

NAME

FileHandle - supply object methods for filehandles

SYNOPSIS

```
use FileHandle;

$fh = new FileHandle;
if ($fh->open("< file")) {
    print <$fh>;
    $fh->close;
}

$fh = new FileHandle "> FOO";
if (defined $fh) {
    print $fh "bar\n";
    $fh->close;
}

$fh = new FileHandle "file", "r";
if (defined $fh) {
    print <$fh>;
    undef $fh;      # automatically closes the file
}

$fh = new FileHandle "file", O_WRONLY|O_APPEND;
if (defined $fh) {
    print $fh "corge\n";
    undef $fh;      # automatically closes the file
}

$pos = $fh->getpos;
$fh->setpos($pos);

$fh->setvbuf($buffer_var, _IOLBF, 1024);

($readfh, $writefh) = FileHandle::pipe;

autoflush STDOUT 1;
```

DESCRIPTION

NOTE: This class is now a front-end to the IO::* classes.

`FileHandle::new` creates a `FileHandle`, which is a reference to a newly created symbol (see the `Symbol` package). If it receives any parameters, they are passed to `FileHandle::open`; if the open fails, the `FileHandle` object is destroyed. Otherwise, it is returned to the caller.

`FileHandle::new_from_fd` creates a `FileHandle` like `new` does. It requires two parameters, which are passed to `FileHandle::fdopen`; if the `fdopen` fails, the `FileHandle` object is destroyed. Otherwise, it is returned to the caller.

`FileHandle::open` accepts one parameter or two. With one parameter, it is just a front end for the built-in `open` function. With two parameters, the first parameter is a filename that may include whitespace or other special characters, and the second parameter is the open mode, optionally

followed by a file permission value.

If `FileHandle::open` receives a Perl mode string ("`>`", "`+<`", etc.) or a POSIX `fopen()` mode string ("`w`", "`r+`", etc.), it uses the basic Perl `open` operator.

If `FileHandle::open` is given a numeric mode, it passes that mode and the optional permissions value to the Perl `sysopen` operator. For convenience, `FileHandle::import` tries to import the `O_XXX` constants from the `Fcntl` module. If dynamic loading is not available, this may fail, but the rest of `FileHandle` will still work.

`FileHandle::fdopen` is like `open` except that its first parameter is not a filename but rather a file handle name, a `FileHandle` object, or a file descriptor number.

If the C functions `fgetpos()` and `fsetpos()` are available, then `FileHandle::getpos` returns an opaque value that represents the current position of the `FileHandle`, and `FileHandle::setpos` uses that value to return to a previously visited position.

If the C function `setvbuf()` is available, then `FileHandle::setvbuf` sets the buffering policy for the `FileHandle`. The calling sequence for the Perl function is the same as its C counterpart, including the macros `_IOFBF`, `_IOLBF`, and `_IONBF`, except that the buffer parameter specifies a scalar variable to use as a buffer. **WARNING:** A variable used as a buffer by `FileHandle::setvbuf` must not be modified in any way until the `FileHandle` is closed or until `FileHandle::setvbuf` is called again, or memory corruption may result!

See *perlfunc* for complete descriptions of each of the following supported `FileHandle` methods, which are just front ends for the corresponding built-in functions:

```
close
fileno
getc
gets
eof
clearerr
seek
tell
```

See *perlvar* for complete descriptions of each of the following supported `FileHandle` methods:

```
autoflush
output_field_separator
output_record_separator
input_record_separator
input_line_number
format_page_number
format_lines_per_page
format_lines_left
format_name
format_top_name
format_line_break_characters
format_formfeed
```

Furthermore, for doing normal I/O you might need these:

```
$fh->print
```

See *"print" in perlfunc*.

```
$fh->printf
```

See *"printf" in perlfunc*.

`$fh->getline`

This works like `<$fh>` described in "*I/O Operators*" in *perlop* except that it's more readable and can be safely called in a list context but still returns just one line.

`$fh->getlines`

This works like `<$fh>` when called in a list context to read all the remaining lines in a file, except that it's more readable. It will also croak() if accidentally called in a scalar context.

There are many other functions available since FileHandle is descended from IO::File, IO::Seekable, and IO::Handle. Please see those respective pages for documentation on more functions.

SEE ALSO

The **IO** extension, *perlfunc*, "*I/O Operators*" in *perlop*.