NAME

IO::Scalar – IO:: interface for reading/writing a scalar

SYNOPSIS

Perform I/O on strings, using the basic OO interface...

```
use 5.005;
use IO::Scalar;
$data = "My message:\n";

### Open a handle on a string, and append to it:
$SH = new IO::Scalar \$data;
$SH->print("Hello");
$SH->print(", world!\nBye now!\n");
print "The string is now: ", $data, "\n";

### Open a handle on a string, read it line-by-line, then close it:
$SH = new IO::Scalar \$data;
while (defined($_ = $SH->getline)) {
    print "Got line: $_";
}
$SH->close;

### Open a handle on a string, and slurp in all the lines:
$SH = new IO::Scalar \$data;
print "All lines:\n", $SH->getlines;

### Get the current position (either of two ways):
$pos = $SH->getpos;
$offset = $SH->tell;

### Set the current position (either of two ways):
$SH->setpos($pos);
$SH->seek($offset, 0);

### Open an anonymous temporary scalar:
$SH = new IO::Scalar;
$SH->print("Hi there!");
print "I printed: ", ${$SH->sref}, "\n";      ### get at value
```

Don't like OO for your I/O? No problem. Thanks to the magic of an invisible **tie()**, the following now works out of the box, just as it does with IO::Handle:

```
use 5.005;
use IO::Scalar;
$data = "My message:\n";

### Open a handle on a string, and append to it:
$SH = new IO::Scalar \$data;
print $SH "Hello";
print $SH ", world!\nBye now!\n";
print "The string is now: ", $data, "\n";

### Open a handle on a string, read it line-by-line, then close it:
$SH = new IO::Scalar \$data;
while (<$SH>) {
    print "Got line: $_";
```

```
}
close $SH;

### Open a handle on a string, and slurp in all the lines:
$SH = new IO::Scalar \$data;
print "All lines:\n", <$SH>;

### Get the current position (WARNING: requires 5.6):
$offset = tell $SH;

### Set the current position (WARNING: requires 5.6):
seek $SH, $offset, 0;

### Open an anonymous temporary scalar:
$SH = new IO::Scalar;
print $SH "Hi there!";
print "I printed: ", ${$SH->sref}, "\n";      ### get at value
```

And for you folks with 1.x code out there: the old **tie()** style still works, though this is *unnecessary and deprecated*:

```
use IO::Scalar;

### Writing to a scalar...
my $s;
tie *OUT, 'IO::Scalar', \$s;
print OUT "line 1\nline 2\n", "line 3\n";
print "String is now: $s\n"

### Reading and writing an anonymous scalar...
tie *OUT, 'IO::Scalar';
print OUT "line 1\nline 2\n", "line 3\n";
tied(OUT)->seek(0,0);
while (<OUT>) {
    print "Got line: ", $_;
}
```

Stringification works, too!

```
my $SH = new IO::Scalar \$data;
print $SH "Hello, ";
print $SH "world!";
print "I printed: $SH\n";
```

DESCRIPTION

This class is part of the IO::Stringy distribution; see IO::Stringy for change log and general information.

The IO::Scalar class implements objects which behave just like IO::Handle (or FileHandle) objects, except that you may use them to write to (or read from) scalars. These handles are automatically tiehandle'd (though please see "WARNINGS" for information relevant to your Perl version).

Basically, this:

```
my $s;
$SH = new IO::Scalar \$s;
$SH->print("Hel", "lo, ");         ### OO style
$SH->print("world!\n");            ### ditto
```

Or this:

```
my $s;
$SH = tie *OUT, 'IO::Scalar', \$s;
print OUT "Hel", "lo, ";           ### non-OO style
print OUT "world!\n";              ### ditto
```

Causes `$s` to be set to:

```
"Hello, world!\n"
```

PUBLIC INTERFACE

Construction

new [ARGS...]
: *Class method.* Return a new, unattached scalar handle. If any arguments are given, they're sent to **open()**.

open [SCALARREF]
: *Instance method.* Open the scalar handle on a new scalar, pointed to by SCALARREF. If no SCALARREF is given, a ''private'' scalar is created to hold the file data.

  Returns the self object on success, undefined on error.

opened
: *Instance method.* Is the scalar handle opened on something?

close
: *Instance method.* Disassociate the scalar handle from its underlying scalar. Done automatically on destroy.

Input and output

flush
: *Instance method.* No-op, provided for OO compatibility.

fileno
: *Instance method.* No-op, returns undef

getc
: *Instance method.* Return the next character, or undef if none remain.

getline
: *Instance method.* Return the next line, or undef on end of string. Can safely be called in an array context. Currently, lines are delimited by ''\n''.

getlines
: *Instance method.* Get all remaining lines. It will **croak()** if accidentally called in a scalar context.

print ARGS...
: *Instance method.* Print ARGS to the underlying scalar.

  Warning: this continues to always cause a seek to the end of the string, but if you perform **seek()**s and **tell()**s, it is still safer to explicitly seek-to-end before subsequent **print()**s.

read BUF, NBYTES, [OFFSET]
: *Instance method.* Read some bytes from the scalar. Returns the number of bytes actually read, 0 on end-of-file, undef on error.

write BUF, NBYTES, [OFFSET]
: *Instance method.* Write some bytes to the scalar.

sysread BUF, LEN, [OFFSET]
: *Instance method.* Read some bytes from the scalar. Returns the number of bytes actually read, 0 on end-of-file, undef on error.

syswrite BUF, NBYTES, [OFFSET]
: *Instance method.* Write some bytes to the scalar.

Seeking/telling and other attributes

autoflush
: *Instance method.* No-op, provided for OO compatibility.

binmode
: *Instance method.* No-op, provided for OO compatibility.

clearerr
: *Instance method.* Clear the error and EOF flags. A no-op.

eof *Instance method.* Are we at end of file?

seek OFFSET, WHENCE
: *Instance method.* Seek to a given position in the stream.

sysseek OFFSET, WHENCE
: *Instance method.* Identical to `seek OFFSET, WHENCE`, *q.v.*

tell *Instance method.* Return the current position in the stream, as a numeric offset.

setpos POS
: *Instance method.* Set the current position, using the opaque value returned by `getpos()`.

getpos
: *Instance method.* Return the current position in the string, as an opaque object.

sref *Instance method.* Return a reference to the underlying scalar.

WARNINGS

Perl's TIEHANDLE spec was incomplete prior to 5.005_57; it was missing support for `seek()`, `tell()`, and `eof()`. Attempting to use these functions with an IO::Scalar will not work prior to 5.005_57. IO::Scalar will not have the relevant methods invoked; and even worse, this kind of bug can lie dormant for a while. If you turn warnings on (via `$^W` or `perl -w`), and you see something like this...

```
attempt to seek on unopened filehandle
```

...then you are probably trying to use one of these functions on an IO::Scalar with an old Perl. The remedy is to simply use the OO version; e.g.:

```
$SH->seek(0,0);     ### GOOD: will work on any 5.005
seek($SH,0,0);      ### WARNING: will only work on 5.005_57 and beyond
```

VERSION

`$Id:` Scalar.pm,v 1.6 2005/02/10 21:21:53 dfs Exp $

AUTHORS

Primary Maintainer

Dianne Skoll (*dfs@roaringpenguin.com*).

Principal author

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

Other contributors

The full set of contributors always includes the folks mentioned in "CHANGE LOG" in IO::Stringy. But just the same, special thanks to the following individuals for their invaluable contributions (if I've forgotten or misspelled your name, please email me!):

Andy Glew, for contributing `getc()`.

Brandon Browning, for suggesting `opened()`.

David Richter, for finding and fixing the bug in `PRINTF()`.

Eric L. Brine, for his offset-using **read()** and **write()** implementations.

Richard Jones, for his patches to massively improve the performance of `getline()` and add `sysread` and `syswrite`.

B. K. Oxley (binkley), for stringification and inheritance improvements, and sundry good ideas.

Doug Wilson, for the IO::Handle inheritance and automatic tie-ing.

SEE ALSO

IO::String, which is quite similar but which was designed more-recently and with an IO::Handle–like interface in mind, so you could mix OO– and native-filehandle usage without using **tied()**.

Note: as of version 2.x, these classes all work like their IO::Handle counterparts, so we have comparable functionality to IO::String.