NAME

Net::netent - by-name interface to Perl's built-in getnet*() functions

SYNOPSIS

```
use Net::netent qw(:FIELDS);
getnetbyname("loopback") or die "bad net";
printf "%s is %08X\n", $n_name, $n_net;

use Net::netent;

$n = getnetbyname("loopback") or die "bad net";
{ # there's gotta be a better way, eh?
@bytes = unpack("C4", pack("N", $n->net));
shift @bytes while @bytes && $bytes[0] == 0;
}
printf "%s is %08X [%d.%d.%d.%d]\n", $n->name, $n->net, @bytes;
```

DESCRIPTION

This module's default exports override the core *getnetbyname()* and *getnetbyaddr()* functions, replacing them with versions that return "Net::netent" objects. This object has methods that return the similarly named structure field name from the C's netent structure from *netdb.h*; namely name, aliases, addrtype, and net. The aliases method returns an array reference, the rest scalars.

You may also import all the structure fields directly into your namespace as regular variables using the :FIELDS import tag. (Note that this still overrides your core functions.) Access these fields as variables named with a preceding `n_`. Thus, `$net_obj->name()` corresponds to `$n_name` if you import the fields. Array references are available as regular array variables, so for example `@{ $net_obj->aliases() }`w ould be simply `@n_aliases`.

The *getnet()* function is a simple front-end that forwards a numeric argument to *getnetbyaddr()*, and the rest to *getnetbyname()*.

To access this functionality without the core overrides, pass the `use` an empty import list, and then access function functions with their full qualified names. On the other hand, the built-ins are still available via the `CORE::` pseudo-package.

EXAMPLES

The *getnet()* functions do this in the Perl core:

```
sv_setiv(sv, (I32)nent->n_net);
```

The *gethost()* functions do this in the Perl core:

```
sv_setpvn(sv, hent->h_addr, len);
```

That means that the address comes back in binary for the host functions, and as a regular perl integer for the net ones. This seems a bug, but here's how to deal with it:

```
use strict;
use Socket;
use Net::netent;

@ARGV = ('loopback') unless @ARGV;

my($n, $net);

for $net ( @ARGV ) {

unless ($n = getnetbyname($net)) {
warn "$0: no such net: $net\n";
next;
}

printf "\n%s is %s%s\n",
$net,
lc($n->name) eq lc($net) ? "" : "*really* ",
$n->name;

print "\taliases are ", join(", ", @{$n->aliases}), "\n"
if @{$n->aliases};

# this is stupid; first, why is this not in binary?
# second, why am i going through these convolutions
# to make it looks right
{
my @a = unpack("C4", pack("N", $n->net));
shift @a while @a && $a[0] == 0;
printf "\taddr is %s [%d.%d.%d.%d]\n", $n->net, @a;
}

if ($n = getnetbyaddr($n->net)) {
if (lc($n->name) ne lc($net)) {
printf "\tThat addr reverses to net %s!\n", $n->name;
$net = $n->name;
redo;
}
}
}
```

NOTE

While this class is currently implemented using the Class::Struct module to build a struct-like class, you shouldn't rely upon this.

AUTHOR

Tom Christiansen