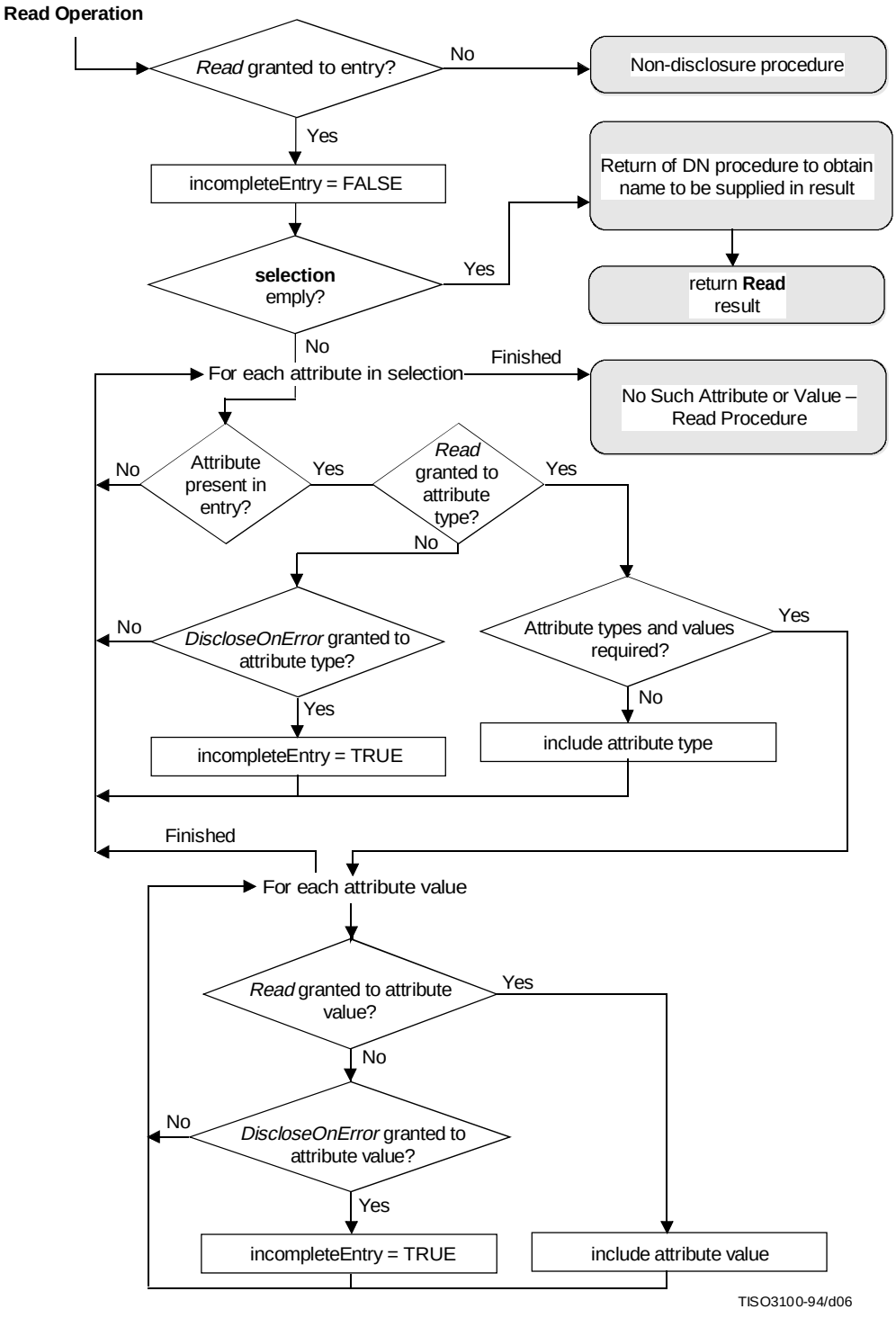

Figure B.5 – Read Operation

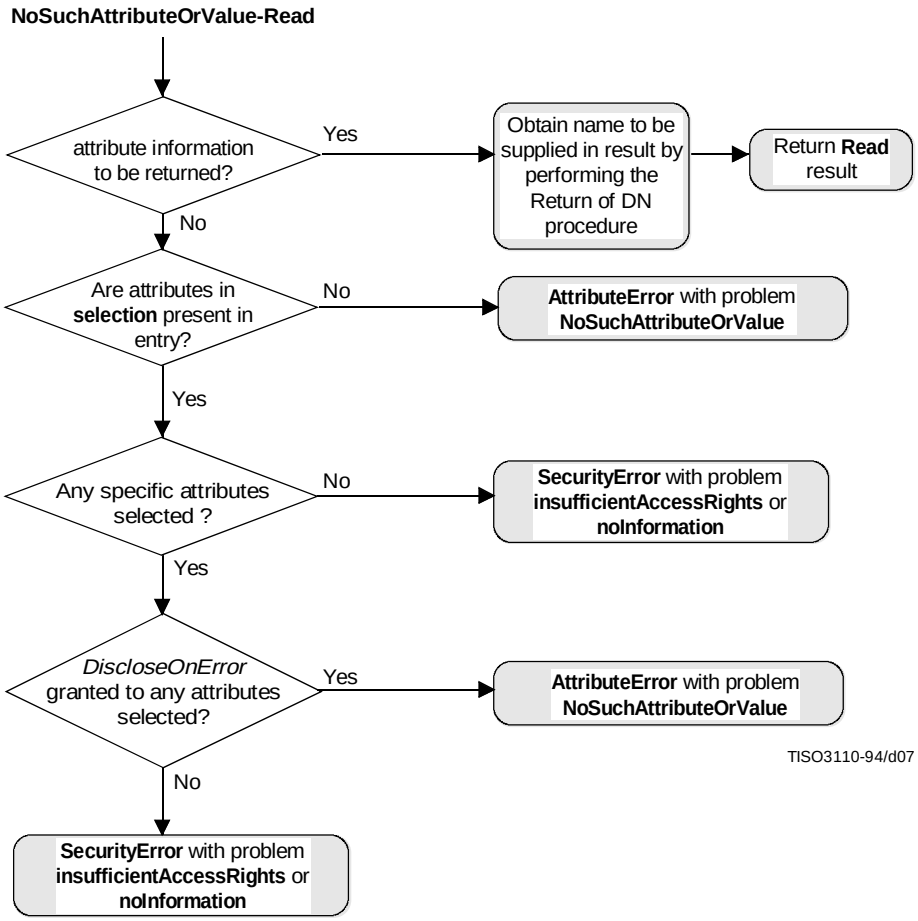

Figure B.6 – No Such Attribute Or Value for Read

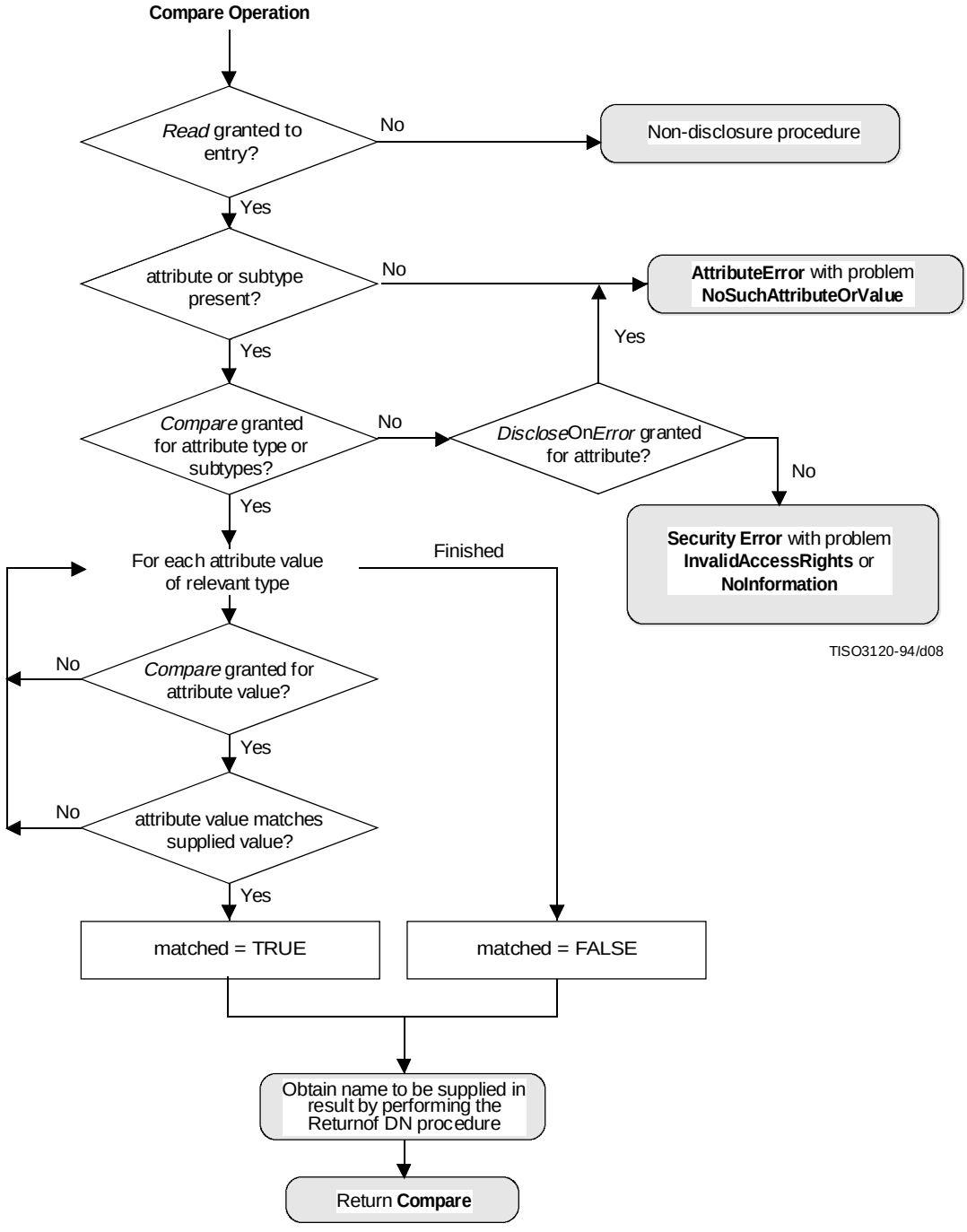

Figure B.7– Compare Operation

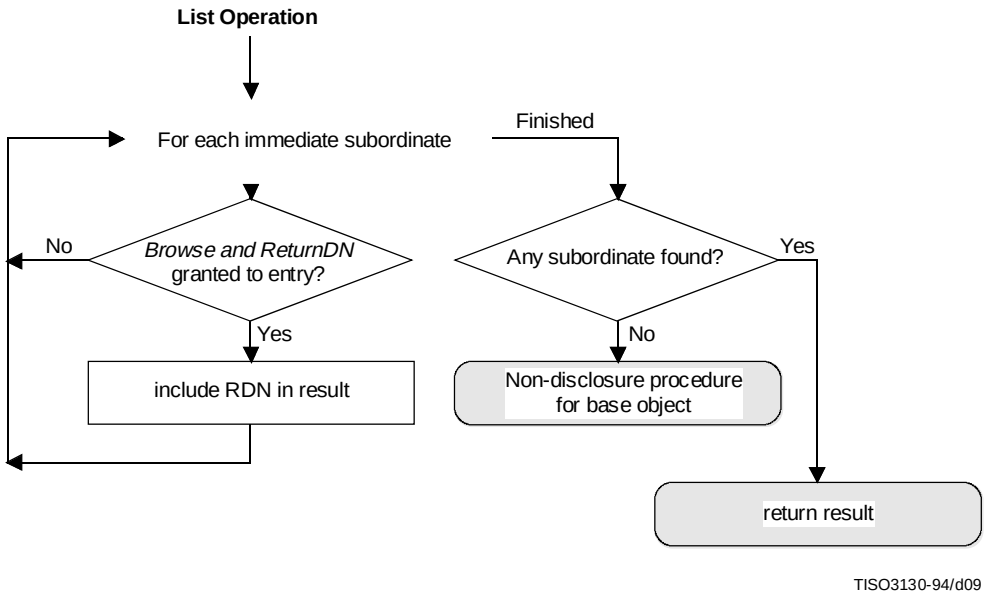

Figure B.8 – List Operation

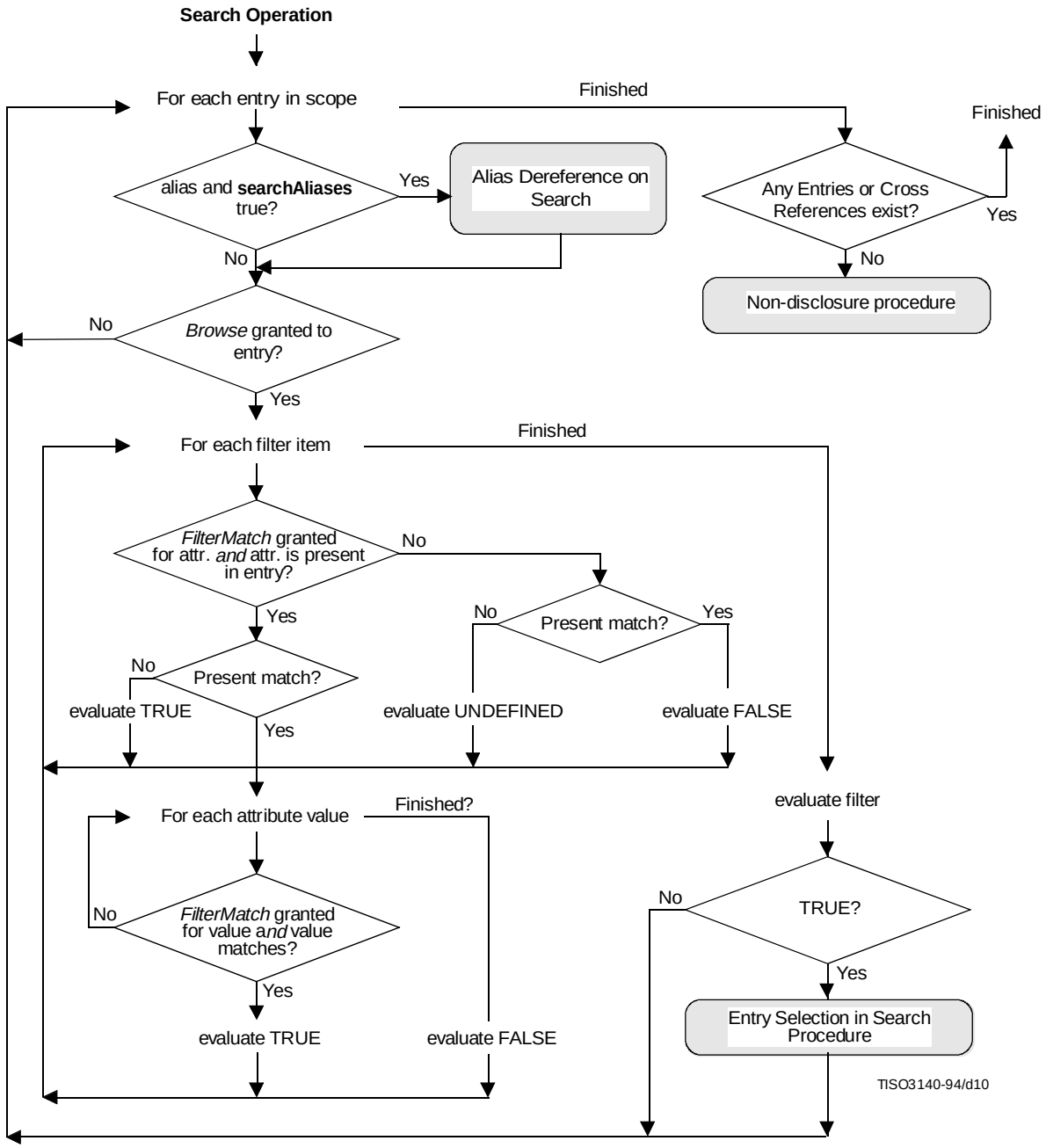

Figure B.9 – Search Operation

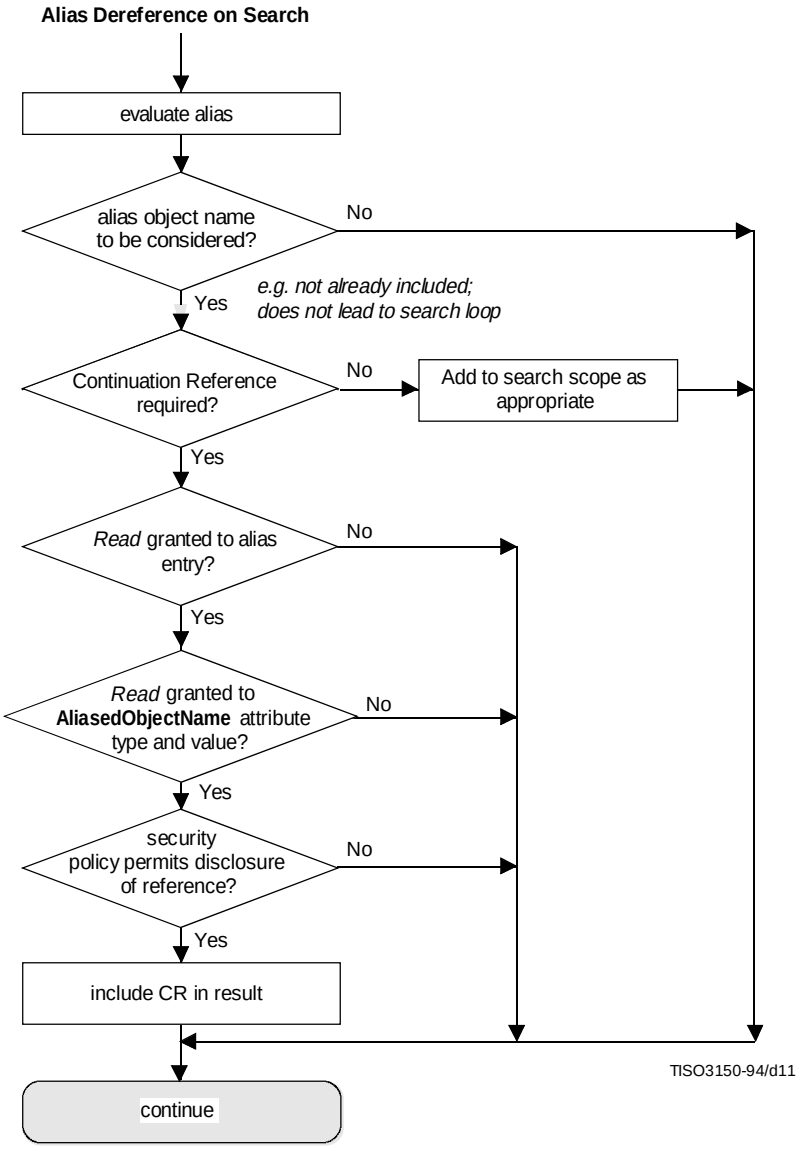

Figure B.10 – Alias Dereference in Search

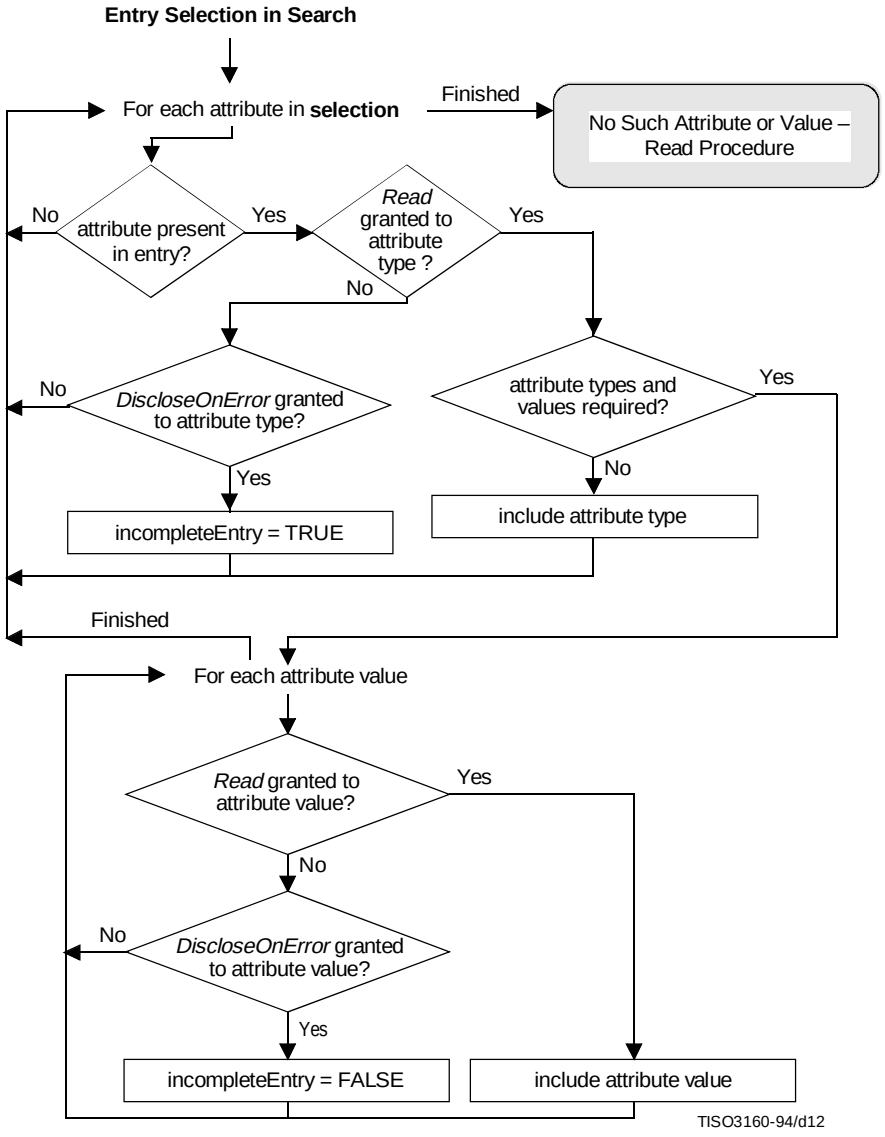

Figure B.11 – Entry Selection in Search

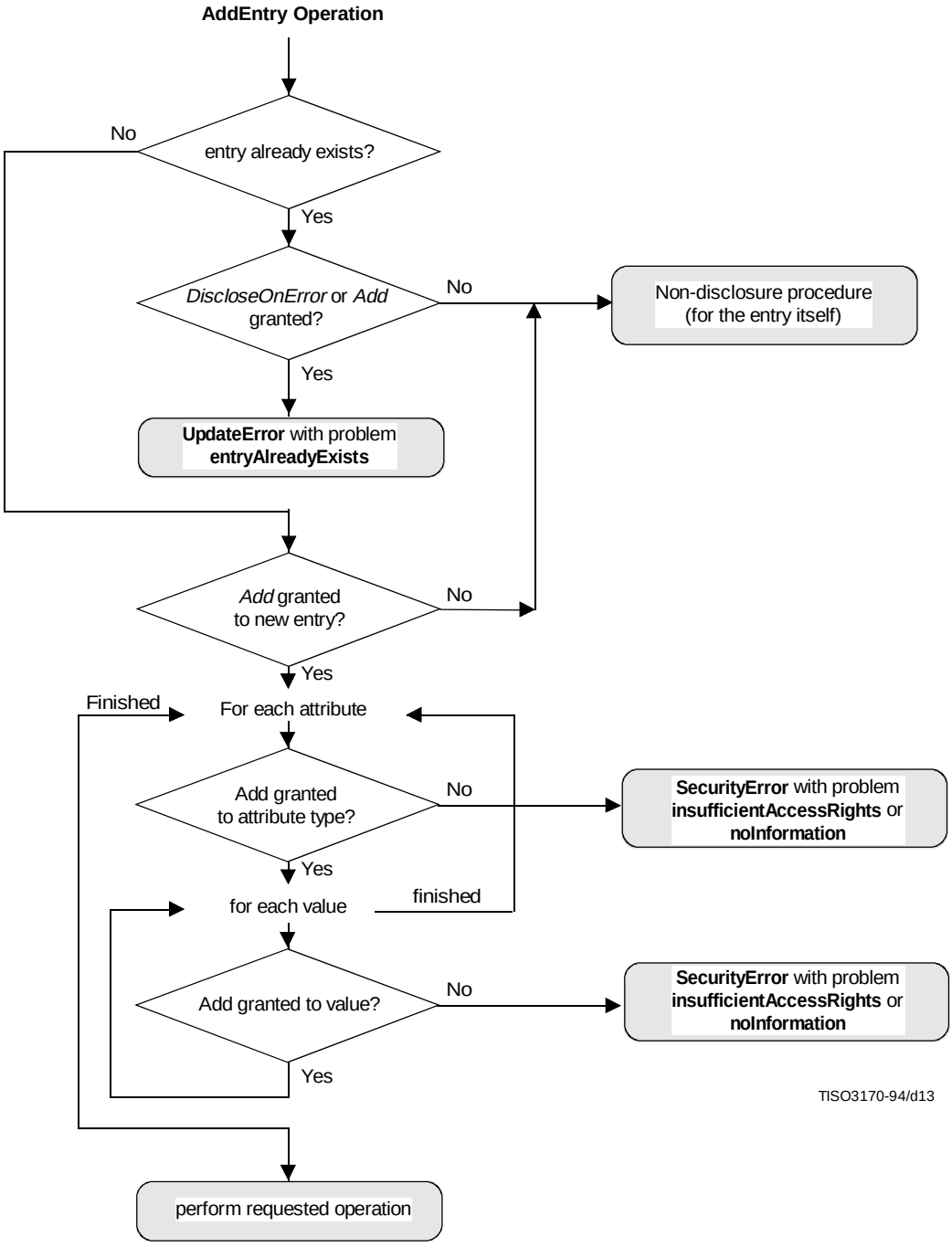

Figure B.12 – Add Entry Operation

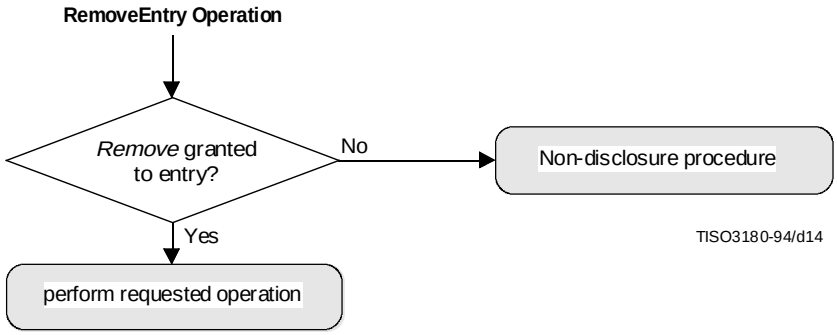

Figure B.13 – Remove Entry Operation

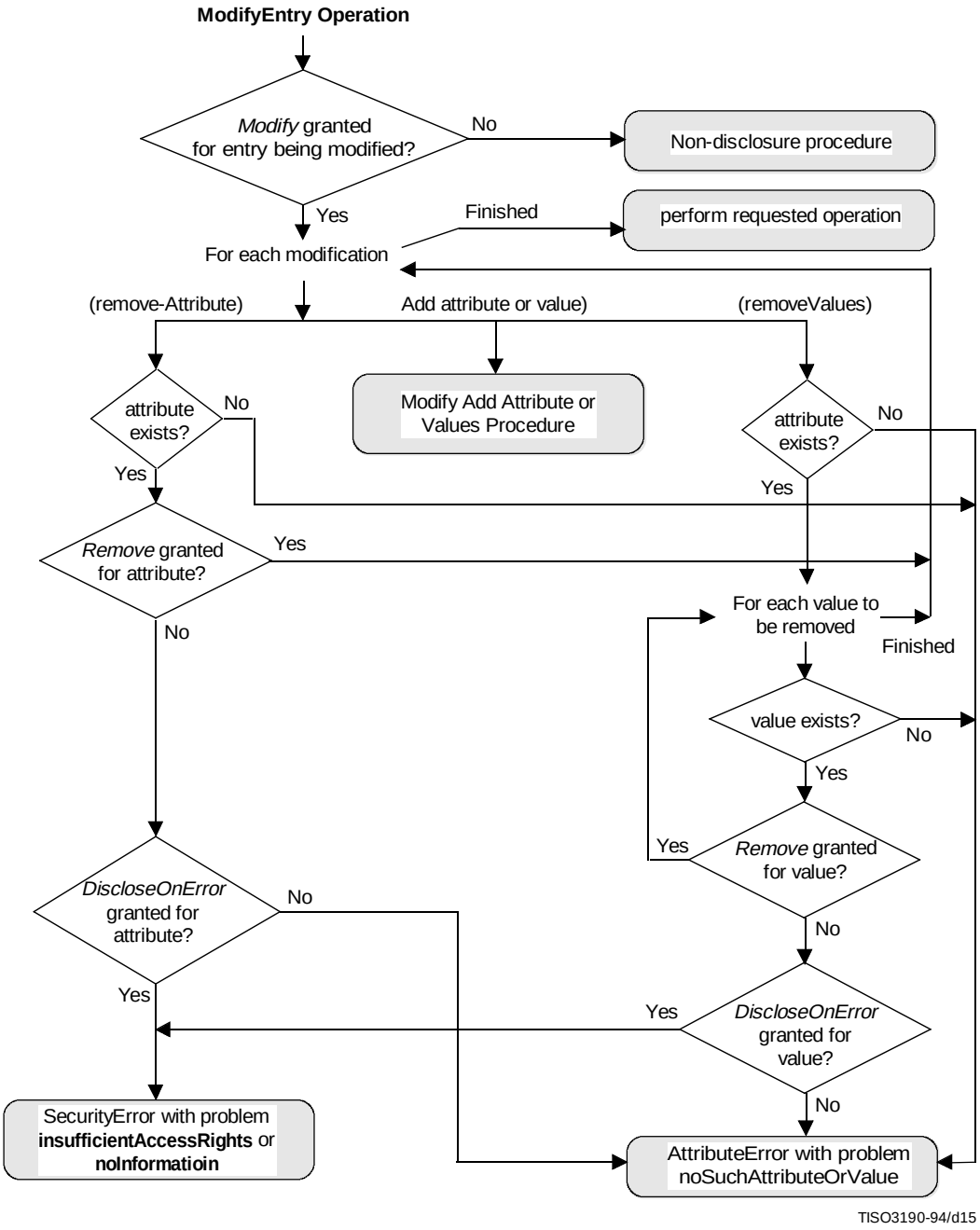

Figure B.14 – Modify Entry Operation

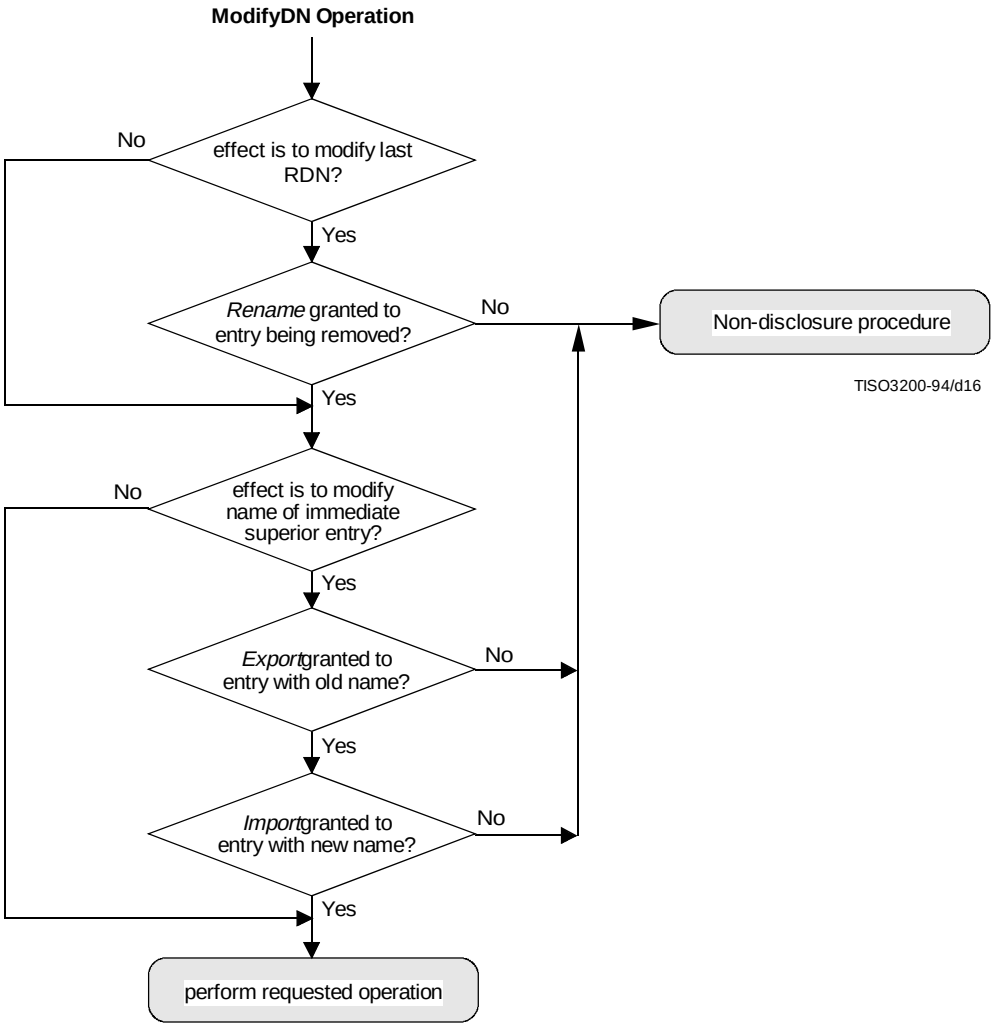

Figure B.15 – ModifyDN Operation

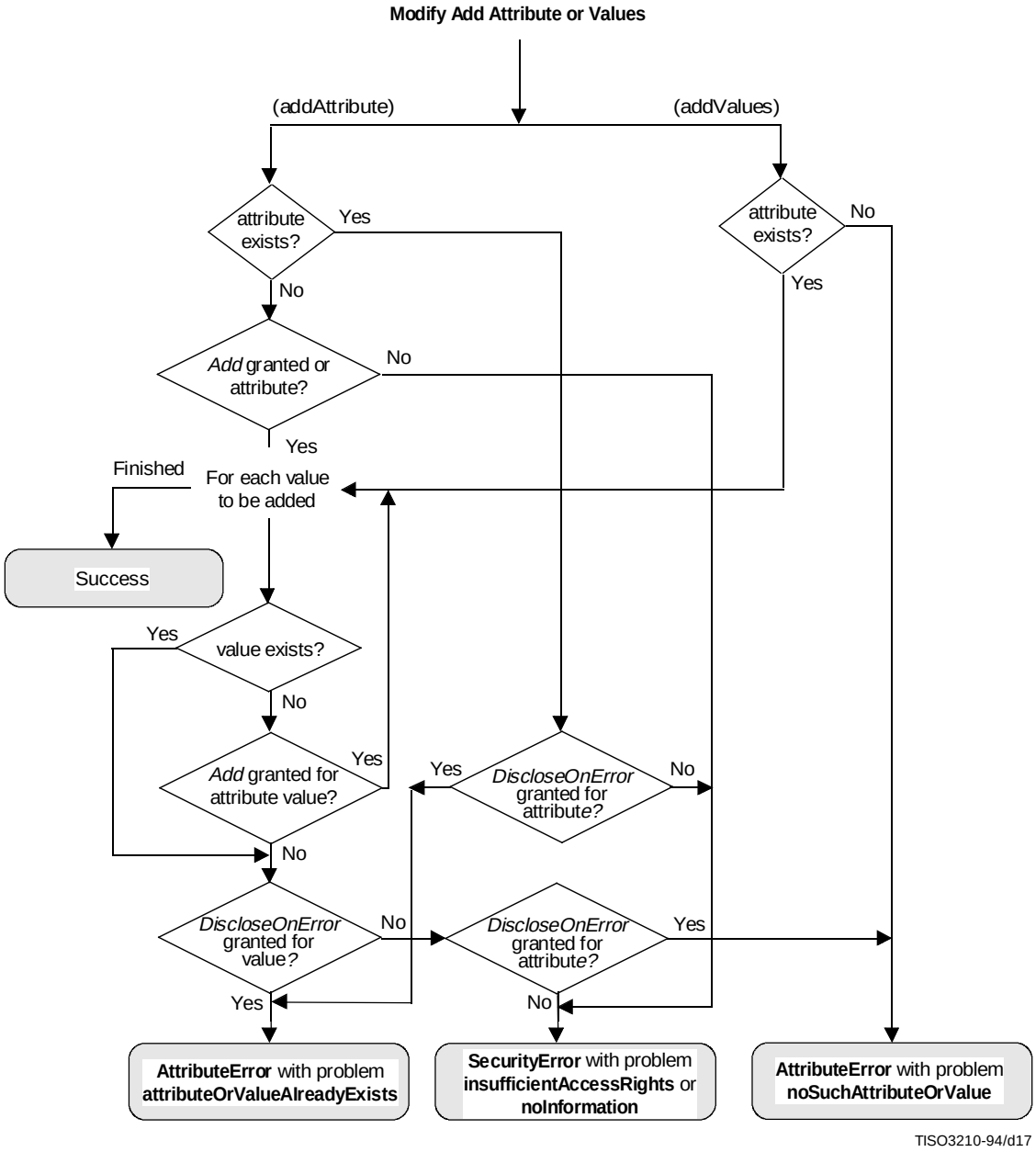

Figure B.16 – Modify Add Attribute or Values

Annex C

Amendments and corrigenda

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

This edition of this Directory Specification includes the following amendments:

- *Amendment 1* – Access Control.
- *Amendment 2* – Replication, Schema and Enhanced Search.

This edition of this Directory Specification includes the following technical corrigenda correcting the defects reported in the following defect reports (some parts of some of the following Technical Corrigenda may have been subsumed by the amendments that formed this edition of this Directory Specification):

- Technical Corrigendum 1 (covering Defect Reports 001, 007, 012, 014, 020, 032);
- Technical Corrigendum 2 (covering Defect Reports 038, 042);
- Technical Corrigendum 3 (covering Defect Report 052);
- Technical Corrigendum 4 (covering Defect Reports 041, 054, 060, 063, 068, 069);
- Technical Corrigendum 5 (covering Defect Report 067).