NAME

DB - programmatic interface to the Perl debugging API

SYNOPSIS

```
package CLIENT;
use DB;
@ISA = qw(DB);

# these (inherited) methods can be called by the client

CLIENT->register() # register a client package name
CLIENT->done() # de-register from the debugging API
CLIENT->skippkg('hide::hide') # ask DB not to stop in this package
CLIENT->cont([WHERE]) # run some more (until BREAK or
# another breakpointt)
CLIENT->step() # single step
CLIENT->next() # step over
CLIENT->ret() # return from current subroutine
CLIENT->backtrace() # return the call stack description
CLIENT->ready() # call when client setup is done
CLIENT->trace_toggle() # toggle subroutine call trace mode
CLIENT->subs([SUBS]) # return subroutine information
CLIENT->files() # return list of all files known to DB
CLIENT->lines() # return lines in currently loaded file
CLIENT->loadfile(FILE,LINE) # load a file and let other clients know
CLIENT->lineevents() # return info on lines with actions
CLIENT->set_break([WHERE],[COND])
CLIENT->set_tbreak([WHERE])
CLIENT->clr_breaks([LIST])
CLIENT->set_action(WHERE,ACTION)
CLIENT->clr_actions([LIST])
CLIENT->evalcode(STRING) # eval STRING in executing code's context
CLIENT->prestop([STRING]) # execute in code context before stopping
CLIENT->poststop([STRING])# execute in code context before resuming

# These methods will be called at the appropriate times.
# Stub versions provided do nothing.
# None of these can block.

CLIENT->init() # called when debug API inits itself
CLIENT->stop(FILE,LINE) # when execution stops
CLIENT->idle() # while stopped (can be a client event loop)
CLIENT->cleanup() # just before exit
CLIENT->output(LIST) # called to print any output that
# the API must show
```

DESCRIPTION

Perl debug information is frequently required not just by debuggers, but also by modules that need some “special” information to do their job properly, like profilers.

This module abstracts and provides all of the hooks into Perl internal debugging functionality, so that various implementations of Perl debuggers (or packages that want to simply get at the “privileged” debugging data) can all benefit from the development of this common code. Currently used by Swat, the perl/Tk GUI debugger.

Note that multiple “front-ends” can latch into this debugging API simultaneously. This is intended to facilitate things like debugging with a command line and GUI at the same time, debugging debuggers etc. [Sounds nice, but this needs some serious support — GSAR]

In particular, this API does **not** provide the following functions:

- data display
- command processing
- command alias management
- user interface (tty or graphical)

These are intended to be services performed by the clients of this API.

This module attempts to be squeaky clean w.r.t `use strict;` and when warnings are enabled.

Global Variables

The following ''public'' global names can be read by clients of this API. Beware that these should be considered ''readonly''.

`$DB::sub`
: Name of current executing subroutine.

`%DB::sub`
: The keys of this hash are the names of all the known subroutines. Each value is an encoded string that has the *sprintf(3)* format`("%s:%d-%d", filename, fromline, toline)`.

`$DB::single`
: Single-step flag. Will be true if the API will stop at the next statement.

`$DB::signal`
: Signal flag. Will be set to a true value if a signal was caught. Clients may check for this flag to abort time-consuming operations.

`$DB::trace`
: This flag is set to true if the API is tracing through subroutine calls.

`@DB::args`
: Contains the arguments of current subroutine, or the `@ARGV` array if in the toplevel context.

`@DB::dbline`
: List of lines in currently loaded file.

`%DB::dbline`
: Actions in current file (keys are line numbers). The values are strings that have the *sprintf(3)* format`("%s\000%s", breakcondition, actioncode)`.

`$DB::package`
: Package namespace of currently executing code.

`$DB::filename`
: Currently loaded filename.

`$DB::subname`
: Fully qualified name of currently executing subroutine.

`$DB::lineno`
: Line number that will be executed next.

API Methods

The following are methods in the DB base class. A client must access these methods by inheritance (*not* by calling them directly), since the API keeps track of clients through the inheritance mechanism.

CLIENT->*register()*
: register a client object/package

CLIENT->evalcode(STRING)
: eval STRING in executing code context

CLIENT->skippkg('D::hide')
: ask DB not to stop in these packages

CLIENT->*run()*
: run some more (until a breakpt is reached)

CLIENT->*step()*
: single step

CLIENT->*next()*
: step over

CLIENT->*done()*
: de-register from the debugging API

Client Callback Methods

The following "virtual" methods can be defined by the client. They will be called by the API at appropriate points. Note that unless specified otherwise, the debug API only defines empty, non-functional default versions of these methods.

CLIENT->*init()*
: Called after debug API inits itself.

CLIENT->prestop([STRING])
: Usually inherited from DB package. If no arguments are passed, returns the prestop action string.

CLIENT->*stop()*
: Called when execution stops (w/ args file, line).

CLIENT->*idle()*
: Called while stopped (can be a client event loop).

CLIENT->poststop([STRING])
: Usually inherited from DB package. If no arguments are passed, returns the poststop action string.

CLIENT->evalcode(STRING)
: Usually inherited from DB package. Ask for a STRING to be `eval`-ed in executing code context.

CLIENT->*cleanup()*
: Called just before exit.

CLIENT->output(LIST)
: Called when API must show a message (warnings, errors etc.).

BUGS

The interface defined by this module is missing some of the later additions to perl's debugging functionality. As such, this interface should be considered highly experimental and subject to change.

AUTHOR

Gurusamy Sarathy gsar@activestate.com

This code heavily adapted from an early version of perl5db.pl attributable to Larry Wall and the Perl Porters.