# Lisp in Summer Projects Submission

| | |
|---|---|
| **Submission Date** | 2013-10-02 01:51:06 |
| **Full Name** | Chip Collier |
| | |
| | |
| | |
| | |
| | |
| | |
| **Country** | USA |
| **Project Name** | plant |
| **Type of software** | command-line/terminal app |
| **General category** | development tool |
| **LISP dialect** | Scheme (MIT, Gambit, Chicken, Guile ...) |
| **GitHub URL** | http://github.com/photex/plant |
| **Did you start this project?** | Yes, all the code is written by me |
| **Project Description** | I want to describe my project in this form. |
| **Purpose** | Plant is a tool to automate a lot of the redundant interactions with your Common Lisp environment. In its current form it serves as a sketch for how the author imagines a fluid interaction between the developer and their environment. |
| **Function** | Plant will build a lisp image with any required quickloads already loaded, run your lisp image and optionally start a swank server for you, and most importantly it makes it easy and fast to test/run your system on multiple computers and unix flavored operating systems. |
| **Motivation** | Plant draws inspiration from tools like Leningen, Cabal, perhaps even Virtualenv. As a new user of Common Lisp, the author feels as though the TTH (time-to-hack) is a bit too high in some cases. There seems to be a gap in how fluid the experience of writing lisp is and how rote and redundant the experience of managing systems and environments are.<br>The author believes that many developers do not share this sentiment or feel as though Quicklisp is probably all they need. These developers should not be encumbered by a project that uses Plant. Because of this Plant is designed |

work on and around Quicklisp and ASDF rather than attempt to replace any aspect of those tools.

**Audience**

Whether you're new to Common Lisp or were an early contributor to the spec, if you would like an easy and repeatable build or deployment of your project then Plant is meant for you.

**Methodology**

The author wishes that they were in a position to describe a wonderful software design that led to some kind of project automation enlightenment. This is not the case. Plant is not much more than a small script which fulfills a very basic set of requirements. It sketches interactions with a more mature project automation or build system. In some ways this is a virtue because rather than getting bogged down by the pursuit of truth and beauty, the author created a simple tool that does a very particular job well enough to inspire future development.

Plant operates on several assumptions and if those are satisfied then it will proceed to do the following:
- Attempt to load settings for the lisp specified by $PLANT_LISP (defaults to sbcl)
- If there is no plant-project.scm in the current directory it will load default project settings and save them to plant-project.scm.
- If it hasn't already been setup, plant will install quicklisp to $PLANT_HOME (defaults to ~/.plant)
- At this point it will dispatch to a command handler based on the first argument specified.

Supported command are: build, run, quickloads, include, and help. The "run" command has an alias "swank" which is equivalent to `plant run --swank`. Several commands will build the project on your behalf if it hasn't been done already. This streamlines the interaction a bit.

Command descriptions are as follows:

'build': `plant build`, `plant build cl-graph sb-cga cl-opengl`
- Calls the setup function, which can take an optional list of quickloads to include in the lisp image. These are added to the list in the project settings.
- This command can be used to build an existing project.
- Build will fetch any dependencies (non-quicklisp systems that are required by the project)

'quickloads': `plant quickloads inferior-shell`
- Takes a list of system names and builds the lisp image with them loaded
- Updates the project settings (caveat: This is only an additive command. You will have to edit plant-project.scm manually to remove a quickloads entry)

'include': `plant include git git://github.com/rpav/cl-autowrap.git`
- Clones a mercurial or git repo into the project deps folder which acts as a Quicklisp local-projects location.
- Includes are fetched before any quickloads entries are loaded so that you can include a system that's not in quicklisp but still have your lisp image quickload it when creating the lisp image. More importantly if a system you're using has a bugfix not yet available in the current quicklisp

dist, this will make it very easy to use the latest version of the system until the next quicklisp release.

'run': `plant run`, `plant run --swank`, `plant run --swank 1234`, `plant swank`, `plant swank 1234`
- Runs your lisp for you! :) Optionally starting a swank server.
- For most cases if you were to clone a project that used plant you would only need to use 'plant run' to start working with it.
- Will fetch dependencies, and build the lisp image if required.

**Conclusion**

The choice of Guile as the implementation language was a chance for the author to give Scheme a try. In particular a Scheme that works pretty darned well as a replacement for zsh.

The next step for the project is a bit of hammock time, followed by a reimplementation in Common Lisp in order to make better use of Quicklisp and ASDF3. Continuing the project in Guile would only require another dependency and barrier to entry.

**Build Instructions**

You'll need Guile 2.0.x (the author tested with 2.0.8), and either SBCL or ClozureCL.

The author uses Ubuntu 12.04 as their primary OS. Guile can be installed via 'sudo apt-get install guile2'. SBCL can also be installed via apt, but the author used version 1.1.10 downloaded directly from sbcl.org.

The author also has an OSX machine that they tested on. Guile, SBCL, and Clozure can all be installed using homebrew: 'brew install guile --devel && brew install sbcl'

SBCL is assumed as the default. Set the env var PLANT_LISP to ccl or ccl64 if you prefer to use ClozureCL.

Plant is intended to be cloned to ~/.plant and ~/.plant/plant symlinked to ~/bin or some other location on your path. The author of the project does the following:

```
cd ~
git clone git://github.com/photex/plant.git .plant
cd ~/bin
ln -s ~/.plant/plant
```

At this point you should be ready to rock.

**Test Instructions**

Once plant is setup you should be able to do the following:

```
mkdir ~/my-projects-dir/test-project
cd ~/my-projects-dir/test-project
plant swank
```

If everything went well you should now find yourself in the REPL with a swank server running on the default port.

**Execution Instructions**

Once you've established that plant runs a repl correctly for you, you can checkout the plant-example project which was created using the authors other LispInSummerProjects submission.

| | |
|---|---|
| | cd ~/my-projects-dir<br>git clone git://github.com/photex/plant-example<br>plant run<br>;; from inside the repl<br>(ql:quickload :plant-example)<br>(plant-example:demo 100)<br><br>This will display a GLUT window which draws a random delaunay triangulation using OpenGL. Right mouse dragging zooms and middle mouse dragging pans.<br><br>This project demonstrates the ability to clone git repos as local dependencies. |
| **Describe any bugs or caveats** | Plant is definitely a rough draft and there are certainly some sharp corners. Probably the easiest thing to spot is that where ever you run plant, a plant-project.scm and .plant folder will be created if they don't already exist. So you probably want to make sure you only run plant inside a project directory. |
| **Official** | I have read rules and have abided by them.<br>I am 18 years of age or older.<br>I am not living in Brazil, Quebec, Saudi Arabia, Cuba, Iran, Myanmar (Burma), North Korea, Sudan, or Syria. |