NAME

Net::DNS − Perl Interface to the Domain Name System

SYNOPSIS

```
use Net::DNS;
```

DESCRIPTION

Net::DNS is a collection of Perl modules that act as a Domain Name System (DNS) resolver. It allows the programmer to perform DNS queries that are beyond the capabilities of ‘‘gethostbyname’’ and ‘‘gethostbyaddr’’.

The programmer should be familiar with the structure of a DNS packet. See RFC 1035 or DNS and BIND (Albitz & Liu) for details.

Resolver Objects

A resolver object is an instance of the Net::DNS::Resolver class. A program may have multiple resolver objects, each maintaining its own state information such as the nameservers to be queried, whether recursion is desired, etc.

Packet Objects

Net::DNS::Resolver queries return Net::DNS::Packet objects. A packet object has five sections:

- header, represented by a Net::DNS::Header object
- question, a list of no more than one Net::DNS::Question object
- answer, a list of Net::DNS::RR objects
- authority, a list of Net::DNS::RR objects
- additional, a list of Net::DNS::RR objects

Update Objects

Net::DNS::Update is a subclass of Net::DNS::Packet used to create dynamic update requests.

Header Object

The Net::DNS::Header object mediates access to the header data which resides within the corresponding Net::DNS::Packet.

Question Object

The Net::DNS::Question object represents the content of the question section of the DNS packet.

RR Objects

Net::DNS::RR is the base class for DNS resource record (RR) objects in the answer, authority, and additional sections of a DNS packet.

Do not assume that RR objects will be of the type requested. The type of an RR object must be checked before calling any methods.

METHODS

Net::DNS exports methods and auxiliary functions to support DNS updates, zone serial number management, and simple DNS queries.

version

```
use Net::DNS;
print Net::DNS->version, "\n";
```

Returns the version of Net::DNS.

rr

```
# Use a default resolver -- can not get an error string this way.
use Net::DNS;
my @rr = rr("example.com");
my @rr = rr("example.com", "AAAA");
my @rr = rr("example.com", "AAAA", "IN");
```

```
# Use your own resolver object.
my $res = Net::DNS::Resolver->new;
my @rr  = rr($res, "example.com" ... );

my ($ptr) = rr("2001:DB8::dead:beef");
```

The `rr()` method provides simple RR lookup for scenarios where the full flexibility of Net::DNS is not required.

Returns a list of Net::DNS::RR objects for the specified name or an empty list if the query failed or no record was found.

See "EXAMPLES" for more complete examples.

mx

```
# Use a default resolver -- can not get an error string this way.
use Net::DNS;
my @mx = mx("example.com");

# Use your own resolver object.
my $res = Net::DNS::Resolver->new;
my @mx  = mx($res, "example.com");
```

Returns a list of Net::DNS::RR::MX objects representing the MX records for the specified name. The list will be sorted by preference. Returns an empty list if the query failed or no MX record was found.

This method does not look up address records; it resolves MX only.

Dynamic DNS Update Support

The Net::DNS module provides auxiliary functions which support dynamic DNS update requests.

yxrrset

Use this method to add an "RRset exists" prerequisite to a dynamic update packet. There are two forms, value-independent and value-dependent:

```
# RRset exists (value-independent)
$update->push( pre => yxrrset("host.example.com AAAA") );
```

Meaning: At least one RR with the specified name and type must exist.

```
# RRset exists (value-dependent)
$update->push( pre => yxrrset("host.example.com AAAA 2001:DB8::dead:beef") );
```

Meaning: At least one RR with the specified name and type must exist and must have matching data.

Returns a Net::DNS::RR object or `undef` if the object could not be created.

nxrrset

Use this method to add an "RRset does not exist" prerequisite to a dynamic update packet.

```
$update->push( pre => nxrrset("host.example.com AAAA") );
```

Meaning: No RRs with the specified name and type can exist.

Returns a Net::DNS::RR object or `undef` if the object could not be created.

yxdomain

Use this method to add a "name is in use" prerequisite to a dynamic update packet.

```
$update->push( pre => yxdomain("host.example.com") );
```

Meaning: At least one RR with the specified name must exist.

Returns a Net::DNS::RR object or `undef` if the object could not be created.

nxdomain

Use this method to add a "name is not in use" prerequisite to a dynamic update packet.

```
$update->push( pre => nxdomain("host.example.com") );
```

Meaning: No RR with the specified name can exist.

Returns a Net::DNS::RR object or `undef` if the object could not be created.

rr_add

Use this method to add RRs to a zone.

```
$update->push( update => rr_add("host.example.com AAAA 2001:DB8::dead:beef") );
```

Meaning: Add this RR to the zone.

RR objects created by this method should be added to the ''update'' section of a dynamic update packet. The TTL defaults to 86400 seconds (24 hours) if not specified.

Returns a Net::DNS::RR object or `undef` if the object could not be created.

rr_del

Use this method to delete RRs from a zone. There are three forms: delete all RRsets, delete an RRset, and delete a specific RR.

```
# Delete all RRsets.
$update->push( update => rr_del("host.example.com") );
```

Meaning: Delete all RRs having the specified name.

```
# Delete an RRset.
$update->push( update => rr_del("host.example.com AAAA") );
```

Meaning: Delete all RRs having the specified name and type.

```
# Delete a specific RR.
$update->push( update => rr_del("host.example.com AAAA 2001:DB8::dead:beef") );
```

Meaning: Delete the RR which matches the specified argument.

RR objects created by this method should be added to the ''update'' section of a dynamic update packet.

Returns a Net::DNS::RR object or `undef` if the object could not be created.

Zone Serial Number Management

The Net::DNS module provides auxiliary functions which support policy-driven zone serial numbering regimes.

SEQUENTIAL

```
$successor = $soa->serial( SEQUENTIAL );
```

The existing serial number is incremented modulo 2**32.

UNIXTIME

```
$successor = $soa->serial( UNIXTIME );
```

The Unix time scale will be used as the basis for zone serial numbering. The serial number will be incremented if the time elapsed since the previous update is less than one second.

YYYYMMDDxx

```
$successor = $soa->serial( YYYYMMDDxx );
```

The 32 bit value returned by the auxiliary `YYYYMMDDxx()` function will be used as the base for the date-coded zone serial number. Serial number increments must be limited to 100 per day for the date information to remain useful.

Sorting of RR arrays

`rrsort()` provides functionality to help you sort RR arrays. In most cases this will give you the result that you expect, but you can specify your own sorting method by using the `Net::DNS::RR::FOO->set_rrsort_func()` class method. See Net::DNS::RR for details.

rrsort

```
use Net::DNS;

my @sorted = rrsort( $rrtype, $attribute, @rr_array );
```

`rrsort()` selects all RRs from the input array that are of the type defined by the first argument. Those RRs are sorted based on the attribute that is specified as second argument.

There are a number of RRs for which the sorting function is defined in the code.

For instance:

```
my @prioritysorted = rrsort( "SRV", "priority", @rr_array );
```

returns the SRV records sorted from lowest to highest priority and for equal priorities from highest to lowest weight.

If the function does not exist then a numerical sort on the attribute value is performed.

```
my @portsorted = rrsort( "SRV", "port", @rr_array );
```

If the attribute is not defined then either the `default_sort()` function or "canonical sorting" (as defined by DNSSEC) will be used.

`rrsort()` returns a sorted array containing only elements of the specified RR type. Any other RR types are silently discarded.

`rrsort()` returns an empty list when arguments are incorrect.

EXAMPLES

The following brief examples illustrate some of the features of Net::DNS. The documentation for individual modules and the demo scripts included with the distribution provide more extensive examples.

See Net::DNS::Update for an example of performing dynamic updates.

Look up host addresses.

```
use Net::DNS;
my $res   = Net::DNS::Resolver->new;
my $reply = $res->search("www.example.com", "AAAA");

if ($reply) {
    foreach my $rr ($reply->answer) {
        print $rr->address, "\n" if $rr->can("address");
    }
} else {
    warn "query failed: ", $res->errorstring, "\n";
}
```

Find the nameservers for a domain.

```
use Net::DNS;
my $res   = Net::DNS::Resolver->new;
my $reply = $res->query("example.com", "NS");

if ($reply) {
    foreach $rr (grep { $_->type eq "NS" } $reply->answer) {
        print $rr->nsdname, "\n";
    }
} else {
    warn "query failed: ", $res->errorstring, "\n";
}
```

Find the MX records for a domain.

```
use Net::DNS;
my $name = "example.com";
my $res  = Net::DNS::Resolver->new;
my @mx   = mx($res, $name);

if (@mx) {
    foreach $rr (@mx) {
        print $rr->preference, "\t", $rr->exchange, "\n";
    }
} else {
    warn "Can not find MX records for $name: ", $res->errorstring, "\n";
}
```

Print domain SOA record in zone file format.

```
use Net::DNS;
my $res   = Net::DNS::Resolver->new;
my $reply = $res->query("example.com", "SOA");

if ($reply) {
    foreach my $rr ($reply->answer) {
        $rr->print;
    }
} else {
    warn "query failed: ", $res->errorstring, "\n";
}
```

Perform a zone transfer and print all the records.

```
use Net::DNS;
my $res  = Net::DNS::Resolver->new;
$res->tcp_timeout(20);
$res->nameservers("ns.example.com");

my @zone = $res->axfr("example.com");

foreach $rr (@zone) {
    $rr->print;
}

warn $res->errorstring if $res->errorstring;
```

Perform a background query and print the reply.

```
use Net::DNS;
my $res    = Net::DNS::Resolver->new;
$res->udp_timeout(10);
$res->tcp_timeout(20);
my $socket = $res->bgsend("host.example.com", "AAAA");

while ( $res->bgbusy($socket) ) {
    # do some work here while waiting for the response
    # ...and some more here
}

my $packet = $res->bgread($socket);
if ($packet) {
    $packet->print;
```

```
} else {
    warn "query failed: ", $res->errorstring, "\n";
}
```

BUGS

Net::DNS is slow.

For other items to be fixed, or if you discover a bug in this distribution please use the CPAN bug reporting system.

COPYRIGHT

LICENSE

AUTHOR INFORMATION

Net::DNS is maintained at NLnet Labs (www.nlnetlabs.nl) by Willem Toorop.

Between 2005 and 2012 Net::DNS was maintained by Olaf Kolkman.

Between 2002 and 2004 Net::DNS was maintained by Chris Reinhardt.

Net::DNS was created in 1997 by Michael Fuhr.

SEE ALSO

perl, Net::DNS::Resolver, Net::DNS::Question, Net::DNS::RR, Net::DNS::Packet, Net::DNS::Update, RFC1035, <http://www.net−dns.org/>, *DNS and BIND* by Paul Albitz & Cricket Liu