

NAME

autodie - Replace functions with ones that succeed or die with lexical scope

SYNOPSIS

```
use autodie; # Recommended: implies 'use autodie qw(:default)'
```

```
use autodie qw(:all); # Recommended more: defaults and system/exec.
```

```
use autodie qw(open close); # open/close succeed or die
```

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

```
{
no autodie qw(open); # open failures won't die
open(my $fh, "<", $filename); # Could fail silently!
no autodie; # disable all autodies
}
```

DESCRIPTION

```
bIlujDI' yIchegh()Qo'; yIHegh()!
```

It is better to die() than to return() in failure.

-- Klingon programming proverb.

The `autodie` pragma provides a convenient way to replace functions that normally return false on failure with equivalents that throw an exception on failure.

The `autodie` pragma has *lexical scope*, meaning that functions and subroutines altered with `autodie` will only change their behaviour until the end of the enclosing block, file, or `eval`.

If `system` is specified as an argument to `autodie`, then it uses `IPC::System::Simple` to do the heavy lifting. See the description of that module for more information.

EXCEPTIONS

Exceptions produced by the `autodie` pragma are members of the `autodie::exception` class. The preferred way to work with these exceptions under Perl 5.10 is as follows:

```
use feature qw(switch);

eval {
use autodie;

open(my $fh, '<', $some_file);

my @records = <$fh>;

# Do things with @records...

close($fh);

};

given ($@) {
when (undef) { say "No error"; }
when ('open') { say "Error from open"; }
when (':io') { say "Non-open, IO error."; }
when (':all') { say "All other autodie errors." }
default { say "Not an autodie error at all." }
```

```
}

```

Under Perl 5.8, the `given/when` structure is not available, so the following structure may be used:

```
eval {
use autodie;

open(my $fh, '<', $some_file);

my @records = <$fh>;

# Do things with @records...

close($fh);
};

if ($@ and $@->isa('autodie::exception')) {
if ($@->matches('open')) { print "Error from open\n"; }
if ($@->matches(':io' )) { print "Non-open, IO error."; }
} elsif ($@) {
# A non-autodie exception.
}
```

See [autodie::exception](#) for further information on interrogating exceptions.

CATEGORIES

Autodie uses a simple set of categories to group together similar built-ins. Requesting a category type (starting with a colon) will enable autodie for all built-ins beneath that category. For example, requesting `:file` will enable autodie for `close`, `fcntl`, `fileno`, `open` and `sysopen`.

The categories are currently:

```
:all
:default
:io
read
seek
sysread
sysseek
syswrite
:dbm
dbmclose
dbmopen
:file
binmode
close
chmod
chown
fcntl
fileno
flock
ioctl
open
sysopen
truncate
:filesystem
chdir
closedir
```

```
opendir
link
mkdir
readlink
rename
rmdir
symlink
unlink
:ipc
pipe
:msg
msgctl
msgget
msgrcv
msgsnd
:semaphore
semctl
semget
semop
:shm
shmctl
shmget
shmread
:socket
accept
bind
connect
getsockopt
listen
recv
send
setsockopt
shutdown
socketpair
:threads
fork
:system
system
exec
```

Note that while the above category `system` is presently a strict hierarchy, this should not be assumed.

A plain use `autodie` implies `use autodie qw(:default)`. Note that `system` and `exec` are not enabled by default. `system` requires the optional `IPC::System::Simple` module to be installed, and enabling `system` or `exec` will invalidate their exotic forms. See “BUGS” below for more details.

The syntax:

```
use autodie qw(:1.994);
```

allows the `:default` list from a particular version to be used. This provides the convenience of using the default methods, but the surety that no behavioral changes will occur if the `autodie` module is upgraded.

`autodie` can be enabled for all of Perl’s built-ins, including `system` and `exec` with:

```
use autodie qw(:all);
```

FUNCTION SPECIFIC NOTES

print

The `autodie` pragma **<does not check calls to `print`>**.

flock

It is not considered an error for `flock` to return false if it fails due to an `EWOULDDBLOCK` (or equivalent) condition. This means one can still use the common convention of testing the return value of `flock` when called with the `LOCK_NB` option:

```
use autodie;

if ( flock($fh, LOCK_EX | LOCK_NB) ) {
    # We have a lock
}
```

Autodying `flock` will generate an exception if `flock` returns false with any other error.

system/exec

The `system` built-in is considered to have failed in the following circumstances:

- The command does not start.
- The command is killed by a signal.
- The command returns a non-zero exit value (but see below).

On success, the autodying form of `system` returns the *exit value* rather than the contents of `$?`.

Additional allowable exit values can be supplied as an optional first argument to autodying `system`:

```
system( [ 0, 1, 2 ], $cmd, @args); # 0,1,2 are good exit values
```

`autodie` uses the `IPC::System::Simple` module to change `system`. See its documentation for further information.

Applying `autodie` to `system` or `exec` causes the exotic forms `system { $cmd } @args` or `exec { $cmd } @args` to be considered a syntax error until the end of the lexical scope. If you really need to use the exotic form, you can call `CORE::system` or `CORE::exec` instead, or use `no autodie qw(system exec)` before calling the exotic form.

GOTCHAS

Functions called in list context are assumed to have failed if they return an empty list, or a list consisting only of a single `undef` element.

DIAGNOSTICS

`:void` cannot be used with lexical scope

The `:void` option is supported in `Fatal`, but not `autodie`. To work around this, `autodie` may be explicitly disabled until the end of the current block with `no autodie`. To disable `autodie` for only a single function (eg, `open`) use `no autodie qw(open)`.

`autodie` performs no checking of called context to determine whether to throw an exception; the explicitness of error handling with `autodie` is a deliberate feature.

No user hints defined for `%s`

You've insisted on hints for user-subroutines, either by pre-pending a `!` to the subroutine name itself, or earlier in the list of arguments to `autodie`. However the subroutine in question does not have any hints available.

See also "DIAGNOSTICS" in `Fatal`.

BUGS

"Used only once" warnings can be generated when `autodie` or `Fatal` is used with package filehandles (eg, `FILE`). Scalar filehandles are strongly recommended instead.

When using `autodie` or `Fatal` with user subroutines, the declaration of those subroutines must appear before the first use of `Fatal` or `autodie`, or have been exported from a module. Attempting to use

`Fatal` or `autodie` on other user subroutines will result in a compile-time error.

Due to a bug in Perl, `autodie` may “lose” any format which has the same name as an autodying built-in or function.

`autodie` may not work correctly if used inside a file with a name that looks like a string eval, such as *eval* (3).

autodie and string eval

Due to the current implementation of `autodie`, unexpected results may be seen when used near or with the string version of `eval`. *None of these bugs exist when using block eval.*

Under Perl 5.8 only, `autodie` *does not* propagate into string `eval` statements, although it can be explicitly enabled inside a string `eval`.

Under Perl 5.10 only, using a string `eval` when `autodie` is in effect can cause the `autodie` behaviour to leak into the surrounding scope. This can be worked around by using a `no autodie` at the end of the scope to explicitly remove `autodie`'s effects, or by avoiding the use of string `eval`.

None of these bugs exist when using block eval. The use of `autodie` with block `eval` is considered good practice.

REPORTING BUGS

Please report bugs via the CPAN Request Tracker at <<http://rt.cpan.org/NoAuth/Bugs.html?Dist=autodie>>.

FEEDBACK

If you find this module useful, please consider rating it on the CPAN Ratings service at <<http://cpanratings.perl.org/rate?distribution=autodie>>

The module author loves to hear how `autodie` has made your life better (or worse). Feedback can be sent to <pjf@perltraining.com.au>.

AUTHOR

Copyright 2008-2009, 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

`Fatal`, [autodie::exception](#), [autodie::hints](#), `IPC::System::Simple`

Perl tips, *autodie* at <<http://perltraining.com.au/tips/2008-08-20.html>>

ACKNOWLEDGEMENTS

Mark Reed and Roland Giersig — Klingon translators.

See the *AUTHORS* file for full credits. The latest version of this file can be found at <<https://github.com/pjf/autodie/tree/master/AUTHORS>>