           Using the OSI Directory to Achieve User Friendly Naming

Status of this Memo

   This document specifies an Internet standards track protocol for the
   Internet community, and requests discussion and suggestions for
   improvements. Please refer to the current edition of the "Internet
   Official Protocol Standards" (STD 1) for the standardization state
   and status of this protocol. Distribution of this memo is unlimited.

Abstract

   The OSI Directory has user friendly naming as a goal. A simple
   minded usage of the directory does not achieve this. Two aspects not
   achieved are:

   o A user oriented notation

   o Guessability

   This proposal sets out some conventions for representing names in a
   friendly manner, and shows how this can be used to achieve really
   friendly naming. This then leads to a specification of a standard
   format for representing names, and to procedures to resolve them.
   This leads to a specification which allows directory names to be
   communicated between humans. The format in this specification is
   identical to that defined in [5], and it is intended that these
   specifications are compatible.

Table of Contents

1. Why a notation is needed

   Many OSI Applications make use of Distinguished Names (DN) as defined
   in the OSI Directory [1]. The main reason for having a notation for
   name format is to interact with a user interface. This specification
   is coming dangerously close to the sin of standardising interfaces.
   However, there are aspects of presentation which it is desirable to
   standardise.

   It is important to have a common format to be able to conveniently
   refer to names. This might be done to represent a directory name on
   a business card or in an email message. There is a need for a format
   to support human to human communication, which must be string based
   (not ASN.1) and user oriented.

   In very many cases, a user will be required to input a name. This
   notation is designed to allow this to happen in a uniform manner
   across many user interfaces. The intention is that the name can just
   be typed in. There should not be any need to engage in form filling
   or complex dialogue. It should be possible to take the "human"
   description given at the meeting, and use it directly. The means in
   which this happens will become clear later.

   This approach uses the syntax defined in [5] for representing
   distinguished names. By relaxing some of the constraints on this
   specification, it is argued that a more user oriented specification
   is produced. However, this syntax cannot be mapped algorithmically
   onto a distinguished name without the use of a directory.

   This notation is targeted towards a general user oriented system, and
   in particular to represent the names of humans. Other syntaxes may
   be more appropriate for other uses of the directory. For example,
   the OSF Syntax may be more appropriate for some system oriented uses.

(The OSF Syntax uses "/" as a separator, and forms names in a manner intended to resemble UNIX filenames).

This notation is targeted towards names which follow a particular DIT structure: organisationally oriented. This may make it inappropriate for some types of application. There may be a requirement to extend this notation to deal more cleanly with fully geographical names.

This approach effectively defines a definition of descriptive names on top of the primitive names defined by the OSI Directory.

2. The Notation

The notation used in this specification is defined in [5]. This notation defines an unambiguous representation of distinguished name, and this specification is designed to be used in conjunction with this format. Both specifications arise from the same piece of research work [4]. Some examples of the specification are given here. The author's User Friendly Name (UFN) might be written:

Steve Kille, Computer Science, University College London, GB

or

S. Kille, Computer Science, University College London, GB

This may be folded, perhaps to display in multi-column format. For example:

Steve Kille,
Computer Science,
University College London,
GB

Another UFN might be:

Christian Huitema, INRIA, FR

or
James Hacker,
Basingstoke,
Widget Inc,
GB

The final example shows quoting of a comma in an Organisation name:

L. Eagle, "Sue, Grabbit and Runn", GB

A purported name is what a user supplies to an interface for
resolution into one or more distinguished names. A system should
almost always store a name as a distinguished name. This will be
more efficient, and avoid problems with purported names which become
ambiguous when a new name appears. A user interface may display a
distinguished name, using the distinguished name notation. However,
it may display a purported name in cases where this will be more
pleasing to the user. Examples of this might be:

o Omission of the higher components of the distinguished name are
   not displayed (abbreviation).

o Omission of attribute types, where the type is unlikely to be
   needed to resolve ambiguity.

The ways in which a purported name may vary from a distinguished name
are now described:

Type Omission

There are two cases of this.

   o Schema defaulting. In this case, although the type is not
      present, a schema defaulting is used to deduce the type. The
      first two types of schema defaulting may be used to deduce a
      distinguished name without the use of the directory. The use
      of schema defaulting may be useful to improve the performance
      of UFN resolution. The types of schema defaulting are:

      -- Default Schema

      -- Context Dependent Default Schema

      -- Data Dependent Default Schema

   o Omission of the type to be resolved by searching.

Default Schema

The attribute type of an attribute may always be present. This may
be done to emphasise the type structure of a name. In some cases,
the typing may be omitted. This is done in a way so that in many
common cases, no attribute types are needed. The following type
hierarchy (schema) is assumed:

Common Name, (((Organisational Unit)*, Organisation,) Country).

Explicitly typed RDNs may be inserted into this hierarchy at any
point. The least significant component is always of type Common
Name. Other types follow the defined organisational hierarchy.
The following are equivalent:

Filestore Access, Bells, Computer Science,
University College London, GB

and

CN=Filestore Access, OU=Bells, OU=Computer Science,
O=University College London, C=GB

To interpet a distinguished name presented in this format, with some
or all of the attributes with the type not specified, the types are
derived according to the type hierarchy by the following algorithm:

  1. If the first attribute type is not specified, it is
      CommonName.

  2. If the last attribute type is not specified, it is Country.

  3. If there is no organisation explicitly specified, the last
      attribute with type not specified is of type Organisation.

  4. Any remaining attribute with type unspecified must be before
      an Organisation or OrganisationalUnit attribute, and is of
      type OrganisationalUnit.

To take a distinguished name, and generate a name of this format with
attribute types omitted, the following steps are followed.

  1. If the first attribute is of type CommonName, the type may be
      omitted.

  2. If the last attribute is of type Country, the type may be
      omitted.

  3. If the last attribute is of type Country, the last
      Organisation attribute may have the type omitted.

  4. All attributes of type OrganisationalUnit may have the type
      omitted, unless they are after an Organisation attribute or
      the first attribute is of type OrganisationalUnit.

   Context Dependent Default Schema

   The distinguished name notation defines a fixed schema for type
   defaulting. It may be useful to have different defaults in different
   contexts. For example, the defaulting convention may be applied in a
   modified fashion to objects which are known not to be common name
   objects. This will always be followed if the least significant
   component is explicitly typed. In this case, the following hierarchy
   is followed:

   ((Organisational Unit)*, Organisation,) Country

   Data Dependent Defaulting

   There are cases where it would be optimal
   to default according to the data. For example, in:

   Einar Stefferud, Network Management Associates, CA, US

   It would be useful to default "CA" to type State. This might be done
   by defaulting all two letter attributes under C=US to type State.

   General Defaulting

   A type may be omitted in cases where it does not follow a default
   schema hierarchy, and then type variants can be explored by
   searching. Thus a distinguished name could be represented by a
   uniquely matching purported name. For example,

   James Hacker,
   Basingstoke,
   Widget Inc,
   GB

   Would match the distinguished name:

   CN=James Hacker,
   L=Basingstoke,
   O=Widget Inc,
   C=GB

   Abbreviation

   Some of the more significant components of the DN will be omitted,
   and then defaulted in some way (e.g., relative to a local context).
   For example:

   Steve Kille

Could be interpreted in the context of an organisational default.

Local Type Keywords

Local values can be used to identify types, in addition to the
keywords defined in [5]. For example, "Organisation" may be
recognised as an alternative to "O".

Component Omission

An intermediate component of the name may be omitted. Typically this
will be an organisational unit. For example:

Steve Kille, University College London, GB

In some cases, this can be combined with abbreviation. For example:

Steve Kille, University College London

Approximation

Approximate renditions or alternate values of one or
more of the components will be supplied. For example:

Stephen Kille, CS, UCL, GB

or

Steve Keill, Comp Sci, Univarstiy College London, GB

Friendly Country

A "friendly country name" can be used instead of the ISO 3166 two
letter code. For example: UK; USA; France; Deutchland.

3. Communicating Directory Names

A goal of this standard is to provide a means of communicating
directory names. Two approaches are given, one defined in [5], and
the other here. A future version of these specifications may contain
only one of these approaches, or recommend use of one approach. The
approach can usually be distinguished implicitly, as types are
normally omitted in the UFN approach, and are always present in the
Distinguished Name approach. No recommendation is made here, but the
merits of each approach is given.

1. Distinguished Name or DN. A representation of the distinguished
    name, according to the specification of [5].

2. User Friendly Name or UFN. A purported name, which is expected to
    unambiguously resolve onto the distinguished name.

When a UFN is communicated, a form which should efficiently and
unambiguously resolve onto a distinguished name should be chosen.
Thus it is reasonable to omit types, or to use alternate values which
will unambiguously identify the entry in question (e.g., by use of an
alternate value of the RDN attribute type). It is not reasonable to
use keys which are (or are likely to become) ambiguous. The approach
used should be implicit from the context, rather than wired into the
syntax. The terms "Directory Name" and "X.500 Name" should be used
to refer to a name which might be either a DN or UFN. An example of
appropriate usage of both forms is given in the Section which defines
the Author's location in Section 12. Advantages of communicating the
DN are:

  o The Distinguished Name is an unambiguous and stable reference to
     the user.

  o The DN will be used efficiently by the directory to obtain
     information.

Advantages of communicating the UFN are:

  o Redundant type information can be omitted (e.g., "California",
     rather than "State=California", where there is known to be no
     ambiguity.

  o Alternate values can be used to identify a component. This might
     be used to select a value which is meaningful to the recipient, or
     to use a shorter form of the name. Often the uniqueness
     requirements of registration will lead to long names, which users
     will wish to avoid.

  o Levels of the hierarchy may be omitted. For example in a very
     small organisation, where a level of hierarchy has been used to
     represent company structure, and the person has a unique name
     within the organisation.

Where UFN form is used, it is important to specify an unambiguous
form. In some ways, this is analogous to writing a postal address.
There are many legal ways to write it. Care needs to be taken to
make the address unambiguous.

4. Matching a purported name

   The following approach specifies a default algorithm to be used with
   the User Friendly Naming approach. It is appropriate to modify this
   algorithm, and future specifications may propose alternative
   algorithms. Two simple algorithms are noted in passing, which may be
   useful in some contexts:

   1. Use type omission only, but otherwise require the value of the RDN
       attribute to be present.

   2. Require each RDN to be identified as in 1), or by an exact match
       on an alternate value of the RDN attribute.

   These algorithms do not offer the flexibility of the default
   algorithm proposed, but give many of the benefits of the approach in
   a very simple manner.

   The major utility of the purported name is to provide the important
   "user friendly" characteristic of guessability. A user will supply a
   purported name to a user interface, and this will be resolved onto a
   distinguished name. When a user supplies a purported name there is a
   need to derive the DN. In most cases, it should be possible to derive
   a single name from the purported name. In some cases, ambiguities
   will arise and the user will be prompted to select from a multiple
   matches. This should also be the case where a component of the name
   did not "match very well".

   There is an assumption that the user will simply enter the name
   correctly. The purported name variants are designed to make this
   happen! There is no need for fancy window based interfaces or form
   filling for many applications of the directory. Note that the fancy
   interfaces still have a role for browsing, and for more complex
   matching. This type of naming is to deal with cases where
   information on a known user is desired and keyed on the user's name.

4.1 Environment

   All matches occur in the context of a local environment. The local
   environment defines a sequence of names of a non-leaf objects in the
   DIT. This environment effectively defines a list of acceptable name
   abbreviations where the DUA is employed. The environment should be
   controllable by the individual user. It also defines an order in
   which to operate.

   This list is defined in the context of the number of name components
   supplied. This allows varying heuristics, depending on the
   environment, to make the approach have the "right" behaviour. In

most cases, the environment will start at a local point in the DIT,
and move upwards. Examples are given in Tables 1 and 2. Table 1
shows an example for a typical local DUA, which has the following
characteristics:

One component

Assumed first to be a user in the department, then a user or
department within the university, then a national organisation, and
finally a country.

Two components

Most significant component is first assumed to be a national
organisation, then a department (this might be reversed in some
organisations), and finally a country.

Three or more components

The most significant component is first assumed to be a country, then
a national organisation, and finally a department.

4.2 Matching

A purported name will be supplied, usually with a small number of
components. This will be matched in the context of an environment.
Where there are multiple components to be matched, these should be
matched sequentially. If an unambiguous DN is determined, the match
continues as if the full DN had been supplied. For example, if

```
+----------+------------------------------------+
|Number of |Environment                         |
|Components |                                    |
+----------+------------------------------------+
|1         |Physics, University College London, GB|
|          |University College London, GB       |
|          |GB                                  |
+----------+------------------------------------+
|2         |GB                                  |
|          |University College London, GB       |
|          |--                                  |
+----------+------------------------------------+
|3+        |--                                  |
|          |GB                                  |
|          |University College London, GB       |
+----------+------------------------------------+
```

             Table 1: Local environment for private DUA

```
         +-----------+----------+
         | Number of |Environment|
         | Components |          |
         +-----------+----------+
         |   1,2     | US       |
         |           | CA       |
         |           | --       |
         +-----------+----------+
         |   3+      | --       |
         |           | US       |
         |           | CA       |
         +-----------+----------+
```

           Table 2: Local environment for US Public DUA

Stephen Kille, UCL

is being matched in the context of environment GB, first UCL is
resolved to the distinguished name:

University College London, GB

Then the next component of the purported name is taken to determine
the final name. If there is an ambiguity (e.g., if UCL had made two
matches, both paths are explored to see if the ambiguity can be
resolved. Eventually a set of names will be passed back to the user.

Each component of the environment is taken in turn. If the purported
name has more components than the maximum depth, the environment
element is skipped. The advantage of this will be seen in the
example given later.

A match of a name is considered to have three levels:

Exact A DN is specified exactly

Good Initially, a match should be considered good if it is
     unambiguous, and exactly matches an attribute value in the entry.
     For human names, a looser metric is probably desirable (e.g.,
     S Kille should be a good match of S. Kille, S.E. Kille or Steve
     Kille even if these are not explicit alternate values).

Poor Any other substring or approximate match

Following a match, the reference can be followed, or the user
prompted. If there are multiple matches, more than one path may be
followed. There is also a shift/reduce type of choice: should any
partial matches be followed or should the next element of the

environment be tried. The following heuristics are suggested, which
may be modified in the light of experience. The overall aim is to
resolve cleanly specified names with a minimum of fuss, but give
sufficient user control to prevent undue searching and delay.

1. Always follow an exact match.

2. Follow all good matches if there are no exact matches.

3. If there are only poor matches, prompt the user. If the user
    accepts one or more matches, they can be considered as good. If
    all are rejected, this can be treated as no matches.

4. Automatically move to the next element of the environment if no
    matches are found.

When the final component is matched, a set of names will be
identified. If none are identified, proceed to the next environment
element. If the user rejects all of the names, processing of the
next environment element should be confirmed.

The exact approach to matching will depend on the level of the tree
at which matching is being done. We can now consider how attributes
are matched at various levels of the DIT.

There is an issue of approximate matching. Sometimes it helps, and
sometimes just returns many spurious matches. When a search is
requested, all relevant attributes should be returned, so that
distinguished and non-distinguished values can be looked at. This
will allow a distinction to be made between good and poor matches.
It is important that where, for example, an acronym exactly matches
an organisation, that the user is not prompted about other
organisations where it matches as a substring.

4.3 Top Level

In this case, a match is being done at the root of the DIT. Three
approaches are suggested, dependent on the length of supplied name.
All lead to a single level search of the top level of the DIT.

Exactly 2

This is assumed to be a 3166 two letter country code, or an exact
match on a friendly country or organisation (e.g., UK or UN). Do
exact match on country and friendly country.

Greater than 2

Make an approximate and substring match on friendly country and
organisation.

4.4 Intermediate Level

Once the root level has been dealt with, intermediate levels will be
looking for organisational components (Organisation, Locality, Org
Unit). In some cases, private schema control will allow the system
to determine which is at the next level. In general this will not be
possible. In each case, make a substring and approximate match
search of one level. The choice depends on the base object used in
the search.

1. If DN has no Organisation or Locality, filter on Organisation and
    Locality.

2. If DN has Org Unit, filter on Org Unit.

3. If DN has Organisation, filter on Locality and Org Unit.

4. If DN has Locality, filter on Organisation.

These allow some optimisation, based on legal choices of schema.
Keeping filters short is usually desirable to improve performance. A
few examples of this, where a base object has been determined (either
by being the environment or by partial resolution of a purported
name), and the next element of a purported name is being considered.
This will generate a single level search. What varies is the types
being filtered against. If the DN is:

University College London, GB

The search should be for Org Unit or Locality. If the DN is:

Organisation=UN

the search should be for Org Unit or Locality.

There may be some improvements with respect to very short keys. Not
making approximate or substring matches in these cases seems sensible
(It might be desirable to allow "*" as a part of the purported name
notation.)

4.5 Bottom Level

   The "Bottom Level" is to deal with leaf entries in the DIT. This will
   often be a person, but may also be a role, an application entity or
   something else.

   The last component of a purported name may either reference a leaf or
   non-leaf. For this reason, both should be tested for. As a
   heuristic, if the base object for the search has two or more
   components it should be tested first as a bottom level name and then
   intermediate. Reverse this for shorter names. This optimises for
   the (normal) case of non-leaves high up the tree and leaves low down
   the tree.

   For bottom level names, make an approximate and substring match
   against Common Name, Surname, and User ID. Where common name is
   looked for, a full subtree search will be used when at the second
   level of the DIT or lower, otherwise a single level search.

   For example, if I have resolved a purported name to the distinguished
   name

   University College London, GB

   and have a single component Bloggs, this will generate a subtree
   search.

5. Examples

   This is all somewhat confusing, and a few examples are given. These
   are all in the context of the environment shown in Table 1 on Page
   13.

   If "Joe Bloggs" is supplied, a subtree search of

   Physics, University College London, GB

   will be made, and the user prompted for "Joseph Z. Bloggs" as the
   only possible match.

   If "Computer Science" is supplied, first

   Physics, University College London, GB

   will be searched, and the user will reject the approximate match of
   "Colin Skin". Then a subtree search of

   University College London, GB

will be made, looking for a person. Then a single level search will
be made looking for Org Unit, and

Computer Science, University College London, GB

will be returned without prompting (exact match). Supplying "Steve
Kille" will lead to a failed subtree search of

Physics, University College London, GB

and lead straight to a subtree search of

University College London, GB

This will lead to an exact value match, and so a single entry
returned without prompting.

If "Andrew Findlay, Brunel" is supplied, the first element of the
environment will be skipped, single level search of "Brunel" under
"GB" will find:

Brunel University, GB

and a subtree search for "Andrew Findlay" initiated. This will yield

Andrew Findlay, Computing and Media Services, Brunel University, GB

Dr A J Findlay, Manufacturing and Engineering Systems, Brunel
University, GB

and the user will be prompted with a choice.

This approach shows how a simple format of this nature will "do the
right thing" in many cases.

6. Support required from the standard

   Fortunately, all that is needed is there! It would be useful to have
   "friendly country name" as a standard attribute.

7. Support of OSI Services

   The major focus of this work has been to provide a mechanism for
   identifying Organisations and Users. A related function is to
   identify applications. Where the Application is identified by an AET
   (Application Entity Title) with an RDN of Common Name, this
   specification leads to a natural usage. For example, if a filestore
   is named "gannet", then this could easily be identified by the name:

Gannet, Computer Laboratory, Cambridge University, GB

In normal usage, this might lead to access (using a purported name)
of:

FTAM gannet,cambridge

A second type of access is where the user identifies an Organisation
(Organisational Unit), and expects to obtain a default service. The
service is implied by the application, and should not require any
additional naming as far as the user is concerned. It is proposed
that this is supported by User Friendly Naming in the following way.

1. Determine that the purported name identifies a non-leaf object,
    which is of object class Organisation or Organisational Unit or
    Locality.

2. Perform a single level search for Application Entities which
    support the required application contexts. This assumes that all
    services which are supporting default access for the organisation
    are registered at one level below (possibly by the use of
    aliases), and that other services (specific machines or parts of
    the organisation) are represented further down the tree. This
    seems to be a reasonable layout, and its utility can be evaluated
    by experiment.

8. Experience

An experimental implementation of this has been written by Colin
Robbins. The example in Figure 1 shows that it can be very effective
at locating known individuals with a minimum of effort. This code has
been deployed within the "FRED" interface of the PSI Pilot [9], and
within an prototype interface for managing distribution lists. The
user reaction has been favourable:

Some issues have arisen from this experience:

  o Where there is more than one level of Organisational Unit, and the
     user guesses one which is not immediately below the organisation,
     the algorithm works badly. There does not appear to be an easy
     fix for this. It is not clear if this is a serious deficiency.

  o Substring searching is currently done with leading and trailing
     wildcards. As many implementations will not implement leading
     wildcards efficiently, it may be preferable to only use trailing
     wildcards. The effect of this on the algorithm needs to be
     investigated.

   Implementors of this specification are encouraged to investigate
   variants of the basic algorithm. A final specification should depend
   on experience with such variants.

9. Relationship to other work

   Colin Robbin's work on the interface "Tom" and implementation of a
   distribution list interface strongly influenced this specification
   [6].

   Some of the ideas used here originally came from a UK Proposal to the
   ISO/CCITT Directory Group on "New Name Forms" [2]. This defined, and
   showed how to implement, four different types of names:

   Typed and Ordered The current Distinguished Name is a restricted
   example of this type of name.

```
    -> t hales, csiro, australia
    Found good match(es) for 'australia'
    Found exact match(es) for 'csiro'
    Please select from the following:
       Trevor Hales, OC, HPCC, DIT, IICT, CSIRO, AU [y/n] ? y
    The following were matched...
       Trevor Hales, OC, HPCC, DIT, IICT, CSIRO, AU

    -> g michaelson, queensland, au
    Found exact match(es) for 'au'
    Please select from the following:
       University of Queensland, AU [y/n] ? y
       Axolotl, AU [y/n] ? n
    Please select from the following:
       George Michaelson, Prentice Computer Centre, University of
       Queensland, AU
    [y/n] ? y
       Manager, University of Queensland, AU [y/n] ? n
    The following were matched...
       George Michaelson, Prentice Computer Centre, University of
       Queensland, AU

    -> r needham, cambridge
    Found good match(es) for 'cambridge'
    Please select from the following:
       Roger Needham, Computer Lab, Cambridge University [y/n] ? y
    The following were matched...
       Roger Needham, Computer Lab, Cambridge University

    -> kirstein
    Found good match(es) for 'kirstein'
    The following were matched...
       Peter Kirstein
```

             Figure 1: Example usage of User Friendly Naming

Untyped and Ordered

This is the type of name proposed here (with some extensions to allow
optional typing). It is seen as meeting the key user requirement of
disliking typed names, and is efficient to implement.

Typed and Unordered

This sort of name is proposed by others as the key basis for user
friendly naming. Neufeld shows how X.500 can be used to provide this
[7], and Peterson proposes the Profile system to provide this [8].

The author contends that whilst typed naming is interesting for some types of searching (e.g., yellow page searching), it is less desirable for naming objects. This is borne out by operational experience with OSI Directories [3].

Untyped and Unordered

Surprisingly this form of name can be supported quite easily. However, a considerable gain in efficiency can be achieved by requiring ordering. In practice, users can supply this easily. Therefore, this type of name is not proposed.

10. Issues

The following issues are noted, which would need to be resolved before this document is progressed as an Internet Standard.

Potential Ambiguity

Whilst the intention of the notation is to allow for specification of alternate values, it inherently allows for ambiguous names to be specified. It needs to be demonstrated that problems of this characteristic are outweighed by other benefits of the notation.

Utility

Determine that the specification is being implemented and used.

Performance

Measurements on the performance implications of using this approach should be made.

Alogrithm

The utility of the algorithm, and possible variants, should be investigated.

This format, and the procedures for resolving purported names, should be evolved to an Internet Standard. The syntax can be expected to be stable. In light of experience, the algorithm for resolving purported names may be changed.

11. References

    [1] The Directory --- overview of concepts, models and services,
        1993. CCITT X.500 Series Recommendations.

    [2] S.E. Kille. New name forms, May 1989. ISO/IEC/JTC 21/ WG4/N797
        UK National Body Contribution to the Oslo Directory Meeting.

    [3] S.E. Kille. The THORN large scale pilot exercise. Computer
        Networks and ISDN Systems, 16(1):143--145, January 1989.

    [4] S.E. Kille. Using the OSI directory to achieve user friendly
        naming. Research Note RN/20/29, Department of Computer Science,
        University College London, February 1990.

    [5] Kille, S., "A String Representation of Distinguished Names", RFC
        1779, ISODE Consortium, March 1995.

    [6] S.E. Kille and C.J. Robbins. The ISO development environment:
        User's manual (version 7.0), July 1991. Volume 5: QUIPU.

    [7] G.W. Neufeld. Descriptive names in X.500. In SIGCOMM 89
        Symposiun Communications Architectures and Protocols, pages 64--
        71, September 1989.

    [8] L.L. Petersen. The profile naming service. ACM Transactions on
        Computing Systems, 6(4):341--364, November 1988.

    [9] M.T. Rose. Realizing the White Pages using the OSI Directory
        Service. Technical Report 90--05--10--1, Performance Systems
        International, Inc., May 1990.

12. Security Considerations

     Security issues are not discussed in this memo.

13. Author's Address

     Steve Kille
     ISODE Consortium
     The Dome
     The Square
     Richmond, Surrey
     TW9 1DT
     England

     Phone:+44-181-332-9091
     EMail: S.Kille@ISODE.COM

     DN: CN=Steve Kille,
     O=ISODE Consortium, C=GB

     UFN: S. Kille,
     ISODE Consortium, GB

A. Pseudo-code for the matching algorithm

   The following pseudo-code is intended to clarify the matching
   algorithm. The language uses ASN.1 data types, with flow control
   "C"-like, but with keywords upper-cased.

```
PurportedName ::= SEQUENCE OF String
                -- simplication, as attribute types can optionally be
                -- specified

                -- Each element of the Purported Name is a string
                -- which has been parsed from the BNF

Attribute ::= SEQUENCE {
        type OBJECT IDENTIFIER,
        value ANY }

RDN ::= Attribute -- simplification, as can be multi-value

DN ::= SEQUENCE OF RDN

Environment ::= SEQUENCE OF DN

EnvironmentList ::= SEQUENCE OF SEQUENCE {
                    lower-bound INTEGER,
                    upper-bound INTEGER,
                    environment Environment }


friendlyMatch(p:  PurportedName; el:  EnvironmentList):   SET OF DN
{
                          -- Find correct environment

        IF length(el) == 0 THEN return(NULL);

        IF length(p) <= head(el).upper-bound
                    && length(p) >= head(el).lower-bound THEN
            return envMatch (p, head(el).environment);
        ELSE
            return(friendlyMatch(p, tail(el));
}

envMatch(p:  PurportedName; e:  Environment):   SET OF DN
{
                          -- Check elements of environment
                          -- in the defined order

        matches:  SET OF DN;
```

```
        IF length(e) == 0 THEN return(NULL);

        matches = purportedMatch(head(e).DN, p)
        IF matches != NULL THEN
                return(matches);
        ELSE
                return(envMatch(p, tail(e)));
}


purportedMatch(base:  DN; p:  PurportedName):  SET OF DN
{
        s:  String = head(p);
        matches:  SET OF DN = NULL;

        IF length(p) == 1 THEN
                IF length(base) == 0 THEN
                        IF (matches = rootSearch(s)) != NULL THEN
                                return(matches);
                        ELSE return(leafSearch(base, s, one-level);
                ELSE IF length(base) == 1 THEN
                        IF (matches = intSearch(base, s)) != NULL THEN
                                return(matches);
                        ELSE return(leafSearch(base, s, one-level);
                ELSE
                        IF (matches = leafSearch(base, s, subtree)) !=
                                NULL THEN return(matches);
                        ELSE return(intsearch(base, s);


        IF length(base) == 0 THEN
                FOR x IN rootSearch(s) DO
                        matches += (purportedMatch(x, tail(p));
        ELSE
                FOR x IN intSearch(base, s) DO
                        matches += (purportedMatch(x, tail(p));
        return(matches);
}
```

```
-- General.    Might need to tighten the filter for short strings,
-- in order to stop being flooded.    Alternatively, this could be
-- done if the loose search hits a size limit

rootSearch(s:  String):  SET OF DN
{
        IF length(s) == 2 THEN
                return(search(NULL, one-level, s, {CountryName,
                        FriendlyCountryName, OrganizationName},
                        {exact}, {Country, Organisation}));
                        -- test exact match only
                        -- probably a country code
        ELSE
                return(search(NULL, one-level, s, {OrganizationName,
                        FriendlyCountryName}, {substring, approx},
                        {Country, Organisation}));
}


intSearch( base:  DN; s:  String)
{
        IF present(base, OrgUnitName) THEN
                return(search(base, one-level, s, {OrgUnitName},
                        {substring, approx}, {OrgUnit}));
        ELSE IF present(base, OrganisationName) THEN
                return(search(base, one-level, s, {OrgUnitName,
                        LocalityName}, {substring, approx},
                        {Organization, OrgUnit, Locality}));
        ELSE IF present(base, LocalityName) THEN
                return(search(base, one-level, s, {OrganisationName},
                        {substring, approx}, {Locality}));
        ELSE
                return(search(base, one-level, s, {OrganisationName,
                        LocalityName}, {substring, approx},
                        {Organisation, Locality}));
}


present(d:  DN; t:  AttributeType):  BOOLEAN
{
        FOR x IN d DO
                IF x.type == t THEN return(TRUE);
        return(FALSE);
}

SearchScope := ENUMERATED (base-object, one-level, subtree)

leafSearch(base:  DN; s:  String; search-scope:  SearchScope)
```

```
{
        return(search(base, search-scope, s, {CommonName, Surname,
                UserId}, {substring, approx}));
}

search(base:  DN; search-scope:  SearchScope; s:  string;
        alist SET OF AttributeType; matchtypes SET OF MatchType
        objectClasses SET OF ObjectClass OPTIONAL): SET OF DN
{
        -- mapped onto Directory Search, with OR conjunction
        -- of filter items

        return dNSelect (s, search-results, alist);
}

read(base:  DN; alist SET OF AttributeType):  SET OF Attribute;
{
        -- mapped onto Directory Read
        -- Types repeated to deal with multiple values
        -- This would be implemented by returning selected info
        -- with the search operation
}

dNSelect(s:  String; dlist SET OF DN;
                    alist:  SET OF AttributeType):16SET0OF DN
{
        exact, good:  SET OF DN;

        FOR x IN dlist DO
                IF last(DN).Value == s THEN
                        exact += x;
                ELSE IF FOR y IN read(x, alist) DO
                        IF y.value == s THEN
                                good += x;

        IF exact != NULL THEN return(exact);
        IF good != NULL THEN return(good);
        return(userQuery(dlist));
}

userQuery(dlist SET OF DN): SET OF DN
{
        -- pass back up for manual checking
        -- user can strip all matches to force progres....
}

head()    -- return first element of list
tail()    -- return list with first element removed
```

```
length()  -- return size of list
last()    -- return last element of list
```

                    Figure 2: Matching Algorithm