CGDB

for version 0.6.6, 6 September 2011

Bob Rossi (bob@brasko.net)

This manual is for GNU CGDB (version 0.6.6, 6 September 2011), the GNU ncurses based front end to GDB.

Table of Contents

Summary of CGDB 1

1 Getting In and Out of CGDB 2

2 Understanding the core concepts of CGDB . . 3

2.1 Understanding the source window. 3
2.2 Understanding the GDB window. 4
2.3 Understanding the file dialog window. 4
2.4 Understanding the TTY window. 4
2.5 Understanding the status bar. 4
2.6 Switch between windows 5

3 CGDB commands 6

3.1 Commands available during CGDB mode 6
3.2 Commands available during GDB mode 7
3.3 Commands available during the file dialog mode 8
3.4 Commands available during TTY mode 8

4 CGDB configuration commands 9

5 CGDB highlighting groups 13

5.1 The different highlighting groups 13
5.2 The different attributes 14
5.3 The different colors 14

6 CGDB key user interface 16

6.1 The KUI's time out options 16
6.2 Using maps 17
6.3 Understanding keycodes 17

7 Sending I/O to the program being debugged 19

8 Allowing terminal control flow in CGDB 20

9 Building CGDB from source 21

Appendix A Copying This Manual **22**
A.1 GNU Free Documentation License 22
A.1.1 ADDENDUM: How to use this License for your documents 28

Index **29**

Summary of CGDB

CGDB is a curses-based interface to the GNU Debugger (GDB). The goal of CGDB is to be lightweight and responsive; not encumbered with unnecessary features.

The interface is designed to deliver the familiar GDB text interface, with a split screen showing the source as it executes. The UI is modeled on the classic Unix text editor, vi. Those familiar with vi should feel right at home using CGDB.

The library responsible for communicating with GDB is called Trivial GDB (tgdb, or more accurately, libtgdb). This abstraction allows the UI code to be independent of the debugger, as well as greatly simplifying its implementation.

Those wanting to develop other interfaces to GDB are welcome to use libtgdb as the basis for their program. Many of the headaches of parsing GDB's output and annotations can be avoided by using it.

Some features offered by CGDB are:

- Syntax-highlighted source window
- Visual breakpoint setting
- Keyboard shortcuts for common functions
- Searching source window (using regexp)

1 Getting In and Out of CGDB

This chapter discusses how to start CGDB, and how to get out of it. The essentials are:

- type '`cgdb`' to start CGDB.
- type *`quit`* or *`C-d`* in the GDB window to exit.
- type *`:quit`* in the source window to exit. This even works if GDB is currently hanging, or operating a long command.

2 Understanding the core concepts of CGDB

The CGDB user interface currently consists of two windows and a status bar. The source window is currently on the top and the GDB window is on the bottom. The *status bar* currently separates the two windows.

The interface has several modes depending on which window is focused. *CGDB mode* is when the source window is focused, *GDB mode* is when the GDB window is focused and *TTY mode* is when the TTY window is focused.

Beginning with CGDB version 1.0, the windows are movable, and the user will be able to create as many or as few that is desired. Currently however, all of my time is spent developing the interface between CGDB and GDB. Once this is complete, the UI of CGDB will become much more polished. If you are a ncurses developer, and have spare time to work on this task, please contact me.

2.1 Understanding the source window.

The *source window* is the window that provides you a view of the source code that the debugged program is made up of. It will display to the user a single source file at a time. While the user is debugging, via `next` and `step`, CGDB will update the source file and line number to keep you informed as to where GDB is debugging.

CGDB has several features that make debugging easier than using plain old GDB. One feature you will notice right away while debugging a C, C++ or ADA program, is that the source files are syntax highlighted. This allows the user to easily navigate through the source file to look for certain places in the source code. If you would like to see another source language highlighted, contact us. To understanding how to navigate through the source window look at the commands in Section 3.1 [CGDB Mode], page 6.

In addition to showing the source code, CGDB also displays to the user the currently executing line. The line number will be highlighted green, to represent that the particular line, is the current line being debugged by GDB. Also, CGDB will display an arrow extending from the line number, to the source line. You can configure what type of arrow CGDB uses with the `:set arrowstyle` configuration option. By default, the `short` arrow is used. However, my personal favorite is the `long` arrow.

As you navigate through the source window, the current line the cursor is on will be highlighted white. This simply helps you keep track of where you are in the file.

Also, you can set or delete breakpoints in CGDB from the source window. Simply navigate to the line that you are interested in setting a breakpoint, and hit the space bar. This will set a breakpoint on the line if one did not already exist. The line number should turn red to indicate that a breakpoint has been set. Hitting the space bar again will delete the breakpoint. If you disable the breakpoint, the line number will turn yellow, to represent the disabled breakpoint.

CGDB also supports regular expression searching within the source window. If you type `/` or `?` you can search in the source window for a string of interest. The C library regular expression functions are used to perform this search, which honors things like '`*`' or '`+`'.

The full list of commands that are available in the source window is in Section 3.1 [CGDB Mode], page 6.

2.2 Understanding the GDB window.

The *GDB window* is how CGDB allows the user to interface with the GNU debugger. If you wish to pass a command to GDB, simply type it into this window and GDB will receive the command. This interface is intended to be 100% identical to using GDB on a terminal.

There is a limited set of keys that can be typed into this window that CGDB interprets and handles, instead of sending to GDB. They are all available in Section 3.2 [GDB Mode], page 7.

CGDB attempts to buffer commands the same way they would be if you typed them at the terminal. So, if you type several commands before a single one finishes, they will each be run in order. There will be no way to stop these commands from being run besides from typing `Ctrl-C`, like you would at any normal terminal when working with GDB.

2.3 Understanding the file dialog window.

The *file dialog window* is available to help the user view and select which file they would like to view. It provides the user with a list of all the files that make up the program being debugged. If there are no files available, because there is no program being debugged or because there is no debug symbols, then the file dialog will not open and a message will be displayed at the status bar.

You can get to the file dialog by hitting `o` when you are at the source window. Once you enter the file dialog, it is possible to leave it by hitting `q`. You can navigate the file dialog using the standard direction keys and you can even use regular expression to find your file. This can save a lot of time as the number of files grow.

The full list of commands that are available in the source window is in Section 3.3 [File Dialog Mode], page 8.

2.4 Understanding the TTY window.

The *TTY window* is available to allow the user to pass input to the program being debugged. This window will act similar to the GDB window, except that the data you type will get sent to the program being debugged. See Chapter 7 [Sending I/O to Inferior], page 19.

You will notice that the TTY window has a terminal device between it and the program being debugged. So, if the program being debugged uses say readline, which allows command line editing, the same interface will be provided via the TTY window as at the terminal. You can see the name of the terminal device in the TTY status bar.

The full list of commands that are available in the source window is in Section 3.4 [TTY Mode], page 8.

Sending I/O to the program being debugged can be confusing. It is described better in Chapter 7 [Sending I/O to Inferior], page 19. Unless the I/O with the program being debugged is simple, I usually prefer starting the application on a separate terminal and attaching to it with CGDB.

2.5 Understanding the status bar.

The *status bar* is the general purpose way for CGDB to show the user which commands they are currently typing or report errors to the user when they occur. CGDB does not use popup's or other forms of I/O to alert the user of information or problems.

While CGDB is running, you can configure it with any of the commands that are valid in CGDB's configuration file. Simply type *:* in the source window, and you will see the colon, and the rest of the command you type appear in the status bar. When you are finished typing the command that you are interested in, type `enter`. This will alert CGDB to execute the command. If at any point you would like to cancel the current command typed so far, type the cgdb mode key. This will put you back into CGDB mode. For a description of the cgdb mode key, see Section 2.6 [Switching Windows], page 5.

The full list of commands that are available in the source window is in Chapter 4 [Configuring CGDB], page 9.

2.6 Switch between windows

When CGDB is invoked, the interface is in *GDB mode*. A '`*`' at the right of the status bar indicates that input will be passed to GDB. To change the focus to the source window, hit the `ESC` key. The *cgdb mode key* is the key that is responsible for switching the user into *CGDB mode* from a different mode. The cgdb mode key is defaulted to the `ESC` key. To change this value, look at the configuration options for CGDB. See Chapter 4 [Configuring CGDB], page 9.

The interface is now in *CGDB mode*. To switch back into *GDB mode*, press *`i`*. This syntax is based on the popular Unix text-editor, vi.

3 CGDB commands

CGDB can be controlled in a variety of different ways. Each mode that CGDB is in acts differently. Currently CGDB implicitly changes modes depending on which window is active. The following information will help you determine what commands are accessible during which modes.

3.1 Commands available during CGDB mode

When you are in the source window, you are implicitly in *CGDB mode*. All of the below commands are available during this mode. This mode is primarily available for the user to view the current source file, search it, or switch to a different mode.

`cgdbmodekey`
: Puts the user into command mode. However, you are already in this mode. This is defaulted to the `ESC` key.

`i` Puts the user into *GDB mode*.

`I` Puts the user into *TTY mode*.

`T` Opens a window to give input to the debugged program.

`Ctrl-T` Opens a new tty for the debugged program.

`k`
`up arrow` Move up a line.

`j`
`down arrow`
: Move down a line.

`h`
`left arrow`
: Move left a line.

`l`
`right arrow`
: Move right a line.

`Ctrl-b`
`page up` Move up a page.

`Ctrl-u` Move up 1/2 a page.

`Ctrl-f`
`page down` Move down a page.

`Ctrl-d` Move down 1/2 a page.

`gg` Move to the top of file.

`G` Move to the bottom of file.

`/` search from current cursor position.

`?` reverse search from current cursor position.

n	next forward search.
N	next reverse search.
o	open the file dialog.
spacebar	Sets a breakpoint at the current line number.
t	Sets a temporary breakpoint at the current line number.
-	Shrink source window 1 line.
=	Grow source window 1 line.
_	Shrink source window 25% (or, shrink tty window 1 line, if visible).
+	Grow source window 25% (or, grow tty window 1 line, if visible).
Ctrl-l	Clear and redraw the screen.
F5	Send a run command to GDB.
F6	Send a continue command to GDB.
F7	Send a finish command to GDB.
F8	Send a next command to GDB.
F10	Send a step command to GDB.

3.2 Commands available during GDB mode

When in *GDB mode*, the user is mostly interested in working with the GDB console. That is, sending commands to GDB and receiving data back from GDB. Almost all data passed into this window is directly sent to readline and then to GDB.

It is important to understand that CGDB parses the keys entered in the GDB window and has the first chance at dealing with them. If it is interested in the keys, it will handle them. Below is a list of keys that CGDB is interested in, and does not pass along any further.

cgdbmodekey	Switch back to source window. This is defaulted to the `ESC` key.
page up	Move up a page.
page down	Move down a page.
F11	Go to the beginning of the GDB buffer.
F12	Go to the end of the GDB buffer.

Any other keys, besides the ones above, CGDB is currently not interested in. CGDB will pass along these keys to the readline library. When readline has determined that a command has been received, it alerts CGDB, and a command is then sent to GDB. This is the same method used when invoking GDB directly.

3.3 Commands available during the file dialog mode

The file dialog is primarily used to allow the user to find and open a source file that the program they are debugging is made up of. The file dialog will be full screen, and will list each file that the debugged program is made up of. A usual instance of the file dialog would be to open it up from the source window using the *`o`* key, and then to search for the file of interest. If you are looking for foo.c type `/foo.c`, press *`enter`* once to finish the regular expression and again to select the file.

The commands available in the file dialog are:

`q` Will exit the file dialog, and return to the source window.

`k`
`up arrow` Move up a line.

`j`
`down arrow`
Move down a line.

`h`
`left arrow`
Move left a line.

`l`
`right arrow`
Move right a line.

`Ctrl-b`
`page up` Move up a page.

`Ctrl-f`
`page down` Move down a page.

`/` search from current cursor position.

`?` reverse search from current cursor position.

`n` next forward search.

`N` next reverse search.

`enter` Select the current file.

3.4 Commands available during TTY mode

`cgdbmodekey`
Switch back to source window. This is defaulted to the `ESC` key.

`page up` Move up a page.

`page down` Move down a page.

`F11` Go to the beginning of the GDB buffer.

`F12` Go to the end of the GDB buffer.

4 CGDB configuration commands

There may be several features that you find useful in CGDB. CGDB is capable of automating any of these commands through the use of the config file called '`cgdbrc`'. It looks in `$HOME`'`/.cgdb/`' for that file. If it exists, CGDB executes each line in the file in order. It is as if the user typed in all the commands into the status bar after the tui was initialized.

The following variables change the behavior of some aspect of CGDB. Many of these commands may be abbreviated in some way, and all boolean commands my be negated by appending '`no`' to the front. For example: *`:set ignorecase`* turns on case-insensitive searching; while *`:set noignorecase`* turns on case-sensitive searching.

`:set as=style`
`:set arrowstyle=style`
: Set the arrow style to *style*. Possible values for *style* are '`short`', '`long`', and '`highlight`'. Changes the style of the arrow that is used to indicate the currently executing line in the source viewer. The default value is short. A longer arrow is available, for easier readability. Finally, the '`highlight`' option draws no arrow at all, instead drawing the entire line in inverse video.

`:set asr`
`:set autosourcereload`
: If this is on, CGDB will automatically reload a source file if it has changed since CGDB has opened it. If it is off, the file will never be reloaded, until you start CGDB again. The default is on. This feature is useful when you are debugging a program, then you modify a source file, recompile, and type *`r`* in GDB's CLI window. The file in this case will be updated to show the new version. Note, CGDB only looks at the timestamp of the source file to determine if it has changed. So if you modify the source file, and didn't recompile yet, CGDB will still pick up on the changes.

`:set cgdbmodekey=key`
: This option is used to determine what key puts CGDB into *CGDB Mode*. By default, the *`ESC`* key is used. *key* can be any normal key on the keyboard. It can also be any keycode, as long as the keycode notation is used. This option is especially useful when the user wants to use readline in vi mode. If the user types `set cgdbmodekey=<PageUp>` then the *`Page Up`* key will put CGDB into CGDB mode and the *`ESC`* key will flow through to readline.

`:set ic`
`:set ignorecase`
: Sets searching case insensitive. The default is off.

`:set stc`
`:set showtgdbcommands`
: If this is on, CGDB will show all of the commands that it sends to GDB. If it is off, CGDB will not show the commands that it gives to GDB. The default is off.

`:set syn=style`
`:set syntax=style`

Sets the current highlighting mode of the current file to have the syntax *style*. Possible values for *syntax* are '`c`', '`ada`', and '`off`'. Normally, the user will never have to do this, since CGDB automatically detects what syntax a file should be based on its file extension. However, this feature can currently be useful for debugging purposes.

`:set to`
`:set timeout`

This option is used along with the *ttimeout* option to determine the behavior CGDB should have when it receives part of a mapped key sequence or a keyboard code sequence. If this option is on, CGDB will time out on both user defined mappings and on key codes from the keyboard. If this option is off, user defined mappings will not be timed out on. In this case, CGDB will determine if it should time out on key codes from the keyboard by examining the *ttimeout* option. To determine how CGDB will time out on mappings and key codes, and what time out lengths CGDB will use, please refer to the chart in Chapter 6 [Key User Interface], page 16. The default value for this option is on.

`:set tm=delay`
`:set timeoutlen=delay`

This option is used along with the *ttimeoutlen* option. It represents the number of milliseconds that CGDB should wait for a key code from the keyboard or for a mapped key sequence to complete. If *delay* is 0, CGDB immediately accepts each character it receives. This will prevent any mappings or key codes to complete. *delay* may be any value between 0 and 10000, inclusive. The default setting for the *delay* variable is 1000 (one second).

`:set ttimeout`

This option is used along with the *timeout* option to determine the behavior CGDB should have when it receives part of keyboard code sequence. If this option is on, CGDB will time out on key codes from the keyboard. If this option is off, CGDB will determine if it should time out on key codes from the keyboard by examining the *timeout* option. To determine how CGDB will time out on key codes, what what time length it will use, please refer to the chart in Chapter 6 [Key User Interface], page 16. The default value for this option is on.

`:set ttm=delay`
`:set ttimeoutlen=delay`

This option is used along with the *ttimeoutlen* option. It represents the number of milliseconds that CGDB should wait for a key code from the keyboard. If *delay* is 0, CGDB immediately accepts each character it receives. This will prevent any key codes to complete. *delay* may be any value between 0 and 10000, inclusive. The default setting for the *delay* variable is 100 (one tenth of a second).

:set ts=number
:set tabstop=number
: Sets the number of spaces that should be rendered on the screen for `TAB` characters. The default value for *number* is 8.

:set wmh=number
:set winminheight=number
: The minimal height of a window. Windows will never become smaller than this value. The default value for *number* is 0.

:set winsplit=style
: Set the split point between source and GDB window. This is especially useful as an init setting in your '`cgdbrc`' file. See Chapter 4 [Configuring CGDB], page 9. The possible values for *style* are '`top_full`', '`top_big`', '`even`', '`bottom_big`', and '`bottom_full`'.

:set ws
:set wrapscan
: Searches wrap around the end of file. The default is on.

:c
:continue
: Send a continue command to GDB.

:down Send a down command to GDB.

:e
:edit reloads the file in the source window. this can be useful if the file has changed since it was opened by cgdb.

:f
:finish Send a finish command to GDB.

:help This will display the current manual in text format, in the *source window*.

:hi group cterm=attributes ctermfg=color ctermbg=color term=attributes
:highlight group cterm=attributes ctermfg=color ctermbg=color term=attributes
: Set the *color* and *attributes* for a highlighting group. The syntax mimics vim's "highlight" command. Possible values for *group*, *attributes* and *color* are available in Chapter 5 [Highlighting Groups], page 13.

You can give as many or as few of the name=value pairs as you wish, in any order. '`ctermfg`' and '`ctermbg`' set the foreground and background colors. These can be specified by color number or by using the same color names that vim uses. When CGDB is linked with ncurses, the number you use to represent the color can be between -1 and COLORS. When CGDB is linked against curses, it must be between 0 and COLORS.

'`cterm`' sets the video attributes for color terminals. '`term`' sets the video attributes for monochrome terminals. Some examples are,

```
:highlight Logo cterm=bold,underline ctermfg=Red ctermbg=Black
:highlight Normal cterm=reverse ctermfg=White ctermbg=Black
:hi Normal term=bold
```

`:insert` Move focus to the GDB window.

`:n`
`:next` Send a next command to GDB.

`:q`
`:quit` Quit CGDB.

`:r`
`:run` Send a run command to GDB.

`:start` Send a start command to GDB.

`:k`
`:kill` Send a kill command to GDB.

`:s`
`:step` Send a step command to GDB.

`:syntax` Turn the syntax on or off.
`:up` Send an up command to GDB.

`:map lhs rhs`
Create a new mapping or overwrite an existing mapping in CGDB mode. After the command is run, if *lhs* is typed, CGDB will get *rhs* instead. For more details on how to use the map command look in Section 6.2 [Using Maps], page 17.

`:unm lhs`

`:unmap lhs`
Delete an existing mapping from CGDB mode. *lhs* is what was typed in the left hand side when the user created the mapping. For example, if the user typed `:map a<Space>b foo` then the user could delete the existing mapping with `:unmap a<Space>b`.

`:im lhs rhs`
`:imap lhs rhs`
Create a new mapping or overwrite an existing mapping in GDB mode. After the command is run, if *lhs* is typed, CGDB will get *rhs* instead. For more details on how to use the map command look in Section 6.2 [Using Maps], page 17.

`:iu lhs`

`:iunmap lhs`
Delete an existing mapping from GDB mode. *lhs* is what was typed in the left hand side when the user created the mapping. For example, if the user typed `:imap a<Space>b foo` then the user could delete the existing mapping with `:iunmap a<Space>b`.

5 CGDB highlighting groups

CGDB is capable of using colors if the terminal it is run in supports them. Until version 0.6.1, CGDB did not allow the user to configure these colors in any way. CGDB color use is now fully configurable.

CGDB's modeled its use of color highlighting after vim. Any data that will be colored in the terminal is represented by a highlighting group. A *highlighting group* represents data that should be formatted using foreground colors, background colors and attributes. There are currently several types of highlighting groups in CGDB. There are syntax highlighting groups, which represent syntax highlighting of sources files. There are also User Interface groups, which represent things like CGDB's logo, or the status bar.

Each highlighting group has a default set of attributes and colors associated with it. You can modify a highlighting groups properties by using the highlight command. See Chapter 4 [Configuring CGDB], page 9.

Note that CGDB currently supports using the same background color the terminal was using before CGDB was started. However, this only works when CGDB was linked with ncurses. If you link CGDB with curses, then CGDB will force the background to Black.

5.1 The different highlighting groups

Below is a list of all the highlighting groups that CDGB will use when syntax highlighting source files.

`Statement`
: This represents the keywords a language defines.

`Type`
: This represents the types a language defines.

`Constant`
: This represents either a string or numeric value.

`Comment`
: This represents the comments in a source file.

`PreProc`
: This represents the C/C++ preprocessor commands.

`Normal`
: This represents all normal text.

Below is a list of all the highlighting groups that CGDB will use when it is displaying it's User Interface.

`StatusLine`
: This represents the *status bar* in CGDB. The file dialog's status bar also uses this group.

`IncSearch`
: This represents the group used when the user is searching in either the source window, or the *file dialog window*.

`Arrow`
: This represents the arrow that CGDB draws to point to the currently viewed line.

`LineHighlight`
: This represents the group used when the user has the `arrowstyle` option set to `highlight`.

`Breakpoint`
: This represents the group that is used when CGDB displays a line that has a breakpoint set.

`DisabledBreakpoint`
: This represents the group that is used when CGDB displays a line that has a disabled breakpoint set.

`SelectedLineNr`
: This represents the group that is used when CGDB is displaying the currently selected line. This is the line that the cursor is on.

`Logo` This is the group CGDB uses to display its logo on startup when no source file can be auto detected.

5.2 The different attributes

CGDB supports many of the attributes that curses provides. It will apply the attributes to the output window, but it is up to the terminal you are using to support such features.

The list of attributes that CGDB currently supports is below.

`normal`
`NONE` This will leave the text normal. Uses A_NORMAL curses attribute.

`bold` This will make the text appear bold. Uses A_BOLD curses attribute.

`underline`
: This will underline the text. Uses A_UNDERLINE curses attribute.

`reverse`
`inverse` This will reverse the foreground and background colors. Uses A_REVERSE curses attribute.

`standout` This is the best highlighting mode of the terminal. Uses A_STANDOUT curses attribute.

`blink` This will cause the text to blink. Uses A_BLINK curses attribute.

`dim` This will cause the text to be 1/2 bright. Uses A_DIM curses attribute.

5.3 The different colors

CGDB supports several colors, depending on how many colors your terminal supports. Below is a chart of the colors that CGDB provides. The heading NR-16 is used to represent terminals that support at least 16 colors. The heading NR-8 is used to represent terminals that support at least 8 colors. The integer values for each color represent the values passed to the curses function init_pair() to ask curses to create a new color.

`COLOR NAME`	`NR-16`	`NR-8`	`NR-8 bold attribute`
Black	0	0	No
DarkBlue	1	4	No
DarkGreen	2	2	No
DarkCyan	3	6	No

DarkRed	4	1	No
DarkMagenta	5	5	No
Brown, DarkYellow	6	3	No
LightGray, LightGrey, Gray, Grey	7	7	No
DarkGray, DarkGrey	8	0	Yes
Blue, LightBlue	9	4	Yes
Green, LightGreen	10	2	Yes
Cyan, LightCyan	11	6	Yes
Red, LightRed	12	1	Yes
Magenta, LightMagenta	13	5	Yes
Yellow, LightYellow	14	3	Yes
White	15	7	Yes

6 CGDB key user interface

The Key User Interface is how CGDB receives input from the user. It is usually referred to as the *KUI*. CGDB simply asks the KUI for the next key the user typed and the KUI will provide it.

The KUI has 2 major responsibilities besides reading normal user input and providing it to CGDB. It needs to detect when the user has typed a user defined map or when the user has hit a special key on the keyboard.

A user defined map, or simply *map*, is used to change the meaning of typed keys. Some users may refer to this type of functionality as a *macro*. An example would be `map a b`. If the user then typed the `a` character, the KUI would detect that it was mapped to `b` and return `b` to CGDB.

When the user types a special key on the keyboard, a *key code* is sent to CGDB. Typically, keys like *`HOME`*, *`DEL`*, `F1`, etc, when pressed will send several characters to the application instead of just one character like a normal key does. These characters combined are called a *key sequence*. The KUI is responsible for assembling the key sequences back together and reporting to CGDB that a particular key was typed by the user. The *`ESC`* key is special because typically most key codes start with that key. This usually gives all key codes a common first key in its key sequence. The KUI uses the terminfo database to determine what key sequences are sent by which keycodes. There are a few commonly used key sequences that are hard coded into CGDB.

A major challenge the KUI has to overcome is determining when a map or a key sequence is received. The KUI sometimes will need to read more than one character to determine this. For example, if the user has 2 maps, `map abc def` and `map abd def`, the KUI would have to buffer at least the characters `a` and `b` before it could determine if the user was going to type a map. After the next key press, if the user types `c` or `d` then a map was received and the KUI will return *`d e f`* to CGDB. Otherwise, no map was received and the KUI must return *`a b`* to CGDB.

The options *timeout*, *ttimeout*, *timeoutlen* and *ttimeoutlen* can be used to tell the KUI if it should timeout on partial mappings or key sequences, and if so, how long it should wait before timing out.

6.1 The KUI's time out options

The KUI may be configured to time out on either maps or key sequences.

When the KUI is matching a partial map or key sequence it is capable of timing out. This means it will simply accepts the keys it has received so far if a certain amount of time elapses between key presses. This is obvious when the user is typing a map because the user must press each key individually. For partial key sequences, this is less obvious. That is because the user only presses a single key, but multiple characters are sent to CGDB. The table below describes how the user can configure the KUI to time out on key codes or maps. The *timeout* and *ttimeout* options control this functionality.

timeout	**ttimeout**	**action**
off	off	do not time out
on	on or off	time out on maps and key codes

off	on	time out on key codes

It is also possible to tell the KUI how long to wait before timing out on a partial match. If *timeout* is on, then the KUI will wait a certain amount of time for the next character, when matching a map, before it decides a match is no longer possible. If *timeout* or *ttimeout* is on, then the KUI will wait a certain amount of time for the next character, when matching a key sequence, before it decides a match is no longer possible. The *timeoutlen* and *ttimeoutlen* options can be configured by the user to tell the KUI how long to wait before timing out. The table below describes when the KUI uses which option.

timeoutlen	**mapping delay**	**key code delay**
< 0	*timeoutlen*	*timeoutlen*
>= 0	*timeoutlen*	*ttimeoutlen*

A value of 0 means that the KUI will time out right away. It will not be possible to match a map or key code in this circumstance.

A common problem could be that when the user types a special key like the left or right arrows, CGDB will go into the source mode and not perform the action requested by the user. This typically means that the key code delay is to small. If you try setting the option `set ttimeoutlen=1000` CGDB should start acting like the user expects. If not, please report this to the CGDB mailing list.

6.2 Using maps

CGDB fully supports the use of maps. It allows the user to change the meaning of typed keys. For example, you could have the following map `:map <F2> ip<Space>argc<CR>`.

When the user is in CGDB mode and they hit `F2`, the value of the map will be used instead. The *`i`* key will first be received by CGDB, and it will put the user into insert mode. Next, CGDB will get *`p argc`* followed by the `Enter` key.

CGDB currently supports two mapping lists. Any mapping that was added with the *map* command will be used by CGDB when it is in CGDB mode. You can delete a mapping that you have created with the *map* command with the *unmap* command. If you want to have mappings in GDB mode, you can use the *imap* command. Similarly, *iunmap* will delete a mapping in the *imap* set. Some examples of this would be

```
map a<Space>b foo
unmap a<Space>b

imap a<CR>b foo
iunmap a<CR>b
```

6.3 Understanding keycodes

The above example could use a little more explaining for people unfamiliar with vim maps. The map takes a key and a value. They are separated by a space. Neither the key or value can have a space in them, or it is considered to be the separator between the key and value. If the user desires to have a space in either the key or value part of a map, they can use the keycode notation <Space>. Below is a table of the keycodes in *keycode notation* form. The keycode notation can be used in any mapping command.

notation	**meaning**
<Esc>	escape key
<Up>	cursor up key
<Down>	cursor down key
<Left>	cursor left key
<Right>	cursor right key
<Home>	home key
<End>	end key
<PageUp>	page up key
<PageDown>	page down key
<Del>	delete key
<Insert>	insert key
<Nul>	zero
<Bs>	backspace key
<Tab>	tab key
<NL>	linefeed
<FF>	formfeed
<CR>	carriage return
<Space>	space
<Lt>	less-than
<Bslash>	backslash
<Bar>	vertical bar
<F1> - <F12>	function keys 1 to 12
<C-...>	control keys
<S-...>	shift keys

7 Sending I/O to the program being debugged

If the program being debugged takes input on the terminal it is recommended that the user start the program on one terminal, and attach to it with CGDB from another terminal. This is the easiest way to pass input to the debugged program.

However, if the user wishes to pass input to the program being debugged from within CGDB, there is a mechanism available for doing so. As of this writing, the technique described below does not work on windows, using a natively compiled GDB. It may work when using the GDB that comes with Cygwin.

This technique is similar to getting in and out of *GDB mode*. The tty window is not visible by default. This is because it is only needed if the user wishes to send data to the program being debugged. To display the tty window, hit `T` while in command mode. After hitting `T` you will notice that there is another window in the middle of the *source window* and the *gdb window*. This is called the *tty window*. You will also see a new status bar called the tty status bar. There will be a '`*`' on the tty status bar after the `T` was hit. This is because when the window is opened with the `T` command, CGDB automatically puts the user into *TTY mode*. To get out of this window hit the cgdb mode key. This will put you back into command mode. To make the tty window appear and disappear hit the `T` key while in command mode. It is a toggle.

Once the tty window is already open, the user can then hit `I` in command mode to get into *TTY mode*. The user can then hit the cgdb mode key in the *TTY mode* to get back into command mode.

When the tty window is open, all data that comes from the program, goes there. Any data typed into the tty window will ONLY go to the program being debugged. It will not go to GDB. When the tty window is closed, all output from the debugged program will go to the *GDB window* AND to the *tty window* (for viewing later when the tty window is opened).

If the user wishes to get a new tty for the program being debugged then they can type `Ctrl-T`. This will delete all the buffered data waiting to be read into the debugged program. This might be useful when you rerun or start a new program.

8 Allowing terminal control flow in CGDB

A user can typically set there control flow behavior by using the stty command like so `stty -ixon -ixoff`. This will disable control flow on the terminal where CGDB is started. If you want to turn control flow back on you can type `stty ixon ixoff`. If flow control is on, when the user types *`Ctrl-s`*, the terminal stops. When the user types *`Ctrl-q`*, the terminal restarts. When using readline, the *`Ctrl-s`* character usually does a forward search. So, if you want to get this, or other functionality out of readline, simply turn off control flow and start CGDB.

9 Building CGDB from source

Building CGDB from source requires several packages. First, CGDB is hosted at http://sf.net/projects/cgdb. You can determine how to get CGDB from source by looking here: http://sourceforge.net/svn/?group_id=72581.

Once you have the source to CGDB, now you can begin to build it. You will of course need many packages to build CGDB. Below is a list of all of them that are required to build CGDB.

GNU `Make`
: I have successfully used version 3.79.1, however, older versions probably will work.

GNU `GCC`
: The GNU C compiler. I've compiled CGDB with versions as old as 2.9.5, and as new as 4.0.2.

GNU `Readline`
: The GNU readline library version 5.1. CGDB will not work with versions before 5.1. Readline was modified specifically to work with CGDB.

GNU `Ncurses`
: I have successfully used libncurses.so.5 successfully. However, older versions probably will work.

Below is a list of optional packages you will need, if modifying certain files in CGDB.

GNU `Flex`
: If you modify any files with an extension of `.l`, you will have to have flex installed. I have used flex 2.5.4 to build CGDB.

GNU `Texinfo`
: If you modify '`doc/cgdb.texinfo`', then you will be required to have this package installed. I have used version 4.7 to build the documentation for CGDB.

`help2man`
: If you are doing a release, then you will be required to have this package installed. In the '`doc/`' build directory, you can execute the command `make cgdb.1`, and the CGDB man page will be generated.

CGDB uses autoconf/automake to build its configure scripts and makefiles. So, if you change any of the autoconf/automake files, you will need this software installed.

GNU `Automake`
: This has the program aclocal, and must be version Version 1.9.5.

GNU `Autoconf`
: This has the program autoconf, and must be version 2.59.

GNU `m4`
: This has the program m4, and must be version 1.4.3.

Appendix A Copying This Manual

A.1 GNU Free Documentation License

A.1.1 ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C)  year  your name.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.2
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover
Texts.  A copy of the license is included in the section entitled ``GNU
Free Documentation License''.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

```
with the Invariant Sections being list their titles, with
the Front-Cover Texts being list, and with the Back-Cover Texts
being list.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Index

A

attributes 14

B

building CGDB 21

C

CGDB key user interface 16
CGDB mode 3
colors 14
commands, in CGDB mode 6
commands, in File Dialog mode 8
commands, in GDB mode 7
commands, in TTY mode 8
configuring CGDB 9
controlling CGDB 6

F

FDL, GNU Free Documentation License 22
file dialog window 4

G

GDB mode 3
GDB window 4

H

highlighting groups 13

I

invocation CGDB 2

R

regular expression search 3

S

sending I/O to inferior 19
source window 3
status bar 3, 4
switch between windows 5

T

terminal control flow 20
timeout 16
timeoutlen 16
ttimeout 16
ttimeoutlen 16
TTY mode 3
tty window 4
TTY window 4

U

understanding CGDB 3