NAME

open - perl pragma to set default PerlIO layers for input and output

SYNOPSIS

```
use open IN  => ':crlf', OUT => ':raw';
open my $in, '<', 'foo.txt' or die "open failed: $!";
my $line = <$in>; # CRLF translated
close $in;
open my $out, '>', 'bar.txt' or die "open failed: $!";
print $out $line; # no translation of bytes
close $out;

use open OUT => ':encoding(UTF-8)';
use open IN  => ':encoding(iso-8859-7)';

use open IO  => ':locale';

# IO implicit only for :utf8, :encoding, :locale
use open ':encoding(UTF-8)';
use open ':encoding(iso-8859-7)';
use open ':locale';

# with :std, also affect global standard handles
use open ':std', ':encoding(UTF-8)';
use open ':std', OUT => ':encoding(cp1252)';
use open ':std', IO => ':raw :encoding(UTF-16LE)';
```

DESCRIPTION

Full-fledged support for I/O layers is now implemented provided Perl is configured to use PerlIO as its IO system (which has been the default since 5.8, and the only supported configuration since 5.16).

The "open" pragma serves as one of the interfaces to declare default "layers" (previously known as "disciplines") for all I/O. Any **open()**, **readpipe()** (aka qx//) and similar operators found within the lexical scope of this pragma will use the declared defaults via the "${^OPEN}" variable.

Layers are specified with a leading colon by convention. You can specify a stack of multiple layers as a space-separated string. See PerlIO for more information on the available layers.

With the "IN" subpragma you can declare the default layers of input streams, and with the "OUT" subpragma you can declare the default layers of output streams. With the "IO" subpragma (may be

omitted for ":utf8", ":locale", or ":encoding") you can control both input and output streams simultaneously.

When **open()** is given an explicit list of layers (with the three-arg syntax), they override the list declared using this pragma. **open()** can also be given a single colon (:) for a layer name, to override this pragma and use the default as detailed in "Defaults and how to override them" in PerlIO.

To translate from and to an arbitrary text encoding, use the ":encoding" layer. The matching of encoding names in ":encoding" is loose: case does not matter, and many encodings have several aliases. See Encode::Supported for details and the list of supported locales.

If you want to set your encoding layers based on your locale environment variables, you can use the ":locale" pseudo-layer. For example:

```
$ENV{LANG} = 'ru_RU.KOI8-R';
# the :locale will probe the locale environment variables like LANG
use open OUT => ':locale';
open(my $out, '>', 'koi8') or die "open failed: $!";
print $out chr(0x430); # CYRILLIC SMALL LETTER A = KOI8-R 0xc1
close $out;
open(my $in, '<', 'koi8') or die "open failed: $!";
printf "%#x\n", ord(<$in>); # this should print 0xc1
close $in;
```

The logic of ":locale" is described in full in "The ":locale" sub-pragma" in encoding, but in short it is first trying nl_langinfo(CODESET) and then guessing from the LC_ALL and LANG locale environment variables. ":locale" also implicitly turns on ":std".

":std" is not a layer but an additional subpragma. When specified in the import list, it activates an additional functionality of pushing the layers selected for input/output handles to the standard filehandles (STDIN, STDOUT, STDERR). If the new layers and existing layer stack both end with an ":encoding" layer, the existing ":encoding" layer will also be removed.

For example, if both input and out are chosen to be ":encoding(UTF-8)", a ":std" will mean that STDIN, STDOUT, and STDERR will also have ":encoding(UTF-8)" set. On the other hand, if only output is chosen to be in ":encoding(koi8r)", a ":std" will cause only the STDOUT and STDERR to be in "koi8r".

The effect of ":std" is not lexical as it modifies the layer stack of the global handles. If you wish to apply only this global effect and not the effect on handles that are opened in that scope, you can isolate

the call to this pragma in its own lexical scope.

```
{ use open ':std', IO => ':encoding(UTF-8)' }
```

IMPLEMENTATION DETAILS

There is a class method in "PerlIO::Layer" "find" which is implemented as XS code. It is called by "import" to validate the layers:

```
PerlIO::Layer::->find("perlio")
```

The return value (if defined) is a Perl object, of class "PerlIO::Layer" which is created by the C code in *perlio.c*. As yet there is nothing useful you can do with the object at the perl level.

SEE ALSO

"binmode" in perlfunc, "open" in perlfunc, perlunicode, PerlIO, encoding