        A Tagged Index Object for use in the Common Indexing Protocol

Status of this Memo

Copyright Notice

Abstract

   This document defines a mechanism by which information servers can
   exchange indices of information from their databases by making use of
   the Common Indexing Protocol (CIP). This document defines the
   structure of the index information being exchanged, as well as the
   appropriate meanings for the headers that are defined in the Common
   Indexing Protocol. It is assumed that the structures defined here
   can be used by X.500 DSAs, LDAP servers, Whois++ servers, CSO Ph
   servers and many others.

Table of Contents

1. Introduction

   The Common Indexing Protocol (CIP) as defined in [1] proposes a
   mechanism for distributing searches across several instances of a
   single type of search engine to create a global directory. CIP
   provides a scalable, flexible scheme to tie individual databases into
   distributed data warehouses that can scale gracefully with the growth
   of the Internet. CIP provides a mechanism for meeting these goals
   that is independent of the access method that is used to access the
   data that underlies the indices. Separate from CIP is the definition
   of the Index Object that is used to contain the information that is
   exchanged among Index Servers. One such Index Object that has
   already been defined is the Centroid that is derived from the Whois++
   protocol [2].

   The Centroid does not meet all the requirements for the exchange of
   index information amongst information servers. For example, it does
   not support the notion of incremental updates natively. For
   information servers that contain millions of records in their
   database, constant exchange of complete dredges of the database is
   bandwidth intensive. The Tagged Index Object is specifically
   designed to support the exchange of index update information. This
   design comes at the cost of an increase in the size of the index
   object being exchanged. The Centroid is also not tailored to always
   be able to give boolean answers to queries. In the Centroid Model,
   "an index server will take a query in standard Whois++ format, search
   its collections of centroids and other forward information, determine
   which servers hold records which may fill that query, and then

notifies the user's client of the next servers to contact to submit
the query." [2] Thus, the exchange of Centroids amongst index servers
allows hints to be given about which information server actually
contains the information. The Tagged Index Object labels the various
pieces of information with identifiers that tie the individual object
attributes back to an object as a whole. This "tagging" of
information allows an index server to be more capable of directing a
specific query to the appropriate information server. Again, this
feature is added to the Tagged Index Object at the expense of an
increase in the size of the index object.

2. Background

   The Lightweight Directory Access Protocol (LDAP) is defined in [3],
   and it defines a mechanism for accessing a collection of information
   arranged hierarchically in such a way as to provide a globally
   distributed database which is normally called the Directory
   Information Tree (DIT). Some distinguishing characteristics of LDAP
   servers are that normally, several servers cooperate to manage a
   common subtree of the DIT. LDAP servers are expected to respond to
   requests that pertain to portions of the DIT for which they have
   data, as well as for those portions for which they have no
   information in their database. For example, the LDAP server for a
   portion of the DIT in the United States (c=US) must be able to
   provide a response to a Search operation that pertains to a portion
   of the DIT in Sweden (c=se). Normally, the response given will be a
   referral to another LDAP server that is expected to be more
   knowledgeable about the appropriate subtree. However, there is no
   mechanism that currently enables these LDAP servers to refer the LDAP
   client to the supposedly more knowledgeable server. Typically, an
   LDAP (v3) server is configured with the name of exactly one other
   LDAP server to which all LDAP clients are referred when their
   requests fall outside the subtree of the DIT for which that LDAP
   server has knowledge. This specification defines a mechanism whereby
   LDAP server can exchange index information that will allow referrals
   to point towards a clearly accurate destination.

   The X.500 series of recommendations defines the Directory Information
   Shadowing Protocol (DISP) [4] which allows X.500 DSAs to exchange
   information in the DIT. Shadowing allows various information from
   various portions of the DIT to be replicated amongst participating
   DSAs. The design point of DISP is improved at the exchange of entire
   portions of the DIT, whereas the design point of CIP and the Tagged
   Index Object is optimized at the exchange of structural index
   information about the DIT, and improving the performance of tree
   navigation amongst various information servers. The Tagged Index
   Object is more appropriate for the exchange of index information than
   is DISP. DISP is more targeted at DIT distribution and fault

tolerance. DISP is thus more appropriate for the exchange of the
data in order to spread the load amongst several information servers.
DISP is tailored specifically to X.500 (and other hierarchical
directory systems), while the Tagged Index Object and CIP can be used
in a wide variety of information server environments.

While DISP allows an individual directory server to collect
information about large parts of the DIT, it would require a huge
database to collect all the replicas for a significant portion of the
DIT. Furthermore, as X.525 states: "Before shadowing can occur, an
agreement, covering the conditions under which shadowing may occur is
required. Although such agreements may be established in a variety
of ways, such as policy statements covering all DSAs within a given
DMD ...", where a DMD is a Directory Management Domain. This is
owing to the case that the data in the DIT is being exchanged amongst
DSA rather than only the information required to maintain an Index.
In many environments such an agreement is not appropriate, and to
collect information for a meaningful portion of the DIT, many
agreements may need to be arranged.

3. Object

What is desired is to have an information server (or network of
information servers) that can quickly respond to real world requests,
like:

-      What is Tim Howes's email address? This is much harder than;
       What email address does Tim Howes at Netscape have ?

-      What is the X.509 certificate for Fred Smith at compuserve.com?
       One certainly doesn't want to search CompuServe's entire
       directory tree to find out this one piece of information. I
       also don't want to have to shadow the entire CompuServe
       directory subtree onto my server. If this request is being made
       because Fred is trying to log into my server, I'd certainly want
       to be able to respond to the BIND in real time.

-      Who are all the people at Novell that have a title of
       programmer?

all these requests can reasonably be translated into LDAP or Whois++,
and other directory access protocol queries. They can also be
serviced in a straightforward way by the users home information
server if it has the appropriate reference information into the
database that contains the source data. Here, the first server would
be able to "chain" the request for the user. Alternatively, a
precise referral could be returned. If the home information server
wants to service (i.e chain) the request based on the index

information that it has on hand, this servicing could be done several different means:

-       issuing LDAP operations to the remote directory server

-       issuing DSP operations to the remote directory server

-       issuing DAP operations to the remote directory server

-       issuing Whois++ operations to the remote Whois++ server

-       ...

4. The Tagged Index Object

This section defines a Tagged Index Object that can be exchanged by Information Servers using CIP. While often it is acceptable for Information Servers to make use of the Centroid definition (from [2]) to exchange index information, the goals in defining a new construct are multi-pronged:

-       When the Information Server receives a search request that warrants that a referral be returned, allow the server to return a referral that will point client to a server that is most likely able to answer the request correctly. False positive referrals (the search turns up hits in the index object that generate referrals to servers that don't hold the desired information) can be reduced, depending on the choice of attribute tokenization types that are used.

-       Potentially allow incremental updates that will then consume substantially less bandwidth then if full updates always had to be used.

4.1. The Agreement

Before a Tagged Index Object can be exchanged, the organization that administers the object supplier and the organization that administers the object consumer must reach an agreement on how the servers will communicate. This agreement contains the following:

-       "index-type": This specification describes the index type "x-tagged-index-1"

-       "dsi": An OID that uniquely identifies the subtree and scope. This field is not explicitly necessary, as it may not provide information beyond what is contained in the "base-uri" below.

        -       "base-uri": One or more URI's that will form the base of any
                referrals created based on the index object that is governed by
                this agreement. For example, in the LDAP URL format [8] the
                base-uri would specify (among other items): the LDAP host, the
                base object to which this index object refers (e.g. c=SE), and
                the scope of the index object (e.g. single container).

        -       "supplier": The hostname and listening portnumber of the
                supplier server, as well as any alternative servers holding that
                same naming contexts, if the supplier is unavailable.

        -       "consumeraddr": This is a URI of the "mailto:" form, with the
                RFC 822 email address of the consumer server. Further versions
                of this draft allow other forms of URI, so that the consumer may
                retrieve the update via the WWW, FTP or CIP.

        -       "updateinterval": The maximum duration in seconds between
                occurances of the supplier server generating an update. If the
                consumer server has not received an update from the supplier
                server after waiting this long since the previous update, it is
                likely that the index information is now out of date. A typical
                value for a server with frequent updates would be 604800
                seconds, or every week. Servers whose DITs are only modified
                annually could have a much longer update interval.

        -       "attributeNamespace": Every set of index servers that together
                wants to support a specific usage of indeces, has to agree on
                which attributenames to use in the index objects. The
                participating directory servers also has to agree on the mapping
                from local attributenames to the attributenames used in the
                index. Since one specific index server might be involved in
                several such sets, it has to have some way to connect a update
                to the proper set of indexes. One possible solution to this
                would be to use different DSIs.

        -       "consistencybase": How consistency of the index is maintained
                over incremental updates:

                    "complete" - every change or delete concerning one object
                    has to contain all tokens connected to that object. This
                    method must be supported by any server who wants to comply
                    with this standard.

                    "tag" - starting at a full update every incremental update
                    refering back to this full updated has to maintain state-
                    information regarding tags, such that a object within the
                    original database is assigned the same tagnumber every time.
                    This method is optional.

"unique" - every object in the Dataset has to have a unique
value for a specific attribute in the index. A example of
such a attribute could be the distinguishedName attribute.
This method is also optional.

- "securityoption": Whether and how the supplier server should
  sign and encrypt the update before sending it to the consumer
  server. Options for this version of the specification are:

  "none" - the update is sent in plaintext

  "PGP/MIME": the update is digitally signed and encrypted
  using PGP [9]

  "S/MIME": the update is digitally signed and encrypted using
  S/MIME [10]

  "SSLv3": the update is digitally signed and encrypted using
  an SSLv3 connection [11]

  "Fortezza": the update is digitally signed and encrypted
  using Fortezza [5]

It is recommended that the "PGP/MIME" option be used when exchanging
sensitive information across public networks, and both the supplier
and consumer have PGP keys. The "Fortezza" option is intended for use
in environments where security protocols are based on Fortezza-
compatible devices. The "S/MIME" option can be used with both the
supplier and consumer have RSA keys and can make use of the PKCS
protocols defined in the S/MIME specification. The "SSLv3" option can
be used when both the supplier and consumer have access to SSL
services, have server certificates, and can mutually authenticate
each other.

- Security Credentials: The long-term cryptographic credentials
  used for key exchange and authentication of the consumer and
  supplier servers, if a security option was selected. For
  "PGP/MIME," this will be the trusted public keys of both
  servers. For "Fortezza," this will be the certificate paths of
  both servers to a common point of trust. For "S/MIME" and
  "SSLv3" these will be the certificates of the supplier and
  consumer.

Note that if the index server maintains the information that
would appear in the agreement in a directory according to the
definitions in [7], then no real formal agreement between the
two parties needs to be put in place, and the information that
is required for communication between the two index servers is
derived automatically from the directory.

4.2. Content Type

The update consists of a MIME object of type application/cip-index-
object. The parameters are:

"type": this has value "application/index.obj.tagged".

"dsi": the DSI (if any) from the agreement.

"base-uri". A set of URIs, separated by spaces. In each URI, the
hostname/portno must be distinct, and based on the "supplier" part
of the agreement.

The payload is mostly textual data but may include bytes with the
high bit set. The originating information server should set the
content-transfer-encoding as appropriate for the information included
in the payload.

This object may be encapsulated in a wrapper content (such as
multipart/signed) or be encrypted as part of the security procedures.
The resulting content can the distributed, for example via electronic
mail. For example,

From: supplier@sup.com Date: Thu, 16 Jan 1997 13:50:37 -0500
Message-Id: <199701161850.NAA29295@sup.com>;
To: consumer@consumer.com   <<-- from consumer server address

Reply-to: supplier-admin@sup.com
MIME-Version: 1.0
Content-Type: application/index.obj.tagged;
dsi=1.3.6.1.4.1.1466.85.85.1.2.3.4.5.6.7.8.9.10.11.12.13.14.15.16;
base-uri="ldap://sup.com/dc=sup,dc=com ldap://alt.com/dc=sup,dc=com"

The payload is series of CRLF-terminated lines. The payload is UTF-8.
Some supplier servers may only be able to generate the printable US-
ASCII subset of UTF-8, but all consumer servers must be able to
handle the full range of Unicode characters when decoding the
attribute values (in the "attr-value" field in the BNF below).

4.3. Tagged Index BNF

   The Tagged Index object has the following grammar, expressed in
   modified BNF format:

```
index-object = 0*(io-part SEP) io-part
io-part      = header SEP schema-spec SEP index-info
header       = version-spec SEP update-type SEP this-update SEP
                  last-update context-size name-space SEP
version-spec = "version:" *SPACE "x-tagged-index-1"
update-type  = "updatetype:" *SPACE ( "total" |
                  ( "incremental" [*SPACE "tagbased"|"uniqueIDbased" ] ) )
this-update  = "thisupdate:" *SPACE TIMESTAMP
last-update  = [ "lastupdate:" *SPACE TIMESTAMP SEP]
context-size = [ "contextsize:" *SPACE 1*DIGIT SEP]
schema-spec  = "BEGIN IO-Schema" SEP 1*(schema-line SEP)
                  "END IO-Schema"
schema-line  = attribute-name ":" token-type
token-type   = "FULL" | "TOKEN" | "RFC822" | "UUCP" | "DNS"
index-info   = full-index | incremental-index
full-index   = "BEGIN Index-Info" SEP 1*(index-block SEP)
                  "END Index-Info"
incremental-index = 1*(add-block | delete-block | update-block)
add-block    = "BEGIN Add Block" SEP 1*(index-block SEP)
                  "END Add Block"
delete-block = "BEGIN Delete Block" SEP 1*(index-block SEP)
                  "END Delete Block"
update-block = "BEGIN Update Block" SEP
                  0*(old-index-block SEP)
                  1*(new-index-block SEP)
                  "END Update Block"
old-index-block = "BEGIN Old" SEP 1*(index-block SEP)
                  "END Old"
new-index-block = "BEGIN New" SEP 1*(index-block SEP)
                  "END New"
index-block  = first-line 0*(SEP cont-line)
first-line   = attr-name ":" *SPACE taglist "/" attr-value
cont-line    = "-" taglist "/" attr-value
taglist      = tag 0*("," tag) | "*"
tag          = 1*DIGIT ["-" 1*DIGIT]
attr-value   = 1*(UTF8)
attr-name    = 1*(NAMECHAR)
TIMESTAMP    = 1*DIGIT
NAMECHAR     = DIGIT | UPPER | LOWER | "-" | ";" | "."
SPACE        = <ASCII space, %x20>;
SEP          = (CR LF) | LF
CR           = <ASCII CR, carriage return, %x0D>;
LF           = <ASCII LF, line feed, %x0A>;
```

```
DIGIT              = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" |
                     "8" | "9"

UPPER              = "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" |
                     "I" | "J" | "K" | "L" | "M" | "N" | "O" | "P" |
                     "Q" | "R" | "S" | "T" | "U" | "V" | "W" | "X" |
                     "Y" | "Z"
LOWER              = "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" |
                     "i" | "j" | "k" | "l" | "m" | "n" | "o" | "p" |
                     "q" | "r" | "s" | "t" | "u" | "v" | "w" | "x" |
                     "y" | "z"

US-ASCII-SAFE      = %x01-09 / %x0B-0C / %x0E-7F
                     ;; US-ASCII except CR, LF, NUL
UTF8               = US-ASCII-SAFE / UTF8-1 / UTF8-2 / UTF8-3
                                   / UTF8-4 / UTF8-5
UTF8-CONT          = %x80-BF
UTF8-1             = %xC0-DF UTF8-CONT
UTF8-2             = %xE0-EF 2UTF8-CONT
UTF8-3             = %xF0-F7 3UTF8-CONT
UTF8-4             = %xF8-FB 4UTF8-CONT
UTF8-5             = %xFC-FD 5UTF8-CONT
```

The set of characters allowed to appear in the attr-name field is
limited to the set of characters used in LDAP and WHOIS++ attribute
names. For other services that have attribute name character sets
that are larger than these, those services should create a profile
that maps the names onto object identifiers, and the sequence of
digits and periods is used by those services in creating the attr-
name fields for their Tagged Index Objects.

It is worth mentioning that updates to a index based in tagged index
objects MUST be performed in the order specified by the tagged index
object itself.

4.3.1. Header Descriptions

The header section consists of one or more "header lines". The
following header lines are defined:

    "version": This line must always be present, and have the value
    "x-tagged-index-1" for this version of the specification.

    "updatetype": This line must always be present. It takes as the
    value either "total" or "incremental". The first update sent by a
    supplier server to a consumer server for a DSI must be a "total"
    update.

"thisupdate": This line must always be present. The value is the
number of seconds from 00:00:00 UTC January 1, 1970 at which the
supplier constructed this update.

"lastupdate": This line must be present if the "updatetype" list
has the value "incremental". The value is the number of seconds
from 00:00:00 UTC January 1, 1970 at which the supplier
constructed the previous update sent to the consumer. This field
allows the consumer to determine if a previous update was missed

"contextsize": This line may be present at the supplier's option.
The value is a number, which is the approximate total number of
entries in the subtree. This information is provided for
statistical purposes only.

4.3.2. Tokenization Types

The Tagged Index Object inherits the "TOKEN" scheme for tokenization
as specified in [2]. In addition, there are several other
tokenization schemes defined for the Tagged Index Object.
 The following table presents these schemes and what character(s) are
used to delimit tokens.

Token Type	Tokenization Characters
FULL	none
TOKEN	white space, "@"
RFC822	white space, ".", "@"
UUCP	white space, "!"
DNS	any character note a number, letter, or "-"

4.3.3. Tag Conventions

In the tag list, multiple consecutive tags may be shortened by using
"#-#". For example, the list "3,4,5,6,7,8,9,10" may be shortened to
"3-10". Tags are to be applied to the data on a per entry level.
Thus, if two index lines in the same index object contain the same
tag, then those two lines always refer to the same "record" in the
directory. In LDAP terminology, the two lines would refer to the
same directory object. Additionally if two index lines in the same
index object contain different tags, then it is always the case that
those two lines refer back to different records in the directory. The
meaning of '*' in the tag position is that that specific token apears
in every record in the directory.

The tag applied to the same underlying record in two separate
transmissions of a full-index may be different. Thus, receiving
index servers should make no assumptions about the values of the tags
across index object boundaries.

4.4. Incremental Indexing

   The tagged index object format supports the ability of information
   servers to distribute only delta index data, rather than distributing
   total index information each time. This scenario, known as
   incremental indexing supports three basic types of operations: add,
   delete and replace. If the incremental updatetype is specified in
   the tagged index object, then the index object contains a snapshot of
   only the changes that have been made since the index object specified
   in the lastupdate header was distributed. If the receiving index
   server did not receive that index object, it should request a total
   index object. If the CIP protocol supports it, the index server may
   request the specific index object that it missed.

   If the tagged index object contains an Add Block, then the lines in
   the Add Block refer to new records that were added to the information
   base of the transmitting index server. It can be guaranteed that
   those records did not exist in any previously received tagged index
   object, and the receiving index server can insert this index
   information in the index that it already maintains for the
   transmitting index server.

   If the tagged index object contains a Delete Block, then the
   structure of the Delete Block depends on how the consistency is
   maintained;

   - "completeRecord": all the tokens connected to the record to be
      deleted has to be included, the tag used to connect tokens in this
      message has no relation to tags used in previously sent tagged
      index objects.

   - "uniqueIDBased": only the unique identifier has to be defined.

   - "tagBased": all the tokens connected to the record has to be
      included but then preceded by the tag used for this specific
      record in the preceding set of the last full update and the there
      on following incremental updates.

   If the tagged index object contains an Update Block, then the lines
   in the Update Block refer to records that were changed in the
   information base of the transmitting index server. Again the specific
   content of the block depends on how the consistency is maintained.

   - "completeRecord": All the tokens representing the old version of
      the record as well as the new ones has to be included.

   - "uniqueIDBased": The unique ID has to be included together with
      the tokens that have changed.

       – "tagBased": Only the changed tokens are included, but then both
          the old version, if there was one, as well as the new one, if
          there is one.

    The Update Block also supports the idea of indexing new attributes
    that were not previously included in the tagged index object. For
    example, if the transmitting index server began including index
    information on postal addresses, then it could include an Update
    Block in the index object that included all the index information on
    postal addresses for all records in its information base, and
    indicate that nothing else has changed.

5. Examples

    In the following sections, for each different consistencybase type,
    the tagged index object is represented for the following scenario;
    The examples starts with one full update and following that a set of
    updates. The underlying information is presented in the LDIF [6]
    format.

5.1 The original database

    dn: cn=Barbara Jensen, ou=Product Development, o=Ace Industry, c=US
    objectclass: top
    objectclass: person
    objectclass: organizationalPerson
    cn: Barbara Jensen
    cn: Barbara J Jensen
    cn: Babs Jensen
    sn: Jensen
    uid: bjensen
    dn: cn=Bjorn Jensen, ou=Accounting, o=Ace Industry, c=US
    objectclass: top
    objectclass: person
    objectclass: organizationalPerson
    cn: Bjorn Jensen
    sn: Jensen
    title: Accounting manager
    dn: cn=Gern Jensen, ou=Product Testing, o=Ace Industry, c=US
    objectclass: top
    objectclass: person
    objectclass: organizationalPerson
    cn: Gern Jensen
    cn: Gern O Jensen
    sn: Jensen
    title: testpilot
    dn: cn=Horatio Jensen, ou=Product Testing, o=Ace Industry, c=US
    objectclass: top

```
   objectclass: person
   objectclass: organizationalPerson
   cn: Horatio Jensen
   cn: Horatio N Jensen
   sn: Jensen
   title: testpilot
```

5.1.1 "Complete" consistency based full update

```
   version: x-tagged-index-1
   updatetype: total
   thisupdate: 855938804
   BEGIN IO-Schema
   cn: TOKEN
   sn: FULL
   title: TOKEN
   END IO-Schema
   BEGIN Index-Info
   cn: 1/Barbara
   -1/J
   -1/Babs
   -*/Jensen
   -2/Bjorn
   -3/Gern
   -3/O
   -4/Horatio
   -4/N
   sn: */Jensen
   title: 1/product
   -1-2/manager
   -1/accounting
   -3,4/testpilot
   END Index-Info
```

5.1.2 "tag" consistency based full update

```
   version: x-tagged-index-1
   updatetype: total
   thisupdate: 855938804
   BEGIN IO-Schema
   cn: TOKEN
   sn: FULL
   title: TOKEN
   END IO-Schema
   BEGIN Index-Info
   cn: 1/Barbara
   -1/J
   -1/Babs
```

```
   -*/Jensen
   -2/Bjorn
   -3/Gern
   -3/O
   -4/Horatio
   -4/N
   sn: */Jensen

   title: 1/product
   -1-2/manager
   -1/accounting
   -3,4/testpilot
   END Index-Info
```

5.1.3 "unique" consistency based full update

```
   version: x-tagged-index-1
   updatetype: total
   thisupdate: 855938804
   BEGIN IO-Schema
   dn: FULL
   cn: TOKEN
   sn: FULL
   title: TOKEN
   END IO-Schema
   BEGIN Index-Info
   dn: 1/cn=Barbara Jensen, ou=Product Development, o=Ace Industry, c=US
   -2/cn=Bjorn Jensen, ou=Accounting, o=Ace Industry, c=US
   -3/cn=Gern Jensen, ou=Product Testing, o=Ace Industry, c=US
   -4/cn=Horatio Jensen, ou=Product Testing, o=Ace Industry, c=US
   cn: 1/Barbara
   -1/J
   -1/Babs
   -*/Jensen
   -2/Bjorn
   -3/Gern
   -3/O
   -4/Horatio
   -4/N
   sn: */Jensen
   title: 1/product
   -1-2/manager
   -1/accounting
   -3,4/testpilot
   END Index-Info
```

5.2 First update

   Gern Jensen's entry above changes to:

   dn: cn=Gern Jensen, ou=Product Testing, o=Ace Industry, c=US
   objectclass: top
   objectclass: person
   objectclass: organizationalPerson
   cn: Gern Jensen
   cn: Gern O Jensen
   sn: Jensen
   title: chiefpilot

5.2.1 First update using "complete"

   version: x-tagged-index-1
   updatetype: incremental
   lastupdate: 855940000
   thisupdate: 855938804
   BEGIN IO-schema
   cn: TOKEN
   sn: FULL
   title: FULL
   END IO-Schema
   BEGIN Update Block
   BEGIN Old
   cn: 1/Gern
   cn: 1/O
   cn: 1/Jensen
   sn: 1/Jensen
   title: 1/testpilot
   END Old
   BEGIN New
   cn: 1/Gern
   cn: 1/O
   cn: 1/Jensen
   sn: 1/Jensen
   title: 1/chiefpilot
   END New
   END Update Block

5.2.2 First update using "tag" consistency

```
version: x-tagged-index-1
updatetype: incremental
lastupdate: 855940000
thisupdate: 855938804
BEGIN IO-schema
cn: TOKEN
sn: FULL
title: FULL
END IO-Schema
BEGIN Update Block
BEGIN Old
title: 3/testpilot
END Old
BEGIN New
title: 3/chiefpilot
END New
END Update Block
```

5.2.3 First update using "unique" ID's

```
version: x-tagged-index-1
updatetype: incremental
lastupdate: 855940000
thisupdate: 855938804
BEGIN IO-schema
cn: TOKEN
sn: FULL
title: FULL
END IO-Schema
BEGIN Update Block
BEGIN Old
dn: 1/cn=Gern Jensen, ou=Product Testing, o=Ace Industry, c=US
title: 1/testpilot
END Old
BEGIN New
dn: 1/cn=Gern Jensen, ou=Product Testing, o=Ace Industry, c=US
title: 1/chiefpilot
END New
END Update Block
```

5.3 Second update

```
# Add a new entry
dn: cn=Bo Didley, ou=Marketing, o=Ace Industry, c=US
changetype: add
objectclass: top
objectclass: person
objectclass: organizationalPerson
cn: Bo Didley
sn: Didley
title: Policy Maker
# Delete an existing entry
dn: cn=Bjorn Jensen, ou=Accounting, o=Ace Industry, c=US
changetype: delete
# Modify all other entries: adding an additional locality value
dn: cn=Barbara Jensen, ou=Product Development, o=Ace Industry, c=US
changetype: modify
add: locality
locality: New Jersey
dn: cn=Gern Jensen, ou=Product Testing, o=Ace Industry, c=US
changetype: modify
add: locality
locality: New Orleans
dn: cn=Horatio Jensen, ou=Product Testing, o=Ace Industry, c=US
changetype: modify
add: locality
locality: New Caledonia
```

5.3.1 "complete"

```
version: x-tagged-index-1
updatetype: incremental
lastupdate: 855938804
thisupdate: 855939525
BEGIN IO-schema
cn: TOKEN
sn: FULL
title: FULL
locality: TOKEN
END IO-Schema
BEGIN Add Block
cn: 1/Bo
-1/Didley
sn: 1/Didley
title: 1/Policy
-1/maker
locality: 1/New
-1/York
```

```
   END Add Block
   BEGIN Delete Block
   cn: 1/Bjorn
   -1/Jensen
   sn: 1/Jensen
   title: 1/Accounting
   -1/Manager
   END Delete Block
   BEGIN Update Block
   BEGIN Old
   cn: 1/Barbara
   -1/J
   -1-3/Jensen
   -2/Gern
   -2/O
   -3/Horatio
   sn: 1-3/Jensen
   title: 1/Production
   -1/Manager
   -2/Testpilot
   -3/Chiefpilot
   END Old
   BEGIN New
   cn: 1/Barbara
   -1/J
   -1-3/Jensen
   -2/Gern
   -2/O
   -3/Horatio

   sn: 1-3/Jensen
   title: 1/Production
   -1/Manager
   -2/Testpilot
   -3/Chiefpilot
   locality: 1/Jersey
   -2/Orleans
   -3/Caledonia
   -1-3/New
   END New     END Update Block
```

5.3.2 "tag"

```
   version: x-tagged-index-1
   updatetype: incremental
   lastupdate: 855938804
   thisupdate: 855939525
   BEGIN IO-schema
```

```
   cn: TOKEN
   sn: FULL
   title: FULL
   locality: TOKEN
   END IO-Schema
   BEGIN Add Block
   cn: 5/Bo
   -5/Didley
   sn: 5/Didley
   title: 5/Policy
   -5/maker
   locality: 5/New
   -5/York
   END Add Block
   BEGIN Delete Block
   cn: 2/Bjorn
   -2/Jensen
   sn: 2/Jensen
   title: 2/Accounting
   -2/Manager
   END Delete Block
   BEGIN Update Block
   BEGIN New
   locality: 1/Jersey
   -2/Orleans
   -4/Caledonia
   -1,2,4/New
   END New
   END Update Block
```

5.3.3 "unique"

```
   version: x-tagged-index-1
   updatetype: incremental
   lastupdate: 855938804
   thisupdate: 855939525
   BEGIN IO-schema
   cn: TOKEN
   sn: FULL
   title: FULL
   locality: TOKEN
   END IO-Schema
   BEGIN Add Block
   dn: 1/cn=Bo Didley, ou=Marketing, o=Ace Industry, c=US
   cn: 1/Bo
   -1/Didley
   sn: 1/Didley
   title: 1/Policy
```

```
     -1/maker
     locality: 1/New
     -1/York
     END Add Block
     BEGIN Delete Block
     dn: 1/cn=Bjorn Jensen, ou=Accounting, o=Ace Industry, c=US
     END Delete Block
     BEGIN Update Block
     BEGIN New
     dn: 1/cn=Barbara Jensen, ou=Product Development, o=Ace Industry, c=US
     -2/cn=Gern Jensen, ou=Product Testing, o=Ace Industry, c=US
     -3/cn=Horatio Jensen, ou=Product Testing, o=Ace Industry, c=US
     locality: 1/Jersey
     -2/Orleans
     -3/Caledonia
     -1-3/New
     END New
     END Update Block
```

6. Aggregation

6.1. Aggregation of Tagged Index Objects

   Aggregation of two tagged index objects is done by merging the two
   lists of values and rewriting each tag list. The tag list rewriting
   process is done so that the resulting index object appears as if it
   came from a single source. An index server that aggregates tagged
   index objects for export MUST ensure that the export URL (i.e. the
   base-uri of the CIP object) for the aggregate index object will route
   all queries that have "hits" on the index object to that server
   (otherwise, query routing will not succeed).

7. Security Considerations

   This specification provides a protocol for transferring information
   between two servers. The information transferred may be protected by
   laws in many countries, so care must be taken in the methods used to
   tokenize the data to ensure that protected data may not be
   reconstructed in full by the receiving server. This protocol does
   not have any inherent protection against spoofing or eavesdropping.
   However, since this protocol is transported in MIME messages (as are
   all CIP index objects), it inherits all the security capabilities and
   liabilities of other MIME messages. Specifically, those wanting to
   prevent eavesdropping or spoofing may use some of the various
   techniques for signing and encrypting MIME messages.

   Information Server administrators must decide what portions of their
   databases are appropriate for inclusion in the Tagged Index Object.

For distribution of information outside the enterprise, information
server developers are encouraged to allow for facilities that hide
the organizational structure when generating the Tagged Index Object
from the underlying information database. To allow for the secure
transmission of Tagged Index Objects across the Internet, Index
Servers should make use of SSL when completing the connection. In
order to strongly verify the identity of the peer index server on the
other side of the connection, SSL version 3 certificate exchange
should be implemented, and the identity in the peer's certificate
verify with the Public Key Infrastructure. If electronic mail is
used to exchange the Tagged Index Objects, then a secure messaging
facility, such as PGP/MIME or S/MIME should be used to sign or
encrypt (or both) the information.

8. References

[1]   Allen, J. and M. Mealling, "The Architecture of the Common
      Indexing Protocol (CIP)," RFC 2651, August 1999.

[2]   Weider, C., Fullton, J. and S. Spero, "Architecture of the
      Whois++ Index Service", RFC 1913, February 1996.

[3]   Wahl, M., Howes, T. and S. Kille, "Lightweight Directory Access
      Protocol (v3)", RFC 2251, December 1997.

[4]   ITU, "X.525 Information Technology - Open Systems
      Interconnection - The Directory: Replication", November 1993.

[5]   "FORTEZZA Application Implementors Guide for the FORTEZZA
      Crypto Card (Production Version)", Document #PD4002102-1.01,
      SPYRUS, 1995.

[6]   Good, G., "The LDAP Data Interchange Format (LDIF) - Technical
      Specification", Work in Progress.

[7]   Hedberg, R., "LDAPv2 Client vs. the Index Mesh", RFC 2657,
      August 1999.

[8]   Howes, T. and M. Smith, "The LDAP URL Format", RFC 2255,
      December 1997.

[9]   Elkins, M., "MIME Security with Pretty Good Privacy (PGP)", RFC
      2015, October 1996.

[10] Ramsdell, B., Editor, "S/MIME Version 3 Message Specification",
      RFC 2633, June 1999.

   [11] Allen, C. and T. Dierks, "The TLS Protocol Version 1.0", RFC
        2246, January 1999.

9. Authors' Addresses

   Roland Hedberg
   Catalogix
   Dalsveien 53
   0387 Oslo
   Norway

   EMail: roland@catalogix.ac.se

   Bruce Greenblatt
   Directory Tools and Application Services, Inc.
   6841 Heaton Moor Drive
   San Jose, CA 95119
   USA

   Phone: +1-408-224-5349
   EMail: bgreenblatt@directory-applications.com

   Ryan Moats
   AT&T
   15621 Drexel Circle
   Omaha, NE 68135-2358
   USA

   Phone: +1 402 894-9456
   EMail: jayhawk@att.com

   Mark Wahl
   Innosoft International, Inc.
   8911 Capital of Texas Hwy, Suite 4140
   Austin, TX 78759
   USA

   Phone +1 626 919 3600
   EMail Mark.Wahl@innosoft.com

10. Full Copyright Statement