Debugging with `rdebug` from GNU Emacs

0.10.4 Edition
October 2010

Rocky Bernstein and Anders Lindgren

Debugging with rdebug in GNU Emacs

This file describes `rdebug`, the Emacs interface to Ruby Debugger, `ruby-debug`, version 0.1

This is the 0.10.4 Edition, 15 October 2010

A special interface which comes with Ruby that allows you to use GNU Emacs to view (and edit) the source files for the program you are debugging with ruby-debug. However you must be using at least version 21 of GNU Emacs, but with GNU Emacs version 22 or 23 there are even more debugging features available. `M-x show-emacs-version` inside GNU Emacs will tell you what version you are running.

This package provide a full-fledged debugging environment, on par with modern integrated development environments. Once the debugger has been activated, the Emacs frame is divided into a number of dedicated debugger windows.[1]

This package comes with a number of predefined window layouts. It is fully customizable so you can create your own.

A rewrite of this interface is underway. See `http://github.com/rocky/emacs-rbdbgr`

[1] If you are an GNU Emacs traditionalist, you can, of course, run this package with only a shell and source buffer

1 Getting started

1.1 Installation

If you want to build the documentation and install Emacs files, you need:

- a POSIX shell
- GNU Make
- texinfo
- texi2html
- GNU Emacs

Download the ruby-debug-extra tarball from rubyforge http://rubyforge.org/frs/?group_id=1900&release_id=28306

For example for the 0.10.3 release:

```
wget http://rubyforge.org/frs/download.php/46883/ruby-debug-extra-0.10.3.tar.gz
```

Untar this, run configure, make, make check and make install. For example:

```
$ tar -xpf ruby-debug-extra-0.10.3.tar.gz
$ cd ruby-debug-extra-0.10.3
$ sh ./configure
$ make && make check
$ sudo make install  # or as root: make install
```

`configure --help` gives customization help.

Environment variable `EMACS` can be used to find which emacs to use, should you need to specify which emacs to use.

After installing the Emacs code on your computer, to use the interface inside Emacs load the file `rdebug.el` as in `M-x load-library rdebug`. This file is a light-weight file, basically it only contains a handful of `autoload` directives.

If you want to automatically load this package, you can place the either of the following in your `~/.emacs` file:

```
(require 'rdebug)
```

or

```
(autoload 'rdebug "rdebug" "ruby-debug interface" t)
```

In addition, you must have Ruby and ruby-debug installed.

1.2 Emacs rdebug

Use the command *`M-x rdebug`* in GNU Emacs to start debugging. Give the executable file you want to debug as an argument. Make sure to use the version that comes with this package as this is newer than that supplied with GNU Emacs.

The *`rdebug`* command starts ruby-debug as a subprocess of Emacs, with input and output through a newly created Emacs buffer.

Using ruby-debug under Emacs is just like using ruby-debug normally except for two things:

- All “terminal” input and output goes through the GNU Emacs buffer.

This applies both to ruby-debug commands and their output, and to the input and output done by the program you are debugging.

This is useful because it means that you can copy the text of previous commands and input them again; you can even use parts of the output in this way.

All the facilities of GNU Emacs' Shell mode are available for interacting with your script. In particular, you can send signals the usual way—for example, `C-c C-c` for an interrupt, `C-c C-z` for a stop.

1.3 Entering rdebug from an existing shell buffer

Many times it's not feasible to enter the debugger from the outset. Instead a call to the debugger is put inside the program.

It is also possible in GNU emacs to use a ("comint") shell and set a mode to watch for ruby-debug prompts and track the source code in another window. See Section "Shell" in *The GNU Emacs Manual*.

To enable, this run `M-x turn-on-rdebug-track-mode`. There is some overhead involved in scanning output, so if you are not debugging Ruby programs you probably want to turn this off which can be done via the `M-x turn-off-rdebugtrack` command.

This command doesn't set up the various buffers such as tracking local variables and the call stack. A more ambitious takeover of the shell which does set up these buffers is can be done via `M-x rdebug-track-attach`. A name of the program is solicited. This is used to make and association between the buffers. After this is done, you might not be able to use the shell for anything other than the debug session.

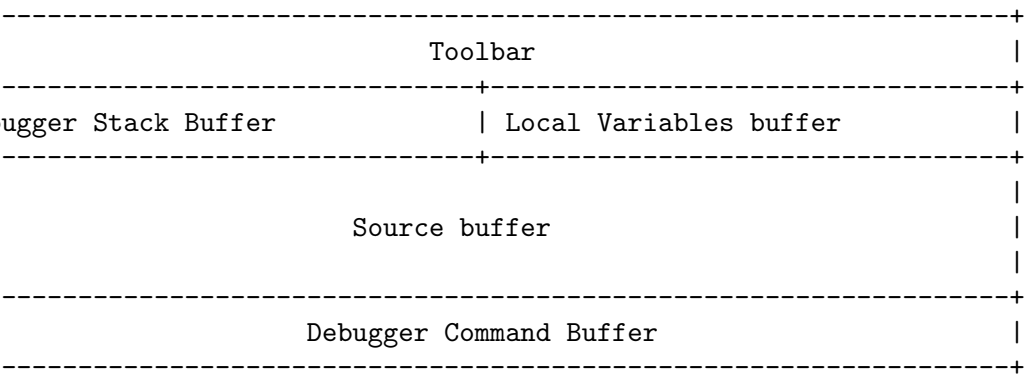
1.4 Emacs Customization

In this manual we present a number of GNU Emacs lisp variables and functions that you can use to configure the debugger interface. In addition, you can use the GNU Emacs *customize* system, see the `<menu-bar> <debugger> <options> <customize>` menu item.

2 Multi-window

In the multi-window debugger mode, a number of buffers are visible when the debugger starts. This chapter will describe each of them, in addition it will describe the features associated with the multi-window mode.

The default multi-window layout looks like the following:



However there are other layouts available. Look at the "Window Layout" drop-down list under the "Debugger" menu in the toolbar.

2.1 Activating Multi-window mode

The variable `rdebug-many-windows` controls if multi-window mode should be used, it is enabled by default. When starting the debugger using the `M-x rdebug` mode the command line option `--emacs 3` must be specified (this is also the default).

When attaching to an already running debugger process, you must give the debugger command *`set annotate 3`*.

2.2 Window Layouts

When the debugger is started, the original window layout of GNU Emacs is replaced with the window layout of the debugger. You can switch back and forth between the original window layout and the debugger layout using *`M-x rdebug-display-original-window-configuration`* and *`M-x rdebug-display-debugger-window-configuration`*.

If, for some reason, the debugger layout has been garbled you can restore it to the original state using *`M-x rdebug-restore-debugger-window-layout`*.

The debugger provides a number of different window layouts. The easies way to try them out is to use the menu *`<menu-bar> <debugger> <layout>`* and select any in the section starting with `Standard`.

2.3 The buffers

All buffers in this section share a set of commands for common debugger operations and for switching between buffers. In addition, each buffer has got a set of dedicated commands.

All debugger buffers, with the exception of source and the debugger shell window, are called *secondary buffers*.

2.3.1 Keybindings for all Debugger Windows

The debugger provides key-bindings that work in all debugger windows, including Ruby source buffers. The key bindings are designed to match keys of commonly used debugger environments.

The variable `rdebug-populate-common-keys-function` can be assigned to a function that should bind the keys use. Three functions are provided `rdebug-populate-common-keys-standard`, `...-eclipse`, and `...-netbeans`.

Command	**Standard**	**Eclipse**	**Netbeans**
Run	f5		
Quit	S-f5		
Toggle Breakpoint	f9		
Enable/Disable Breakpoint	C-f9	S-C-b	S-f8
Step over	f10	f6	f8
Step into	f11	f5	f7
Step out	S-f11	f7	M-S-f7

2.3.2 Keybindings for Secondary Buffers

The following commands are available in all secondary windows.

Capital letters move between secondary buffers as mentioned above (jump to if visible or replace a secondary if not).

SPACE	step (edebug compatible)
<	Up in the stack trace
>	Down in the stack trace
?	Help
B	Display breakpoints buffer
C	Display command buffer
O	Display program output
S	Display source window
T	Display stack trace buffer
V	display variables buffer
W	display watch buffer
b	Set breakpoint
c	Continue (i.e. run)
d	Remove breakpoint
f	Finish (i.e. step out of the current function)
n	Next (i.e. step into function)

p	print
q	Quit
r	Restart
s	Step (i.e. step over function)

You can use the same commands in the source buffer if you enable `rdebug-short-key-mode`. The best way to do this is to add the following to your init file:

```
(add-hook 'rdebug-mode-hook 'rdebug-turn-on-short-key-mode)
```

2.3.3 The Debugger Shell Buffer

The *debugger shell window* is the main communication channel between ruby-debug and GNU Emacs. You can use the shell to issue debugger commands directly. In addition, any GNU Emacs debugger command you issue will be translated into shell commands, and the output will be parsed.

It is the ambition that the GNU Emacs debugger interface should be in a state where the debugger shell window would not need to be visible.

2.3.4 The Source Buffer

The *source buffers* (or buffers) contains the actual Ruby source code that is being debugged. A small arrow in the left fringe displays the current line. Active breakpoints are displayed as red dots and passive as grey.

2.3.5 The Output Buffer

The *output buffer* displays any output the debugged program emits.

The option `rdebug-use-separate-io-buffer` controls if the output buffer should be used, or if the output would go into the debugger shell buffer.

2.3.6 The Variables Buffer

In this buffer, local and object variables are displayed. The values of the variables can be edited.

RET	Edit the value
e	Print the value
x	Pretty-print the value

2.3.7 The Stack Trace Buffer

The *stack trace* buffer displays the function that is currently being debugger, the function that called it, etc., all the way up to the originally called function.

You can navigate in the stack trace buffer in order to see the source of any function in the call chain. The Variables buffer will also be updated to reflect the local variables of that function.

RET	Select a function to display
<digits>	Go to a stack frame

2.3.8 The Watch Buffer

The *Watch Buffer* can display arbitrary expressions, including, but not limited to, global variables.

`a`	Add a watch expression
`C-d, d`	Delete a watch expression
`RET, e`	Edit a watch expression
`<digits>`	Go to the expression

2.3.9 The Breakpoints Buffer

The *Breakpoints Buffer* displays all breakpoints that currently are defined and shows if they are enabled or disabled.

`t`	Toggle a breakpoint between enabled and disabled
`i`	Add a breakpoint condition
`ret`	Goto a breakpoint
`C-d`	Delete a breakpoint
`<digits>`	Go to the expression

2.3.10 The Help Buffer

The *Help Buffer* is displayed whenever you press `?`. It will display a help text on the available debugger commands and commands to navigate between the buffers.

3 Debugger Buffers

3.1 Emacs Debugger Command buffer

Each time ruby-debug displays a stack frame, Emacs automatically finds the source file for that frame and puts an arrow (‘=>’) at the left margin of the current line. Emacs uses a separate buffer for source display, and splits the screen to show both your ruby-debug session and the source.

Explicit ruby-debug `list` or search commands still produce output as usual, but you probably have no reason to use them from GNU Emacs.

> *Warning:* If the directory where your script resides is not your current directory, it can be easy to confuse Emacs about the location of the source files, in which case the auxiliary display buffer does not appear to show your source. ruby-debug can find programs by searching your environment's `PATH` variable, so the ruby-debug input and output session proceeds normally; but Emacs does not get enough information back from ruby-debug to locate the source files in this situation. To avoid this problem, either start ruby-debug mode from the directory where your script resides, or specify an absolute file name when prompted for the *`M-x gdb`* argument.
>
> A similar confusion can result if you use the ruby-debug `file` command to switch to debugging a program in some other location, from an existing ruby-debug buffer in Emacs.

(preceded by *`M-:`* or *`ESC :`*, or typed in the `*scratch*` buffer, or in your ‘`.emacs`’ file).

In the ruby-debug I/O buffer, you can use the Emacs commands listed below in addition to the standard Shell mode commands. The I/O buffer name name is usually `*gud-`*script-name*`*`, where *script-name* is the name of the script you are debugging.

Many of the commands listed below are also bound to a second key sequence which also can be used in the also be used in the source script. These are listed in Section 3.2 [Emacs Source], page 11.

In secondary buffers many commands are available the corresponding final keystroke. For example `C-c n` in a secondary buffer is `n`.

`C-h m` Describe the features of Emacs' ruby-debug Mode.

`C-x C-a C-b (gud-break)`
Set breakpoint at current line.

`C-x C-a C-d (gud-remove)`
Remove breakpoint at current line.

`C-x C-a C-l (gud-refresh)`
Fix up a possibly garbled display, and redraw the arrow.

`C-c RET (comint-copy-old-input)`
Insert after prompt old input at point as new input to be edited. Calls ‘comint-get-old-input’ to get old input.

`C-c n (gud-next)`
Step one line, skipping functions. (Step over).

C-x C-a C-o (comint-delete-output)
: Delete all output from interpreter since last input. Does not delete the prompt.

C-x C-a C-r (gud-cont)
C-c SPC (gud-step arg)
C-x C-a C-s (gud-step arg)
: Step one source line. Same as ruby-debug `step` command. The GNU Emacs command name is `gud-step` and `C-x C-a C-s` is an alternate binding which can be used in the source script.

  With a numeric argument, run that many times. See Section "Numeric Arguments" in *The GNU Emacs Manual*.

C-x C-a C-t (gud-tbreak arg)
: Set temporary breakpoint at current line.

C-x C-a C-w (backward-kill-word)
C-x C-a C-x (comint-get-next-from-history)
C-x C-a C-z (comint-stop-subjob)
: Stop the current subjob. This command also kills the pending input between the process mark and point.

  WARNING: if there is no current subjob, you can end up suspending the top-level process running in the buffer. If you accidentally do this, use M-x comint-continue-subjob to resume the process. (This is not a problem with most shells, since they ignore this signal.)

C-x C-a C-\ (comint-quit-subjob)
: Send quit signal to the current subjob. This command also kills the pending input between the process mark and point.

C-c + (gud-step-plus)
: Run `step+`.

C-c . (comint-insert-previous-argument index)
: Insert the *index-th* argument from the previous Comint command-line at point. Spaces are added at beginning and/or end of the inserted string if necessary to ensure that it's separated from adjacent arguments. Interactively, if no prefix argument is given, the last argument is inserted. Repeated interactive invocations will cycle through the same argument from progressively earlier commands (using the value of index specified with the first command).

C-c < (gud-up)
: Go up a stack frame. With a numeric argument, go up that many stack frames. Same ruby-debug `up` command. See Section "Numeric Arguments" in *The GNU Emacs Manual*.

C-c > (gud-down)
: Go down a stack frame. Same as ruby-debug `down`. With a numeric argument, go down that many stack frames. See Section "Numeric Arguments" in *The GNU Emacs Manual*.

C-c ? (rdebug-display-secondary-window-help-buffer)
: Display the rdebug help buffer.

C-c B (rdebug-display-breakpoints-buffer)
: Display the rdebug breakpoints buffer.

C-x C-a C (rdebug-display-cmd-buffer)
: Display the rdebug command buffer.

C-c O (rdebug-display-output-buffer)
: Display the rdebug output buffer.

C-c R (gud-run)
C-c r (gud run)
: Restart or run the script. Same as ruby-debug `run` command.

C-c S (gud-source-resync)
C-c T (rdebug-display-stack-buffer)
: Display the rdebug stack buffer.

C-c V (rdebug-display-variables-buffer)
: Display the rdebug variables buffer.

C-c W (rdebug-display-watch-buffer)
: Display the rdebug watch buffer.

C-c f (gud-finish arg)
: Finish executing current function.

C-x C-a C-f (gud-finish)
: Finish executing current function. The same as ruby-debug `finish` command.

C-c n (gud-next)
: Execute to next source line in this function, skipping all function calls. Same as ruby-debug `next` command.

  With a numeric argument, run that many times.

C-c q (gud-quit)
C-x C-a C-l
: Resynchronize the current position with the source window. The GNU Emacs command name is `gud-refresh` and `C-x C-a C-l` is an alternate binding which also can be used in the source script.

C-c a
: Shows argument variables (e.g. `$1`, `$2`) of the current stack frame. Same as ruby-debug `info args` command. The GNU Emacs command name is `gud-args` and `C-x C-a a` is an alternate binding which also can be used in the source script.

C-c T
: Show stack trace. Same as ruby-debug `where` command. The GNU Emacs command name is `gud-where` and `C-x C-a T` is an alternate binding which can be used in the source script.

In any source file, the Emacs command *C-x SPC* (`gud-break`) tells ruby-debug to set a breakpoint on the source line point is on.

If you accidentally delete the source-display buffer, an easy way to get it back is to type the command `frame` in the ruby-debug buffer, to request a frame display; when you run under Emacs, this recreates the source buffer if necessary to show you the context of the current frame.

The source files displayed in Emacs are in ordinary Emacs buffers which are visiting the source files in the usual way. You can edit the files with these buffers if you wish; but keep in mind that ruby-debug communicates with Emacs in terms of line numbers. If you add or delete lines from the text, the line numbers that ruby-debug knows cease to correspond properly with the code.

See Section "Debugger Operation" in *The GNU Emacs Manual*.

3.2 Commands from the source script

C-x SPC

tells ruby-debug to set a breakpoint on the source line point is on. (`gud-break`)

C-x C-a t

`gud-linetrace`

C-x C-a C-f

Restart or run the script. Same as ruby-debug `run` command. The GNU Emacs command name is `gud-finish`. In the corresponding I/O buffer, `C-c R` is an alternate binding.

C-x C-a T Show stack trace. Same as ruby-debug `where` command. In the corresponding I/O buffer, `C-c T` is an alternate binding.

C-x C-a <

Go up a stack frame. With a numeric argument, go up that many stack frames. Same ruby-debug `up` command. See Section "Numeric Arguments" in *The GNU Emacs Manual*.

The GNU Emacs command name is `gud-up`. In the corresponding I/O buffer, `C-c <` is an alternate binding.

C-x C-a >

Go down a stack frame. Same as ruby-debug `down`. With a numeric argument, go down that many stack frames. See Section "Numeric Arguments" in *The GNU Emacs Manual*.

The GNU Emacs command name is `gud-down`. In the corresponding I/O buffer, `C-c >` is an alternate binding.

C-x C-a C-t

`gud-tbreak`

C-x C-a C-s

Step one source line. Same as ruby-debug `step` command.

With a numeric argument, run that many times. See Section "Numeric Arguments" in *The GNU Emacs Manual*.

The GNU Emacs command name is `gud-step`. In the corresponding I/O buffer, `C-x C-a C-s` is an alternate binding.

C-x C-a C-e

`gud-statement`

C-x C-a R Restart or run the script. Same as ruby-debug `run` command. The GNU Emacs command name is `gud-run`. In the corresponding I/O buffer, `C-c R` is an alternate binding.

C-x C-a C-d
Delete breakpoint. `gud-remove`

C-x C-a C-p
`gud-print`

C-x C-a C-n
Execute to next source line in this function, skipping all function calls. Same as ruby-debug `next` command. With a numeric argument, run that many times. See Section "Numeric Arguments" in *The GNU Emacs Manual*.

The GNU Emacs command name is `gud-next`. In the corresponding I/O buffer, `C-x C-a C-n` is an alternate binding.

C-x C-a f C-f
`gud-finish`

C-x C-a C-r
Continue execution of your script Same as ruby-debug `continue` command. The GNU Emacs command name is `gud-cont`. In the corresponding I/O buffer, `C-x C-a C-r` is an alternate binding.

C-x C-a C-b
`gud-break`

C-x C-a a

`gud-args` Shows argument variables (e.g. `$1`, `$2`) of the current stack frame. Same as ruby-debug `info args` command. The GNU Emacs command name is `gud-args`. In the corresponding I/O buffer, `C-c a` is an alternate binding which also can be used in the source script.

C-x C-a C-l
Move to current position in this source window. The GNU Emacs command name is `gud-refresh`. In the corresponding I/O buffer, `C-x C-a C-l` is an alternate binding.

4 Emacs Debugger Commands

4.1 Emacs Debugger Common Commands

The commands in this section are used to make a secondary buffer visible. If the buffer doesn't exist, nothing is done. The way the buffer is made visible is follows the following rules tried in order:

1. If the buffer doesn't exist, do nothing.
2. If the buffer is already displayed, switch to it.
3. If the current buffer is a secondary buffer, bury it replacing with the requested buffer.
4. If there is secondary buffer visible, that is replaced instead.
5. Just pick a visible buffer to bury and replace.

The commands are also have key bindings that end in an uppercase letter. This letter is given in parenthesis. When in one of the secondary buffers, the uppercase letter is bound to the command as well.

(rdebug-display-breakpoints-buffer) (B)
: Display the rdebug breakpoints buffer. Bound to: *C-x C-a B*, *<menu-bar> <debugger> <view> <breakpoints>*. Secondary buffers: *O*.

(rdebug-display-cmd-buffer) (C)
: Display the debugger command buffer.

  Bound to: *C-x C-a C*, *<menu-bar> <debugger> <view> <shell>*.

(rdebug-display-output-buffer) (O)
: Display the debugger output buffer.

  Bound to: *C-x C-a O*, *<menu-bar> <debugger> <view> <output>*. Secondary buffers: *O*.

(rdebug-display-secondary-window-help-buffer) (?)
(rdebug-display-stack-buffer) (T)
: Display the debugger stack buffer. Bound to: *C-x C-a T*, *<menu-bar> <debugger> <view> <stack>*. Secondary buffers: *T*.

(rdebug-display-variables-buffer) (V)
: Display the debugger variables buffer. Bound to: *C-x C-a V*, *<menu-bar> <debugger> <view> <variables>*. Secondary buffers: *V*.

(rdebug-display-watch-buffer) (W)
: Display the debugger variables buffer. Bound to: *C-x C-a W*, *<menu-bar> <debugger> <view> <watch>*. Secondary buffers: *V*.

(rdebug-display-debugger-window-configuration)
: Display the current layout of windows of the rdebug Ruby debugger.

(rdebug-display-original-window-configuration)
: Display the layout of windows prior to starting the rdebug Ruby debugger. This function is called upon quitting the debugger and *rdebug-many-windows* is not nil.

(rdebug-goto-entry-n)
: Breakpoints, Display expressions and Stack Frames all have numbers associated with them which are distinct from line numbers. In a secondary buffer, this function is usually bound to a numeric key. which will position you at that entry number. To go to an entry above 9, just keep entering the number. For example, if you press 1 and then 9, you should jump to entry 1 (if it exists) and then 19 (if that exists). Entering any non-digit will start entry number from the beginning again.

(rdebug-quit) - `q`
: Kill the debugger process associated with the buffer.

  When *rdebug-many-windows* is active, the original window layout is restored.

(rdebug-restore-windows)
: Display the initial ruby debugger window layout.

4.2 Emacs Debugger Breakpoint Buffer Commands

(rdebug-goto-breakpoint)
(rdebug-goto-breakpoint-mouse)
(rdebug-breakpoints-mode)
: Major mode for displaying breakpoints in a secondary window. Uses *rdebug-breakpoints-mode-map*.

4.3 Emacs Debugger Stack Buffer Commands

(rdebug-goto-stack-frame)
(rdebug-frames-mode)
: Major mode for displaying the stack trace. Uses *rdebug-frames-mode-map*.

4.4 Emacs Debugger Variable Buffer Commands

(rdebug-variables-edit)

4.5 Emacs Debugger Watch Buffer Commands

(rdebug-watch-add)
: Add a display expression.

(rdebug-watch-delete)
: Delete a display expression.

(rdebug-watch-edit)
: Edit a display expression.

(rdebug-watch-mode)
: Major mode for displaying the display expressions. Uses *rdebug-watch-mode-map*.

4.6 Emacs Debugger GUD Commands

(comint-copy-old-input)
: Insert after prompt old input at point as new input to be edited. Calls 'comint-get-old-input' to get old input.

(comint-delete-output)
: Delete all output from interpreter since last input. Does not delete the prompt.

(gud-break)
: Set a breakpoint on the source line point is on.

(gud-cont) - *c*
: Continue execution.

(gud-next) - *n*
: Step one line, skipping functions. (Step over).

(gud-refresh)
: Fix up a possibly garbled display, and redraw the arrow.

(gud-remove)
: Remove breakpoint at current line.

(gud-step) - *s*
: Step one statement. (Step into)

(gud-step-plus) - +
: Run `step+`—like `gud-step` but ensure we go to a new line.

(gud-tbreak arg)
: Set temporary breakpoint at current line.

Emacs Command Index

Emacs Function Index

C

G

R

Key Binding Index

(Index is nonexistent)

The body of this manual is set in
cmr10 at 10.95pt,
with headings in **cmb10 at 10.95pt**
and examples in `cmtt10 at 10.95pt`.
cmti10 at 10.95pt,
cmb10 at 10.95pt, and
cmsl10 at 10.95pt
are used for emphasis.