INTERNATIONAL STANDARD

ISO/IEC 9594-3

First edition
1990-12-15

Information technology — Open Systems Interconnection — The Directory —

Part 3: Abstract service definition

Technologies de l'information — Interconnexion de systèmes ouverts — L'annuaire —

Partie 3: Définition du service abstrait

Reference number
ISO/IEC 9594-3 : 1990 (E)

Contents

ISO/IEC Copyright Office ● Case postale 56 ● CH-1211 Genève 20 ● Switzerland
Printed in Switzerland

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

International Standard ISO/IEC 9594-3 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*.

ISO/IEC 9594 consists of the following parts, under the general title *Information technology — Open Systems Interconnection — The Directory*:

- *Part 1: Overview of concepts, models and services*
- *Part 2: Models*
- *Part 3: Abstract service definition*
- *Part 4: Procedures for distributed operation*
- *Part 5: Protocol specifications*
- *Part 6: Selected attribute types*
- *Part 7: Selected object classes*
- *Part 8: Authentication framework*

Annexes A and B form an integral part of this part of ISO/IEC 9594.

Introduction

0.1 This part of ISO/IEC 9594, together with the other parts, has been produced to facilitate the interconnection of information processing systems to provide directory services. The set of all such systems, together with the directory information which they hold, can be viewed as an integrated whole, called the *Directory*. The information held by the Directory, collectively known as the Directory Information Base (DIB), is typically used to facilitate communication between, with or about objects such as application entities, people, terminals, and distribution lists.

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

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

0.3 This part of ISO/IEC 9594 defines the capabilities provided by the Directory to its users.

0.4 Annex A provides the ASN.1 module which contains all the definitions associated with the Abstract Service

Information technology — Open Systems Interconnection — The Directory —

Part 3:
Abstract service definition

SECTION 1: GENERAL

1 Scope

1.1 This part of ISO/IEC 9594 defines in an abstract way the externally visible service provided by the Directory.

1.2 This part of ISO/IEC 9594 does not specify individual implementations or products.

2 Normative references

The following standards contain provisions which, through reference in this text, constitute provisions of this part of ISO/IEC 9594. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this part of ISO/IEC 9594 are encouraged to investigate the possibility of applying the most recent editions of the standards listed below. Members of IEC and ISO maintain registers of currently valid International Standards.

ISO 7498:1984, *Information Processing Systems — Open Systems Interconnection — Basic Reference Model.*

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

ISO/IEC 9072-1:1989, *Information Processing Systems — Text Communication — Remote Operations — Part 1: Model, Notation and Service Definition.*

ISO/IEC 9072-2:1989, *Information Processing Systems — Text Communication — Remote Operations — Part 2: Protocol Specification.*

ISO/IEC 9594-1:1990, *Information Technology — Open Systems Interconnection — The Directory — Part 1: Overview of Concepts, Models and Services.*

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

ISO/IEC 9594-4:1990, *Information Technology — Open Systems Interconnection — The Directory — Part 4: Procedures for Distributed Operation.*

ISO/IEC 9594-5:1990, *Information Technology — Open Systems Interconnection — The Directory — Part 5: Protocol Specifications.*

ISO/IEC 9594-6:1990, *Information Technology — Open Systems Interconnection — The Directory — Part 6: Selected Attribute Types.*

ISO/IEC 9594-7:1990, *Information Technology — Open Systems Interconnection — The Directory — Part 7: Selected Object Classes.*

ISO/IEC 9594-8:1990, *Information Technology — Open Systems Interconnection — The Directory — Part 8: Authentication Framework.*

ISO/IEC 10021-3:1990, *Information Technology — Text Communication — Message Oriented Text Interchange System (MOTIS) — Part 3: Abstract Service Definition.*

3 Definitions

3.1 Basic Directory Definitions

This part of ISO/IEC 9594 makes use of the following terms defined in ISO/IEC 9594-1:

a) Directory;

b) Directory Information Base;

c) (Directory) User.

3.2 Directory Model Definitions

This part of ISO/IEC 9594 makes use of the following terms defined in ISO/IEC 9594-2:

a) Directory System Agent;

b) Directory User Agent.

3.3 Directory Information Base Definitions

This part of ISO/IEC 9594 makes use of the following terms defined in 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

This part of ISO/IEC 9594 makes use of the following terms defined in ISO/IEC 9594-2:

a) attribute;

b) attribute type;

c) attribute value;

d) attribute value assertion.

3.5 Name Definitions

This part of ISO/IEC 9594 makes use of the following terms defined in 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

This part of ISO/IEC 9594 makes use of the following terms defined in ISO/IEC 9594-4:

a) chaining;

b) referral.

3.7 Abstract Service Definitions

For the purpose of this part of ISO/IEC 9594 the following definitions apply:

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

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

4 Abbreviations

This part of ISO/IEC 9594 makes use of the following abbreviations:

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

This part of ISO/IEC 9594 makes use of the abstract service definition conventions defined in ISO/IEC 10021-3.

SECTION 2: ABSTRACT SERVICE

6 Overview of the Directory Service

6.1 As described in 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.

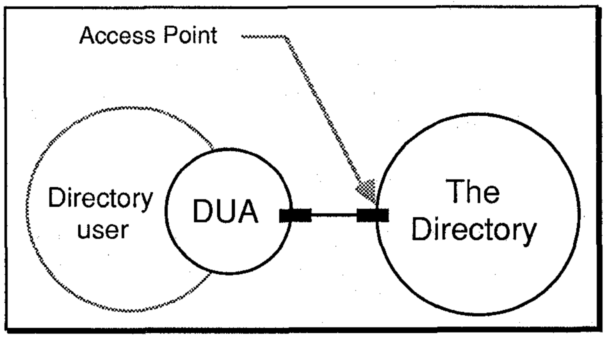

Figure 1 —Access to the Directory

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

6.3 Using the notation defined in ISO/IEC 8505-3, the Directory can be defined as follows:

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

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

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

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

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

The DUA consumes the services provided by the Directory.

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

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

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

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

6.7 These ports are used only as a method of structuring the description of the Directory service. Conformance to Directory operations is specified in ISO/IEC 9594-5.

7 Information Types

7.1 Introduction

7.1.1 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.

7.1.2 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.

7.2 Information Types Defined Elsewhere

7.2.1 The following information types are defined in ISO/IEC 9594-2:

a) **Attribute;**

b) **AttributeType;**

c) **AttributeValue;**

d) **AttributeValueAssertion;**

e) **DistinguishedName;**

f) **Name;**

g) **RelativeDistinguishedName.**

7.2.2 The following information type is defined in ISO/IEC 9594-6:

a) **PresentationAddress.**

7.2.3 The following information types are defined in ISO/IEC 9594-8:

a) ***Certificate;***

b) **SIGNED;**

c) **CertificationPath.**

7.2.4 The following information types are defined in ISO 9072-1:

a) **InvokeID.**

7.2.5 The following information types are defined in ISO 9594-4:

a) **OperationProgress;**

b) **ContinuationReference.**

7.3 Common Arguments

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

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

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

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

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

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

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

7.3.2.4 The **OperationProgress** defines the role that the DSA is to play in the distributed evaluation of the request. It is more fully defined in ISO/IEC 9594-4.

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

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

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

The subcomponents are as defined in 7.3.2.6.1 through 7.3.2.6.3.

7.3.2.6.1 The **identifier** serves to identify a particular extension. Values of this component shall be assigned only by future versions of ISO/IEC 9594.

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

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

7.4 Common Results

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

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

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

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

7.4.2.2 The **performer** 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.

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

7.5 Service Controls

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

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

          priority [1] INTEGER {
                    low (0),
                    medium (1),
                    high (2) } DEFAULT medium,

          timeLimit      [2]      INTEGER OPTIONAL,

          sizeLimit      [3]      INTEGER OPTIONAL,

          scopeOfReferral        [4]      INTEGER{
                                            dmd(0),
                                            country(1)}
                                            OPTIONAL }
```

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

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

a) **preferChaining** indicates that the preference is that chaining, rather than referrals, be used to provide the service. The Directory is not obliged to follow this preference;

b) **chainingProhibited** indicates that chaining, and other methods of distributing the request around the Directory, are prohibited;

c) **localScope** indicates that the operation is to be limited to a local scope . The definition of this option is itself a local matter, for example, within a single DSA or a single DMD;

d) **dontUseCopy** indicates that copied information (as defined in 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 - This is necessary to allow reference to an alias entry itself rather than the aliased entry, e.g., in order to read the alias entry.

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

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

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

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

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

7.5.2.5 The **scopeOfReferral** indicates the scope to which a referral returned by a DSA should be relevant. Depending on whether the 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.

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

7.6 Entry Information Selection

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

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

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

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

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

a) If the **select** option is chosen, then the attributes involved are listed. If the list is empty, then no attributes shall be returned. Information about a selected attribute shall be returned if the attribute is present. An **attributeError** with the **noSuchAttribute** problem shall only be returned if none of the attributes selected is present.

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

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

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

7.7 Entry Information

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

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

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

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

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

7.8 Filter

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

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

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

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

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

a) an **item**, it is **TRUE** if and only if the corresponding **FilterItem** is **TRUE**;

b) an **and**, it is **TRUE** unless any of the nested **Filters** is **FALSE**.

Note - thus, if there are no nested **Filter**s the **and** evaluates to **TRUE**;

c) an **or**, it is **FALSE** unless any of the nested **Filter**s is **TRUE**.

Note - thus, if there are no nested **Filter**s the **or** evaluates to **FALSE**;

d) a **not**, it is **TRUE** if and only if the nested **Filter** is **FALSE**.

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

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

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

Notes

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

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

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

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

7.8.3.4 A **FilterItem** may be undefined (as described in 7.8.2–3 above). Otherwise, where the **FilterItem** asserts:

a) **equality**, it is TRUE if and only if there is a value of the attribute which is equal to that asserted;

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

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

c) **greaterOrEqual**, it is TRUE if and only if the relative ordering (as defined by the appropriate ordering algorithm) places the supplied value before or equal to any value of the attribute;

d) **lessOrEqual**, it is TRUE if and only if the relative ordering (as defined by the appropriate ordering algorithm) places the supplied value after or equal to any value of the attribute;

e) **present**, it is TRUE if and only if such an attribute is present in the entry;

f) **approximateMatch**, it is TRUE if and only if there is a value of the attribute which matches that which is asserted by some locally-defined approximate matching algorithm (e.g., spelling variations, phonetic match, etc). There are no specific guidelines for approximate matching in this version of this part of ISO/IEC 9594. If approximate matching is not supported, this FilterItem should be treated as a match for equality.

7.9 Security Parameters

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

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

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

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

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

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

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

7.9.2.3 The **time** is the intended expiry time for the validity of the signature, when signed arguments are used.

It is used in conjunction with the random number to enable the detection of replay attacks.

7.9.2.4 The **random 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.

7.9.2.5 The **target ProtectionRequest** may appear only in the request for an operation to be carried out, and indicates the requestor's preference regarding the degree of protection to be provided to the result. Two levels are provided: **none** (no protection requested, 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.

7.10 OPTIONALLY-SIGNED

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

```
OPTIONALLY-SIGNED MACRO ::=
BEGIN

TYPE NOTATION       ::= type (Type)
VALUE NOTATION      ::= value (VALUE
                CHOICE { Type, SIGNED Type})

END
```

7.10.2 The SIGNED macro, which describes the form of the signed form of the information, is specified in 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 A DirectoryBind operation is used at the beginning of a period of accessing the Directory.

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

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

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

SimpleCredentials ::=  SEQUENCE{
      name              [0]     DistinguishedName,
      validity          [1]     SET {
      time1             [0]     UTCTime OPTIONAL,
      time2             [1]     UTCTime OPTIONAL,
      random1           [2]     BIT STRING  OPTIONAL,
      random2           [3]     BIT STRING  OPTIONAL}
                                      OPTIONAL,
              - - in most instances the arguments for
              - - time and random are relevant in
              - - dialogues employing protected
              - - password mechanisms and derive
              - - their meaning as per bilateral
              - - agreement
      password          [2]     OCTET STRING OPTIONAL}
              - - the value would be an unprotected
              - - password, or Protected1 or Protected2
              - - as specified in ISO/IEC 9594-8

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

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

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

DirectoryBindResult     ::= DirectoryBindArgument

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

8.1.2 The various arguments are defined in 8.1.2.1 and 8.1.2.2.

8.1.2.1 The **Credentials** of the DirectoryBindArgument allow the Directory to establish the identity of the user. They may be either simple, or strong or externally defined (external Procedure)(as described in ISO/IEC 9594-8].

8.1.2.1.1 SimpleCredentials consist of a name (always the distinguished name of an object) and (optionally) a password. This provides a limited degree of security. If the password is protected as described in clause 5 of ISO/IEC 9594-8, then **SimpleCredentials** includes **name**, **password**, and (optionally) time and/or random numbers which are used to detect replay. In some instances a protected password may be checked by an object which knows the password only after locally regenerating the protection to its own copy of the password and comparing the result with the value in the bind argument (**password**). In other instances a direct comparison may be possible.

8.1.2.1.2 **StrongCredentials** consist of a bind token, and, optionally, a certificate and sequence of certification-authority cross-certificate (as defined in ISO/IEC 9594-8). 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.

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

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

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

a) Any elements of **DirectoryBindArgument** other than those defined in this part of ISO/IEC 9594 shall be accepted and ignored.

b) Additional options for named bits of **DirectoryBindArgument** (e.g., Versions) not defined shall be accepted and ignored.

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

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

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

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

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

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

• **securityError**	**inappropriateAuthentication** **invalidCredentials**
• **serviceError**	**unavailable**

8.2 Directory Unbind

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

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

8.2.2 The DirectoryUnbind has no arguments.

9 Directory Read Operations

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

9.1 Read

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

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

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

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

9.1.2 The various arguments have the meanings as defined in 9.1.2.1-3.

9.1.2.1 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).

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

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

9.1.3 Should the request succeed, the result shall be returned. The result parameters have the meanings as defined in 9.1.3.1–2 and 7.4.

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

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

9.2 Compare

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

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

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

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

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

9.2.2.1 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).

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

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

9.2.3 Should the request succeed (i.e., the comparison is actually carried out), the result shall be returned. The result parameters have the meanings as described in 9.2.3.1 through 9.2.3.4 and 7.4.

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

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

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

9.2.4 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.

9.3 Abandon

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

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

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

AbandonResult ::= NULL
```

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

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

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

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

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

10 Directory Search Operations

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

10.1 List

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

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

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

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

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

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

10.1.2 The various arguments have the meanings as defined in 10.1.2.1, 10.1.2.2, and 7.3.

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

10.1.3 The request succeeds if the **object** is located, regardless of whether there is any subordinate information to return. The result parameters have the meanings as defined in 10.1.3.1 through 10.1.3.5 and 7.4.

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

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

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

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

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

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

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

10.1.3.3.2 The **unexplored** parameter shall be present if regions of the DIT were not explored. Its information allows the DUA to continue the processing of the List operation by contacting other access points if it so chooses. The parameter consists of a set (possibly empty) of **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.

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

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

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

10.2 Search

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

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

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

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

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

10.2.2 The various arguments have the meanings as defined in 10.2.2.1 through 10.2.2.5 and 7.3.

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

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

a) the **baseobject** only;

b) the immediate subordinates of the base object only (**oneLevel**);

c) the base object and all its subordinates (**wholeSubtree**).

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

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

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

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

Note — As a corollary to this, the outcome of an unfiltered search applied to a single entry may not be identical to a read which seeks to interrogate the same set of attributes of the entry. This is because the latter shall return an **AttributeError** if none of the selected attributes exist in the entry.

The result parameters have the meanings as defined in 10.2.3.1 through 10.2.3.5.

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

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

10.2.3.3 The **PartialOutcomeQualifier** consists of three subcomponents as described for the List operation in clause 10.1.3.3.

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

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

11 Directory Modify Operations

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

Notes

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

2 The success of **AddEntry**, **RemoveEntry**, and **ModifyRDN** operations shall be dependant on the physical distribution of the DIB across the Directory. Failure shall be reported with an **UpdateError** and problem **affectsMultipleDSAs**. See ISO/IEC 9594-4 .

11.1 Add Entry

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

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

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

AddEntryResult       ::=      NULL
```

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

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

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

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

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

11.1.4 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 RemoveEntry

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

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

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

RemoveEntryResult  ::= NULL
```

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

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

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

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

11.2.4 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 Modify Entry

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

a) add a new attribute

b) remove an attribute

c) add attribute values

d) remove attribute values

e) replace attribute values

f) modify an alias.

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

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

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

ModifyEntryResult ::= NULL

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

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

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

11.3.2.2 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 — This operation is not allowed if the attribute type is present in the RDN.

c) **addValues**. This identifies an attribute by the attribute type in the argument, and specifies one or more attribute values to be added to the attribute. An attempt to add an already existing value results in an error. An attempt to add a value to a non-existent type results in an error.

d) **removeValues**. This identifies an attribute by the attribute type in the argument, and specifies one or more attribute values to be removed from the attribute. If the values are not present in the attribute, this results in an **AttributeError**. If an attempt is made to modify the object class attribute, an **UpdateError** is returned.

Note — 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.

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

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

11.3.4 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.

11.4 Modify RDN

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

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

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

ModifyRDNResult  ::= NULL
```

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

11.4.2.1 The **object** argument identifies the entry whose Relative Distinguished Name is to be modified. Aliases in the name shall not be dereferenced. The immediate superior entry shall not have any Non-Specific Subordinate References (see ISO/IEC 9594-4).

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

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

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

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

11.4.4 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 As defined in this part of ISO/IEC 9594, this operation shall only be used on a leaf entry.

12 Errors

12.1 Error Precedence

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

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.

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

a) **NameError**

b) **UpdateError**

c) **AttributeError**

d) **SecurityError**

e) **ServiceError**

12.1.3 The following errors do not present any precedence conflicts:

a) **AbandonFailed**, because it is specific to one operation, **Abandon**, which can encounter no other error;

b) **Abandoned**, which is not reported if an Abandon operation is received simultaneously with the detection of an error. In this case an **AbandonFailed** error, reporting the problem **tooLate** is reported along with the report of the actual error encountered;

c) **Referral**, which is not a "real" error, only an indication that the Directory has detected that the DUA must present its request to another access point.

12.2 Abandoned

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

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

12.2.2 There are no parameters associated with this error.

12.3 Abandon Failed

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

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

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

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

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

a) **noSuchOperation**, when the Directory has no knowledge of the operation which is to be abandoned (this could be because no such invoke took place, or because the Directory has forgotten about it);

b) **tooLate**, when the Directory has already responded to the operation;

c) **cannotAbandon**, when an attempt has been made to abandon an operation for which this is prohibited (e.g., modify), or the abandon could not be performed.

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

12.4 Attribute Error

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

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

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

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

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

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

a) **noSuchAttributeOrValue**. The named entry lacks one of the attributes or attribute values specified as an argument of the operation;

b) **invalidAttributeSyntax**. A purported attribute value, specified as an argument of the operation, does not conform to the attribute syntax of the attribute type;

c) **undefinedAttributeType**. An undefined attribute type was provided as an argument to the operation. This error may occur only in relation to **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 abstract-operation does not conform to the constraints imposed by 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

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

```
NameError ::= ABSTRACT-ERROR
      PARAMETER SET {
            problem       [0]     NameProblem,
            matched       [1]     Name}
      NameProblem         ::=     INTEGER {
            noSuchObject (1),
            aliasProblem (2),
            invalidAttributeSyntax (3) ,
            aliasDereferencingProblem (4) }
```

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

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

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.

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

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

12.6 Referral

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

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

12.6.2 The error has a single parameter which contains a **ContinuationReference** which can be used to progress the operation (see ISO/IEC 9594-4).

12.7 Security Error

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

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

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

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

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

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

```
ServiceError ::= ABSTRACT-ERROR
        PARAMETER SET {
                problem        [0]      ServiceProblem}

ServiceProblem        ::=      INTEGER {
                busy(1),
                unavailable (2),
                unwillingToPerform (3),
                chainingRequired (4),
                unableToProceed (5),
                invalidReference (6)
                timeLimitExceeded (7),
                administrativeLimitExceeded (8),
                loopDetected (9),
                unavailableCriticalExtension (10),
                outOfScope (11),
                ditError (12)}
```

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

a) **busy**. The Directory, or some part of it, is presently too busy to perform the requested operation, but may be able to do so after a short while;

b) **unavailable**. The Directory, or some part of it, is currently unavailable;

c) **unwillingToPerform**. The Directory, or some part of it, is not prepared to execute this request, e.g., because it would lead to excessive consumption of resources or violates the authority 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.

12.9 Update Error

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

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

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

12.9.2 The error has a single 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 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 definition provided by its object class, or with the definitions of 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** operation names an entry which already exists.

f) **affectsMultipleDSAs**. An attempted update would need to operate on multiple DSAs, which is not permitted.

g) **objectClassModificationProhibited**. An operation attempted to modify the object class attribute.

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

Annex A
(Normative)
Abstract Service in ASN.1

This annex includes all of the ASN.1 type, value and macro definitions contained in this part of ISO/IEC 9594 in the form of the ASN.1 module "DirectoryAbstractService".

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

EXPORTS
        directory, readPort, searchPort, modifyPort,
        DirectoryBind, DirectoryBindArgument,
        DirectoryUnbind,
        Read, ReadArgument, ReadResult,
        Abandon, AbandonArgument, AbandonResult,
        Compare, CompareArgument, CompareResult,
        List, ListArgument, ListResult,
        Search, SearchArgument, SearchResult,
        AddEntry, AddEntryArgument, AddEntryResult,
        RemoveEntry, RemoveEntryArgument, RemoveEntryResult,
        ModifyEntry, ModifyEntryArgument, ModifyEntryResult,
        ModifyRDN, ModifyRDNArgument, ModifyRDNResult,
        Abandoned, AbandonFailed, AttributeError, NameError,
        Referral, SecurityError, ServiceError, UpdateError, OPTIONALLY-SIGNED, SecurityParameters;

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

        OBJECT, PORT, ABSTRACT-BIND, ABSTRACT-UNBIND,
        ABSTRACT-OPERATION, ABSTRACT-ERROR
                FROM AbstractServiceNotation {joint-iso-ccitt mhs-motis(6) asdc(2) modules(0) notation(1) }

        Attribute, AttributeType, AttributeValue, AttributeValueAssertion,
        DistinguishedName, Name, RelativeDistinguishedName
                FROM   InformationFramework informationFramework

        id-ot-directory, id-ot-dua, id-pt-read, id-pt-search, id-pt-modify
                FROM DirectoryObjectIdentifiers directoryObjectIdentifiers

        ContinuationReference
        FROM DistributedOperations distributedOperations

        Certificate, CertificationPath, SIGNED,
        PROTECTED, AlgorithmIdentifier
                FROM   AuthenticationFramework authenticationFramework
        InvokeID,
                FROM   Remote-Operations-Notation {joint-iso-ccitt remoteOperations(4) notation(0)};

- - macro for representing optional signing - -

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

-  objects and ports - -
```

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

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

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

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

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

- - bind and  unbind- -

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

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

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

SimpleCredentials ::=            SEQUENCE {   name       [0]     DistinguishedName,
                                              validity   [1]     SET {    time1          [0]     UTCTime OPTIONAL,
                                                                          time2          [1]     UTCTime OPTIONAL,
                                                                          random1        [2]     BIT STRING OPTIONAL,
                                                                          random2        [3]     BIT STRING OPTIONAL}
                                                                                                           OPTIONAL,
                                                          - in most instances the arguments for time and random are relevant in
                                                         - - dialogues employing protected password mechanisms and derive
                                                         - - their meaning as per bilateral agreement
                                              password   [2]     OCTET STRING OPTIONAL}
                                                   - - the value would be an unprotected password, or Protected1 or Protected2
                                                   - - as specified in SO/IEC 9594-8]


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

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