NAME

PerlIO - On demand loader for PerlIO layers and root of PerlIO::* name space

SYNOPSIS

```
  open($fh,"<:crlf", "my.txt"); # support platform-native and CRLF text
files

  open($fh,"<","his.jpg");      # portably open a binary file for reading
  binmode($fh);

  Shell:
    PERLIO=perlio perl ....
```

DESCRIPTION

When an undefined layer 'foo' is encountered in an `open` or `binmode` layer specification then C code performs the equivalent of:

```
  use PerlIO 'foo';
```

The perl code in PerlIO.pm then attempts to locate a layer by doing

```
  require PerlIO::foo;
```

Otherwise the `PerlIO` package is a place holder for additional PerlIO related functions.

The following layers are currently defined:

:unix

Lowest level layer which provides basic PerlIO operations in terms of UNIX/POSIX numeric file descriptor calls (open(), read(), write(), lseek(), close()).

:stdio

Layer which calls `fread`, `fwrite` and `fseek`/`ftell` etc. Note that as this is "real" stdio it will ignore any layers beneath it and got straight to the operating system via the C library as usual.

:perlio

A from scratch implementation of buffering for PerlIO. Provides fast access to the buffer for `sv_gets` which implements perl's readline/<> and in general attempts to minimize data copying.

`:perlio` will insert a `:unix` layer below itself to do low level IO.

:crlf

A layer that implements DOS/Windows like CRLF line endings. On read converts pairs of CR,LF to a single "\n" newline character. On write converts each "\n" to a CR,LF pair. Note that this layer likes to be one of its kind: it silently ignores attempts to be pushed into the layer stack more than once.

It currently does *not* mimic MS-DOS as far as treating of Control-Z as being an end-of-file marker.

(Gory details follow) To be more exact what happens is this: after pushing itself to the stack, the `:crlf` layer checks all the layers below itself to find the first layer that is capable of being a CRLF layer but is not yet enabled to be a CRLF layer. If it finds such a layer, it enables the CRLFness of that other deeper layer, and then pops itself off the stack. If not, fine, use the one we just pushed.

The end result is that a `:crlf` means "please enable the first CRLF layer you can find, and if you can't find one, here would be a good spot to place a new one."

Based on the `:perlio` layer.

:mmap

A layer which implements "reading" of files by using `mmap()` to make (whole) file appear in the process's address space, and then using that as PerlIO's "buffer". This *may* be faster in certain circumstances for large files, and may result in less physical memory use when multiple processes are reading the same file.

Files which are not `mmap()`-able revert to behaving like the `:perlio` layer. Writes also behave like `:perlio` layer as `mmap()` for write needs extra house-keeping (to extend the file) which negates any advantage.

The `:mmap` layer will not exist if platform does not support `mmap()`.

:utf8

Declares that the stream accepts perl's *internal* encoding of characters. (Which really is UTF-8 on ASCII machines, but is UTF-EBCDIC on EBCDIC machines.) This allows any character perl can represent to be read from or written to the stream. The UTF-X encoding is chosen to render simple text parts (i.e. non-accented letters, digits and common punctuation) human readable in the encoded file.

Here is how to write your native data out using UTF-8 (or UTF-EBCDIC) and then read it back in.

```
open(F, ">:utf8", "data.utf");
print F $out;
close(F);

open(F, "<:utf8", "data.utf");
$in = <F>;
close(F);
```

Note that this layer does not validate byte sequences. For reading input, using `:encoding(utf8)` instead of bare `:utf8`, is strongly recommended.

:bytes

This is the inverse of `:utf8` layer. It turns off the flag on the layer below so that data read from it is considered to be "octets" i.e. characters in range 0..255 only. Likewise on output perl will warn if a "wide" character is written to a such a stream.

:raw

The `:raw` layer is *defined* as being identical to calling `binmode($fh)` - the stream is made suitable for passing binary data i.e. each byte is passed as-is. The stream will still be buffered.

In Perl 5.6 and some books the `:raw` layer (previously sometimes also referred to as a "discipline") is documented as the inverse of the `:crlf` layer. That is no longer the case - other layers which would alter binary nature of the stream are also disabled. If you want UNIX line endings on a platform that normally does CRLF translation, but still want UTF-8 or encoding defaults the appropriate thing to do is to add `:perlio` to PERLIO environment variable.

The implementation of `:raw` is as a pseudo-layer which when "pushed" pops itself and then any layers which do not declare themselves as suitable for binary data. (Undoing :utf8 and :crlf are implemented by clearing flags rather than popping layers but that is an implementation detail.)

As a consequence of the fact that `:raw` normally pops layers it usually only makes sense to have it as the only or first element in a layer specification. When used as the first element it

provides a known base on which to build e.g.

```
open($fh,":raw:utf8",...)
```

will construct a "binary" stream, but then enable UTF-8 translation.

:pop

A pseudo layer that removes the top-most layer. Gives perl code a way to manipulate the layer stack. Should be considered as experimental. Note that `:pop` only works on real layers and will not undo the effects of pseudo layers like `:utf8`. An example of a possible use might be:

```
open($fh,...)
...
binmode($fh,":encoding(...)");  # next chunk is encoded
...
binmode($fh,":pop");            # back to un-encoded
```

A more elegant (and safer) interface is needed.

:win32

On Win32 platforms this *experimental* layer uses native "handle" IO rather than unix-like numeric file descriptor layer. Known to be buggy as of perl 5.8.2.

Custom Layers

It is possible to write custom layers in addition to the above builtin ones, both in C/XS and Perl. Two such layers (and one example written in Perl using the latter) come with the Perl distribution.

:encoding

Use `:encoding(ENCODING)` either in open() or binmode() to install a layer that does transparently character set and encoding transformations, for example from Shift-JIS to Unicode. Note that under `stdio` an `:encoding` also enables `:utf8`. See *PerlIO::encoding* for more information.

:via

Use `:via(MODULE)` either in open() or binmode() to install a layer that does whatever transformation (for example compression / decompression, encryption / decryption) to the filehandle. See *PerlIO::via* for more information.

Alternatives to raw

To get a binary stream an alternate method is to use:

```
open($fh,"whatever")
binmode($fh);
```

this has advantage of being backward compatible with how such things have had to be coded on some platforms for years.

To get an un-buffered stream specify an unbuffered layer (e.g. `:unix`) in the open call:

```
open($fh,"<:unix",$path)
```

Defaults and how to override them

If the platform is MS-DOS like and normally does CRLF to "\n" translation for text files then the default layers are :

```
unix crlf
```

(The low level "unix" layer may be replaced by a platform specific low level layer.)

Otherwise if `Configure` found out how to do "fast" IO using system's stdio, then the default layers are:

```
unix stdio
```

Otherwise the default layers are

```
unix perlio
```

These defaults may change once perlio has been better tested and tuned.

The default can be overridden by setting the environment variable PERLIO to a space separated list of layers (`unix` or platform low level layer is always pushed first).

This can be used to see the effect of/bugs in the various layers e.g.

```
cd .../perl/t
PERLIO=stdio  ./perl harness
PERLIO=perlio ./perl harness
```

For the various value of PERLIO see *"PERLIO" in perlrun*.

Querying the layers of filehandles

The following returns the **names** of the PerlIO layers on a filehandle.

```
my @layers = PerlIO::get_layers($fh); # Or FH, *FH, "FH".
```

The layers are returned in the order an open() or binmode() call would use them. Note that the "default stack" depends on the operating system and on the Perl version, and both the compile-time and runtime configurations of Perl.

The following table summarizes the default layers on UNIX-like and DOS-like platforms and depending on the setting of the `$ENV{PERLIO}`:

```
PERLIO     UNIX-like                  DOS-like
------     ---------                  --------
unset / "" unix perlio / stdio [1]    unix crlf
stdio      unix perlio / stdio [1]    stdio
perlio     unix perlio                unix perlio
mmap       unix mmap                  unix mmap

# [1] "stdio" if Configure found out how to do "fast stdio" (depends
# on the stdio implementation) and in Perl 5.8, otherwise "unix perlio"
```

By default the layers from the input side of the filehandle is returned, to get the output side use the optional `output` argument:

```
my @layers = PerlIO::get_layers($fh, output => 1);
```

(Usually the layers are identical on either side of a filehandle but for example with sockets there may be differences, or if you have been using the `open` pragma.)

There is no set_layers(), nor does get_layers() return a tied array mirroring the stack, or anything fancy like that. This is not accidental or unintentional. The PerlIO layer stack is a bit more complicated than just a stack (see for example the behaviour of `:raw`). You are supposed to use open() and

binmode() to manipulate the stack.

Implementation details follow, please close your eyes.

The arguments to layers are by default returned in parenthesis after the name of the layer, and certain layers (like `utf8`) are not real layers but instead flags on real layers: to get all of these returned separately use the optional `details` argument:

```
my @layer_and_args_and_flags = PerlIO::get_layers($fh, details => 1);
```

The result will be up to be three times the number of layers: the first element will be a name, the second element the arguments (unspecified arguments will be `undef`), the third element the flags, the fourth element a name again, and so forth.

You may open your eyes now.

AUTHOR

Nick Ing-Simmons <nick@ing-simmons.net>

SEE ALSO

"binmode" in perlfunc, *"open" in perlfunc*, *perlunicode*, *perliol*, *Encode*