INTERNATIONAL TELECOMMUNICATION UNION

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

X.952

(12/97)

SERIES X: DATA NETWORKS AND OPEN SYSTEM COMMUNICATION

Open distributed processing

Information technology – Open distributed processing – Trading function: Provision of trading function using OSI Directory service

ITU-T Recommendation X.952

(Previously CCITT Recommendation)

ITU-T X-SERIES RECOMMENDATIONS

DATA NETWORKS AND OPEN SYSTEM COMMUNICATION

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 SYSTEM 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
Satellite data transmission systems	X.350–X.399
MESSAGE HANDLING SYSTEMS	X.400–X.499
DIRECTORY	X.500–X.599
OSI NETWORKING AND SYSTEM ASPECTS	
Networking	X.600–X.629
Efficiency	X.630–X.639
Quality of service	X.640–X.649
Naming, Addressing and Registration	X.650–X.679
Abstract Syntax Notation One (ASN.1)	X.680–X.699
OSI MANAGEMENT	
Systems Management framework and architecture	X.700–X.709
Management Communication Service and Protocol	X.710–X.719
Structure of Management Information	X.720–X.729
Management functions and ODMA functions	X.730–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**

For further details, please refer to ITU-T List of Recommendations.

INTERNATIONAL STANDARD 13235-3

ITU-T RECOMMENDATION X.952

INFORMATION TECHNOLOGY – OPEN DISTRIBUTED PROCESSING – TRADING FUNCTION: PROVISION OF TRADING FUNCTION USING OSI DIRECTORY SERVICE

Summary

This Recommendation | International Standard describes how the ODP Trading Function can be realised using information entries and support mechanisms of the OSI Directory. This Specification is to be used in conjunction with the ODP Trading Function Standard (see ITU-T Rec. X.950 | ISO/IEC 13235-1). If there are any discrepancies between the prescriptive statements in ITU-T Rec. X.950 and those in this Specification, the prescriptive statements in ITU-T Rec. X.950 take precedence.

The scope of this Specification is:

- standardised templates for Trading Function information objects in the DIT;
- descriptions of mapping of Trading Function operations to appropriate Directory operations;
- description of use of other Directory features to provide the support mechanisms for implementing the ODP Trading Function.

The field of application of this Specification is for the construction of the ODP Trading Function using the OSI Directory, when required.

Source

The ITU-T Recommendation X.952 was approved on the 12th of December 1997. The identical text is also published as ISO/IEC International Standard 13235-3.

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. The ITU-T is responsible for studying technical, operating and tariff questions and issuing Recommendations on them with a view to standardizing telecommunications on a worldwide basis.

The World Telecommunication Standardization Conference (WTSC), which meets every four years, establishes the topics for study by the ITU-T Study Groups which, in their turn, produce Recommendations on these topics.

The approval of Recommendations by the Members of the ITU-T is covered by the procedure laid down in WTSC Resolution No. 1.

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.

NOTE

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

INTELLECTUAL PROPERTY RIGHTS

The ITU draws attention to the possibility that the practice or implementation of this Recommendation may involve the use of a claimed Intellectual Property Right. The ITU takes no position concerning the evidence, validity or applicability of claimed Intellectual Property Rights, whether asserted by ITU members or others outside of the Recommendation development process.

As of the date of approval of this Recommendation, the ITU had not received notice of intellectual property, protected by patents, which may be required to implement this Recommendation. However, implementors are cautioned that this may not represent the latest information and are therefore strongly urged to consult the TSB patent database.

CONTENTS

Introduction

The ODP Trading Function (see ITU-T Rec. X.950-Series | ISO/IEC 13235) provides the means to offer a service and the means to discover services that have been offered. ITU-T Rec. X.950 | ISO/IEC 13235-1 defines an enterprise Specification, an information Specification and a computational Specification of this Trading Function. No engineering Specification is defined in ITU-T Rec. X.950 | ISO/IEC 13235-1. This Recommendation | International Standard describes how the Specifications of the Trading Function in ITU-T Rec. X.950 | ISO/IEC 13235-1 can be engineered using OSI Directory Service (see ITU-T Rec. X.500 | ISO/IEC 9594-1) to store information and to provide support mechanisms. This Specification does not prescribe that a trader must be engineered by using OSI Directory. But if OSI Directory is used, this Specification defines standardised templates for information entries (e.g. service offer and link information objects) in the Directory DIT.

Clause 5 gives an overview of how the Trading Function is implemented as a combination of X.500 DUA and DSA. The X.500 DSA is used to store the Trader Information Object and a Trader DUA (T-DUA) implements the functionality required by a Trader, which is difficult, or impossible, to implement using OSI Directory services.

Clause 6 defines the standardised templates for information entries of the Trader Information Object, the information known to a particular Trader.

Clause 7 describes mapping of Trading Function operations to appropriate Directory operations.

Clause 8 specifies a minimal Type Repository Function necessary to enable the correct functioning of the X.500 Directory for Trading.

Clause 9 describes the mechanisms used to enable the handling of dynamic properties of a Trader's service offers.

This Specification contains two annexes.

Annex A is a normative schema definition of Trader definitions.

Annex B is an informative schema definition of a sample service description.

INTERNATIONAL STANDARD

ITU-T RECOMMENDATION

INFORMATION TECHNOLOGY – OPEN DISTRIBUTED PROCESSING – TRADING FUNCTION: PROVISION OF TRADING FUNCTION USING OSI DIRECTORY SERVICE

1 Scope and field of application

This Specification describes how the ODP Trading Function can be realised using information entries and support mechanisms of the OSI Directory. This Specification is to be used in conjunction with the ODP Trading Function Standard (ITU-T Rec. X.950 | ISO/IEC 13235-1). If there are any discrepancies between the prescriptive statements in ITU-T Rec. X.950 | ISO/IEC 13235-1 and those in this Specification, the prescriptive statements in ITU-T Rec. X.950 | ISO/IEC 13235-1 take precedence.

The scope of this Specification is:

- standardised templates for Trading Function information objects in the DIT;
- descriptions of mapping of Trading Function operations to appropriate Directory operations;
- description of use of other Directory features to provide the support mechanisms for implementing the ODP Trading Function.

This Specification does not prescribe that a trader must be engineered by using OSI Directory. But if OSI Directory is used, this Specification defines standardised templates for information entries (e.g. service offer and link information objects) in the Directory DIT. This Specification does not put any restrictions on where these entries are placed in the Directory DIT. That is, this Specification does not standardise any structure rules. This Specification does describe a mechanism to provide the Trading Function using OSI Directory.

The field of application of this Specification is for the construction of the ODP Trading Function using the OSI Directory, when required.

2 Normative References

The following Recommendations and International Standards contain provisions which, through reference in this text, constitute provisions of this Recommendation | International Standard. At the time of publication, the editions indicated were valid. All Recommendations and Standards are subject to revision, and parties to agreements based on this Recommendation | International Standard are encouraged to investigate the posibility of applying the most recent edition of the Recommendations and Standards listed below. Members of IEC and ISO maintain registers of currently valid Internationa 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 9594-1:1995, *Information technology – Open Systems Interconnection – The Directory: Overview of concepts, models and services.*
- ITU-T Recommendation X.501 (1993) | ISO/IEC 9594-2:1995, *Information technology – Open Systems Interconnection – The Directory: Models.*
- ITU-T Recommendation X.509 (1993) | ISO/IEC 9594-8:1995, *Information technology – Open Systems Interconnection – The Directory: Authentication framework.*
- ITU-T Recommendation X.511 (1993) | ISO/IEC 9594-3:1995, *Information technology – Open Systems Interconnection – The Directory: Abstract service definition.*
- ITU-T Recommendation X.519 (1993) | ISO/IEC 9594-5:1995, *Information technology – Open Systems Interconnection – The Directory: Protocol specifications.*

- ITU-T Recommendation X.520 (1993) | ISO/IEC 9594-6:1995, *Information technology – Open Systems Interconnection – The Directory: Selected attribute types.*
- ITU-T Recommendation X.521 (1993) | ISO/IEC 9594-7:1995, *Information technology – Open Systems Interconnection – The Directory: Selected object classes.*
- ITU-T Recommendation X.680 (1994) | ISO/IEC 8824-1:1995, *Information technology – Abstract Syntax Notation One (ASN.1): Specification of basic notation.*
- ITU-T Recommendation X.681 (1994) | ISO/IEC 8824-2:1995, *Information Technology – Abstract Syntax Notation One (ASN.1): Information object specification.*
- ITU-T Recommendation X.682 (1994) | ISO/IEC 8824-3:1995, *Information technology – Abstract Syntax Notation One (ASN.1): Constraint specification.*
- ITU-T Recommendation X.683 (1994) | ISO/IEC 8824-4:1995, *Information technology – Abstract Syntax Notation One (ASN.1): Parameterisation of ASN.1 specifications.*
- ITU-T Recommendation X.902 (1995) | ISO/IEC 10746-2:1996, *Information technology – Open distributed processing – Reference Model: Foundations.*
- ITU-T Recommendation X.903 (1995) | ISO/IEC 10746-3:1996, *Information technology – Open distributed processing – Reference Model: Architecture.*
- ITU-T Recommendation X.950 (1997) | ISO/IEC 13235-1[1)], *Information technology – Open distributed processing – Trading function: Specification.*

3 Definitions

This Recommendation | International Standard makes use of the following terms defined in ITU-T Rec. X.902 | ISO/IEC 10746-2:

- activity;
- behaviour;
- client object;
- failure;
- identifier;
- instance;
- interaction;
- interface;
- interface signature;
- name;
- object;
- obligation;
- ODP system;
- policy;
- server object;
- subtype;
- <X> template;
- trading;
- type;
- viewpoint.

1) To be published.

This Recommendation | International Standard makes use of the following terms defined in ITU-T Rec. X.903 | ISO/IEC 10746-3:

- administrator;
- community;
- computational viewpoint;
- engineering interface reference;
- engineering viewpoint;
- enterprise viewpoint;
- exporter;
- importer;
- information viewpoint;
- service export;
- service import;
- service offer;
- technology viewpoint;
- Trading Function;
- Type Repository Function.

This Recommendation | International Standard makes use of the following terms defined in ITU-T Rec. X.950 | ISO/IEC 13235-1:

- federated traders;
- iterator;
- link;
- proxy offer;
- service type;
- service property;
- trader;
- trader attribute;
- trading graph.

This Recommendation | International Standard makes use of the following terms defined in ITU-T Rec. X.500 | ISO/IEC 9594-1:

- Directory;
- Directory Information Base;
- (Directory) User.

This Recommendation | International Standard makes use of the following terms defined in ITU-T Rec. X.501 | ISO/IEC 9594-2:

- attribute;
- attribute type;
- attribute value;
- Directory Information Tree;
- Directory System Agent;
- Directory User Agent;
- distinguished name;
- (Directory) entry;
- filter;

- matching rule;
- (Directory) name;
- name form;
- object;
- object class;
- object entry;
- relative distinguished name;
- structure rule;
- subclass;
- subordinate;
- superclass.

This Recommendation | International Standard makes use of the following operations defined in ITU-T Rec. X.511 | ISO/IEC 9594-3:

- addEntry;
- modifyEntry;
- read;
- removeEntry;
- search.

This Recommendation | International Standard makes use of the following terms defined in ITU-T Rec. X.509 | ISO/IEC 9594-8:

- authentication;
- password.

4 Abbreviations

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

DIB	Directory Information Base
DIT	Directory Information Tree
DN	Distinguished Name
DSA	Directory System Agent
DUA	Directory User Agent
ODP	Open Distributed Processing
OID	Object Identifier
RDN	Relative Distinguished Name
T-DUA	Trader Directory User Agent

5 Overview

In this Specification, the Trading Function is implemented as a combination of X.500 DUA and DSA. As far as possible, the features of X.500 are used to directly implement the Trading Function, but not all Trader features can be directly supported by X.500. For this reason, the Trader (the object that provides the Trading Function) is composed of two components: an X.500 Directory which stores the Trader Information and a Trader DUA (T-DUA) which implements the functionality required by a Trader which is difficult, or impossible, to implement in X.500. The X.500 Directory is used to store the Trader Information Object. Requests from trader clients (importers and exporters) are mapped into operations on the X.500 database. Figure 1 shows the components of a Trader and its interactions with its clients.

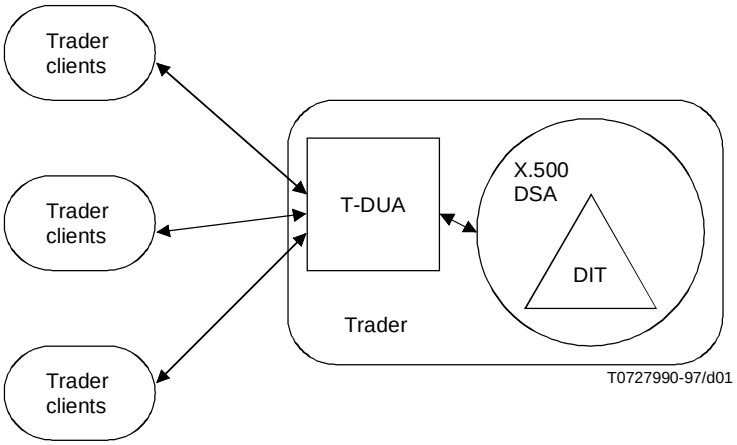

Figure 1 – The trader with its components and clients

The T-DUA and the trader clients (importers and exporters) communicate using a Trader protocol. The Trader protocol is not defined in this Specification. It may be any protocol which implements the functionality specified by ITU-T Rec. X.950 | ISO/IEC 13235-1. The purpose of this Specification is to specify how the T-DUA uses an X.500 Directory to support the functionality specified by ITU-T Rec. X.950 | ISO/IEC 13235-1.

The information stored by the X.500 Directory comprises:

- The Trader Attributes (i.e. information about the Trader itself).
- The Trader Enterprise policies (i.e. rules to determine and guide Trader behaviour).
- The set of Service Offers (i.e. information used by Trader when acting as a server).
- The set of Trader Links (i.e. information used by Trader when acting as a client).
- The set of Proxy Offers (i.e. information used by Trader when acting as a server for Proxy Offers).

X.500 is used to store this information for several reasons:

- The information model required by the ODP Trader is very similar to that provided by X.500.
- X.500 provides significant flexibility in allowing the definition of new X.500 attributes at runtime.
- It makes sense to use the existing investment in X.500 rather than to attempt to create a completely new infrastructure.
- It allows the Trader to use the general X.500 infrastructure to look up presentation addresses of linked Traders and Clients, and to use the security features of X.500 to authenticate users.

  NOTE – Details of how to provide the X.500 infrastructure and the security features of X.500 for the Trading Function are outside the scope of this Specification.

It is not possible to implement an ODP Trader completely using X.500 because of the significant differences in the operational model used by the ODP Trader. These include:

- The Trader operations that do not directly map to X.500 operations.
- Distributed operations that are implemented using information stored in Trader Links and Trader Attributes whose meaning differ from the distribution implemented in X.500.

6 Schema

This X.500 schema describes the portion of an X.500 DIT used to store the information known to one Trader. The schema is based on the X.500 Directory model and is given in Annex A.

6.1 General

The information known to a particular Trader (the Trader Information Object) is kept in a subtree of the X.500 DIT. This subtree can be attached anywhere in the global DIT and no Structure Rules are defined for controlling its position. It is expected that the Trader subtree would be commonly attached beneath organisational and organisational unit entries (representing, respectively, the information known to organisational and organisational unit Traders). The information known to each trader is kept separately in the DIT and no attempt is made to map the distribution model used in X.500 to the very different distribution model of federated Traders.

Trader Information is stored in the X.500 DIT as self contained parcels. Each parcel contains the information known to one Trader. In the example shown in Figure 2, there are two Traders: one for the organisation as a whole and a second for a unit within the organisation. Linkages between these two Traders is via the Trader protocol, not via X.500 protocol.

The Trader Information Object (see Figure 3) is composed of five types of entries:

- The Trader Entry contains details about the Trader itself.
- The Trader Policy Entry contains details about the Trader enterprise policies.
- The Service Offer Entries contain details about the Service Offers known to the Trader.
- The Trader Link Entries contain details about the Links with other Traders.
- The Proxy Offer Entries contain details about the Proxy Offers known to the Trader.

NOTE 1 – The structuring of the Service Offers, Links, and Proxy Offers shown in Figure 3 is only one example of possible information structure.

NOTE 2 – In addition to the X.500 attributes listed in each entry, the presence of other attributes in an entry is not a violation of this Specification. Other X.500 attributes may be required for the following reasons:

- if a particular trader application requires specific additional X.500 attributes, they can be defined in that trader application Specification;
- if a particular trader implementation requires specific additional X.500 attributes, they can be defined in the documentation for that implementation.

Additional attributes can be included as Auxiliary Object Classes.

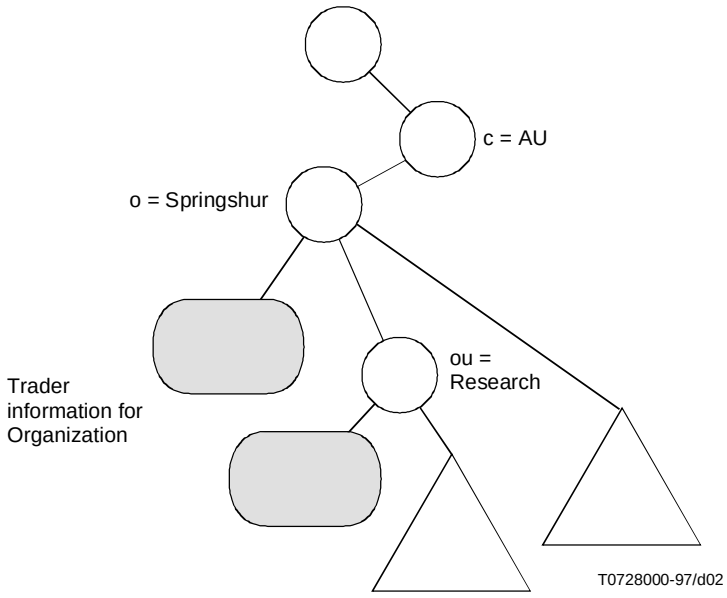

Figure 2 – An example of two traders stored in the X.500 DIT

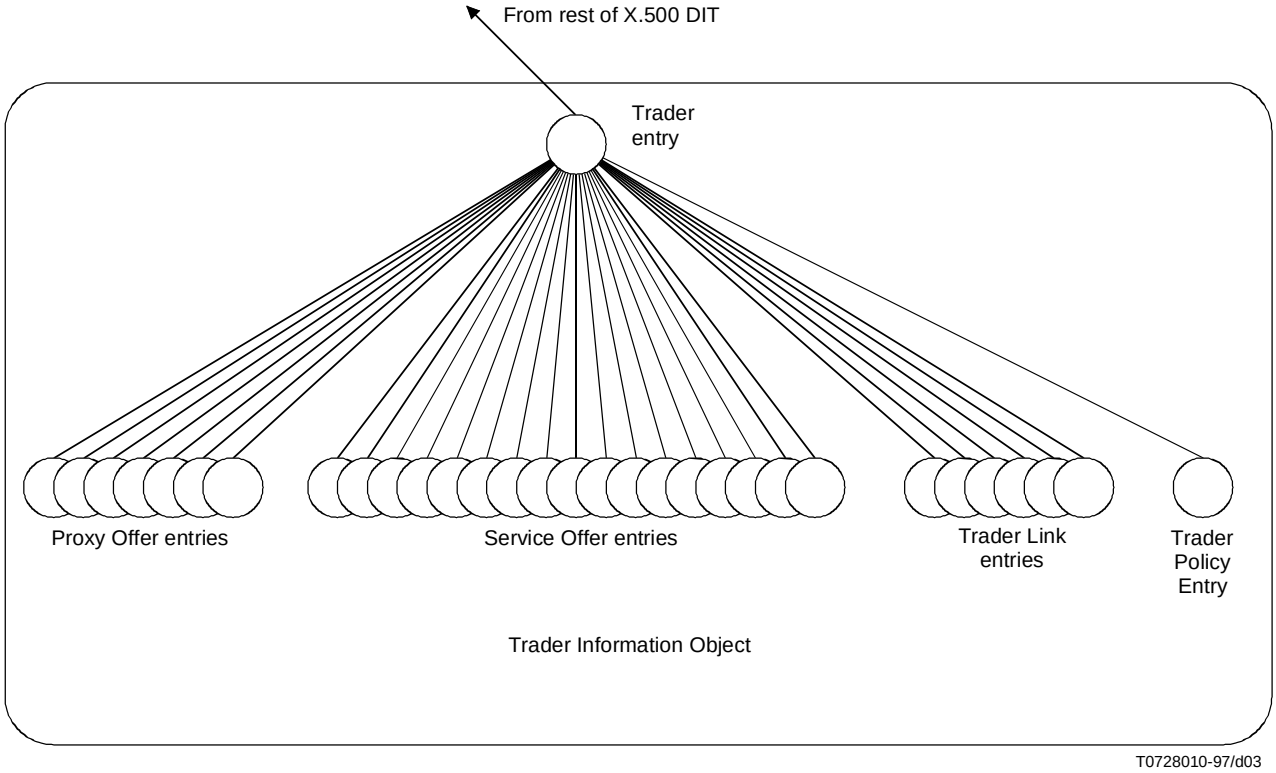

Figure 3 – An example of a Trader Information Object with five types of X.500 entry

6.2 Trader Entry

The root of the Trader subtree is the Trader Entry. This entry contains information about the Trader itself (the Trader Attributes – standardised Trader characteristics and trading policies) and is used as configuration information by the Trader (T-DUA) when it boots. The information is expressed as a set of X.500 attributes which represent Trader Attributes.

```
traderEntry OBJECT-CLASS ::= {

        SUBCLASS OF                     {top}

        MUST CONTAIN                {commonName | traderInterface | dsaName | typeRepos |
                                    defSearchCard | maxSearchCard | defMatchCard |
                                    maxMatchCard | defReturnCard | maxReturnCard |
                                    defHopCount | maxHopCount | defFollowPolicy |
                                    maxFollowPolicy | maxLinkFollowPolicy |
                                    supportsModifiableProperties| supportsDynamicProperties |
                                    supportsProxyOffers | maxList | requestIdStem}

        MAY CONTAIN                     {description | userPassword}

        ID                          id-trader-oc-traderEntry}
```

6.2.1 commonName

The name of this Trader. The commonName attribute forms the RDN of the Trader Entry. The full name of a Trader is the Distinguished Name of this entry (i.e. the full 'pathname' of the Trader Entry in the global X.500 DIT). The full Distinguished Name uniquely identifies this Trader amongst all other Traders in the X.500 Directory. This is a standard X.500 attribute defined in ITU-T Rec. X.520 | ISO/IEC 9594-6.

6.2.2 traderInterface

The address of the trader. The 'Address' is the Presentation Address at which this Trader can be contacted. This X.500 attribute is used by the Trader when booting as part of its configuration information and also by other Traders when they wish to distribute a Trader import amongst federated Traders.

```
traderInterface ATTRIBUTE ::= {
        SUBTYPE OF                presentationAddress
        SINGLE VALUE              TRUE
        ID                        id-trader-at-traderInterface}
```

6.2.3 dsaName

The name for the DSA associated with the Trader object.

```
dsaName ATTRIBUTE ::= {
        SUBTYPE OF                distinguishedName
        SINGLE VALUE              TRUE
        ID                        id-trader-at-dsaName}
```

6.2.4 typeRepos

The name of the Type Repository used by the Trader for the repository of definitions of Service Types, Interface Types and Service Properties Types.

```
typeRepos ATTRIBUTE ::= {
        SUBTYPE OF                distinguishedName
        SINGLE VALUE              TRUE
        ID                        id-trader-at-typeRepos}
```

6.2.5 defSearchCard

The default upper bound of service offers to be considered before terminating a search. This value is used if none is specified by an importer. It must not exceed the value of maxSearchCard.

```
defSearchCard ATTRIBUTE ::= {
        WITH SYNTAX                   INTEGER
        EQUALITY MATCHING RULE    integerMatch
        SINGLE VALUE              TRUE
        ID                        id-trader-at-defSearchCard }
```

6.2.6 maxSearchCard

The maximum upper bound of service offers a Trader considers before terminating any search.

```
maxSearchCard ATTRIBUTE ::= {
        WITH SYNTAX                   INTEGER
        EQUALITY MATCHING RULE    integerMatch
        SINGLE VALUE              TRUE
        ID                        id-trader-at-maxSearchCard}
```

6.2.7 defMatchCard

The default upper bound of matched offers found before a Trader terminates a search. This value is used if none is specified by an importer. It must not exceed the value of maxMatchCard.

```
defMatchCard ATTRIBUTE ::= {
        WITH SYNTAX                  INTEGER
        EQUALITY MATCHING RULE    integerMatch
        SINGLE VALUE              TRUE
        ID                        id-trader-at-defMatchCard}
```

6.2.8 maxMatchCard

The maximum upper bound of matched offers found before a Trader terminates any search.

```
maxMatchCard ATTRIBUTE ::= {
        WITH SYNTAX                  INTEGER
        EQUALITY MATCHING RULE    integerMatch
        SINGLE VALUE              TRUE
        ID                        id-trader-at-maxMatchCard}
```

6.2.9 defReturnCard

The default upper bound of service offers returned to an importer. This value is used if none is specified by an importer. It must not exceed the value of maxReturnCard.

```
defReturnCard ATTRIBUTE ::= {
        WITH SYNTAX                  INTEGER
        EQUALITY MATCHING RULE    integerMatch
        SINGLE VALU    E             TRUE
        ID                        id-trader-at-defReturnCard}
```

6.2.10 maxReturnCard

The maximum upper bound of service offers returned to an importer.

```
maxReturnCard ATTRIBUTE ::= {
        WITH SYNTAX                  INTEGER
        EQUALITY MATCHING RULE    integerMatch
        SINGLE VALUE              TRUE
        ID                        id-trader-at-maxReturnCard}
```

6.2.11 defHopCount

The default upper bound of depth of Links to be traversed before terminating a search. This value is used if none is specified by an importer. It must not exceed maxHopCount.

```
defHopCount ATTRIBUTE ::= {
        WITH SYNTAX                   INTEGER
        EQUALITY MATCHING RULE    integerMatch
        SINGLE VALUE              TRUE
        ID                        id-trader-at-defHopCount}
```

6.2.12 maxHopCount

The maximum upper bound of depth of Links to be traversed before terminating any search.

```
maxHopCount ATTRIBUTE ::= {
        WITH SYNTAX                   INTEGER
        EQUALITY MATCHING RULE    integerMatch
        SINGLE VALUE              TRUE
        ID                        id-trader-at-maxHopCount}
```

6.2.13 defFollowPolicy

The default Link follow behaviour when no link follow behaviour is specified by an importer. The follow behaviour on a Link can be one of the following:

- local_only – Never follow unless explicitly named in an operation;
- if_no_local – Follow only if no local offer can match;
- always – Always follow except when overridden by some policies.

```
defFollowPolicy ATTRIBUTE ::= {
        WITH SYNTAX                   FollowOption
        EQUALITY MATCHING RULE    integerMatch
        SINGLE VALUE              TRUE
        ID                        id-trader-at-defFollowPolicy }

FollowOption ::= ENUMERATED{
        localOnly  (0),
        ifNoLocal  (1),
        always     (2)}
```

6.2.14 maxFollowPolicy

The limiting link follow behaviour on all the links in a Trader for a given Query. It can override both link and importer policies on link follow behaviour.

```
maxFollowPolicy ATTRIBUTE ::= {
        WITH SYNTAX                  FollowOption
        EQUALITY MATCHING RULE    integerMatch
        SINGLE VALUE              TRUE
        ID                        id-trader-at-maxFollowPolicy }
```

6.2.15 maxLinkFollowPolicy

The most permissive link follow behaviour for a link when creating or modifying the limiting behaviour on any link in a trader.

```
maxLinkFollowPolicy ATTRIBUTE ::= {
        WITH SYNTAX                  FollowOption
        EQUALITY MATCHING RULE    integerMatch
        SINGLE VALUE              TRUE
        ID                        id-trader-at-maxLinkFollowPolicy }
```

6.2.16 supportsModifiableProperties

If 'true', this attribute permits the Modify Offer operation to be invoked.

```
supportsModifiableProperties ATTRIBUTE ::= {
        WITH SYNTAX                  BOOLEAN
        EQUALITY MATCHING RULE    booleanMatch
        SINGLE VALUE              TRUE
        ID                        id-trader-at-supportsModifiableProperties}
```

6.2.17 supportsDynamicProperties

If 'true', this attribute permits service offers to have dynamic properties. That is, properties whose values need to be obtained when required for matching or describing of service offers and/or matching of proxy offers.

```
supportsDynamicProperties ATTRIBUTE ::= {
        WITH SYNTAX                  BOOLEAN
        EQUALITY MATCHING RULE    booleanMatch
        SINGLE VALUE              TRUE
        ID                        id-trader-at-supportsDynamicProperties}
```

6.2.18 supportsProxyOffers

If 'true', this attribute permits the export, withdraw, describe and listing of Proxy Offers, which are special service offers that provide run-time determination of the interface at which the advertised service is provided.

```
supportsProxyOffers ATTRIBUTE ::= {
        WITH SYNTAX                   BOOLEAN
        EQUALITY MATCHING RULE    booleanMatch
        SINGLE VALUE              TRUE
        ID                        id-trader-at-supportsProxyOffers}
```

6.2.19 maxList

The maximum list size for an iterator the Trader is willing to support.

```
maxList ATTRIBUTE ::= {
        WITH SYNTAX                   INTEGER
        EQUALITY MATCHING RULE    integerMatch
        SINGLE VALUE              TRUE
        ID                        id-trader-at-maxList }
```

6.2.20 requestIdStem

An identification of the Trader, to be used as the stem for the production of an identifier for a Query request that is initiated by this trader to be passed onto a linked trader.

```
requestIdStem ATTRIBUTE ::= {
        WITH SYNTAX                   OCTET STRING (SIZE (0..ub-request-id-stem))
        EQUALITY MATCHING RULE    octetStringMatch
        SINGLE VALUE              TRUE
        ID                        id-trader-at-requestIdStem }
```

6.2.21 description

A textual description of the Trader. The 'description' is a free text field which describes the Trader (e.g. "CSIRO Division of Information Technology Trader"). This is a standard X.500 attribute defined in ITU-T Rec. X.520 | ISO/IEC 9594-6.

6.2.22 userPassword

A password to provide simple authentication when accessing the Trader information object by the T-DUA. The access control rules for this attribute should not allow this attribute to be generally readable. This is a standard X.500 attribute defined in ITU-T Rec. X.509 | ISO/IEC 9594-8.

6.2.23 Other X.500 attributes

Additional X.500 attributes required by a specific implementation or by a specific application can be included as Auxiliary Object Classes. Examples of possible additional X.500 attributes in this entry are:

- Access Control information (on the Trader Entry).
- Contact information for the Trader Administrator.
- Human contact information for the Service, including a textual description of the service.
- Limit information (e.g. the maximum amount of resource to be consumed by a Query, the maximum lifetime of an Offer).

6.3 Trader Policy Entry

The Trader Policy Entry is located immediately underneath the Trader Entry in the information subtree. It contains the Trader enterprise policies (policies defined in the Enterprise Specification of the Trading Function in ITU-T Rec. X.950 | ISO/IEC 13235-1), expressed as a collection of policy constraints. Each policy constraint may be expressed as a string describing the policy rule which consists of an X.500 filter composed of attributes of the Trader entry, or the name of an object which implements that policy.

```
traderPolicyEntryNF NAME-FORM ::= {
        NAMES                       traderPolicyEntry
        WITH ATTRIBUTES             {commonName}
        ID                          id-trader-nf-traderPolicy}


traderPolicyEntry OBJECT-CLASS ::= {
        SUBCLASS OF                     {top}
        MUST CONTAIN                {commonName }
        MAY CONTAIN                     {typeManagementConstraint | searchConstraint |
                                    offerAcceptanceConstraint }
        ID                          id-trader-oc-traderPolicy}
```

Policy constraints are defined as:

```
PolicySpecification ::= CHOICE {
        stringRule                  [0]     DirectoryString{ub-policy-string-rule}
        policyObjectId              [1]     DistinguishedName }


policySpecificationMatch MATCHING-RULE ::= {
        SYNTAX                      PolicySpecification
        ID                          id-trader-mr-policySpecificationMatch}
-- The rule returns TRUE if two Specifications contain exactly the same characters.
```

6.3.1 commonName

The commonName attribute forms the RDN of the Trader Policy Entry and has the value "Trader Policies". This is a standard X.500 attribute defined in ITU-T Rec. X.520 | ISO/IEC 9594-6.

6.3.2 typeManagementConstraint

This constraint is a rule related to the Specification of types and the relationship between types.

```
typeManagementConstraint ATTRIBUTE ::= {
        WITH SYNTAX                     PolicySpecification
        EQUALITY MATCHING RULE      policySpecificationMatch
        SINGLE VALUE                TRUE
        ID                          id-trader-at-typeManagementConstraint}
```

6.3.3 searchConstraint

This constraint is a rule guiding the search for suitable offers through the Trader system, for example, sequential search vs parallel search of federated traders.

```
searchConstraint ATTRIBUTE ::= {
        WITH SYNTAX                  PolicySpecification
        EQUALITY MATCHING RULE    policySpecificationMatch
        SINGLE VALUE              TRUE
        ID                        id-trader-at-searchConstraint}
```

6.3.4 offerAcceptanceConstraint

This constraint is a rule restricting the set of service offers acceptable to the Trader. For example, a trader may only accept service offers with specific service types.

```
offerAcceptanceConstraint ATTRIBUTE ::= {
        WITH SYNTAX                  PolicySpecification
        EQUALITY MATCHING RULE    policySpecificationMatch
        SINGLE VALUE              TRUE
        ID                        id-trader-at-offerAcceptanceConstraint}
```

6.3.5 Other X.500 attributes

Additional X.500 attributes required by a specific implementation or application can be included as Auxiliary Object Classes. Examples of possible additional X.500 attributes in this entry are:

- Access Control information (on the Trader Policy Entry).
- Additional Trader enterprise policy constraints.

6.4 Service Offer Entry

Zero or more Service Offer Entries are immediately underneath the Trader Entry in the information subtree. Each Service Offer Entry contains the details of one Service Offer. Collectively, the Service Offer Entries form the set of Service Offers. Every Service Offer Entry has a structural object class of serviceOfferEntry and exactly one auxiliary object class of those listed in the content rule for the Service Offer Entry. Each auxiliary object class corresponds to a service type and defines the service properties that are mandatory or optional. Every Service Offer Entry therefore contains the mandatory and optional attributes of the Service Offer Entry, and the mandatory and optional attributes determined by the corresponding auxiliary object class, specifying the mandatory and optional service properties. Each Service Offer Entry is named by using the sOfferId attribute of the entry.

```
serviceOfferEntryNF NAME-FORM ::= {
        NAMES                     serviceOfferEntry
        WITH ATTRIBUTES           {sOfferId}
        ID                        id-trader-nf-serviceOffer}
serviceOfferEntry OBJECT-CLASS ::= {
        SUBCLASS OF                   {top}
        MUST CONTAIN              {sOfferId | serviceInterfaceId |serviceTypeId
                                  hasDynamicProperties | hasModifiableProperties }
        MAY CONTAIN                   {dynamicProps}
        ID                        id-trader-oc-serviceOffer}
```

Service properties in a Service Offer are stored as X.500 attributes. The particular attributes (mandatory service properties) which must be present in a Service Offer and those which are allowed (optional service properties) in a Service Offer are controlled by the X.500 schema. The Property Name maps to an X.500 Attribute Type and the Property Values map to X.500 Attribute Values. Property Names are object identifiers. The service properties are controlled by the auxiliary object class associated with the Service Type of the Service Offer. In addition, a Property in a Service Offer can be controlled by access control rules to be non-modifiable by a user (that is, read only).

Each Service Type Identifier has an associated Interface Type and an associated Service Properties Type which can be obtained from the Type Repository. The Exporter must specify a valid Service Type Identifier known to the Trader. The associated Service Properties Type Identifier is represented as an X.500 Auxiliary Object Class and is consequently stored in each Service Offer Entry as a value in the Object Class attribute. This Auxiliary Object Class defines the mandatory and optional X.500 attributes (service properties) which must/may be contained in a Service Offer of this Service Type.

Auxiliary Object Classes may be derived from other X.500 Object Classes (known as 'subclassing' in X.500). This is equivalent to deriving the definition of one Service Properties Type from another. The rules for subclassing may best be understood in relation to the must and may contain lists:

- The subclass must contain any attributes of the 'must contain list' of its superclass.
- The subclass may contain any attributes of the 'may contain list' of its superclass.
- An object class may be a subclass of two or more other classes. The subclass must contain all the attributes in all of the 'must contain lists' of all its superclasses and 'may contain' any of the attributes of its superclasses.

NOTE – In the definition of Service Offer Entries, the structural object class of a Service Entry is the generic 'Service Offer', while the 'Service Properties Type' which specialises the Service Offer to a particular service type is an auxiliary object class. It would have been possible to make 'Service Offer' an abstract object class, and then the various 'Service Properties Types' would have been structural object class derived from the Service Offer abstract class. That would have the disadvantage of requiring a complex X.500 schema with many rules.

For a given offer, an exporter can also nominate that a particular property has a dynamic value. That is, the value of the property is not specified at the time of export but to be evaluated when required by matching or describing service offers (for example, by use of the Trader operations of Query or Describe). Instead, the exporter specifies the interface at which the property can be evaluated.

6.4.1 sOfferId

The sOfferId is the identifier of the Service Offer Entry which is allocated by the Trader. It forms the RDN of the Service Offer Entry. The identifier is chosen to be unique amongst all other Service Offer Identifiers held by that Trader. The sOfferId forms part of the Service Offer Identifier which the T-DUA passes back to the exporter of the service offer after the successful processing of an Export operation request.

NOTE 1 – The T-DUA may choose names using any algorithm – provided the resulting identifiers are unique. An incrementing counter is sufficient.

```
sOfferId ATTRIBUTE ::= {
        WITH SYNTAX                   DirectoryString{ub-s-offer-id}
        EQUALITY MATCHING RULE    caseIgnoreMatch
        SUBSTRINGS MATCHING RULE  caseIgnoreSubstringsMatch
        SINGLE VALUE              TRUE
        ID                        id-trader-at-sOfferId}
```

NOTE 2 – The exporter receives a Service Offer Identifier which is the string representation of the partial Directory Name (under the root of the Trader) of the Service Offer entry using ASN.1 value notation. For the example in Figure 3, a Service Offer Identifier returned to the exporter could be {sOfferId, 5}.

6.4.2 serviceInterfaceId

The Service Interface contains the name of an Interface Entry (see 6.7) which contains information of the interface of the exported service. The Trader uses this name to read the Interface Entry to retrieve information (e.g. the presentation address or the engineering interface reference) about this interface.

```
serviceInterfaceId ATTRIBUTE ::= {
        SUBTYPE OF                distinguishedName
        SINGLE VALUE              TRUE
        ID                        id-trader-at-serviceInterfaceId}
```

NOTE – When dealing with only one infrastructure, an alternative is to replace the above serviceInterfaceId Specification by:

```
        ServiceIntId ::= SEQUENCE {
            serviceInterfaceType    DirectoryString{ub-service-interface-type}
            interfaceRef            OPEN TYPE}

        serviceInterfaceId ATTRIBUTE ::= {
            WITH SYNTAX             ServiceIntId
            EQUALITY MATCHING RULE  caseExactMatch
            ID                      id-trader-at-serviceInterfaceId}
```

6.4.3 serviceTypeId

The serviceTypeId is the identifier of the Service Type of the service that is being on offer. The Service Type Identifier controls the Interface Type and the Service Properties Type that can be associated with this service offer.

```
serviceTypeId ATTRIBUTE ::= {
        WITH SYNTAX                 OBJECT IDENTIFIER
        EQUALITY MATCHING RULE    objectIdentifierMatch
        ID                        id-trader-at-serviceTypeId }
```

6.4.4 hasDynamicProperties

If set to 'true', this attribute indicates that this service offer contains dynamic properties. That is, properties whose values need to be obtained when required for matching or describing service offers. In its import policy, an importer may request not to consider offers with dynamic properties.

```
hasDynamicProperties ATTRIBUTE ::= {
        WITH SYNTAX                 BOOLEAN
        EQUALITY MATCHING RULE    booleanMatch
        SINGLE VALUE              TRUE
        ID                        id-trader-at-hasDynamicProperties}
```

6.4.5 hasModifiableProperties

If set to 'true', this attribute indicates that this Service Offer contains properties that are not Read Only. That is, properties whose values can be changed when required by the exporter. In its import policy, an importer may request not to consider offers with modifiable properties.

```
hasModifiableProperties ATTRIBUTE ::= {
        WITH SYNTAX                  BOOLEAN
        EQUALITY MATCHING RULE    booleanMatch
        SINGLE VALUE              TRUE
        ID                        id-trader-at-hasModifiableProperties}
```

6.4.6 dynamicProps

The dynamicProps attribute is set when a Service Offer contains dynamic properties. Dynamic properties are properties whose values are to be obtained when they are needed for matching or describing service offers. Each exporter can nominate which properties (mandatory or optional) are dynamic. To be nominated for dynamic evaluation of a property, however, the property must not be a Read Only property. The dynamicProps attribute contains the information required to obtain the values for one or more dynamic properties. A dynamicPropValue is set for each dynamic property in a Service Offer.

```
dynamicProps ATTRIBUTE ::= {
        WITH SYNTAX                  SEQUENCE OF DynamicPropValue
        ID                        id-trader-at-dynamicProps}

dynamicPropValue ::= SEQUENCE {
        propertyType              OBJECT IDENTIFIER,
        dynamicPropEvalIf         DistinguishedName,
        extraInfo                 DirectoryString{ub-dynamic-value-extra-info}}

DynamicPropValueMatch MATCHING-RULE ::=(
        SYNTAX                    DynamicPropValue,
        ID                        id-trader-mr-dynamicPropValueMatch}
-- The rule returns TRUE if two values contain exactly the same characters.
```

6.4.7 Other X.500 attributes

In addition to the attributes already listed, a Service Offer Entry normally contains other attributes which represent the service properties. The service property attributes are specific to the type of service and are controlled by an auxiliary object class. The Service Properties Type specifies whether a property is mandatory or optional, and (with access control information) whether a user can modify its value. The dynamicProps attribute is specific to a particular offer, for example, an offer of a particular service by an exporter may have no dynamic values while a different Service Offer of the same service type may contain several dynamic properties.

Additional X.500 attributes required by a specific implementation or application can also be included. Examples of possible additional X.500 attributes in this entry are:

- Access Control information (on the Service Offer Entry).
- Expiry date of a Service Offer.
- Human contact information for the service, including a textual description of the service.

6.5 Trader Link Entry

Zero or more Trader Link Entries are immediately underneath the Trader Entry in the information subtree. Each Trader Link Entry contains the details of one Trader Link. Collectively the Trader Link Entries form the set of Trader Links. Each Trader Link Entry is named by using the linkId attribute of this entry.

```
traderLinkEntryNF NAME-FORM ::= {
        NAMES                       traderLinkEntry
        WITH ATTRIBUTES             {linkId}
        ID                          id-trader-nf-traderLink}

traderLinkEntry OBJECT-CLASS ::= {
        SUBCLASS OF                     {top}
        MUST CONTAIN                {linkName | linkId | targetTraderInterfaceId |
                                    defPassOnFollowRule | limitingFollowRule}
        ID                          id-trader-oc-traderLink}
```

6.5.1 linkName

The linkName is the name of the Link as supplied by the administrator of the source trader when creating the Link. The name must be unique amongst all Link names held by the Trader.

```
linkName ATTRIBUTE ::= {
        WITH SYNTAX                     DirectoryString {ub-link-name}
        EQUALITY MATCHING RULE      caseIgnoreMatch
        SUBSTRINGS MATCHING RULE    caseIgnoreSubstringsMatch
        SINGLE VALUE                TRUE
        ID                          id-trader-at-linkName}
```

6.5.2 linkId

The linkId is the identifier of the Link which is allocated by the Trader. It forms the RDN of a Trader Link Entry. The identifier must be unique amongst all other Trader Link Identifiers held by that Trader.

NOTE – The T-DUA may choose names using any algorithm – provided that the resulting identifiers are unique. An incrementing counter is sufficient.

```
linkId ATTRIBUTE ::= {
        WITH SYNTAX                     DirectoryString {ub-link-id}
        EQUALITY MATCHING RULE      caseIgnoreMatch
        SUBSTRINGS MATCHING RULE    caseIgnoreSubstringsMatch
        SINGLE VALUE                TRUE
        ID                          id-trader-at-linkId}
```

6.5.3 targetTraderInterfaceId

This attribute contains the name of the interface of the target Trader pointed to by the Link. The name is used to look up the presentation address of the remote Trader.

```
targetTraderInterfaceId ATTRIBUTE ::= {
        SUBTYPE OF                distinguishedName
        SINGLE VALUE              TRUE
        ID                        id-trader-at-targetTraderInterfaceId}
```

6.5.4 defPassOnFollowRule

The default link follow behaviour on a Link is specified when a Link is created. It must not exceed the limitingFollowRule for this link.

```
defPassOnFollowRule ATTRIBUTE ::= {
        WITH SYNTAX                   FollowOption
        EQUALITY MATCHING RULE    integerMatch
        SINGLE VALUE              TRUE
        ID                        id-trader-at-defPassOnFollowRule }
```

6.5.5 limitingFollowRule

The limiting link follow behaviour on a particular Link is specified when the link is created. Its value must not exceed the source Trader's maxLinkFollowPolicy at the time of the Link creation. However, it is permissible if a later change in the Trader's maxLinkFollowPolicy causes the limitingFollowRule to exceed the value of the changed Trader maxLinkFollowPolicy.

```
limitingFollowRule ATTRIBUTE ::= {
        WITH SYNTAX                   FollowOption
        EQUALITY MATCHING RULE    integerMatch
        SINGLE VALUE              TRUE
        ID                        id-trader-at-limitingFollowRule }
```

6.5.6 Other X.500 attributes

In addition to the X.500 attributes already listed, a Trader Link Entry normally contains other attributes which represent information of the target trader as perceived by the source trader. These Link attributes are used by the source Trader to help decide which Link to follow next. These X.500 attributes and other additional X.500 attributes required by a specific implementation or application can be included as Auxiliary Object Classes. Examples of possible additional X.500 attributes in this entry are:

- Access Control information (on the Trader Link Entry).
- Attributes about the linked (remote) Trader Entry, including human contact information for the linked Trader and a textual description of the linked trader.
- The period over which this Link is valid.
- The last time the Link was invoked.

6.6 Proxy Offer Entry

Zero or more Proxy Offer Entries are immediately underneath the Trader Entry in the information subtree. There need not be any Proxy Offer Entries if the Trader Attribute of supportsProxyOffers is set to 'false'. If the supportsProxyOffers attribute of the Trader entry is set to 'true', then each Proxy Offer Entry contains the details of one Proxy Offer. Collectively, the Proxy Offer Entries form the set of Proxy Offers. Every Proxy Offer Entry has a structural object class of proxyOfferEntry and exactly one auxiliary object class of those listed in the content rule for the Proxy Offer Entry. Each auxiliary object class corresponds to a Service Properties Type and defines the service properties that are mandatory or optional. Every Proxy Offer Entry therefore contains the mandatory and optional attributes of the Proxy Offer Entry, and the mandatory and optional attributes determined by the corresponding auxiliary object class, specifying the mandatory and optional service properties. Each Proxy Offer Entry is named by using the proxyOfferId attribute of the entry.

```
proxyOfferEntryNF NAME-FORM ::= {

        NAMES                       proxyOfferEntry

        WITH ATTRIBUTES             {proxyOfferId}

        ID                          id-trader-nf-proxyOffer}


proxyOfferEntry OBJECT-CLASS ::= {

        SUBCLASS OF                     {top}

        MUST CONTAIN                {proxyOfferId | proxyLookUpInterfaceId |
                                    hasDynamicProperties | hasModifiableProperties |
                                    ifMatchAll | constraintRecipe}

        MAY CONTAIN                     { dynamicProps }

        ID                          id-trader-oc-proxyOffer}
```

A Proxy Offer is a cross between a Service Offer and a form of a restricted Link. Like normal Service Offers, Proxy Offers have a service type and a set of service properties. However, a Proxy Offer does not include an interface at which the offered service is provided. Instead, it provides a LookUp Interface, from which a modified Query operation is invoked and from which a run-time determination of the interface to provide the service is obtained.

In addition, a Proxy Offer has a constraintRecipe (used to formulate a modified constraint on the modified query). A Proxy Offer can also have a sequence of name-value pairs to be appended to the original importer policy to form the importer policy on the modified Query.

6.6.1 proxyOfferId

The proxyOfferId is the RDN of the entry and is chosen to be unique amongst all other Proxy Offer names held by that Trader. The proxyOfferId forms part of the Proxy Offer Identifier which the T-DUA passes back to the exporter of the Proxy Offer after the successful processing of an Export Proxy operation request.

NOTE 1 – The T-DUA may choose names using any algorithm – provided the resulting names are unique. An incrementing counter is sufficient.

NOTE 2 – The exporter receives a Proxy Offer Identifier which is the string representation of the partial Directory Name (under the root of the Trader) of the Proxy Offer Entry using ASN.1 value notation.

```
proxyOfferId ATTRIBUTE ::= {

        WITH SYNTAX                     DirectoryString {ub-proxy-offer-id}

        EQUALITY MATCHING RULE      caseIgnoreMatch

        SUBSTRINGS MATCHING RULE    caseIgnoreSubstringsMatch

        SINGLE VALUE                TRUE

        ID                          id-trader-at-proxyOfferId}
```

6.6.2 proxyLookUpInterfaceId

The Proxy LookUp Interface Identifier is the name of an entry which represents an Interface that can accept a Trader Query operation. The Trader uses this name to read the entry to retrieve information (e.g. the presentation address) about this interface.

```
proxyLookUpInterfaceId ATTRIBUTE ::= {
        SUBTYPE OF                distinguishedName
        SINGLE VALUE              TRUE
        ID                        id-trader-at-proxyLookUpInterfaceId}
```

6.6.3 constraintRecipe

The constraintRecipe attribute provides the recipe to convert an importer's matching constraint in the original Query request to a new matching constraint to be used in the modified Query request at the Proxy LookUp Interface. The constraint language is specified in Annex C of ITU-T Rec. X.950 | ISO/IEC 13235-1.

```
constraintRecipe ATTRIBUTE ::= {
        WITH SYNTAX                   DirectoryString {ub-constraint-recipe}
        EQUALITY MATCHING RULE    caseIgnoreMatch
        SUBSTRINGS MATCHING RULE  caseIgnoreSubstringsMatch
        SINGLE VALUE              TRUE
        ID                        id-trader-at-constraintRecipe}
```

6.6.4 ifMatchAll

If set to 'true', then the trader should match this Proxy Offer to an importer's Query on Service Type only; that is, there is no requirement to match the importer's 'constraint' expression against the properties associated with this Proxy Offer for a search.

```
ifMatchAll ATTRIBUTE ::= {
        WITH SYNTAX                   BOOLEAN
        EQUALITY MATCHING RULE    booleanMatch
        SINGLE VALUE              TRUE
        ID                        id-trader-at-ifMatchAll }
```

6.6.5 Other X.500 attributes

In addition to the attributes already listed, a Proxy Offer entry, like a Service Offer Entry, normally contains other attributes which represent the service properties, and whether there are dynamic properties and modifiable properties. The service property attributes are specific to the type of service and are controlled by an auxiliary object class. A property attribute type specifies that an attribute is mandatory or optional, and (with access control information) whether a user can modify its value. If there are dynamic properties, then they are stored in dynamicProps attribute (see 6.4.6).

The Trading Function Specification in ITU-T Rec. X.950 | ISO/IEC 13235-1 also allows additional name-value pairs (policies_to_pass_on) to be included in a Proxy Offer. These name-value pairs, if present, are to be appended onto the policy parameter of the original Query request, in order to pass these name-value pairs to a service factory for use during service creation by the service factory. The name-value pairs are represented as X.500 Attribute Type and Attribute Value pairs.

Other additional X.500 attributes required by a specific implementation or application can be included. Examples of possible additional X.500 attributes in this entry are:

- Access Control information (on the Proxy Offer Entry).
- Human contact information for the Proxy Offer, including a textual description of the service.

6.7 Other X.500 entries used by the T-DUA

In addition to the five types of entry in the Trader Information Object, the T-DUA may also make use of other entries in the X.500 database. These include entries representing:

- Individuals, Organisational Units, or Organisations which may store authentication information.
- Interfaces, which contain information about the interface, for example, the address (or an engineering interface reference) used to access that interface. Part 1 of the Trader Function (see ITU-T Rec. X.950 | ISO/IEC 13235-1) specifies that an engineering interface reference needs to be specified for each infrastructure. To maintain the maximum flexibility, we represent each interface of the computational Specification by a separate Directory Entry. The Directory Entry can have attributes tailored for each infrastructure as required. This adds a level of indirection to accessing interface addresses; the Trader stores the name of an entry containing interface information and that entry is accessed to retrieve the address. An advantage is that if the address at which the interface is accessed changes, then only one entry needs to be changed, instead of all references to that interface in all Traders. This is particularly important where the interface needs to be widely known by many Traders – a Type Repository for example.

  Interface entries contain an interfaceReference attribute and an interfaceType attribute. The Object Identifier for the Interface Type must be known to the Type Repository.

```
interfaceEntry OBJECT-CLASS ::= {
        SUBCLASS OF                  {top}
        KIND                      auxiliary
        MUST CONTAIN              {interfaceReference | interfaceType}
        ID                        id-trader-oc-interfaceEntry}


interfaceReference ATTRIBUTE ::= {
        WITH SYNTAX                  InterfaceId
        EQUALITY MATCHING RULE    caseExactMatch
        ID                        id-trader-at-interfaceReference}


InterfaceId    ::=               DirectoryString {ub-interface-id}


interfaceType ATTRIBUTE ::= {
        WITH SYNTAX                  InterfaceTypeName
        EQUALITY MATCHING RULE    objectIdentifierMatch
        ID                        id-trader-at-interfaceType}
        InterfaceTypeName ::=     OBJECT IDENTIFIER
```

7 Operations

This clause describes the mapping between the Trader functions specified in ITU-T Rec. X.950 | ISO/IEC 13235-1 and the operations the T-DUA performs on the X.500 Directory to implement these functions. It is assumed that exporters and importers communicate with the T-DUA via a Trader protocol which supports the functionality required in ITU-T Rec. X.950 | ISO/IEC 13235-1.

7.1 Initialisation

When setting up a new Trader, the Trader Entry is created by the administrator (either directly or via a program which reads configuration information). The new entry contains the configuration information for the new Trader.

Whenever the Trader (T-DUA) is started, it binds to the Directory and reads the Trader Entry to discover the information about itself. The only bootstrap information needed by the T-DUA is:

- The Distinguished Name of its Trader Entry.
- A Password.

The T-DUA binds to the X.500 DSA when the Trader is created and remains bound until the Trader is destroyed (except for crashes of either the Trader or the X.500 system). The Trader must bind to the DSA with sufficient privileges to modify the Trader Entry, and to create, read, remove, and modify the Service Offer Entries, the Trader Link Entries and, if Proxy Offer is supported, to create, read and remove Proxy Offer entries.

7.2 Client operations

A client uses a Trader protocol to communicate with the T-DUA. This protocol is not defined further in this Specification. The operation requested by the client results in one or more operations on the X.500 database. The mapping between the Trader operations and the X.500 operations are described in the following subclauses. A mapping takes place only after the T-DUA has checked for the syntax of each parameter as specified in Annex A of ITU-T Rec. X.950 | ISO/IEC 13235-1; a syntactically incorrect parameter is returned as an exception with the prefix 'illegal' and the operation is not processed further.

Several Trader operations require some form of security (e.g. only a client acting in an administrator role can add or delete Links). This security is implemented by taking advantage of the security provided by X.500. To perform a Trader operation, a client must supply a name and a password. In this Specification these are assumed to be, or to directly map to, an X.500 user name and user password, possibly protected in some form. How this is done depends on the Trader protocol used to bind a client to a Trader.

The T-DUA uses this authentication to bind to the X.500 DSA as that user. X.500 provides facilities to protect the password when it is being sent across the network between the T-DUA and the X.500 DSA. There is a security implication in this approach: the Client must trust the T-DUA, as the T-DUA can see the Client's password. The X.500 protocol provides mechanisms for protecting passwords when transmitted between DUAs and DSAs. It should be feasible to extend this protection to operate between a Client and the DSA since the T-DUA does not need to know the password.

Once bound as the Client to the DSA, the T-DUA uses the X.500 security features to control access to entries in the Trader information tree. No specific access control exception is defined in ITU-T Rec. X.950 | ISO/IEC 13235-1, and a system specific exception is returned when access is not allowed (for example, in CORBA, the standard exception of No_Permission is raised).

7.3 Register operations

The Register operations form a set of operations that allow a client to export, withdraw, modify, and describe Service Offers in a trader. In addition, there is an operation to resolve a context relative name of a trader.

7.3.1 Export

The Trader receives an Export operation. This operation is mapped to an X.500 addEntry operation to add a new Service Offer Entry to the X.500 Directory. The data in the Export operation are:

Parameter	X.500 attribute syntax	Comment
reference	Name	The name of the entry in the X.500 database which contains the address of the Service Interface. See 6.7.
type	Object Identifier	The Type specifies the service type. It is assumed to be the Service Type Identifier. In the X.500 Trader, it is used to obtain the Auxiliary Object class for the new entry.
properties	Any (Optional)	The actual X.500 attributes supplied by the exporter for the service properties are controlled by the Service Properties Type Identifier.

The T-DUA maps the export operation to an X.500 addEntry. The name of the new entry is selected by the T-DUA such that it is unique amongst all Service Offers. Any scheme may be used; a simple method would be an incremental counter. The structural object class of the new entry is serviceOffer. The auxiliary object class is the Service Properties Type Identifier (OID) associated with the Service Type Identifier specified by the Exporter. The T-DUA uses its Type Repository to obtain the Interface Type Identifier and the Service Properties Type Identifier associated with the Service Type.

Access control information must be included in the new entry to prevent other X.500 users from accessing or modifying the trader information. Additional access control information is added to allow the X.500 DSA to control access to trader information by the Clients. This can be extended to only allowing certain clients the privilege of exporting services.

The reference parameter gives the name of the interface that provides the service. This entry is read to obtain the type associated with the interface. If it can be determined that this type is not a subtype of the Interface Type associated with the specified Service Type, then an InterfaceTypeMismatch exception may be returned. The exception InvalidInterfaceRef may be raised if the parameter contains 'nil'.

If the Service Type Identifier or the Auxiliary Object class is not known to the Trader, the DSA returns an Update error of type objectClassViolation. The T-DUA returns an exception of UnknownServiceType. The Service Type Identifier is stored in the Service Offer entry.

The Service Property Values which may/must be present in the new entry are controlled by the Auxiliary Object Class (Service Properties Type Identifier) of the entry. If the exporter does not supply the necessary mandatory attributes for the specified Service Properties Type Identifier, the DSA returns an Update error of type objectClassViolation. The T-DUA returns an exception of MissingMandatoryProperty. Similarly, if the user supplies service properties which are not listed in the definition of the Service, an Update error of type objectClassViolation is returned. (The current specification in ITU-T Rec. X.950 | ISO/IEC 13235-1 allows exporter to include an occasional optional property that is not included in the Service Properties Type. However, this is not allowed in X.500. The exporter must first specify a subtype that includes the additional property before exporting). The schema checking in X.500 does not check that the service properties supplied by the Exporter are valid (e.g. that the printer type is one of the three valid values). This checking has to be performed by the T-DUA and the PropertyTypeMismatch exception is raised if the property value is not of the required type. If an Attribute Error of undefinedAttributeType occurs, the T-DUA can directly map this onto a PropertyTypeMismatch exception.

The T-DUA also checks for the presence of dynamic properties in the offer. For each dynamic service property identified (see clause 9), the T-DUA substitutes its value by a 'dummy' value and creates an associated dynamicPropValue in the dynamicProps attribute of the entry. The T-DUA also sets the hasDynamicProperties attribute of the Offer Entry to 'true' if there are dynamic values in the offer. If a property with a dynamic value is non-modifiable according to the access control information on the property, then a ReadonlyDynamicProperty exception is returned.

If a Trader does not support dynamic properties (that is, the Trader Attribute supportsDynamicProperties is set to 'false') and there are dynamic properties in the offer, then the exception PropertyTypeMismatch is raised and the Service Offer is not accepted.

If all the properties of the service are non-modifiable, then the hasModifiableProperties attribute of the offer entry is set to 'false'.

If two or more properties with the same property name are included in the property list, then the DuplicatePropertyName exception is raised.

If the export is successful, the T-DUA returns success with the following information:

Parameter	X.500 attribute syntax	Comment
offerId	DirectoryString	A Service Offer Identifier

where the offerId is the string representation of a partial Directory Name, which gives the pathname from the Trader Entry to the new Service Offer. The RDN (sOfferId) of the new Service Offer Entry forms the final RDN in the sequence of names.

7.3.2 Withdraw

The T-DUA receives a request to delete a Service Offer. The Withdraw operation is mapped to an X.500 removeEntry operation to remove the Service Offer entry. The data in the Withdraw operation are:

Parameter	X.500 attribute syntax	Comment
id	DirectoryString	A Service Offer identifier

The entry is identified by the id, which is the partial DN that uniquely identifies the Service Offer. This operation simply removes the Service Offer Entry.

If the Client supplies an unknown Offer Identifier, the removeEntry operation returns a nameError of noSuchObject. The T-DUA then uses the partial DN to try to read a Proxy Offer Entry. If the read is successful, then the exception of NotServiceOfferId is raised. If the read is unsuccessful, then the exception of UnknownOfferId is raised.

Access control is enforced by the DSA based on the access control information stored when the Service Offer was exported and on the Client Identifier of the Withdraw operation. If the Client has insufficient access rights, a Security Error of insufficientAccessRights is returned to the T-DUA which is mapped to system specific exception. If the implementation requires the Trader administrator to purge 'old' Offers, then the administrator must also have permission to remove offers.

7.3.3 Modify

The Trader receives a request to modify the property values of a Service Offer. The Modify operation is mapped to an X.500 read operation and then an X.500 modifyEntry operation to change the Service Offer Entry. The data in the modify operation are:

Parameter	X.500 attribute syntax	Comment
id	DirectoryString	A Service Offer Identifier
del_list	set of OIDs (optional)	List of service properties to be deleted from the offer
modify_list	Any (optional)	List of service properties whose values are to be added or replaced

The T-DUA first reads the Service Offer entry identified by the Service Offer Identifier. If this entry does not exist, the DSA returns a nameError of noSuchObject, which is mapped to an UnknownOfferId Trader exception. If the client does not have sufficient permission to read the entry, a security error of insufficientAccessRights is returned to the T-DUA. This X.500 read operation is performed so that the T-DUA can build a list of the property attributes actually present in the Service Offer Entry. The T-DUA modifies the entry accordingly: properties from the deleteList are removed, existing property values are changed, and new property values are added.

An X.500 modifyEntry operation is then constructed which first deletes all the existing service properties and then adds the properties from the modified offer. The parameters for this new Service Offer have the same name and function as the Export operation. If a name in the del_List does not exist in the original offer, then the UnknownPropertyName exception is returned.

The newly constructed entry may violate the schema (as represented by the Auxiliary Object Class – the Service Properties Type Identifier). In this case, when the X.500 modifyEntry operation is performed, the DSA returns an Update Error of object class violation, and the existing Service Offer remains unchanged. The T-DUA maps the Update Error to various exceptions. A MandatoryProperty exception is returned if a mandatory property is deleted but not added back. Access control is enforced by the DSA if the operation tries to change the value of a non-modifiable property or delete a non-modifiable property, and a ReadOnly exception is returned; a PropertyTypeMismatch is returned if the values do not match the specified types.

If the modification changes a non-dynamic property into a dynamic property, the T-DUA must add the appropriate dynamicPropValue to the dynamicProps attribute of the entry. If the modification changes a dynamic property to a non-dynamic property, the T-DUA must delete both the 'dummy' value and the associated dynamicPropValue in the dynamicProps attribute and add the new static value. The hasDynamicProperties flag may also need to be reset accordingly.

The results of adding the new properties are equivalent to exporting the Service Offer, and the discussion as to the errors which may occur in 7.3.1 applies here as well.

The invocation of the modifyEntry operation causes the T-DUA to return an exception of NotImplemented if the Trader Attribute of supportsModifiableProperties is set to 'false'.

7.3.4 Describe

The T-DUA receives a request to describe a Service Offer. The Describe operation is mapped to an X.500 read operation to read the Service Offer Entry. The data in the Describe operation are:

Parameter	X.500 attribute syntax	Comment
id	DirectoryString	A Service Offer identifier

For the read operation, the entry is identified by the id (the partial DN of a Service Offer Entry) with the entry information set to 'allUserAttributes'. The T-DUA must examine the dynamicProps attribute of the returned entry. For each dynamicPropValue set in the dynamicProps attribute, the associated 'dummy' value must be removed and replaced by the value returned from the evalDP operation invoked on the associated dynamicPropEvalIf interface to retrieve the current value of that property (see 7.9).

The T-DUA filters the attributes returned and passes back to its client the following information:

Parameter	X.500 attribute syntax	Comment
reference	Name	Name of Interface that provides the service
type	Object Identifier	The Service Type Identifier of the offer
properties	Set of attributes	Each attribute type maps to a Property Name and the corresponding attribute value maps to a Property Value

If the client supplies an unknown Service Offer Identifier, a nameError of noSuchObject is returned. The T-DUA returns an UnknownOfferId exception.

Access control is enforced by the DSA based on the access control information stored when the Service Offer was exported and on the Client Identifier of the Describe operation. If the client has insufficient access rights, a Security Error of insufficientAccessRights is returned to the T-DUA which is mapped to the appropriate system error.

7.3.5 Withdraw with constraint

The T-DUA receives a request to delete a set of Service Offers within a trader. This set is identified in the same way as a Query operation (see 7.4.1) identifies a set of offers to be returned to an importer.

The data in the Withdraw with constraint operation are:

Parameter	X.500 Attribute Syntax	Comment
type	Object Identifier	Identifies the Service Type Identifier to be used to select offers to be removed
constr	Filter	The constraint used to select offers to be removed

The first step in processing this operation is to construct an X.500 filter from the constraint. This is complicated by dynamic properties in the service offers. Each property test in the constraint is expanded to succeed if the associated dynamicPropValue is present. For example, the test 'dotsPerInch = 600' is expanded to 'dotsPerInch = 600 OR dotsPerInchDynamicPropValue present'. The filter must include the matching of the 'type' of the service.

Once the X.500 filter has been constructed, the T-DUA invokes an X.500 search for matching Service Offer Entries. The entry information for the X.500 search is selected to sOfferId. The sOfferId is the RDN of a matched offer.

No error is reported by X.500 if a service property in the constraint specified by the client is not recognised by the DSA. Similarly, no error is reported if a service property name is not recognised by the DSA.

An X.500 removeEntry operation is used to remove each of the Service Offers matched by the DSA. The removeEntry operation simply removes the Service Offer entry. The entry is identified by the sOfferId which is returned by the DSA for the matched offers.

Access control is enforced by the DSA based on the access control information stored when the Service Offer was exported and on the Client Identifier of the Withdraw operation. If the client has insufficient access rights, a Security Error of insufficientAccessRights is returned to the T-DUA which is mapped to system specific exception. If the implementation requires the Trader administrator to purge 'old' Offers, then the administrator must also have permission to remove offers.

An exception of NoMatchingOffers is returned if the X.500 search operation yields no matching offers.

7.3.6 Resolve

The T-DUA receives a request to resolve a context relative name of another Trader. In particular, this operation is used when exporting to a Trader that is known by its name (a sequence of Link Names that identifies a path of linked Traders in a Trading Graph), rather than by an interface reference. The client provides the name of a sequence of name components. Each component is used to match a Link name along a Trading Graph. The data in the describe operation are:

Parameter	X.500 attribute syntax	Comment
name	A sequence of Names	A trader name formed by a sequence of Names, where each Name is a Link name

The T-DUA extracts and removes the first component of the sequence (a linkName) and uses this component as a filter for 'linkName' in a one-level search operation on its Link entries. The entry information for the X.500 search is selected to 'targetTraderInterfaceId'.

If another name exists in the sequence, then the T-DUA uses an X.500 read operation to map the 'targetTraderInterfaceId' found to an Address of the target Trader. The T-DUA then connects to the linked target Trader and invokes a Resolve operation with the residual names as its parameter. This process continues until the sequence is empty; at which point, the T-DUA (that performs the last search) returns the final 'targetTraderInterfaceId' to its client, which in turn returns the interface address to its client, and so on. Eventually the final Trader Interface Address corresponding to the final Trader name is returned to the original invoker of this operation.

An UnknownTraderName exception is returned if any of the X.500 search operation is unsuccessful along the path.

If the client does not have sufficient permission to search the Link Entries, then a security error of insufficient AccessRights is returned to the T-DUA which is mapped to a system specific exception.

7.4 Look Up operations

The Look Up operations provide the import functionality for the Trading function. Query is the only operation. It is used to import Service Offer Entries stored in a Trader or a group of federated Traders.

7.4.1 Query operation

The Trader receives a request to search for service offers matching the client's specified service type and constraint within an expected search scope identified by the importer policy. And, optionally, the set of matched offers are further filtered by the client's preference criteria. The Query operation can also include property names of those properties whose values are of interest to the importer. The data in the Query operation are:

Parameter	X.500 attribute syntax	Comment
type	Object Identifier	The type specifies the Service type to be matched
constr	Filter	It represents the constraint used for the offer matching criteria – see below
pref	Not used	Preference criteria – used by T-DUA only
policies	(Optional)	Importer's expectations – see below
desired_props	(Optional)	Service properties of interest – see below
how_many	Not used	Iterator control used by T-DUA only

Because of the complexity of an import, the description is broken into two components: searching the local Trader Service Offer space; and searching the offer spaces of federated Traders. At the choice of the T-DUA, these two search components may be done either sequentially or in parallel.

7.4.2 Policies

The first step in processing a Query operation is to process the Policies. Policies are represented as <name, value> pairs. An importer can specify zero or more importer policies which specify the importer's expectations for the search. Each Trader has its own search policies that guide its behaviour in searching. The Trader policies interact with the importer policies. If an importer policy is not specified, then the Trader uses the Trader's default policy. If an importer policy exceeds the maximum (or limiting) policy values set by the Trader, then the Trader overrides the importer's expectations with the Trader's maximum/limiting policy value.

It is the T-DUA's responsibility to check and set the actual upper bounds for the cardinality policies and hop_count policy. It is also the T-DUA's responsibility to place a policy name in the return parameter of 'limits_applied', if a search is prematurely terminated by an upper bound.

A Trader has three standardised capability supported policies. If a Trader does not support a capability, then an importer can not override it; the non-use of an importer's request of a capability is reported by including the capability policy name in 'limits-applied'. However, if a Trader supports a capability and an importer does not wish to use that capability, then the Trader must respect the importer's wish. The Trader's capability applies if an importer does not specify an expected capability. There are three standardised capability choices for an importer:

- use_proxy_offers – If set to 'false', then the Proxy Offer Entries are not searched for matches.
- use_modifiable_properties – If set to 'false', then the T-DUA 'And' s the term 'hasModifiableProperties = false' in the matching filter.
- use_dynamic_properties – If set to 'false', then the T-DUA 'And' s the term 'hasDynamicProperties = false' in the matching filter.

The next step is to process the Link Behaviour Policy to determine if any interworking can take place. This is described in 7.4.4.

If an importer states a starting_trader policy, then the first component in the TraderName is searched against the name held in each Link Entry. If no match is found, then the exception InvalidPolicyValue is returned. Otherwise, the T-DUA invokes the Query operation on the named Link, and passes the 'starting_trader' policy with the first component removed (See also 7.3.6 for Resolve operation).

7.4.3 Searching locally

Searching locally commences with the construction of an X.500 filter to match the Service Type Identifier. In addition to Service Offers which exactly match the Service Type Identifier, the filter can also match any entry with a Service Type Identifier which is a refinement (subclass) of the specified type (Service Type Identifier). However, only exact service type matching is done if the importer has set the importer policy of exact_type_match to 'true' in the Query operation. If the Trader supports dynamic properties and the importer wishes to use offers with dynamic properties, then the 'hasDynamicProperties' attribute is selected for the EntryInformationSelection for the X.500 search operation. A one-level search on the Service Offer Entry is performed to see if there are any Service Offers with the required service type.

A second filter is constructed from the constraint specified for matching. This can be complicated by the support and presence of dynamic properties in the Service Offers (knowledge obtained from the first search). If the Trader supports dynamic properties and the importer wishes to use offers with dynamic properties, then each property test in the constraint is expanded to succeed if the associated 'dynamicPropValue' is present. For example, the test 'dotsPerInch = 600' is expanded to 'dotsPerInch = 600 OR dotsPerInchDynamicPropValue present'.

The second filter is 'And' ed with the first filter. The T-DUA invokes an X.500 search for matching Service Offer Entries. All userAttributes are selected in the X.500 EntryInformationSelection; it is the T-DUA's responsibility to filter the attributes returned. A user may wish to have the description of values for 'none', 'some' or 'all' of the service properties.

No error is reported by X.500 if a service property in the constraint specified by the client is not recognised by the DSA. Similarly, no error is reported if a Service Property Name is not recognised by the DSA. However, an Attribute Error of noSuchAttributeOrValue is returned if no attributes are returned.

If a DSA matches many Service Offers against the filter, the DSA need only return some of the matches. In this case the partialOutcomeQualifier of sizelimitExceeded is set. The T-DUA may need to restart the search in this case using the paged results option of the X.500 search operation.

The entries returned from the X.500 searches may contain dynamic properties. The T-DUA must examine each entry for 'dynamicProps' attribute. If it is found, then the associated dummy attribute for every dynamicPropValue in the dynamicProps attribute is removed from the search result and the associated interface for evaluation of the property is invoked to retrieve the current value of that dynamic property (see 7.9 for evaluation of dynamic properties). The Query constraint must then be evaluated by the T-DUA to determine if the Service Offers actually do satisfy the constraint.

7.4.4 Searching Federated Traders

Hop_count is used by a T-DUA to control the depth of Links traversed for an import. A hop_count of zero terminates the propagation of an import through linked traders. One is subtracted from a hop_count whenever a Link is traversed to pass on a Query operation.

Associated with each Link are two Link follow policies of def_pass_on_follow_rule and limiting_follow_rule that are set at Link creation time. The Trader itself has three link related policies:

- maxLinkFollowPolicy – Limits the value of limiting_follow_rule allowed while a link is first established.
- maxFollowPolicy – Limits the link follow behaviour on any link in the trader for a given Query operation.
- defFollowPolicy – Provides the default link follow rule if an importer fails to specify one on its Query.

The follow policy for a particular Link is, therefore, either:

- min (trader.max_follow_policy, link.limiting_follow_rule, query.link_follow_rule); or
- min (trader.max_follow_policy, link.limiting_follow_rule, trader.def_follow_policy).

If any suitable Links are to be followed, the T-DUA uses X.500 read operations to map the targetTraderInterfaceId names (from the Trader Link entries) to the Address of each target Trader. The T-DUA then connects to the selected Traders and passes the Query operation to them with the link_follow_rule specified as either:

- min (trader.max_follow_policy, link.limiting_follow_rule, query.link_follow_rule); or
- min (trader.max_follow_policy, link.def_pass_on_follow_rule).

In addition, if the source trader that initiates a federated Query operation, wishes to include an identifier for the Query operation in the policy parameter, it can do so. A Trader is not obliged to generate such an operation identifier, but a Trader is obliged to pass this policy down a Link.

7.4.5 Searching Proxy Offers

If the importer's use_proxy_offer policy is not set to 'false' and the trader's supportsProxyOffer attribute is set to 'true', then the X.500 searches of 7.4.3 are repeated on the Proxy Offer Entries. For each matched Proxy Offer, the T-DUA applies a modified Query operation using information stored in the matched Proxy Offer. The T-DUA then uses an X.500 read operation to map the proxyLookUpInterfaceId name (from the Proxy Offer entry) to an 'address' which the T-DUA uses to invoke the modified Query operation.

7.4.6 Service Offer returned

Once the Trader finds a set of Service Offers, the preference parameter and the various cardinality policies are used to order and limit the offers returned to the Importer. This is a T-DUA function and is not considered further in this Specification. The implementation of an offer iterator is also the responsibility of the T-DUA.

The exception of UnknownType (service type name not known) or NoMatchingType (no offers with the required service type) is returned if either of these is true from the total scoped search space.

7.5 Link operations

7.5.1 Add Link

The Trader receives an Add Link operation. This operation is mapped to an X.500 addEntry operation to add a new Trader Link entry to the X.500 Directory. The data in the Add Link operation are:

Parameter	X.500 attribute syntax	Comment
name	DirectoryString	Name of the new Link
target	Name	Name of another Trader
def_pass_on_follow_rule	FollowOption	Must not exceed limiting_follow_rule
limiting_follow_rule	FollowOption	Must not exceed max_link_follow_policy of the Trader

The T-DUA first checks the permissible values for the parameters. The exception DefaultFollowTooPermissive is raised if the def_pass_on_follow_rule exceeds the limiting_follow_rule. The LimitingFollowTooPermissive exception is raised if the limiting_follow_rule exceeds the Trader's max_link_follow_policy. The T-DUA must first read the Trader Entry to determine its max_link_follow_policy. The T-DUA also ensures that the new Link Name is unique within the set of Trader Links by searching for an existing instance of the name. If a match is found, then the DuplicateLinkname exception is raised.

The T-DUA maps the Add Link operation to an X.500 addEntry operation. The identifier of the new entry is selected by T-DUA such that it is unique amongst all Trader Links. Any scheme may be used; a simple method would be an incremental counter. The structural object class of the new entry is traderLink. The traderLink auxiliary object class can also specify additional Link Properties that may be included in the addEntry operation.

Access control information must be included in the new entry to prevent other X.500 or Trader users from accessing or modifying the Link information.

If the Add Link is successful, no exception is raised.

7.5.2 Remove Link

The T-DUA receives a request to delete a Trader Link. The data in the Remove Link operation is:

Parameter	X.500 attribute syntax	Comment
name	DirectoryString	Identifies the Link to be removed by its linkName

The T-DUA must first use the X.500 search operation on the Link Entries for an entry with that linkName. If the search fails to find a Link Entry, then the exception UnknownLinkname is raised.

The T-DUA then uses an X.500 removeEntry operation to remove the Trader Link Entry by using the linkId of the matched entry. This operation simply removes the Trader Link Entry.

Access control is enforced by the DSA based on the access control information stored when the Trader Link was created and on the Client Identifier of the remove operation. If the Client has insufficient access rights, a Security Error of insufficientAccessRights is returned to the T-DUA which is mapped to a system specific exception.

7.5.3 Modify Link

The Trader receives a request to modify a Trader Link. The Modify operation is mapped to an X.500 search to find the Link using the linkName, and then to a modifyEntry operation to change the Trader Link entry. The data in the Modify Link operation is:

Parameter	X.500 attribute syntax	Comment
name	DirectoryString	Identifies the Link to be modified
def_pass_on_follow_rule	FollowOption	Must not exceed limiting_follow_rule
limiting_follow_rule	FollowOption	Must not exceed limiting_follow_policy of the Trader

The T-DUA first checks the permissible values for the parameters. The exception DefaultFollowTooPermissive is raised if the def_pass_on_follow_rule exceeds the limiting_follow_rule. The LimitingFollowTooPermissive exception is raised if the limiting_follow_rule exceeds the Trader's max_link_follow_policy. The T-DUA then searches the Trader Link entry identified by the linkName. AllUserAttributes are selected on the X.500 entry information selection. If this entry does not exist, the DSA returns a nameError of noSuchObject, which is mapped to the UnknownLinkName exception.

An X.500 modifyEntry operation is then constructed which first deletes all the existing Link follow behaviour (and properties, if any) and then adds the new follow behaviour (and properties, if any) from the Modify Link operation. If the Client does not have sufficient permission to delete attributes of the entry, a security error of insufficient AccessRights is returned to the T-DUA.

The trader Link Auxiliary Object class can also specify Link properties that may be included in the modifyEntry operation. The newly modified entry may violate the schema. In this case, when the newly modified Link Entry is added, the DSA returns an Update Error of object class violation. The existing Trader Link remains unchanged. The X.500 schema checking cannot check that the values are those specified by the Link schema. An Attribute Error of undefinedAttributeType occurs if the DSA does not know one of the attributes supplied by the client.

7.5.4 Describe Link

The Trader receives a request to return details of a particular Link. This operation is mapped into an X.500 search operation. The data are:

Parameter	X.500 attribute syntax	Comment
Name	DirectoryString	Identifies the Link to be retrieved

The T-DUA searches the Trader Link entry identified by the linkName with the entry information selection set for 'allUserAttributes'. If this entry does not exist, the DSA will return a nameError of noSuchObject, which is mapped to an UnknownLinkName exception. If the client does not have sufficient permission to search the entry, a security error of insufficientAccessRights is returned to the T-DUA.

Once the data have been returned from the X.500 DSA, the T-DUA returns the data, minus the linkId, to the client.

7.5.5 List Links

The Trader receives a request to return the names of all Trader Links known to the Trader. This operation is mapped into an X.500 search operation. The data are:

Parameter	X.500 attribute syntax	Comment
(None)		

The T-DUA performs an X.500 search operation to return the linkName attribute of all Trader Link entries. The search is a one level search at the Link entries. The Link names (if any) are returned to the User:

Parameter	X.500 attribute syntax	Comment
LinkNameSeq	Set of DirectoryString	

where each DirectoryString (a Link name) is unique for each Link entry.

7.6 Proxy Offer operations

The Proxy Offer operations form a set of operations that allow a client to export, withdraw, and describe Proxy Offers in Trader.

7.6.1 Export Proxy

The Trader receives an Export Proxy operation. This operation is mapped to an X.500 addEntry operation to add a new Proxy Offer entry to the X.500 Directory. The data in the Export Proxy operation are:

Trader parameter	X.500 attribute syntax	Comment
type	Object Identifier	The Type name is assumed to be the Service Type Identifier. It is used to obtain the Auxiliary object class for the new entry.
target	Name	The name of the entry in the X.500 database which contains the address of the proxyLookUpInterfaceId.
properties	Any (optional)	The actual X.500 attributes supplied by the exporter for the Service Properties are controlled by the Service Properties Type Identifier.
if_match_all	Boolean	
recipe	String	
policies_to_pass_on	Set of Attributes (optional)	

The T-DUA maps the Export Proxy operation to an X.500 addEntry. The name of the new entry is selected by T-DUA such that it is unique amongst all Proxy Offers. Any scheme may be used; a simple method would be an incremental counter. The structural object class of the new entry is proxyOffer. The auxiliary object class is the Service Properties Type Identifier (OID) associated with the Service Type Identifier parameter of the Proxy Offer. The T-DUA uses its Type Repository to obtain the Interface Type Identifier and the Service Properties Type Identifier associated with this Service Type.

Access control information must be included in the new entry to prevent other X.500 users from accessing or modifying the trader information. Additional access control information is added to allow the X.500 DSA to control access to trader information by the clients. This can be extended to only allowing certain clients the privilege of exporting Proxy Offers.

If the Service Type Identifier or auxiliary object class is not known to the Trader, the DSA returns an Update error of type objectClassViolation. The T-DUA returns an exception of UnknownServiceType. The Service Type Identifier is stored in the Proxy Offer Entry.

The Service Property Values which may/must be present in the new entry are controlled by the auxiliary object class (Service Properties Type Identifier) of the entry. The processing of these property values are the same as for the operation Export service offer (see 7.3.1).

The if_match_all parameter specifies whether this offer requires matching of the service type only.

The target parameter gives the name of the interface that is used to obtain the Service interface at runtime. If 'nil' is specified, then the exception InvalidInterfaceRef is raised.

The recipe parameter provides the modification that can be applied to the constraint of a Query operation that matches this Proxy Offer in service type, and property values when if_match_all is not 'true'.

The policies_to_pass_on consists of <name, value> pairs to be appended to the policy parameter of the incoming Query operation to form the modified policy parameter of the modified Query operation.

If the export of the Proxy Offer is successful, the T-DUA returns success with the following information:

Parameter	X.500 attribute syntax	Comment
OfferId	DirectoryString	A Proxy Offer identifier

where the OfferId is the string representation of the partial DN of the new Proxy Offer entry.

7.6.2 Withdraw Proxy

The T-DUA receives a request to delete a Proxy Offer. The Withdraw Proxy operation is mapped to an X.500 removeEntry operation to remove the Proxy Offer Entry. The data in the Withdraw Proxy operation are:

Parameter	X.500 attribute syntax	Comment
id	DirectoryString	A Proxy Offer identifier

This operation simply removes the Proxy Offer Entry. The entry is identified by the 'id' which uniquely identifies a Proxy Offer Entry.

If the client supplies an unknown Proxy Offer Identifier, the removeEntry operation returns a nameError of noSuchObject. The T-DUA then uses the Proxy Offer Identifier to read a Service Offer Entry. If the read is successful, then the exception of NotProxyOfferId is raised. If the read is unsuccessful, then the exception of UnknownOfferId is raised.

Access control is enforced by the DSA based on the access control information stored when the Proxy Offer was exported and on the Client Identifier of the Withdraw Proxy operation. If the Client has insufficient access rights, a Security Error of insufficientAccessRights is returned to the T-DUA which is mapped to system specific exception. If the implementation requires the Trader administrator to purge 'old' Proxy Offers, then the administrator must also have permission to remove Proxy Offers.

7.6.3 Describe Proxy

The T-DUA receives a request to describe a Proxy Offer. The describe operation is mapped to an X.500 read operation to read the Proxy Offer Entry. The data in the Describe operation are:

Parameter	X.500 attribute syntax	Comment
id	DirectoryString	A Proxy Offer Identifier

The entry is identified by the 'id' which uniquely identifies the entry. The entry information for the read operation is set to 'allUserAttributes'. The T-DUA is not required to evaluate the dynamic property values in the Proxy Offer.

The T-DUA filters the attributes returned and passes back to its client the following information:

Parameter	X.500 attribute syntax	Comment
type	OID	Service Type Identifier
target	DN	Name of Interface entry that provides the service interface reference
serviceProperties	Set of attributes	An attribute type maps to Property Name and an attribute value maps to a Property Value
if_no_match	boolean	
recipe	string	
policies_to_pass_on	Set of attributes (if any)	

If the Client supplies an unknown Proxy Offer Identifier, the read operation returns a nameError of noSuchObject. The T-DUA then uses the Proxy Offer Identifier to try to read a Service Offer Entry. If the read is successful, then the exception of NotProxyOfferId is raised. If the read is unsuccessful, then the exception of UnknownOfferId is raised.

Access control is enforced by the DSA based on the access control information stored when the Proxy Offer was exported and on the Client Identifier of the Describe Proxy operation. If the client has insufficient access rights, a Security Error of insufficientAccessRights is returned to the T-DUA which is mapped to the appropriate System error.

7.7 Trader Attribute Operations

Trader properties and policies are expressed as X.500 attributes in the Trader Entry. All registered clients of a Trader can 'get' information on the Trader characteristics by the use of the X.500 read operation on the Trader Entry for the characteristics attributes. All registered Exporters can, in addition, have the access rights to read the attributes of: supportsModifiableProperties, supportsDynnamicProperties, supportsProxyOffers, and typeReposIf. All registered Importers can, in addition, have the access rights to read the following Trader attributes: defSearchCard, maxSearchCard, defMatchCard, maxMatchCard, defReturnCard, maxReturnCard, maxList, defHopCount, maxHopCount, supportsModifiableProperties, supportsDynnamicProperties, supportsProxyOffers, defFollowPolicy, maxFollowPolicy, and typeReposIf.

The administrator of the Trader can 'get' and 'set' all Trader attributes, where the 'get' operation is mapped onto X.500 read operation and the 'set' operation is mapped onto the X.500 modifyEntry operation.

7.8 Administrative operations

7.8.1 List Offers

The Trader receives a request to return the identifiers of all Service Offers known to the Trader. This operation is mapped into an X.500 search operation. The data are:

Parameter	X.500 attribute syntax	Comment
how_many	Not used	This parameter is kept by the T-DUA in its implementation of the Id iterator

The T-DUA performs an X.500 search operation to return the identifiers of all Service Offer Entries. The search is a one level search for any Service Offer entries. No information is requested. These identifiers (if any) are returned to the User as:

Parameter	X.500 attribute syntax	Comment
ids	Set of DirectoryString	

where each id (string representation of a partial DN) uniquely identifies a Service Offer Entry.

The T-DUA is responsible for implementing the Id Iterator.

Access control is enforced by the DSA based on the access control information stored when the Service Offer was exported and only administrators can perform the List Offers operation. If a client has insufficient access rights, a Security Error of insufficientAccessRights is returned to the T-DUA which is mapped to the appropriate system error.

7.8.2 List Proxies

The Trader receives a request to return the identifiers of all Proxy Offers known to the Trader. This operation is mapped into an X.500 search operation. The data are:

Parameter	X.500 attribute syntax	Comment
how_many	Not used	This parameter is kept by the T-DUA in its implementation of the Id iterator

The T-DUA performs an X.500 search operation to return the identifiers of all Proxy Offer Entries. The search is a one level search for any Proxy Offer Entries. No information is requested. These identifiers (if any) are returned to the User as:

Parameter	X.500 attribute syntax	Comment
ids	Set of DirectoryString	

where each id (string representation of a partial DN) uniquely identifies a Proxy Offer Entry.

The T-DUA is responsible for implementing the Id Iterator.

Access control is enforced by the DSA based on the access control information stored when the Proxy Offer was exported and only administrators can perform this List Proxies operation. If a client has insufficient access rights, a Security Error of insufficientAccessRights is returned to the T-DUA which is mapped to the appropriate system error.

7.9 Dynamic Property Evaluation operations

The T-DUA acts as a client for the Dynamic Property Evaluation operations to obtain values for dynamic properties. EvalDP is the only operation specified.

7.9.1 EvalDP

To obtain the value of a dynamic property at runtime, the signature for the operation 'evalDP' is specified. The data in the operation is:

Parameter	X.500 attribute syntax	Comment
name	Object Identifier	Identifies the Property name
returned_type	Object identifier	Identifies the Property Type
extra_info	Any (optional)	May include information required to obtain the property value

The operation returns 'any' value which should contain a value for the property with a type of 'returned_type'. The DPEvalFailure exception is raised if the value for the property cannot be determined.

How the value of a dynamic property is actually obtained is not specified in ITU-T Rec. X.950 | ISO/IEC 13235-1.

8 Type Repository

The concept of the Type Repository is central to the Trader. A minimal Type Repository is defined in this clause. This minimal type repository covers only those parts of the Type Repository that are necessary to enable the correct functioning of the X.500 Directory.

NOTE 1 – More complicated Type Repositories are feasible, but work in this area rapidly becomes research into Type Repositories rather than specifying a Trader using X.500.

NOTE 2 – The ODP Type Repository Function work is in progress at the time of publication of this Specification.

8.1 X.500 schema and the Minimal Type Repository

The Minimal Type Repository consists of the normal X.500 schema. The functions provided by this component (and the limitations) are described below.

Each Service Type Identifier is an OID (Object Identifier) and identifies a Service Type. Associated with each Service Type is an Interface Type (identified by an OID) and a Service Properties Type (also identified by an OID). The Interface Type identifies the operations (and their associated parameters) for the service and the Service Properties Type identifies the service properties associated with the service.

When the T-DUA receives a Service Type Identifier from its exporter, it sends the OID to its Type Repository and receives the associated Interface Type Identifier (OID) and the Service Properties Type Identifier (OID) from the Type Repository. The T-DUA stores the Service Type identifier in the Service Offer Entry and uses the Service Properties Type Identifier as the auxiliary object class for that Service Offer Entry.

The auxiliary object class defines the property attributes which can be included in a Service Offer. It does this by two lists:

- The 'must contain' list which contains the property attributes which must be present in the Service Offer.
- The 'may contain' list which contains the property attributes which may be present in the Service Offer.

Property attributes which are not present in either the 'must contain' or 'may contain' lists cannot be included in the service properties of the Service Offer. The appropriate use of access control rules for a property determines whether a particular property is read-only and cannot be modified or deleted. The treatment of dynamic properties is discussed in clause 9.

The property attributes are also identified by OIDs. Attribute definitions can control:

- Whether multiple values can be stored in the attribute.
- The syntax (e.g. string, integer) of the values. Complex attribute syntaxes (e.g. a value consists of a string followed by an integer) can also be defined. Some syntaxes (e.g. telephoneNumberSyntax) have restrictions on the characters which can appear in the value.
- How values are compared, i.e. when can two values be 'equal' – this might not be when they are identical.

A major limitation of attribute definitions (with respect to properties) is that it is not possible to control what values are to be accepted for that attribute.

Attribute definitions are independent of object class definitions. Any attribute can be used with any object class definition. Subtyping of property values are permitted.

The methods used to allocate OIDs assist in ensuring that a given OID does not identify two different types (Service Types, Interface Types, Service Properties Types) or properties (property attributes), even between multiple Traders owned by different organisations. There should never be a conflict because two different Traders allocated the same OID to two different types or properties.

The types and properties definitions are known by both the DSA storing the Trader database and the T-DUA. In a DSA, the X.500 schema is stored as part of the X.500 database, and, in theory, can be dynamically managed.

9 Dynamic properties

The use of the X.500 schema management to implement part of the Type Repository has implications for storage of properties within X.500. The X.500 schema management rejects an attempt to create an entry which is missing mandatory attributes (properties), even though these may be dynamic properties.

The solution adopted is for the T-DUA to add instances of any dynamic property to the Service Offer before storing it in the X.500 Directory. An instance of a dynamic property in an Export is 'marked' by the use of the dynamicProp Structure instead of a normal value for the 'any' value of a property. A Trader that supports dynamic property recognises this structure and accepts it as a property with a dynamic value. The T-DUA stores a dynamic property with a 'dummy' value and creates, at the same time, an associated 'dynamicPropValue' in the dynamicProps attribute which contains the necessary information for the T-DUA to obtain a correctly-typed property value when required. The 'dummy' value of the corresponding property attribute is ignored during matching of a service offer. A separate 'dynamicPropValue' is defined for every possible dynamic property to be stored in the Trader.

For example, suppose that an Exporter exports a Service Offer with the property 'queueLength' marked as 'dynamic'. When creating the Service Offer Entry in the X.500 database, the T-DUA creates an instance of the 'queueLength' and the 'queueLengthDynamicPropValue' dynamicPropValue in that entry. When reading that Service Offer Entry, the presence of the 'queueLengthDynamicPropValue' in the dynamicProps attribute indicates that the value of the corresponding attribute (property) 'queueLength' found in the X.500 database is to be ignored and the real value must be obtained from the interface stored in the dynamicPropValue.

An alternative approach would be to include the 'dynamic' switch as part of each attribute definition (i.e. any attribute either has a real value or the 'dynamic' choice set). This approach was rejected as it would mean that all attribute definitions and processing would be complicated.

9.1 Exporting a Service Offer

When exporting a Service Offer, the client specifies the OID of the Service Type and a set of properties (attributes). If the Trader supports dynamic properties and there are dynamic properties indicated in the Service Offer, then the T-DUA constructs 'dummy' values and the dynamicProps attribute with a dynamicPropValue for each dynamic property in the Service Offer. The T-DUA also sets the hasDynamicProperties attribute to 'true'. If the Trader does not support dynamic properties, then any Service Offer with dynamic properties is rejected. The X.500 Directory also rejects the export if the Service Offer does not match the schema.

If the Client wishes to export an offer of a new service type, the export becomes a two stage process. First, any new 'Type' definitions and any new 'Property' definitions are added to the X.500 schema held by the Type Repository. Once the new Types and Attributes (Properties) have been registered with the DSA, the Client can export the new Service Offer.

9.2 Importing a Service Offer

When importing a Service Offer, the client specifies the Service Type (OID) it wishes to find and a 'matching constraint'.

A limited form of subclassing is available in X.500. An object class can be defined to be a 'subclass' of one or more existing object class. In this case, specifying the Service Properties Type (OID) of a superclass will also select Service Offers of the subclass. The limitation of the super/subclass concept is with the 'must contain' and 'may contain' lists of the object class definition. A subclass inherits the 'must contain' and 'may contain' lists of its superclasses and can only add additional attributes.

The T-DUA searches the X.500 database for matching Service Offers and Proxy Offers (if required) for the required service type first, followed by the matching offers with the required constraint and type.

If the Trader supports dynamic properties and the importer requests the use of dynamic properties, then the value of the hasDynamicProperties attribute is requested also on the first search. If this attribute is 'true', then the matching constraint filter is rewritten to succeed. For each entry returned, the T-DUA goes through the list of attributes returned. If any 'dynamicPropValue' is present, the T-DUA requests the 'real' value of the associated property from the interface contained in the dynamicPropValue. Once the dynamic properties have been returned, the T-DUA must complete the evaluation of the matching constraint.

Annex A

Trader definitions schema definition

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

```
TraderDefinitions {joint-iso-itu-t 2}
DEFINITIONS ::=
BEGIN
IMPORTS
        informationFramework, selectedAttributeTypes, authenticationFramework
            FROM UsefulDefinitions {joint-iso-ccitt ds(5) modules(1)
                                                usefulDefinitions(0) 2}

        CONTENT-RULE, NAME-FORM, STRUCTURE-RULE, OBJECT-CLASS, MATCHING-RULE,
        ATTRIBUTE, top, ObjectClassKind, objectIdentifierMatch,
        DistinguishedName
            FROM InformationFramework informationFramework

        DirectoryString {}, commonName, description, presentationAddress,
        distinguishedName, caseIgnoreMatch, caseIgnoreSubstringsMatch,
        caseExactMatch, booleanMatch,integerMatch, octetStringMatch
            FROM SelectedAttributeTypes selectedAttributeTypes

        userPassword
            FROM AuthenticationFramework authenticationFramework ;


-- Trader Entry
traderEntry OBJECT-CLASS ::= {
        SUBCLASS OF             { top }
        MUST CONTAIN            {commonName | traderInterface | dsaName |
                                typeRepos | defSearchCard | maxSearchCard |
                                defMatchCard | maxMatchCard | defReturnCard |
                                maxReturnCard | defHopCount | maxHopCount |
                                defFollowPolicy | maxFollowPolicy |
                                maxLinkFollowPolicy | supportsModifiableProperties |
                                supportsDynamicProperties | supportsProxyOffers |
                                maxList | requestIdStem}
        MAY CONTAIN             {description | userPassword}
        ID                      id-trader-oc-traderEntry}

traderInterface ATTRIBUTE ::= {
        SUBTYPE OF              presentationAddress
        SINGLE VALUE            TRUE
        ID                      id-trader-at-traderInterface}
```

```
dsaName ATTRIBUTE ::= {
        SUBTYPE OF                distinguishedName
        SINGLE VALUE              TRUE
        ID                        id-trader-at-dsaName}

typeRepos ATTRIBUTE ::= {
        SUBTYPE OF                distinguishedName
        SINGLE VALUE              TRUE
        ID                        id-trader-at-typeRepos}

defSearchCard ATTRIBUTE ::= {
        WITH SYNTAX               INTEGER
        EQUALITY MATCHING RULE    integerMatch
        SINGLE VALUE              TRUE
        ID                        id-trader-at-defSearchCard }

maxSearchCard ATTRIBUTE ::= {
        WITH SYNTAX               INTEGER
        EQUALITY MATCHING RULE    integerMatch
        SINGLE VALUE              TRUE
        ID                        id-trader-at-maxSearchCard}

defMatchCard ATTRIBUTE ::= {
        WITH SYNTAX               INTEGER
        EQUALITY MATCHING RULE    integerMatch
        SINGLE VALUE              TRUE
        ID                        id-trader-at-defMatchCard}

maxMatchCard ATTRIBUTE ::= {
        WITH SYNTAX               INTEGER
        EQUALITY MATCHING RULE    integerMatch
        SINGLE VALUE              TRUE
        ID                        id-trader-at-maxMatchCard}

defReturnCard ATTRIBUTE ::= {
        WITH SYNTAX               INTEGER
        EQUALITY MATCHING RULE    integerMatch
        SINGLE VALUE              TRUE
        ID                        id-trader-at-defReturnCard}

maxReturnCard ATTRIBUTE ::= {
        WITH SYNTAX               INTEGER
        EQUALITY MATCHING RULE    integerMatch
        SINGLE VALUE              TRUE
        ID                        id-trader-at-maxReturnCard}
```

```
defHopCount ATTRIBUTE ::= {
        WITH SYNTAX               INTEGER
        EQUALITY MATCHING RULE    integerMatch
        SINGLE VALUE              TRUE
        ID                        id-trader-at-defHopCount}
maxHopCount ATTRIBUTE ::= {
        WITH SYNTAX               INTEGER
        EQUALITY MATCHING RULE    integerMatch
        SINGLE VALUE              TRUE
        ID                        id-trader-at-maxHopCount}

defFollowPolicy ATTRIBUTE ::= {
        WITH SYNTAX               FollowOption
        EQUALITY MATCHING RULE    integerMatch
        SINGLE VALUE              TRUE
        ID                        id-trader-at-defFollowPolicy }

FollowOption ::= ENUMERATED{
        localOnly   (0),
        ifNoLocal   (1),
        always      (2)}

maxFollowPolicy ATTRIBUTE ::= {
        WITH SYNTAX               FollowOption
        EQUALITY MATCHING RULE    integerMatch
        SINGLE VALUE              TRUE
        ID                        id-trader-at-maxFollowPolicy }

maxLinkFollowPolicy ATTRIBUTE ::= {
        WITH SYNTAX               FollowOption
        EQUALITY MATCHING RULE    integerMatch
        SINGLE VALUE              TRUE
        ID                        id-trader-at-maxLinkFollowPolicy }

supportsModifiableProperties ATTRIBUTE ::= {
        WITH SYNTAX               BOOLEAN
        EQUALITY MATCHING RULE    booleanMatch
        SINGLE VALUE              TRUE
        ID                        id-trader-at-supportsModifiableProperties}
supportsDynamicProperties ATTRIBUTE ::= {
        WITH SYNTAX               BOOLEAN
        EQUALITY MATCHING RULE    booleanMatch
        SINGLE VALUE              TRUE
        ID                        id-trader-at-supportsDynamicProperties}
```

```
supportsProxyOffers ATTRIBUTE ::= {
        WITH SYNTAX               BOOLEAN
        EQUALITY MATCHING RULE    booleanMatch
        SINGLE VALUE              TRUE
        ID                        id-trader-at-supportsProxyOffers}

maxList ATTRIBUTE ::= {
        WITH SYNTAX               INTEGER
        EQUALITY MATCHING RULE    integerMatch
        SINGLE VALUE              TRUE
        ID                        id-trader-at-maxList }

requestIdStem ATTRIBUTE ::= {
        WITH SYNTAX               OCTET STRING (SIZE (0..ub-request-id-stem))
        EQUALITY MATCHING RULE    octetStringMatch
        SINGLE VALUE              TRUE
        ID                        id-trader-at-requestIdStem }

--Trader Policy Entry

traderPolicyEntryNF NAME-FORM ::= {
        NAMES                     traderPolicyEntry
        WITH ATTRIBUTES           {commonName}
        ID                        id-trader-nf-traderPolicy}

traderPolicyEntry OBJECT-CLASS ::= {
        SUBCLASS OF               {top}
        MUST CONTAIN              {commonName }
        MAY CONTAIN               {typeManagementConstraint | searchConstraint |
                                  offerAcceptanceConstraint }
        ID                        id-trader-oc-traderPolicy}
PolicySpecification ::= CHOICE {
        stringRule                [0]    DirectoryString{ub-policy-string-rule},
        policyObjectId            [1]    DistinguishedName }

policySpecificationMatch MATCHING-RULE ::= {
        SYNTAX                    PolicySpecification
        ID                        id-trader-mr-policySpecificationMatch}

-- The rule returns TRUE if two specifications contain exactly
-- the same characters.

typeManagementConstraint ATTRIBUTE ::= {
        WITH SYNTAX               PolicySpecification
        EQUALITY MATCHING RULE    policySpecificationMatch
        SINGLE VALUE              TRUE
        ID                        id-trader-at-typeManagementConstraint}
```

```
searchConstraint ATTRIBUTE ::= {
        WITH SYNTAX               PolicySpecification
        EQUALITY MATCHING RULE    policySpecificationMatch
        SINGLE VALUE              TRUE
        ID                        id-trader-at-searchConstraint}

offerAcceptanceConstraint ATTRIBUTE ::= {
        WITH SYNTAX               PolicySpecification
        EQUALITY MATCHING RULE    policySpecificationMatch
        SINGLE VALUE              TRUE
        ID                        id-trader-at-offerAcceptanceConstraint}

-- Service Offer Entry

serviceOfferEntryNF NAME-FORM ::= {
        NAMES                     serviceOfferEntry
        WITH ATTRIBUTES           {sOfferId}
        ID                        id-trader-nf-serviceOffer}
serviceOfferEntry OBJECT-CLASS ::= {
        SUBCLASS OF               {top}
        MUST CONTAIN              {sOfferId | serviceInterfaceId |serviceTypeId
                                  hasDynamicProperties | hasModifiableProperties }
        MAY CONTAIN               {dynamicProps}
        ID                        id-trader-oc-serviceOffer}

sOfferId ATTRIBUTE ::= {
        WITH SYNTAX               DirectoryString{ub-s-offer-id}
        EQUALITY MATCHING RULE    caseIgnoreMatch
        SUBSTRINGS MATCHING RULE  caseIgnoreSubstringsMatch
        SINGLE VALUE              TRUE
        ID                        id-trader-at-sOfferId}

serviceInterfaceId ATTRIBUTE ::= {
        SUBTYPE OF                distinguishedName
        SINGLE VALUE              TRUE
        ID                        id-trader-at-serviceInterfaceId}
serviceTypeId ATTRIBUTE ::= {
        WITH SYNTAX               OBJECT IDENTIFIER
        EQUALITY MATCHING RULE    objectIdentifierMatch
        ID                        id-trader-at-serviceTypeId }

hasDynamicProperties ATTRIBUTE ::= {
        WITH SYNTAX               BOOLEAN
        EQUALITY MATCHING RULE    booleanMatch
        SINGLE VALUE              TRUE
        ID                        id-trader-at-hasDynamicProperties}
```

```
hasModifiableProperties ATTRIBUTE ::= {
        WITH SYNTAX               BOOLEAN
        EQUALITY MATCHING RULE    booleanMatch
        SINGLE VALUE              TRUE
        ID                        id-trader-at-hasModifiableProperties}
dynamicProps ATTRIBUTE ::= {
        WITH SYNTAX               SEQUENCE OF DynamicPropValue
        ID                        id-trader-at-dynamicProps}

DynamicPropValue ::= SEQUENCE {
        propertyType              OBJECT IDENTIFIER,
        dynamicPropEvalIf         DistinguishedName,
        extraInfo                 DirectoryString{ub-dynamic-value-extra-info}}

dynamicPropValueMatch MATCHING-RULE ::= {
        SYNTAX                    DynamicPropValue
        ID                        Id-trader-mr-dynamicPropValueMatch}
-- The rule returns TRUE if two values contain exactly the same characters.

-- Trader Link Entry

traderLinkEntryNF NAME-FORM ::= {
        NAMES                     traderLinkEntry
        WITH ATTRIBUTES           {linkId}
        ID                        id-trader-nf-traderLink}

traderLinkEntry OBJECT-CLASS ::= {
        SUBCLASS OF               {top}
        MUST CONTAIN              {linkName | linkId | targetTraderInterfaceId |
                                  defPassOnFollowRule | limitingFollowRule}
        ID                        id-trader-oc-traderLink}
linkName ATTRIBUTE ::= {
        WITH SYNTAX               DirectoryString {ub-link-name}
        EQUALITY MATCHING RULE    caseIgnoreMatch
        SUBSTRINGS MATCHING RULE  caseIgnoreSubstringsMatch
        SINGLE VALUE              TRUE
        ID                        id-trader-at-linkName}

linkId ATTRIBUTE ::= {
        WITH SYNTAX               DirectoryString {ub-link-id}
        EQUALITY MATCHING RULE    caseIgnoreMatch
        SUBSTRINGS MATCHING RULE  caseIgnoreSubstringsMatch
        SINGLE VALUE              TRUE
        ID                        id-trader-at-linkId}
```

```
targetTraderInterfaceId ATTRIBUTE ::= {
        SUBTYPE OF                distinguishedName
        SINGLE VALUE              TRUE
        ID                        id-trader-at-targetTraderInterfaceId}

defPassOnFollowRule ATTRIBUTE ::= {
        WITH SYNTAX               FollowOption
        EQUALITY MATCHING RULE    integerMatch
        SINGLE VALUE              TRUE
        ID                        id-trader-at-defPassOnFollowRule }

limitingFollowRule ATTRIBUTE ::= {
        WITH SYNTAX               FollowOption
        EQUALITY MATCHING RULE    integerMatch
        SINGLE VALUE              TRUE
        ID                        id-trader-at-limitingFollowRule }

-- Proxy Offer Entry

proxyOfferEntryNF NAME-FORM ::= {
        NAMES                     proxyOfferEntry
        WITH ATTRIBUTES           {proxyOfferId}
        ID                        id-trader-nf-proxyOffer}

proxyOfferEntry OBJECT-CLASS ::= {
        SUBCLASS OF               {top}
        MUST CONTAIN              {proxyOfferId | proxyLookUpInterfaceId |
                                  hasDynamicProperties | hasModifiableProperties |
                                  ifMatchAll | constraintRecipe}
        MAY CONTAIN               { dynamicProps }
        ID                        id-trader-oc-proxyOffer}

proxyOfferId ATTRIBUTE ::= {
        WITH SYNTAX               DirectoryString {ub-proxy-offer-id}
        EQUALITY MATCHING RULE    caseIgnoreMatch
        SUBSTRINGS MATCHING RULE  caseIgnoreSubstringsMatch
        SINGLE VALUE              TRUE
        ID                        id-trader-at-proxyOfferId}

proxyLookUpInterfaceId ATTRIBUTE ::= {
        SUBTYPE OF                distinguishedName
        SINGLE VALUE              TRUE
        ID                        id-trader-at-proxyLookUpInterfaceId}
```

```
constraintRecipe ATTRIBUTE ::= {
        WITH SYNTAX               DirectoryString {ub-constraint-recipe}
        EQUALITY MATCHING RULE    caseIgnoreMatch
        SUBSTRINGS MATCHING RULE  caseIgnoreSubstringsMatch
        SINGLE VALUE              TRUE
        ID                        id-trader-at-constraintRecipe}

ifMatchAll ATTRIBUTE ::= {
        WITH SYNTAX               BOOLEAN
        EQUALITY MATCHING RULE    booleanMatch
        SINGLE VALUE              TRUE
        ID                        id-trader-at-ifMatchAll }

-- Interface Entry

interfaceEntry OBJECT-CLASS ::= {
        SUBCLASS OF               {top}
        KIND                      auxiliary
        MUST CONTAIN              {interfaceReference | interfaceType}
        ID                        id-trader-oc-interfaceEntry}

interfaceReference ATTRIBUTE ::= {
        WITH SYNTAX               InterfaceId
        EQUALITY MATCHING RULE    caseExactMatch
        ID                        id-trader-at-interfaceReference}

InterfaceId ::=                   DirectoryString {ub-interface-id}

interfaceType ATTRIBUTE ::= {
        WITH SYNTAX               InterfaceTypeName
        EQUALITY MATCHING RULE    objectIdentifierMatch
        ID                        id-trader-at-interfaceType}

InterfaceTypeName ::=             OBJECT IDENTIFIER

-- Object Identifier Assignments

id-trader                         OBJECT IDENTIFIER ::= {joint-iso-itu-t
                                  trader(100)}

id-trader-at                      OBJECT IDENTIFIER ::= {id-trader 4}
id-trader-oc                      OBJECT IDENTIFIER ::= {id-trader 6}
id-trader-mr                      OBJECT IDENTIFIER ::= {id-trader 13}
id-trader-nf                      OBJECT IDENTIFIER ::= {id-trader 15}
id-trader-oc-traderEntry          OBJECT IDENTIFIER ::= {id-trader-oc 0}
id-trader-oc-serviceOffer         OBJECT IDENTIFIER ::= {id-trader-oc 1}
id-trader-oc-proxyOffer           OBJECT IDENTIFIER ::= {id-trader-oc 2}
```

```
id-trader-oc-traderLink                    OBJECT IDENTIFIER ::= {id-trader-oc 3}
id-trader-oc-traderPolicy                  OBJECT IDENTIFIER ::= {id-trader-oc 4}
id-trader-oc-interfaceEntry                OBJECT IDENTIFIER ::= {id-trader-oc 5}

id-trader-nf-serviceOffer                  OBJECT IDENTIFIER ::= {id-trader-nf 1}
id-trader-nf-traderLink                    OBJECT IDENTIFIER ::= {id-trader-nf 2}
id-trader-nf-traderPolicy                  OBJECT IDENTIFIER ::= {id-trader-nf 3}
id-trader-nf-proxyOffer                    OBJECT IDENTIFIER ::= {id-trader-nf 4}

id-trader-at-traderInterface               OBJECT IDENTIFIER ::= {id-trader-at 0}
id-trader-at-typeRepos                     OBJECT IDENTIFIER ::= {id-trader-at 1}
id-trader-at-defSearchCard                 OBJECT IDENTIFIER ::= {id-trader-at 2}
id-trader-at-maxSearchCard                 OBJECT IDENTIFIER ::= {id-trader-at 3}
id-trader-at-defMatchCard                  OBJECT IDENTIFIER ::= {id-trader-at 4}
id-trader-at-maxMatchCard                  OBJECT IDENTIFIER ::= {id-trader-at 5}
id-trader-at-commonName                    OBJECT IDENTIFIER ::= {id-trader-at 6}
id-trader-at-dsaName                       OBJECT IDENTIFIER ::= {id-trader-at 7}
id-trader-at-defReturnCard                 OBJECT IDENTIFIER ::= {id-trader-at 10}
id-trader-at-maxReturnCard                 OBJECT IDENTIFIER ::= {id-trader-at 11}
id-trader-at-defHopCount                   OBJECT IDENTIFIER ::= {id-trader-at 12}
id-trader-at-maxHopCount                   OBJECT IDENTIFIER ::= {id-trader-at 13}
id-trader-at-defFollowPolicy               OBJECT IDENTIFIER ::= {id-trader-at 14}
id-trader-at-maxLinkFollowPolicy           OBJECT IDENTIFIER ::= {id-trader-at 15}
id-trader-at-maxFollowPolicy               OBJECT IDENTIFIER ::= {id-trader-at 16}
id-trader-at-supportsModifiableProperties  OBJECT IDENTIFIER ::= {id-trader-at 20}
id-trader-at-supportsDynamicProperties     OBJECT IDENTIFIER  ::= {id-trader-at 21}
id-trader-at-supportsProxyOffers           OBJECT IDENTIFIER ::= {id-trader-at 22}
id-trader-at-maxList                       OBJECT IDENTIFIER ::= {id-trader-at 23}
id-trader-at-requestIdStem                 OBJECT IDENTIFIER ::= {id-trader-at 24}
id-trader-at-typeManagementConstraint      OBJECT IDENTIFIER ::= {id-trader-at 25}
id-trader-at-searchConstraint              OBJECT IDENTIFIER ::= {id-trader-at 30}
id-trader-at-offerAcceptanceConstraint     OBJECT IDENTIFIER ::= {id-trader-at 31}
id-trader-at-sOfferId                      OBJECT IDENTIFIER ::= {id-trader-at 32}
id-trader-at-serviceTypeId                 OBJECT IDENTIFIER ::= {id-trader-at 33}
id-trader-at-serviceInterfaceId            OBJECT IDENTIFIER ::= {id-trader-at 34}
id-trader-at-hasDynamicProperties          OBJECT IDENTIFIER ::= {id-trader-at 35}
id-trader-at-hasModifiableProperties       OBJECT IDENTIFIER ::= {id-trader-at 40}
id-trader-at-dynamicProps                  OBJECT IDENTIFIER ::= {id-trader-at 41}
id-trader-at-linkId                        OBJECT IDENTIFIER ::= {id-trader-at 42}
id-trader-at-linkName                      OBJECT IDENTIFIER ::= {id-trader-at 43}
id-trader-at-targetTraderInterfaceId       OBJECT IDENTIFIER ::= {id-trader-at 44}
id-trader-at-defPassOnFollowRule           OBJECT IDENTIFIER ::= {id-trader-at 45}
id-trader-at-limitingFollowRule            OBJECT IDENTIFIER ::= {id-trader-at 50}
id-trader-at-proxyOfferId                  OBJECT IDENTIFIER ::= {id-trader-at 51}
id-trader-at-proxyLookUpInterfaceId        OBJECT IDENTIFIER ::= {id-trader-at 52}
id-trader-at-constraintRecipe              OBJECT IDENTIFIER ::= {id-trader-at 53}
```

```
id-trader-at-ifMatchAll                    OBJECT IDENTIFIER ::= {id-trader-at 55}
id-trader-at-interfaceReference            OBJECT IDENTIFIER ::= {id-trader-at 60}
id-trader-at-interfaceType                 OBJECT IDENTIFIER ::= {id-trader-at 61}

id-trader-mr-policySpecificationMatch      OBJECT IDENTIFIER ::= {id-trader-mr 1}
id-trader-mr-dynamicPropValueMatch         OBJECT IDENTIFIER ::= {id-trader-mr 2}

-- Upperbounds

ub-common-name                             INTEGER ::= 64
ub-request-id-stem                         INTEGER ::= 1024
ub-policy-string-rule                      INTEGER ::= 1024
ub-s-offer-id                              INTEGER ::= 64
ub-dynamic-value-extra-info                INTEGER ::= 1024
ub-link-name                               INTEGER ::= 64
ub-link-id                                 INTEGER ::= 64
ub-proxy-offer-id                          INTEGER ::= 64
ub-constraint-recipe                       INTEGER ::= 1024
ub-interface-id                            INTEGER ::= 1024

END
```

Annex B

Sample service description schema definition

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

```
PrinterServiceOfferDefinitions {joint-iso-itu-t 2}

DEFINITIONS ::=

BEGIN

IMPORTS

       informationFramework, selectedAttributeTypes

           FROM UsefulDefinitions {joint-iso-itu-t ds(5) modules(1)
                usefulDefinitions(0) 2}


     OBJECT-CLASS, MATCHING-RULE,ATTRIBUTE, top, ObjectClassKind, DistinguishedName

           FROM InformationFramework informationFramework


     DirectoryString {}, caseIgnoreMatch, caseIgnorSubstringsMatch, caseExactMatch,
     booleanMatch, integerMatch, integerOrderMatch

           FROM SelectedAttributeTypes  selectedAttributeTypes


     id-trader-at, id-trader-oc-serviceOffer

           FROM id-trader{joint-iso-itu-t trader(100)} ;


printerServiceOffer OBJECT-CLASS ::= {

       SUBCLASS OF              {top}

       KIND                     auxiliary

       MUST CONTAIN             printerType}

       MAY CONTAIN              {locationRoom | locationBuilding | costPerPage |
                                languagesSupported | pagesPerMinute | pageSize |
                                dotsPerInch | colourCapable | driverName | queueLength}

       ID                       id-trader-oc-serviceOffer-printer}


printerType ATTRIBUTE ::= {

       WITH SYNTAX              DirectoryString {ub-trader-so-printerType}

       EQUALITY MATCHING RULE   caseIgnoreMatch

       SUBSTRINGS MATCHING RULE caseIgnoreSubstringsMatch

       SINGLE VALUE             TRUE

       ID                       id-trader-at-so-printerType}


locationRoom ATTRIBUTE ::= {

       WITH SYNTAX              DirectoryString {ub-trader-so-locationRoom}

       EQUALITY MATCHING RULE   caseIgnoreMatch

       SUBSTRINGS MATCHING RULE caseIgnoreSubstringsMatch

       SINGLE VALUE             TRUE

       ID                       id-trader-at-so-locationRoom }
```

```
locationBuilding ATTRIBUTE ::= {
        WITH SYNTAX               DirectoryString {ub-trader-so-locationBlg}
        EQUALITY MATCHING RULE    caseIgnoreMatch
        SUBSTRINGS MATCHING RULE  caseIgnoreSubstringsMatch
        SINGLE VALUE              TRUE
        ID                        id-trader-at-so-locationBlg}

costPerPage ATTRIBUTE ::= {
        WITH SYNTAX               INTEGER
        EQUALITY MATCHING RULE    integerMatch
        ORDERING MATCHING RULE    integerOrderMatch
        SINGLE VALUE              TRUE
        ID                        id-trader-at-so-costPerPage}

languagesSupported ATTRIBUTE ::= {
        WITH SYNTAX               DirectoryString {ub-trader-so-langSupp}
        EQUALITY MATCHING RULE    caseIgnoreMatch
        SUBSTRINGS MATCHING RULE  caseIgnoreSubstringsMatch
        SINGLE VALUE              TRUE
        ID                        id-trader-at-so-langSupp}

pagesPerMinute ATTRIBUTE ::= {
        WITH SYNTAX               INTEGER
        EQUALITY MATCHING RULE    integerMatch
        ORDERING MATCHING RULE    integerOrderMatch
        SINGLE VALUE              TRUE
        ID                        id-trader-at-so-pagesPerMinute}

pagesPerMinute ATTRIBUTE ::= {
        WITH SYNTAX               INTEGER
        EQUALITY MATCHING RULE    integerMatch
        ORDERING MATCHING RULE    integerOrderMatch
        SINGLE VALUE              TRUE
        ID                        id-trader-at-so-pagesPerMinute}

dotsPerInch ATTRIBUTE ::= {
        WITH SYNTAX               INTEGER
        EQUALITY MATCHING RULE    integerMatch
        ORDERING MATCHING RULE    integerOrderMatch
        SINGLE VALUE              TRUE
        ID                        id-trader-at-so-dotsPerInch}
```

```
colourCapable ATTRIBUTE ::= {
        WITH SYNTAX               BOOLEAN
        EQUALITY MATCHING RULE    booleanMatch
        SINGLE VALUE              TRUE
        ID                        id-trader-at-so-colourCapable}

driverName ATTRIBUTE ::= {
        WITH SYNTAX               DirectoryString {ub-trader-so-driverName}
        EQUALITY MATCHING RULE    caseIgnoreMatch
        SUBSTRINGS MATCHING RULE  caseIgnoreSubstringsMatch
        SINGLE VALUE              TRUE
        ID                        id-trader-at-so-driverName}

queueLength ATTRIBUTE ::= {
        WITH SYNTAX               INTEGER
        EQUALITY MATCHING RULE    integerMatch
        ORDERING MATCHING RULE    integerOrderMatch
        SINGLE VALUE              TRUE
        ID                        id-trader-at-so-queueLength }

-- Object Identifiers

id-trader-oc-serviceOffer-printer
                                  OBJECT IDENTIFIER ::= {id-trader-oc-serviceOffer 0}

id-trader-at-so                   OBJECT IDENTIFIER ::= {id-trader-at 100}
id-trader-at-so-printerType       OBJECT IDENTIFIER ::= {id-trader-at-so 0}
id-trader-at-so-locationRoom      OBJECT IDENTIFIER ::= {id-trader-at-so 1}
id-trader-at-so-locationBlg       OBJECT IDENTIFIER ::= {id-trader-at-so 2}
id-trader-at-so-costPerPage       OBJECT IDENTIFIER ::= {id-trader-at-so 3}
id-trader-at-so-langSupp          OBJECT IDENTIFIER ::= {id-trader-at-so 4}
id-trader-at-so-pagesPerMinute    OBJECT IDENTIFIER ::= {id-trader-at-so 5}
id-trader-at-so-pageSize          OBJECT IDENTIFIER ::= {id-trader-at-so 6}
id-trader-at-so-dotsPerInch       OBJECT IDENTIFIER ::= {id-trader-at-so 7}
id-trader-at-so-colourCapable     OBJECT IDENTIFIER ::= {id-trader-at-so 8}
id-trader-at-so-driverName        OBJECT IDENTIFIER ::= {id-trader-at-so 9}
id-trader-at-so-queueLength       OBJECT IDENTIFIER ::= {id-trader-at-so 10}

-- Upperbounds

ub-trader-so-printerType          INTEGER ::= 64
ub-trader-so-locationRoom         INTEGER ::= 64
ub-trader-so-locationBlg          INTEGER ::= 64
ub-trader-so-langSupp             INTEGER ::= 64
ub-trader-so-pageSize             INTEGER ::= 64
ub-trader-so-driverName           INTEGER ::= 64
END
```

ITU-T RECOMMENDATIONS SERIES

Series A	Organization of the work of the ITU-T
Series B	Means of expression: definitions, symbols, classification
Series C	General telecommunication statistics
Series D	General tariff principles
Series E	Overall network operation, telephone service, service operation and human factors
Series F	Non-telephone telecommunication services
Series G	Transmission systems and media, digital systems and networks
Series H	Audiovisual and multimedia systems
Series I	Integrated services digital network
Series J	Transmission of television, sound programme and other multimedia signals
Series K	Protection against interference
Series L	Construction, installation and protection of cables and other elements of outside plant
Series M	TMN and network maintenance: international transmission systems, telephone circuits, telegraphy, facsimile and leased circuits
Series N	Maintenance: international sound programme and television transmission circuits
Series O	Specifications of measuring equipment
Series P	Telephone transmission quality, telephone installations, local line networks
Series Q	Switching and signalling
Series R	Telegraph transmission
Series S	Telegraph services terminal equipment
Series T	Terminals for telematic services
Series U	Telegraph switching
Series V	Data communication over the telephone network
Series X	**Data networks and open system communication**
Series Y	Global information infrastructure
Series Z	Programming languages