Network Working Group
INTERNET-DRAFT
<draft-ietf-find-cip-01.txt>

Chris Weider
CNIDR
May 1996

The Common Indexing Protocol

Status of this memo:

The author describes a protocol designed to provide common indexing
for the currently deployed Internet Directory Services.

Part 0: Differences between this and the WHOIS++ Indexing protocol

This protocol is designed to be more flexible than version 1.0 of the WHOIS++ Indexing protocol. The version number on this protocol is 2.0, as when this stabilizes it will become version 2.0 of the WHOIS++ Index protocol as well. It adds the ability to specify tokenization method for the strings generated for the centroids, weighting capabilities, and the ability to specify which original host a specific term was derived from. These capabilities should provide a much more robust indexing protocol.

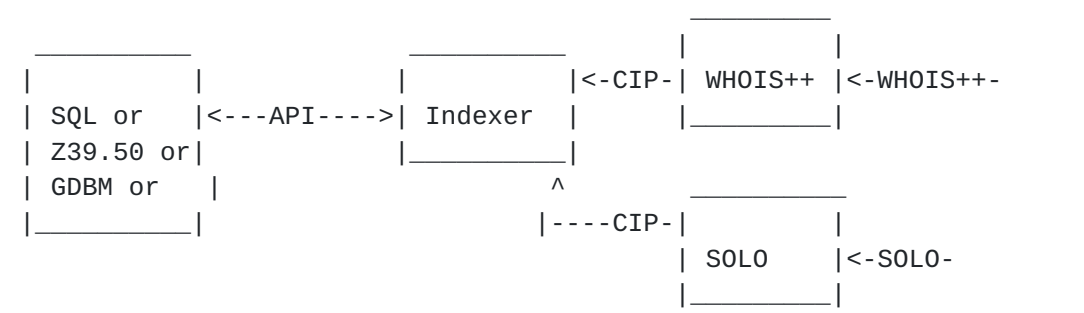
Part 1: Introduction

This protocol is designed to allow general indexing from most of the attribute-value based directory services. It is an extension of the current WHOIS++ indexing protocol. Also, this protocol is designed to be implemented in such a way as to allow many different types of protocols (X.500, WHOIS++, SOLO, LDAP, etc) to use a basic server which speaks all these protocols.

Part 2: Using the Indexing Architecture

The basic software architecture consists of APIs to back-end databases, an indexing 'object' which speaks the indexing protocol, and a variety of

interfaces for various query protocols.



Each Indexer participates in a forward knowledge mesh as shown below.

Part 3: Protocol Functionality and components of the Index Service

3.1 Base data servers

Most directory services today specify only the query language, the information model, and the server responses for their servers. Most also use a basic 'template-based' information model, in which each entry consists of a set of attribute-value pairs. Thus the basic service can be provided by a wide variety of databases and directory services. However, to participate in the Index Service, that underlying database must also be able to generate a 'centroid', or some other type of forward knowledge, for the data it serves.

Connections out from the indexing service to the base data servers will be accomplished using URLs for the various end protocols. This will avoid the need to rewrite the data from its native formats.

3.2. Centroids as forward knowledge

The centroid of a server is comprised of a list of the templates and attributes used by that server, and a word list for each attribute. The word list for a given attribute contains one occurrence of every word which appears at least once in that attribute in some record in that server's data, and nothing else.

For example, if a server contains exactly three records, as follows:

```
Record 1                        Record 2
Template: User                  Template: User
First Name: John                First Name: Joe
Last Name: Smith                Last Name: Smith
Favourite Drink: Labatt Beer    Favourite Drink: Molson Beer

Record 3
Template: Domain
Domain Name: foo.edu
```

Contact Name: Mike Foobar

the centroid for this server would be

Template: User
First Name: Joe
-John
Last Name: Smith
Favourite Drink: Beer
-Labatt
-Molson

Template: Domain
Domain Name: foo.edu
Contact Name: Mike
-Foobar

It is this information which is handed up the tree to provide forward knowledge. As we mention above, this may not turn out to be the ideal solution for forward knowledge, and we suspect that there may be a number of different sets of forward knowledge used in the Index Service. However, the indexing architecture is in a very real sense independent of what types of forward knowledge are handed around, and it is entirely possible to build a unified directory which uses many types of forward knowledge.

3.3 Other types of forward information

There are several other types of forward information that might be useful in an indexing service. The first is untokenized values for the given attributes, as opposed to the tokenized values given in the centroid. A second type is forward information generated by a typical query; this can be used for replication of databases or of specific records in a database. A third type is forward information which specifies from which server a given value was obtained. All of these are given in the protocol.

3.4. Index servers and Index server Architecture

A index server collects and collates the centroids (or other forward knowledge) of either a number of base servers or of a number of other index servers. An index server must be able to generate a centroid for the information it contains. In addition, an index server can index any other server it wishes, which allows one base level server (or index server) to participate in many hierarchies in the directory mesh.

3.4.1. Queries to index servers

An index server will take a query in standard 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 (referral in the X.500 model). An index server can also contain primary data of its own;

and thus act a both an index server and a base level server. In this case, the index server's response to a query may be a mix of records and referral pointers.

Protocol-specific use of an index server (for example, issuing a WHOIS++ query to the index service) will require a protocol-specific front end to the index server, which is responsible for translating the query and formatting the output as required.

3.4.2. Index server distribution model and forward knowledge propogation

The diagram on the next page illustrates how a mesh of index servers might be created for a set of base servers. Although it looks like a hierarchy, the protocols allow (for example) server A to be indexed by both server D and by server H.

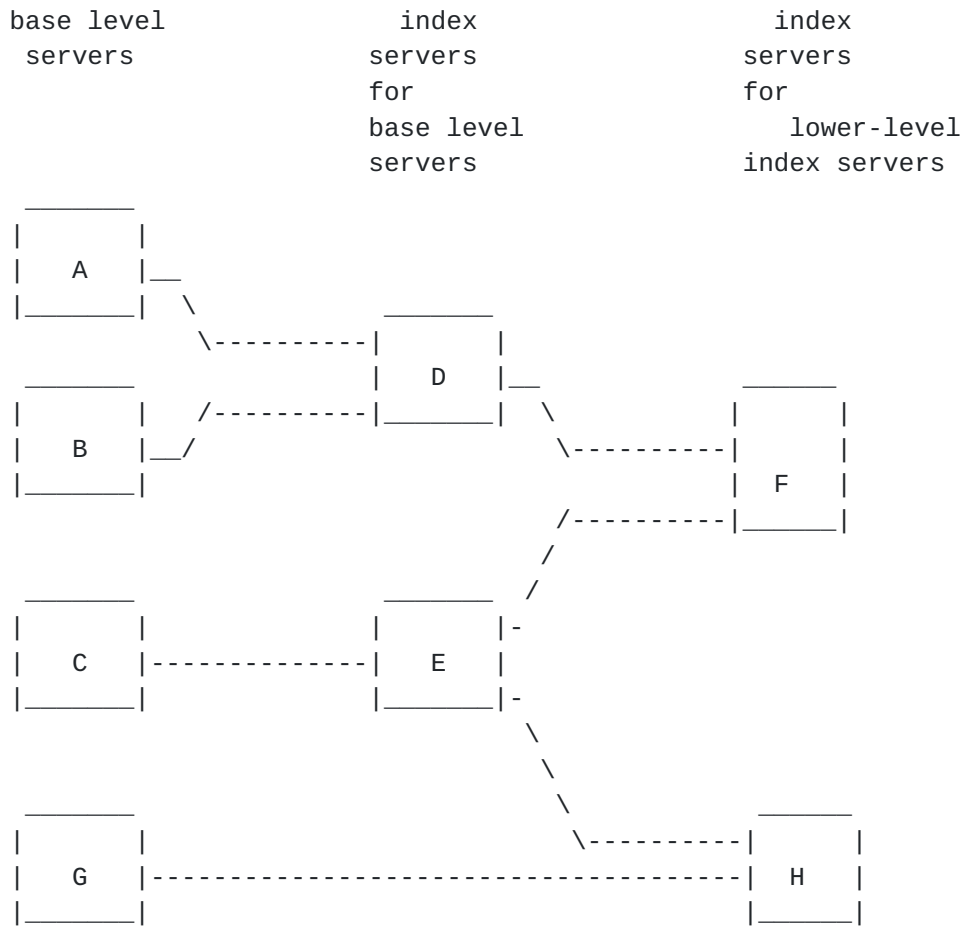


Figure 1: Sample layout of the Index Service mesh

In the portion of the index tree shown above, base servers A and B hand their centroids up to index server D, base server C hands its centroid up

to index server E, and index servers D and E hand their centroids up to index server F. Servers E and G also hand their centroids up to H.

The number of levels of index servers, and the number of index servers at each level, will depend on the number of base servers deployed, and the response time of individual layers of the server tree. These numbers will have to be determined in the field.

3.4.3. Forward knowledge propogation and changes to forward knowledge

Forward knowledge propogation is initiated by an authenticated POLL command (sec. 4.2). The format of the POLL command allows the poller to request the forward knowledge of any or all templates and attributes held by the polled server. After the polled server has authenticated the poller, it determines which of the requested forward knowledge the poller is allowed to request, and then issues a CENTROID-CHANGES report (sec. 4.3) to transmit the data. When the poller receives the CENTROID-CHANGES report, it can authenticate the pollee to determine whether to add the new changes to its data. Additionally, if a given pollee knows what pollers hold forward knowledge from the pollee, it can
signal to those pollers the fact that its information has changed by issuing a DATA-CHANGED command. The poller can then determine if and when to issue a new POLL request to get the updated information. The DATA-CHANGED command is included in this protocol to allow 'interactive' updating of critical information.

3.4.4. Forward knowledge propogation and mesh traversal

When an index server issues a POLL request, it may indicate to the polled server what relationship it has to the polled. This information can be used to help traverse the directory mesh. Two fields are specified in the current proposal to transmit the relationship information, although it is expected that richer relationship information will be shared in future revisions of this protocol.

One field used for this information is the Hierarchy field, and can take on three values. The first is 'topology', which indicates that the indexing server is at a higher level in the network topology (e.g. indexes the whole regional ISP). The second is 'geographical', which indicates that the polling server covers a geographical area subsuming the pollee. The third is 'administrative', which indicates that the indexing server covers an administrative domain subsuming the pollee.

The second field used for this information is the Description field, which contains the DESCRIBE record of the polling server. This allows users to obtain richer metainformation for the directory mesh, enabling them to expand queries more effectively.

3.4.5 Loop control

Since there are no a priori restrictions on which servers may poll which other

servers, and since a given server may participate in many sub-meshes, mechanisms must be installed to allow the detection of cycles in the polling relationships. This is accomplished in the current protocol by including a hop-count on polling relationships. Each time a polled server generates forward information, it informs the polling server about its current hopcount, which is the maximum of the hopcounts of all the servers it polls, plus 1. A base level server (one which polls no other servers) will have a hopcount of **0. When a server decides to poll a new server, if its hopcount goes up, then** it must information all the other servers which poll it about its new hopcount. A maximum hopcount (8 in the current version) will help the servers detect polling loops.

A second approach to loop detection is to do all the work in the client; which would determine which new referrals have already appeared in the referral list, and then simply iterate the referral process until there are no new servers to ask. An algorithm to accomplish this in WHOIS++ is detailed in [Faltstrom 95].

3.4.6. Query handling and passing algorithms

When an index server receives a query, it searches its collection of forward knowledge and determines which servers hold records which may fill that query. As this service becomes widely deployed, it is expected that some index servers may specialize in indexing certain template types or perhaps even certain fields within those templates. If an index server obtains a match with the query _for those template fields and attributes the server indexes_, it is to be considered a match for the purpose of forwarding the query.

3.4.7. Query referral

Query referral is the process of informing a client which servers to contact next to resolve a query. The syntax for notifying a client is outlined in section 4.5. A query can specify the 'trace' option, which causes each server which receives the query to send its server handle and an identification string to the client.

3.5 Security considerations

In the opinion of this author, until a generally accepted Internet wide security service is available (or until a web of such services reaches into most of the Internet) all security applied to queries should be done by the base server. Propogating this information through the common index mesh will run immediately into the problems of common authentication, access control, and incommensurable security features. Thus any index information propogated to this service should be considered unsecured. Server to Server authentication is provided, however.

4: Integrating disparate services

4.1 The service model

The basic service model uses a common data model, a common set of schema,

and allows the use of different access protocols to access a CIP server. A large number of additions to the 2.0 protocol are made to do this work.

4.1 Integration of data models

The basic data model for most of the existing directory services is essentially the same, a set of templates or object classes which are composed of attribute value pairs. Therefore integration of the data models should not prove too difficult.

4.2 Integration of schema

The various protocols use different attribute names for attributes which typically contain the same data. Since the common indexing protocol must provide a uniform index for these protocols, the attributes from the various protocols must be normalized. The protocol achieves this by mapping the various schema into a single attribute set. The CIP server is responsible for creating the URLs necessary to search the next server in the mesh.

4.3 Using different protocols to access the CIP service

As this document is presently constituted, one can use many protocols to access a CIP server, but the client must use the CIP attribute names when speaking to a CIP server. This issue will be debated on the mailing list.

5: Protocol Specification for the Index Service:

The syntax for each protocol component is listed below. In addition, each section contains a listing of which of these attributes is required and optional for each of the components. All timestamps must be in the format YYYYMMDDHHMM+ZZZZ or YYYYMMDDHHMM-ZZZZ, using a 24 hour clock, where ZZZZ indicates the difference between the local clock and GMT.

5.1. Data changed syntax

The data changed template look like this:

```
# DATA-CHANGED
 Version-number: // version number of index service software, used to insure
                 // compatibility. Current value is 2.0
 Time-of-latest-centroid-change: // time stamp of latest forward information
                                 // change,GMT
 Time-of-message-generation: // time when this message was generated, GMT
 DSI: // Data set identifier. This uniquely identifies a given data set in case
the
        // server manages multiple logical data sets
 Server-handle: // IANA unique identifier for this server
                // Or  Distinguished Name of the root of the subtree this
server
                // is responsible for.
 Host-Name: // Host name of this server (current name)
 Host-Port: // Port number of this server (current port)
```

```
 Protocol: // Access protocol to use when speaking to this server
 Best-time-to-poll: // For heavily used servers, this will identify when
                    // the server is likely to be lightly loaded
                    // so that response to the poll will be speedy, GMT
 Authentication-type: // Type of authentication used by server, or NONE
 Authentication-data: // data for authentication
# END // This line must be used to terminate the data changed message

Required/optional table

Version-Number  REQUIRED
Time-of-latest-centroid-change  REQUIRED
Time-of-message-generation      REQUIRED
DSI     OPTIONAL
Server-handle   REQUIRED
Host-Name       REQUIRED
Host-Port       REQUIRED
Protocol                REQUIRED
Best-time-to-poll       OPTIONAL
Authentication-type     OPTIONAL
Authentication-data     OPTIONAL
```

5.2. Polling syntax

```
# POLL:
 Version-number: // version number of poller's index software, used to
                 // insure compatibility
 Charset: // specifies character set in which the centroid changes are to be
        // transmitted. Must be one of ISO-8859-1 or UNICODE-1-1-UTF-8
 DSI: // Data set identifier. Indicates which data set of multiple data sets
        // should be indexed.
 DSI-Description: // Human readable string about the data set
  Type-of-poll: // type of forward data requested. CENTROID and QUERY
               // are the only one currently defined
 Poll-scope: // Selects bounds within which data will be returned. See note.
 Start-time: // give me all the centroid changes starting at this time, GMT
 End-time: // ending at this time, GMT
 Template: // a standard template or object class name, or the keyword ALL, for
a
           // full update.
 Field:    // used to limit centroid update information to specific fields,
           // is either a specific field name, a list of field names,
           // or the keyword ALL
 Starting-point: // location in the DIT or other hierarchical structure
        // to start the index. If used, it implies that the entire subtree is
        // indexed as well
 Server-handle: // IANA unique identifier for the polling server.
                // this handle may optionally be cached by the polled
                // server to announce future changes
 Host-Name: // Host name of the polling server.
 Host-Port: // Port number of the polling server.
```

```
 Relationship: // This field indicates the relationship which the poller
              // bears to the pollee. Typical values might include
              // 'Topology', 'Geographical", or "Administrative"
 Description: // This field contains the DESCRIBE record of the
                // polling server
 Tokenization-type: // The tokenization algorithm used
       // Can be one of: "TOKENS", "RECORDS" or "ATTRIBUTES".
       // Default is "TOKENS", which means space-delimited values.
 Options: // Can be used to request the WEIGHT, HANDLE, and/or HOST information
        // for the returned values
 Content-Transfer-Encoding: // What encoding type, standard values...
 Authentication-type: // Type of authentication used by poller, or NONE
 Authentication-data: // Data for authentication
# END  // This line must by used to terminate the poll message
```

For poll type CENTROID, the allowable values for Poll Scope are FULL and RELATIVE. Support of the FULL value is required, this provides a complete listing of the centroid or other forward information. RELATIVE indicates that these are the relative changes in the centroid since the last report to the polling server.

The allowable values for OPTION are WEIGHT, HANDLE, and HOST. Support for the HANDLE and HOST values are required. HANDLE indicates that each attribute value must be listed with the server handle of the server from which this value was obtained by the polled server; HOST indicates that each attribute value must be listed with the host name and port number of the server from which this value was obtained. WEIGHT is optional, and allows each value to be assigned a relative weight according to a defined and specified weighting scheme. This value is included for future clarification. Since a weighting scheme will need to be identified, WEIGHT will take additional scheme identifiers in a syntax to be determined.

For poll type QUERY, the allowable values for Poll Scope are a blank line, which indicates that all records are to be returned, or a valid WHOIS++ query, which indicates that just those records which satisfy the query are to be returned. N.B. Security considerations may require additional authentication for successful response to the Blank Line Poll Scope. This value has been included for server replication.

As there are different types of tokenization available in future versions of this protocol, and as there may well be organizations which wish to construct servers which each index the same set of servers, but wish to do it in slightly different fashions, the command POLLED-FOR can be used to determine all the servers the current server polls.

Required/Optional Table

Version-Number REQUIRED, value is 2.0

```
Charset         REQUIRED, values ISO-8859-1 and UNICODE-1-1-UTF-8 are required
DSI     OPTIONAL
DSI-Description OPTIONAL
Type-Of-Poll    REQUIRED, values CENTROID and QUERY are required
Poll-scope      REQUIRED  If Type-of-poll is CENTROID, FULL is required,
                          RELATIVE is optional
                          If Type-of-poll is QUERY, Blank line is required,
                          and WHOIS++-type queries are required
Start-time      OPTIONAL
End-Time        OPTIONAL
Template        REQUIRED
Field           REQUIRED
Starting-point OPTIONAL
Server-handle   REQUIRED
Host-Name       REQUIRED
Host-Port       REQUIRED
Hierarchy       OPTIONAL
Description     OPTIONAL
Tokenization-Type       REQUIRED, value TOKENS is required, RECORDS and
                                  ATTRIBUTES are optional
Options         OPTIONAL
Content-Tranfer-Encoding        OPTIONAL. If not given, default is 8-bit
                                Available values are 8-bit, Quoted-Printable,
                                and Base64
Authentication-Type:    OPTIONAL
Authentication-data:    OPTIONAL


Example of a POLL command:
# POLL:
 Version-number: 2.0
 Charset: UNICODE-1-1-UTF-8
 Type-of-poll: CENTROID
 Poll-scope: FULL
 Start-time: 199501281030+0100
 Template: ALL
 Field: ALL
 Server-handle: BUNYIP01
 Host-Name: services.bunyip.com
 Host-Port: 7070
 Hierarchy: Geographical
 Tokenization-type: TOKENS
# END
```

5.3. Centroid change report

As the centroid change report contains nested multiply-occuring blocks, each multiply occurring block is surrounded *in this paper* by curly braces '{', '}'. These curly braces are NOT part of the syntax, they are for identification purposes only.

The syntax of a Data: item is either a list of values (words or other phases, depending on the tokenization value), one value per line, with the syntax:

-word
+<weight> weight (if requested)
+<handle> server handle (if requested)
+<host> host (if requested)
+<port> port (if requested)

or the keyword:

  *

The weight, handle, host, and port are not required, but are expected to be used by advanced servers. The weight is the relative weight of the value for weighting servers, and the handle, host, and port values are to indicate from which host the polled server received the value. This allows a polling server to construct direct pointers to the servers lower in the mesh rather than adding an additional level of indirection.

The keyword ANY as the only item of a Data: list means that any value for this field should be treated as a hit by the indexing server.

The field Any-field: needs more explanation than can be given in the body of the syntax description below. It can take two values, True or False. If the value is True, the pollee is indicating that there are fields in this template which are not being exported to the polling server, but wishes to treat as a hit. Thus, when the polling server gets a query which has a term requesting a field not in this list for this template, the polling server will treat that term as a 'hit'. If the value is False, the pollee is indicating that there are no other fields for this template which should be treated as a hit. This field is required because the basic model for the WHOIS++ query syntax requires that the results of each search term be 'and'ed together. This field allows polled servers to export data only for non-sensitive fields, yet still get referrals of queries which contain sensitive terms.

The attributes used by the polled server must also be mapped into the attributes used by the CIP. This allows the specification of the creation of the URLs to refer to the search on the final server.

```
# CENTROID-CHANGES
 Version-number: // version number of pollee's index software, used to
                 // insure compatibility
 Character-set: // Specifies which character set the data is in. Allowable
values
                // are ISO-8859-1 and UNICODE-1-1-UTF-8
 Start-time: // change list starting time, GMT
 End-time: // change list ending time, GMT
 Server-handle: // IANA unique identifier of the responding server
```

```
 Hop-Count: // One more than the largest value the polled server has received
            // when polling other servers. If the polled server is a leaf ,
            // server, hop-count should be zero. The current maximum value
            // (Feb. 95) is 8.
 Options: // Which options the polled server was able to satisfy. Values are
        // WEIGHT, HANDLE, and HOST
 Authentication-type: // Type of authentication used by pollee, or NONE
 Authentication-data: // Data for authentication
 Compression-type: // Type of compression used on the data, or NONE
 Size-of-compressed-data: // size of compressed data if compression is used
 Protocol: // Protocol spoken by the polled server. Used to construct the URLs
                // for referrals. One of WHOIS++, LDAP, CCSO, CIP
 Operation: // One of 3 keywords: ADD, DELETE, FULL
            // ADD - add these entries to the centroid for this server
            // DELETE - delete these entries from the centroid of this
            // server
            // FULL - the full centroid as of end-time follows
 Tokenization-type: // The tokenization algorithm used
       // Can be one of: "TOKENS", "RECORDS" or "ATTRIBUTES".
       // Default is "TOKENS".
 Token: // Character(s) used in the tokenization algorithm
{ // The multiply occurring template block starts here
# BEGIN TEMPLATE
 Template: // a standard template name
        // May be repeated to indicate that the CIP template name
        // should map to all of these
 CIP-Template-Name: // What template name this maps to in the CIP
 Any-field: // TRUE or FALSE. See beginning of 6.3 for explanation.
 { // the template contains multiple field blocks
# BEGIN FIELD
 Field: // a field name within that template
        // May be repeated to indicate that the CIP attribute name should
        // map to all of these
 CIP-Field-Name: // The attribute name used by CIP
 Value-rewrite-method: // specifies how values should be rewritten for the
        // final query to the end server
 Content-Transfer-Encoding: // If existing, one of Base64 or Quoted-Printable
 Data: // Either the keyword *, or
       // the value list itself, one per line, cr/lf terminated,
       // each line starting with a dash character ('-').
       // Each value may be optionally followed by other lines containing
      // weight, handle, and/or host information, these other lines begin
     // with the plus sign (+) and a tag which states which information
     // is placed on this line. Allowable tags are <weight>, <handle>, <host>,
     // and <port>
# END FIELD
  } // the field ends with END FIELD
# END TEMPLATE
} // the template block ends with END TEMPLATE
# END CENTROID-CHANGES // This line must be used to terminate the centroid
                       // change report
```

For each template, all fields must be listed, or queries will not be referred correctly.

Required/Optional table

```
Version-number  REQUIRED, value is 2.0
Character-set REQUIRED, values of ISO-8859-1 and UNICODE-1-1-UTF-8 must be
                supported
Start-time      REQUIRED (even if the centroid type is FULL)
End-time        REQUIRED (even if the centroid type is FULL)
Server-handle   REQUIRED
Hop-Count       REQUIRED
Options         OPTIONAL If the polling server has requested options a polled
                         server is unable to satisfy, an error message will be
                         generated
Authentication-Type     OPTIONAL
Authentication-Data     OPTIONAL
Compression-type        OPTIONAL
Size-of-compressed-data OPTIONAL (even if compression is used)
Protocol                REQUIRED
Operation       REQUIRED, Support for all three values is required
Tokenization-type       REQUIRED
Token   OPTIONAL, if missing the default is blanks and newlines
# BEGIN TEMPLATE REQUIRED
Template        REQUIRED
CIP-Template-Name REQUIRED
Any-field       REQUIRED
# BEGIN FIELD    REQUIRED
Field           REQUIRED
CIP-Field-Name REQUIRED
Value-Rewrite-Method OPTIONAL
Content-Transfer-Encoding       OPTIONAL, same values as above
Data            REQUIRED
# END FIELD      REQUIRED
# END TEMPLATE   REQUIRED
# END CENTROID-CHANGES REQUIRED
```

Example:

```
# CENTROID-CHANGES
 Version-number: 2.0
 Charset: UNICODE-1-1-UTF-8
 Start-time: 197001010000+0100
 End-time: 199503012336+0100
 Server-handle: BUNYIP01
 Hop-Count: 3
 Tokenization-Type: TOKENS
# BEGIN TEMPLATE
 Template: USER
 CIP-Template-Name: USER
```

```
 Any-field: TRUE
# BEGIN FIELD
 Field: Name
 CIP-Field-Name: Name
 Data: Patrik
+<handle> NADA01
+<host> ui.nada.kth.se
+<port> 7070
-Faltstrom
+<handle> NADA01
+<host> ui.nada.kth.se
+<port> 7070
-Malin
-Linnerborg
# END FIELD
# BEGIN FIELD
 Field: Email
 CIP-Field-Name: Email
 Data: paf@bunyip.com
-malin.linnerborg@paf.se
# END FIELD
# END TEMPLATE
# END CENTROID-CHANGES
```

4.4 QUERY and POLLEES responses

The response to a QUERY command is done in WHOIS++ format.

4.5. Query referral

```
# SERVERS-TO-ASK
 Version-number: // version number of index software, used to insure
                 // compatibility
 Body-of-Query: // the original query goes here
 URL: // URL of the interaction required to issue this query to the next
        // server
 Priority: // Relative priority of this server among all replicas

# END SERVERS-TO-ASK
```

Required/Optional table

```
Version-number  REQUIRED, value should be 2.0
Body-of-query   OPTIONAL
URL     REQUIRED
Priority        OPTIONAL
```

5. Client-Server Interaction

Access can be made to a CIP server using most access protocols. As of this writing, the attribute and template names used in the query must be those

used by the CIP protocol itself.

6: Reply Codes

In addition to the reply codes listed in [Deutsch 95] for the basic WHOIS++ client/server interaction, the following reply codes are used by the Common Indexing Protocol.

113 Requested method not available	Unable to provide a requested tokenization, compression, or transfer encoding method. Contacted server will send requested data in different format.
114 Requested option not available	Unable to provide a requested option in CENTROID-CHANGES. No options have been used but raw data will be transmitted.
227 Update request acknowledged	A DATA-CHANGED transmission has been accepted and logged for further action.
503 Required attribute missing	A REQUIRED attribute is missing in an interaction.
504 Desired server unreachable	The desired server is unreachable.
505 Desired server unavailable	The desired server fails to respond to requests, but host is still reachable.

7: References

[Deutsch 95] Deutsch, P., Patrik Faltstrom, Rickard Schoultz, and Chris Weider, Architecture of the WHOIS++ Service, RFC 1835, Proposed Standard, March 1995

[Faltstrom 95] Faltstrom, Patrik, Rickard Schoultz, and Chris Weider, "How to interact with a WHOIS++ mesh", RFC 1914, Proposed Standard, November 1995