# The CCSO Nameserver – A Description

by
Steven Dorner   s–dorner@uiuc.edu
Computer and Communications Services Office
University of Illinois at Urbana

July 26, 1989

updated by
Paul Pomes   paul–pomes@uiuc.edu
Computer and Communications Services Office
University of Illinois at Urbana

August 2, 1992

## Introduction

This document provides an overview of the CCSO Nameserver. It should give the reader a good idea of the capabilities, implementation and performance of the Nameserver.

## Overview

The CCSO Nameserver is a computer resident "phone book". It can keep a relatively small amount of information about a relatively large number of people or things, and provide fast access to that information over the Internet.[1] Here at the University of Illinois, we keep the contents of the "white pages" of our *Student/Staff Directory* as well as other selected information, in the Nameserver.

Unlike a printed directory, the information in the CCSO Nameserver is dynamic. It can be updated at any time, from any computer on the Internet capable of running the "client" program, *ph*.[2] The Nameserver can also be taught to keep new **types** of information, such as electronic mail addresses or office hours, without recompilation or change to the existing database.

The remainder of this document will examine in somewhat further depth three aspects of the Nameserver; what it does (**Capabilities**), how it does them (**Implementation**), and how well it does them (**Performance**). There are in–depth documents describing some of these aspects of the Nameserver; the interested reader may refer to the **References** section for the titles of these other documents.

## Capabilities — The Database

The CCSO Nameserver manages a database that consists of many individual *entries*. Each entry contains one or more *fields*, each field consisting of a one or more printable ASCII characters (including tab and newline). Each field is associated with a particular *field description* that is used to specify the behavior of the field. A field description includes a name, a maximum length for the fields it describes, and certain

Converted to portable n/troff format using the -me macros from funky Next WriteNow format (icch).

[1] The collection of local, regional, and national networks using the TCP/IP protocols.

[2] At present this means 4.[23] BSD UNIX, VMS, VM/CMS, DOS, or Macintosh.

*properties* that determine how the field is used.

There are essentially no intrinsic limits on the size of the database, in number of entries, numbers of field descriptions, numbers of fields per entry, or sizes of fields.[3]

Certain fields[4] in the database are indexed. Words from these fields can be used as keys to select entries in the database. Words from any field may be used to refine the selection made by the key fields. The indexing scheme used is "double–hashing", and results in very fast lookups for key fields. The hash table is also indexed to facilitate pattern matching on the hash table (and hence the database).

## Capabilities — The Server

The database resides entirely on one computer and is managed by a server program, *qi* (query interpreter). Multiple instances of *qi* may be executing at any one time; access to the database is controlled by advisory locks. Any number of processes may read the database, unless a process is writing the database, in which case all processes must wait for that process to complete its work before beginning their own.

*Qi* uses a command–reply scheme like that used by FTP.[5] It accepts commands from its standard input, and writes replies on its standard output. Both commands and replies are couched in "netascii"; lines consisting of printable ASCII characters terminated with a newline (ASCII 10) or carriage–return newline (ASCII 13 ASCII 10) pair. Additionally, the backslash "\" is used to "escape" certain characters, as in the C programming language.[6]

Commands consist of a keyword optionally followed by one or more arguments or keywords. Commands include: **query** for querying the database; **change** for changing fields in entries; **add** for adding new entries. Replies consist of a numerical code ranging from −599 to 599, and additional text. The numerical codes may indicate an operation in progress (100–199), success (200–299), a request for further information (300–399), temporary failure (400–499), or permanent failure (500–599). Replies in the range from −599 to −100 indicate that further replies are to be expected for the current command; they otherwise have the same meanings as their positive counterparts.

The behavior of *qi* may be modified by use of certain *options*, accessed by the **set** command. The number of available options is small; the most important options are *echo*, which causes *qi* to print commands on its output before executing them, and *limit*, which allows the user to specify a maximum number of entries to which a command may be applied.

*Qi* operates in three different modes; anonymous, login, and hero. Each mode is more liberal than the previous in the operations it allows, and consequently more difficult to access. Anonymous mode is used to make queries of public information[7] and for a few other innocuous purposes. In anonymous mode, there is a maximum number of entries that can be viewed with one command; the purpose of this limitation is to discourage the use of the Nameserver for the preparation of mailing lists. Anonymous mode is used for most queries of the Nameserver.

To enter login mode, a user must identify himself as the owner of a particular Nameserver entry by giving an *alias* (login name) and a password.[8] In addition to the capabilities of anonymous mode, login mode

[3] Actually there are limits imposed by the 32-bit pointers used throughout the system. Before those limits could be reached, however, the database would likely be too large to manage.

[4] Those whose field descriptions in the *.cnf* file contain the property "Indexed".

[5] See RFC-959, *File Transfer Protocol (FTP)*, J. Postel and J. Reynolds.

[6] The set of such escapes is much more limited than in C; only "\n" for newline, "\t" for tab, "\"" for double-quote, and "\\" for backslash are allowed.

[7] I.e., to view fields whose field description contain the property "Public".

[8] Actually the user is asked to encrypt a string using his password, and *qi* compares the result returned with the result it obtained by encrypting the same string with the user's stored password. This is to provide additional security when running *qi* over networks; the user's password is never sent "in the clear" over a potentially insecure network.

allows the logged in user to change fields from his or her own entry in the Nameserver.[9]

Hero mode is entered either by entering login mode as a Nameserver "hero" (superuser) or by running *qi* directly from a terminal, rather than over a network. In this mode, all artificial limits are removed; the hero may change any field in any entry in the database, as well as view as many entries as he wishes. Hero mode is used mostly for administrative purposes.

## Capabilities — Queries

Since most of what the Nameserver does is answer queries, it is fitting to describe queries more fully here. A nameserver query consists of five elements; the "query" keyword, values for one or more indexed fields, values for zero or more non–indexed fields, optionally the "return" keyword, and optionally a list of fields to print from the selected entries. A couple of examples will clarify. First, a plain query; the arguments are interpreted as requests for words from the name or nickname fields, both of them indexed fields:

```
qi> query steven dorner
-200:1:         alias: s-dorner
-200:1:          name: dorner steven c.
-200:1:         email: dorner@garcon.cso.uiuc.edu
-200:1:         phone: (w) 244-1765
-200:1:       address: 181 DCL, MC 256
-200:1:              : 1201 W. Washington, C, 61821
-200:1:         title: res programmer
-200:1:      nickname: Steve
-200:1:         hours: 8-4 weekdays
200:Ok.
```

Here is an example that uses all five elements. The "department" field is not indexed.

```
qi> query dorner department=computing return name email department
-200:1:          name: dorner steven c.
-200:1:         email: dorner@garcon.cso.uiuc.edu
-200:1:    department: computing services office
200:Ok.
```

## Capabilities — The Client

Usually, the Nameserver is accessed via the "client" program *ph*. This program makes a connection to a copy of *qi* on the machine that keeps the Nameserver database. It then provides assistance to the user of the Nameserver; it formulates queries, formats Nameserver responses, and provides other conveniences.

*Ph* operates in two modes; command-line and interactive. In command-line mode, *ph* forms a Nameserver query from the arguments given it, sends it to *qi*, prints the result, and exits. In interactive mode, *ph* reads commands from the user, relays them to *qi*, and prints *qi*'s responses. The responses are automatically sent through a paging program. Some commands given to ph are expanded into more than one qi command. For example, the *ph* "edit" command first asks *qi* for the value of the desired field, puts that value in an editor where the user edits it as s/he pleases, and then issues a "change" command to change the field to its desired new value.

## Implementation — The Source

The Nameserver is written in C (a small parser is written in lex[10]), and runs on UNIX systems. The client,

[9] Actually a user may change only those fields whose field description contain the property "Change".

[10] See *Lex–A Lexical Analyzer Generator*, M.E. Lesk and E. Schmidt.

*ph*, may be run on 4.[23]BSD derived systems. A version of *ph* exists for VMS, DOS, Mac, and a limited version exists for VM/CMS systems.

There were at last count 320,000 bytes of C and lex source code; some 6,000 statements in 63 files. This source is divided into several distinct categories; *qi* (230,000 bytes, 28 files, 3500 statements), *ph* (46,000 bytes, 3 files, 700 statements), utilities (89,000 bytes, 21 files, 1700 statements), and libraries (19,500 bytes, 11 files, 300 statements).

The database and *qi* reside on a Digital Equipment Corporation VAXServer 3500 running Ultrix.

## Implementation — The Database

The database is kept in six files with the extensions *.dir*, *.dov*, *.idx*, *.iov*, *.seq*, and *.bdx*. The *.dir* and *.dov* files contain the actual data. The *.idx* and *.iov* files contain the hash table, with pointers into the data files. The *.seq* file contains all the words from the hash table, sorted alphabetically, along with pointers into the hash table; it is used for pattern-matching on the hash table. The *.bdx* file contains a tree of four-letter nodes, each node pointing to where entries with those four letters begin in the *.seq* file; the *.bdx* file speeds search of the *.seq* file.

The *.dir* file consists of a header and one fixed-length record for each entry in the database. If there is too much data for one record, the remainder is placed in the *.dov* file. The *.dov* file also consists of fixed-length records, and if one is not enough, the remainder can be placed in more *.dov* records. Thus, an entry is really a linked list of fixed-length records, and is not limited in size. It is relatively easy to play with the sizes of the *.dir* and *.dov* records (before compilation and installation of the database) for optimum performance. We use a fairly small record size in the *.dir* file, to minimize space wastage,[11] and a fairly large record size in the *.dov* file, to minimize linking. Most entries are wholly contained in the *.dir* file; most of the rest require only one *.dov* record.

Each entry begins with some fixed-length information, followed by the fields that make up its data. Each field is a null-terminated ASCII string. A field begins with an ASCII string that is the id of the field description for that field, and a colon. The field's data follows, and then the null terminator (ASCII 0). Tagging each field with its description number means that the database is not sensitive to the presence, absence or order of the fields. This in turn means that field descriptions can be added to the Nameserver at will, and the newly-defined fields used, without recompilation or rebuilding of the database (see **Implementation — Field Descriptions** below).

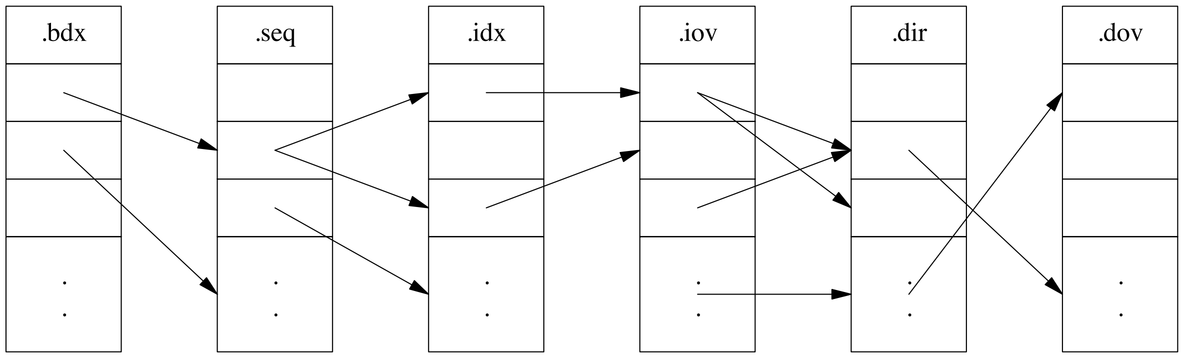

**Figure 1. Database Organization**

[11] Not entirely successfully – see **Performance — Database Size** below.

The *.idx* file is made up of a fixed number of fixed-length records. Each record that is in use contains a word from an indexed field, and a set of pointers to the *.dir* records that contain the word in an indexed field. Overflow in the *.idx* file is handled like overflow in the *.dir* file; the excess pointers are put in one or more fixed-length records in the *.iov* file. Words are indexed by computing a hash function. If the selected location is not empty but does not contain the desired word, the hash function is iterated, until a limit is reached (implying the failure of the index) or the word or an empty spot is found. If the spot is empty, the word and a pointer to the entry in which it occurs is placed in the record. If the spot is not empty, a pointer to the entry is appended to the list of pointers for that word.

The *.seq* file uses fixed-length records (called *leaves*) to keep a sorted list of all the words in the hash table (*.idx* and *.iov* files). Each leaf contains up to four words, and a pointer to the next leaf in alphabetical order. With each word is stored a pointer into the hash table where that word is found.

The *.bdx* file has records (called *nodes*) that contain one four-byte key, and two pointers; one to the previous node in alphabetical order, and one to the following node in alphabetical order. If a particular four-byte key happens to begin a leaf, that key's node will contain a pointer to that leaf instead of a pointer to another node.

## Implementation — Queries

An incoming query is first broken down into its component parts. Then, the selection arguments of the query are checked for indexed arguments. The longest indexed arguments[12] are looked up one by one in the hash table (or, if they contain pattern-matching characters, a search is made through the *.bdx* and *.seq* files for each pattern). The index entries are "anded" together to select only those entries that contain all of the indexed words.

Next, the selected entries are fetched one by one, and matched against the argument list. This is done for two reasons. First, the fact that an entry appears in the index for a word says nothing about which **field** the word is to be found in; it merely notes that the word does appear. Therefore, it is necessary to recheck indexed fields, and make sure the words in question appear in the proper fields. Second, the non-indexed words must be checked, to see that they appear in the proper fields in the entry.

If the entry passes the checks, the selected fields (or a set of default fields) are printed.

## Implementation — Field Descriptions

Field descriptions are kept in a file that *qi* reads each time it is run. This file consists of lines describing each field, in ASCII, with colons separating the elements in a line. First comes the id number of the field, then the name of the field and its maximum length. Finally, there is a colon-separated list of properties for the field.

Since this file is read each time *qi* starts up, lines can be added to define new fields at will. All subsequent invocations of *qi* will be able to recognize and use the fields.

The major properties fields may have are Indexed, Public, Default, Lookup, and Change. Fields marked Indexed are kept track of in the database's index. At least one such field **must** be included in every query. Fields marked Public may be viewed by anyone using *qi* in anonymous or login mode. Fields not marked Public may only be viewed by the entry's owner in login mode, or by someone using *qi* in hero mode. Default fields are printed if no "return" clause is given in a query. Lookup fields may be used in the selection part of a query; a field not marked Lookup cannot be used to select entries.[13] Finally, a user in login mode in *qi* may change any of his or her fields that are marked Change.

[12] Actually, the longest indexed arguments free of pattern-matching metacharacters. Pattern matches take much longer than normal index lookups since the *.bdx* and *.seq* files must be searched, and since such searches frequently result in a large number of matches being selected.

[13] You might decide, for example, that no one should be allowed to be found by his or her phone number. You could mark the phone number field as Public (so it could be viewed) but not Lookup (so no one could use it in searches).

## Performance — Database Size

Our database contains 80,140 entries, totalling 16 megabytes of information. The *.dir* and *.dov* files together are 33 megabytes; nearly half the space is wasted. This percentage could be reduced by reducing the record size of the *.dir* file.

The hash table, which has room for 450,001 words, actually contains 157,324 words and 270,784 pointers, for a total of 1.3 megabytes of hash table. The *.idx* and *.iov* files are 19.5 megabytes in size; even allowing for a large number of empty hash table slots (necessary for performance), most of the space is wasted. As with the *.dir* file, reducing the record size in the *.idx* file would help the situation.

Rounding out the database is 7.2 megabytes in the *.bdx* and *.seq* files.

## Performance — Speed

To test speed, we took 300 words from different parts of the index, and looked each one up using *qi*. *Qi* found 396 entries in 78 seconds; that is about ¼ second per lookup. Using four letter keys and wildcarding the rest, *qi* found 9213 entries in 460 seconds, for about 1½ seconds per lookup.

In actual use over a network, response is slower, since the client program must establish a connection with the host that has the database. Looking up 100 indexed words in separate invocations of *ph* took 109 seconds, or 1 second per lookup; 118 entries were found.

## Performance — Usage

In a recent week, typical of most weeks, we had 3100 uses from over 70 campus machines.[14] By far most of the commands given were queries (3643). There were also 175 logins, 264 changes, and a few hundred other commands issued. Of the commands: 58% were successful; 26% were queries that found no entries; 8% were queries that found too many entries; 4% were other errors; 3% were rejected because they required login mode, but were being given in anonymous mode; and 1% failed due to command syntax errors.

## Further Directions

Overall, we are fairly satisfied with what has been done to date. Ongoing efforts will be centered around making the Nameserver convenient to use in a distributed environment. This will primarily involve allowing users to specify a server, although some peripheral issues are also in need of resolution.

Additionally, we will make some attempts to remove wasted space from the database and its associated index; this is not a high priority since the database, for all its wasted space, is still not unmanageably large.

## Distribution

The CCSO Nameserver is Copyright © 1988 by the University of Illinois Board of Trustees. Portions of the software are Copyright © 1985 by CSNet. It is distributed free of charge, and is available for anonymous ftp from uxc.cso.uiuc.edu, in the net/qi subdirectory as well as the pub/qi.tar.Z file. The client software for UNIX and VMS is available on the same computer, in the net/ph subdirectory and in the file pub/ph.tar.Z. No support will be provided by the University, and the University is not liable for anything bad that happens as a result of its use. The software may not be redistributed without permission from CSNet.

## References

*UNIX Manual Pages*. Manual pages are available on *ph*(1) and *qi*(8).

[14] It is impossible to get an exact count of the number of machines, since there are some machines that use another computer as a relay; these machines do not show up in the count.

*The CCSO Nameserver, An Introduction* by Steven Dorner. A brief introduction geared at a new user of *ph*.

*The CCSO Nameserver, Why* by Steven Dorner. A recap of the design decisions that made our Nameserver what it is, including evaluations of some similar systems available when our Nameserver was designed.

*The CCSO Nameserver, Server−Client Protocol* by Steven Dorner. Full documentation of the language used between the Nameserver server program, *qi*, and the outside world.

*The CCSO Nameserver, Guide to Installation* by Steven Dorner. How to install the programs that make up the CCSO Nameserver.

*The CCSO Nameserver, A Programmer's Guide* by Steven Dorner. In depth documentation for anyone maintaining or wishing to completely understand the CCSO Nameserver.

*Rebuilding a Nameserver Database In 24 Easy Steps* by Steven Dorner. Describes how we build a database, beginning with raw data we receive from our Administrative branch.

**Acknowledgement**

Our Nameserver is very similar in function and philosophy the CSNet nameserver. In fact, the database management code from that nameserver, with some modification, is used in our Nameserver. We are grateful to CSNet that their program was made available to us.