NAME

IO::ScalarArray – IO:: interface for reading/writing an array of scalars

SYNOPSIS

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

```
use IO::ScalarArray;
@data = ("My mes", "sage:\n");

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

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

### Open a handle on an array, and slurp in all the lines:
$AH = new IO::ScalarArray \@data;
print "All lines:\n", $AH->getlines;

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

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

### Open an anonymous temporary array:
$AH = new IO::ScalarArray;
$AH->print("Hi there!");
print "I printed: ", @{$AH->aref}, "\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 IO::ScalarArray;
@data = ("My mes", "sage:\n");

### Open a handle on an array, and append to it:
$AH = new IO::ScalarArray \@data;
print $AH "Hello";
print $AH ", world!\nBye now!\n";
print "The array is now: ", @data, "\n";

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

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

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

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

### Open an anonymous temporary scalar:
$AH = new IO::ScalarArray;
print $AH "Hi there!";
print "I printed: ", @{$AH->aref}, "\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::ScalarArray;

### Writing to a scalar...
my @a;
tie *OUT, 'IO::ScalarArray', \@a;
print OUT "line 1\nline 2\n", "line 3\n";
print "Array is now: ", @a, "\n"

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

DESCRIPTION

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

The IO::ScalarArray class implements objects which behave just like IO::Handle (or FileHandle) objects, except that you may use them to write to (or read from) arrays of scalars. Logically, an array of scalars defines an in-core "file" whose contents are the concatenation of the scalars in the array. The handles created by this class are automatically tiehandle'd (though please see "WARNINGS" for information relevant to your Perl version).

For writing large amounts of data with individual **print()** statements, this class is likely to be more efficient than IO::Scalar.

Basically, this:

```
my @a;
$AH = new IO::ScalarArray \@a;
$AH->print("Hel", "lo, ");          ### OO style
$AH->print("world!\n");             ### ditto
```

Or this:

```
my @a;
$AH = new IO::ScalarArray \@a;
print $AH "Hel", "lo, ";           ### non-OO style
print $AH "world!\n";              ### ditto
```

Causes `@a` to be set to the following array of 3 strings:

```
( "Hel" ,
  "lo, " ,
  "world!\n" )
```

See IO::Scalar and compare with this class.

PUBLIC INTERFACE

Construction

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

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

  Returns the self object on success, undefined on error.

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

close
: *Instance method.* Disassociate the array handle from its underlying array. 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. This does a **read**(1), which is somewhat costly.

getline
: *Instance method.* Return the next line, or undef on end of data. 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 array.

  Currently, this always causes a ''seek to the end of the array'' and generates a new array entry. This may change in the future.

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

write BUF, NBYTES, [OFFSET];
: *Instance method.* Write some bytes into the array.

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 POS,WHENCE
: *Instance method.* Seek to a given position in the stream. Only a WHENCE of 0 (SEEK_SET) is supported.

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

setpos POS
: *Instance method.* Seek to a given position in the array, using the opaque **getpos()** value. Don't expect this to be a number.

getpos
: *Instance method.* Return the current position in the array, as an opaque value. Don't expect this to be a number.

aref
: *Instance method.* Return a reference to the underlying array.

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::ScalarArray will not work prior to 5.005_57. IO::ScalarArray 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::ScalarArray with an old Perl. The remedy is to simply use the OO version; e.g.:

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

VERSION

`$Id:` ScalarArray.pm,v 1.7 2005/02/10 21:21:53 dfs Exp $

AUTHOR

Primary Maintainer

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

Principal author

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

Other contributors

Thanks to the following individuals for their invaluable contributions (if I've forgotten or misspelled your name, please email me!):

Andy Glew, for suggesting `getc()`.

Brandon Browning, for suggesting `opened()`.

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

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