Gimp Mode

Niels Giesen

September 13, 2009

1 Short description

- Emacs[1] mode for developing scheme (script-fu) scripts for the GIMP[2].
- Interaction with the GIMP as subprocess or as a client.
- Smart and fuzzy code completion .
- Documentation echoing.
- Hypertext help system with history.

2 News: people do use Gimp Mode

2.1 Version v1.52:

- Multiple patches by Brent Goodrick to fix bugs thrown by completion functions inside comments and let/let* forms.

2.2 Version v1.51:

- gimp-mode.el (gimp-get-closure-code): traced down hanging emacs on quickly repeated keypresses (TAB and SPACE), to this call. Added a timeout argument of one second to the call of gimp-eval-to-string. (gimp-eval-to-string): add timeout option to avoid infinite loop. Throw error if timeout is exceeded. (Note: this bug could have also been avoided by setting 'gimp-try-and-get-closure-code-p' to non-nil).

2.3 Version v1.50:

- gimp-mode.el (gimp-selector): Add ESC as binding to cancel the selector + mention it under ?. Mention binding of ?c to gimp-cl-connect under ?. (gimp-first-run-action): Save input ring when gimp is closed externally. (gimp-buffer):

[1] http://www.gnu.org/software/emacs
[2] http://gimp.org

Just (get-buffer "*GIMP*") if process is finished. (run-gimp): clear gimp-output before going any further. This closes the bug "Wrong argument: sequencep Copyright" (or something like that..).

- gimp-mode.el, gimp-install.el: change default gimp version from 2.4 to 2.6

- gimp-init.el (magic-mode-alist): when the string "gimp" is part of buffer-file-name (anywhere) AND filename ends in ".scm", gimp-mode will be activated. This effectively means any scheme script distributed with the GIMP, or that is in your gimp-configuration directory will use gimp-mode.

2.4 Version v1.48 (the FUD release):

This version adds a betabetabeta version of `FUD`, meaning the Fu Debugger. It is however not yet set up by default yet (as it is *very* beta, as in "don't expect it to work" (actually it does...) and would break the client mode right now). With FUD, you can define breakpoints and instruct functions for stepping, poking at the environment at run-time &c.

Uncomment the `(require 'fud)` in `gimp-mode.el` *before* compilation to experience a bit of what lies ahead. Also, symlink or copy the file fud.scm to `~/.gimp-2.[45]/scripts/`. See contents of `fud.scm` and `fud.el` for pointers. Expect more `FUD` in about three weeks time. First vacation.

- gimp-mode.el (gimp-snippets): drop radio snippet, as the param type does not exist

- gimp-mode.el (gimp-comint-filter): add prettification to output of the GIMP.

- fud.scm: add stepping inside and function instruction.

- fud.el: add fud-bullet bitmap to show breakpoints.

- gimp-mode.el (gimp-open-image): changed order so that message is not put in the *GIMP* buffer.

- gimp-mode.el (gimp-completion-rules): enhance rule for palettes (gimp-shortcuts): gimp-clear (,clear from REPL) was added, to clear the REPL screen.

- (gimp-fnsym-in-current-sexp): fix for list beginning with a number (that bugged 'gimp-echo' higher up the stack): now returns nil when the symbol begins with a number.

- (gimp-echo): change cache-resp into response, as we're (a long time..) not using caches anymore to save the echoing.

- gimp-install.el (gimp-install): put installation in a function, then call it.

- fud.el (Module): new FUD FU debugger - elisp side

- fud.scm (Module): new FUD FU debugger - scheme side

- gimp-mode.el (gimp-switch-to-latest-source-buffer): made this functionality a separate command, and enhanced it so that you will switch to another source buffer if already in a source buffer.

- (gimp-help-mode-map): removed gimp-help-refresh (which isn't a command)

- (gimp-mode): Add some rules to highlight the frequent use of UPPER-CASE CONSTANTS in script-fu. This required turning off CASE-FOLD in 'gimp-mode'.

- (gimp-comint-filter): add FUD actions

2.5 Version v1.45

Fix for v1.44: use gimp-menu-map as mixin, not as parent.

2.6 Version v1.44

Added menu entries for most important commands.

2.7 Version v1.43

Echoing and completion made optional (and completion toggable with C-cc).
Reference to SICP added in `gimp-documentation`.
Some bugfixes.

2.8 Version v1.42:

Auto-insertion of template in empty `script-fu-*.scm` files.
Fixes for unconnected gimp-mode editing.
Fix for menu entry in GIMP from svn

2.9 Version 1.41:

Fix for installation on emacs23 on Windows (thanks Lennart Borgman).

3 Features

Interaction:

Gimp Mode can either run the Gimp as subprocess in Emacs, or connect as a client to the script-fu-server. The latter is less stable and has less features, but is sometimes better than nothing, see below. Interaction with the Gimp script-fu engine is possible directly from the code that is being edited and/or from a special REPL (read-eval-print-loop) buffer.

Smart 'n' fuzzy TAB-completion similar (but not equal) to that of Slime: the entire tiny-fu oblist is available and variables, functions and macros that are defined during a session are added to completion [3]. As you can see, the fuzziness accepts the first letters of a part of a hyphenated string to as input as a kind of abbreviation, so you can for instance write s-f-u-m and have it expand to script-fu-unsharp-mask. If for some reason you do not like this fuzzy factor, you can switch it off anytime by pressing C-cr or by customising the variable `gimp-complete-fuzzy-p`.

[3] This happens only when issued with the usual define/define-macro construct, and only at top-level, not if you e.g. wrap \texttt{define} in another macro.

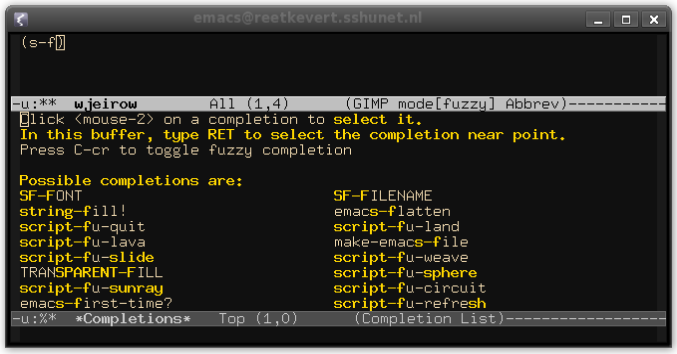

Figure 1: Fuzzy completion

Completion on arguments is done via a system of rules, based on regexp-matching/custom commands that work on the registration of the functions. It means you've got completion available for stuff like fonts, palettes, brushes, images, choices of booleans etc.

The completion system is open to be improved and extended upon (see variable `gimp-completion-rules`). For script-fu registered functions, default arguments are offered as completion.

Argument echoing whereby current argument is highlighted.[4]

More in-depth documentation echoing: both for argument at point and for the entire function.

In both documentation (echoing) and completion, the *procedural* first argument `run-mode` (Interactive, non-interactive) is omitted for script-fu functions, as this argument has to be omitted from a call made by script-fu.

A complete hyper-textual rework of the Procedure Browser implemented in Emacs Lisp, with history, apropos function, menu-driven look-up of plug-ins, nice faces (if I may say so) etc.

Some 'Bookmarks' to Gimp/Fu/Scheme resources on the 'net.

Handy shortcuts for various stuff oft needed when developing. Type `,shortcuts` at the REPL for an overview.

One of these shortcuts is `,trace` and `,untrace`, that toggle, well, tracing. Output from tracing is not put into Emacs' undo list, as GIMP's tracing can be *very* extensive. Do *not* use (tracing 1) yourself, as this can hamper behind-the-scenes interaction with the GIMP.

Some snippets are provided through the library `snippet.el` (included) by Pete Kazmier. A registration template is provided (type reg SPACE), and handy templates for script-fu-register arguments (type `sf* SPACE`). Type `M-x`

[4]For the interested: Gimp Mode gets its information in this regard from the following sources: the procedural database, the TinyScheme-function \texttt{get-closure-code} and lastly from \texttt{scheme-complete.el} (included) by Alex Shinn. In the echoing for script-fu functions the arguments as registered in the procedural database alternate with the arguments derived from the closure itself - if any.

Figure 2: The Gimp Mode Help Browser

`gimp-list-snippets` or `,list-snippets RET` from the REPL to show snippets.

Input history (which is saved on `,quit`).

Basic code lookup (using grep).

4 Caveats

1. The main mode is developed on and for a GNU/Linux environment, on Emacs22 and Emacs from CVS. I only have had very little time to test this on a win32 machine. The problem on that 'system' is that somehow the GIMP opens a second 'console', so it does not return anything useful to the calling program. I do not know about the behaviour on OSX. Not tested on BSD either. If you have any results (or a spare MacBook), please let me know.

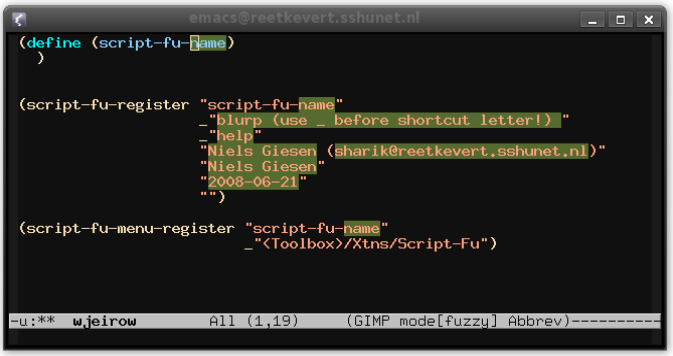

Figure 3: Snippet to set up the framework of a script-fu script

To overcome the problem of not being able to interact with the GIMP as a subprocess, Gimp Mode comes with another, similar, mode that hooks into the script-fu server provided by the GIMP as a client. You can start this mode with `M-x gimp-cl-connect` (after having started the server from within the GIMP). It was a PITA to get this to work well. The mode lacks some features of the 'normal' `inferior-gimp-mode` and has some idiosyncrasies due to the behaviour of the server: the GIMP script-fu server produces a new call frame each time around, making it unable (or possibly: quite hard) to define variables, functions and macros and saving their new values without hacking the gimp source. The way around this is the macro `emacs-cl-output` in `emacs-interaction.scm`, that writes the form (wrapped in another `(with-output-to-file ...)`) to evaluate to a temporary file, and subsequently loads that file, so that the new definition will be part of any new call frame. I'd love to be able to work something out using continuations, but I have not yet found out whether that is at all possible technically.

The features the client mode (as opposed to the truly inferior mode) currently lacks are:

- tracing.
- scheme functions `display`, `write` and any derivatives do not work. For these exact reasons, the FU Debugger does not work in this mode.

Note: behaviour when using both modes together is unspecified (what a lovely fall-back that word is...) and unsupported (although I do use it when developing the client mode. The trick is to first run `M-x run-gimp`, start the server, and then run `M-x gimp-cl-connect`).

Note that on any system gimp-mode is perfectly capable of performing quite well 'off-line', as it reads in most data through the use of caches. Stuff it cannot do unconnected (i.e. evaluation, and some echoing and completion that are dependent on evaluation) it will simply ignore. If not, that is a bug and should be filed as such (`M-x gimp-report-bug`).

2. I have not written many script-fu files to test this mode on, to wit: two, one of which is `emacs-interaction.scm` that comes with Gimp Mode (for the curious: the other one[5] makes a selection of sun rays).

3. There is no way currently to recover from non-returning forms, such an infinite loop, save for killing the process altogether. I'd love to find a way to deal with this. In this regard, the client mode is the better choice, as you are able to spawn several servers and just re-connect from emacs.

5 Comparison with other modes

Gimp Mode differs from `gimp-shell.el` in that the main objective of Gimp Mode is to run the GIMP as a subprocess instead of hooking into the script-fu server as a client. For the client mode, I have adopted code from `gimp-shell.el` (see `gimp-cl-send-string`), and am very grateful for it (I could not have come up with these awkward but apparently necessary byte-sequences myself - sooo low-level...). Further, Gimp Mode simply has a lot more. Oh, and then there is `gimp.el` (included with `nxhtml-mode`), which does nothing more than open an image in the GIMP (and then only on windows). Both of these other libraries do not seem to conflict with Gimp Mode; however, they are probably unnecessary as their features are implemented in Gimp Mode too.

6 Getting started/Download

First uncompress and unpack the files:

`gimp-mode.tar.bz2` (Linux etc.)

```
tar xjvf gimp-mode.tar.bz2
```

or

`gimp-mode.zip` (Windows)

```
unzip gimp-mode.zip
```

Now run the installation script from Emacs:

```
M-x load-file /path/to/gimp-install.el RET
```

And follow the directions given there.

(It will direct you to put something like the following in your load file and evaluate it:

```
(load "/path/to/gimp-mode/gimp-init.el")
```

...that file basically sets up everything so that stuff is only loaded when needed.)

[5] http://registry.gimp.org/node/6226

6.1 Now run the gimp inferiorly...

(Note for Windows users: skip this section and 6.2)

Now run `M-x run-gimp` to start the GIMP and its REPL within Emacs. This takes a little while (just a little longer than the graphical start-up phase - the Gimp is told by `emacs-interaction.scm` to set up some caches in your local gimp directory), so just be patient. In any event, do not type anything while the message "Reading in caches..." is displayed—for some reason I have not been able to track down yet this causes the reading in of caches to go on forever. We don't want that now do we? Visiting other buffers in the mean time is no problem however (if you *do* happen to have typed something at that moment, just quit with `C-g`, and issue `M-x restore-caches`—most will be well). See the customization for variables `gimp-program` and `gimp-command-line-args` to tweak GIMP's incantation to your needs.

Anyway, where were we. Yes. Once you see this, you can let the fun begin:

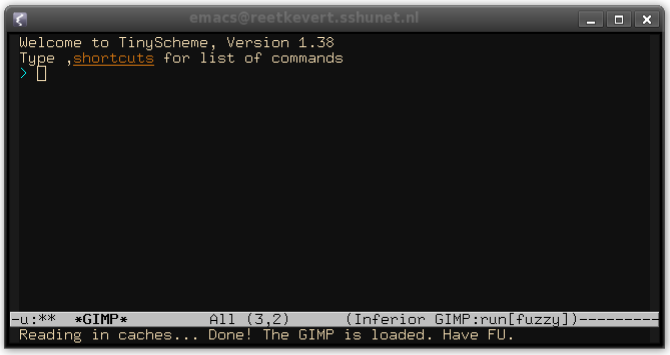

Figure 4: First encounter with the REPL

6.2 ...or attach as a client

If you want to hook into a running GIMP session as a client (this is the only means of interaction on a windows system) use `M-x gimp-cl-connect` after having started the script-fu server from the GIMP. In GIMP 2.4, the server is hidden under `Xtns > Script-Fu > Start Server`. In the development versions >= 2.5 it is tucked away (somewhat strangely in my humble opinion) under `Filters > Script-Fu > Start Server`. You will also find an entry there called "Dump internals for Emacs' Gimp Mode", which can be of use when `emacs-interaction.scm` is has been loaded before other scripts in your user directory on start-up.

Gimp Mode sets up buffers to to automatically use gimp-mode when the file begins with "s-f-" or "script-fu-" and ends with ".scm" (just ending in "scm" does *not* suffice, as it would be improper to impose gimp-mode on any scheme file). Also any file ending in ".scm" and in a path containing the string "gimp"

is automatically started in gimp-mode (as of v1.49). There are other ways, such as to put:

```
;; -*- mode: Gimp; -*-
```

in the first line of a script-fu file.

If you have set 'auto-insert' set to a non-nil value, gimp-mode will insert that line and a registration when visiting a blank file in gimp-mode or using `M-x auto-insert`.

7 To Do (?)

- A debugger (inferior mode only - if any).
- Find a way to get to python/C stuff REGISTERED on procedure arguments. (such as: lower and upper bounds, step, precision, default values...) (this has already been done for script-fu)
- Test on an OSX box (interesting).
- Add completion on `script-fu-register`, `script-fu-menu-register`.
- Add function for constructing a basic `script-fu-register` form from a `define` form.
- Make completion selection better (navigation and selection between candidates).
- Show tracing in a separate buffer (?).
- And of course I am open to suggestions

8 Known Bugs

The interactive client mode suffers from a strange lag, combined with returning the last value multiple times, especially with errors, and more especially on windows. I haven't been able to pin down its cause yet. The inferior mode does not suffer from this problem.

Please file any bugs you might find via `M-x gimp-report-bug RET`.

9 Possibly Asked Questions

Q. What was your incentive to write Gimp Mode?

A. When writing a script for the GIMP, I found the discoverability to be quite low, existing interaction mechanisms clunky and, well, to be honest, I simply have the habit of writing an Emacs mode whenever I embark on any new project that can be handled by emacs and has not been handled be emacs

yet—or insufficiently. I have to admit that often this stops me from engaging in the project itself, becoming too absorbed in its interaction with emacs.

Q: Why do you offer both an inferior mode and a client mode for the script-fu server?

A: It is way easier to start the GIMP as a subprocess, defining gimp-inferior mode as a derived mode from inferior-scheme-mode than hacking up (write (convert-form-to-emacs-readable FORM)) stuff, and so I naturally started with that. In order to be universally attractive (read: also on those pesky windows machines), gimp-mode asked me to be adapted to a client-mode operation. I had no option but to obey.

Q: What about a Python interface?

A:

- Just like the script-fu server, the Python batch-interpreter doesn't talk back (except on error). Therefore, this would require quite some tweaking.
- I like Lisp.
- I do not know Python that well.
- Furthermore, the python console provided with the Gimp is pretty good.

Q. Why don't you simply use the script-fu console that is shipped with the Gimp?

A. I wanted dynamic completion, interaction with source buffers, instant evaluation. The script-fu console does not have those.

Q. I want to find a script or plug-in whose place in the menu I know, but whose name I do not know.

A. C-cm to browse the menu structure to the rescue. This gives you the Gimp Mode Help page on that script or plug-in. Note that the sub-menus shown on top of that Help page are clickable too.

Q. Stuff does not work correctly when I turn on tracing via (tracing 1)

A. Use the wrapper functions `gimp-trace` / `gimp-untrace` for that; at the REPL: `,trace ENTER` and `,untrace ENTER`. NOTE that this tracing feature is only on at the REPL, not from .scm files (*and* not in client mode).

Q. Why didn't you name it `gimp.el`?

A. There is already a `gimp.el` in the nxhtml distribution (which actually does nothing more than open an image (and then only on win32)). And Gimp Mode is nicer for wiki pages anyway. So `gimp-mode.el` it shall be. Just harder to interpret it as a recursive acronym: gimp interaction mode for programmers mildly... gimp interaction mode performed mostly on... (and gimpel would rhyme so lovely with 'met vlag en wimpel...') Yes, life can be hard sometimes.

Q. What's with the strange versioning numbers?

A. That's just the CVS revision number for the file `gimp-mode.el`. I found that easiest.

Q. Where can I get that shirt the Gnu on GimpMode's homepage is wearing?

A. ...

10 Related

TIP on using `emacs-w3m` to browse gimp documentation:

To use `emacs-w3m` as the browser for help files from within the Gimp, put the following in your .gimprc:

```
(help-browser web-browser)
(web-browser "emacsclient -e '(w3m-browse-url \"%s\")'")
```

and `(server-start)` in your .emacs

Browsing the help with `emacs-w3m` is very nice, esp. since you can make use of the nice `w3m-scroll-down-or-previous-url`, `w3m-scroll-up-or-next-url` (SPACE) and `w3m-scroll-down-or-previous-url` (b), as relative links are provided by the HTML documentation of the Gimp.

11 Licence

Gimp Mode is licenced under the GPLv3[6].

12 Earlier versions

[ ]

13 Contact

Please use `M-x gimp-report-bug` from Emacs. This is OK too for feature suggestions.

If that should fail for some reason, you can contact me at `nielsforkgiesen@gmailknifecom`, but please replace the kitchen utensils with a dot before hitting "Send", lest anyone get hurt in the process.

[6] http://www.gnu.org/licenses/gpl.html

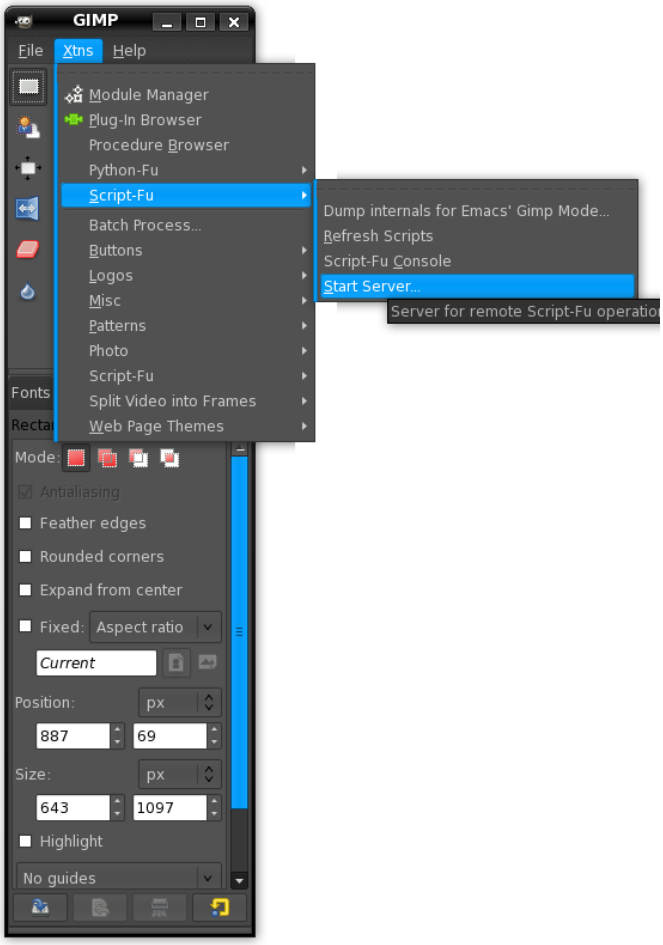

Figure 5: Start up the server from the Gimp