NAME

askmara - do simple dns queries

DESCRIPTION

askmara queries the user-specified dns server for records, and outputs the reply in a csv2-compatible format (csv2 is the format of zone files that **maradns** uses).

USAGE

askmara [-n] [-v | -t timeout] query [server]

OPTIONS

-t If this is present, the following argument is the askmara timeout, in seconds. Note that **askmara** can not both have a user-defined timeout and verbose output.

-v If this is set, **askmara** will verbosely output the complete reply that the server sent. Note that this verbose output is not csv2-compatible.

-n If this is set, **askmara**, when sending out a query, will not request DNS recursion; in other words, askmara will request that the remote DNS server not contact other DNS server to answer the query in question.

query
: dns record to be queried. The query has two sections: The type of record we desire, and the hostname we want this record for.

The type of query can have two forms: A one-letter mnemonic, or a numeric rtype followed by a colon. This is immediately concatenated by the full name of the host name we wish to look up.

For example, to ask for the IP of 'example.com.', we can use the one-letter mnemonic, in the form 'Aexample.com.', or we can use the numeric RR followed by a colon, giving the query '1:example.com.' (since A has the record type of one). Note that the query name needs the trailing dot at the end.

Askmara supports a handful one-letter mnemonics, as follows:

A signifies a request for an A (ipv4 address) RR

N signifies a NS RR

C signifies that we are asking for a CNAME RR

S signifies that we want a SOA RR

P signifies that we want a PTR RR

@ signifies that we mant a MX RR

T signifies that we want a TXT RR

Z signifies that we want to ask for all RRs.

server
: IP address of the dns server to be queried. If no server is given, askmara will query 127.0.0.1.

EXAMPLES

Asking the server with the ip 127.0.0.1 for the IP address of example.com:

askmara Aexample.com.

Asking the server with the ip 198.41.0.4 for the IP address of example.com:

askmara Aexample.com. 198.41.0.4

Asking the server with the ip address 127.0.0.1 for the IP address of example.com, using the rr_number:query format:

askmara 1:example.com.

Asking the server with the ip address 127.0.0.1 for a SRV record. In particular, we ask for the "http over tcp" service for example.net. Since askmara doesn't have a mnemonic for SRV record types, we use the numeric code (33 for SRV):

askmara 33:_http._tcp.example.net.

Asking the server with the ip address 127.0.0.1 for the AAAA (ipv6 ip) record for example.net:

askmara 28:example.net.

Note that the output will be a raw DNS packet in both the SRV and AAAA examples.

BUGS

When askmara is asked for an SOA record, the output of **askmara** closely resembles the format of a csv2 file, but can not be parsed as a csv2 file without modification.

askmara outputs multi-chunk ("character-string") TXT records incorrectly (it only outputs the first chunk).

SEE ALSO

maradns(8)

http://www.maradns.org

LEGAL DISCLAIMER

INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

AUTHOR

MaraDNS is written by Sam Trenholme. Jaakko Niemi used 5 minutes to roll this manpage together, which Sam has subsequently revised.

NAME

csv2 - Description of the csv2 zone file that MaraDNS uses

DESCRIPTION

The csv2 zone file format is the new zone file format for MaraDNS 1.2. This zone file format uses any kind of whitespace (space, tab, and carriage return), or the '|' character, to deliminate fields. The zone file parser is smart enough to know how many fields the record data for a given record type needs; once all the fields for a given record type is processed, the parser parses the next entry it sees as the name for the next record to process.

This zone file format has records in the following form:

name [+ttl] [rtype] rdata

The name is the name of the record we will add, such as "www.example.net.". This must be placed at the beginning of a line. The rtype is the record type for the record, such as "A" (ipv4 IP address), "MX" (mail exchanger), or "AAAA" (ipv6 IP address). The ttl is how long other DNS servers should store this data in their memory (in seconds); this field needs a '+' as its initial character. The rdata is the actual data for this record; the format for the rdata is type-specific.

Anything in square brackets is an optional field. If the ttl is not specified, the ttl is set to the default ttl value (see "Default TTL" below). If the rtype is not specified, it is set to be an "A" (ipv4 address) record.

The zone file supports comments; comments are specified by having a '#' anywhere between fields or records; when a '#' is seen, the csv2 parser ignores any character it sees (with the exception of the '{' character, which is not currently allowed in csv2 zone files) until a newline. A '#' can usually be placed inside a field, and indicates the end of a field when placed there.

The following record types are supported; a description of the record data format accommodates the record type:

A

An A record stores an ipv4 address. This is the default record type should the record type not be specified. The record type has one field in it: the IP for the record. Examples:

```
a.example.net.            10.11.12.13
b.example.net.       A    10.11.12.14
c.example.net. +64000 A    10.11.12.15
```

PTR

A PTR record stores the name for a given ipv4 or ipv6 address, and is used for reverse DNS lookups. This record type has one field in it: The name for the record in question. Examples:

```
13.12.11.10.in-addr.arpa.       PTR    a.example.net.
14.12.11.10.in-addr.arpa.       PTR    b.example.net.
15.12.11.10.in-addr.arpa. +64000 PTR    c.example.net.
```

MX

A MX record stores a mail exchange record, and is used for mail delivery. This record type has two fields in it: The priority (or "preference" in traditional DNS parlance) of the MX record (lower numbers get higher priority), and the name of the mail exchanger. Example of mail for example.net being mailed to mail.example.net, which has the IP "10.11.12.16":

```
example.net.      MX   10 mail.example.net.
mail.example.net.      10.11.12.16
```

AAAA

An AAAA record stores the ipv6 address for a given name. The IP is in standard ipv6 "colon delimited" format: eight 16-bit hexadecimal numbers are separated by colons. Two colons together indicate multiple streams of all-zero hex numbers. This record has only one field, the v6 IP. Example:

```
a.example.net.   AAAA    3ffe:ffff:ffe:501:ffff::b:c:d
```

SRV

An SRV record stores a "service" definition. This record has four fields: Priority, weight, port, and target. For more information, please refer to RFC 2782. Example:

```
_http._tcp.% SRV 0 0 80 a.%
```

NS

An NS record specifies the name servers for a given zone. If the name servers are not delegation name servers (in other words, if the the name servers are the authoritative name servers for the zone), they need to be at the beginning of the zone, either as the first records in the zone, or right after the SOA record. The NS records are optional; if not present, MaraDNS will make an educated guess of that NS records should be there, based on the IPs the MaraDNS process is bound to. This record has one field: The name of the name server machine. Example:

```
example.net.    NS    ns1.example.net.
example.net.    NS    ns2.example.net.
```

SOA

An SOA record stores the start of authority for a given zone file. This record is optional in a CSV2 zone file; should the record not be in the zone file, MaraDNS will synthesize an appropriate SOA record. This record can only exist once in a zone file: As the first record of the zone file. This record has seven fields: The name of the zone, the email address of the person responsible for the zone, and five numeric fields (serial, refresh, retry, expire, and minimum). Note that the SOA minimum does *not* affect other TTLs in MaraDNS. Example:

x.org. SOA x.org. email@x.org. 1 7200 3600 604800 1800

The serial numeric field may be replaced by the string '/serial'; this string tells the CSV2 zone parser to synthesize a serial number for the zone based on the timestamp for the zone file. This allows one to have the serial number be automatically updated whenever the zone file is edited. Here is how this special field looks in a SOA record:

x.org. SOA x.org. email@x.org. /serial 7200 3600 604800 1800

The '/serial' string is case-sensitive; only '/serial' in all lower case will parse.

TXT

A TXT record stores arbitrary text and/or binary data for a given host name. This record has one field: The text data for the record.

A basic text record can be stored by placing ASCII data between two single quotes, as follows:

example.com. TXT 'This is an example text field'

Any binary data can be specified; see the **csv2_txt(5)** manual page for full details.

SPF

A SPF record is, with the exception of the numeric rtype, identical to a TXT record. SPF records are designed to make it more difficult to forge email. More information about SPF records can be found in RFC4408, or by performing a web search for 'sender policy framework'.

RAW

The RAW record is a special meta-record that allows any otherwise unsupported record type to be stored in a csv2 zone file. The syntax is:

RAW [numeric rtype] [data]

The numeric rtype is a decimal number.

The data field can, among other thing, have backslashed hex sequences outside of quotes, concatenated by ASCII data inside quotes, such as the following example:

example.com. RAW 40 \x10\x01\x02'Kitchen sink'\x40' data'

The above example is a "Kitchen Sink" RR with a "meaning" of 16, a "coding" of 1, a "subcoding" of 2, and a data string of "Kitchen sink@ data" (since hex code 40 corresponds to a @ in ASCII). Note that unquoted hex sequences are concatenated with quoted ASCII data, and that spaces are *only* inside quoted data.

The format for a data field in a RAW record is almost identical to the format for a TXT data field. Both formats are described in full in the **csv2_txt(5)** manual page.

FQDN4

The FQDN4 (short for "Fully Qualified Domain Name for IPv4") record is a special form of the "A" record (see above) that instructs MaraDNS to automatically create the corresponding PTR record. For example, the following is one way of setting up the reverse DNS lookup for x.example.net:

x.example.net. A 10.3.28.79
79.28.3.10.in-addr.arpa. PTR x.example.net.

But the above two lines in a zone file can also be represented thusly:

x.example.net. FQDN4 10.3.28.79

Note that the csv2 parser does not bother to check that any given IP only has a single FQDN4 record; it is up to the DNS administrator to ensure that a given IP has only one FQDN4 record. In the case of there being multiple FQDN4 records with the same IP, MaraDNS will have multiple entries in the corresponding PTR record, which is usually not the desired behavior.

FQDN4 records are not permitted in a csv2_default_zonefile. If you do not know what a csv2_default_zonefile is, you do not have to worry about this limitation.

CNAME

A CNAME record is a pointer to another host name. The CNAME record, in MaraDNS, affects any record type not already specified for a given host name. While MaraDNS allows CNAME and non-CNAME records to share the same host name, this is considered bad practice and is not compatible with some other DNS servers.

CNAME records are not permitted in a csv2_default_zonefile. If you do not know what a csv2_default_zonefile is, this fact is of no relevance.

Historical and uncommon resource records

The following resource records are mainly of historical interest, or are not commonly used.

HINFO

An HINFO record is a description of the CPU (processor) and OS that a given host is using. The format for this record is identical to a TXT record, except that the field must

have precisely two chunks.

The first chunk of a HINFO record is the CPU the host is running; the second chunk is the OS the host is running.

Example:

example.com. HINFO 'Intel Pentium III';'CentOS Linux 3.7'

This resource record is not actively used--the IANA has a list of CPUs and OSes that this record is supposed to have. However, this list has not been updated since 2002.

WKS

WKS records are historical records which have been superseded by SRV records. The format of the record is an IP, followed by a protocol number (6 means TCP), followed by a list of ports that a given server has available for services.

For example, to advertise that example.net has the IP 10.1.2.3, and has a SSH, HTTP (web), and NNTP server:

example.net. WKS 10.1.2.3 6 22,80,119

MaraDNS only allows up to 10 different port numbers in a WKS record, and requires that the listed port numbers are not be higher than 1023.

MD and MF

MD and MF records are RR types that existed before MX records, and were made obsolete by MX records. RFC1035 says that a DNS server can either reject these records or convert these records in to MX records. BIND rejects these records; MaraDNS converts them.

Example:

example.net. MD a.example.net.
example.net. MF b.example.net.

Is equivalent to:

example.net. MX 0 a.example.net.
example.net. MX 10 b.example.net.

MB, MG, MINFO, and MR

In the late 1980s, an alternative to MX records was proposed. This alternative utilized MB, MG, MINFO, and MR records. This alternative failed to gather popularity. However, these records were codified in RFC1035, and are supported by MaraDNS. Here is what the records look like:

```
example.net. MB mail.example.net.
example.net. MG mg@example.net.
example.net. MINFO rm@example.net. re@example.net.
example.net. MR mr@example.net.
```

More information about these records can be found in RFC1035.

AFSDB, RP, X25, ISDN, and RT

AFSDB, RP, X25, ISDN, and RT are resource records which were proposed in RFC1183. None of these resource records are widely used.

With the exception of the ISDN record, the format of these records is identical to the examples in RFC1183. The format of the ISDN record is identical unless the record has a subaddress (SA). If an ISDN record has a subaddress, it is separated from the ISDN-address by a ';' instead of whitespace.

If used, here is how the records would look in a csv2 zone file:

```
example.net. AFSDB 1 afsdb.example.net.
example.net. RP rp@example.net. rp.example.net.
example.net. RP rp2@example.net. .
example.net. X25 311061700956
example.net. ISDN 150862028003217
example.net. ISDN 150862028003217;004
example.net. RT 10 relay.example.net.
```

NSAP and NSAP-PTR

NSAP and NSAP-PTR records were proposed in RFC1706. A NSAP record is a hexadecimal number preceeded by the string "0x" and with optional dots between bytes. This hexadecimal number is converted in to a binary number by MaraDNS. A NSAP-PTR record is idenical to a PTR record, but has a different RTYPE.

More information about these records can be obtained from RFC1706.

If used, here is how the records would look in a csv2 zone file:

```
example.net. NSAP 0x47.0005.80.005a00.0000.0001.e133.ffffff000162.00
example.net. NSAP-PTR nsap.example.net.
```

PX

The PX RR is an obscure RR described in RFC2163. A PX record looks like this in a CSV2 zone file:

```
example.net. PX 15 px1.example.net. px2.example.net.
```

GPOS

An GPOS record is a description of the location of a given server. The format for this record is identical to a TXT record, except that the field must have precisely three chunks.

The first chunk of a GPOS record is the longitude; the second chunk is the latitude; the third chunk is the altitude (in meters).

Example:

example.net. GPOS '-98.6502';'19.283';'2134'

More information about this record can be found in RFC1712.

This resource record is not actively used; for the relatively few people who encode their position in DNS, the LOC record is far more common.

LOC

The LOC recource record is an uncommonly used resource record that describes the position of a given server. LOC records are described in RFC1876.

Note that MaraDNS' LOC parser assumes that the altitude, size, horizontal, and vertical precision numbers are always expressed in meters. Also note that that sub-meter values for size, horizontal, and vertical precision are not allowed. Additionally, the altitude can not be greater than 21374836.47 meters.

Example:

example.net. LOC 19 31 2.123 N 98 3 4 W 2000m 2m 4m 567m

SLASH COMMANDS

In addition to being able to have resource records and comments, csv2 zone files can also have special slash commands. These slash commands, with the exception of the '/serial' slash command (see "SOA" above), can only be placed where the name for a record would be placed. Note that slash commands are case-sensitive, and the command in question must be in all-lower-case.

These commands are as follows:

Default TTL

The default TTL is the TTL for a resource record without a TTL specified. This can be changed with the '/ttl' slash command. This command takes only a single argument: The time, in seconds, for the new default TTL. The '/ttl' slash command only affects the TTL of records that follow the command. A zone file can have multiple '/ttl' slash commands.

The default TTL is 86400 seconds (one day) until changed by the '/ttl' slash command.

In the following example, a.ttl.example.com will have a TTL of 86400 seconds (as long as the zone file with this record has not previously used the '/ttl' slash command), b.ttl.example.com and d.ttl.example.com will have a TTL of 3600 seconds,

c.ttl.example.com will have a TTL of 9600 seconds, and e.ttl.example.com will have a TTL of 7200 seconds:

```
a.ttl.example.com.      10.0.0.1
/ttl 3600
b.ttl.example.com.      10.0.0.2
c.ttl.example.com. +9600 10.0.0.3
d.ttl.example.com.      10.0.0.4
/ttl 7200
e.ttl.example.com.      10.0.0.5
```

Origin

It is possible to change the host name suffix that is used to substitute the percent in a csv2 zone file. This suffix is called, for historical and compatibility reasons, "origin". This is done as the slash command '/origin', taking the new origin as the one argument to this function. Note that changing the origin does *not* change the domain suffix used to determine whether a given domain name is authoritative.

Here is one example usage of the '/origin' slash command:

```
/origin example.com.
www.% 10.1.0.1
% MX 10 mail.%
mail.% 10.1.0.2
/origin example.org.
www.% 10.2.0.1
% MX 10 mail.%
mail.% 10.2.0.2
```

Which is equivalent to:

```
www.example.com. 10.1.0.1
example.com. MX 10 mail.example.com.
mail.example.com. 10.1.0.2
www.example.org. 10.2.0.1
example.org. MX 10 mail.example.org.
mail.example.org. 10.2.0.2
```

It is also possible to make the current origin be part of the new origin:

```
/origin example.com.
% 10.3.2.1 # example.com now has IP 10.3.2.1
/origin mail.%
% 10.3.2.2 # mail.example.com now has IP 10.3.2.2
```

Opush and Opop

The '/opush' and '/opop' slash commands use a stack to remember and later recall values for the origin (see origin above). The '/opush' command is used just like the '/origin' command; however, the current origin is placed on a stack instead of discarded. The '/opop' command removes ("pops") the top element from this stack and makes the element the origin.

For example:

```
/origin example.com.
/opush mail.% # origin is now mail.example.com; example.com is on stack
a.% 10.4.0.1 # a.mail.example.com has IP 10.4.0.1
/opush web.example.com. # mail.example.com and example.com are on stack
a.% 10.5.0.1 # a.web.example.com has IP 10.5.0.1
b.% 10.5.0.2 # b.web.example.com has IP 10.5.0.2
/opop # origin is now mail.example.com again
b.% 10.4.0.2 # b.mail.example.com has IP 10.4.0.2
/opop # origin is now example.com
% MX 10 a.mail.% # example.com. MX 10 a.mail.example.com.
% MX 20 b.mail.% # example.com. MX 20 b.mail.example.com.
```

The opush/opop stack can have up to seven elements on it.

Read

The '/read' slash commands allows one to have the contents of another file in a zone. The '/read' command takes a single argument: A filename that one wishes to read. The filename is only allowed to have letters, numbers, the '-' character, the '_' character, and the '.' character in it.

The file needs to be in the same directory as the zone file. The file will be read with the same privileges as the zone file; content in the file should come from a trusted source or be controlled by the system administrator.

Let us suppose that we have the following in a zone file:

```
mail.foo.example.com. 10.3.2.1
/read foo
foo.example.com. MX 10 mail.foo.example.com.
```

And a file foo with the following contents:

```
foo.example.com. 10.1.2.3
foo.example.com. TXT 'Foomatic!'
```

Then foo.example.com will have an A record with the value 10.1.2.3, a TXT value of 'Foomatic!', and a MX record with priority 10 pointing to mail.foo.example.com. mail.foo.example.com will have the IP 10.3.2.1.

Note that no pre-processing nor post-processing of the origin is done by the '/read' command; should the file read change the origin, this changed value will affect any records after the '/read' command. For example, let us suppose db.example.com looks like this:

```
/origin foo.example.com.
% TXT 'Foomatic!'
/read foo
% MX 10 mail.foo.example.com.
```

And the file foo looks like this:

```
% 10.1.2.3
/origin mail.%
% 10.3.2.1
```

Then the following records will be created:

```
foo.example.com.      TXT   'Foomatic!'
foo.example.com.      A     10.1.2.3
mail.foo.example.com. A     10.3.2.1
mail.foo.example.com. MX 10 mail.foo.example.com.
```

To have something that works like '$INCLUDE filename' in a RFC1035 master file, do the following:

```
/opush %
/read filename
/opop
```

Or, for that matter, the equivalent of '$INCLUDE filename neworigin':

```
/opush neworigin.
/read filename
/opop
```

EXAMPLE ZONE FILE

```
# This is an example csv2 zone file

# First of all, csv2 zone files do not need an SOA record; however, if
# one is provided, we will make it the SOA record for our zone
# The SOA record needs to be the first record in the zone if provided

#%      SOA    % email@% 1 7200 3600 604800 1800
```

```
# Second of all, csv2 zone files do not need authoritative NS records.
# If they aren't there, MaraDNS will synthesize them, based on the IP
# addresses MaraDNS is bound to.  (She's pretty smart about this; if
# Mara is bound to both public and private IPs, only the public IPs will
# be synthesized as NS records)

#%      NS      a.%
#%      NS      b.%

# Here are some A (ipv4 address) records; since this is the most
# common field, the zone file format allows a compact representation
# of it.
a.example.net.          10.10.10.10
b.example.net.  10.10.10.11
b.example.net.  10.10.10.12

# We can have the label in either case; it makes no difference
Z.EXAMPLE.NET.    10.2.3.4
Y.EXAMPLE.net.  10.3.4.5

# We can use the percent shortcut.  When the percent shortcut is present,
# it indicates that the name in question should terminate with the name
# of the zone we are processing.
percent.%       a               10.9.8.7

# And we can have star records
#*.example.net.  A               10.11.12.13

# We can have a ttl in a record; however the ttl needs a '+' before it:
# Note that the ttl has to be in seconds, and is before the RTYPE
d.example.net. +86400 A 10.11.12.13

f.example.net. # As you can see, records can span multiple lines
        A       10.2.19.83

# This allows well-commented records, like this:
c.example.net.                  # Our C class machine
      +86400    # This record is stored for one day
      A         # A record
      10.1.1.1  # Where we are

# We can even have something similiar to csv1 if we want...
e.example.net.|+86400|a|10.2.3.4
h.example.net.|a|10.9.8.7
# Here, we see we can specify the ttl but not the rtype if desired
```

```
g.example.net.|+86400|10.11.9.8

# Here is a MX record
% mx 10 mail.%
mail.% +86400 IN A 10.22.23.24

# We even have a bit of ipv6 support
a.example.net.                  aaaa    3ffe:ffff:1:2:3::4:f

# Not to mention support for SRV records
_http._tcp.% srv 0 0 80 a.%

# And, of course, TXT records
example.net.    txt 'This is some text'
```

LEGAL DISCLAIMER

AUTHOR

Sam Trenholme http://www.samiam.org/

NAME

csv2_txt - Description of txt and raw resource records in the csv2 zone file

DESCRIPTION

Due to the complexity of TXT and RAW records, this man page is dedicated to describing the csv2 format of this RR.

TXT and RAW rrs in MaraDNS' csv2 zone files can store any arbitrary binary data. Additionally, it is possible to arbitrarily divide up TXT records in to chunks (chunks, which RFC1035 call "character-string"s, are described below).

ASCII AND UTF-8 DATA

If a given TXT field or RAW record contains only ASCII data, creating a record is easy: Place the full data between single quotes, like this:

```
a.example.com. TXT 'This is some text'
```

It is also possible, with two notable exceptions, to place any byte with a value less than 0x80 (128) between quotes. If there are any bytes with a value of 0x80 or more, the data must be UTF-8 encoded Unicode.

The two ASCII characters not allowed in quotes are the ' character, and the '{' character. See BACKSLASH ESCAPE SEQUENCES below for information on adding these characters to TXT or RAW fields.

UNQUOTED DATA

Note that the record does not have to be quoted. As long as the record only contains ASCII alphanumeric data, and/or the characters '-', '_', '+', '%', '!', '^', and '=', the data can be unquoted as follows:

```
c.example.com. TXT This_is_100%_unquoted_text_+symbols!
```

It is also possible to mix quoted and unquoted text, such as this:

```
d.example.com. TXT This' is a mix 'of_unquoted' and quoted 'text!
```

Which will have its data look like this:

```
This is a mix of_unquoted and quoted text!
```

When mixing quoted and unquoted data, it is important to have all whitespace *inside* quotes.

BACKSLASH ESCAPE SEQUENCES

In order to accommodate storing non-UTF-8 high bit characters, the single quote character, the '{' character, and to permit multi-line TXT/RAW records (with comments allowed mid-record), the TXT/RAW RR allows backslashes. These backslashes only have significance *outside* of quoted text; if they are placed inside single quotes, they are

not interpreted and result in a literal backslash being added to the resource record data.

The following characters can be backslashed:

' When backslashed, the adds a literal quote to the resource record.

whitespace
: When any whitespace is backslashed (space, newline, cr, and tab), this indicates that the record has not ended, and that more data for this recource will follow. This also allows comments to be placed in TXT and RAW resource records. What happens is that the backslash indicates that any whitespace characters (space, tab, carraige return, and line feed) are to be ignored until the next non-whitespace character that is not a # (hash). If a # is seen, this indicates that we ignore any and all characters until the next carriage return or line feed, and continue to ignore everything until the next non-whitespace character. See the section on multi-line and commented records for examples.

0123
: When a number between 0 and 3 is backslashed, this indicates the beginning of a three-digit octal number.

x When an x is backslashed, this indicates the beginning of a two-digit hexadecimal number.

Note that, with the exception of the single quote, the backslash character is *not* used to remove the meta-significance of a given character. In particular, unlike other environments, it is not possible to backslash spaces. Spaces can be represented either as ' ' in quotes, \x20, or as \040.

Here are some examples of backslashed data. In this example, we see backslash sequences being used to store non-UTF-8 hi-bit data:

```
e.example.com. TXT \x80\x81\x82\x83
```

This same data can also be created as follows:

```
f.example.com. TXT \200\201\202\203
```

Octal and hex information can be mixed:

```
g.example.com. TXT \200\x81\202\x83
```

Literal single quotes can be placed in resource records:

```
h.example.com. TXT 'perl -e \"print "A Perl of a TXT record!\n"\'
```

The above example produces this record:

```
perl -e 'print "A Perl of a TXT record!\n"'
```

To render the '{' character, use the escape sequence \x7b (outside of quotes). For example:

h1.example.com. TXT 'for(a=0;a<10;a++)'\x7b'printf("%d\n",a);sleep(1)}'

Produces this record:

for(a=0;a<10;a++){printf("%d\n",a);sleep(1)}

MULTI-LINE AND COMMENTED RECORDS

By utilizing backslashes followed by comments, it is possible to have multi-line and commented TXT and RAW records. The following resource record will span more than one line on an 80-column display:

i.example.com. TXT 'Not only did the quick brown fox jump over the lazy dog, but the lazy dog jumpe

Without affecting this resource record, the same data can be split over multiple lines:

```
j.example.com. TXT 'Not only did the quick brown fox jump '\
                   'over the lazy dog, but the lazy dog'\
                   ' jumped over the cat.'
```

Some points:

* The backslash must be outsize of the quotes (or a literal backslash will be added to the record)
* The backslash must be present *before* any unquoted white space. Usually, the backslash is placed immediately after the quote character.
* Unlike other environments, it does not matter whether or not there is invisible whitespace after the backslash.

It is also possible to add comments after such a backslash as follows:

```
k.example.com. TXT 'Not only did the quick brown fox jump '\ # The fox
                   'over the lazy dog, but the lazy dog'\    # The dog
                   ' jumped over the cat.'                   # The cat
```

Note that, since the third comment is not preceeded by a backslash, this indicates the end of the resource record.

There can also be multiple lines dedicated to comments (and, optionally, even blank lines) in the middle of TXT and RAW record data:

```
k2.example.com. TXT 'This is some data '\
# Here we have some comments followed by a blank line
```

```
# Now we have some more comments,
# followed by the rest of the data
   'and this is the rest of the data'
```

MULTIPLE TXT CHUNKS

TXT RRs may be divided up in to multiple "chunks" (RFC1035 calls these "character-string"s). A single chunk can be anywhere from zero to 255 bytes long. The default is to have one chunk, as follows:

```
o.example.com. TXT 'TXT record with only one chunk'
```

It is also possible to have a record with multiple chunks. Chunks are delimited by an unquoted ';' character:

```
p.example.com. TXT 'This is chunk one';'This is chunk two'
```

Or:

```
q.example.com. TXT 'This is chunk one';\ # Our first chunk
                    This_is_chunk_two;\  # Our second chunk
                   'This is chunk three' # Our final chunk
```

Quoted ; characters simply add a ; to the record data.

If a single TXT chunk is longer than 255 bytes long, the csv2 parser will report an error in the zone file: Single TXT chunk too long

In order to resolve this, place unquoted ; characters in the record data so that each chunk is under 255 octets (bytes or characters) in length.

It is possible to have zero length chunks:

```
r.example.com. TXT 'chunk one';;'chunk three' # Chunk two zero-length
```

In particular, is is possible to have zero length chunks at the beginning and end of a TXT record:

```
s.example.com. TXT ;'chunk two'; # Chunks one and three zero-length
```

Do not place semicolons at the beginning nor end of TXT records unless you wish to have these zero-length chunks.

Chunk support only exists for TXT records. An unquoted ; character will cause a syntax error in a RAW record.

RAW RECORDS

With the exception of no support for chunk delimiters, and the addition of a numeric record type before the record data, the format for RAW records is identical to text

records. For example, if we wish to have a "Kitchen Sink" RR record, which has the 8-bit binary numbers "16", "1", and "2", followed by the ASCII string "Kitchen sink+ data", we can specify this in any of the following manners:

```
t1.example.com. RAW 40 \x10\x01\x02'Kitchen sink'\x2b' data'


t.example.com. RAW 40 \020\001\002Kitchen' sink+ data'


u.example.com. RAW 40 \x10\x01\x02Kitchen\x20sink+\x20data


v.example.com. RAW 40 \x10\001\x02\
        'Kitchen sink+ data'


w.example.com. RAW 40 \x10\ # Meaning: 16
        \x01\ # Coding: 1
        \x02\ # Sub-coding: 2
        'Kitchen sink+ data' # Data: 'Kitchen sink+ data'
```

LEGAL DISCLAIMER

AUTHOR

Sam Trenholme http://www.samiam.org/

NAME

duende - run a child process as a daemon

DESCRIPTION

duende makes a given child process a daemon. The standard output and standard error of the child process is logged via syslog() with a priority of LOG_INFO.

USAGE

duende child_process [all subsequent arguments passed on to child]

DETAILS

When **duende** is invoked, it spawns two processes. In addition to spawning the daemonized child process, **duende** also spawns a process which reads and logs the standard output of the daemonized process. The parent process stays alive so as to monitor the daemonized process.

duende requires a blank directory named /etc/maradns/logger to run.

Should the parent duende process HUP signal, **duende** will restart the child process. Should the daemonized or logging process received an untrapped HUP signal or exit with an exit code of 8, **duende** will restart the process. Should the daemonized or logging process exit for any other reason, **duende** will send the logger process a TERM signal and exit. Should the duende parent process receive a TERM or INT signal, **duende** sends all of its children TERM signals, then exits.

The duende process must be started as the superuser; this is because Duende's intended child processes (maradns and zoneserver) need to bind to privileged ports, and because duende uses a setuid() call to change the user ID of the logging process to the user with ID 66.

EXAMPLES

Using duende to start maradns, where the mararc file is /etc/mararc.2

```
duende maradns -f /etc/mararc.3
```

Using duende to start zoneserver, where the mararc file is /etc/mararc.4

```
duende zoneserver -f /etc/mararc.4
```

BUGS

Duende assumes that all of its children are well-behaved, eating their vegetables, going to bed when told, and terminating when receiving a TERM signal.

SEE ALSO

maradns(8)

http://www.maradns.org

LEGAL DISCLAIMER

AUTHOR

Duende and this man page are written by Sam Trenholme. D Richard Felker III provided some invaluable assistance with the piping code which **duende** uses.

NAME

fetchzone - get dns zone from server

DESCRIPTION

fetchzone transfers a user-specified dns zone from a zone server and displays it in csv2 format on the standard output.

USAGE

fetchzone zone_name zone_server_IP [query_class]

OPTIONS

zone_name

Name of the dns zone to be transferred.

zone_server_IP

IP address of dns server

query_class

Optional argument which can change the query class from 1 (the default) to 255. This may be needed for some versions of Bind.

EXAMPLES

To obtain the zone example.com from the server 192.168.9.8:

```
fetchzone example.com 192.168.9.8
```

To obtain the zone example.org from the server 10.9.8.78 using a query class of 255:

```
fetchzone example.com 10.9.8.78 255
```

BUGS

Fetchzone will not correctly output host names with utf-8 characters in them.

SEE ALSO

The man pages **maradns(8)** and **csv2(5)**

http://www.maradns.org

LEGAL DISCLAIMER

INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

AUTHOR

Sam Trenholme

Erre con erre cigarro
Erre con erre barril
Rápido ruedan los carros
En el ferrocarril

NAME

maradns - DNS server

SYNOPSIS

maradns [-v | -f mararc_file_location]

TABLE OF CONTENTS

This man page has the following sections:

DESCRIPTION

maradns is a DNS server written with security, simplicity, and performance in mind.

maradns has two forms of arguments, both of which are optional.

The first is the location of a **mararc** file which MaraDNS obtains all configuration information from. The default location of this file is **/etc/mararc**. This is specified in the form **maradns -f mararc_file_location**; *mararc_file_location* is the location of the mararc file.

It is also possible to have MaraDNS display the version number and exit. This is specified by invoking maradns in the form **maradns -v** or **maradns --version**

USAGE

If MaraDNS is functioning only as a recursive nameserver, just one file needs to be set up: The mararc file.

In order for MaraDNS to function as an authoritative nameserver, two or more files need to be set up: the mararc file and one or more "csv1" zone files.

The configuration formation of a csv1 zone file can be obtained from the **csv1(5)** manual page. The configuration format of the mararc file can be obtained from the

mararc(5) manual page.

In order to have MaraDNS run as a daemon, the duende program is used to daemonize MaraDNS. See the **duende(8)** manual page for details.

FIREWALL CONFIGURATION

If MaraDNS is being used as an authoritative nameserver, allow UDP connections from all hosts on the internet to UDP port 53 for the IP that the authoritative nameserver uses.

If MaraDNS is being used as a recursive nameserver, the firewall needs to allow the following packets to go to and from the IP the recursive nameserver uses:

* Allow UDP connections from the MaraDNS-running server to any machine on the internet where the UDP destination port is 53
* Allow UDP connections from any machine on the internet to the IP of the recursive server, where the source port from the remote server is 53, and the destination port is between 15000 and 19095 (inclusive)
* Allow UDP connections from IPs that use MaraDNS as a recursive DNS server to port 53 of the MaraDNS server

MaraDNS uses a strong secure RNG for both the query (16 bits of entropy) and the source port of the query (12 bits of entropy). This makes spoofing replies to a MaraDNS server more difficult, since the attacker has only a one in 250 million chance that a given spoofed reply will be considered valid.

FREQUENTLY ASKED QUESTIONS

INDEX

ANSWERS

1. I'm still using version 1.0 of MaraDNS

MaraDNS 1.0 will continue to be supported until December 21, 2007; this means that MaraDNS 1.0 bug fixes will still be applied. After 2007/12/21, MaraDNS 1.0 will no longer be fully supported; the only updates, at that point, would be bugtraq-worthy

critical security fixes. Not even these security updates will be applied after December 21, 2010.

People who wish to run MaraDNS 1.0 unsupported after 2010/12/21 need to keep in mind that MaraDNS 1.0 is *not* Y2038 compliant, and will have problems starting in 2036 or so. MaraDNS 1.2, on the other hand, is fully Y2038 compliant.

There is still a FAQ for version 1.0 of MaraDNS available here.

Updating from 1.0 to 1.2 requires a minimum number of changes; with most configurations, MaraDNS 1.2 is fully compatible with MaraDNS 1.0 data files. Details are in the updating document in the tutorial.

While csv1 zone files are fully supported in MaraDNS 1.2, there is a Perl script for updating from CSV1 to CSV2 zone files in the tools/ directory of MaraDNS 1.2.

2. How do I try out MaraDNS?

Read the quick start guide, which is the file named 0QuickStart in the MaraDNS distribution.

3. What license is MaraDNS released under?

MaraDNS 1.2 is released with the following two-clause BSD-type license:

4. How do I report bugs in MaraDNS?

Please contact me; my email address is at http://www.maradns.org/contact.html. Please be sure to include all information requested there, including the operating system you are using, the version of MaraDNS you are using, your mararc configuration file, and all relevant zone files.

5. Some of the postings to the mailing list do not talk about MaraDNS!

In cases where I post something to the mailing list which does not directly talk about MaraDNS, the subject line will not have [MARA] in it, but will have some form of the word CHATTER in it.

This way, people who do not like this can set up mail filters to filter out anything that comes from this list and doesn't have [MARA] in the subject line, or simply unsubscribe from the list and read the list from the archives; if one needs to report a bug, they can subscribe to the list again, post their bug, then unsubscribe after a week.

Another option is to set up one's Freshmeat preferences to be notified in email every time I update MaraDNS at Freshmeat. This will give one email notice of any critical bug fixes without needing to be subscribed to the mailing list.

The web page http://www.maradns.org/ has a link to the mailing list archives.

6. How do I get off the mailing list?

Send an email to list-request@maradns.org with "unsubscribe" as the subject line.

7. How do I set up reverse DNS on MaraDNS?

Reverse DNS (sometimes called "reverse mapping") is set up by using PTR (pointer) records. For example, the PTR record which performs the reverse DNS lookup for the ip 10.2.3.4 looks like this in a CSV2 zone file:

```
4.3.2.10.in-addr.arpa. PTR www.example.com.
```

It is also possible, with MaraDNS 1.2.05 and more recent releases, to use a special "FQDN4" which automatically sets up the reverse mapping of a given record:

```
www.example.com. FQDN6 10.2.3.4
```

If you wish to have a PTR (reverse DNS lookup; getting a DNS name from a numeric IP) record work on the internet at large, it is not a simple matter of just adding a record like this to a MaraDNS zonefile. One also needs control of the appropriate in-addr.arpa. domain.

While it can make logical sense to contact the IP 10.11.12.13 when trying to get the reverse DNS lookup (fully qualified domain name) for a given IP, DNS servers don't do this. DNS server, instead, contact the root DNS servers for a given in-addr.arpa name to get the reverse DNS lookup, just like they do with any other record type.

When an internet service provider is given a block of IPs, they are also given control of the DNS zones which allow them to control reverse DNS lookups for those IPs. While it is possible to obtain a domain and run a DNS server without the knowledge or intervention of an ISP, being able to control reverse DNS lookups for those IPs requires ISP intervention.

8. I am on a slow network, and MaraDNS can not process recursive queries

MaraDNS, by default, only waits two seconds for a reply from a remote DNS server. This default can be increased by adding a line like this in the mararc file:

```
timeout_seconds = 5
```

Note that making this too high will slow MaraDNS down when DNS servers are down,

which is, alas, all too common on today's internet.

9. When I try to run MaraDNS, I get a cryptic error message.

There is usually some context of where there is a syntax error in a data file before the cryptic error message. For example, when there is a syntax error in a csv2 zone file, MaraDNS will tell you exactly at what point it had to terminate parsing of the zone file.

If MaraDNS does return a cryptic error message without letting you know what is wrong, let me know so that I can fix the bug. MaraDNS is designed to be easy to use; cryptic error messages go against this spirit.

10. After I start MaraDNS, I can not see the process when I run netstat -na

Udp services do not have a prominent "LISTEN" when netstat is run.

When MaraDNS is up, the relevant line in the netstat output looks like this: udp 0 0 127.0.0.1:53 0.0.0.0:*

While on the topic of netstat, if you run netstat -nap as root on Linux and some other *nix operating systems, you can see the names of the processes which are providing internet services.

11. What string library does MaraDNS use?

MaraDNS uses its own string library, which is called the "js_string" library. Man pages for most of the functions in the js_string library are in the folder doc/man of the MaraDNS distribution

12. Why does MaraDNS use a multi-threaded model?

The multi-threaded model is, plain and simple, the simplest way to write a functioning recursive DNS server. There is a reason why MaraDNS, pdnsd, and BIND 9 all use the multi-threaded model.

I am planning on improving MaraDNS' threaded model to not spawn a thread for each and every uncached request.

13. I feel that XXX feature should be added to MaraDNS

The only thing that will convince me to implement a given feature for MaraDNS is cold, hard cash. If you want me to keep a given feature proprietary, you better have lots of cold hard cash. If you're willing to opensource your feature, less cash should be sufficient.

Keep in mind that both the BIND and NSD name servers were developed by having the programmers paid to work on the programs. PowerDNS was originally commercial software with the author only reluctantly made GPL after seeing that the market for a commercial DNS server is very small. All of the other DNS servers which have been developed as hobbyist projects (Posadis, Pdnsd, and djbdns) are no longer being actively worked on by the primary developer.

My current plans for MaraDNS are visible on the roadmap page for MaraDNS.

If I see a large MaraDNS community and a strong demand for new features from that community, I will consider their wishes. Especially if some of the members of the community have large bank accounts. Should ipv6 start to become dominant, I will update MaraDNS to have full ipv6 support. Should some other technology come along that will require an update to MaraDNS for MaraDNS to continue to function as a DNS server, I may very well update MaraDNS to use that technology.

14. I feel that MaraDNS should use another documentation format

The reason that MaraDNS uses its own documentation format is to satisfy both the needs of translators to have a unified document format and my own need to use a documentation format that is simple enough to be readily understood and which I can add features on an as needed basis.

The documentation format is essentially simplified HTML with some special tags added to meet MaraDNS' special needs.

This gives me more flexibility to adapt the documentation format to changing needs. For example, when someone pointed out that it's not a good idea to have man pages with hi-bit characters, it was a simple matter to add a new HIBIT tag which allows man pages to be without hi-bit characters, and other document formats to retain hi-bit characters.

Having a given program have its own documentation format is not without precedent; Perl uses its own "pod" documentation format.

15. Is there any process I need to follow to add a patch to MaraDNS?

Yes.

Here is the procedure for making a proper patch:

* Enter the directory that the file is in, for example maradns-1.2.00/server
* Copy over the file that you wish to modify to another file name. For example: cp MaraDNS.c MaraDNS.c.orig
* Edit the file in question, e.g: vi MaraDNS.c
* After editing, do something like this:
  diff -u MaraDNS.c.orig MaraDNS.c > maradns.patch
* Make sure the modified version compiles cleanly

Send a patch to me in email, along with a statement that you place the contents of the patch under MaraDNS' BSD license. If I find that the patch works well, I will integrate it in to MaraDNS.

16. Can MaraDNS act as a primary nameserver?

Yes.

The zoneserver program serves zones so that other DNS servers can be secondaries for zones which MaraDNS serves. This is a separate program from the maradns server, which processes both authoritative and recursive UDP DNS queries.

See the DNS master document in the MaraDNS tutorial for details.

17. Can MaraDNS act as a secondary nameserver?

Yes.

Please read the DNS slave document, which is part of the MaraDNS tutorial.

18. What is the difference between an authoritative and a recursive DNS server?

A recursive DNS server is a DNS server that is able to contact other DNS servers in order to resolve a given domain name label. This is the kind of DNS server one points to in /etc/resolve.conf

An authoritative DNS server is a DNS server that a recursive server contacts in order to find out the answer to a given DNS query.

19. The fetchzone client isn't allowing me to add certain hostnames to my zone

For security reasons, MaraDNS' fetchzone client does not add records which are not part of the zone in question. For example, if someone has a zone for example.com, and this record in the zone:

1.1.1.10.in-addr.arpa. PTR dns.example.com.

MaraDNS will not add the record, since the record is out-of-bailiwick. In other words, it is a host name that does not end in .example.com.

There are two workarounds for this issue:

* Create a zone file for 1.1.10.in-addr.arpa., and put the PTR records there.
* Use rcp, rsync, or another method to copy over the zone files in question.

20. Is MaraDNS portable?

MaraDNS is developed on a CentOS 3 and Windows XP dual boot laptop. MaraDNS may compile or run on other systems--there are official MaraDNS ports for Debian/Ubuntu, Slackware, FreeBSD, and NetBSD. Note that MaraDNS needs a system with a robust threading library, which some systems do not have.

21. Can I use MaraDNS in Windows?

Yes. There is both a partial mingw32 (native win32 binary) port and a full Cygwin port of MaraDNS; both of these ports are part of the native build of MaraDNS.

22. MaraDNS freezes up after being used for a while

If you are using MaraDNS 1.2.03.1 (or any 1.1 release, for that matter) on Linux, upgrade to version 1.2.03.2. There is a bug with the Linux kernel which causes UDP clients to freeze unless code is written to work around the kernel bug. This workaround was first introduced in MaraDNS 1.0.28 and 1.1.35 and accidently disabled in 1.2.03.1.

If using your ISP's name servers or some other name servers which are not, in fact, root name servers, please make sure that you are using the upstream_servers dictionary variable instead of the root_servers dictionary variable.

If you still see MaraDNS freeze up after making this correction, please send a bug report to the mailing list.

23. What kind of Python integration does MaraDNS have

The mararc file uses the same syntax that Python uses; in fact, Python can parse a properly formatted mararc file.

There is currently no other integration with Python.

24. Doesn't "kvar" mean "four" in Esperanto?

Indeed, it does. However the use of "kvar" in the MaraDNS source code only coincidentally is an Esperanto word. "kvar" is short for "Kiwi variable"; a lot of the parsing code comes from the code used in the Kiwi spam filter project.

25. How scalable is MaraDNS?

MaraDNS is optimized for serving a small number of domains as quickly as possible. That said, MaraDNS is remarkably efficnent for serving a large number of domains, as long as the server MaraDNS is on has the memory to fit all of the domains, and as long as the startup time for loading a large number of domains can be worked around.

The "big-O" or "theta" growth rates for various MaraDNS functions are as follows, where N is the number of authoritative host names being served:

Startup time	N
Memory usage	N
Processing incoming DNS requests	1

As can be seen, MaraDNS will process 1 or 100000 domains in the same amount of time, once the domain names are loaded in to memory.

26. I am having problems setting upstream_servers

The upstream_servers mararc variable is set thusly:

```
upstream_servers["."] = "10.3.28.79, 10.2.19.83"
```

Note the ["."]. The reason for this is so future versions of MaraDNS may have more fine-grained control over the upstream_servers and root_servers values.

Note that the upstream_servers variable needs to be initialized before being used via upstream_servers = {} (the reason for this is so that a mararc file has 100% Python-compatible syntax). A complete mararc file that uses upstream_servers may look like this:

```
ipv4_bind_addresses = "127.0.0.1"
chroot_dir = "/etc/maradns"
recursive_acl = "127.0.0.1/8"
upstream_servers = {}
upstream_servers["."] = "10.1.2.3, 10.2.4.6"
```

27. Why doesn't the MaraDNS.org web page validate?

HTML pages on the MaraDNS.org web site should validate as HTML 4.0 Transitional. However, the CSS will not validate.

I have designed MaraDNS' web page to be usable and as attractive as possible in any major browser released in the last ten years. Cross-browser support is more important than strict W3 validation. The reason why the CSS does not validate is because I need a way to make sure there is always a scrollbar on the web page, even if the content is not big enough to merit one; this is to avoid the content jumping from page to page. There is no standard CSS tag that lets me do this. I'm using a non-standard tag to enable this in Gecko (Firefox's rendering engine); this is enabled by default in Trident (Internet Explorer's rendering engine). The standards are deficient and blind adherence to them would result in an inferior web site.

There are also two validation warnings generated by redefinitions which are needed as part of the CSS filters used to make the site attractive on older browsers with limited CSS support.

On a related note, the reason why I use tables instead of CSS for some of the layout is because Microsoft Internet Explorer 6 and other browsers do not have support for the max-width CSS property. Without this property, the web page will not scale down correctly without using tables. Additionally, tables allow a reasonably attractive header in browsers without CSS support.

28. How do MX records work?

How MX records work:

* The mail transport agent (Sendmail, Postfix, Qmail, MS Exchange, etc.) looks up the MX record for the domain
* For each of the records returned, the MTA (mail transport agent) looks up the IP for the names.
* It will choose, at random, any of the MXes with the lowest priority number.
* Should that server fail, it will try another server with the same priority number.
* Should all MX records with a given priority number fail, the MTA will try sending email to any of the MX records with the second-lowest priority value.

As an aside, do not have MX records point to CNAMEs.

29. Does MaraDNS have support for SPF?

SPF, or sender policy framework, is method of using DNS that makes it more difficult to forge email. MaraDNS has full support for SPF, both via TXT records and, starting with MaraDNS 1.2.08, via RFC4408 SPF records.

SPF configuration is beyond the scope of MaraDNS' documentation. However, at the time of this FAQ entry being written (June, 2006), information and documentation concerning SPF is available at http://openspf.org. The BIND examples will work in MaraDNS csv2 zone files as long as the double quotes (") are replaced by single quotes

(') . For example, a SPF TXT record that looks like example.net. IN TXT "v=spf1 +mx a:colo.example.com/28 -all" in a BIND zone file will look like example.net. TXT 'v=spf1 +mx a:colo.example.com/28 -all' in a MaraDNS zone file. MaraDNS version 1.2.08 and higher can also make the corresponding SPF record, which will have the syntax example.net. SPF 'v=spf1 +mx a:colo.example.com/28 -all'.

30. I'm having problems resolving CNAMES I have set up.

This is probably because you have set up what MaraDNS calls a dangling CNAME record.

Let us suppose we have a CNAME record without an A record in the local DNS server's database, such as:

```
google.example.com. CNAME www.google.com.
```

This record, which is a CNAME record for "google.example.com", points to "www.google.com". Some DNS servers will recursively look up www.google.com, and render the above record like this:

```
google.example.com. CNAME www.google.com.
www.google.com. CNAME 66.102.7.104
```

For security reasons, MaraDNS doesn't do this. Instead, MaraDNS will simply output:

```
google.example.com. CNAME www.google.com.
```

Some stub resolvers will be unable to resolve google.example.com as a consequence.

If you set up MaraDNS to resolve CNAMEs thusly, you will get a warning in your logs about having a dangling CNAME record.

If you want to remove these warnings, add the following to your mararc file:

```
no_cname_warnings = 1
```

Information about how to get MaraDNS to resolve dangling CNAME records is in the tutorial file dangling.html

I have a NS delegation, and MaraDNS is doing strange things.

In the case of there being a NS delegation, MaraDNS handles recursive queries and non-recursive DNS queries differently. Basically, unless you use askmara with the -n option, dig with the +norecuse option, or nslookup with the -norec option, MaraDNS will try to recursively resolve the record that is delegated.

The thinking is this: A normal recursive DNS query is usually one where one wants to know the final DNS output. So, if MaraDNS delegates a given record to another DNS server, and gets a recursive request for said query, MaraDNS will recursively resolve the query for you.

For example, let us suppose we have a mararc file that looks like this:

```
chroot_dir = "/etc/maradns"
ipv4_bind_addresses = "10.1.2.3"
chroot_dir = "/etc/maradns"
recursive_acl = "127.0.0.1/8, 10.0.0.0/8"
csv2 = {}
csv2["example.com."] = "db.example.com"
```

And a db.example.com file that looks like this:

```
www.example.com.    10.1.2.3
joe.example.com.    NS ns.joe.example.com.
ns.joe.example.com. A 10.1.2.4
```

Next, you are trying to find out why www.joe.example.com is not resolving. If you naively send a query to 10.1.2.3 for www.joe.example.com as askmara Awww.joe.example.com. 10.1.2.3 or as dig @10.1.2.3 www.joe.example.com. or as nslookup www.joe.example.com. 10.1.2.3, you will **not** get any information that will help you solve the problem, since 10.1.2.3 will try to contact 10.1.2.4 to resolve www.joe.example.com.

The solution is to run your DNS query client thusly:

* Askmara would be run thusly:

askmara -n Awww.joe.example.com. 10.1.2.3

* Dig would be run thusly:

dig +norecurse @10.1.2.3 www.joe.example.com

* Nslookup would be run thusly:

nslookup -norec www.joe.example.com 10.1.2.3

This will allow you to see that packets MaraDNS actually sends to a recursive DNS server.

As an aside, this particular problem will not happen if MaraDNS is run only as an authoritative nameserver.

I am transferring a zone from another server, but the NS records are these strange "synth-ip" records.

MaraDNS expects, in csv2 zone files, for all delegation NS records to be between the SOA record and the first non-NS record.

If a zone looks like this:

```
example.net. +600 soa ns1.example.net. hostmaster@example.net
10 10800 3600 604800 1080
example.net. +600 mx 10 mail.example.net.
```

```
example.net. +600 a 10.2.3.5
example.net. +600 ns ns1.example.net.
example.net. +600 ns ns3.example.net.
mail.example.net. +600 a 10.2.3.7
www.example.net. +600 a 10.2.3.11
```

Then the NS records will be "synth-ip" records.

The zone should look like this:

```
example.net. +600 soa ns1.example.net. hostmaster@example.net
10 10800 3600 604800 1080
example.net. +600 ns ns1.example.net.
example.net. +600 ns ns3.example.net.
example.net. +600 mx 10 mail.example.net.
example.net. +600 a 10.2.3.5
mail.example.net. +600 a 10.2.3.7
www.example.net. +600 a 10.2.3.11
```

This will remove the "synth-ip" records.

To automate this process, this awk script is useful:

```
fetchzone whatever.zone.foo 10.1.2.3 | awk '
{if($3 ~ /ns/ || $3 ~ /soa/){print}
else{a = a "\n" $0}}
END{print a}' > zonefile.csv2
```

Replace "whatever.zone.foo" with the name of the zone you are fetchin 10.1.2.3 with the IP address of the DNS master, and zonefile.csv2 with the name of the zone file MaraDNS loads.

Where is the root.hints file?

MaraDNS, unlike BIND, does not need a complicated root.hints file in order to have custom root servers. In order to change the root.hints file, add something like this to your mararc file:

```
root_servers["."] =  "131.161.247.232,"
root_servers["."] += "208.185.249.250,"
root_servers["."] += "66.227.42.140,"
root_servers["."] += "66.227.42.149,"
root_servers["."] += "65.243.92.254"
```

Note that there is no "+=" in the first line, and the last line does not have a comma at the end. Read the recursive tutorial document for more information.

Are there any plans to use autoconf to build MaraDNS?

No. OK, let me qualify that: I won't do it unless you pay me enough money.

In more detail, MaraDNS does not use autoconf for the following reasons:

* Autoconf is designed to solve a problem that existed in the mid 1990s but does not exist today: A large number of different incompatible C compilers and libc implementations. These days, most systems are using gcc as the compiler and some version of glibc as the libc. There is no longer a need, for example, to figure out whether a given implementation of getopt() allows '--' options. MaraDNS's ./configure script can be run in only a second or two; compare this to the 3-5 minute process autoconf's ./configure needs.
* Autoconf leaves GPL-tained files in a program's build tree. MaraDNS is licensed under a BSD license that is *not* GPL-compatible, so MaraDNS can not be distributed with these GPL-licensed files.

This leads us to the next question:

How do I change the compiler or compile-time flags with MaraDNS' build process?

To change the compiler used by MaraDNS:

* Run the ./configure script
* Open up the file Makefile with an editor
* Look for a line that starts with CC
* If there is no line that starts with CC, create one just before the line that starts with FLAGS
* Change (or create) that line to look something like CC=gcc296 In this example, the 2.96 version of gcc is used to compile MaraDNS.
* Note that it is important to **not** remove anything from this line you do not understand; doing so will make MaraDNS unable to compile or run. So, if the CC line looks like CC=gcc $(LDFLAGS) -DNO_FLOCK and you want to compile with gcc 2.96, change the line to look like CC=gcc296 $(LDFLAGS) -DNO_FLOCK retaining the flags added by the configuration script.

Changing compile-time flags is a similar process:

* Run the ./configure script
* Open up the file Makefile with an editor
* Look for a line that starts with FLAGS
* Change (or create) that line to look something like FLAGS=-O3 In this example, MaraDNS is compiled with the -O3 option.
* Note that it is important to **not** remove anything from this line you do not understand; doing so will make MaraDNS unable to compile or run. So, if the FLAGS line looks like FLAGS=-O2 -Wall -DSELECT_PROBLEM and you want to compile at optimization level three, change this line to look like

FLAGS=-O2 -Wall -DSELECT_PROBLEM retaining the flags added by the configuration script. -DSELECT_PROBLEM for example, is needed in the Linux compile or MaraDNS will have problems with freezing up.

Will you make a package for the particular Linux distribution I am using?

No. OK, let me qualify that: I won't do it unless you pay me enough money.

There are MaraDNS packages for a number of different distributions of Linux and other operating systems. On the MaraDNS site, there is a MaraDNS package for CentOS/Red Hat Enterprise Linux available. There is also usually an up-to-date Slackware package available. In addition, there is a Debian package in the Debian packages collection, a FreeBSD port of MaraDNS, a Ubuntu package which is derived from the Debian package, and undoubtably other MaraDNS packages floating around the internet.

If you wish to have a package for your particular version of Linux (or MacOS X or BSD or...), you can use one of the above packages as a starting point for making your package. For example, other RPM-based distributions can use the CentOS RPM package as a baseline (the .spec file is in the build/ directory). I can not help you with any problems you may encounter making this package since I do not have your particular version of Linux installed on my computer.

As an aside, some of the MaraDNS packages floating around on the internet are out of date. Please make sure, that if you get a third-party package from the internet, the package is for either MaraDNS 1.0.40, MaraDNS 1.2.12.05, or MaraDNS 1.3.04. Older versions of MaraDNS are not supported.

I am using the native Windows port of MaraDNS, and some features are not working.

Since Windows 32 does not have some features that *NIX OSes have, the native Windows port does not have all of the features of the *NIX version of MaraDNS. In particular, the following features are disabled:

* ipv6 (this is actually a mingw32, not a Windows deficiency)
* The chroot_dir mararc variable
* The maradns_gid and maradns_uid mararc variables
* The maxprocs mararc variable
* The synth_soa_serial variable can not have a value of 2

If any of the above features are desired, try compiling MaraDNS using Cygwin. Note that the Cygwin port of MaraDNS does not have ipv6 support, and that while chroot_dir works in Cygwin, it does not have the security that the *NIX chroot() call has.

MaraDNS isn't starting up

This is usually caused by a syntax error in one's mararc file, or by another MaraDNS process already running. To see what is happening, look at your system log (/var/log/messages in Centos 3) to see what errors MaraDNS reports. If you do not know how to look at a system log, you can also invoke MaraDNS from the command line as root; any errors will be visible when starting MaraDNS.

You make a lot of releases of MaraDNS; at our ISP/IT department, updating software is non-trivial.

The number of releases seen in the changelog is not an accurate reflection of how often someone using a stable branch of MaraDNS will need to update.

There were only three updates to the 1.0 legacy branch in 2006. The 1.2 branch was updated frequently in the first half of 2006, since I felt MaraDNS 1.2 needed some features that didn't make it in to 1.2.00. During this update cycle, there was always a stable bugfix-only branch of MaraDNS.

In August of 2006, I stabilized the 1.2 branch and only three updates have been done since then. Unless there is a critical bug, I only update the 1.2 branch approximately once every three months or so.

I go to a great deal of effort to make sure MaraDNS releases are as painless to update as possible. I ensure configuration file format compatibility, even between major versions of MaraDNS. With the exception of configuration file parser bugfixes, MaraDNS 1.0 configuration files are compatible with MaraDNS 1.2 and 1.3.

It is impossible to make code that is bug-free or without security problems. This is especially true with code that runs on the public internet.^{1} Code has to be updated from time to time. What I do in order to minimize the disruption caused by an update is to always have a stable bugfix-only branch of MaraDNS (right now I have *two* bugfix-only branches), and to, as much as possible, evenly space out the bugfix updates.

Footnote 1: Even DJB's code has security problems. Both Qmail and DjbDNS have known security problems, and need to be patched before put on a public internet server.

BUGS

In the unusual case of having a csv2 zone file with Macintosh-style newlines (as opposed to DOS or UNIX newlines), while the file will parse, any errors in the file will be reported as being on line 1.

The maximum allowed number of threads is 5000.

The system startup script included with MaraDNS assumes that the only MaraDNS processes running are started by the script; it stops *all* MaraDNS processes running on the server when asked to stop MaraDNS.

When a resolver asks for an A record, and the A record is a CNAME which points to a list of IPs, MaraDNS' recursive resolver only returns the first IP listed along with the CNAME. This is somewhat worked around by having a CNAME record only stay in the recursive cache for 15 minutes.

When a resolver asks for an A record, and the A record is a CNAME that points to another CNAME (and possibly a longer CNAME chain), while MaraDNS returns the correct IP (as long as the glueless level is not exceeded), MaraDNS will incorrectly state that the first CNAME in the chain directly points to the IP.

If a NS record points to a list of IPs, and the NS record in question is a "glueless" record

(MaraDNS had to go back to the root servers to find out the IP of the machine in question), MaraDNS' recursive resolver only uses the first listed IP as a name server.

When MaraDNS' recursive resolver receives a "host not there" reply, instead of using the SOA minimum of the "host not there" reply as the TTL (Look at RFC1034 section 4.3.4), MaraDNS uses the TTL of the SOA reply.

MaraDNS keeps referral NS records in the cache for one day instead of the TTL specified by the remote server.

MaraDNS needs to use the **zoneserver** program to serve DNS records over TCP. See **zoneserver(8)** for usage information.

MaraDNS does not use the zone file ("master file") format specified in chapter 5 of RFC1035.

MaraDNS default behavior with star records is not RFC-compliant. In more detail, if a wildcard MX record exists in the form "*.example.com", and there is an A record for "www.example.com", but no MX record for "www.example.com", the correct behavior (based on RFC1034 section 4.3.3) is to return "no host" (nothing in the answer section, SOA in the authority section, 0 result code) for a MX request to "www.example.com". Instead, MaraDNS returns the MX record attached to "*.example.com". This can be changed by setting bind_star_handling to 1.

Star records (what RFC1034 calls "wildcards") can not be attached to NS records.

MaraDNS recursive resolver treats any TTL shorter than min_ttl seconds (min_ttl_cname seconds when the record is a CNAME record) as if the TTL in question was min_ttl (or min_ttl_cname) seconds long when determining when to expire a record from MaraDNS' cache.

TTLs which are shorter than 20 seconds long are given a TTL of 20 seconds; TTLs which are more than 63072000 (2 years) long are given a TTL of 2 years.

MaraDNS' recursive resolver's method of deleting not recently accessed records from the cache when the cache starts to fill up can deleted records from the cache before they expire. Some people consider this undesirable behavior; I feel it is necessary behavior if one wishes to place a limit on the memory resources a DNS server may use.

MaraDNS' recursive resolver stops resolving when it finds an answer in the AR section. This is a problem in the case where a given host name and IP is registered with the root name servers, and the registered IP is out of date. When this happens, a server "closer" to the root server will give an out-of-date IP, even though the authoritative DNS servers for the host in question have the correct IP. Note that resolving this will result in increased DNS traffic.

MaraDNS, like every other known DNS implementation, only supports a QDCOUNT of 0 or 1.

MaraDNS spawns a new thread for every single recursive DNS request when the data in question is not in MaraDNS' cache; this makes MaraDNS an excellent stress tester for pthread implementations. Many pthread implementations can not handle this kind of

load; symptoms include high memory usage and termination of the MaraDNS process.

MaraDNS does not handle the case of a glueless in-bailiwick NS referral very gracefully; this usually causes the zone pointed to by the offending NS record to be unreachable by MaraDNS, even if other DNS servers for the domain have correct NS referrals.

UNIMPLEMENTED FEATURES

These are features which will not be implemented in the 1.2 release of MaraDNS:

MaraDNS does not have a disk-based caching scheme for authoritative zones.

MaraDNS' UDP server only loads zone files while MaraDNS is first started. UDP Zone information can only be updated by stopping MaraDNS, and restarting MaraDNS again. Note that TCP zone files are loaded from the filesystem at the time the client requests a zone.

MaraDNS does not have support for allowing given host names to only resolve for a limited range of IPs querying the DNS server, or for host names to resolve differently, depending on the IP querying the host name.

MaraDNS only has limited authoritative-only support for IPv6.

MaraDNS only allows wildcards at the beginning or end of a host name. E.g. names with wildcards like "foo.*.example.com". "www.*" will work, however, if a default zonefile is set up.

MaraDNS does not have support for MRTG or any other SNMP-based logging mechanism.

LEGAL DISCLAIMER

AUTHORS

Sam Trenholme (http://www.samiam.org) is responsible for this man page.

MaraDNS is written by me, Sam Trenholme, with a little help from my friends. Naturally, all errors in MaraDNS are my own (but read the disclaimer above).

Here is a partial list of people who have provided assistance:

Floh has generously set up a FreeBSD 4, FreeBSD 6, and Mac OS X system so that I can port MaraDNS to more platforms.

Albert Lee has provided countless bug reports, and, nicely enough, patches to fix said bugs. He has also made improvements to the code in the tcp "zoneserver".

Franky Van Liedekerke has provided much invaluable assistance. As just one example, he provided invaluable assistance in getting MaraDNS to compile on Solaris. In addition, he has provided much valuable SQA help.

Christian Kurz, who has provided invaluable bug reports, especially when I had to re-implement the core hashing algorithm.

Remmy, who is providing both the web space and a mailing list for maradns.org.

Phil Homewood, who provided invaluable assistance with finding and fixing bugs in the authoritative portion of the MaraDNS server. He helped me plug memory leaks, find uninitialized variables being used, and found a number of bugs I was unable to find.

Albert Prats kindly provided Spanish translations for various text files.

Shin Zukeran provided a patch to recursive.c which properly makes a normal null-terminated string from a js_string object, to send as an argument to open() so we can get the rijndael key for the PRNG.

D Richard Felker III has provided invaluable bug reports. By looking at his bug reports, I have been able to hunt down and fix many problems that the recursive nameserver had, in addition to at least one problem with the authoritative nameserver.

Ole Tange has also given me many valuable MaraDNS bug reports.

Florin Iucha provided a tip in the FAQ for how to compile MaraDNS on OpenBSD.

Roy Arends (one of the BIND developers, as it turns out) found a serious security problem with MaraDNS, where MaraDNS would answer answers, and pointed it out to me.

Code used as the basis for the psudo-random-number generator was written by Vincent Rijmen, Antoon Bosselaers, and Paulo Barreto. I appreciate these programmers making the code public domain, which is the only license under which I can add code to MaraDNS under.

Ross Johnson and others have made a Win32 port of the Pthreads library; this has made a native win32 port of MaraDNS possible.

I also appreciate the work of Dr. Brian Gladman and Fritz Schneider, who have both written independent implementations of AES from which I obtained test vectors. With the help of their hard work, I was able to discover a subtle security problem that previous releases of MaraDNS had.

NAME

mararc - Format of the mararc zone file that MaraDNS uses

MARARC FILE FORMAT

Mararc files use a syntax that is a subset of Python 2.2.3 syntax. In particular, Python 2.2.3 (and possibly other versions of Python) can read a properly formatted mararc file without error.

Unlike Python, however, a mararc file can only use certain variable names, and the variables can only be declared as described below.

COMMENTS

Comments (lines ignored by the MaraDNS parser) start with the '#' character, like this:

```
# This is a comment
```

The MaraDNS parser also ignores lines which contain only white space.

OPERATORS

The MaraRC file supports two operators: = and +=

The = operator can be used to assign both numeric and string values

The += operator can only be used on string values, and concatenates the value to the right of the += operator to the string specified to the left of the += operator.

Examples:

```
ipv4_bind_addresses = "10.2.19.83"
ipv4_bind_addresses += ",10.2.66.74"
ipv4_bind_addresses += ",10.3.87.13"

ipv4_bind_addresses now has the value "10.2.19.83,10.2.66.74,10.3.87.13"

ipv4_alias["icann"] = "198.41.0.4"
ipv4_alias["icann"] += ",192.228.79.201"
ipv4_alias["icann"] += ",192.33.4.12,128.8.10.90"
```

MARARC VARIABLES

Follows is a listing of variables that can be declared in the mararc file.

DICTIONARY VARIABLE FORMAT

A **dictionary variable** is an array that can have multiple elements. Unlike a traditional array, these arrays are indexed by strings instead of numbers. These are analogous to associative arrays, or what Perl somewhat inaccurately calls hashes.

The syntax of a dictionary variable is in the following form:

```
name["index"] = "value"
```

Where **name** is the name of the dictionary variable, **index** is the index of the array, and **value** is the value stored at that index.

Every time we have a dictionary-type variable (such as csv2), we must first initialize it using a line in the following form:

```
csv2 = {}
```

Here, csv2 is the name of the "dictionary" variable that we are initializing.

DICTIONARY VARIABLES

Here is a listing of all "dictionary"-style variables that MaraDNS uses:

csv2

The csv2 dictionary variable stores all of the zone names and file names for the zone files that MaraDNS uses. Note that csv2 files are read after MaraDNS is chrooted. Hence the filename is relative to the chroot_dir. Example:

```
csv2["example.net."] = "db.example.net"
```

See **csv2(5)** for a description of this file's format.

csv1

csv1: Used to indicate the filename to use for a given zone stored in the legacy csv1 zone file format. This is primarily for compatibility with people who have maradns-1.0 zone files.

```
csv1["zone"] = "filename"
```

csv1: A pipe-separated-file. See **csv1(5)**.

zone: the zone that file in question is authoritative for

filename: the file with the CSV1 zone data

Note that csv1 files are read after MaraDNS is chrooted, and, hence the filename is relative to the chroot_dir.

See the **csv1(5)** man page for more information on this file format.

ipv4_alias

ipv4_alias: Used to give nicknames or aliases for ip/netmask pairs for ipv4 (standard 32-bit) IP addresses.

```
ipv4_alias["name"] = "ip1/netmask,ip2/netmask,etc"
```

name: The name of the alias in question

ip: The ip portion of an ip/netmask pair

netmask: the mask portion of an ip/netmask pair

,: Used to separate ip/netmask pairs. Spaces may be placed before or after this comma.

An ip is in dotted-decimal format, e.g. "10.1.2.3".

The netmask can be in one of two formats: A single number between 1 and 32, which indicates the number of leading "1" bits in the netmask, or a 4-digit dotted-decimal netmask.

The netmask is used to specify a range of IPs.

ipv4_alias examples

10.1.1.1/24 indicates that any ip from 10.1.1.0 to 10.1.1.255 will match.

10.1.1.1/255.255.255.0 is identical to 10.1.1.1/24

10.2.3.4/16 indicates that any ip from 10.2.0.0 to 10.2.255.255 will match.

10.2.3.4/255.255.0.0 is identical to 10.2.3.4/16

127.0.0.0/8 indicates that any ip with "127" as the first octet (number) will match.

127.0.0.0/255.0.0.0 is identical to 127.0.0.0/8

The netmask is optional, and, if not present, indicates that only a single IP will "match". e.g:

10.9.9.9/32, **10.9.9.9/255.255.255.255**, and **10.9.9.9** are all functionally identical, and indicate that only the ip 10.9.9.9 will match.

The significance of "match" depends on what we use the ipv4 alias for.

ipv4 aliases can nest. E.g:

```
ipv4_alias["susan"] = "10.6.7.8/24"
ipv4_alias["office"] = "susan,10.9.9.9"
```

Where "susan" in the "office" alias matches the value of the ipv4_alias susan.

Multiple levels of nesting are allowed. Self-referring nests will result in an error.

root_servers

root_servers: This is a special "dictionary" element that can (currently) only have one element: ".", which points to either an ip, or a pointer to an ipv4 alias which is a listing of root name servers.

```
root_servers["."] = "list_of_servers"
```

Where "." is the only allowed array reference for the root servers (this format is used to allow potential future expansion), and list_of_servers is a list of root name servers in the exact same format as ipv4_aliases.

Note that, while ips in the list of root name servers can have netmasks, the netmask portion is ignored.

The root_servers should only point to root servers. If one wishes to use MaraDNS as a forwarding name server, which forwards DNS requests on to another server, use the upstream_servers variable instead.

upstream_servers

This is identical to the root_servers variable (can have only one element, the element is a list of ipv4_addresses, the variable is a dictionary variable, etc.), but is used when one wishes to use MaraDNS to query other recursive servers, instead of querying the actual root name servers for an answer.

Note that one can not have both root_servers and upstream_servers set in a given mararc file; MaraDNS will return with a fatal error if one attempts to do this.

If you get a syntax error when trying to use upstream_servers, please search for upstream_servers in the MaraDNS FAQ and read this entry, or look for upstream_servers in the example mararc file below for an example of correct usage of this variable.

Final note on dictionary variables

csv1, csv2, ipv4_alias, and root_servers are currently the only existing dictionary variables.

NORMAL VARIABLE FORMAT

Normal variables. These are variables that can only take a single value.

The syntax of a normal variable is in the form

name = "value"

Where **name** is the name of the normal variable, and **value** is the value of the variable in question.

NORMAL VARIABLES

Here is a listing of normal variables that MaraDNS uses:

ipv4_bind_addresses

ipv4_bind_addresses: The IP addresses to give the MaraDNS server.

This accepts one or more ipv4 IPs in dotted-decimal (e.g. "127.0.0.1") notation, and specifies what IP addresses the MaraDNS server will listen on. Multiple bind addresses are separated with a comma, like this: "10.1.2.3, 10.1.2.4, 127.0.0.1"

admin_acl

This is a list of ip/netmask pairs that are allowed to get certain administrative information about MaraDNS, including:

* The version number of MaraDNS running
* The number of threads MaraDNS has

* MaraDNS' internal timestamp value

Note that this information is not available unless the mararc variable debug_msg_level is sufficiently high. See the information on debug_msg_level below for details on this and on the TXT queries sent to get the above information.

bind_address

bind_address: The IP address to give the MaraDNS server.

This accepts a single IP in dotted-decimal (e.g. "127.0.0.1") notation, and specifies what IP address the MaraDNS server will listen on. Note that ipv4_bind_addresses has the same functionality. This name is included so that MaraDNS 1.0 configuration files will continue to work with MaraDNS 1.2.

bind_star_handling

In the case where there is both a star record for a given name and recordtype, a non-star record with the same name but a different recordtype, and no record for the given name and recordtype, MaraDNS will usually return the star record. BIND, on the other hand, will return a "not there" reply. In other words:

* If a non-A record for foo.example.com exists
* An A record for *.example.com exists
* No A record for foo.example.com exists
* And the user asks for the A record for foo.example.com
* MaraDNS will usually return the A record attached to *.example.com
* BIND, on the other hand, returns a "not there" for foo.example.com

If the BIND behavior is desired, set bind_star_handling to 1. Otherwise, set this to 0 (the default value if this is not set at all in the mararc file).

In addition, if there is a star record that could match any given record type, when bind_star_handling is 1, it makes sure that MaraDNS does not incorrectly return a NXDOMAIN (RFC 4074 section 4.2).

MaraDNS will exit with a fatal error if bind_star_handling has any value besides 0 or 1.

chroot_dir

chroot_dir: The directory MaraDNS chroots to

This accepts a single value: The full path to the directory to use as a chroot jail.

Note that csv1 zone files are read after the chroot operation. Hence, the chroot jail needs to have any and all zone files that MaraDNS will load.

csv2_default_zonefile

This is a special zone file that allows there to be stars at the *end* of hostnames. This file is similar to a normal csv2 zone file, but has the following features and limitations:

* Stars are allowed at the end of hostnames
* A SOA record is mandatory
* NS records are mandatory
* Neither CNAME nor FQDN4 records are permitted in the zone file
* Delegation NS records are not permitted in the zone file
* Default zonefiles may not be transferred via zone transfer
* Both recursion and default zonefiles may not be enabled at the same time.

csv2_synthip_list

Sometimes the IP list of nameservers will be different than the nameservers one is bound to. This allows the synthetic nameserver list to have different IPs.

Note that this may act in an unexpected manner if routable and non-routable (localhost and RFC1918) addresses are combined; in particular, a list with both routable and non-routable addresses will discard the non-routable IP addresses, and a list with rfc1918 and localhost addresses will discard the localhost addresses.

debug_msg_level

This is a number indicating what level of information about a running MaraDNS process should be made public. When set to 0, no information will be made public.

When set to one (the default), or higher, a Tversion.maradns. (TXT query for "version.maradns.") query will return the version number of MaraDNS.

When set to two or higher, a Tnumthreads.maradns. (TXT query for "numthreads.maradns.") query will return the number of threads that MaraDNS is currently running, and a Tcache-elements.maradns. query will return the number of elements in MaraDNS' cache.

If MaraDNS is compiled with debugging information on, a Tmemusage.maradns. query will return the amount of memory MaraDNS has allocated. Note that the overhead for tracking memory usage is considerable and that compiling MaraDNS with "make debug" will greatly slow down MaraDNS. A debug build of MaraDNS is **not** reccomended for production use.

When set to three or higher, a Ttimestamp.maradns. query will return, in seconds since the UNIX epoch, the timestamp for the system MaraDNS is running on.

default_rrany_set

This variable used to determine what kind of resource records were returned when an ANY query was sent. In MaraDNS 1.2, the data structures have been revised to return any resource record type when an ANY query is sent; this variable does nothing, and is only here so that MaraDNS 1.0 mararc files will continue to work. The only accepted values for this variable were 3 and 15.

dos_protection_level

If this is set to a non-zero value, certain features of MaraDNS will be disabled in order to speed up MaraDNS' response time. This is designed for situtations when a MaraDNS server is receiving a large number of queries, such as during a denial of service attack.

This is a numeric variable; its default value is zero, indicating that all of MaraDNS' normal features are enabled. Higher numeric values disable more features:

* A dos_protection_level of 1 or above disables getting MaraDNS status information remotely
* A dos_protection_level of 8 or above disables CNAME lookups.
* A dos_protection_level or 12 or above diables delegation NS records.
* A dos_protection_level of 14 or above disables ANY record processing
* A dos_protection_level of 18 or above disables star record processing at the beginning of hostnames (default_zonefiles still work, however)

ipv6_bind_address

If MaraDNS is compiled with as an authoritative server, then this variable will tell MaraDNS which ipv6 address for the UDP server to; for this variable to be set, MaraDNS must be bound to at least one ipv4 address.

handle_noreply

This is a numeric variable which determines how the recursive resolver informs the client that Mara was unable to contact any remote DNS servers when trying to resolve a given domain. If this is set to 0, no reponse will be sent to the DNS client. If this is set to 1, a "server fail" message will be sent to the DNS client. If this is set to 2, a "this host does not exist" message will be sent to the DNS client. The default value for this is 1.

hide_disclaimer

If this is set to "YES", MaraDNS will not display the legal disclaimer when starting up.

long_packet_ipv4

This is a list of IPs which we will send UDP packets longer than the 512 bytes RFC1035 permits if necessary. This is designed to allow zoneserver, when used send regular DNS packets over TCP, to receive packets with more data than can fit in a 512-byte DNS packet.

This variable only functions if MaraDNS is compiled as an authoritative only server.

maradns_uid

maradns_uid: The numeric UID that MaraDNS will run as

This accepts a single numerical value: The UID to run MaraDNS as.

MaraDNS, as soon as possible drops root privileges, minimizing the damage a potential attacker can cause should there be a security problem with MaraDNS. This is the UID maradns becomes.

The default UID is 99.

maradns_gid

maradns_gid: The numeric GID that MaraDNS will run as.

This accepts a single numerical value: The GID to run MaraDNS as.

The default GID is 99.

maximum_cache_elements

maximum_cache_elements: The maximum number of elements we can have in the cache of recursive queries.

This cache of recursive queries is used to store entries we have previously obtained from recursive queries.

If we approach this limit, the "custodian" kicks in to effect. The custodian removes elements at random from the cache (8 elements removed per query) until we are at the 99% or so level again.

The default value for this variable is 1024.

maxprocs

maxprocs: The maximum number of threads or processes that MaraDNS is allowed to run at the same time.

This variable is used to minimize the impact on the server when MaraDNS is heavily loaded. When this number is reached, it is impossible for MaraDNS to spawn new threads/processes until the number of threads/processes is reduced.

The default value for this variable is 64.

The maximum value this can have is 500.

max_ar_chain

max_ar_chain: The maximum number of records to display if a record in the additional section (e.g., the IP of a NS server or the ip of a MX exchange) has more than one value.

This is similar to max_chain, but applies to records in the "additional" (or AR) section.

Due to limitations in the internal data structures that MaraDNS uses to store RRs, if this has a value besides one, round robin rotates of records are disabled.

The default value for this variable is 1.

max_chain

max_chain: The maximum number of records to display in a chain of records.

With DNS, it is possible to have more than one RR for a given domain label. For example, "example.com" can have, as the A record, a list of multiple ip addresses.

This sets the maximum number of records MaraDNS will show for a single RR.

MaraDNS normally round-robin rotates records. Hence, all records for a given DNS label (e.g. "example.com.") will be visible, although not at the same time if there are

more records than the value allowed with max_chain

The default value for this variable is 8.

max_glueless_level

Maximum glueless level allowed when performing recursive lookups. The default value is 10.

This is the maximum number of times MaraDNS will "go back to the root servers" in order to find out the IP of a name server for which we do not have a glue IP for, or to find out the A value for a given CNAME record.

max_queries_total

Maximum number of queries to perform when performing recursive lookups. The default value is 32.

This is the maximum number of times MaraDNS will send a query to nameservers in order to find out the answer to a DNS question.

max_tcp_procs

max_tcp_procs: The (optional) maximum number of processes the zone server is allowed to run.

Sometimes, it is desirable to have a different number of maximum allowed tcp processes than maximum allowed threads. If this variable is not set, the maximum number of allowed tcp processes is "maxprocs".

max_total

max_total: The maximum number of records to show total for a given DNS request.

This is the maximum total number of records that MaraDNS will make available in a DNS reply.

The default value for this variable is 20.

min_ttl

min_ttl: The minimum amount of time a resource record will stay in MaraDNS' cache, regardless of the TTL the remote server specifies.

Setting this value changes the minimum amount of time MaraDNS' recursive server will keep a record in the cache. The value is in seconds.

The default value of this is 300 (5 minutes); the minimum value for this is 180 (2 minutes).

min_ttl_cname

min_ttl_cname: The minimum amount of time a resource record will stay in MaraDNS' cache, regardless of the TTL the remote server specifies.

Setting this value changes the amount of time a CNAME record stays in the cache. The value is in seconds.

The default value for this is the value min_ttl has; the minimum value for this is 180 (2 minutes).

min_visible_ttl

min_visible_ttl: The minimum value that we will will show as the TTL (time to live) value for a resource record to other DNS servers and stub resolvers. In other words, this is the minimum value we will ask other DNS server to cache (keep in their memory) a DNS resource record.

The value is in seconds. The default value for this is 30; the minimum value this can have is 5. People running highly loaded MaraDNS servers may wish to increase this value to 3600 (one hour) in order to reduce the number of queries recursively processed by MaraDNS.

As an aside, RFC1123 section 6.1.2.1 implies that zero-length TTL records should be passed on with a TTL of zero. This, unfortunatly, breaks some stub resolvers (such as Mozilla's stub resolver).

no_fingerprint

no_fingerprint: Flag that allows MaraDNS to be harder to detect.

Some people do not feel it is appropriate to have some information, such as the version number of MaraDNS being run, be publicly available.

The default value is 0.

By setting no_fingerprint to 1, it is possible to have MaraDNS not reveal this information publicly.

random_seed_file

randsom_seed_file: The file from which we read 16 bytes from to get the 128-bit seed for the secure pseudo random number generator.

This localcation of this file is relative to the root of the filesystem, not MaraDNS' chroot directory.

This is ideally a file which is a good source of random numbers (e.g. /dev/urandom), but can also be a fixed file if your OS does not have a decent random number generator. In that case, make sure the contents of that file is random and with 600 perms, owned by root. We read the file **before** dropping root privileges.

recursive_acl

recursive_acl: List of ips allowed to perform recursive queries with the recursive portion of the MaraDNS server

The format of this string is identical to the format of an ipv4_alias entry.

remote_admin

remote_admin: Whether we allow verbose_level to be changed after MaraDNS is started.

If remote_admin is set to 1, and admin_acl is set, any and all IPs listed in admin_acl

will be able to reset the value of verbose_level from any value between 0 and 9 via a TXT query in the form of 5.verbose_level.maradns. What this will do is set verbose_query to the value in the first digit of the query.

This is useful when wishing to temporarily increase the verbose_level to find out why a given host name is not resolving, then decreasing verbose_level so as to minimize the size of MaraDNS' log.

retry_cycles

retry_cycles: The number of times the recursive resolver will try to contact all of the DNS servers to resolve a given name before giving up. This feature was added to MaraDNS 1.2.08, and has a default value of 2.

spammers

spammers: A list of DNS servers which the recursive resolver will not query.

This is mainly used to not allow spam-friendly domains to resolve, since spammers are starting to get in the habit of using spam-friendly DNS servers to resolve their domains, allowing them to hop from ISP to ISP.

The format of this string is identical to the format of an ipv4_alias entry.

synth_soa_origin

When a CSV2 zone file doesn't have a SOA record in it, MaraDNS generates a SOA record on the fly. This variable determines the host name for the "SOA origin" (which is called the MNAME in RFC1035); this is the host name of the DNS server which has the "master copy" of a given DNS zone's file.

This host name is in human-readable format without a trailing dot, e.g.:

synth_soa_origin = "ns1.example.com"

If this is not set, a synthetic SOA record will use the name of the zone for the SOA origin (MNAME) field.

synth_soa_serial

This determines whether we strictly follow RFC1912 section 2.2 with SOA serial numbers. If this is set to 1 (the default value), we do not strictly follow RFC1912 section 2.2 (the serial is a number, based on the timestamp of the zone file, that is updated every six seconds), but this makes it so that a serial number is guaranteed to be automatically updated every time one edits a zone file.

If this is set to 2, the SOA serial number will be in YYYYMMDDHH format, where YYYY is the 4-digit year, MM is the 2-digit month, DD is the 2-digit day, and HH is the 2-digit hour of the time the zone file was last updated (GMT; localtime doesn't work in a chroot() environment). While this format is strictly RFC1912 compliant, the disadvantage is that more than one edit to a zone file in an hour will not update the serial number.

I strongly recommend, unless it is extremely important to have a DNS zone that generates no warnings when tested at dnsreport.com, to have this set to 1 (the default value). Having this set to 2 can result in updated zone files not being seen by slave DNS servers.

Note that synth_soa_serial can only have a value of 1 on the native Windows port.

tcp_convert_acl

This only applies to the zoneserver (general DNS-over-TCP) program.

This is a list of IPs which are allowed to connect to the zoneserver and send normal TCP DNS requests. The zoneserver will convert TCP DNS requests in to UDP DNS requests, and send the UDP request in question to the server specified in **tcp_convert_server**. Once it gets a reply from the UDP DNS server, it will convert the reply in to a TCP request and send the reply back to the original TCP client.

Whether the RD (recursion desired) flag is set or not when converting a TCP DNS request in to a UDP DNS request is determined by whether the TCP client is on the **recursive_acl** list.

tcp_convert_server

This only applies to the zoneserver (general DNS-over-TCP) program.

This is the UDP server which we send a query to when converting DNS TCP queries in to DNS UDP servers. Note that, while this value allows multiple IPs, all values except the first one are presently ignored.

timeout_seconds

This only applies when performing recursive lookups.

The amount of time, in seconds, to wait for a reply from a remote DNS server before giving up and trying the next server on this list. The default value is 2 seconds.

This is for setups where a recursive MaraDNS server is on a slow network which takes more than two seconds to send and receive a DNS packet.

Note that, the larger this value is, the slower MaraDNS will process recursive queries when a DNS server is not responding to DNS queries.

timestamp_type

timestamp_type: The type of timestamp to display. The main purpose of this option is to supress the output of timestamps. Since duende uses syslog() to output data, and since syslog() adds its own timestamp, this option should be set to 5 when maradns is invoked with the duende tool.

This option also allows people who do not use the duende tool to view human-readable timestamps. This option only allows timestamps in GMT, due to issues with showing local times in a chroot() environment.

This can have the following values:

0 The string "Timestamp" followed by a UNIX timestamp

1 Just the bare UNIX timestamp

2 A GMT timestamp in the Spanish language

3 A (hopefully) local timestamp in the Spanish language

4 A timestamp using asctime(gmtime()); usually in the English language

5 No timestamp whatsoever is shown (this is the best option when maradns is invoked with the duende tool).

6 ISO GMT timestamp is shown

7 ISO local timestamp is shown

The default value for this variable is 5.

verbose_level

verbose_level: The number of messages we log to stdout

This can have five values:

0 No messages except for the legal disclaimer and fatal parsing errors

1 Only startup messages logged (Default level)

2 Error queries logged

3 All queries logged

4 All actions adding and removing records from the cache logged

The default value for this variable is 1.

verbose_query

verbose_query: Whether to verbosely output all DNS queries that the recursive DNS server receives. If this is set to 1, then all recursive queries sent to MaraDNS will be logged.

This is mainly used for debugging.

zone_transfer_acl

zone_transfer_acl: List of ips allowed to perform zone transfers with the zone server

The format of this string is identical to the format of an ipv4_alias entry.

EXAMPLE MARARC FILE

```
# Example mararc file (unabridged version)

# The various zones we support

# We must initialize the csv2 hash, or MaraDNS will be unable to
# load any csv2 zone files
csv2 = {}
```

```
# This is just to show the format of the file
#csv2["example.com."] = "db.example.com"

# The address this DNS server runs on.  If you want to bind
# to multiple addresses, separate them with a comma like this:
# "10.1.2.3,10.1.2.4,127.0.0.1"
ipv4_bind_addresses = "127.0.0.1"
# The directory with all of the zone files
chroot_dir = "/etc/maradns"
# The numeric UID MaraDNS will run as
maradns_uid = 99
# The (optional) numeric GID MaraDNS will run as
# maradns_gid = 99
# The maximum number of threads (or processes, with the zone server)
# MaraDNS is allowed to run
maxprocs = 96
# It is possible to specify a different maximum number of processes that
# the zone server can run.  If this is not set, the maximum number of
# processes that the zone server can have defaults to the 'maxprocs' value
# above
# max_tcp_procs = 64

# Normally, MaraDNS has some MaraDNS-specific features, such as DDIP
# synthesizing, a special DNS query ("erre-con-erre-cigarro.maradns.org."
# with a TXT query returns the version of MaraDNS that a server is
# running), unique handling of multiple QDCOUNTs, etc.  Some people
# might not like these features, so I have added a switch that lets
# a sys admin disable all these features.  Just give "no_fingerprint"
# a value of one here, and MaraDNS should be more or less
# indistinguishable from a tinydns server.
no_fingerprint = 0

# Normally, MaraDNS only returns A and MX records when given a
# QTYPE=* (all RR types) query.  Changing the value of default_rrany_set
# to 15 causes MaraDNS to also return the NS and SOA records, which
# some registars require.  The default value of this is 3
default_rrany_set = 3

# These constants limit the number of records we will display, in order
# to help keep packets 512 bytes or smaller.  This, combined with round_robin
# record rotation, help to use DNS as a crude load-balancer.

# The maximum number of records to display in a chain of records (list
# of records) for a given host name
max_chain = 8
```

```
# The maximum number of records to display in a list of records in the
# additional section of a query.  If this is any value besides one,
# round robin rotation is disabled (due to limitations in the current
# data structure MaraDNS uses)
max_ar_chain = 1
# The maximum number of records to show total for a given question
max_total = 20

# The number of messages we log to stdout
# 0: No messages except for fatal parsing errors and the legal disclaimer
# 1: Only startup messages logged (default)
# 2: Error queries logged
# 3: All queries logged (but not very verbosely right now)
verbose_level = 1

# Initialize the IP aliases, which are used by the list of root name servers,
# the ACL for zone transfers, and the ACL of who gets to perform recursive
# queries
ipv4_alias = {}

# Various sets of root name servers
# Note: Netmasks can exist, but are ignored when specifying root name server

# ICANN: the most common and most controversial root name server
# http://www.icann.org
# This list can be seen at http://www.root-servers.org/
ipv4_alias["icann"]  = "198.41.0.4, 192.228.79.201, 192.33.4.12, 128.8.10.90,"
ipv4_alias["icann"] += "192.203.230.10, 192.5.5.241, 192.112.36.4,"
ipv4_alias["icann"] += "128.63.2.53, 192.36.148.17, 192.58.128.30,"
ipv4_alias["icann"] += "193.0.14.129, 198.32.64.12, 202.12.27.33"

# OpenNIC: http://www.opennic.unrated.net/
# Current as of 2005/11/30; these servers change frequently so please
# look at their web page
ipv4_alias["opennic"]  = "157.238.46.24, 209.104.33.250, 209.104.63.249,"
ipv4_alias["opennic"] += "130.94.168.216, 209.21.75.53, 64.114.34.119,"
ipv4_alias["opennic"] += "207.6.128.246, 167.216.255.199, 62.208.181.95,"
ipv4_alias["opennic"] += "216.87.153.98, 216.178.136.116"

# End of list of root name server lists

# Here is a ACL which restricts who is allowed to perform zone transfer from
# the zoneserver program

# Simplest form: 10.1.1.1/24 (IP: 10.1.1.1, 24 left bits in IP need to match)
```

```
# and 10.100.100.100/255.255.255.224 (IP: 10.100.100.100, netmask
# 255.255.255.224) are allowed to connect to the zone server
# NOTE: The "maradns" program does not serve zones.  Zones are served
# by the "zoneserver" program.
#zone_transfer_acl = "10.1.1.1/24, 10.100.100.100/255.255.255.224"

# More complex: We create two aliases: One called "office" and another
# called "home".  We allow anyone in the office or at home to perform zone
# transfers
#ipv4_alias["office"] = "10.1.1.1/24"
#ipv4_alias["home"] = "10.100.100.100/255.255.255.224"
#zone_transfer_acl = "office, home"

# More complex then the last example.  We have three employees,
# Susan, Becca, and Mia, whose computers we give zone transfer rights to.
# Susan and Becca are system administrators, and Mia is a developer.
# They are all part of the company.  We give the entire company zone
# transfer access
#ipv4_alias["susan"]     = "10.6.7.8/32"  # Single IP allowed
#ipv4_alias["becca"]     = "10.7.8.9"     # also a single IP
#ipv4_alias["mia"]       = "10.8.9.10/255.255.255.255" # Also a single IP
#ipv4_alias["sysadmins"] = "susan, becca"
#ipv4_alias["devel"]     = "mia"
#ipv4_alias["company"]   = "sysadmins, devel"
# This is equivalent to the above line
#ipv4_alias["company"]   = "susan, becca, mia"
#zone_transfer_acl       = "company"

# If you want to enable recursion on the loopback interface, uncomment
# the relevent lines in the following section

# Recursive ACL: Who is allowd to perform recursive queries.  The format
# is identical to that of "zone_transfer_acl", including ipv4_alias support

#ipv4_alias["localhost"] = "127.0.0.0/8"
#recursive_acl = "localhost"

# Random seed file: The file from which we read 16 bytes from to get the
# 128-bit random Rijndael key.  This is ideally a file which is a good source
# of random numbers, but can also be a fixed file if your OS does not have
# a decent random number generator (make sure the contents of that file is
# random and with 600 perms, owned by root, since we read the file *before*
# dropping root privileges)

#random_seed_file = "/dev/urandom"
```

```
# The maximum number of elements we can have in the cache.  If we have more
# elements in the cache than this amount, the "custodian" kicks in to effect,
# removing elements not recently accessed from the cache (8 elements removed
# per query) until we are at the 99% level or so again.

#maximum_cache_elements = 1024

# It is possible to change the minimul "time to live" for entries in the
# cache; this is the minimum time that an entry will stay in the cache.
# Value is in seconds; default is 300 (5 minutes)
#min_ttl = 300
# CNAME records generally take more effort to resolve in MaraDNS than
# non-CNAME records; it is a good idea to make this higher then min_ttl
# default value is to be the same as min_ttl
#min_ttl_cname = 900

# The root servers which we use when making recursive queries.

# The following line must be uncommented to enable custom root servers
# for recursive queries
#root_servers = {}

# You can choose which set of root servers to use.  Current values (set above)
# are: icann, osrc, alternic, opennic,  pacificroot, irsc, tinc, and
# superroot.
#root_servers["."] = "icann"

# If you prefer to contact other recursive DNS servers instead of the ICANN
# root servers, this is done with the upstream_servers mararc variable:
#upstream_servers["."] = "192.168.0.1, 192.168.0.2"

# You can tell MaraDNS to *not* query certain DNS servers when in recursive
# mode.  This is mainly used to not allow spam-friendly domains to resolve,
# since spammers are starting to get in the habit of using spam-friendly
# DNS servers to resolve their domains, allowing them to hop from ISP to
# ISP.  The format of this is the same as for zone_transfer_acl and
# recursive_acl

# For example, at the time of this document (August 12, 2001), azmalink.net
# is a known spam-friendly DNS provider (see doc/detailed/spammers/azmalink.net
# for details.)  Note that this is based on IPs, and azmalink.net constantly
# changes IPs (as they constantly have to change ISPs)
# 2002/10/12: Azmalink changed ISP again, this reflect their current ISP
ipv4_alias["azmalink"] = "12.164.194.0/24"
```

```
# As of September 20, 2001, hiddenonline.net is a known spam-friendly
# DNS provider (see doc/detailed/spammers/hiddenonline for details).
ipv4_alias["hiddenonline"] = "65.107.225.0/24"
spammers = "azmalink,hiddenonline"

# It is also possible to change the maximum number of times MaraDNS will
# follow a CNAME record or a NS record with a glue A record.  The default
# value for this is ten.
#max_glueless_level = 10
# In addition, one can change the maximum number of total queries that
# MaraDNS will perform to look up a host name.  The default value is 32.
#max_queries_total = 32
# In addition, one can change the amount of time that MaraDNS will wait
# for a DNS server to respond before giving up and trying the next DNS
# server on a list.  Note that, the larger this value is, the slower
# MaraDNS will process recursive queries when a DNS server is not
# responding to DNS queries.  The default value is two seconds.
#timeout_seconds = 2


# And that does it for the caching at this point
```

BUGS

If one should declare the same the same index twice with a dictionary variable, MaraDNS will exit with a fatal error. This is because earlier versions of MaraDNS acted in a different manner than Python 2.3.3. With Python 2.3.3, the last declaration is used, while MaraDNS used to use the first declaration.

LEGAL DISCLAIMER

NAME

zoneserver - handle zone transfers and other TCP functions for MaraDNS

DESCRIPTION

zoneserver listens on port 53/tcp and handles DNS zone transfers and any DNS query done over TCP instead of UDP. **zoneserver** uses a configuration file, **/etc/mararc** by default, to determine its parameters.

USAGE

zoneserver -f pointer_to_mararc_file

OPTIONS

-f Specifies the location of the configuration file. MaraDNS uses the same configuration file for both the main dns server and the zoneserver.

CONFIGURATION FILE FORMAT

The file format for the mararc file can be found in the **mararc(5)** manual page. In particular, the zoneserver uses the zone_transfer_acl, tcp_convert_acl, tcp_convert_server, bind_address, and recursive_acl mararc parameters.

SEE ALSO

The man pages **maradns(8)** and **mararc(5)**

http://www.maradns.org

BUGS

zoneserver assumes that the authoritative NS records are immediately after the SOA record, and that there is at least one non-NS between that last authority NS record for the zone and the first delegation NS record.

IXFR requests are incremental zone transfers, meaning that the DNS server should only display records changed since the last IXFR request. **zoneserver**, however, treats an IXFR as if it were an AXFR request, outputting all of the records for the zone in question.

zoneserver closes the TCP connection after transferring the requested zone.

If an unauthorized client attempts to connect to the zoneserver, **zoneserver** immediately disconnects the unauthorized client.

LEGAL DISCLAIMER

NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

AUTHOR

MaraDNS is written by Sam Trenholme. Jaakko Niemi used 5 minutes to put the original version this manpage together. Sam has subsequently revised this manual page.