

1 APR::Error - Perl API for APR/Apache/mod_perl exceptions

1.1 Synopsis

```
eval { $obj->mp_method() };
if ($@ && $ref $@ eq 'APR::Error' && $@ == $some_code) {
    # handle the exception
}
else {
    die $@; # rethrow it
}
```

1.2 Description

`APR::Error` handles APR/Apache/mod_perl exceptions for you, while leaving you in control.

Apache and APR API return a status code for almost all methods, so if you didn't check the return code and handled any possible problems, you may have silent failures which may cause all kind of obscure problems. On the other hand checking the status code after each call is just too much of a kludge and makes quick prototyping/development almost impossible, not talking about the code readability. Having methods return status codes, also complicates the API if you need to return other values.

Therefore to keep things nice and make the API readable we decided to not return status codes, but instead throw exceptions with `APR::Error` objects for each method that fails. If you don't catch those exceptions, everything works transparently - perl will intercept the exception object and `die()` with a proper error message. So you get all the errors logged without doing any work.

Now, in certain cases you don't want to just die, but instead the error needs to be trapped and handled. For example if some IO operation times out, may be it is OK to trap that and try again. If we were to die with an error message, you would have had to match the error message, which is ugly, inefficient and may not work at all if locale error strings are involved. Therefore you need to be able to get the original status code that Apache or APR has generated. And the exception objects give you that if you want to. Moreover the objects contain additional information, such as the function name (in case you were eval'ing several commands in one block), file and line number where that function was invoked from. More attributes could be added in the future.

`APR::Error` uses Perl operator overloading, such that in boolean and numerical contexts, the object returns the status code; in the string context the full error message is returned.

When intercepting exceptions you need to check whether `$@` is an object (reference). If your application uses other exception objects you additionally need to check whether this is a an `APR::Error` object. Therefore most of the time this is enough:

```
eval { $obj->mp_method() };
if ($@ && $ref $@ && $@ == $some_code)
    warn "handled exception: $@";
}
```

But with other, non-mod_perl, exception objects you need to do:

```
eval { $obj->mp_method() };
if ($@ && $ref $@ eq 'APR::Error' && $@ == $some_code)
    warn "handled exception: $@";
}
```

In theory you could even do:

```
eval { $obj->mp_method() };
if ($@ && $@ == $some_code)
    warn "handled exception: $@";
}
```

but it's possible that the method will die with a plain string and not an object, in which case `$@ == $some_code` won't quite work. Remember that mod_perl throws exception objects only when Apache and APR fail, and in a few other special cases of its own (like `exit`).

```
warn "handled exception: $@" if $@ && $ref $@;
```

There are two ways to figure out whether an error fits your case. In most cases you just compare `$@` with an the error constant. For example if a socket has a timeout set and the data wasn't read within the timeout limit a `APR::Const::TIMEUP`

```
use APR::Const -compile => qw(TIMEUP);
$sock->timeout_set(1_000_000); # 1 sec
my $buff;
eval { $sock->recv($buff, BUFF_LEN) };
if ($@ && ref $@ && $@ == APR::Const::TIMEUP) {
}
}
```

However there are situations, where on different Operating Systems a different error code will be returned. In which case to simplify the code you should use the special subroutines provided by the `APR::Status` class. One such condition is socket `recv()` timeout, which on Unix throws the `EAGAIN` error, but on other system it throws a different error. In this case `APR::Status::is_EAGAIN` should be used.

Let's look at a complete example. Here is a code that performs a socket read:

```
my $rlen = $sock->recv(my $buff, 1024);
warn "read $rlen bytes\n";
```

and in certain cases it times out. The code will die and log the reason for the failure, which is fine, but later on you may decide that you want to have another attempt to read before dying and add some fine grained sleep time between attempts, which can be achieved with `select`. Which gives us:

```
use APR::Status ();
# ....
my $tries = 0;
my $buffer;
RETRY: my $rlen = eval { $sock->recv($buffer, SIZE) };
if ($@)
    die $@ unless ref $@ && APR::Status::is_EAGAIN($@);
```

```

    if ($tries++ < 3) {
        # sleep 250msec
        select undef, undef, undef, 0.25;
        goto RETRY;
    }
    else {
        # do something else
    }
}
warn "read $rlen bytes\n"

```

Notice that we handle non-object and non-APR::Error exceptions as well, by simply re-throwing them.

Finally, the class is called APR::Error because it needs to be used outside mod_perl as well, when called from APR applications written in Perl.

1.3 API

1.3.1 *cluck*

`cluck` is an equivalent of `Carp::cluck` that works with `APR::Error` exception objects.

1.3.2 *confess*

`confess` is an equivalent of `Carp::confess` that works with `APR::Error` exception objects.

1.3.3 *strerror*

Convert APR error code to its string representation.

```
$error_str = APR::Error::strerror($rc);
```

- **ret: \$rc (APR::Const status constant)**

The numerical value for the return (error) code

- **ret: \$error_str (string)**

The string error message corresponding to the numerical value inside `$rc`. (Similar to the C function `strerror(3)`)

- **since: 2.0.00**

Example:

Try to retrieve the bucket brigade, and if the return value doesn't indicate success or end of file (usually in protocol handlers) die, but give the user the human-readable version of the error and not just the code.

```
my $rc = $c->input_filters->get_brigade($bb_in,
                                       Apache2::Const::MODE_GETLINE);
if ($rc != APR::Const::SUCCESS && $rc != APR::Const::EOF) {
    my $error = APR::Error::strerror($rc);
    die "get_brigade error: $rc: $error\n";
}
```

It's probably a good idea not to omit the numerical value in the error message, in case the error string is generated with non-English locale.

1.4 See Also

mod_perl 2.0 documentation.

1.5 Copyright

mod_perl 2.0 and its core modules are copyrighted under The Apache Software License, Version 2.0.

1.6 Authors

The mod_perl development team and numerous contributors.

Table of Contents:

1	APR::Error - Perl API for APR/Apache/mod_perl exceptions	1
1.1	Synopsis	2
1.2	Description	2
1.3	API	4
1.3.1	cluck	4
1.3.2	confess	4
1.3.3	strerror	4
1.4	See Also	5
1.5	Copyright	5
1.6	Authors	5