NAME

Net::DNS::Packet − DNS packet object class

SYNOPSIS

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

DESCRIPTION

A `Net::DNS::Packet` object represents a DNS packet.

METHODS

new

```
$packet = Net::DNS::Packet->new("example.com");
$packet = Net::DNS::Packet->new("example.com", "MX", "IN");

$packet = Net::DNS::Packet->new(\$data);
$packet = Net::DNS::Packet->new(\$data, 1);  # set debugging

($packet, $err) = Net::DNS::Packet->new(\$data);

$packet = Net::DNS::Packet->new();
```

If passed a domain, type, and class, `new` creates a packet object appropriate for making a DNS query for the requested information. The type and class can be omitted; they default to A and IN.

If passed a reference to a scalar containing DNS packet data, `new` creates a packet object from that data. A second argument can be passed to turn on debugging output for packet parsing.

If called in array context, returns a packet object and an error string. The error string will only be defined if the packet object is undefined (i.e., couldn't be created).

Returns **undef** if unable to create a packet object (e.g., if the packet data is truncated).

If called with an empty argument list, `new` creates an empty packet.

data

```
$data = $packet->data;
```

Returns the packet data in binary format, suitable for sending to a nameserver.

header

```
$header = $packet->header;
```

Returns a `Net::DNS::Header` object representing the header section of the packet.

question, zone

```
@question = $packet->question;
```

Returns a list of `Net::DNS::Question` objects representing the question section of the packet.

In dynamic update packets, this section is known as `zone` and specifies the zone to be updated.

answer, pre, prerequisite

```
@answer = $packet->answer;
```

Returns a list of `Net::DNS::RR` objects representing the answer section of the packet.

In dynamic update packets, this section is known as `pre` or `prerequisite` and specifies the RRs or RRsets which must or must not preexist.

authority, update

```
@authority = $packet->authority;
```

Returns a list of `Net::DNS::RR` objects representing the authority section of the packet.

In dynamic update packets, this section is known as `update` and specifies the RRs or RRsets to be added or deleted.

additional

```
@additional = $packet->additional;
```

Returns a list of `Net::DNS::RR` objects representing the additional section of the packet.

print

```
$packet->print;
```

Prints the packet data on the standard output in an ASCII format similar to that used in DNS zone files.

string

```
print $packet->string;
```

Returns a string representation of the packet.

answerfrom

```
print "packet received from ", $packet->answerfrom, "\n";
```

Returns the IP address from which we received this packet. User-created packets will return undef for this method.

answersize

```
print "packet size: ", $packet->answersize, " bytes\n";
```

Returns the size of the packet in bytes as it was received from a nameserver. User-created packets will return undef for this method (use `length $packet->data` instead).

push

```
$ancount = $packet->push(pre        => $rr);
$nscount = $packet->push(update     => $rr);
$arcount = $packet->push(additional => $rr);

$nscount = $packet->push(update => $rr1, $rr2, $rr3);
$nscount = $packet->push(update => @rr);
```

Adds RRs to the specified section of the packet.

Returns the number of resource records in the specified section.

unique_push

```
$ancount = $packet->unique_push(pre        => $rr);
$nscount = $packet->unique_push(update     => $rr);
$arcount = $packet->unique_push(additional => $rr);

$nscount = $packet->unique_push(update => $rr1, $rr2, $rr3);
$nscount = $packet->unique_push(update => @rr);
```

Adds RRs to the specified section of the packet provided that the RRs do not already exist in the packet.

Returns the number of resource records in the specified section.

safe_push

A deprecated name for `unique_push()`.

pop

```
my $rr = $packet->pop("pre");
my $rr = $packet->pop("update");
my $rr = $packet->pop("additional");
my $rr = $packet->pop("question");
```

Removes RRs from the specified section of the packet.

dn_comp

```
$compname = $packet->dn_comp("foo.example.com", $offset);
```

Returns a domain name compressed for a particular packet object, to be stored beginning at the given offset within the packet data. The name will be added to a running list of compressed domain names for future use.

dn_expand

```
use Net::DNS::Packet qw(dn_expand);
($name, $nextoffset) = dn_expand(\$data, $offset);

($name, $nextoffset) = Net::DNS::Packet::dn_expand(\$data, $offset);
```

Expands the domain name stored at a particular location in a DNS packet. The first argument is a reference to a scalar containing the packet data. The second argument is the offset within the packet where the (possibly compressed) domain name is stored.

Returns the domain name and the offset of the next location in the packet.

Returns **(undef)** if the domain name couldn't be expanded.

sign_tsig

```
$key_name = "tsig-key";
$key      = "awwLOtRfpGE+rRKF2+DEiw==";

$update = Net::DNS::Update->new("example.com");
$update->push("update", rr_add("foo.example.com A 10.1.2.3"));

$update->sign_tsig($key_name, $key);

$response = $res->send($update);
```

Signs a packet with a TSIG resource record (see RFC 2845). Uses the following defaults:

```
algorithm   = HMAC-MD5.SIG-ALG.REG.INT
time_signed = current time
fudge       = 300 seconds
```

If you wish to customize the TSIG record, you'll have to create it yourself and call the appropriate Net::DNS::RR::TSIG methods. The following example creates a TSIG record and sets the fudge to 60 seconds:

```
$key_name = "tsig-key";
$key      = "awwLOtRfpGE+rRKF2+DEiw==";

$tsig = Net::DNS::RR->new("$key_name TSIG $key");
$tsig->fudge(60);

$query = Net::DNS::Packet->new("www.example.com");
$query->sign_tsig($tsig);

$response = $res->send($query);
```

You shouldn't modify a packet after signing it; otherwise authentication will probably fail.

sign_sig0

SIG0 support is provided through the Net::DNS::RR::SIG class. This class is not part of the default Net::DNS distribution but resides in the Net::DNS::SEC distribution.

```
$update = Net::DNS::Update->new("example.com");
$update->push("update", rr_add("foo.example.com A 10.1.2.3"));
$update->sign_sig0("Kexample.com+003+25317.private");
```

SIG0 support is experimental see Net::DNS::RR::SIG for details.

The method will call `Carp::croak()` if Net::DNS::RR::SIG cannot be found.

truncate

The truncate method takes a maximum length as argument and then tries to truncate the packet an set the TC bit according to the rules of RFC2181 Section 9.

The minimum maximum length that is honored is 512 octets.

COPYRIGHT

same terms as Perl itself.

SEE ALSO

perl (1), Net::DNS, Net::DNS::Resolver, Net::DNS::Update, Net::DNS::Header, Net::DNS::Question, Net::DNS::RR, RFC 1035 Section 4.1, RFC 2136 Section 2, RFC 2845