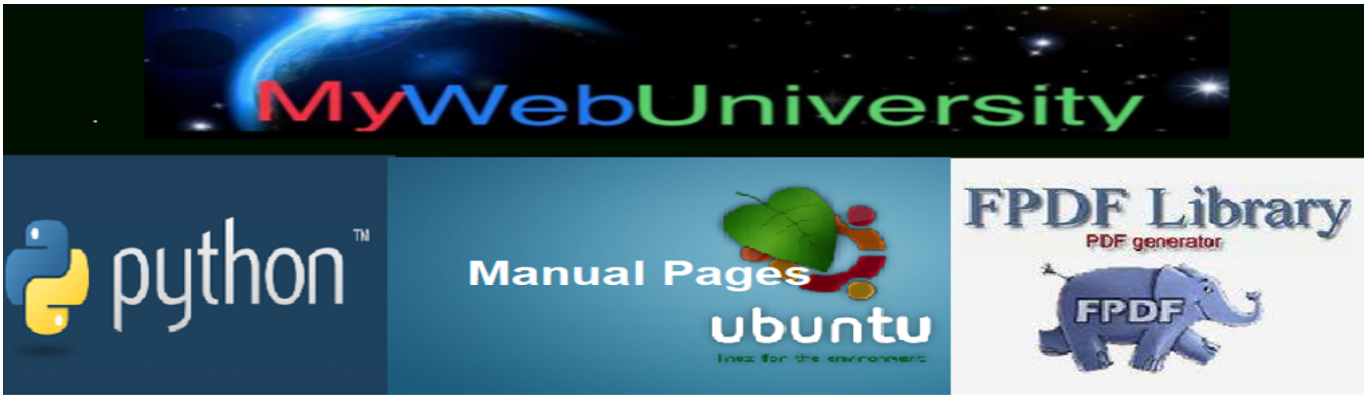

Rocky Enterprise Linux 9.2 Manual Pages on command 'PerlIO::via.3perl'

$ man PerlIO::via.3perl

NAME

PerlIO::via - Helper class for PerlIO layers implemented in perl

SYNOPSIS

```
use PerlIO::via::Layer;
open($fh,"<:via(Layer)",...);

use Some::Other::Package;
open($fh,">:via(Some::Other::Package)",...);
```

DESCRIPTION

The PerlIO::via module allows you to develop PerlIO layers in Perl, without having to go into the nitty gritty of programming C with XS as the interface to Perl.

One example module, PerlIO::via::QuotedPrint, is included with Perl 5.8.0, and more example modules are available from CPAN, such as PerlIO::via::StripHTML and PerlIO::via::Base64. The PerlIO::via::StripHTML module for instance, allows you to say:

```
use PerlIO::via::StripHTML;
open( my $fh, "<:via(StripHTML)", "index.html" );
my @line = <$fh>;
```

to obtain the text of an HTML-file in an array with all the HTML-tags automagically removed.

Please note that if the layer is created in the PerlIO::via:: namespace, it does not have to be fully qualified. The PerlIO::via module will prefix the PerlIO::via:: namespace if the specified modulename does not exist as a fully qualified module name.

EXPECTED METHODS

To create a Perl module that implements a PerlIO layer in Perl (as opposed to in C using XS as the interface to Perl), you need to supply some of the following subroutines. It is recommended to create these Perl modules in the PerlIO::via:: namespace, so that they can easily be located on CPAN and use the default namespace feature of the PerlIO::via module itself.

Please note that this is an area of recent development in Perl and that the interface described here is therefore still subject to change (and hopefully will have better documentation and more examples).

In the method descriptions below $fh will be a reference to a glob which can be treated as a perl file handle. It refers to the layer below. $fh is not passed if the layer is at the bottom of the stack, for this reason and to maintain some level of "compatibility" with TIEHANDLE classes it is passed last.

$class->PUSHED([$mode,[$fh]])

Should return an object or the class, or -1 on failure. (Compare TIEHANDLE.) The arguments are an optional mode string ("r", "w", "w+", ...) and a filehandle for the PerlIO layer below. Mandatory.

When the layer is pushed as part of an "open" call, "PUSHED" will be called before the actual open occurs, whether that be via "OPEN", "SYSOPEN", "FDOPEN" or by letting a lower layer do the open.

$obj->POPPED([$fh])

Optional - called when the layer is about to be removed.

$obj->UTF8($belowFlag,[$fh])

Optional - if present it will be called immediately after PUSHED has returned. It should return a true value if the layer expects data to be UTF-8 encoded. If it returns true, the result is as if the caller had done

":via(YourClass):utf8"

If not present or if it returns false, then the stream is left with the UTF-8 flag clear. The $belowFlag argument will be true if there is a layer below and that layer was expecting UTF-8.

$obj->OPEN($path,$mode,[$fh])

Optional - if not present a lower layer does the open. If present, called for normal opens after the layer is pushed. This function is subject to change as there is no easy way to get a lower layer to do the open and then regain control.

$obj->BINMODE([$fh])

Optional - if not present the layer is popped on binmode($fh) or when ":raw" is pushed. If present it should return 0 on success, -1 on error, or undef to pop the layer.

$obj->FDOPEN($fd,[$fh])

Optional - if not present a lower layer does the open. If present, called after the layer is pushed for opens which pass a numeric file descriptor. This function is subject to change as there is no easy way to get a lower layer to do the open and then regain control.

$obj->SYSOPEN($path,$imode,$perm,[$fh])

Optional - if not present a lower layer does the open. If present, called after the layer is pushed for sysopen style opens which pass a numeric mode and permissions.

This function is subject to change as there is no easy way to get a lower layer to do the open and then regain control.

$obj->FILENO($fh)

Returns a numeric value for a Unix-like file descriptor. Returns -1 if there isn't one. Optional. Default is fileno($fh).

$obj->READ($buffer,$len,$fh)

Returns the number of octets placed in $buffer (must be less than or equal to $len). Optional. Default is to use FILL instead.

$obj->WRITE($buffer,$fh)

Returns the number of octets from $buffer that have been successfully written.

$obj->FILL($fh)

Should return a string to be placed in the buffer. Optional. If not provided, must provide READ or reject handles open for reading in PUSHED.

$obj->CLOSE($fh)

Should return 0 on success, -1 on error. Optional.

$obj->SEEK($posn,$whence,$fh)

Should return 0 on success, -1 on error. Optional. Default is to fail, but that is likely to be changed in future.

$obj->TELL($fh)

Returns file position. Optional. Default to be determined.

$obj->UNREAD($buffer,$fh)

Returns the number of octets from $buffer that have been successfully saved to be returned on future FILL/READ calls. Optional. Default is to push data into a temporary layer above this one.

$obj->FLUSH($fh)

Flush any buffered write data. May possibly be called on readable handles too.

Should return 0 on success, -1 on error.

$obj->SETLINEBUF($fh)

Optional. No return.

$obj->CLEARERR($fh)

Optional. No return.

$obj->ERROR($fh)

Optional. Returns error state. Default is no error until a mechanism to signal error (die?) is worked out.

$obj->EOF($fh)

Optional. Returns end-of-file state. Default is a function of the return value of FILL or READ.

EXAMPLES

Check the PerlIO::via:: namespace on CPAN for examples of PerlIO layers implemented in Perl. To give you an idea how simple the implementation of a PerlIO layer can look, a simple example is included here.

Example - a Hexadecimal Handle

Given the following module, PerlIO::via::Hex :

```
package PerlIO::via::Hex;

sub PUSHED
{
 my ($class,$mode,$fh) = @_;
 # When writing we buffer the data
 my $buf = '';
```

```
 return bless \$buf,$class;
}

sub FILL
{
 my ($obj,$fh) = @_;
 my $line = <$fh>;
 return (defined $line) ? pack("H*", $line) : undef;
}

sub WRITE
{
 my ($obj,$buf,$fh) = @_;
 $$obj .= unpack("H*", $buf);
 return length($buf);
}

sub FLUSH
{
 my ($obj,$fh) = @_;
 print $fh $$obj or return -1;
 $$obj = '';
 return 0;
}

1;
```

The following code opens up an output handle that will convert any output to a hexadecimal dump of the output bytes: for example "A" will be converted to "41" (on ASCII-based machines, on EBCDIC platforms the "A" will become "c1")

```
use PerlIO::via::Hex;
open(my $fh, ">:via(Hex)", "foo.hex");
```

and the following code will read the hexdump in and convert it on the fly back into bytes:

```
open(my $fh, "<:via(Hex)", "foo.hex");
```