Using Petite *Chez* Scheme with Emacs

Claude Anderson Revised August 31, 2014

This document's notation for special keys and key sequences

`[Ctrl]-x k`	hold down the `Ctrl` key while pressing `x`, then press `k` (without the `Ctrl` key).
`[Ctrl]-x [Ctrl]-k`	hold down the `Ctrl` key while pressing `x`, then hold down `Ctrl` while pressing `k`.
`[Esc] v`	Press and release the `Esc` key, then press `v`. (On some machines, you can instead hold down the `Alt` key while pressing `v` to get the same functionality)
`[Return]`	Press the normal `Return` (or `Enter` on an MS-Windows machine) key.
`[Esc] [ctrl]-f`	Press and release `Esc`, then hold down `Ctrl` while pressing `f`. on some machines, you can instead hold down the `Alt` and `Ctrl` keys while pressing `v` to get the same functionality)

You may use any text editor on any machine to edit your Scheme programs. I think that you will find it worth your while to learn and use E*macs*, because of the many ways that it supports Scheme programming. This is not surprising, since both Scheme and Emacs have their roots at MIT. Originally developed as an editor to use on character-only terminals, Emacs still works quite well in that environment, but it also takes advantage of GUI features. There have been many versions of Emacs (one was developed by James Gosling, do a google search for "father of Java"). For most people today, "Emacs" means "GNU Emacs" or XEmacs. GNU stands for "GNU's Not UNIX". All GNU products are developed and distributed free of charge by the Free Software Foundation. What does "Emacs" stand for? See the `Jokes` file in the `etc` folder that is part of the Emacs installation.

You can use Emacs on your computer, or run it on remotely on addiator.rose-hulman.edu or sliderule.csse.rose-hulman.edu. If you have Linux, you already have Emacs. If you have Windows, you can use the instructions on the Assignment 0 handout to install Emacs. **The rest of this document assumes that you have done that.**

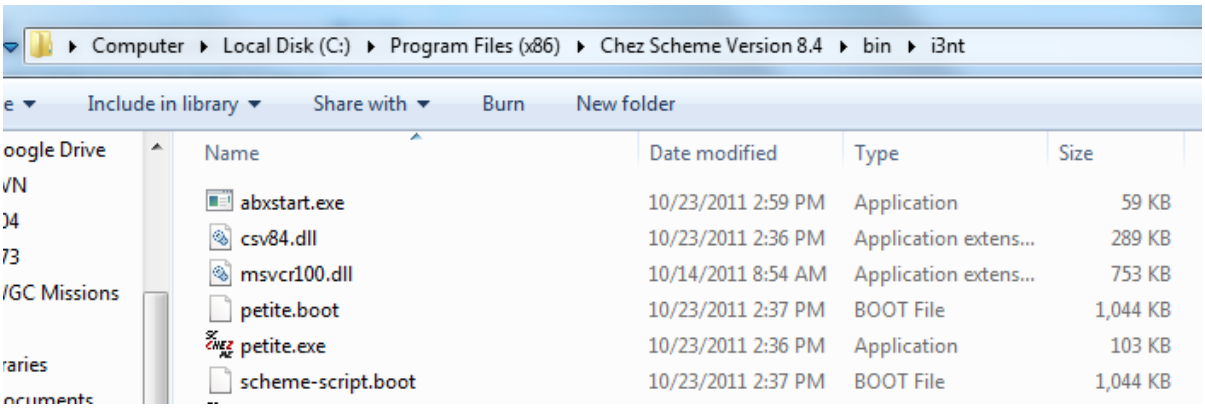
Note that you can do most editing actions in Emacs from the menus, but you can edit much faster if you take the time to learn the keystrokes for some common commands.

On Rose servers (such as sliderule or addiator): (Getting set up to use Emacs, which is already installed)

If you don't already have a `.emacs` file in your home directory, you may want to copy Michael Wollowski's from http://www.rose-hulman.edu/class/csse/csse304/emacs/Addiator.emacs into your home directory and rename it to **.emacs**. Simply type: `cp /class/csse/csse304/emacs/Addiator.emacs ~/.emacs`

On Windows Machines

Install Petite *Chez* Scheme, according to the instructions in Assignment 0.
In Windows Explorer, go to the folder where you installed it (probably `C:\Program files(x86)\Chez…`), and find the `bin\i3nt` folder.



In that folder make a new text file called **`scheme.bat`** that contains one line: `petite`
Follow the instructions in the Assignment 0 document for installing Emacs and adding my `init.el` file.

Basic Emacs commands

If you are running Emacs with a GUI, you don't need to know many commands at first, since most of the things that you'll want to do are also in the menus. Eventually you'll want to run the tutorial (which you get to by typing `[ctrl]-h t`) which will help you learn the keyboard commands, because they are faster than using the mouse, and because there are a few useful commands that are not in the menus.

The visible cursor in Emacs (that indicates where you are in the file) is a dark block. The actual insertion *point* is **before** the character that is highlighted by the cursor. This is a vestige from the days when there were no bitmapped character display terminals. This position before the highlighted character is called the *point*.

The Emacs tutorial (again, you can get to it by pressing `[ctrl]-h t`) is a good place to start learning commands.
Here are some additional lists of basic commands. In at least one of the lists, `Mx` (Meta x) means `[esc] x`
(You can also get the `Mx` effect by typing [ALT]-x on most machines.
Gnu Emacs Command Reference card

Launching Scheme within Emacs, interacting with Scheme.

a) `[ctrl]-x %` opens a new window in Emacs and runs Scheme in that window. **Infinite loops:** If your Scheme program gets into an infinite loop, type `[ctrl]-c` a few times to get into the debugger. Then type `r [Enter]` to reset and get back to the normal Scheme prompt.

b) **Re-executing previous commands.** If the cursor is at the end of the Scheme window, `[esc] p` (or `Alt p`) allows you to move backwards through your history of recently-executed commands. If you move too far back, `[esc] n` lets you move forward (`p` and `n` are used for "previous" and "next" in many Emacs commands). You can then edit and/or re-execute previous commands.

c) **Exiting the Scheme session.** `(exit) [Enter].`

d) **Loading files into Scheme** You can type `(load "filename.ss")` into Scheme (running in Emacs or in a terminal window). But in Emacs, there is a simpler way! With Scheme running in an Emacs window, type `[ctrl]-c [ctrl]-l` (that's "el", not "one"). In the minibuffer (the single line at the bottom of the Emacs window), enter the filename that you want (you can do filename completion by typing the first part of the name and pressing `[tab]`). Press `[Return]`, and Scheme will load that file. If, in the same Emacs session, you want to load the same file again, simply enter `[ctrl]-c [ctrl]-l [Enter]` and Emacs will reload the last file that you loaded in this way. This is a very common operation when you are debugging your Scheme code.

e) **Send-to-scheme.** I wrote an Emacs keyboard macro that I find very useful; perhaps you may also.

Pre-requisites:

- You have my `.emacs` or `init.el` file (or at least the part near the end that defines this macro) properly installed.
- You have split your Emacs screen into exactly two windows (`[ctrl]-x 2` splits the window, `[ctrl-x] 1` gets you back to a single window).
- You are running Scheme in one window and have your file in the other one.
- The point (cursor position) in the Scheme window is at the bottom of that window.
- The point in the other window is at the beginning of the s-expression that you want to send to Scheme, and this window is the current Emacs window (i.e. you see the cursor there now).

Do it: Now press `[ctrl]-x x` to see the magic happen.

I often find it useful to have two windows open in Emacs: one running Scheme and the other containing my source file.

A few useful commands that you might not find in the tutorial right away. Some are especially useful for editing Scheme code.

`[ctrl]-g`	Oops! Cancel the current command.
`[ctrl]-x u`	Undo last action. Multiple undos are supported.
`[esc] [ctrl]-f`	Move the cursor forward one s-expression (list). (Recall that `[ctrl]-f` alone moves forward one character, while `[esc] f` moves forward one word).
`[esc] [ctrl]-b`	Move the cursor backward one s-expression (list).
`[esc] [ctrl]-k`	Delete the s-expression that immediately follows the cursor (point). You can then use `[ctrl]-y` to get it back or to paste it in somewhere else.
`[esc] [ctrl]-t`	Transpose the s-expressions before and after the point. (`[ctrl]-t` transposes characters, `[esc] t` transposes words, the combination transposes s-expression).
`[tab]`	Indent this line according to Scheme indentation rules.

[Enter]	Go to a new line
[esc] [ctrl]-\	Properly indent the entire current region
[esc] /	Expand previous word "dynamically". Expands to the most recent, preceding word for which this is a prefix. If no suitable preceding word is found, words later in the file are considered. This is very useful when you are using a long variable name and don't want to retype the same name a second time.
[esc] [space]	Collapse all surrounding spaces to a single space.
[esc] \	Delete all surrounding spaces.

Parenthesis-matching

Furthermore, Emacs gives you a lot of help with parenthesis-matching. *Chez* Scheme allows you to use either '(' and ')' or '[' and ']' for any matched pair of parentheses. Using the square brackets can improve readability and make it easier to match the correct number of parentheses at a given point in the program:

```
. . . (cond [(null? a) '()]
            [(atom? a) (cons a b)]
            [(atom? (car a)) (cons (cdr a) (cons a b))]
            [else (display "an error")]) ...
```

When you type a closing ")" or "]", Emacs momentarily blinks the cursor on the matching opening symbol. And if you type "]" in a place where ")" is the correct matching symbol or vice-versa, Emacs signals mismatched parentheses.

If you need help with anything on this handout, feel free to come to my office for a demonstration, or ask in class sometime when I have my laptop there. Other students can also be a valuable source of help.