NAME

Fatal - Replace functions with equivalents which succeed or die

SYNOPSIS

```
use Fatal qw(open close);

open(my $fh, "<", $filename); # No need to check errors!

use File::Copy qw(move);
use Fatal qw(move);

move($file1, $file2); # No need to check errors!

sub juggle { . . . }
Fatal->import('juggle');
```

BEST PRACTICE

Fatal has been obsoleted by the new autodie pragma. Please use autodie in preference to `Fatal`. autodie supports lexical scoping, throws real exception objects, and provides much nicer error messages.

The use of `:void` with Fatal is discouraged.

DESCRIPTION

`Fatal` provides a way to conveniently replace functions which normally return a false value when they fail with equivalents which raise exceptions if they are not successful. This lets you use these functions without having to test their return values explicitly on each call. Exceptions can be caught using `eval{}`. See perlfunc(1) and perlvar(1) for details.

The do-or-die equivalents are set up simply by calling Fatal's `import` routine, passing it the names of the functions to be replaced. You may wrap both user-defined functions and overridable CORE operators (except `exec`, `system`, `print`, or any other built-in that cannot be expressed via prototypes) in this way.

If the symbol `:void` appears in the import list, then functions named later in that import list raise an exception only when these are called in void context—that is, when their return values are ignored. For example

```
use Fatal qw/:void open close/;

# properly checked, so no exception raised on error
if (not open(my $fh, '<', '/bogotic') {
warn "Can't open /bogotic: $!";
}

# not checked, so error raises an exception
close FH;
```

The use of `:void` is discouraged, as it can result in exceptions not being thrown if you *accidentally* call a method without void context. Use autodie instead if you need to be able to disable autodying/Fatal behaviour for a small block of code.

DIAGNOSTICS

Bad subroutine name for Fatal: `%s`

You've called `Fatal` with an argument that doesn't look like a subroutine name, nor a switch that this version of Fatal understands.

`%s` is not a Perl subroutine

You've asked `Fatal` to try and replace a subroutine which does not exist, or has not yet been defined.

`%s` is neither a builtin, nor a Perl subroutine
: You've asked `Fatal` to replace a subroutine, but it's not a Perl built-in, and `Fatal` couldn't find it as a regular subroutine. It either doesn't exist or has not yet been defined.

Cannot make the non-overridable `%s` fatal
: You've tried to use `Fatal` on a Perl built-in that can't be overridden, such as `print` or `system`, which means that `Fatal` can't help you, although some other modules might. See the "SEE ALSO" section of this documentation.

Internal error: `%s`
: You've found a bug in `Fatal`. Please report it using the `perlbug(1)` command.

BUGS

`Fatal` clobbers the context in which a function is called and always makes it a scalar context, except when the `:void` tag is used. This problem does not exist in autodie.

"Used only once" warnings can be generated when `autodie` or `Fatal` is used with package filehandles (eg, `FILE`). It's strongly recommended you use scalar filehandles instead.

AUTHOR

Original module by Lionel Cons (CERN).

Prototype updates by Ilya Zakharevich <ilya@math.ohio-state.edu>.

autodie support, bugfixes, extended diagnostics, `system` support, and major overhauling by Paul Fenwick <pjf@perltraining.com.au>

LICENSE

This module is free software, you may distribute it under the same terms as Perl itself.

SEE ALSO

autodie for a nicer way to use lexical Fatal.

IPC::System::Simple for a similar idea for calls to `system()` and backticks.