NAME

autodie::exception - Exceptions from autodying functions.

SYNOPSIS

```
eval {
    use autodie;

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

    ...
};

if (my $E = $@) {
    say "Ooops!  ",$E->caller," had problems: $@";
}
```

DESCRIPTION

When an *autodie* enabled function fails, it generates an `autodie::exception` object. This can be interrogated to determine further information about the error that occurred.

This document is broken into two sections; those methods that are most useful to the end-developer, and those methods for anyone wishing to subclass or get very familiar with `autodie::exception`.

Common Methods

These methods are intended to be used in the everyday dealing of exceptions.

The following assume that the error has been copied into a separate scalar:

```
if ($E = $@) {
    ...
}
```

This is not required, but is recommended in case any code is called which may reset or alter $@.

args

```
my $array_ref = $E->args;
```

Provides a reference to the arguments passed to the subroutine that died.

function

```
my $sub = $E->function;
```

The subroutine (including package) that threw the exception.

file

```
my $file = $E->file;
```

The file in which the error occurred (eg, `myscript.pl` or `MyTest.pm`).

package

```
my $package = $E->package;
```

The package from which the exceptional subroutine was called.

caller

```
my $caller = $E->caller;
```

The subroutine that *called* the exceptional code.

line

```
my $line = $E->line;
```

The line in `$E->file` where the exceptional code was called.

context

```
my $context = $E->context;
```

The context in which the subroutine was called. This can be 'list', 'scalar', or undefined (unknown). It will never be 'void', as `autodie` always captures the return value in one way or another.

return

```
my $return_value = $E->return;
```

The value(s) returned by the failed subroutine. When the subroutine was called in a list context, this will always be a reference to an array containing the results. When the subroutine was called in a scalar context, this will be the actual scalar returned.

errno

```
my $errno = $E->errno;
```

The value of `$!` at the time when the exception occurred.

NOTE: This method will leave the main `autodie::exception` class and become part of a role in the future. You should only call `errno` for exceptions where `$!` would reasonably have been set on failure.

eval_error

```
my $old_eval_error = $E->eval_error;
```

The contents of $@ immediately after autodie triggered an exception. This may be useful when dealing with modules such as *Text::Balanced* that set (but do not throw) $@ on error.

matches

```
if ( $e->matches('open') ) { ... }

if ( $e ~~ 'open' ) { ... }
```

`matches` is used to determine whether a given exception matches a particular role. On Perl 5.10, using smart-match (~~) with an `autodie::exception` object will use `matches` underneath.

An exception is considered to match a string if:

- For a string not starting with a colon, the string exactly matches the package and subroutine that threw the exception. For example, `MyModule::log`. If the string does not contain a package name, `CORE::` is assumed.
- For a string that does start with a colon, if the subroutine throwing the exception *does* that behaviour. For example, the `CORE::open` subroutine does `:file`, `:io` and `:all`.

See *"CATEGORIES" in autodie* for futher information.

Advanced methods

The following methods, while usable from anywhere, are primarily intended for developers wishing to subclass `autodie::exception`, write code that registers custom error messages, or otherwise work closely with the `autodie::exception` model.

register

```
autodie::exception->register( 'CORE::open' => \&mysub );
```

The `register` method allows for the registration of a message handler for a given subroutine. The full subroutine name including the package should be used.

Registered message handlers will receive the `autodie::exception` object as the first parameter.

add_file_and_line

```
say "Problem occurred",$@->add_file_and_line;
```

Returns the string `at %s line %d`, where `%s` is replaced with the filename, and `%d` is replaced with the line number.

Primarily intended for use by format handlers.

stringify

```
say "The error was: ",$@->stringify;
```

Formats the error as a human readable string. Usually there's no reason to call this directly, as it is used automatically if an `autodie::exception` object is ever used as a string.

Child classes can override this method to change how they're stringified.

format_default

```
my $error_string = $E->format_default;
```

This produces the default error string for the given exception, *without using any registered message handlers*. It is primarily intended to be called from a message handler when they have been passed an exception they don't want to format.

Child classes can override this method to change how default messages are formatted.

new

```
my $error = autodie::exception->new(
    args => \@_,
    function => "CORE::open",
    errno => $!,
    context => 'scalar',
    return => undef,
);
```

Creates a new `autodie::exception` object. Normally called directly from an autodying function. The `function` argument is required, its the function we were trying to call that generated the exception. The `args` parameter is optional.

The `errno` value is optional. In versions of `autodie::exception` 1.99 and earlier the code would try to automatically use the current value of `$!`, but this was unreliable and is no longer supported.

Atrributes such as package, file, and caller are determined automatically, and cannot be specified.

SEE ALSO

autodie, *autodie::exception::system*

LICENSE

AUTHOR

Paul Fenwick <pjf@perltraining.com.au>