

## NAME

Encode::PerLIO -- a detailed document on Encode and PerLIO

## Overview

It is very common to want to do encoding transformations when reading or writing files, network connections, pipes etc. If Perl is configured to use the new 'perlio' IO system then `Encode` provides a "layer" (see *PerLIO*) which can transform data as it is read or written.

Here is how the blind poet would modernise the encoding:

```
use Encode;
open(my $iliad, '<:encoding(iso-8859-7)', 'iliad.greek');
open(my $utf8, '>:utf8', 'iliad.utf8');
my @epic = <$iliad>;
print $utf8 @epic;
close($utf8);
close($iliad);
```

In addition, the new IO system can also be configured to read/write UTF-8 encoded characters (as noted above, this is efficient):

```
open(my $fh, '>:utf8', 'anything');
print $fh "Any \x{0021} string \N{SMILEY FACE}\n";
```

Either of the above forms of "layer" specifications can be made the default for a lexical scope with the use `open ... pragma`. See *open*.

Once a handle is open, its layers can be altered using `binmode`.

Without any such configuration, or if Perl itself is built using the system's own IO, then write operations assume that the file handle accepts only *bytes* and will *die* if a character larger than 255 is written to the handle. When reading, each octet from the handle becomes a byte-in-a-character. Note that this default is the same behaviour as bytes-only languages (including Perl before v5.6) would have, and is sufficient to handle native 8-bit encodings e.g. iso-8859-1, EBCDIC etc. and any legacy mechanisms for handling other encodings and binary data.

In other cases, it is the program's responsibility to transform characters into bytes using the API above before doing writes, and to transform the bytes read from a handle into characters before doing "character operations" (e.g. `lc`, `/\W+/`, ...).

You can also use `PerLIO` to convert larger amounts of data you don't want to bring into memory. For example, to convert between ISO-8859-1 (Latin 1) and UTF-8 (or UTF-EBCDIC in EBCDIC machines):

```
open(F, "<:encoding(iso-8859-1)", "data.txt") or die $!;
open(G, ">:utf8", "data.utf") or die $!;
while (<F>) { print G }

# Could also do "print G <F>" but that would pull
# the whole file into memory just to write it out again.
```

More examples:

```
open(my $f, "<:encoding(cp1252)")
open(my $g, ">:encoding(iso-8859-2)")
open(my $h, ">:encoding(latin9)") # iso-8859-15
```



Fortunately iso-2022-kr is hardly used (according to Jungshik) and MIME-\* are very unlikely to be fed to PerIO because they are for mail headers. See *Encode::MIME::Header* for details.

### How can I tell whether my encoding fully supports PerIO ?

As of this writing, any encoding whose class belongs to Encode::XS and Encode::Unicode works. The Encode module has a `perlio_ok` method which you can use before applying PerIO encoding to the filehandle. Here is an example:

```
my $use_perlio = perlio_ok($enc);
my $layer = $use_perlio ? "<:raw" : "<:encoding($enc)";
open my $fh, $layer, $file or die "$file : $!";
while(<$fh>){
    $_ = decode($enc, $_) unless $use_perlio;
    # ....
}
```

### SEE ALSO

*Encode::Encoding*, *Encode::Supported*, *Encode::PerIO*, *encoding*, *perlebcdic*, *"open" in perlfunc*, *perlunicode*, *utf8*, the Perl Unicode Mailing List <perl-unicode@perl.org>