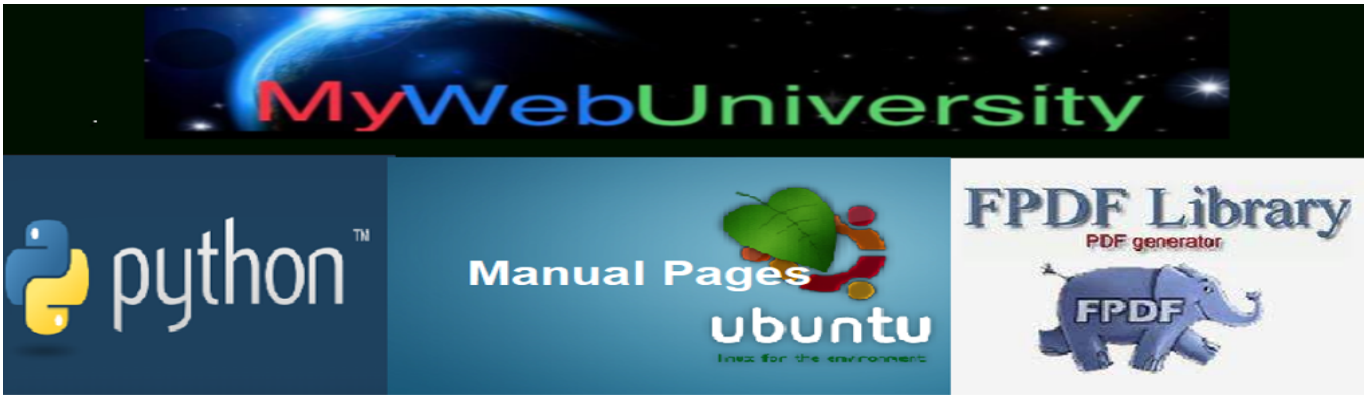


Full credit is given to the above companies including the Operating System (OS) that this PDF file was generated!

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

$ man IO::WrapTie.3pm

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

NAME

IO::WrapTie - wrap tieable objects in IO::Handle interface

This is currently Alpha code, released for comments.

Please give me your feedback!

SYNOPSIS

First of all, you'll need tie(), so:

```
require 5.004;
```

Function interface (experimental). Use this with any existing class...

```
use IO::WrapTie;
use FooHandle;                  ### implements TIEHANDLE interface
### Suppose we want a "FooHandle->new(&FOO_RDWR, 2)".
### We can instead say...
$FH = wraptie('FooHandle', &FOO_RDWR, 2);
### Now we can use...
print $FH "Hello, ";            ### traditional operator syntax...
$FH->print("world!\n");          ### ...and OO syntax as well!
```

OO interface (preferred). You can inherit from the IO::WrapTie::Slave mixin to get a nifty "new_tie()" constructor...

```
#------------------------------
package FooHandle;                      ### a class which can TIEHANDLE
use IO::WrapTie;
@ISA = qw(IO::WrapTie::Slave);          ### inherit new_tie()
```

```
    ...
    #------------------------------
    package main;
    $FH = FooHandle->new_tie(&FOO_RDWR, 2);   ### $FH is an IO::WrapTie::Master
    print $FH "Hello, ";                      ### traditional operator syntax
    $FH->print("world!\n");                   ### OO syntax
```

See IO::Scalar as an example. It also shows you how to create classes which work both with and without 5.004.

DESCRIPTION

Suppose you have a class "FooHandle", where...

- FooHandle does not inherit from IO::Handle; that is, it performs filehandle-like I/O, but to something other than an underlying file descriptor. Good examples are IO::Scalar (for printing to a string) and IO::Lines (for printing to an array of lines).
- FooHandle implements the TIEHANDLE interface (see perltie); that is, it provides methods TIEHANDLE, GETC, PRINT, PRINTF, READ, and READLINE.
- FooHandle implements the traditional OO interface of FileHandle and IO::Handle; i.e., it contains methods like getline(), read(), print(), seek(), tell(), eof(), etc.

Normally, users of your class would have two options:

- Use only OO syntax, and forsake named I/O operators like 'print'.
- Use with tie, and forsake treating it as a first-class object (i.e., class-specific methods can only be invoked through the underlying object via tied()... giving the object a "split personality").

But now with IO::WrapTie, you can say:

```
    $WT = wraptie('FooHandle', &FOO_RDWR, 2);
    $WT->print("Hello, world\n");   ### OO syntax
    print $WT "Yes!\n";             ### Named operator syntax too!
    $WT->weird_stuff;               ### Other methods!
```

And if you're authoring a class like FooHandle, just have it inherit from "IO::WrapTie::Slave" and that first line becomes even prettier:

```
    $WT = FooHandle->new_tie(&FOO_RDWR, 2);
```

The bottom line: now, almost any class can look and work exactly like an IO::Handle... and be used both with OO and non-OO filehandle syntax.

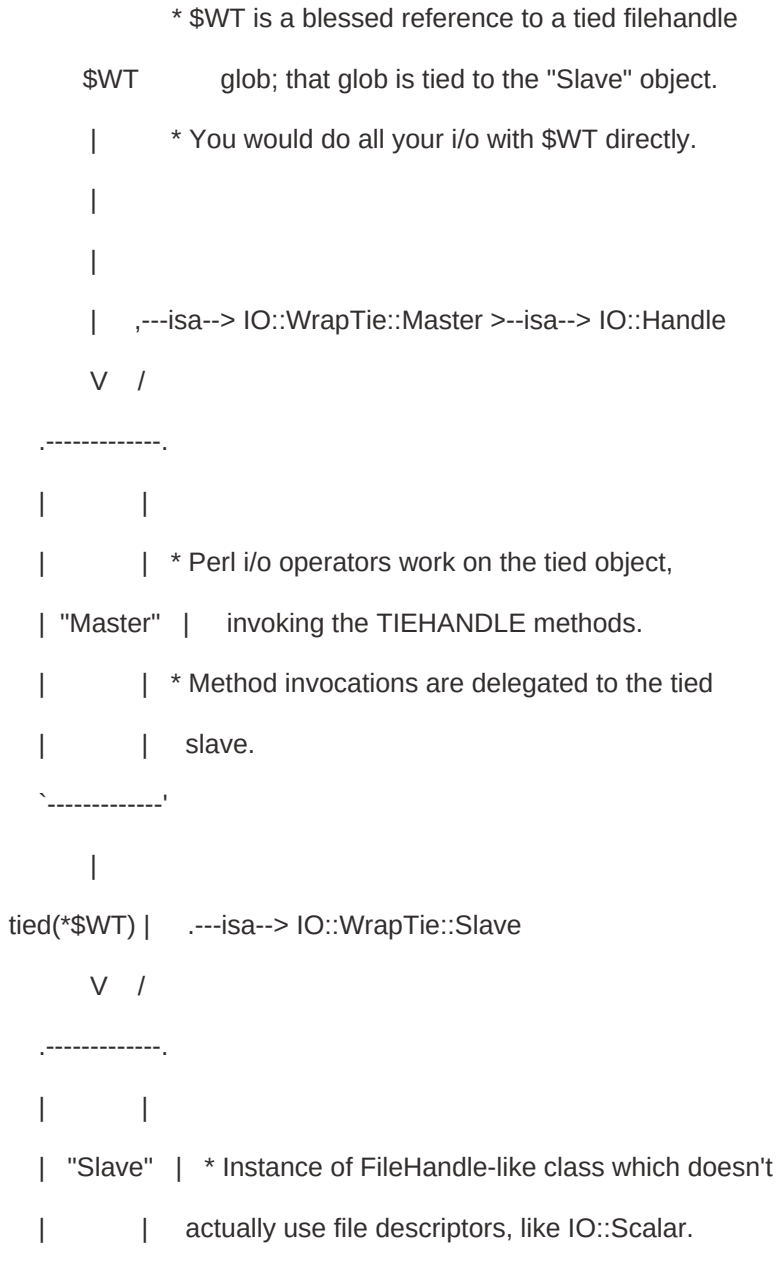
HOW IT ALL WORKS

The data structures

Consider this example code, using classes in this distribution:

```
use IO::Scalar;
use IO::WrapTie;
$WT = wraptie('IO::Scalar',\$s);
print $WT "Hello, ";
$WT->print("world!\n");
```

In it, the wraptie() function creates a data structure as follows:



NOTE: just as an IO::Handle is really just a blessed reference to a traditional filehandle glob... so also, an IO::WrapTie::Master is really just a blessed reference to a filehandle glob which has been tied to some "slave" class.

How wraptie() works

1. The call to function "wraptie(SLAVECLASS, TIEARGS...)" is passed onto "IO::WrapTie::Master::new()". Note that class IO::WrapTie::Master is a subclass of IO::Handle.
2. The "IO::WrapTie::Master::new" method creates a new IO::Handle object, reblessed into class IO::WrapTie::Master. This object is the master, which will be returned from the constructor. At the same time...
3. The "new" method also creates the slave: this is an instance of SLAVECLASS which is created by tying the master's IO::Handle to SLAVECLASS via "tie(HANDLE, SLAVECLASS, TIEARGS...)". This call to "tie()" creates the slave in the following manner:
4. Class SLAVECLASS is sent the message "TIEHANDLE(TIEARGS...)"; it will usually delegate this to "SLAVECLASS::new(TIEARGS...)", resulting in a new instance of SLAVECLASS being created and returned.
5. Once both master and slave have been created, the master is returned to the caller.

How I/O operators work (on the master)

Consider using an i/o operator on the master:

```
print $WT "Hello, world!\n";
```

Since the master ($WT) is really a [blessed] reference to a glob, the normal Perl i/o operators like "print" may be used on it. They will just operate on the symbol part of the glob.

Since the glob is tied to the slave, the slave's PRINT method (part of the TIEHANDLE interface) will be automatically invoked.

If the slave is an IO::Scalar, that means IO::Scalar::PRINT will be invoked, and that method happens to delegate to the "print()" method of the same class. So the real work is ultimately done by IO::Scalar::print().

How methods work (on the master)

Consider using a method on the master:

```
$WT->print("Hello, world!\n");
```

Since the master ($WT) is blessed into the class IO::WrapTie::Master, Perl first attempts to find a "print()" method there. Failing that, Perl next attempts to find a "print()"

method in the superclass, IO::Handle. It just so happens that there is such a method; that method merely invokes the "print" i/o operator on the self object... and for that, see above!

But let's suppose we're dealing with a method which isn't part of IO::Handle... for example:

```
my $sref = $WT->sref;
```

In this case, the intuitive behavior is to have the master delegate the method invocation to the slave (now do you see where the designations come from?). This is indeed what happens: IO::WrapTie::Master contains an AUTOLOAD method which performs the delegation. So: when "sref()" can't be found in IO::Handle, the AUTOLOAD method of IO::WrapTie::Master is invoked, and the standard behavior of delegating the method to the underlying slave (here, an IO::Scalar) is done.

Sometimes, to get this to work properly, you may need to create a subclass of IO::WrapTie::Master which is an effective master for your class, and do the delegation there.

NOTES

Why not simply use the object's OO interface?

Because that means forsaking the use of named operators like print(), and you may need to pass the object to a subroutine which will attempt to use those operators:

```
$O = FooHandle->new(&FOO_RDWR, 2);
$O->print("Hello, world\n");  ### OO syntax is okay, BUT....
sub nope { print $_[0] "Nope!\n" }
X  nope($O);                  ### ERROR!!! (not a glob ref)
```

Why not simply use tie()?

Because (1) you have to use tied() to invoke methods in the object's public interface (yuck), and (2) you may need to pass the tied symbol to another subroutine which will attempt to treat it in an OO-way... and that will break it:

```
tie *T, 'FooHandle', &FOO_RDWR, 2;
print T "Hello, world\n";   ### Operator is okay, BUT...
tied(*T)->other_stuff;      ### yuck! AND...
sub nope { shift->print("Nope!\n") }
X  nope(\*T);               ### ERROR!!! (method "print" on unblessed ref)
```

Why a master and slave?

Why not simply write FooHandle to inherit from IO::Handle?

I tried this, with an implementation similar to that of IO::Socket. The problem is that the whole point is to use this with objects that don't have an underlying file/socket descriptor.. Subclassing IO::Handle will work fine for the OO stuff, and fine with named operators if you tie()... but if you just attempt to say:

```
$IO = FooHandle->new(&FOO_RDWR, 2);
print $IO "Hello!\n";
```

you get a warning from Perl like:

```
Filehandle GEN001 never opened
```

because it's trying to do system-level i/o on an (unopened) file descriptor. To avoid this, you apparently have to tie() the handle... which brings us right back to where we started! At least the IO::WrapTie mixin lets us say:

```
$IO = FooHandle->new_tie(&FOO_RDWR, 2);
print $IO "Hello!\n";
```

and so is not too bad. ":-)"

WARNINGS

Remember: this stuff is for doing FileHandle-like i/o on things without underlying file descriptors. If you have an underlying file descriptor, you're better off just inheriting from IO::Handle.

Be aware that new_tie() always returns an instance of a kind of IO::WrapTie::Master... it does not return an instance of the i/o class you're tying to!

Invoking some methods on the master object causes AUTOLOAD to delegate them to the slave object... so it looks like you're manipulating a "FooHandle" object directly, but you're not.

I have not explored all the ramifications of this use of tie(). Here there be dragons.

VERSION

$Id: WrapTie.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).