



Full credit is given to the above companies including the Operating System (OS) that this PDF file was generated!

Rocky Enterprise Linux 9.2 Manual Pages on command 'IO::Wrap.3pm'

\$ man IO::Wrap.3pm

IO::Wrap(3pm) User Contributed Perl Documentation IO::Wrap(3pm)

NAME

IO::Wrap - wrap raw filehandles in IO::Handle interface

SYNOPSIS

```
use IO::Wrap;

### Do stuff with any kind of filehandle (including a bare globref), or
### any kind of blessed object that responds to a print() message.
###

sub do_stuff {
    my $fh = shift;

    ### At this point, we have no idea what the user gave us...

    ### a globref? a FileHandle? a scalar filehandle name?

    $fh = wraphandle($fh);

    ### At this point, we know we have an IO::Handle-like object!

    $fh->print("Hey there!");

    ...
}
```

DESCRIPTION

Let's say you want to write some code which does I/O, but you don't want to force the caller to provide you with a FileHandle or IO::Handle object. You want them to be able to

say:

```
do_stuff(*STDOUT);
do_stuff(STDERR);
```

```
do_stuff($some_FileHandle_object);
```

```
do_stuff($some_IO_Handle_object);
```

And even:

```
do_stuff($any_object_with_a_print_method);
```

Sure, one way to do it is to force the caller to use `tiehandle()`. But that puts the burden on them. Another way to do it is to use `IO::Wrap`, which provides you with the following functions:

`wraphandle SCALAR`

This function will take a single argument, and "wrap" it based on what it seems to be...

? A raw scalar filehandle name, like "STDOUT" or "Class::HANDLE". In this case, the filehandle name is wrapped in an `IO::Wrap` object, which is returned.

? A raw filehandle glob, like "*STDOUT". In this case, the filehandle glob is wrapped in an `IO::Wrap` object, which is returned.

? A blessed FileHandle object. In this case, the FileHandle is wrapped in an `IO::Wrap` object if and only if your FileHandle class does not support the "read()" method.

? Any other kind of blessed object, which is assumed to be already conformant to the `IO::Handle` interface. In this case, you just get back that object.

If you get back an `IO::Wrap` object, it will obey a basic subset of the `IO::` interface.

That is, the following methods (note: I said methods, not named operators) should work on the thing you get back:

`close`

`getline`

`getlines`

`print ARGS...`

`read BUFFER,NBYTES`

`seek POS,WHENCE`

`tell`

NOTES

Clearly, when wrapping a raw external filehandle (like `*STDOUT`), I didn't want to close the file descriptor when the "wrapper" object is destroyed... since the user might not appreciate that! Hence, there's no `DESTROY` method in this class.

When wrapping a FileHandle object, however, I believe that Perl will invoke the FileHandle::DESTROY when the last reference goes away, so in that case, the filehandle is closed if the wrapped FileHandle really was the last reference to it.

WARNINGS

This module does not allow you to wrap filehandle names which are given as strings that lack the package they were opened in. That is, if a user opens FOO in package Foo, they must pass it to you either as "*FOO" or as "Foo::FOO". However, "STDIN" and friends will work just fine.

VERSION

\$Id: Wrap.pm,v 1.2 2005/02/10 21:21:53 dfs Exp \$

AUTHOR

Primary Maintainer

Dianne Skoll (dfs@roaringpenguin.com).

Original Author

Eryq (eryq@zeegee.com). President, ZeeGee Software Inc (<http://www.zeegee.com>).

perl v5.28.1

2019-02-28

IO::Wrap(3pm)