NAME

CGI::Carp - CGI routines for writing to the HTTPD (or other) error log

SYNOPSIS

```
use CGI::Carp;

croak "We're outta here!";
confess "It was my fault: $!";
carp "It was your fault!";
warn "I'm confused";
die  "I'm dying.\n";

use CGI::Carp qw(cluck);
cluck "I wouldn't do that if I were you";

use CGI::Carp qw(fatalsToBrowser);
die "Fatal error messages are now sent to browser";
```

DESCRIPTION

CGI scripts have a nasty habit of leaving warning messages in the error logs that are neither time stamped nor fully identified. Tracking down the script that caused the error is a pain. This fixes that. Replace the usual

```
use Carp;
```

with

```
use CGI::Carp
```

And the standard warn(), die (), croak(), confess() and carp() calls will automagically be replaced with functions that write out nicely time-stamped messages to the HTTP server error log.

For example:

```
[Fri Nov 17 21:40:43 1995] test.pl: I'm confused at test.pl line 3.
[Fri Nov 17 21:40:43 1995] test.pl: Got an error message: Permission
denied.
[Fri Nov 17 21:40:43 1995] test.pl: I'm dying.
```

REDIRECTING ERROR MESSAGES

By default, error messages are sent to STDERR. Most HTTPD servers direct STDERR to the server's error log. Some applications may wish to keep private error logs, distinct from the server's error log, or they may wish to direct error messages to STDOUT so that the browser will receive them.

The `carpout()` function is provided for this purpose. Since carpout() is not exported by default, you must import it explicitly by saying

```
use CGI::Carp qw(carpout);
```

The carpout() function requires one argument, which should be a reference to an open filehandle for writing errors. It should be called in a `BEGIN` block at the top of the CGI application so that compiler errors will be caught. Example:

```
BEGIN {
   use CGI::Carp qw(carpout);
```

```
      open(LOG, ">>/usr/local/cgi-logs/mycgi-log") or
        die("Unable to open mycgi-log: $!\n");
      carpout(LOG);
    }
```

carpout() does not handle file locking on the log for you at this point.

The real STDERR is not closed -- it is moved to CGI::Carp::SAVEERR. Some servers, when dealing with CGI scripts, close their connection to the browser when the script closes STDOUT and STDERR. CGI::Carp::SAVEERR is there to prevent this from happening prematurely.

You can pass filehandles to carpout() in a variety of ways. The "correct" way according to Tom Christiansen is to pass a reference to a filehandle GLOB:

```
      carpout(\*LOG);
```

This looks weird to mere mortals however, so the following syntaxes are accepted as well:

```
      carpout(LOG);
      carpout(main::LOG);
      carpout(main'LOG);
      carpout(\LOG);
      carpout(\'main::LOG');


      ... and so on
```

FileHandle and other objects work as well.

Use of carpout() is not great for performance, so it is recommended for debugging purposes or for moderate-use applications. A future version of this module may delay redirecting STDERR until one of the CGI::Carp methods is called to prevent the performance hit.

MAKING PERL ERRORS APPEAR IN THE BROWSER WINDOW

If you want to send fatal (die, confess) errors to the browser, ask to import the special "fatalsToBrowser" subroutine:

```
      use CGI::Carp qw(fatalsToBrowser);
      die "Bad error here";
```

Fatal errors will now be echoed to the browser as well as to the log. CGI::Carp arranges to send a minimal HTTP header to the browser so that even errors that occur in the early compile phase will be seen. Nonfatal errors will still be directed to the log file only (unless redirected with carpout).

Note that fatalsToBrowser does **not** work with mod_perl version 2.0 and higher.

Changing the default message

By default, the software error message is followed by a note to contact the Webmaster by e-mail with the time and date of the error. If this message is not to your liking, you can change it using the set_message() routine. This is not imported by default; you should import it on the use() line:

```
      use CGI::Carp qw(fatalsToBrowser set_message);
      set_message("It's not a bug, it's a feature!");
```

You may also pass in a code reference in order to create a custom error message. At run time, your code will be called with the text of the error message that caused the script to die. Example:

```
      use CGI::Carp qw(fatalsToBrowser set_message);
```

```
BEGIN {
   sub handle_errors {
      my $msg = shift;
      print "<h1>Oh gosh</h1>";
      print "<p>Got an error: $msg</p>";
   }
   set_message(\&handle_errors);
}
```

In order to correctly intercept compile-time errors, you should call set_message() from within a BEGIN{} block.

DOING MORE THAN PRINTING A MESSAGE IN THE EVENT OF PERL ERRORS

If fatalsToBrowser in conjunction with set_message does not provide you with all of the functionality you need, you can go one step further by specifying a function to be executed any time a script calls "die", has a syntax error, or dies unexpectedly at runtime with a line like "undef->explode();".

```
use CGI::Carp qw(set_die_handler);
BEGIN {
   sub handle_errors {
      my $msg = shift;
      print "content-type: text/html\n\n";
      print "<h1>Oh gosh</h1>";
      print "<p>Got an error: $msg</p>";

      #proceed to send an email to a system administrator,
      #write a detailed message to the browser and/or a log,
      #etc....
   }
   set_die_handler(\&handle_errors);
}
```

Notice that if you use set_die_handler(), you must handle sending HTML headers to the browser yourself if you are printing a message.

If you use set_die_handler(), you will most likely interfere with the behavior of fatalsToBrowser, so you must use this or that, not both.

Using set_die_handler() sets SIG{__DIE__} (as does fatalsToBrowser), and there is only one SIG{__DIE__}. This means that if you are attempting to set SIG{__DIE__} yourself, you may interfere with this module's functionality, or this module may interfere with your module's functionality.

MAKING WARNINGS APPEAR AS HTML COMMENTS

It is now also possible to make non-fatal errors appear as HTML comments embedded in the output of your program. To enable this feature, export the new "warningsToBrowser" subroutine. Since sending warnings to the browser before the HTTP headers have been sent would cause an error, any warnings are stored in an internal buffer until you call the warningsToBrowser() subroutine with a true argument:

```
use CGI::Carp qw(fatalsToBrowser warningsToBrowser);
use CGI qw(:standard);
print header();
warningsToBrowser(1);
```

You may also give a false argument to warningsToBrowser() to prevent warnings from being sent to the browser while you are printing some content where HTML comments are not allowed:

```
warningsToBrowser(0);    # disable warnings
print "<script type=\"text/javascript\"><!--\n";
print_some_javascript_code();
print "//--></script>\n";
warningsToBrowser(1);    # re-enable warnings
```

Note: In this respect warningsToBrowser() differs fundamentally from fatalsToBrowser(), which you should never call yourself!

OVERRIDING THE NAME OF THE PROGRAM

CGI::Carp includes the name of the program that generated the error or warning in the messages written to the log and the browser window. Sometimes, Perl can get confused about what the actual name of the executed program was. In these cases, you can override the program name that CGI::Carp will use for all messages.

The quick way to do that is to tell CGI::Carp the name of the program in its use statement. You can do that by adding "name=cgi_carp_log_name" to your "use" statement. For example:

```
use CGI::Carp qw(name=cgi_carp_log_name);
```

. If you want to change the program name partway through the program, you can use the `set_progname()` function instead. It is not exported by default, you must import it explicitly by saying

```
use CGI::Carp qw(set_progname);
```

Once you've done that, you can change the logged name of the program at any time by calling

```
set_progname(new_program_name);
```

You can set the program back to the default by calling

```
set_progname(undef);
```

Note that this override doesn't happen until after the program has compiled, so any compile-time errors will still show up with the non-overridden program name

CHANGE LOG

1.29 Patch from Peter Whaite to fix the unfixable problem of CGI::Carp not behaving correctly in an eval() context.

1.05 carpout() added and minor corrections by Marc Hedlund <hedlund@best.com> on 11/26/95.

1.06 fatalsToBrowser() no longer aborts for fatal errors within eval() statements.

1.08 set_message() added and carpout() expanded to allow for FileHandle objects.

1.09 set_message() now allows users to pass a code REFERENCE for really custom error messages. croak and carp are now exported by default. Thanks to Gunther Birznieks for the patches.

1.10 Patch from Chris Dean (ctdean@cogit.com) to allow module to run correctly under mod_perl.

1.11 Changed order of > and < escapes.

1.12 Changed die() on line 217 to CORE::die to avoid **-w** warning.

1.13 Added cluck() to make the module orthogonal with Carp. More mod_perl related fixes.

1.20 Patch from Ilmari Karonen (perl@itz.pp.sci.fi): Added warningsToBrowser(). Replaced <CODE>

tags with <PRE> in fatalsToBrowser() output.

1.23 ineval() now checks both $^S and inspects the message for the "eval" pattern (hack alert!) in order to accommodate various combinations of Perl and mod_perl.

1.24 Patch from Scott Gifford (sgifford@suspectclass.com): Add support for overriding program name.

1.26 Replaced CORE::GLOBAL::die with the evil $SIG{__DIE__} because the former isn't working in some people's hands. There is no such thing as reliable exception handling in Perl.

1.27 Replaced tell STDOUT with bytes=tell STDOUT.

AUTHORS

Copyright 1995-2002, Lincoln D. Stein. All rights reserved.

This library is free software; you can redistribute it and/or modify it under the same terms as Perl itself.

Address bug reports and comments to: lstein@cshl.org

SEE ALSO

Carp, CGI::Base, CGI::BasePlus, CGI::Request, CGI::MiniSvr, CGI::Form, CGI::Response if (defined($CGI::Carp::PROGNAME)) { $file = $CGI::Carp::PROGNAME; }